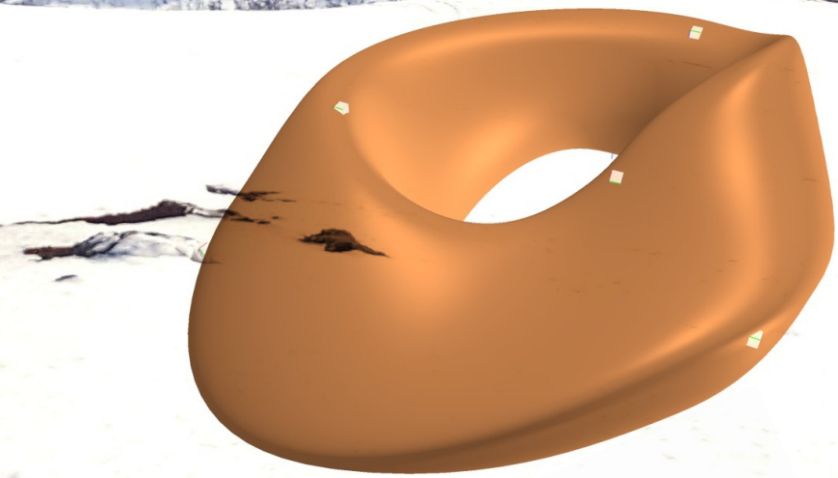
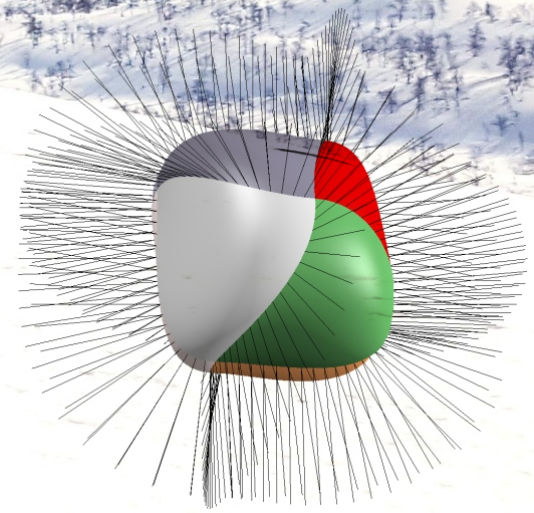
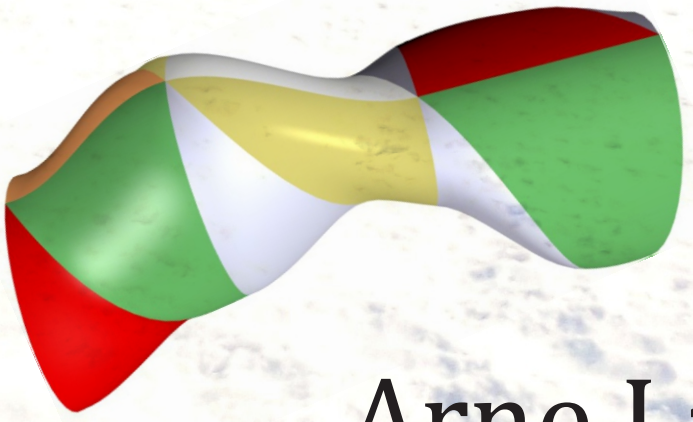
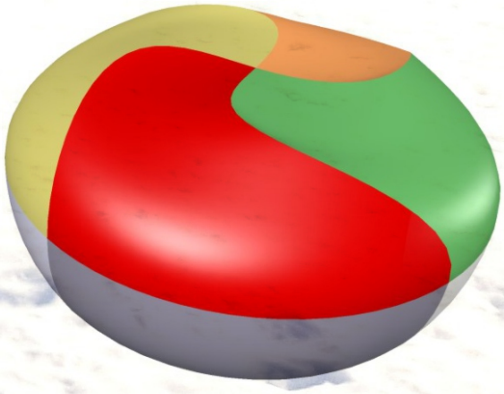
Blending techniques in Curve and Surface constructions







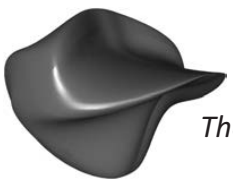


Arne Lakså

Arne Lakså

Blending techniques in Curve and Surface constructions

ISBN 978-82-693065-1-4



GEOFO
The geometry publishing house

supported by
THE ARCTIC UNIVERSITY OF NORWAY
UiT

Narvik 2022

Arne Lakså
Department of Computer Science and Computational Engineering,
Faculty of Engineering Science and Technology,
UiT The Arctic University of Norway,
mailbox 385, N-8505 Narvik, Norway.

The geometry publishing house is an ideal publisher for geometry/mathematics/programming, and which works to make digital books freely available. Paper editions can in some cases be purchased at cost price, mainly printing costs and shipping.

The geometry publishing house is supported by UiT The Arctic University of Norway.

Website: www.geofo.no
Email: post@geofo.no

Preface

Geometry – from Ancient Greek, earth measurement – has been an important ingredient of the development of science and later also industry, design and production.

In modern time, the Geometric modeling community was established, and now has a more than 50-year-old history. The first pioneers were motivated by the introduction of computers for use in design, construction and manufacturing. The goal was initially to provide methods and algorithms for curve and surface representations and calculations, and to combine curve and surface methods in computer graphics and simulations. Thus, they started the development of the new discipline called geometric modeling, including computer aided geometric design, solid modeling, algebraic geometry, and computational geometry. Computer aided geometric design (CAGD) is the central part of the field. It started with the development of methods and algorithms for CAD/CAM. Today, however, support for virtual reality, virtual design, computer games, simulators and animations in movies and TV productions are equally important areas.

Curves and surfaces for computer aided geometric design started with classical geometric objects as line, arc, plane, sphere, cylinder etc. The next step in the development of free-form geometry was Bézier, Hermite, B-spline, rational variants as rational Bézier and NURBS, and subdivision surfaces. The development began in earnest in the 1960s and went with full force until 2000s. It is, however, still an ongoing development that includes T-splines, LR-splines and other improvement as Multivariate splines. New results also includes the introduction of generalized B-splines and thus it is still a way to go. This way of constructing curves and surface is the most important today and will probably be it for a long time. It is the defacto industrial standard. What we can call a 2nd generation curve and surface construction started after the year 2000, and it is omly in its initial phase. It includes different types of curves and surfaces constructed by blending technics.

For me, the work with blending technics accelerated in the summer of 2003. I was in the office of my colleague Børre Bang to discuss improvements to a C++ programming library for geometric modeling (see [107]). Lubomir Dechevsky then came and asked if we could look at a function he meant was the limit of a polynomial B-spline when the degree (and thus the number of knots) tends to infinite (described in [44]). Since we were then working on the graphical part of a programming library, we implemented this new basis function and tried it for curves in $\mathbb{R}^3$. The result was a piecewise linear curve just like 1^{st}-degree B-spline curves, but there was a big difference; this new curve was C^∞-smooth but obviously only G^0, i.e. piecewise linear. After a short discussion

we replaced the coefficients of the curve with something we called “local curves”, and a G^∞ curve appeared. Lubomir suggested the name Expo-Rational B-splines (ERBS). A couple of years later, Beta-function B-splines was introduced by Lubomir and ERBS was expanded to a more general concept which we called Generalized Expo-Rational B-spline (GERBS).

Some years later, the concept of B-functions was developed, other rational and expo-rational types that previously had been published by different authors, and new classes of B-functions were included, among them the Fabius function and general trigonometric B-functions. With this, expo-rational B-functions became just a class of many and it was therefore natural to refer to the technique as blending splines. At the same time, it is important to show that it is built up as a B-spline and is a B-spline of 2nd order, with all the properties that follows from being a B-spline.

This manuscript was originally intended to be exclusively about blending technics. However, due to input and questions from students, and that this manuscript is also influenced by the fact that parts of it have been used as compendiums in courses in a master’s program in computer science at UiT The Arctic University of Norway, I decided to expand it to include some material about splines and basic geometry. However, the manuscript has a long history and has for a long time been available to students who have followed my courses. A first draft already existed in 2009.

Acknowledgements
First I will express my gratitude to my colleagues Lubomir T. Dechevsky and Børre Bang. They have been absolutely crucial for the initial development of ERBS. Then all PhD students that has been involved in the work, both during their PhD period and after; Joakim Gundersen, Arnt Kristoffersen, Peter Zanaty, Rune Dalmo, Jostein Bratlie, Hans Olofsen, Tatiana Kravetc, Tanita Fossli Brustad and Aleksander Pedersen. Next I am also grateful to Tom Lyche and Knut Mørken for very valuable discussions, and to Malcolm Sabin who was opponent of my doctoral defense and there asked some important questions.

Finally, I would like to give big thanks to my wife Marit, for bearing with me through this work.

May 2021 Arne Lakså

Contents

Chapter 1

Introduction

Development of geometry can be traced back to early Egyptian, Babylonian and Vedic India period. In this early period, the focus was on circles, triangles, length, area, volume, etc. This was mainly connected to empirically discovered principles and was for use in practical applications to meet needs in surveying, construction, astronomy and various crafts. The more mature period, however, began with the Greeks. Therefore, the development of geometry as we know it can be traced along a line that roughly divides history into the following five periods:

1. *The synthetic geometry of the Greeks* (600BC to 200BC). Greek geometry ended mainly by Archimedes. Beside or after the Greek area, geometry was in some sense developed in parallel or followed by Indian, Chinese and Arabic geometry. Greek geometry, however, was the direct predecessor of the analytical geometry of point 2, because Euclides 13 books on geometry, titled "The Elements of Geometry" [60] was the direct predecessor.

2. *The birth of analytic geometry* (1600 to 1650), in which the synthetic geometry of Guldin, Desargues, Kepler and Roberval merged into the coordinate geometry of Descartes [48] and Fermat.

3. *Application of the calculus to geometry* (1650 to 1800), including the names of Newton, Leibnitz, the Bernoullis, Clairaut, Maclaurin, Euler, and Lagrange (each an analyst rather than a geometer).

4. *The beginning of modern geometry*, (19th-Century). That is, Lobachevskian geometry, projective geometry, analytical geometry, non-Euclidean geometry, and differential geometry. In addition, it is worth mentioning some specific areas, Riemannian geometry, Klein's Erlangen program for classification, Lie groups, homogeneous coordinates, Hilbert's axioms.

5. *New advanced geometry and applied geometry*. This includes modern algebraic topology/geometry and new developments in differential geometry. For these we find application in, for example, modern physics. Other areas are fractal geometry, discrete geometry and new applied geometry. The development here is linked to needs in the industry and is the theme further in this book.

Figure 1.1: *Euclid of Alexandria, the "Father of Geometry". A statue at Oxford University Museum of Natural History.*

In the 1950s, the automation process in industry accelerated. At the same time, the computer was introduced. Together, this led to a strong need for the development of applied geometry. Therefor, the community of geometry began to expand. A new branch of geometry is beginning to develop, industrial geometry. The pioneers of this period were often associated with ship, airplane or automotive industry and where motivated by the introduction of computers in design, construction and manufacturing. The aim was to provide methods and algorithms for curve and surface representations and calculations on these, and to apply curve and surface methods in computer graphics and simulations. By doing this, they started the development of a new discipline called geometric modeling, including computer aided geometric design, solid modeling, computational geometry, digital geometry and space partitioning. It started with the development of methods and algorithms for CAD/CAM and was followed by modeling of oil and mining-field, human bodies etc for simulation purpose. Today, however, support for the development of virtual reality, virtual design, virtual prototyping, virtual engineering, computer games, simulations and animations in movies and TV productions is an equally important area for geometric modeling.

This book attempts to link today's industrial standard, B-spline, with its precursors and offshoots, with new blending techniques for curves and surfaces constructed. Occasionally we will go back to historical notes about geometry development.

In Figure 1.1, there is a statue of the father of geometry, Euclid. The 13 books of geometry "The Elements of Geometry" has been and is still important to give insight in geometry.

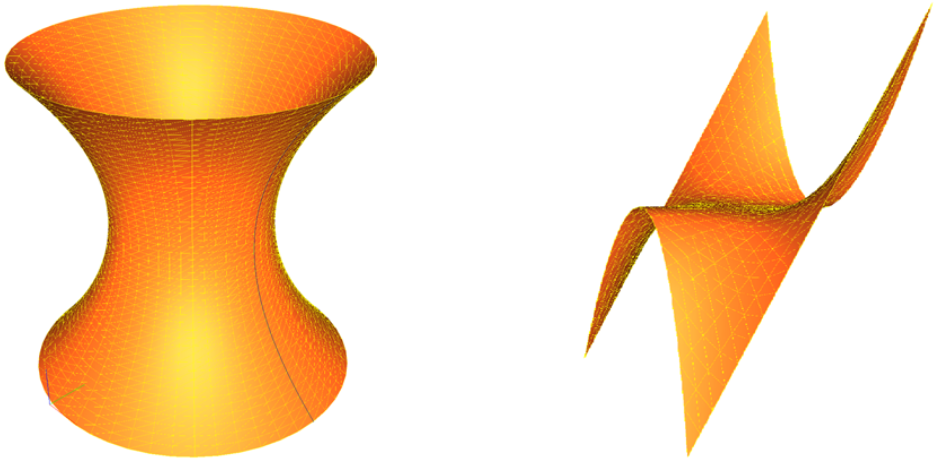


Figure 1.2: *On left hand side is a rotational surface made by rotating a curve, on right hand side is there a monkey saddle computed directly from a formula.*

1.1 Industrial geometry

It is geometry for industrial purpose. one part is design, another part is calculations on geometry. Applications are today important for areas such as:

- ✓ Computer graphics
- ✓ Computer Aided Design - CAD (product design)
- ✓ Computer Aided Manufacturing - CAM (productions)
- ✓ Virtual and Augmented reality VR and AR.
- ✓ Virtual art (virtual sculptures, mathematical sculptures, ...)
- ✓ Simulations and animations (movies, games, scientific simulations, ...)
- ✓ Map-related applications, geodesic
- ✓ Self-driving cars / busses / lorries / boats / airplanes / drones / spacecrafts
- ✓ Computations in general, for example strength calculations, thermodynamics, fluid mechanics, kinematics, vibration, resonance etc.
- ✓ Artificial Intelligence, object recognition, face recognition, ...

Industrial/Applied geometry is mainly for implementation on computers. In Figure 1.2 we see some examples. On left hand side is a surface of revolution (made by rotating a curve, see section 9.2) and on right hand side is a monkey saddle made directly from a formula, ie (9.1).

Curves, surfaces, volume, ... can be described implicitly or parametrical. Parametric description is most commonly used. One important type of parametric geometry is splines. Today the spline family is quite big. We will later look at classical cubic splines, Hermite splines, B-splines, NURBS, uniform splines and subdivision, circle splines and blending splines. We will also briefly look at “patchwork”, especially with triangles. This is a type of surface that is composed of many sub-surfaces/triangles in a more or less smooth way.

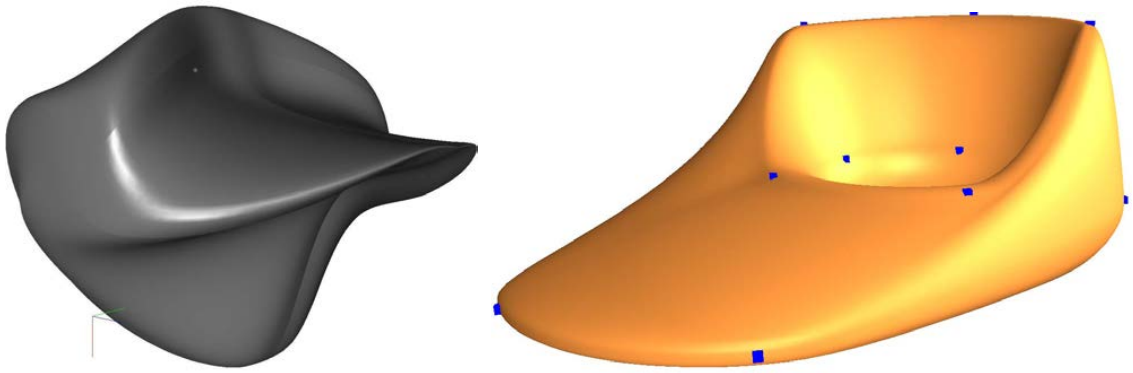


Figure 1.3: *On left hand side is a surface that initially was a sphere, on right hand side was the surface initially a torus. Both surfaces have been changed by interpolation and interactive editing.*

There are different uses for geometry, ie

- High quality static geometry.
- Geometry that makes precision calculation possible.
- Geometry that can be easily shaped, copied or reshaped.
- Geometry that can dynamically change shape in "real time".

Here different types of geometry have different properties, B-splines are for high quality and calculations, subdivision surfaces are best for graphics, images, computer games ... and blending splines are good for difficult shaping and dynamic change of shapes. In Figure 1.3 two surfaces are shown. To the left is an initially sphere that has been copied and then edited to become a duck face, to the right is an initially torus that has been copied and then changed to look more like a duck foot.

1.2 Geometric modeling

The term geometric modeling first came into use in the late 1960s, a time of rapidly developing in computer graphics and computer aided design and manufacturing technologies. The discipline of geometric modeling is an interrelated, although somewhat loosely integrated, collection of mathematical methods that we use to describe the shape of an object or to express some physical process in terms of an appropriate geometric metaphor. These methods include;

✓ Computer aided geometric design (CAGD) applies the mathematics of curves and surfaces to modeling, primarily using the parametric equations of differential geometry, supported by interpolation and approximation theory. It is here that we find the roots of contemporary geometric modeling.
✓ Solid modeling usually encountered as constructive solid geometry (CSG), allows us to combine simple shapes to create complex solid models.
✓ Algebraic geometry is the contemporary extension of classical analytic geometry, including differential geometry.

✓ Computational geometry is concerned with design and analysis of algorithms, and is related to numerical methods, computation theory and complexity analysis.

However, geometric modeling is constantly expanding and now also includes fractal and digital geometry and space partitioning.

It has been most common to study geometric objects that are two- or three-dimensional, although many of the tools and principles can be applied to sets of any limited dimension. Today, however, the models are mostly three- or four-dimensional in computer-aided design and production, CAD / CAM (3D and time). The same applies to applied technical fields such as civil and mechanical engineering, architecture, geology, health and a large field we can call virtual world systems, computer games, VR systems and simulators. Geometric modeling is the basis for applications such as animation and special effects for advanced modeling software for industrial design and architecture and for 3D printers. Geometric models represent shapes and spatial relationships to the environment being studied, which enables a much deeper analysis than would otherwise be possible. How these models are coded, and how the algorithms that use them are, include the field of computer-aided geometric design (CAGD).

For Computer Aided Geometric Design, an important event was a conference held at the University of Utah in 1974 organized by Barnhill and Riesenfeld [7]. Before this conference, we can say that the field was in its embryonic stage, after this conference, we can say that CAGD (Computer Aided Geometric Design) was born as a discipline. Today CAGD is a mature discipline, where B-spline and spline methods are industrial standards and implemented in thousands of applications. However, the field is even a more exciting field than ever, because the field's significance is expanding since the introduction of virtual and augmented worlds, computer games, animated movies, social worlds, artificial intelligence against recognition and track choices. The primary objects of interest are curves, surfaces, and volumes such as splines (NURBS), meshes, subdivision surfaces as well as algorithms to generate, analyze, and manipulate them. This book will give an overview of the field and look on new developments in CAGD and its applications, including but not restricted to the following:

- To collect and disseminate information on computer aided design.
- To provide the community with methods/algorithms for representing curves/surfaces.
- To illustrate computer aided geometric design by means of interesting applications.
- To combine curve and surface methods with computer graphics.
- To explain scientific phenomena by means of computer graphics.
- To concentrate on the interaction between theory and application.
- To expose unsolved problems of the practice.
- To develop new methods in computer aided geometry.

In this book we also introduce a new generation of curve/surface construction based on blending technics. Historical is this not new technics. Both Coons patch [26] and different types of fillet constructions are using blending technics, and Circle splines [163] is very close to the construction presented in different articles about what first was called GERBS (Generalized Expo-rational B-splines), see [104, 44, 45, 102, 43, 105, 106].

1.3 Algorithmic language

By "algorithmic language" we mean the language used to describe the algorithms that are presented. All programming is done in C++ . This is naturally also reflected in the algorithmic language. One important factor in the definition of the syntax is that it should be compact and at the same time clear and easy to understand. The items that describe the algorithmic language are:

- ✓ The notation in the algorithm is a simplification and essentially a mix of C++ and a mathematical notation.
- ✓ C++ template notation is used. Also the C++ standard template library (stl) are used - example, vector<double>, a vector of numbers in double precision.
- ✓ To make the notation more compact, "{" and "}" are only marked by indentation.
- ✓ A routine might only be notated as a set, as example $vector\langle T\rangle\ C\ =\ \{D^j c(t)\}_{j=0}^{n}$. But it will always be followed by an explaining comment.
- ✓ Ordinary C++ standard is used for comments.

Below is an example, an implementation of a function **bspline** defined in Section 6.2.9. The C++ code is first shown, then the algorithm using the algorithmic language.

```
vector<double> bspline(vector<double> k, int d, int ii , double t) {
   vector<double> b(d+1);
   vector<double> w(d);
   b[1] = (t - k[ii])/(k[ii+1] - k[ii]);
   b[0] = 1 - b[1];
   for(int i=2; i<=d; i++) {
     for(int j=0; j<i; j++)
       w[j] = (t - k[ii+j])/(k[ii-j+i] - k[ii-j]);
     b[i] = (1-w[0])*b[i-1];
     for(int j=i-1; j>0; j--)
       b[j] = w[i-j]*b[j-1] + (1-w[i-j-1])*b[j];
     b[0] *= w[i-1];
   }
   return b;
}
```

vector⟨double⟩ **bspline(** *vector⟨double⟩* τ, *int* d, *int* ς , *double* t **)**

vector⟨double⟩ b(d+1);	// The return vector, dimension $d+1$.
vector⟨double⟩ w(d);	
$b_1 = W_{1,\varsigma}(t;\tau)$;	// see (6.11)
$b_0 = 1 - b_1$;	// The general Cox/deBoor algorithm for
for (*int* $i = 2; i \le d; i++$)	// - B-splines, computing the set
for (*int* $j = 0; j < i;$ j++)	// - of all B-spline values of degree d at t
$w_j = W_{i,\varsigma-j}(t;\tau)$;	// - when $\tau_\varsigma \le t < \tau_{\varsigma+1}$.
$b_i = (1 - w_0)\, b_{i-1}$;	
for (*int* $j = i-1; j > 0; j--$)	
$b_j = w_{i-j}\, b_{j-1} + (1 - w_{i-j-1})\, b_j$;	
$b_0 = w_{i-1}\, b_0$;	
return b;	

This example can be found on page 96, algorithm 3. The C++ code in this example is used in the evaluator for B-splines. Normally the derivatives are also computed, code for that can be found in Algorithm 4 on page 97.

1.4 Overview of this book

The book is divided into 4 parts and 14 chapters. The parts are:

- ✓ Introductions
- ✓ Curves
- ✓ Surfaces
- ✓ Appendix

The three introductory chapters are:

Chapter 1 gives the reader an historical background, first in general about geometry, then more specifically about industrial geometry and geometric modeling. The chapter also provides a description of the algorithm language used in the book, as well as this summary.

Chapter 2 is about mathematical spaces, different types of coordinates and notations. In order for the reader to better understand applied geometry, the formulas, the algorithms and implementation of geometry, this chapter is important.

Chapter 3 is about computer programming of geometry, what affine and projective spaces mean for programming, why inner product is the most central function and how local coordinate systems work. In addition, some tools for interactive design and simulations are provided.

The five chapters about curves are:

Chapter 4 is about parametrisation of curves in general, and a little deeper about polynomial based curves of different formats and some basic algorithms for computing position and derivatives on these curves.

Chapter 5 is about classical interpolation theory. It deals with divided differences, about differen forms of interpolation and Hermite interpolation. Finally, we look at piecewise polynomial-based curves that are more convenient to use for interpolation.

Chapter 6 is about B-splines and B-spline curves of different types, cubic spline interpolation, history of B-splines, modern B-splines and uniform B-splines and subdivision curves.

Chapter 7 is about blending, ie blending functions and how to use them. B-functions, short for blending functions exist in a great variety, and this chapter will show some of the most important ones.

Chapter 8 is about B-splines with B-functions, and also where we replace control points with control curves. Thus, this chapter introduce the concept of blending spline. Gives a lot of examples also about dynamic change of form. Finally a sub-curve

construction is given, where and formula for any parametric curve can be expanded with a knot vector so that the curve can change shape.

The six chapters about surfaces are:

Chapter 9 introduces parametric surfaces. Both definitions and aspects concerning implementation are discussed. In addition, different surface construction are shown.

Chapter 10 is about sub-division surfaces, ie uniform discrete B-splines where we start with a set of points organized in polygons, and end up with a much denser set of points and polygons.

Chapter 11 is about a blending surface construction where only two surfaces blend, and where the edges are given, both position and cross derivative. Ie a competitor to Coons patch.

Chapter 12 introduces tensor product blending surfaces. Both definitions and aspects concerning implementation are discussed. In addition, complete evaluators are introduced, as are Bézier surfaces as local surfaces. Hermite interpolation is discussed, and free form sculpturing using affine transformation of local surfaces is also discussed. Finally a sub surface concept is given, where formulas for any parametric surface can be expanded with two knot vectors so that the surface can change shape.

Chapter 13 deals with triangular surfaces. First there is a short repetition/definition of homogeneous barycentric coordinates and Bézier triangles. Then B-functions in homogeneous barycentric coordinates are introduced, and the basic properties are discussed. Then the blending-triangle is introduced, based on two types of local triangles, the Bézier triangle and the sub-triangle in general parameterized surfaces. The last one leads to surface approximations on triangulations. Finally, many examples are shown.

Chapter 14 is about surfaces that are a patchwork of triangles. Each triangle is the result of blending triangular pieces of local surfaces that are connected to their respective vertices. The surface is continuous and smooth at the vertices, but not necessarily smooth over the edges. However, there is a dual set of square patches that provide a completely smooth surface.

And finally we have three appendices:

Appendix A deals with the implementation of an evaluator for ERB-functions of type 1. We want a reliable, precise and efficient evaluator for the ERB-function. Reliability based on the IEEE standard for binary floating arithmetic is discussed. It also introduces algorithms for evaluation, including the derivatives. Tests are performed both in terms of precision and efficiency. Finally, a reliable and very fast evaluator is introduced, wrapped in a C++ class.

Appendix B is about external programming libraries to solve linear algebra, and to make better use of computer resources, ie heterogeneous computing and parallelization.

Appendix C Here, a number of proofs from different chapters are collected.

Chapter 2

Mathematical spaces and notations

Throughout the ages, "space" has been a geometric abstraction of the three-dimensional space observed in real life. In mathematics, Euclidean space is the two or three-dimensional space of Euclidean geometry, as well as the generalizations of these notions to higher dimensions. The term "Euclidean" distinguishes these spaces from the curved spaces of non-Euclidean geometry, and is named for the Greek mathematician Euclid of Alexandria (300 bc). It is common to define Euclidean spaces using Cartesian coordinates that we find in the real coordinate space and which are typically described by a center point called the origin and n unit vectors (coordinate axes) that are orthogonal to each other. It is common to denote the spaces $\mathbb{E}^n$ if we wish to emphasize its Euclidean nature, but $\mathbb{R}^n$ is used as well since the latter is assumed to have the standard Euclidean structure.

In modern mathematics, spaces are defined as sets with some added structure. They can also be described as different types of manifolds, which are spaces that is locally equivalent to Euclidean space, and where the properties are defined largely on local connectedness of points that lie on the manifold. Curves and Surfaces are manifolds if they are regular and not self intersecting.

There are however, many diverse mathematical objects that are called spaces. For example, vector spaces such as function spaces may have infinite numbers of independent dimensions and a notion of distance very different to Euclidean space, and topological spaces replace the concept of distance with a more abstract idea of nearness.

Mathematical spaces often form a hierarchy, i.e., one space may inherit all the characteristics of a parent space. For instance, all inner product spaces are also normed vector spaces, because the inner product induces a norm on the inner product space such that

$$\|s\| = \sqrt{\langle s, s \rangle}.$$

Beside Euclidean spaces, vector spaces and finite dimensional function spaces we shall in the following look at Compact spaces, Affine spaces, Projective spaces and Grassmannien, and we will look at some maps (functions) between spaces.

2.1 Euclidean spaces, cartesian coordinates and vector spaces

The notation in this book is following standard commonly used. Because geometry is in most used embedded in the plane or in the three dimensional space the functions are commonly vector valued or point valued (affine space will be described in section 2.4). A vector space is a mathematical structure formed by a collection of elements called vectors, which may be added together and multiplied /scaled by numbers, called scalars in this context. Scalars are often taken to be real numbers, but they can also be something else.

A list explaining notations is:

- ✓ We denotes Euclidean spaces $\mathbb{E}^d$ or $\mathbb{R}^d$, where d is the dimension of the space, $\mathbb{R}^2$ is the plane and $\mathbb{R}^3$ is the 3D space.
- ✓ Cartesian coordinate system, is typical used for Euclidean spaces. It specifies each point uniquely in an Euclidean space by a set of numerical coordinates, which are the signed distances from the point to fixed perpendicular directed lines, measured in the same unit of length, i.e. $p = (x, y, z)$.
- ✓ Each reference line of a Cartesian coordinate system is called a coordinate axis or just axis of the system, and the point where they meet is called its origin, $\mathscr{O} = (0,0,0)$. The coordinates can also be defined as the positions of the perpendicular projections of the point onto the axes, expressed as a signed distances from the origin.
- ✓ A vector or point is notated with a letter, Latin or Greek. A vector can be expressed as either a row-vector or a column-vector,

$$r = (r_1, r_2, r_3) = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} \quad \in \mathbb{R}^3.$$

- ✓ An inner product between two vectors r and $s \in \mathbb{R}^d$, $d > 1$ is denoted

$$\langle r, s \rangle = (r_1, \cdots, r_d) \begin{pmatrix} s_1 \\ \vdots \\ s_d \end{pmatrix} = (r_1 s_1 + r_2 s_2 + ... + r_d s_d) \quad \in \mathbb{R}.$$

  Note that if $\langle r, s \rangle = 0$, then r and s are othogonale, ie the angle between them is 90°. An inner product in an Euclidean space is the same as the scalar and dot product.
- ✓ A vector product in $\mathbb{R}^3$ (related to wedge product) is, given the vectors $r = (r_1, r_2, r_3)$ and $s = (s_1, s_2, s_3)$:

$$r \wedge s = \left(\begin{vmatrix} r_2 & r_3 \\ s_2 & s_3 \end{vmatrix}, \begin{vmatrix} r_1 & r_3 \\ s_1 & s_3 \end{vmatrix}, \begin{vmatrix} r_1 & r_2 \\ s_1 & s_2 \end{vmatrix} \right) = (r_2 s_3 - s_2 r_3,\ r_1 s_3 - s_1 r_3,\ r_1 s_2 - s_1 r_2)$$

  It can be shown that the vector product is orthogonal to both of its vectors, i.e.

$$\langle r \wedge s, r \rangle = \langle r \wedge s, s \rangle = 0.$$

✓ A wedge product in $\mathbb{R}^2$ is, given the vectors $r = (r_1, r_2)$ and $s = (s_1, s_2)$. A wedge product is

$$r \wedge s = \begin{vmatrix} r_1 & r_2 \\ s_1 & s_2 \end{vmatrix} = r_1 s_2 - s_1 r_2$$

A wedge product in $\mathbb{R}^2$ is the same as the "rotated" inner product

$$\langle r, s^L \rangle = r \wedge s,$$

where the vector s^L is vector s rotated 90° counter clockwise.

2.2 Homeomorphism, diffeomorphism and manifolds

We will short (and superficial seen from a mathematical point of view) look at the mathematical foundation of parametric curves and surfaces. To those readers who wish to study this more thoroughly and more seriously, we recommend reading Spivak [150] or DoCarmo [49, 50].

We start by explaining homomorphism. From Greek, meaning something like "similar shape", and in a geometric context it can be compared to a sheet of clay that is deformed by stretching, bending, ... but where we can not cut and glue. Neighboring elements must remain neighboring elements. A more formal explanation is

Definition 2.1. *A homeomorphism, ie a continuous transformation, is an equivalence relation and one-to-one correspondence between points in two geometric objects or topological spaces, that is continuous in both directions. It is a map which preserve all the topological properties of a given space. Two objects/spaces with a homeomorphism between them are called homeomorphic, and from a topological viewpoint they are the same. A transformation / map / function is a homeomorphism if it:*

- *is a bijection, i.e. one to one and onto,*
- *is continuous,*
- *the inverse function is continuous.*

A homeomorphism which also preserves distances is called an isometry. Affine transformations as rotation, scaling, translation, sharing are another type of common geometric homeomorphism. An isometry is like bending a sheet of paper to become part of a cylinder.

A related expression is diffeomorphism,

Definition 2.2. *A diffeomorphism is a one-to-one continuously-differentiable mapping. It is an invertible function that maps one differentiable manifold to another, such that both the function and its inverse are smooth. A function is a diffeomorphism if:*

- *the function is differentiable,*
- *the inverse function is differentiable,*

A manifold is an object / space that locally resembles Euclidean space around each point, ie there is a homeomorphism between an open set around each point and an Euclidean

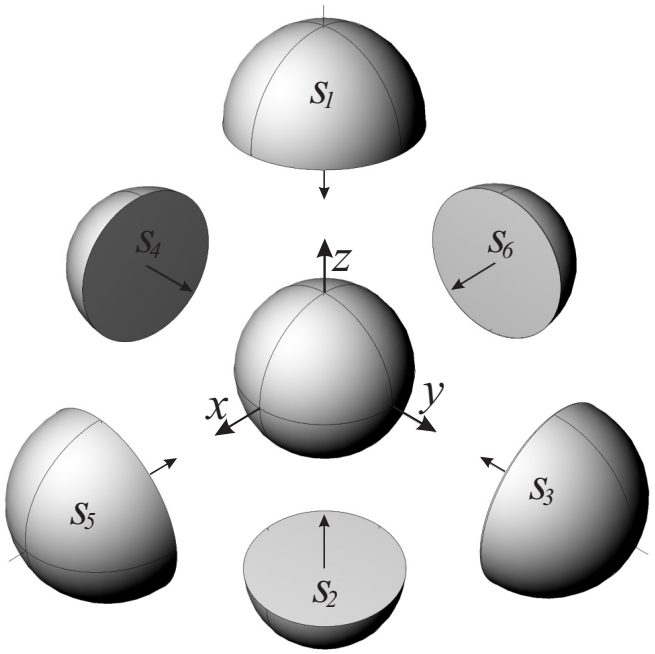


Figure 2.1: *Example of Sphere that is covered by an atlas that has six charts/maps. The six maps are denoted S_1, S_2, S_3, S_4, S_5 and S_6.*

space. One-dimensional manifolds are typically curves. Two-dimensional manifolds are also called surfaces.

2.2.1 Local/global parametrization, charts and atlas

We first look at what we can call local curves and surfaces and their parametric form.

1. A local parametric curve is a curve defined by a parametric equation, involving one parameter, most commonly s or t. Typically they will be curves in $\mathbb{R}^2$ or $\mathbb{R}^3$ (and commonly denoted $c(t)$). More precisely, a parameterized differentiable curve is a differentiable map

$$c : I \subset \mathbb{R} \to \mathbb{R}^n, \quad n \in (\mathbb{Z}^+ \quad \text{but usually } \{2,3\}),$$

i.e. from an open interval $I = (a,b)$ of the real line $\mathbb{R}$ into $\mathbb{R}^n, n = 2,3$.

2. A local parametric surface is a surface defined by a parametric equation, involving two parameters, most commonly (u,v) or (s,t). Typically they will be surfaces in $\mathbb{R}^3$ (and commonly denoted $s(u,v)$). More precisely, a parameterized differentiable local surface is a differentiable map

$$s : U \subset \mathbb{R}^2 \to \mathbb{R}^n, \quad n \in (\mathbb{Z}^+ \quad \text{but generally } \{3\}).$$

i.e. from an open set $U \subset \mathbb{R}^2$ into $\mathbb{R}^3$.

Both curves and surfaces are usually defined on closed or half open domains $\overline{I}$ or $\overline{U}$ that are subsets of the open domains used in the definition.

Every curve can be described with one parametrization. This might not always be practical and it is actually possible to use several intersecting intervals (with nonempty intersections) as domains for several different parameterizations which together cover the whole curve.

For surfaces, it is not always possible to parameterize a surface using one parametrization. This especially applies to parameterized, bounded, compact and connected surfaces of different topological genus g (number of holes/handles). One classical example is the sphere. There is no homeomorphism (Definition 2.1) between a sphere and $\mathbb{R}^2$. But if a sphere is being punctured, ie one point is taken away, then the rest of the sphere and $\mathbb{R}^2$ is homeomorphic.

In Figure 2.1 there are six parametrization that together cover a sphere,

$$\begin{aligned}
s_1(u,v) &= \left(\ u, \quad v, \quad \sqrt{1-(u^2+v^2)} \ \right), \\
s_2(u,v) &= \left(\ u, \quad v, \quad -\sqrt{1-(u^2+v^2)} \ \right), \\
s_3(u,v) &= \left(\ u, \quad \sqrt{1-(u^2+v^2)}, \quad v \ \right), \\
s_4(u,v) &= \left(\ u, \quad -\sqrt{1-(u^2+v^2)}, \quad v \ \right), \\
s_5(u,v) &= \left(\ \sqrt{1-(u^2+v^2)}, \quad u, \quad v \ \right), \\
s_6(u,v) &= \left(\ -\sqrt{1-(u^2+v^2)}, \quad u, \quad v \ \right),
\end{aligned}$$

where for all six maps, $\{s_i\}_{i=1}^{6}$, the domain is the open disk $u^2+v^2<1$.

If a curve or a surface is regular and not self intersecting, it can be considered as a manifold. This is a topological space that is locally homeomorphic to Euclidean space[1] by a collection (called an atlas) of homeomorphisms called charts. The composition of one chart with the inverse of another chart is a function called a transition map, and defines a homeomorphism of an open subset of Euclidean space onto another open subset of Euclidean space.[2]

Remark 1. *As will be shown later, in the construction of blending splines both local and global maps are used (in local and global geometry). This can be seen in both curves and tensor product surfaces. Surfaces that is a patchwork of triangles are even more in accordance with the definition of manifolds, thus, there is no global parametrization. This opens for consistent constructions of bounded, compact and connected surfaces of all possible topological genus.*

2.3 Compact spaces

Curves and surfaces or geometric objects in general are usually geometrically bounded or "geometrically not infinite", and in general they also includes their endpoints or edges.

They are denoted as compact objects (manifolds or spaces). These are curves including the start and end point, rectangles including the four edges, surfaces in general including their edges, or a sphere, a torus, etc.

[1] That is, around every point, there is a neighborhood that is topologically the same as the open unit ball in $\mathbb{R}^2$ or respective $\mathbb{R}^3$, and in fact the whole of $\mathbb{R}^2$ or $\mathbb{R}^3$ itself.

[2] A closer study of manifolds can be found in [150] or [50].

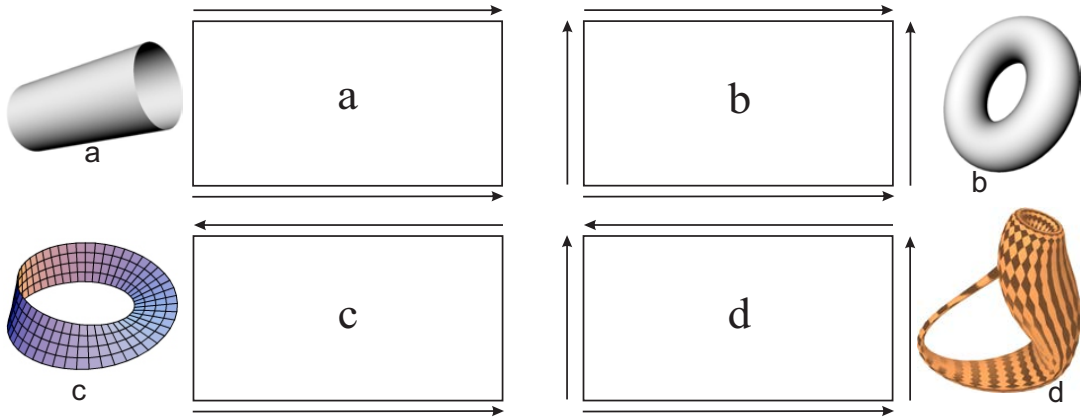


Figure 2.2: *Four examples of 2D compact objects (sets), cylinder, torus, Möbius band and Klein bottle. A cylinder has two edges, a Möbius band has only one edge, and a torus and a Klein bottle have none edges.*

Formally, a topological space X is called compact if each of its open covers (an infinite set of open sets covering the space totaly) has a finite subcover (can be reduced to a finite set open sets covering the space totaly) . Otherwise it is called non-compact.

In Figure 2.2 is there four examples of compact objects.

a) The rectangle **a** is bent so that two edges can be glued together. The result is a cylinder.

b) The rectangle **b** is first bent and glued to a cylinder, then the cylinder is bent and the other two edges are also glued together. The result is a torus.

c) The rectangle **c** is bent but now the two edges are turned before they are glued together. The result is a Möbius band.

d) The rectangle **d** is bent but now the two edges are turned before they are glued together, then it is also bent in the other direction and the other two edges are then glued together. The result is Klein bottle.

The Euclidean space and the Affine space described in the next section are not compact spaces (they do not include points at infinite), but the Projective and Grassmann spaces are compact spaces as we will se in section 2.5.

2.4 Affine space

A vector space is a set of elements we call vectors. A vector space is closed under finite vector addition and scalar multiplication. This means that a sum of two vectors also is a vector and to scale a vector by scalar multiplication also give a vector. This gives the following legal operations,

$$v = k_1\ v_1 + k_2\ v_2$$

where v, v_1 and v_2 are vectors and k_1 and k_2 are scalars (real numbers). It follows that you can sum many vectors into one if the number of vectors to sum is finite. In the previous

section we recognized that we not only have vectors but that we often have points instead. Points behave differently than, but are also dependent on vectors.

We therefor introduce a new structure on Euclidean Spaces.

The affine space

is a space of points p, and associated vectors v. The following two operations describes the connection between points and vectors and operations on vectors:

$$\begin{aligned} p &= p_1 + k\ v, && \text{connection between points and vectors,} \\ v &= k_1\ v_1 + k_2\ v_2, && \text{operations on vectors only,} \end{aligned} \tag{2.1}$$

where the k's are scalars, i.e. real numbers.

◇ In addition there is one more legal operation. It is an operation on points only, and it is called the affine combination and will be further described below.

From the first line in (2.1), turning the expression we get

$$v = \widetilde{k}\ (p - p_1), \quad \text{where} \quad \widetilde{k} = \frac{1}{k}. \tag{2.2}$$

Further, from a combination of all three expressions above (2.1 and 2.2) we get:

The affine combination

also called the barycentric combination. **This is the only legal operations on points only** and is formulated as follows (where p_i are points and k_i, $i = 0, ..., n$, are scalars)

$$p = \sum_{i=0}^{n} k_i\ p_i, \quad \text{where} \quad \sum_{i=0}^{n} k_i = 1. \tag{2.3}$$

◇ The name indicate to compute the barycenter. It is to sum up weighted points where the weights sum up to 1.

◇ If all weights $k_i,\ i = 0, \ldots, n$, are nonnegative, $k_i \geq 0,\ i = 0, 1, ..., n$, we call (2.3) for a **convex** affine combination.

The affine combination follows because (2.3) can be rewritten to fit the expression in (2.1),

$$p = p_0 + v, \quad \text{where} \quad v = \sum_{i=1}^{n} k_i\ (p_i - p_0),$$

and it therefore follows that

$$k_0 = 1 - \sum_{i=1}^{n} k_i.$$

Note from the Hermite basis functions (page 40) that the two basis functions blending points was summing up to 1. Also note that the basis functions for Bézier curves, the set of Bernstein polynomials that are only blending points sums up to 1 for all degrees (lemma 4.2 on page 49).

Later we will show that this is the case also for B-splines and NURBS.

The reason why this is so important is that to fulfill an affine combination are invariant under affine maps. These are the most used maps in computer graphics, CAD/CAM etc.

affine maps

are translation, scaling, rotation, shear and parallel projections, and are in general taken on the familiar form

$$\Theta p = A\ p + v \tag{2.4}$$

where p is a point and v is an associated vector in an affine space. If this is as usual $\mathbb{R}^3$, then A is a 3×3 matrix.

Geometrically, an affine map (affine transformation) in an Euclidean space is one that preserves:

1. The collinearity relation between points; ie three points which lie on a line continue to be collinear after the transformation.
2. The ratio of distances along a line; ie for distinct collinear points $p_1,\ p_2,\ p_3$, the ratio $\frac{|p_2-p_1|}{|p_3-p_2|}$ is preserved.

An affine map is invertible if and only if the matrix A is invertible. This is typically scaling, rotation, share and translation. Parallel projection is not invertible. A scaling matrix is a diagonal matrix where the inverse is a matrix where each number on the diagonal is inverted. This matrix is scaling a vector different in each coordinate.

$$\mathbf{A}_s = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{pmatrix}, \quad \text{and where} \quad \mathbf{A}^{-1} = \begin{pmatrix} \frac{1}{\alpha} & 0 & 0 \\ 0 & \frac{1}{\beta} & 0 \\ 0 & 0 & \frac{1}{\gamma} \end{pmatrix}.$$

A rotational matrix is an orthonormal matrix where the transposed matrix is the inverse. Rotating with the angle α counter clockwise around the z-axis

$$\mathbf{A}_{\alpha,x} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}, \quad \text{and where} \quad \mathbf{A}^{-1} = \mathbf{A}^T.$$

Rotating with the angle α counter clockwise around the y-axis

$$\mathbf{A}_{\alpha,y} = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix}, \quad \text{and where} \quad \mathbf{A}^{-1} = \mathbf{A}^T.$$

Rotating with the angle α counter clockwise around the z-axis

$$\mathbf{A}_{\alpha,z} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \text{and where} \quad \mathbf{A}^{-1} = \mathbf{A}^T.$$

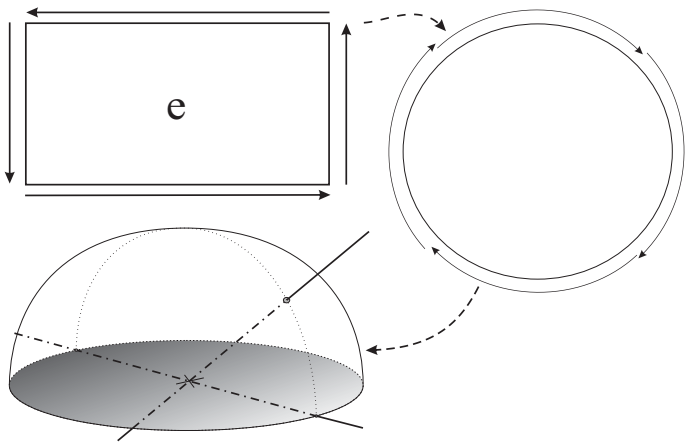


Figure 2.3: *An illustration of a projective plane $\mathbb{P}^2$. We start with a plane with four edges. Two and two edges are glued together, one edge and the edge on the opposite side are glued after one of them are turned. The plane is then deformed to a circular plate where the antipodal point on the boundary are the same point. The plate is then deformed to a hemisphere where the boundary is still in the xy-plane. There is now a one to one map (homeomorphism) to the set of all infinite straight lines through origin in $\mathbb{R}^3$.*

If we want to rotate around an arbitrary vector, we must do the following. First we restrict the rotation vector r to be a unit vector,

$$\mathbf{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \text{where} \quad |\mathbf{r}| = 1.$$

We then introduce the matrices ($\otimes$ is notation for outer product)

$$T = \mathbf{r} \otimes \mathbf{r}^T = \begin{bmatrix} x^2 & xy & xz \\ xy & y^2 & yz \\ xz & yz & z^2 \end{bmatrix} \quad \text{and} \quad S = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}.$$

And finally the rotation matrix is

$$\mathbf{R}_{\alpha,r} = T + \cos\alpha\ (I - T) + \sin\alpha\ S, \tag{2.5}$$

and where $\mathbf{R}^{-1} = \mathbf{R}^T$

2.5 Projective space and Grassmannien

A projective space is the space of one-dimensional vector subspaces of a given vector space. $\mathbb{P}^n$ denotes a projective space of dimension n. In general we assume it to be a projective space on real numbers. A more precise notation is $\mathbb{RP}^n$ for real projective spaces, and $\mathbb{CP}^n$ for complex projective spaces and so on.

$\mathbb{P}^n$ can also be viewed as the set consisting of $\mathbb{R}^n$ together with its points at infinity (in Figure 2.3 is this the edges glued with each other).

In Figure 2.3 is there an illustration of the projective plane, $\mathbb{P}^2$. We start with a plane where the edges are glued together, one edge is glued with the edge on the opposite side, but turned before they are glued together. The figure shows us the maps/homeomorphism from the initial plane via a plate where the antipodal points are the same point and then an hemisphere including the edge circle in the xy-plane, to the set of all infinite lines through origin. This follows because there is a one to one map between the hemisphere and the set of lines. This is obvious for all points on the hemisphere except for the edge, but it is also true for the lines in the xy-plane because the antipodal points are the same point because of the initial gluing.

It follows that $\mathbb{P}^2$ is the set of all lines through origin in $\mathbb{R}^3$, ie. a one-dimensional vector subspace. From the example in Figure 2.3 it is also clear that the projective plane $\mathbb{P}^2$ is compact.

A point in $\mathbb{P}^n$ can be described by $n+1$ cartesian coordinates but these coordinates can be scaled by any nonzero scalar. As an example, using this description it follows that $q \in \mathbb{P}^3$ can be expressed by

$$q = (kx, ky, kz, kw),$$

where q is independent of k, i.e. k can be any nonzero real.

There is a canonical injection of $\mathbb{R}^n$ into $\mathbb{P}^n$. This means that an affine space $\mathbb{R}^n$ can be embedded isomorphically in $\mathbb{P}^n$ by the standard injection

$$(x_1, \ldots, x_n) \mapsto (x_1, \ldots, x_n, 1). \tag{2.6}$$

Affine points can be recovered from projective ones with the mapping

$$(x_1, \ldots, x_n, x_{n+1}) \sim \left(\frac{x_1}{x_{n+1}}, \ldots, \frac{x_n}{x_{n+1}}, 1\right) \mapsto \left(\frac{x_1}{x_{n+1}}, \ldots, \frac{x_n}{x_{n+1}}\right). \tag{2.7}$$

In general can $\mathbb{P}^n$ be seen as the set of all lines through origin in $\mathbb{R}^{n+1}$. Contrary to Euclidean spaces $\mathbb{R}^n$ is the projective spaces $\mathbb{P}^n$ compact, as clearly can be seen in Figure 2.3. The points at infinite in $\mathbb{R}^n$ is the same as the horizontal lines in $\mathbb{P}^n$, so the projective space also includes the points at infinite and is therefor compact.

Grassmannian is a generalization of the Projective spaces; it is the space of d-dimensional vector subspaces of a given n-dimensional vector space, $0 < d < n$, and is denoted $Gr(n, d)$. It follows that a projective space $\mathbb{P}^n = Gr(n, 1)$.

For example, $Gr(n, 2)$ can be the space of all planes through origin in an Euclidean space $\mathbb{R}^n$.

2.6 Homogeneous coordinates

Homogeneous coordinates, introduced by August Ferdinand Möbius in 1827 (Der barycentrische Calcül), is a system of coordinates used in projective geometry much like Cartesian coordinates are used in Euclidean geometry. They have the advantage that the coordinates

of points, including points at infinity, can be represented using finite coordinates. This because the projective space is compact and thus includes the points at infinity. Formulas involving homogeneous coordinates are often simpler and more symmetric than their Cartesian counterparts. Homogeneous coordinates have a range of applications, including computer graphics and 3D computer vision, where they allow affine transformations and, in general, projective transformations to be easily represented by a matrix.

Homogeneous coordinates have one coordinate more than the dimension of the space. It follows that if the last coordinate is not 0, then any scaled version of the element are the same element, so to compare we chose a common value for the last coordinate. If the dimension is 3, then we have

Point $\mathbf{p} = (p_x, p_y, p_z, 1)$, ie the last coordinate is 1.

Vector $\mathbf{v} = (v_x, v_y, v_z, 0)$, ie the last coordinate is 0.

Thus, in homogeneous coordinates, scaling of a point is impossible because if a point is multiplied by a non-zero scalar, the resulting coordinates represent the same point. However, a vector can be scaled because the last coordinate is still 0 after the scaling.

We see that to follow this scheme makes operation in affine space more complete, summing points give no meaning, but summing weighted point , where the weighted sum up to 1 gives a point, adding a vector to a point give a point (last coordinate is 1) and so on.

Especially affine maps get a much simpler expression because we now only get one matrix for all operations. Remember that an affine map (2.4) is on the form $\mathbf{A}p + \mathbf{v}$. There is a point p that is processed by, a matrix $\mathbf{A}$ which is a combination of scaling, rotation and sharing, and a translation vector $\mathbf{v}$, ie

$$\mathbf{A} = \begin{bmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{bmatrix} \quad \text{and} \quad \mathbf{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Putting this into one matrix will be on the form

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{v} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x_x & y_x & z_x & v_x \\ x_y & y_y & z_y & v_y \\ x_z & y_z & z_z & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.8}$$

and the inverse is

$$\mathbf{H}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{v} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x_x & x_y & x_z & -\langle \mathbf{x}, \mathbf{v} \rangle \\ y_x & y_y & y_z & -\langle \mathbf{y}, \mathbf{v} \rangle \\ z_x & z_y & z_z & -\langle \mathbf{z}, \mathbf{v} \rangle \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.9}$$

Only rotation makes $\mathbf{A}$ orthonormal (orthogonal and det $= 1$) and $\mathbf{A}^{-1} = \mathbf{A}^T$. If scaling is used, and if column number i is scaled by a then row number i in the inverse matrix is scaled by $\frac{1}{a}$.

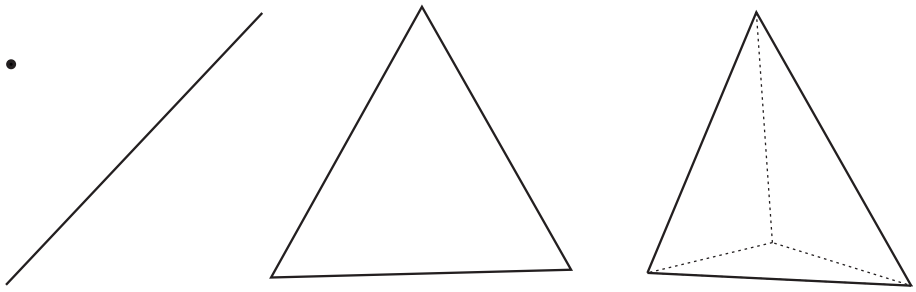


Figure 2.4: *We see examples of the first four simplexes. From the left we see a point - Δ_1, a line segment - Δ_2, a triangle - Δ_3 and a tetrahedron - Δ_4.*

2.7 Simplexes

In section 2.2.1, curves were defined as 1-dimensional object, surfaces as 2-dimensional. This is related to the coupling/homeomorphism with a Euclidean space and it manifests itself in the number of parameters in a parametrization. It follows that a point is 0-dimensional and a volume 3-dimensional, etc.

A polygon is thus a 2-dimensional object, i.e. a 2-dimensional polytope. We will look at the simplest polytopes of given dimensions, these call simplexes, Δ_n. The first 5 simplexes are:

- punkt	er en 1-simplex - Δ_1 - and has 1 point	the dimension is 0
- line segment	er en 2-simplex - Δ_2 - and has 2 points	the dimension is 1
- triangle	er en 3-simplex - Δ_3 - and has 3 points	the dimension is 2
- tetrahedron	er en 4-simplex - Δ_4 - and has 4 points	the dimension is 3
- 5-cell	er en 5-simplex - Δ_5 - and has 5 points	the dimension is 4

Note that the edges of an n-simplex are n pieces of (n-1)-simplexes. For example, a triangle has 3 edges (line segments). If we insert a point inside an n-simplex, we divide the simplex into n pieces of n-simplexes. A final observation that is easy to see for low dimensions is that for any point there exists a straight line (edge) between the point and all the other points. In figure 2.4 the four first simplexes are plotted.

2.8 Homogeneous Barycentric coordinates for simplexes

Barycentric coordinates were introduced by Möbius in 1827, see [65]. These can be used either to express a point inside a simplex Δ_n as a convex combination of the $n+1$ points in the simplex or to linearly interpolate data given in the points. The coordinates correspond to masses placed in the vertices and that the point itself is then the center of mass (barycentre). Much work has been done regarding the behavior and applications of the barycentric coordinates. Among others Warren, in [161] and later publications.

First, some facts about homogeneous barycentric coordinates. Note that unlike barycentric coordinates in general, homogeneous barycentric coordinates are normalized and thus

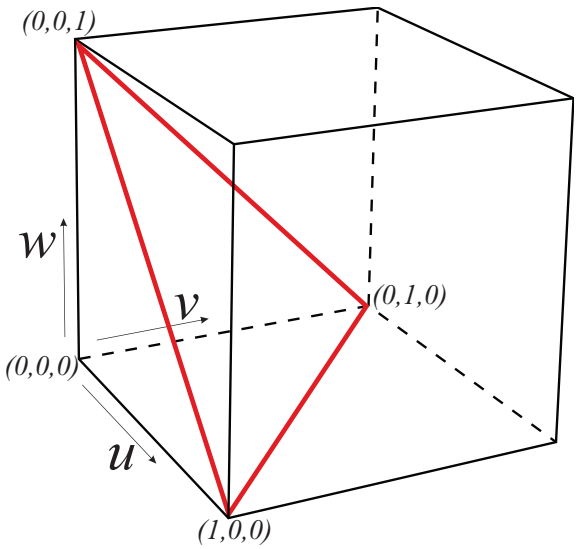


Figure 2.5: *A parametric unit cube described by the coordinates u, v, and w. "The main diagonal" is marked with red lines, and can be seen as a triangle in the figure.*

unique.[3] We start with triangles. We start with a surface $\Omega \subset \mathbb{R}^2$, with a Cartesian coordinate system (u,v). We then let this be a unit square. In figure 2.5, we have expanded the unit square to a unit cube where we have a coordinate w that is orthogonal to the other two coordinates. If we cut this cube with a plane passing through the three corners where one of the coordinates is 1 and the other two are 0, we get the main diagonal. In figure 2.5, the main diagonal is plotted as a triangle marked with red lines. This triangle (the Δ_2 simplex) is then a domain described in homogeneous barycentric coordinates. Note that the plane in which the triangle lies has the following implicit formula,

$$u+v+w=1.$$

At the same time, it is natural to restrict the domain to be in the triangle in figure 2.5, which then means the following restriction on the parameters,

$$u,v,w \geq 0.$$

the name "barycenter" is Latin for center of gravity. It refers to the coordinates of the "mass center" which are determined via a convex combination of the points of a material system of points. Here follows the definition of homogeneous barycentric coordinates for points:

Definition 2.3. *The convex set of points forming the main diagonal of an (n+1)-dimensional unit hypercube is the domain of an n-dimensional simplex denoted Δ_n. In homogeneous barycentric coordinates, a point $\mathbf{p} \in \Delta_n$ is defined by*

$$\mathbf{p} = \{u_i\}_{i=0}^n, \quad \textit{where} \quad \sum_{i=0}^{n} u_i = 1,$$

[3]Note the similarities between homogeneous barycentric coordinates vs. homogeneous coordinates in the projective space. Points sum to 1, vectors to 0 vs. last coordinate is 1 or 0 in the projective space.

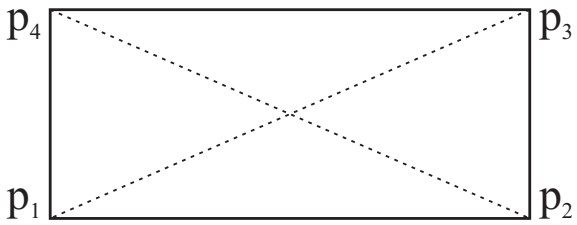


Figure 2.6: *We see a rectangle with corners p_1, p_2, p_3 and p_4. The two dashed lines (diagonals) are also plotted. Where these two lines intersect, one expects the same coordinates regardless of which line we follow.*

which fulfill the convexity property when

$$u_i \geq 0, \qquad i = 0,1,2,...,n.$$

A point $\mathbf{p} \in \Delta_2$ (point in the triangle in figure 2.5) is defined as follows

$$\mathbf{p} = (u,v,w), \quad \text{where} \quad u+v+w = 1 \quad \text{og } u,v,w \geq 0.$$

We then extend the definition of homogeneous barycentric coordinates to include vectors. In figure 2.5, we insert a new plane, which is parallel to the "main diagonal" triangle we created earlier, but which now passes through the origin. This new plane can be thought of as a 2D vector space in barycentric coordinates. The implicit formula for this plane is

$$u+v+w = 0$$

The definition of homogeneous barycentric coordinates for vectors is:

Definition 2.4. *We denote the n-dimensional vector space that passes through the origin and is parallel to the "main diagonal" of an (n+1)-dimensional unit hypercube for Υ_n. In homogeneous barycentric coordinates, a vector $\mathbf{d} \in \Upsilon_n$ is defined by*

$$\mathbf{d} = \mathbf{p}_2 - \mathbf{p}_1 = \{r_i\}_{i=0}^n, \quad \textit{hvor} \quad \sum_{i=0}^{n} r_i = 0 \quad \textit{and} \quad \mathbf{p}_1, \mathbf{p}_2 \in \Xi_n.$$

For a triangle (2-dimensional simplex), a vector $d \in \Upsilon_2$ is defined as follows;

$$\mathbf{d} = (r,s,t), \quad \text{where} \quad r+s+t = 0.$$

It can be shown that it is only for simplexes that homogeneous Barycentric coordinates are unique. For dimension 2 — surfaces, this can be shown easily. We start from a square convex polygon, for example a rectangle. We create the following formula

$$s(u_1,u_2,u_3,u_4) = u_1p_1 + u_2p_2 + u_3p_3 + u_4p_4, \quad \text{hvor} \quad u_1+u_2+u_3+u_4 = 1.$$

We then create the curves that connect the two opposite corners. We create them by setting the other two weights to 0, i.e. $c_1(t) = s(t,0,1-t,0)$ and $c_2(r) = s(0,r,0,1-r)$. These two curves will intersect at a certain t and r value. This is illustrated in figure 2.6. This shows that two different sets of coordinates give the same point.

However, there are "adjustments" that provide unique coordinates for convex polygons and also 3-dimensional polytopes. Michael Floater et al. launched mean values with barycentric coordinates in 2D in [67] and [68], and in 3D in [69].

Chapter 3

Implementing geometry in C++

All the theory and algorithms described in this book are implemented and tested, and all tables and figures, except Figure 1.1 and Figure 6.3, are thus made using these test programs. The implementations are done in C++ and are mainly based on an in-house open source programming library called GMlib .

Some operations are very resource intensive and can take a relatively large amount of time on a computer. These are operations that other people have spent a lot of time optimizing. This applies, for example, to matrix vector calculations with large matrices that are necessary for some algorithms described later. Here BLAS compatible program libraries can help. BLAS is an abbreviation for‘ Basic Linear Algebra Subprograms. In Appendix B.1 there is a description, as well as a list of BLAS compatible program libraries. To be able to use all available resources on the computer, there is a list of aids for “Heterogeneous computing and parallelization” in Appendix B.2.

3.1 Mathematical spaces for geometric programming

A combination of Affine and Projective spaces are the most convenient spaces to use in geometric programming. One reason to use Affine spaces is to be as independent as possible of the coordinate system. In an Affine space, origin is just one point among the other points, and nothing directly depends on it. For each point we can assign a vector space where the point itself is the origin. Thus, points can be moved using a vector. Another important application is local coordinate systems. In a geometric system (computer graphic) in $\mathbb{R}^3$, the vector space associated with a point can have three orthogonal unit vectors which together form a local coordinate system, located at the point.

In a projective space we can use homogeneous coordinates. Then an extra coordinate is added, 1 for points and 0 for vectors, see page 19. This fits perfect to Affine spaces, and there is a simple map between these spaces, (2.6) and (2.7). But in computer programming we sometimes want to use the data directly in external routines such as BLAS-compatible functions. Then we can not include the extra coordinate 1 for points and 0 for vectors. Thus a solution is to distinguish them by type as shown in the next section.

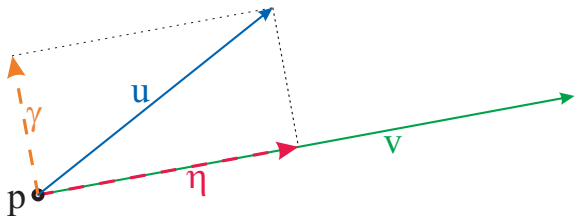


Figure 3.1: *A projection of a vector u on a vector v, both starting from a point p. The projection is visualized as a vector* η.

3.2 Homogeneous coordinates and programming

Points and vectors are the basic objects in affine spaces and thus geometric programming.

To distinguish the affine 2D / 3D / ... vector from the std::vector which is a container, we can use a capital first letter on them. It is especially advantageous to use template programming, because you can switch between single and double precision, choose dimension and also define types recursively. In template programming, it is also possible to create a separate version for specific template parameters. For example, Vector⟨T,3⟩ may have a vector product operator that produces a vector, while Vector⟨T,2⟩ may have the same operator that then produces a signed scalar.

We can make the following definition based on the types, *template⟨typename T, int n⟩*

Point⟨T,n⟩ - for example Point⟨double,3⟩ a point in a double precision 3D affine space.

Vector⟨T,n⟩ - for example Vector⟨float,3⟩ a vector in a single precision 3D vector space.

The relationship between Point and Vector must be as consistent as possible, and they should be closely linked, but not so narrowly defined that it causes problems. These two basic types can be expand to:

Simplex⟨T,n,m⟩ - Vector⟨Point⟨T,n⟩,m⟩. Note that Simplex⟨float,3,2⟩ is a line segment, Simplex⟨float,3,3⟩ is a triangle and Simplex⟨float,3,4⟩ is a tetrahedron.

Matrix⟨T,n,m⟩ - Vector⟨Vector⟨T,n⟩,m⟩. This is an $n \times m$ matrix. If $n = m$ then the matrix ia a square matrix and might be non singular and an inverse can be made.

HMatrix⟨T,n⟩ - Vector⟨Vector⟨T,n+1⟩,n+1⟩. Is an homogeneous matrix, see (2.8)

One operation that is heavily used is the inner product between two vectors. In Figure 3.1 is this operation visualized. We see that $\frac{\langle u,v\rangle}{\langle v,v\rangle} = \frac{|\eta|}{|v|}$ and that $\eta = \frac{\langle u,v\rangle}{\langle v,v\rangle}v$. If v is a unit vector (length 1) then the formula is $\eta = \langle u,v\rangle v$, and $|\eta| = \langle u,v\rangle$.

An homogeneous matrix can also represent a local coordinate system[1]. A coordinate system for $\mathbb{R}^3$ is typically described by three coordinate vectors, usually denoted x-axis, y-axis and z-axis. They are all unit vectors (length 1) and usually orthogonal, the angle between them are 90°. It follows that the inner product between them are all zero.

[1] In blending splines, local coordinate systems are used by local curves and surfaces, this will be described in chapters 8 and 12.

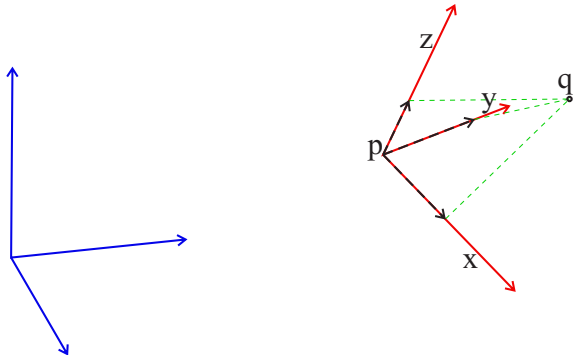


Figure 3.2: *A local coordinate system* **p** *with its red coordinate axes. The blue axis are the parent coordinate system. The point* **q** *is projected down on the red vectors, and the black vectors are the decomposition of q with respect to* **p** *and its coordinate axes.*

An homogeneous matrix is shown in (2.8) where it is connected to affine maps. But it can also be interpreted as a local coordinate system, ie.

$$\mathbf{H} = \begin{bmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} & \mathbf{p} \end{bmatrix} = \begin{bmatrix} x_x & y_x & z_x & p_x \\ x_y & y_y & z_y & p_y \\ x_z & y_z & z_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where **x**, **y** and **z** are the coordinate axes and **p** is a point that indicates the location of the local coordinate system, see Figure 3.2. If **q** is a point in the local coordinate system, and we multiply with **H**,

$$\widehat{\mathbf{q}} = \mathbf{H}\,\mathbf{q} = \mathbf{p} + q_x\,\mathbf{x} + q_y\,\mathbf{y} + q_z\,\mathbf{z},$$

where $\widehat{\mathbf{q}}$ is **q** in the parent coordinate system (the blue axis in Figure 3.2). The inverse of **H** was given in (2.9), it is

$$\mathbf{H}^{-1} = \begin{bmatrix} x_x & x_y & x_z & -\langle \mathbf{x}, \mathbf{p} \rangle \\ y_x & y_y & y_z & -\langle \mathbf{y}, \mathbf{p} \rangle \\ z_x & z_y & z_z & -\langle \mathbf{z}, \mathbf{p} \rangle \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

If we multiply $\widehat{\mathbf{q}}$ with this matrix we get

$$\mathbf{q} = \mathbf{H}^{-1}\,\widehat{\mathbf{q}} = \begin{bmatrix} \langle \mathbf{x}, \widehat{\mathbf{q}} \rangle - \langle \mathbf{x}, \mathbf{p} \rangle \\ \langle \mathbf{y}, \widehat{\mathbf{q}} \rangle - \langle \mathbf{y}, \mathbf{p} \rangle \\ \langle \mathbf{z}, \widehat{\mathbf{q}} \rangle - \langle \mathbf{z}, \mathbf{p} \rangle \\ 1 \end{bmatrix} = \begin{bmatrix} \langle \mathbf{x}, \widehat{\mathbf{q}} - \mathbf{p} \rangle \\ \langle \mathbf{y}, \widehat{\mathbf{q}} - \mathbf{p} \rangle \\ \langle \mathbf{z}, \widehat{\mathbf{q}} - \mathbf{p} \rangle \\ 1 \end{bmatrix}.$$

In Figure 3.2 is the local coordinate system illustrated.

Homogeneous matrices can be used in display hierarchy, ie a scene graph. In such a system, the scene is the reference coordinate system for all objects in the scene graph and can be used for simulations involving several objects. Thats why object should have two homogeneous matrices one local and one for the reference system.

The graphic system can be connected to virtual cameraes, which can also be objects in the scene graph, for example inserted into a moving car. In that case, the final coordinate system is that which is connected to the camera, and the inverse of the camera's reference matrix multiplied by the reference matrix of each object is the matrix to be inserted into the graphic system.

The next topic is simulation, and we will take a closer look at two examples of the use of local coordinate systems/homogeneous matrices in simulations.

We will first look at how we can lock the direction of a local axis towards another object. This can be used, for example, by a camera. Because each time step in a simulation is small, we assume that the changes are also small. At each time step, we start by finding the distance vector between the points in the two local coordinate systems. Then we set the desired axis, for example $\mathbf{x}$ equal to this, and normalize it. Then we correct $\mathbf{y}$ by

$$\mathbf{y} = \mathbf{y} - \langle \mathbf{y}, \mathbf{x} \rangle \mathbf{x},$$

and then normalize $\mathbf{y}$. Finally we set $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$. Now the direction is locked even though both objects are moving.

The second example is also useful for a camera, but can be used by many other types of objects as well. If you want to rotate an object without doing so, you can rotate the camera around the object, or around a point in the space. We can do this by locking the camera's direction towards the object/point, and at each time step moving the camera in the plane that is orthogonal to the axis pointing forward, ie $\mathbf{p} = \mathbf{p} + dy\ \mathbf{y} + dz\ \mathbf{z}$, see[2]. After each locking and moving operation is performed, we correct the position $\mathbf{p}$ in the homogeneous matrix so that the distance to the object/point is maintained.

3.3 Tools for interactive design

Some geometric objects, curves and surfaces may change shape. To do this interactively in graphics mode, we need some tools. These geometric objects are either control points and/or vectors that can be moved or changed. Vectors often change by moving the end-points. A tool for moving points in graphic mode is a **selector**. It is an object that has a reference to the point to be edited and that can report to the parent object of the point to update itself. A selector can be selected and moved using a mouse pointer. A selector can typically be displayed as a small ball. Remember that moving an object can be done based on the local coordinate system of the camera and then change the position of the point by following the translation vector first to the reference point (scene) and then out to the object.

Another tool is a **placeholder**. A curve or surface or another geometric object can be displayed on the screen by a placeholder. This is then an object that can be both moved and rotated and that is typically displayed as a cube on the screen. The motion then changes the point $\mathbf{p}$ while the rotation then changes the coordinate axes $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ in

[2]Here the yz-plane is the plane orthogonal to the direction of the camerae. dy and dz are scalars depending on the time step dt and the speed and direction of the mouse in the user interface.

the matrix of the object. A placeholder is used in "editing" blending splines that will be described in chapter 8, 12, 13 and 14.

3.4 Implementation, about curves and surfaces

In section 2.3 about compact spaces is stated that curves and surfaces for computer graphics are compact objects either including their endpoints/edges or is a circle like curve, sphere like surface, a torus like surface, ... A computer screen is a matrix of pixels, to display a curve we can calculate each pixel to see if there is an object there that is displayed as a curve with a given color. This is typically done in ray tracing. Another method that has been more common so far is to divide a curve into small line segments and then display each line segment. We call the process of dividing for tessellation. For surfaces, the same process is to divide into sets of small triangles. We call the corners of the triangles vertices. For surfaces, in addition to the position of the vertices, we also need the surface normals in the vertices. Therefore, all curves must have functions for calculating position, and all surfaces have functions for calculating position and surface normal at the sample points.

There are many ways to implement a geometry structure, either abstract based on dimension, properties and topology, or a simple inheritance structure where one must also distinguish between parametric and non-parametric type. We will look at a simple inheritance structure for parametric curves and surfaces. For curves, we must ensure that there are functions that define the parameter domain, we need a function to calculate the position and derivative in a parameter value, and we must know whether the curve is closed or open. All these functions can be defined in a base class as a 0-function, which means that all derived classes must implement these functions. For surfaces, we must also ensure that there are functions that define the parameter domain, we need a function to calculate the position and the partial derivatives in a parameter value, and we must know whether the surface is closed or open in both directions. As for curves, all of these functions can be defined in a base class as a 0-function, which means that all derived classes must implement these functions. However, the base classes can take care of the tessellation and contact with the graphics system themselves.

Part I

Curves

Chapter 4

Parametric Curves

Imagine the space of real numbers $\mathbb{R}$ as an infinitely straight line. To make a curve we just pick a segment of this infinite straight line. We deform it by stretching or pressing it and bending it in different ways. We put it into a space that can be a plane (Euclidian – $\mathbb{R}^2$) or a 3 dimensional space, (Euclidian – $\mathbb{R}^3$). Finally we place it in the desired position with the desired orientation and scale. The result is a parametric curve. Note that cutting and gluing is *not* an option here. Figure 4.1 give an example of this concept. We categorize a curve as a 1-dimensional object (one parameter). If we put constraints on a parametric curve such that the curve does not degenerate to a point or intersect itself, then a curve can be called a 1-dimensional manifold (see [150] or [50]).

This is an attempt to visualize the concept of a parametric curve. We can also think about a parametric curve as a path of an object in motion and where the path also contains a time indication of when the object is there. To do this in a more mathematical way we first generalize the output not only to be in $\mathbb{R}^2$ or $\mathbb{R}^3$ but more general in $\mathbb{R}^n$, $n > 0$. A more formal definition is:

Definition 4.1. *A parameterized differentiable curve is a differentiable map $\alpha : I \longrightarrow \mathbb{R}^n$ of an open interval $I = (a,b)$ of the real line $\mathbb{R}$ into $\mathbb{R}^n$. (Note that a half open or a closed*

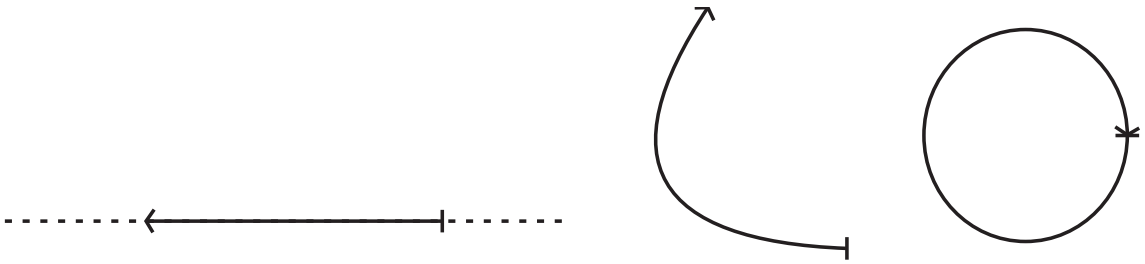


Figure 4.1: *On left hand side is a part of the real numbers plotted as a dashed line, the half open interval $I = (0, 2\pi]$ is marked as solid. In the mid-figure is the interval bent and finally, on right hand side, is it bent into a circle. If the circle is a unit circle with radius $r = 1$ as in expression (4.1), then the length of the interval and the curve is the same, there is no stretching, while expression (4.2) is stretching to twice the length.*

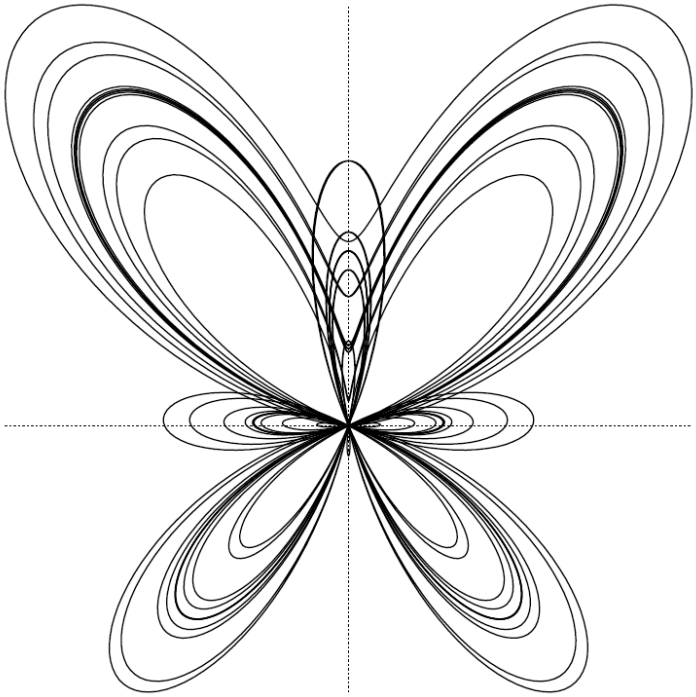


Figure 4.2: *A plot of the butterfly curve following from expression (4.3).*

interval is just a restriction of an open interval.)

The first example is a circle in a plane, where the parameter interval $I = (0, 2\pi]$ is half open. In the expression is therefor t going from 0 (not included) to 2π (included). A given t-value results in a vector with an x and y component, i.e. a vector in the Euclidean space $\mathbb{R}^2$. Therefore, we call the following function (4.1) for vector-valued;

$$\alpha(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}, \quad 0 < t \leq 2\pi. \tag{4.1}$$

The center of the circle defined in (4.1) is in origin and the radius is 1. If we want to change the radius of the circle to $r = 2$, and move the center to another position, for example to a point with coordinates $x = 1$, $y = 2$ we get:

$$\alpha(t) = 2\begin{pmatrix} \cos t \\ \sin t \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2\cos t + 1 \\ 2\sin t + 2 \end{pmatrix}, \quad 0 < t \leq 2\pi. \tag{4.2}$$

A more sophisticated curve example is the butterfly, made by Temple H. Fay in 1989 and published in [66]. The curve can be seen in Figure 4.2 and the equation is as following:

$$\alpha(t) = \begin{pmatrix} \left(e^{\cos t} - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right)\sin t \\ \left(e^{\cos t} - 2\cos(4t) - \sin^5\left(\frac{t}{12}\right)\right)\cos t \end{pmatrix}, \quad 0 < t \leq 24\pi. \tag{4.3}$$

This curve, figure 4.2, is 12 following circles deformed by cyclic changing and collapsing radius as can be seen from equation (4.3). Parametric curves can be made by all types

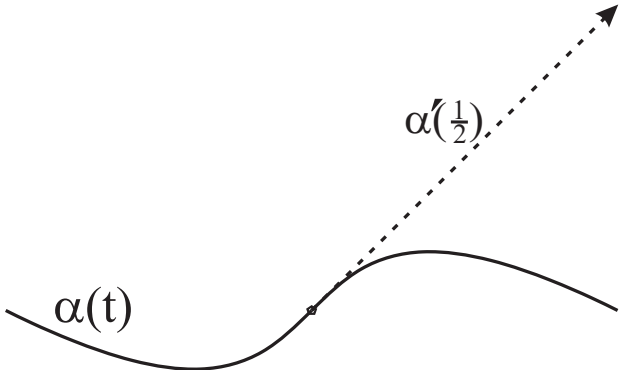


Figure 4.3: *A plot of the polynomial based curve from expression (4.4). The* 1^{st}*-derivative and tangent vector* $\alpha'\left(\frac{1}{2}\right)$ *is also plotted.*

of functions, trigonometric functions, logarithmic functions, exponential functions and of course polynomials that has been among the most popular choices. The next example is therefore a 2D - vector valued parametric curve based on polynomials:

$$\alpha(t) = \begin{pmatrix} 6t - 9t^2 + 6t^3 \\ -3t + 9t^2 - 6t^3 \end{pmatrix}, \quad 0 \leq t \leq 1. \tag{4.4}$$

This curve (4.4) can be seen in Figure 4.3, and we see that the curve starts at the origin of the plane. Note that for a given t-value, $\alpha(t) = (x(t), y(t))$ is a point on the curve. The variable t is called the parameter of the curve and the parameter interval $I = [0,1]$ is called the domain of the curve. The image $\alpha(I) \subset \mathbb{R}^2$ is called the trace of α. And the word differentiable means that derivatives of $x(t)$ and $y(t)$ exist everywhere on the domain.

Vector spaces (Linear spaces)

Before we move on, let's look briefly at the concept of vectors and sets of vectors called a vector space. In general, we expect vector spaces, normed vector spaces and inner product spaces and their properties to be known to the reader.

Definition 4.2. *A vector space is a set that is closed under finite vector addition and scalar multiplication (ie the result of the operations are elements in the vector space). The most common example is n-dimensional Euclidean spaces* $\mathbb{R}^n$*, where each element is a list of n real numbers, where scalars are also real numbers, additions are componentwise, and scalar multiplication is multiplication on each term separately.*

In this book is a classic vector notated with a letter, for example v. If $v \in \mathbb{R}^2$ (element in a 2D plane) it will have 2 coordinates $v = (x, y)$, can also be written transposed as $v = \begin{pmatrix} x \\ y \end{pmatrix}$. If $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ are vectors $\in \mathbb{R}^2$ and k is a scalar $\in \mathbb{R}$ then

$$v = v_1 + kv_2 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + k \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 + kx_2 \\ y_1 + ky_2 \end{pmatrix}$$

An inner product, which is a scalar, will have the following notation

$$\langle v_1, v_2 \rangle = x_1 x_2 + y_1 y_2$$

A norm (length) of a vector is defined by

$$|v| = \sqrt{\langle v, v \rangle} = \sqrt{x^2 + y^2}$$

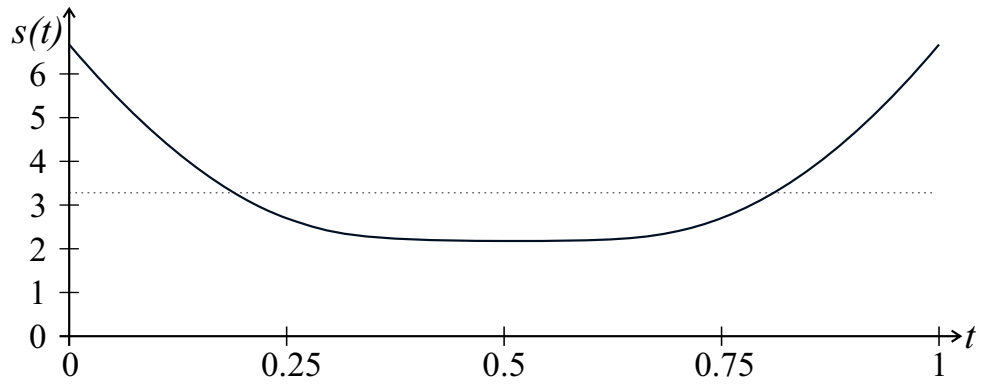


Figure 4.4: *The figure shows the speed $s(t) = |\alpha'(t)|$ of the curve from expression (4.4).*

4.1 Differentiations

Recall that a curve $\alpha(t)$ can be considered as a path to an object that moves in time t.

- We denote the first derivative[1], $\frac{d\alpha}{dt}(t)$ as $\alpha'(t) = (x'(t), y'(t))$.
- $\alpha'(t)$ is called the tangent or velocity vector of the curve at the point $\alpha(t)$.
- The length of the tangent/velocity vector $|\alpha'(t)|$ is called the speed at the point $\alpha(t)$.

In Figure 4.3, the tangent vector at $\alpha(0.5)$ is plotted. We can clearly see that it is tangential to the curve. Because the parameter range is 1, from 0 to 1, the average length of the tangent/velocity vector will be equal the curve length. The speed of the curve $s(t) = |\alpha'(t)|$, $t \in [0,1]$ defined in (4.4) is plotted in Figure 4.4. We can calculate the curve length from the average speed times the time used. In Figure 4.4, this is the same as the area under the function $s(t)$, $t \in [0,1]$. In general, we get the following integral for computing the curve/arc length over the interval $I = [a,b]$:

$$l(\alpha)_I = \int_a^b s(t)\, dt = \int_a^b |\alpha'(t)|\, dt = \int_a^b \sqrt{\langle \alpha'(t), \alpha'(t) \rangle}\, dt \tag{4.5}$$

Using the curve defined in (4.4) we get the speed function

$$\begin{aligned} s(t) &= \sqrt{(18t^2 - 18t + 6, \quad -18t^2 + 18t - 3) \begin{pmatrix} 18t^2 - 18t + 6 \\ -18t^2 + 18t - 3 \end{pmatrix}} \\ &= 3\sqrt{72t^4 - 144t^3 + 108t^2 - 36t + 5}. \end{aligned}$$

And the curve/arc length is

$$l(\alpha)_{0,1} = \int_0^1 3\sqrt{72t^4 - 144t^3 + 108t^2 - 36t + 5}\, dt \approx 3.28.$$

[1] In section 9.1.1, we will introduce a differential operator d. This is a generalization of differentiation to also include multivariate functions. With such a general description, can the first derivative of a curve $c(t)$ be described by the expression $d(c)_t(1)$, which means that we at a point $c(t)$ have a vector describing where we will be in 1 time unit if we move without changing direction and speed.

4.1.1 Regular curves - arc length parametrization

A regular curve is a curve, $\alpha(t), t \in I$, where the tangent/velocity vector does not vanish, ie $s(t) = |\alpha'(t)| \neq 0$ for all $t \in I$.

In order to emphasize some important properties, we will look into a special type of curves, namely curves that are arc length parameterized.

Given an arc length parameterized curve $\zeta(t)$, it follows that,

- $|\zeta'| = 1$, speed is 1 over the entire curve.
- $\langle \zeta'', \zeta' \rangle = 0$, the second derivative is always orthogonal to the first derivative, because it is only the direction of the first derivative that change, not the length (speed).
- $\kappa = |\zeta''|$, We denote the curvature for κ. The curvature of a curve is defined to be the length of the second derivatives of an arc length parameterized curve.

For arc length parameterized curves embedded in $\mathbb{R}^3$ we also have the following properties,

- Frenet frame (also called TNB frame) is a frame, ie three unit vectors, T, N, B, orthogonal to each other in a right hand system. $T = \zeta'$, $N = \frac{\zeta''}{\kappa}$ and $B = T \wedge N$ (the vector product). The existence of a TNB-frame at a point requires that $\kappa \neq 0$.
- τ is the torsion of a curve. It measures the speed of rotation of the binormal vector N at the given point, and is given by $\tau = -\langle N, B' \rangle$.

4.1.2 Reparameterization

Reparameterization has no effect on the shape of a curve (the trace). It changes speed and parameter interval (domain) of a curve, but it does not affect any of the properties of a curve that we call the intrinsic properties (shape, curve length, curvature, torsion, etc.).

Given a curve $c(t)$, $t \in I \subset \mathbb{R}$. A curve

$$\rho(t) = c(\omega(t)) = c \circ \omega(t)$$

is a reparameterization of $c(t)$ if

- $\omega(t)$ is differentiable for $t \in I$,
- and there exist an inverse ω^{-1} that also is differentiable for $t \in I$.

It follows that $\omega(t)$ must be a strongly monotone function.

The 1^{st}-derivative of the curve $\rho(t)$ is the tangent/velocity vector to the curve, and the length of this vector is certainly affected by the reparameterization. The first derivative is

$$\rho'(t) = \omega'(t) c' \circ \omega(t).$$

It follows that the speed is

$$s(t) = |\omega'(t) c' \circ \omega(t)| = |\omega'(t)| \, |c' \circ \omega(t)|. \tag{4.6}$$

4.1.3 Curvature

We denote the 2^{nd}-derivative of a curve $c(t)$ for $c''(t)$. The 2^{nd}-derivative describe the linear change of the 1^{st}-derivative. The 2^{nd}-derivative can be decomposed into changing the speed and changing the direction.

It is obvious that a change of direction depends on the speed. Everyone who has driven a car has experienced it. Curvature is an intrinsic property that is independent of parametrization and speed. To find the curvature, we reparameterize a curve so that the velocity is always 1, ie arc length parametrization. Given a curve

$$\zeta(t) = c \circ \omega(t).$$

It follows from (4.6) that $\zeta(t)$ is arc length parameterized if $\omega'(t) = \frac{1}{|c'|}$.

To simplify we skip the parameter notation in the expression in the following. Using the kernel rule and the product rule for derivatives, we get

$$\zeta' = \omega' c'$$

and

$$\zeta'' = \omega'' c' + (\omega')^2 c''.$$

To calculate the curvature and to avoid having to calculate ω'' we just use the vector product to find $|\zeta''|$. This can be done because the vector product of two parallel vectors is zero. Further, we know that the curvature, κ, is equal the length of the second derivative, $|\zeta''|$, and that ζ'' is normal to the first derivative ζ' and that $|\zeta'| = 1$. This knowledge will be used in the calculation. To simplify, we first look at curves in $\mathbb{R}^3$. We thus get,

$$\begin{aligned} \kappa = |\zeta''| &= |\zeta' \wedge \zeta''| \\ &= |\omega''||\zeta' \wedge c'| + |\omega'|^2 |\zeta' \wedge c''| \\ &= 0 + |\omega'|^3 |c' \wedge c''|, \end{aligned} \tag{4.7}$$

which gives

Curvature for curves in $\mathbb{R}^3$ and $\mathbb{R}^2$

$$\kappa = \frac{|c' \wedge c''|}{|c'|^3}, \quad c \hookrightarrow \mathbb{R}^3.$$

where $\wedge$ denotes the 3D vector (cross) product.
In R^2 we can use the same formula but here we use the the wedge product giving a scalar, $a \wedge b = a_x b_y - a_y b_x$. This also open for signed curvatures,

$$\kappa = \frac{c' \wedge c''}{|c'|^3}, \quad c \hookrightarrow \mathbb{R}^2.$$

which give a positive curvature on left hand side and negative curvature on right hand side of the curve.
Related to the curvature is the radius of curvature: $r = \frac{1}{\kappa}$.

4.2 Function spaces and Basis functions

The curve expression (4.4) can be rewritten as follows

$$\alpha(t) = \begin{pmatrix} 6t - 9t^2 + 6t^3 \\ -3t + 9t^2 - 6t^3 \end{pmatrix} = \begin{pmatrix} 6 \\ -3 \end{pmatrix} t + \begin{pmatrix} -9 \\ 9 \end{pmatrix} t^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix} t^3, \quad 0 \le t \le 1. \tag{4.8}$$

This expression is on the general form

$$\alpha(t) = a_0\, 1 + a_1\, t + a_2\, t^2 + a_3\, t^3, \quad 0 \le t \le 1. \tag{4.9}$$

where $a_0 = (0,0)$, $a_1 = (6,-3)$, $a_2 = (-9,9)$ and $a_3 = (6,-6)$ are coefficient vectors, and the basis functions are on the monomial form, ie the power basis $\{t^i\}_{i=0}^3$.

Parametric Polynomial Curves

In general, a parametric curve of degree d polynomials using the power basis will be on the following form

$$\alpha(t) = \sum_{i=0}^{d} a_i\, t^i, \quad t \in I \subset \mathbb{R} \tag{4.10}$$

where $a_i,\ i = 0,1,...,d$ are vectors in the space where the curve is embedded.

One can easily see that the power basis in (4.10) can act in the same way as basis vectors in a $(d+1)$-dimensional real vector space. They are clearly linear independent because you can not get one of the basis functions from a linear combination of the others. We can therefor treat the set of all polynomial based functions of degree most d as a vector space where $1, t, t^2, ..., t^d$ are basis vectors as in (4.10).

This leads to a general expression of curve formulas,

Parametric Curves in general

$$\alpha(t) = \sum_{i=1}^{k} c_i\, b_i(t), \quad t \in I \subset \mathbb{R} \tag{4.11}$$

where $c_i,\ i = 1,...,k$ are vectors/points in the space where the curve is embedded, and $b_i(t),\ i = 1,...,k$ is a set of linearly independent functions spanning a given finite dimensional function space.

Note that these are finite dimensional vector spaces, and the curves therefore have big restrictions in the shaping possibilities. A second degree curve has no inflection points, a third degree curve has only one inflection points and so on. Imagine that you are drawing a curve freehand. To find an expression for this curve will require an infinite degree of the polynomial and, thus, an infinite dimensional vector space. Infinite dimensional vector spaces can be useful theoretically, but they are certainly not possible to implement for curves and surfaces in design applications on a computer.

Function space

is a very useful concept in constructing and analyzing curves for geometric modeling and design (remember that a function space is a vector space).

Definition 4.3. *A function space is a set of functions of a given kind from a set X to a set Y. It is a topological vector space whose "vectors" are functions.*

A typical example is the set of all polynomial of degree most 3, mapping the interval $I = [0,1]$ to $\mathbb{R}^n$, $n > 0$ but finite. We denote this function space for $\mathscr{P}_3(I)$

In the following sections we will show that in the polynomial case there are several set of basis functions for a given function space and thus to use for the same curves, all with valuable properties.

Infinite dimensional function spaces are useful theoretically.

Definition 4.4. *An infinite dimensional function space is a set $\mathscr{F}(I)$ that is the collection of all real-valued continuous functions defined on some interval I, and $\mathscr{F}^{(n)}(I)$ is the collection of all functions $\in \mathscr{F}(I)$ with n continuous derivatives.*

Example of an infinite dimensional function space is the Hilbert space L^2, the set of all functions $f : \mathbb{R} \to \mathbb{R}$ such that the integral of $|f(x)|^2$ over the whole real line is finite. In this case, the inner product is

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(x)g(x)dx$$

4.3 Hermite Curves

We use the polynomial example first formulated in (4.4) and reformulated in (4.8).

$$\alpha(t) = \begin{pmatrix} 6 \\ -3 \end{pmatrix} t + \begin{pmatrix} -9 \\ 9 \end{pmatrix} t^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix} t^3 \quad 0 \le t \le 1.$$

We compute the first derivative

$$\alpha'(t) = \begin{pmatrix} 6 \\ -3 \end{pmatrix} + 2\begin{pmatrix} -9 \\ 9 \end{pmatrix} t + 3\begin{pmatrix} 6 \\ -6 \end{pmatrix} t^2, \quad 0 \le t \le 1.$$

We then compute the expressions at the start parameter value $t = 0$ and the end parameter value $t = 1$,

$$\alpha(0) = \begin{pmatrix} 6 \\ -3 \end{pmatrix} 0 + \begin{pmatrix} -9 \\ 9 \end{pmatrix} 0^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix} 0^3 = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

$$\alpha(1) = \begin{pmatrix} 6 \\ -3 \end{pmatrix} 1 + \begin{pmatrix} -9 \\ 9 \end{pmatrix} 1^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix} 1^3 = \begin{pmatrix} 3 \\ 0 \end{pmatrix},$$

$$\alpha'(0) = \begin{pmatrix} 6 \\ -3 \end{pmatrix} + 2\begin{pmatrix} -9 \\ 9 \end{pmatrix} 0 + 3\begin{pmatrix} 6 \\ -6 \end{pmatrix} 0^2 = \begin{pmatrix} 6 \\ -3 \end{pmatrix},$$

$$\alpha'(1) = \begin{pmatrix} 6 \\ -3 \end{pmatrix} + 2\begin{pmatrix} -9 \\ 9 \end{pmatrix} 1 + 3\begin{pmatrix} 6 \\ -6 \end{pmatrix} 1^2 = \begin{pmatrix} 6 \\ -3 \end{pmatrix}.$$

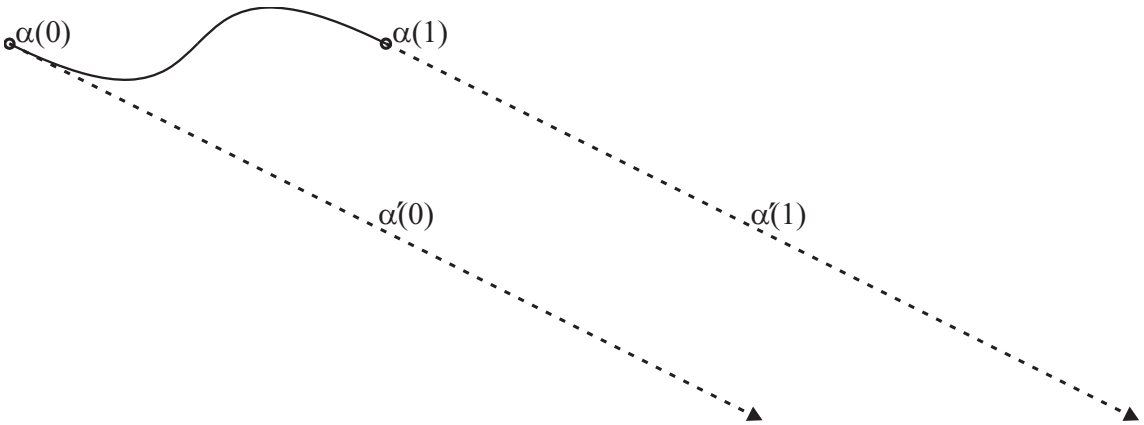


Figure 4.5: *A plot of the curve from expression (4.4) and the four coefficients, the start point and respective first derivative and the end point and the respective first derivative.*

In Figure 4.5 there is a new plot of $\alpha(t)$ plotted in Figure 4.3. Now the start and the end point of the curve, $\alpha(0)$ and $\alpha(1)$ are marked with circles, and the respective tangent/velocity vectors at the start and end of the curve, $\alpha'(0)$ and $\alpha'(1)$ are plotted as dashed arrows.

The next step is to look at the general expression for polynomial based curves of degree 3, ie equation (4.9) in section 4.2, and its 1^{st}-derivative,

$$\begin{aligned} \alpha(t) &= a_0 + a_1\, t + a_2\, t^2 + a_3\, t^3, \\ \alpha'(t) &= a_1 + 2a_2\, t + 3a_3\, t^2. \end{aligned} \tag{4.12}$$

From the expression (4.12) we compute the position of the starting point, $\alpha(0)$, the position of the endpoint, $\alpha(1)$, the 1^{st}-derivative (the tangent/velocity vector) at the start point, $\alpha'(0)$, and the 1^{st}-derivative at the end point, $\alpha'(1)$;

$$\begin{aligned} \alpha(0) &= a_0, \\ \alpha(1) &= a_0 + a_1 + a_2 + a_3, \\ \alpha'(0) &= a_1, \\ \alpha'(1) &= a_1 + 2a_2 + 3a_3. \end{aligned}$$

We reorganize in matrix notation and get

$$\begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}. \tag{4.13}$$

We turn the equation in (4.13) and invert the matrix

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}. \tag{4.14}$$

We rewrite the curve $\alpha(t)$ from (4.12) to vector notation using inner product, ie

$$\alpha(t) = \begin{pmatrix} 1, & t, & t^2, & t^3 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}.$$

The next step is to replace the vector of coefficients ($a_i,\ i = 0,1,2,3$) on right hand side in the equation above with the expression on right hand side in (4.14), ie

$$\alpha(t) = \begin{pmatrix} 1, & t, & t^2, & t^3 \end{pmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}. \tag{4.15}$$

Finally we compute the vector with the power basis on left hand side with the matrix,

$$\alpha(t) = \begin{pmatrix} 1-3t^2+2t^3, & 3t^2-2t^3, & t-2t^2+t^3, & -t^2+t^3 \end{pmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}.$$

In the vector on left hand side we now have a new set of basis functions for a 3^{rd}-degree polynomial based curve. This gives:

Hermite Curves, 3^{rd}-degree

are polynomial based curves on the following form:

$$\alpha(t) = \alpha(0)\, H_1(t) + \alpha(1)\, H_2(t) + \alpha'(0)\, H_3(t) + \alpha'(1)\, H_4(t), \quad t \in [0,1], \tag{4.16}$$

where the coefficients are the position at start $\alpha(0)$ and end $\alpha(1)$, and the first derivative at start $\alpha'(0)$ and end $\alpha'(1)$. The four 3^{rd}-degree Hermite basis functions are:

$$\begin{aligned} H_1(t) &= 1-3t^2+2t^3 &&= (2t+1)(1-t)^2, \\ H_2(t) &= 3t^2-2t^3 &&= (3-2t)t^2, \\ H_3(t) &= t-2t^2+t^3 &&= t(t-1)^2, \\ H_4(t) &= -t^2+t^3 &&= t^2(t-1). \end{aligned} \tag{4.17}$$

The fact that the matrix in (4.13) is inverted in (4.14) and therefore obvious is invertible proves that the set of Hermite basis functions are linearly independent and thus is a basis set for 3^{rd}-degree polynomial curves. The four basis functions are plotted in Figure 4.6.

The special about Hermite curves is that the four coefficients are the starting point and it's related velocity vector (the first derivative) and the end point and it's related velocity vector. This is very practical for constructing curves. This also explains the name[2];

[2]Hermite Curves and Hermite interpolation is named after Charles Hermite (1822 – 1901), a French mathematician who did research on number theory, quadratic forms, invariant theory, orthogonal polynomials, Abelian and elliptic functions, and algebra.

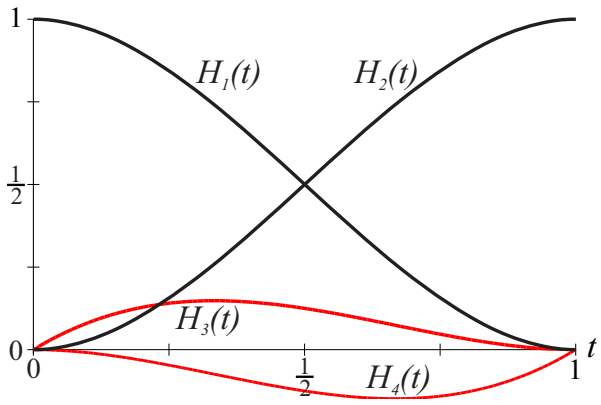


Figure 4.6: *A plot of a the four Hermite basis function in (4.17). The functions connected to vectors are red. One can clearly see the symmetry in the pairs of functions.*

Hermite interpolation is interpolating a point and the derivative vectors at the same point. This is also called osculatory interpolation (osculatory means kissing on Latin).

The property that is most important for the set of Hermite basis function of degree three, $\mathbf{H}_3(t) = [H_1(t),\ H_2(t),\ H_3(t),\ H_4(t)]$, is,

$$\begin{aligned} \mathbf{H}_3(0) &= [1,\ 0,\ 0,\ 0], \\ \mathbf{H}_3(1) &= [0,\ 1,\ 0,\ 0], \\ \mathbf{H}_3'(0) &= [0,\ 0,\ 1,\ 0], \\ \mathbf{H}_3'(1) &= [0,\ 0,\ 0,\ 1]. \end{aligned} \tag{4.18}$$

Notice that $H_1(t) + H_2(t) = 1$. This is called fulfilling the property of partition of unity. These two basis functions, $H_1(t)$ and $H_2(t)$, are blending the points. The other two basis functions $H_3(t)$ and $H_4(t)$ are blending vectors and does not sum to 1. The importance of this was explained in the section of Affine spaces, see page 14. The use of a power basis set is sometimes called the algebraic form while using the Hermite basis set is called the geometric form.

Hermite interpolation can also be done for higher degree polynomials. Thus, we will look at general expression for polynomial-based curves of degree 5 and it´s first and second derivatives,

$$\begin{aligned} \alpha(t) &= a_0 + a_1\, t + a_2\, t^2 + a_3\, t^3 + a_4\, t^4 + a_5\, t^5 \\ \alpha'(t) &= a_1 + 2a_2\, t + 3a_3\, t^2 + 4a_4\, t^3 + 5a_5\, t^4. \\ \alpha''(t) &= 2a_2 + 6a_3\, t + 7a_4\, t^2 + 20a_5\, t^3. \end{aligned} \tag{4.19}$$

We start by computing from (4.19) the position in the starting point $\alpha(0)$, the position of the endpoint $\alpha(1)$, the 1^{st}-derivative (the tangent vector) in the starting point $\alpha'(0)$ and the first derivative of the endpoint $\alpha'(1)$, the 2^{nd}-derivative in the starting point $\alpha''(0)$ and

the 2^{nd}-derivative of the endpoint $\alpha''(1)$;

$$\begin{aligned}
\alpha(0) &= a_0 \\
\alpha(1) &= a_0 + a_1 + a_2 + a_3 + a_4 + a_5 \\
\alpha'(0) &= a_1 \\
\alpha'(1) &= a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 \\
\alpha''(0) &= 2a_2 \\
\alpha''(1) &= 2a_2 + 6a_3 + 7a_4 + 20a_5.
\end{aligned}$$

We reorganize the equations in matrix notation and get

$$\begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \\ \alpha''(0) \\ \alpha''(1) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6 & 12 & 20 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix}. \tag{4.20}$$

We invert the matrix and turn the total equation in (4.20) to

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ -10 & 10 & -6 & -4 & \frac{-3}{2} & \frac{1}{2} \\ 15 & -15 & 8 & 7 & \frac{3}{2} & -1 \\ -6 & 6 & -3 & -3 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \\ \alpha''(0) \\ \alpha''(1) \end{pmatrix}. \tag{4.21}$$

We write the curve $\alpha(t)$ on vector notation using inner product, ie

$$\alpha(t) = \left(1,\, t,\, t^2,\, t^3,\, t^4,\, t^5\right) \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix}.$$

The next step is to replace the vector of coefficients $(a_i,\ i = 0,1,2,3,4,5)$ on right hand side in the equation above with the expression on right hand side in (4.21):

$$\alpha(t) = \left(1,\, t,\, t^2,\, t^3,\, t^4,\, t^5\right) \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ -10 & 10 & -6 & -4 & \frac{-3}{2} & \frac{1}{2} \\ 15 & -15 & 8 & 7 & \frac{3}{2} & -1 \\ -6 & 6 & -3 & -3 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \\ \alpha''(0) \\ \alpha''(1) \end{pmatrix}. \tag{4.22}$$

Finally we compute the vector with the power basis on left hand side with the matrix, and we get the new set of basis functions for a 5^{th}-degree polynomial based curve. The six

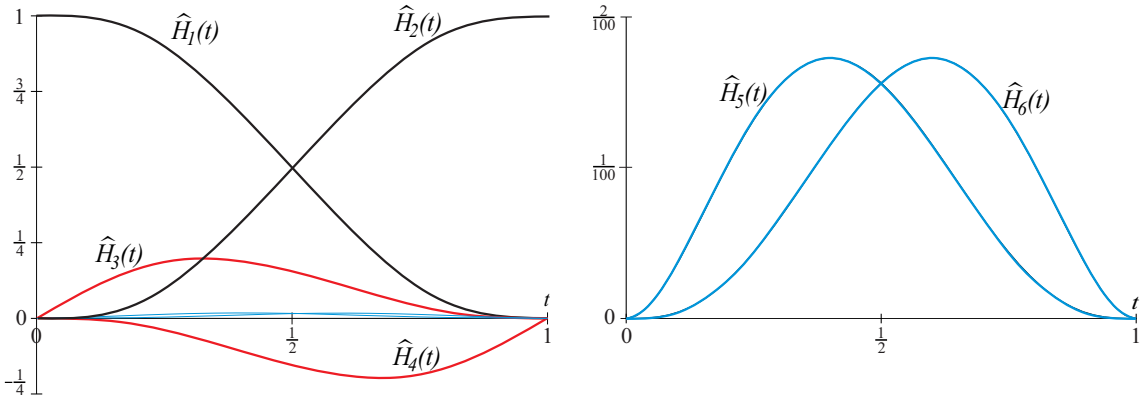


Figure 4.7: *A plot of a the six 5^{th}-degree Hermite basis functions defined in (4.24). The functions connected to the second derivatives are also plotted separately on right hand side, since the values are so small that we can nearly see them together with the other basis functions on the plot on left hand side.*

basis functions are plotted in figure 4.7. Thus we get:

Hermite Curves, 5^{th}-degree

are polynomial based curves on the following form:

$$\begin{aligned}\alpha(t) = \alpha(0)\,\widehat{H}_1(t) + \alpha(1)\,\widehat{H}_2(t) + \alpha'(0)\,\widehat{H}_3(t) + \\ \alpha'(1)\,\widehat{H}_4(t) + \alpha''(0)\,\widehat{H}_5(t) + \alpha''(1)\,\widehat{H}_6(t), \quad t \in [0,1],\end{aligned} \tag{4.23}$$

where the coefficients are the position at start $\alpha(0)$ and end $\alpha(1)$, the first derivative at start $\alpha'(0)$ and end $\alpha'(1)$, and the second derivatives at start $\alpha''(0)$ and end $\alpha''(1)$. The six 5^{th}-degree Hermite basis functions are

$$\begin{aligned}\widehat{H}_1(t) &= 1 - 10t^3 + 15t^4 - 6t^5 &&= \left(6t^2 + 3t + 1\right)(1-t)^3 \\ \widehat{H}_2(t) &= 10t^3 - 15t^4 + 6t^5 &&= \left(6t^2 - 15t + 10\right)t^3 \\ \widehat{H}_3(t) &= t - 6t^3 + 8t^4 - 3t^5 &&= (3t+1)(1-t)^3 t \\ \widehat{H}_4(t) &= -4t^3 + 7t^4 - 3t^5 &&= (3t-4)(1-t)t^3 \\ \widehat{H}_5(t) &= \tfrac{1}{2}t^2 - \tfrac{3}{2}t^3 + \tfrac{3}{2}t^4 - \tfrac{1}{2}t^5 &&= \tfrac{1}{2}(1-t)^3 t^2 \\ \widehat{H}_6(t) &= \tfrac{1}{2}t^3 - t^4 + \tfrac{1}{2}t^5 &&= \tfrac{1}{2}(1-t)^2 t^3.\end{aligned} \tag{4.24}$$

4.4 Bézier Curves

In Figure 4.5 was the curve from (4.4) plotted together with the four Hermite coefficients, two points and two vectors. The next thing we can do is to replace the two vectors with two points so that we only have points as coefficients. Recall that the dimension of the polynomial function space is 4, and that we will thus have 4 points as coefficients. This means that the points are connected by 3 lines (see Figure 4.8). The length of the 1^{st}-derivative vector is the speed, and since the parameter range is 1, the speed and the curve length are on average equal. We therefor use $\frac{1}{3}$ of the length of the vectors to find the

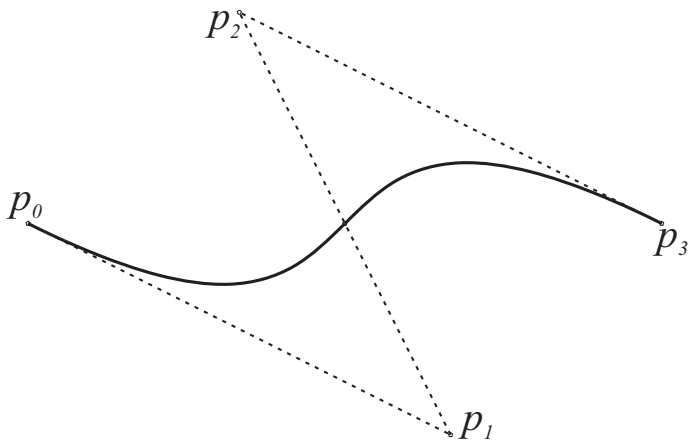


Figure 4.8: *A plot of a curve first formulated in expression (4.4). The Bézier control polygon and the four control points (coefficients) are also plotted.*

points. We name the points p_i and get

$$\begin{aligned} p_0 &= \alpha(0), \\ p_1 &= \alpha(0) + \tfrac{1}{3}\alpha'(0), \\ p_2 &= \alpha(1) - \tfrac{1}{3}\alpha'(1), \\ p_3 &= \alpha(1). \end{aligned}$$

Reorganizing this in matrix notation,

$$\begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & \frac{1}{3} & 0 \\ 0 & 1 & 0 & -\frac{1}{3} \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}, \tag{4.25}$$

inverting,

$$\begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}. \tag{4.26}$$

From expression (4.15) we have

$$\alpha(t) = \left(1,\, t,\, t^2,\, t^3\right) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}.$$

Replacing the Hermite coefficient with (4.26) we get

$$\alpha(t) = \left(1,\, t,\, t^2,\, t^3\right) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}.$$

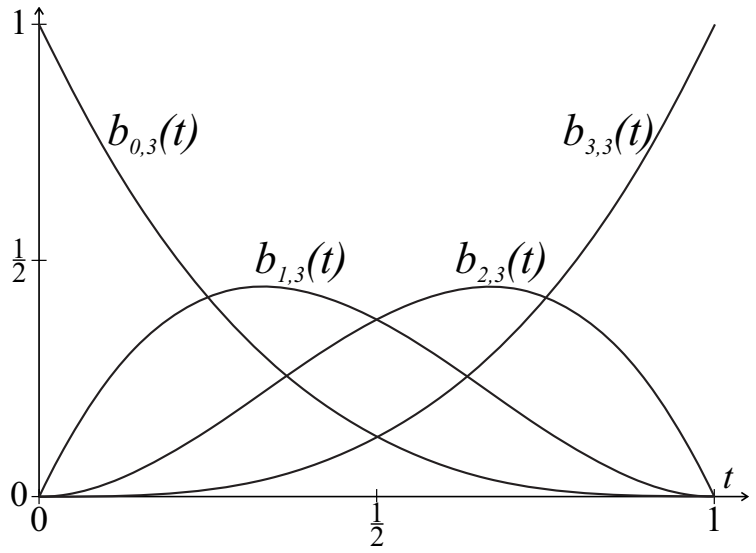


Figure 4.9: *A plot of the four Bézier basis functions of degree 3 specified in the formula (4.28) used to generate a 3^{rd}-degree Bézier curves defined in (4.27).*

Finally, we computing the vector on left hand side with the two matrices and we get,

$$\alpha(t) = \begin{pmatrix} (1-t)^3, & 3t(1-t)^2, & 3t^2(1-t), & t^3 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}.$$

We now have another new set of basis functions for the curve where we previously have the power basis in (4.9) and afterwards the Hermite basis in (4.16). This new sets provides:

Bézier Curves of degree 3

is uniquely defined by a set of four ordered points describing a control polygon (plotted as dotted lines in the example in figure 4.8),

$$\alpha(t) = p_0\, b_{0,3}(t) + p_1\, b_{1,3}(t) + p_2\, b_{2,3}(t) + p_3\, b_{3,3}(t) = \sum_{i=0}^{3} p_i\, b_{i,3}(t), \quad t \in [0,1], \tag{4.27}$$

where the 4 Bézier basis functions are the set of 3^{rd}-degree Bernstein polynomials

$$\begin{aligned} b_{0,3}(t) &= (1-t)^3 \\ b_{1,3}(t) &= 3t(1-t)^2 \\ b_{2,3}(t) &= 3t^2(1-t) \\ b_{3,3}(t) &= t^3. \end{aligned} \tag{4.28}$$

The fact that the matrix in (4.25) is inverted in (4.26) proves that the set of 3^{rd}-degree Bézier basis functions (the Bernstein polynomials) is linearly independent and thus is a basis for 3^{rd} polynomial degree curves. The four basis functions for third degree Bézier curves are plotted in Figure 4.9.

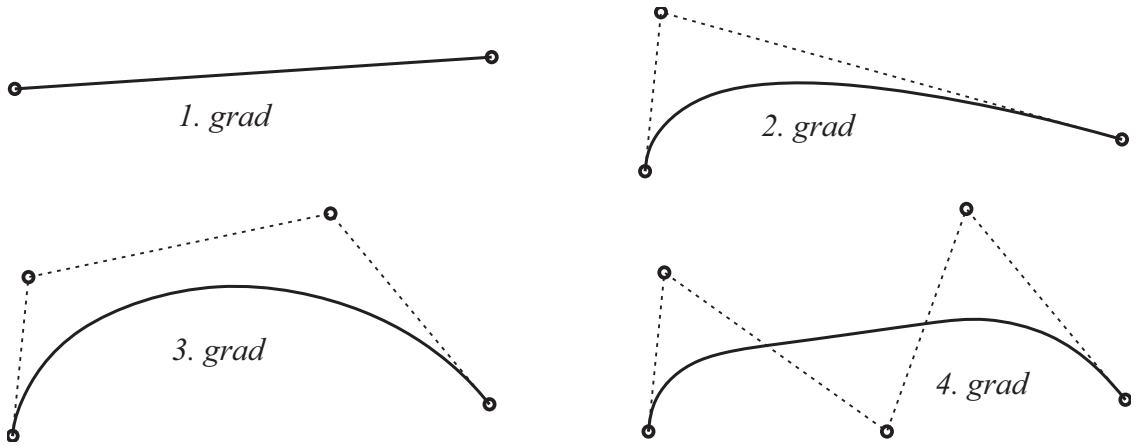


Figure 4.10: *A plot of four Bézier curves of degree 1, 2, 3 and 4. Also the control points (marked with circles) and the control polygons are plotted. The first degree curve and it's control polygon coincide.*

The set of basis functions for 3^{rd}-degree Bézier curves is called the Bernstein polynomials of degree 3. There is a set of Bernstein polynomials for each degree. They will be described further in the following subsection, but every set are fulfilling the requirement to be a set of basis functions for their respective degrees of polynomial functions on the domain $[0,1]$. Bézier curves are using a set of Bernstein polynomials as their blending functions. It follows that there are Bézier curves of all polynomial degrees d and that they are uniquely defined by an ordered set of $d+1$ points.

Bézier Curves of degree d

is uniquely defined by $d+1$ control points describing a control polygon. The general expression for a Bézier curve of degree d is,

$$\alpha(t) = \sum_{i=0}^{d} p_i \, b_{i,d}(t), \quad t \in [0,1], \tag{4.29}$$

where the coefficients $p_i,\ i = 0,1,...,d$ are the $d+1$ points defining the control polygon of the Bézier curve. The set of the $d+1$ basis functions $b_{i,d}(t),\ i = 0,..,d$ are the Bernstein polynomials;

$$b_{i,d}(t) = \begin{pmatrix} d \\ i \end{pmatrix} t^i (1-t)^{d-i},\ i = 0,1,...,d.$$

The Bézier curve start at the first control point and ends at the last control point. The tangent of the curve are equal the direction of the control polygon at the start and end point. The control polygon is modeling the curve in such a way that the variation of the curve are smaller than the variation of the control polygon (called the variation diminishing property). Examples can be seen in the figures 4.8 and 4.10.

Bézier curves[3] are among the most popular curve formats. They are defined only by a set of points called a control polygon. As we can see in figure 4.10 the control polygon is modeling the curve. In the figure, a first degree, a second degree, a third degree and a fourth degree Bézier curve are plotted together with their respective control polygons. The first degree curve and it's control polygon is actually coinciding. As we will se later Bézier curves are actually a special case of B-splines. Several algorithms for computing Bézier curves will therefor be shown when B-spline is treated.

4.4.1 Bernstein polynomial

An algorithm for computing Bernstein polynomials can be made recursively. And we will also see that they, on the domain $[0,1]$, can form a basis for curves and that each set always sums up to 1.

The general expression for a set of Bernstein polynomial of degree d, restricted to the domain $[0,1]$, are

$$b_{i,d}(t) = \binom{d}{i} t^i (1-t)^{d-i}, \quad i = 0,1,2,\ldots,d.$$

The name Bernstein polynomials were first used for the function itself were these polynomials were the basis functions.[4] Today, however, the set of basis functions are usually called the Bernstein polynomials. Below follows the 6 first set of Bernstein basis functions ($d = 0,1,2,3,4,5$), one set on each row,

$$\begin{array}{ccccccccccc} & & & & & 1 & & & & & \\ & & & & 1-t & & t & & & & \\ & & & (1-t)^2 & & 2t(1-t) & & t^2 & & & \\ & & (1-t)^3 & & 3t(1-t)^2 & & 3t^2(1-t) & & t^3 & & \\ & (1-t)^4 & & 4t(1-t)^3 & & 6t^2(1-t)^2 & & 4t^3(1-t) & & t^4 & \\ (1-t)^5 & & 5t(1-t)^4 & & 10t^2(1-t)^3 & & 10t^3(1-t)^2 & & 5t^4(1-t) & & t^5 \end{array} \tag{4.30}$$

The notation is as following

$$\begin{array}{ccccccc} & & & b_{0,0}(t) & & & \\ & & b_{0,1}(t) & & b_{1,1}(t) & & \\ & b_{0,2}(t) & & b_{1,2}(t) & & b_{2,2}(t) & \\ b_{0,3}(t) & & b_{1,3}(t) & & b_{2,3}(t) & & b_{3,3}(t) \end{array}$$

——————————- and so on ——————————

[3] Bézier Curves are named after Pierre Bézier (1910 – 1999). He worked for Renault from 1933 – 1975, where he developed his UNISURF CAD-CAM system. He published and patented the results in 1962 although Paul de Castejau (1930 –) already in 1959 had made an algorithm for computing Bézier Curves but only published this in an internal Citroén report.

[4] The polynomial occurred as result of the work of Sergei Natanovich Bernstein, an Ukrainian mathematician (1880-1968). Bernstein introduced the polynomials in 1911 (published in [10]), using them for constructive proof of the Stone-Weierstrass approximation theorem. For a closer study of these polynomials see page 183–186 in [91] or page 108–126 in [35].

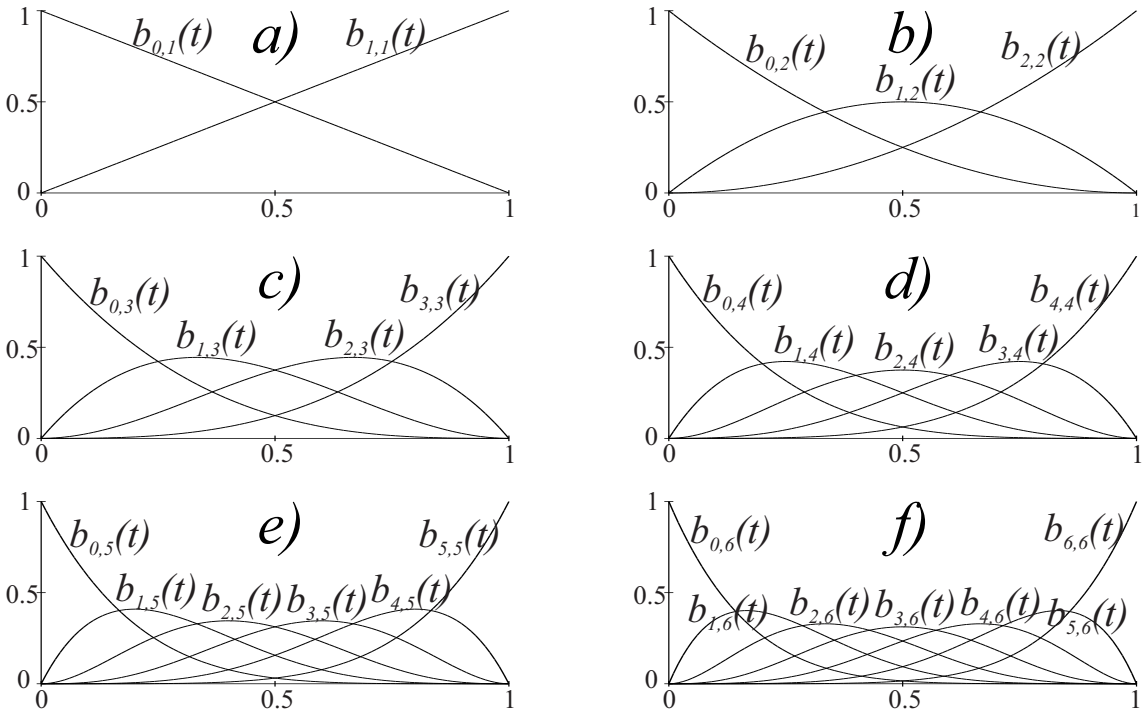


Figure 4.11: *Six sets of Bernstein basis polynomials.* **a)** *is the set of degree, forming first degree Bézier curves.* **b)** *is the set of degree two,* **c)** *is the set of degree three,* **d)** *is the set of degree four,* **e)** *is the set of degree five,* **f)** *is the set of degree six. All sets are basis functions for Bézier curves of the respective degree.*

In figure 4.11 are the six sets of Bernstein polynomials of degree 1 to 6 plotted. These are the functions on each row of expression 4.30.

The algorithm for computing the sets of $d+1$ Bernstein polynomials of degree d is recursive, and using the set of d Bernstein polynomials of degree $d-1$.

1. First we compute the one on the left hand side, $b_{0,d}(t) = (1-t)\, b_{0,d-1}(t)$,
2. if $d > 1$ (we have more than two polynomials) we compute all internal polynomials in the row: for $i = 1, ..., d-1$, $b_{i,d}(t) = t\, b_{i-1,d-1}(t) + (1-t)\, b_{i,d-1}(t)$.
3. Finally we compute the one on the right hand side, $b_{d,d}(t) = t\, b_{d-1,d-1}(t)$.

We start with the function $b_{0,0}(t) = 1$ (a polynomial of degree 0). We then compute the next set, the degree $d = 1$ based on $b_{0,0}(t)$ and then the next, and so on. The algorithm is also called a pyramid algorithm (see [77]). Another view of the algorithm will be shown in the next section.

Algorithm 1. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes a vector $\mathbf{b}_d(t) \in \mathbb{R}^{d+1}$*, containing the values of the* $d+1$ *Bernstein polynomials* $\{b_{d,i}(t)\}_{i=0}^{d}$*. The input variables are: the degree* d *of the Bernstein polynomials and the parameter value* $t \in [0,1]$.

vector⟨double⟩ **bernstein (** int *d,* double *t* **)**

```
vector⟨double⟩ b(d+1);          // The return vector, dimension d+1.
b_0 = 1;                        // A general Cox/deBoor like algorithm for
for ( int i=1; i ≤ d; i++ )     // - Bernstein polynomials, computing the
    b_i = t b_{i-1};            // - value for the set of polynomial of degree d.
    for ( int j=i-1; j > 0; j-- )
        b_j = t b_{j-1} + (1-t) b_j;
    b_0 = (1-t) b_0;
return b;
```

Lemma 4.1. *The Bernstein polynomials of degree d form a basis-set for polynomial functions of degree d and lower, on the domain* $[0,1]$.

Proof. This is the same as answering the following question; are the functions in the set linearly independent of each other or can we get one of them as a linear combination of the other?

The answer is; they are linearly independent. This can be shown in the following way, the argument is clearly illustrated in the pyramid (4.30):

- The function on the right is the only one with only one term (of degree d). It is obvious linearly independent from the other.
- the next function from right hand side is the only one with only two terms (degree d and $d-1$. It is obvious independent from the other functions.
- The same argument can be used for all other functions because the number of terms increase in the same way as the second one when we go from right to left.

□

Lemma 4.2. *The set of Bernstein polynomials of degree d sums up to 1 for all d. We call this property; to form a partition of unity on the domain* $[0,1]$*, and it is expressed by*

$$\sum_{i=0}^{d} b_{i,d}(t) = 1, \quad for \quad d = 1,2,3,...$$

Proof. We start with the first function (on the first row) $b_{0,0}(t) = 1$ that obvious sum up to 1. The next row, $b_{0,1}(t) + b_{1,1}(t) = (1-t) + t = 1$ i.e. also sum up to 1. If we now look at the sum $s_d(t)$ of a set (a row) of degree d defined by the recursive algorithm on the previous page, we get

$$s_d(t) = (1-t)\ b_{0,d-1}(t) + \sum_{i=1}^{d-1} (t\ b_{i-1,d-1}(t) + (1-t)\ b_{i,d-1}(t)) + t\ b_{d-1,d-1}(t).$$

If we reorganize this we get,

$$s_d(t) = \sum_{i=0}^{d-1} b_{i,d-1}(t) = s_{d-1}(t)$$

which shows that the sum of a row is equal the sum of the row above. By induction this shows that a set of Bernstein basis functions for all degrees sum up to 1. □

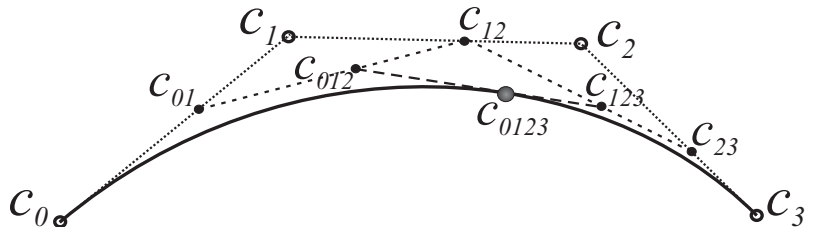


Figure 4.12: *A Bézier curve (bold) and the points (bullets) and lines (dotted) illustrating the de Casteljau's algorithm for a third degree Bézier curve at $t = 0.6$.*

4.4.2 Factorization and de Casteljau's Corner cutting algorithm

To further investigate the algorithm for Bernstein polynomials and Bézier curves, we start with linear interpolation between two points c_0 and c_1 (in the plane or the 3D space):

$$c(t) = (1-t)\, c_0 + t\, c_1.$$

For $t \in [0,1]$ we get the line segment between c_0 and c_1.

Given a sequence of points c_0, c_1, c_2, c_3. If, for a given $t \in [0,1]$ interpolate between pairs of two points, c_0 and c_1, c_1 and c_2, c_2 and c_3, we get three new points, which we call c_{01}, c_{12} and c_{23}, respectively. In matrix vector notation we have done the following,

$$\begin{pmatrix} c_{01} \\ c_{12} \\ c_{23} \end{pmatrix} = \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}.$$

If we repeat the process with these new control points we get

$$\begin{pmatrix} c_{012} \\ c_{123} \end{pmatrix} = \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_{01} \\ c_{12} \\ c_{23} \end{pmatrix},$$

and finally

$$c_{0123} = \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} c_{012} \\ c_{123} \end{pmatrix}.$$

This process is called de Casteljau's[5] corner cutting algorithm for evaluating Bézier curves (see [39] and [41]), and it is illustrated in Figure 4.12.

If we connect these three multiplications, we get the following equation for a third degree Bézier curve,

$$c(t) = \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}. \quad (4.31)$$

[5] Paul de Casteljau (1930 -) is a French physicist and mathematician. In 1959, while working at Citroën, he developed an algorithm for evaluating calculations on what later was named Bézier curves because he was not permitted to publish his early work. We also call the multilinear polynomials "blossoming",

If the matrices are multiplied from the right, we follow the pyramid in (4.30) and Algorithm 1. For each matrix we multiply in, we get a new row in the pyramid in (4.30). This means that when we have multiplied together all the matrices we have the four Bernstein polynomials of degree 3, ie

$$\left((1-t)^3, \quad 3(1-t)^2 t, \quad 3(1-t)t^2, \quad t^3\right).$$

We call these matrices for the Bernstein factor matrices and denote them $T_d(t)$. For example if $d = 2$ we get

$$T_2(t) = \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix}.$$

We now have the following general definition of the Bernstein factor matrices.

Definition 4.5. *$T_d(t)$ is a matrix with dimension $d \times (d+1)$ where all lines numbered j*

a) only have element number j and $j+1$ different from zero.
b) where (in the Bézier case) element number j is $1-t$ and element number $j+1$ is t.

Note that in the ordinary Bézier case we use t. We can also scale and translate the domain by using $w_{s,e}(t) = \frac{t-s}{e-s}$ instead of t. Later we will see that in B-splines we use a translation and scaling function $w_{d,i}(t) = \frac{t-t_i}{t_{i+d}-t_i}$, see page 82 and 87.

Because we have chosen this notation for the set of Bernstein factor matrices, we will use the following notation for the vector of Bernstein polynomials of degree d:

$$\mathbf{T}^d(t) = T_1(t)T_2(t)\cdots T_d(t) = (b_0(t), b_1(t), \ldots, b_d(t)).$$

It follows that equation (4.31) can be expressed as

$$c(t) = T_1(t)T_2(t)T_3(t)\ \mathbf{c} = \mathbf{T}^3(t)\ \mathbf{c},$$

where $\mathbf{c} = (c_0, c_1, c_2, c_3)^T$ is a vector of points. In traditional notation this will be

$$c(t) = \sum_{i=0}^{3} b_{i,d}(t)\ c_i.$$

Notice that the derivative of the matrix $T_d(t)$ is a matrix of constants. We, therefor, denote it T'_d. For $d = 1$ we get

$$T'_1 = (-1, \quad 1).$$

We can now look at a matrix version of a Bézier curve as described in expression (4.31). In the following equations we will see a 3^{rd}-degree Bézier curve and its three derivatives in matrix form.

$$\begin{array}{rcll} c(t) = & \mathbf{T}^3(t)\ C & = & T_1(t)T_2(t)T_3(t)\ C, \\ c'(t) = & 3\ \mathbf{T}^2(t)\mathbf{T}'\ C & = & 3\ T_1(t)T_2(t)\ T'_3\ C, \\ c''(t) = & 6\ \mathbf{T}^1(t)\mathbf{T}'^2\ C & = & 6\ T_1(t)\ T'_2T'_3\ C, \\ c'''(t) = & 6\ \mathbf{T}'^3\ C & = & 6\ T'_1T'_2T'_3\ C. \end{array} \tag{4.32}$$

The indices denotes the number of rows in the matrix. The derivative of the matrix $T(t)$, denoted T' is a matrix independent of t. Expanding (4.32), we get,

$$
\begin{aligned}
c(t) &= \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\
c'(t) &= 3 \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}, \\
c''(t) &= 6 \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}, \\
c'''(t) &= 6 \begin{pmatrix} -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}.
\end{aligned}
\tag{4.33}
$$

The equations also depends on the commutativity relations between $T(t)$ matrices and their derivatives. For example is $T_1'T_2(t)T_3(t) = T_1(t)T_2'T_3(t) = T_1(t)T_2(t)T_3'(t)$, and it follows that

$$c'(t) = \left(T_1'T_2(t)T_3(t) + T_1(t)T_2'T_3(t) + T_1(t)T_2(t)T_3'\right) C = 3\, T_1(t)T_2(t)\, T_3'\, C.$$

A proof of the commutativity relations is given in subsection **??**.

Summing up the factorisation, we can clearly see that:

i) If we compute from the right hand side (skipping the zeros), we get the de Casteljau corner cutting algorithm.
ii) If we compute from the left hand side, we get an algorithm of Cox/de Boor type.
iii) If we multiply all matrices without the coefficient vector on the right hand side, we will get the Bernstein polynomials and their derivatives. [6]

4.4.3 The Bernstein/Hermite matrix

We take the formulas in (4.33), multiply the three matrices in each row together and merge the result into a matrix formulation, ie

$$
\begin{pmatrix} c(t) \\ c'(t) \\ c''(t) \\ c'''(t) \end{pmatrix} = \begin{pmatrix} (1-t)^3 & 3t(1-t)^2 & 3t^2(1-t) & t^3 \\ -3(1-t)^2 & 9t^2-12t+3 & -3t(3t-2) & 3t^2 \\ 6-6t & 18t-12 & 6-18t & 6t \\ -6 & 18 & -18 & 6 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}. \tag{4.34}
$$

[6] In section 6.2.3, is the same matrix notation used on B-splines. This easily gives us both the Cox/de Boor algorithm for B-splines, a geometric de Casteljau version for B-splines, and if we multiply the matrices, a fast expanded version of an evaluator. Matrix notation on B-splines is also discussed in [116].

This matrix (4.34) is the Bernstein/Hermite matrix of order 4. It is the set of Bernstein polynomials of degree 3 and all their derivatives. Note that if we turn the expression in 4.34 and invert the matrix we get the Hermite interpolation formula

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 & -t & \frac{1}{2}t^2 & -\frac{1}{6}t^3 \\ 1 & \frac{1}{3}-t & \frac{1}{2}t^2-\frac{1}{3}t & \frac{1}{6}t^2(1-t) \\ 1 & \frac{2}{3}-t & \frac{1}{2}t^2-\frac{2}{3}t+\frac{1}{6} & -\frac{1}{6}t(1-t)^2 \\ 1 & 1-t & \frac{1}{2}t^2-t+\frac{1}{2} & \frac{1}{6}(1-t)^3 \end{pmatrix} \begin{pmatrix} c(t) \\ c'(t) \\ c''(t) \\ c'''(t) \end{pmatrix}. \tag{4.35}$$

Remember that a set of Bernstein polynomials of a given degree sums up to 1, so it follows that the first row of the Bernstein/Hermite matrix (4.34) sums up to 1 and all the other rows sums up to 0. It also follows, as we can see in (4.34) that all elements in the first column of the inverted matrix are 1, cf. Taylor expansion in Section 5.5.

Recall from the sections 2.4, 2.5 and 2.6 that for a point in homogeneous coordinates the last coordinate is equal to 1, and the last coordinate of a vector is equal to 0. The properties of the Bernstein/Hermite matrix reflect this: c_0, c_1, c_2, c_3, $c(t)$ are points, while $c'(t)$, $c''(t)$, $c'''(t)$ are vectors. Both (4.34) and (4.35) follows these states for points and vectors.

The Bernstein/Hermite matrix of order k

where $k = d+1$ and d is the polynomial degree of the Bernstein polynomials it represent. The matrix is a $k \times k$ invertible matrix defined as

$$\mathbf{B}_d(t,\delta) = \begin{pmatrix} b_{0,d}(t) & b_{1,d}(t) & \dots & b_{d,d}(t) \\ \delta\, Db_{0,d}(t) & \delta\, Db_{1,d}(t) & \cdots & \delta\, Db_{d,d}(t) \\ \vdots & \vdots & \ddots & \vdots \\ \delta^d D^d b_{0,d}(t) & \delta^d D^d b_{1,d}(t) & \cdots & \delta^d D^d b_{d,d}(t) \end{pmatrix}. \tag{4.36}$$

where $d > 0$, $t \in [0,1]$ and $\delta > 0$.
Usually $\delta = 1$, but if the domain is $[a,b]$ and not $[0,1]$, and the input parameter is $s \in [a,b]$ we then must scale and translate, and use $t = \frac{s-a}{b-a}$ and $\delta = \frac{1}{b-a}$.

The special feature about this definition of the matrix (4.36) is the scaling δ^j, where j, the power exponent, is the row number (the first row is numbered 0). The reason for this scaling is of course the scaling of the parameter domain.

As we saw in (4.34) and (4.35) there are two ways we can use this matrix:

1. Compute the Bernstein polynomials and it's derivatives and thus the position and the derivatives of a Bézier curve, expression (4.34).

2. Create a Bézier curve that for a given parameter value t interpolates a given position and d subsequent derivatives (Hermite interpolation), expression (4.35).

The following algorithm creates the matrix $\mathbf{B}_d(t,\delta)$ described in (4.36).

Algorithm 2. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes the extended square matrix $\mathbf{B}_d(t,\delta) \in \mathbb{R}^{d+1\times d+1}$*, contained in the first row, the values of the* $d+1$ *Bernstein polynomials* $\{b_{d,i}(t)\}_{i=0}^{d}$*, and, in the following rows values for each of the d derivatives,* $\{D^j b_{d,i}(t)\}_{i=0}^{d}, \quad j = 1,2,\ldots,d$*, respectively, in each of the* j *following rows. In addition, all rows where the number is* $j > 0$ *(the rows are numbered from 0 to d), are multiplied by* δ^j*. If the matrix is supposed to be used in a general evaluator, then* $\delta = 1$*, and if the matrix is supposed to be used in Hermite interpolation and evaluation of local curves, then we have to use* δ_i*, see subsection 8.2.2, where i is the index of the local curve. The input variables are: the degree d of the Bernstein polynomials, the parameter value* $t \in [0,1]$*, and the scaling factor* δ*.*

matrix⟨double⟩ **BernsteinHermiteMat (** int d, double t, double δ **)**
matrix⟨double⟩ *B(d+1,d+1);* *// The return matrix, dimension* $(d+1)\times(d+1)$*.*
$B_{d-1,0} = 1-t$;
$B_{d-1,1} = t$; *// The general Cox/deBoor like algorithm for*
for *(* int *i=d-2;* $i \geq 0$*;* $i--$ *)* *// - Bernstein polynomials, computing the*
$B_{i,0} = (1-t)\,B_{i+1,0}$; *// - triangle of values of* $b_{i,j}(t)$
for *(* int *j=1;* $j < d-i$*;* j*++)* *// - of degree 1 to d, respectively in each row.*
$B_{i,j} = t\,B_{i+1,j-1} + (1-t)\,B_{i+1,j}$;
$B_{i,d-i} = t\,B_{i+1,d-i-1}$;
$B_{d,0} = -\delta$;
$B_{d,1} = \delta$; *// Multiply all rows except the upper one*
for *(* int *k=2;* $k \leq d$*;* k*++)* *// - with the derivative matrices in the*
double $s = k\,\delta$; *// - expression (4.33), and the scalings,*
for *(* int $i = d$*;* $i > d-k$*;* $i--$ *)* *// - so every row extends the number*
$B_{i,k} = s\,B_{i,k-1}$; *// - of nonzero elements to d.*
for *(* int $j = k-1$*;* $j > 0$*;* $j--$ *)*
$B_{i,j} = s\left(B_{i,j-1} - B_{i,j}\right)$;
$B_{i,0} = -s\,B_{i,0}$;
return *B;*

The order of this algorithm is clearly between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$, and the following table shows the number of multiplications depending on d, for d up to 10.

d	1	2	3	4	5	6	7	8	9	10
multiplications	0	11	30	59	100	155	226	315	424	555

The speed of the algorithm is not essential because the algorithm is only supposed to be executed when the curves are made, or when the sampling is changed (preevaluation). But for small d values ($d < 4$) the algorithm is fairly fast.

4.4.4 Degree Elevation of Bézier Curves

Remember that polynomial based curves can be on the power form,

$$\alpha(t) = \sum_{i=0}^{d} a_i\, t^i,$$

where the coefficients a_i are elements in the embedded space, typically $\mathbb{R}^2$ or $\mathbb{R}^3$.

Any polynomial curve of degree $g < d$ can be expressed on this form, but where the coefficients $|a_i| = 0,\ i > g$. Degree elevation of Bézier curves is just about this, but where we use Bernstein polynomials as basis functions instead of the power basis. Thus, degree elevation can be done by basis change from Bernstein to power basis, raising the degree, and convert the basis back to Bernstein polynomials. An easier way to find the formula is the following,

$$
\begin{aligned}
c(t) &= \sum_{i=0}^{d} b_{i,d}(t)\ c_i = (1-t)\sum_{i=0}^{d} b_{i,d}(t)\ c_i + t\sum_{i=0}^{d} b_{i,d}(t)\ c_i \\
&= (1-t)\sum_{i=0}^{d} \binom{d}{i} t^i (1-t)^{d-i}\ c_i + t\sum_{i=0}^{d} \binom{d}{i} t^i (1-t)^{d-i}\ c_i \\
&= \sum_{i=0}^{d} \binom{d}{i} t^i (1-t)^{d+1-i}\ c_i + \sum_{i=0}^{d} \binom{d}{i} t^{i+1} (1-t)^{d-i}\ c_i \\
&= \frac{d+1-i}{d+1}\sum_{i=0}^{d} b_{i,d+1}(t)\ c_i + \frac{i+1}{d+1}\sum_{i=0}^{d} b_{i+1,d+1}(t)\ c_i \\
&= b_{0,d+1}(t)\ c_0 + \sum_{i=1}^{d} b_{i,d+1}(t)\left(\frac{i}{d+1}\ c_{i-1} + \left(1-\frac{i}{d+1}\right)\ c_i\right) + b_{d+1,d+1}(t)\ c_d,
\end{aligned}
$$

which can be expressed in matrix form:

The Bézier degree elevation matrix

Given a Bézier curve of degree d, $c(t) = \sum_{i=0}^{d} b_{i,d}(t)\ c_i$. To increase the degree to $d+1$ we must generate a new set of control points where we keep the first point, create d new points in the middle and then keep the last one. On vector/matrix form vi get, $\tilde{\mathbf{C}} = \mathbb{D}_d\ \mathbf{C}$, where $\tilde{\mathbf{C}} = (\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_d)$, $\mathbf{C} = (c_0, c_1, \ldots, c_d)$, and $\mathbb{D}_d$ is the Bézier degree elevation matrix of dimension $d \times (d+1)$, it is

$$
\begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ \vdots \\ \tilde{c}_d \end{pmatrix} = \begin{pmatrix} 1-\frac{d}{d+1} & \frac{d}{d+1} & 0 & \cdots & 0 \\ 0 & 1-\frac{d-1}{d+1} & \frac{d-1}{d+1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1-\frac{1}{d+1} & \frac{1}{d+1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{d-1} \\ c_d \end{pmatrix} \tag{4.37}
$$

For the Bézier curve of degree $d+1$ is the set of $d+2$ control points $\{c_0, \tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_{d-1}, \tilde{c}_d, c_d\}$.

From the matrix in (4.37) we can clearly see that degree elevation is corner cutting, which is clearly illustrated in Figure 4.13. All the new control points we get by increasing the degree by one are on the old control polygon. The degree elevation matrix from degree 2 to 3, the matrix from degree 3 to 4, the matrix from degree 4 to 5 and the matrix from

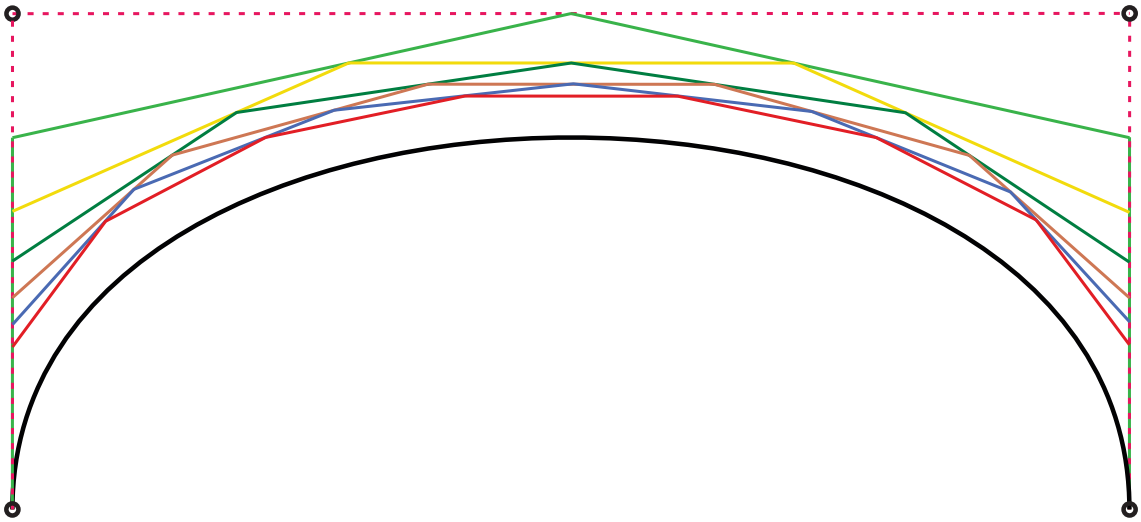


Figure 4.13: *A 3^{rd}-degree Bézier curve (black) and the four control points (bullets) and control polygon (dotted red) and the control polygons of 4^{th}-degree (green), 5^{th}-degree (orange), 6^{th}-degree (dark green), 7^{th}-degree (dark brown), 8^{th}-degree (blue) and 9^{th}-degree (dark red) Bézier curve generated from the first one with the degree elevation algorithm.*

degree 5 to 6 are all shown below,

$$\begin{pmatrix} \frac{1}{3} & \frac{2}{3} & 0 \\ 0 & \frac{2}{3} & \frac{1}{3} \end{pmatrix}, \quad \begin{pmatrix} \frac{1}{4} & \frac{3}{4} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{3}{4} & \frac{1}{4} \end{pmatrix}, \quad \begin{pmatrix} \frac{1}{5} & \frac{4}{5} & 0 & 0 & 0 \\ 0 & \frac{2}{5} & \frac{3}{5} & 0 & 0 \\ 0 & 0 & \frac{3}{5} & \frac{2}{5} & 0 \\ 0 & 0 & 0 & \frac{4}{5} & \frac{1}{5} \end{pmatrix}, \quad \begin{pmatrix} \frac{1}{6} & \frac{5}{6} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{2}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{5}{6} & \frac{1}{6} \end{pmatrix}.$$

It follows that for simple degree elevation we obtain the following coefficients expressed with the coefficients from the polynomial degree below:

$$\begin{array}{l|llllll} 2 \to 3 & c_1, & \frac{1}{3}(c_1+2c_2), & \frac{1}{3}(2c_2+c_3), & c_3. & & \\ 3 \to 4 & c_1, & \frac{1}{4}(c_1+3c_2), & \frac{1}{2}(c_2+c_3), & \frac{1}{4}(3c_3+c_4), & c_4. & \\ 4 \to 5 & c_1, & \frac{1}{5}(c_1+4c_2), & \frac{1}{5}(2c_2+3c_3), & \frac{1}{5}(3c_3+2c_4), & \frac{1}{5}(4c_4+c_5), & c_5. \end{array}$$

An example of continuous degree elevation is given in Figure 4.13. It is initially a 3^{rd}-degree Bézier curve where the four control points are marked with black circles. The degree of the curve is raised degree by degree up to degree 9, and as we can see on the dark red control polygon we will finally have 10 control points. The example also shows that the control polygon converges quite slowly towards the curve.

Table 4.1: *Matrices for basis change*

Type of basis function	from power basis	to power basis
Hermite 3^{rd}-degree	$\mathfrak{H} = \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$	$\mathfrak{H}^{-1} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 3 \end{bmatrix}$
Bernstein 3^{rd}-degree	$\mathfrak{B} = \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\mathfrak{B}^{-1} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & \frac{1}{3} & \frac{2}{3} & 1 \\ 0 & 0 & \frac{1}{3} & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

4.5 Converting between Hermite- and Bézier- format

Hermite curves and Bézier curves are widely used in computer programs. Therefore, it can be very useful to be able to convert from one format to another. In fact, all polynomial-based curve formats of the same degree and over the same domain can be converted to each other with a basis-shift matrix.

In Table 4.1 we find the basis-shift matrices for the 3^{rd}-degree Hermite and Bernstein basis functions. If we name the set (vectors) of 3^{rd}-degree basis functions defining $\mathscr{P}_3[0,1]$ for

$$\begin{aligned}
\mathbf{M}_3(t) &= \begin{pmatrix} 1, & t, & t^2, & t^3 \end{pmatrix}^T && \text{Power basis,} \\
\mathbf{H}_3(t) &= \begin{pmatrix} 1-3t^2+2t^3, & 3t^2-2t^3, & t-2t^2+t^3, & -t^2+t^3 \end{pmatrix}^T && \text{Hermite basis,} \\
\mathbf{T}^3(t) &= \begin{pmatrix} (1-t)^3, & 3t(1-t)^2, & 3t^2(1-t), & t^3 \end{pmatrix}^T && \text{Bézier basis,}
\end{aligned}$$

we get

$$\mathbf{T}^3(t) = \mathfrak{B}\ \mathfrak{H}^{-1}\ \mathbf{H}_3(t)$$

ie

$$\begin{pmatrix} (1-t)^3 \\ 3t(1-t)^2 \\ 3t^2(1-t) \\ t^3 \end{pmatrix} = \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix} \begin{pmatrix} 1-3t^2+2t^3 \\ 3t^2-2t^3 \\ t-2t^2+t^3 \\ -t^2+t^3 \end{pmatrix}$$

Given a curve $c(t) = \mathbf{p}\mathbf{H}_3(t)$, where $\mathbf{p} = (c(0),\ c(1)\ c'(0),\ c'(1),)$. It follows that we can find the control points $\mathbf{c} = (c_0,\ c_1,\ c_2,\ c_3)$ of the equivalent Bézier curve, $c(t) = \mathbf{c}\mathbf{T}^3(t)$, by

$$\begin{aligned}
\mathbf{c}\ \mathbf{T}^3(t) &= \mathbf{p}\ \mathbf{H}_3(t) \\
\mathbf{c}\ \mathfrak{B}\ \mathfrak{H}^{-1}\mathbf{H}_3(t) &= \mathbf{p}\ \mathbf{H}_3(t) \\
\mathbf{c}\ \mathfrak{B}\ \mathfrak{H}^{-1} &= \mathbf{p} \\
\mathbf{c} &= \mathbf{p}\ \mathfrak{H}\ \mathfrak{B}^{-1}.
\end{aligned}$$

It gives

$$(c_0,\ c_1,\ c_2,\ c_3) = \begin{pmatrix} c(0),\ c(1),\ c'(0),\ c'(1) \end{pmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & -\frac{1}{3} & 0 \end{bmatrix}.$$

4.6 Implementation and Tessellation

Rendering parametric curves normally requires tessellation which can be done by either the graphics device, GPU, or the CPU. Hermite curves and Bézier curves normally have no internal discontinuities for any order of derivatives, with the exception of geometric discontinuity from singularities based on cusp, i.e. where the velocity, the length of the 1^{st} derivative, is zero. Therefore, it is not natural to divide the parameter domain into partitions. What we normally do is tessellate the curve, ie. divide the curve into small parts that are usually rendered as line segments. There are three widely used tessellation methods for curves,

- Uniform tessellation
- Tessellation based on speed
- Tessellation based on curvature

Uniform tessellation is based on $n+1$ points, which means dividing the domain into n equal parts. For Hermite and Bézier curves, the domain is 1, ie. $dt = \frac{1}{n}$ so that the $n+1$ points that determine the rendering become $\{c(i*dt)\}_{i=0}^{n}$. This is the most used method.

Tessellation based on speed can be performed more or less complex. A simple version is to first calculate the curve length $l(c)$, see (4.5), with Romberg integration, modified Algorithm 14. Then $dl = l(c)/n$, then start with the first sample point $c(0)$ and $t = 0$. For each new t-value, we must first calculate the time step dt. It is based on $|c'(t)|\ dt = dl$, and it follows that $dt = \frac{dl}{|c'(t)|}$. Then we get the next sample point $c(t+dt)$ and for the next step we update $t = t + dt$. The final sample point is $c(1)$. A more complex method is to correct for that the speed may change over a step. To calculate the time step we then get $\left(|c'(t)| + \frac{\langle c'(t), c''(t)\rangle}{2|c'(t)|}dt\right) dt = dl$. This means solving a quadratic equation instead of a linear one.

Tessellation based on curvature do not start with number of sample points, but only indicates a minimum number of sample points. The main input is δ, the maximum allowed distance between a curve segment and a straight line. Calculating a step dt is based on the arc length $b \approx dt\ |c'|$, the curvature κ, (4.7) (where $r = \frac{1}{\kappa}$) and $|c'|$. We have $\cos\alpha = \frac{r-\delta}{r}$ and $\alpha = \frac{b}{r}$, so

$$\begin{aligned} \cos\frac{b}{r} &= 1 - \frac{\delta}{r}, \\ b &= r\ \arccos\left(1 - \delta\ \kappa\right), \\ dt &= \frac{\arccos\left(1 - \delta\ \kappa\right)}{|c'|\kappa}. \end{aligned}$$

This expression assumes that the curvature (actually the product of the curvature and the speed) is not very small. Therefore, if we have anything near a straight line, we must base the algorithm on a small number of points and uniform sampling.

Chapter 5

Classical interpolation theory

In his book on interpolation, Thiele characterized interpolation as *the art of reading between lines in a numerical table* (see [155]). Until the introduction of computers and especially computer graphics this was actually a very precise formulation.

The history includes Astronomy in Mesopotamia and Babylon, mathematicians in ancient Greece, Chinese mathematicians and Indian astronomer.

In Europe, apparently unaware of results obtained much earlier in other parts of the world, interpolation theory started to develop after the great "scientific revolution" around 1500 AD. In particular the new developments in astronomy and physics, initiated by Copernicus, continued by Kepler and Galileo, and culminating in the theories of Newton, gave strong impetus to further advancement of mathematics, including what is now called "classical" interpolation theory.

5.1 Divided differences

Given a point set $\{p_i\}_{i=0}^n = \{(x_0,y_0),(x_1,y_1),...,(x_n,y_n)\}$. A definition of the first-order divided difference is,

$$[y_0,y_1] = \frac{y_0 - y_1}{x_0 - x_1}.$$

Higher-order divided differences is defined recursively,

$$[y_0,\ldots,y_n] = \frac{[y_0,\ldots,y_{n-1}] - [y_1,\ldots,y_n]}{x_0 - x_n}, \quad \text{for} \quad n > 1.$$

Order 0 divided difference is notated $[y_0] = y_0$. The recursive process can be put in a tabular form, i.e.

$$\begin{array}{lll} x_0 & y_0 = [y_0] & \\ & & [y_0,y_1] \\ x_1 & y_1 = [y_1] & & [y_0,y_1,y_2] \\ & & [y_1,y_2] & & [y_0,y_1,y_2,y_3]. \\ x_2 & y_2 = [y_2] & & [y_1,y_2,y_3] \\ & & [y_2,y_3] & \\ x_3 & y_3 = [y_3] & \end{array}$$

As an example, given a point set $\{(x_i,y_i)\}_{i=0}^{3} = \{(0,0),(1,1),(2,1),(3,0)\}$. From this set of data we get the following recursion,

$$
\begin{array}{llll}
0 & [y_0] = 0 & & \\
 & & [y_0,y_1] = \frac{0-1}{0-1} = 1 & \\
1 & [y_1] = 1 & & [y_0,y_1,y_2] = \frac{1-0}{0-2} = \frac{-1}{2} \\
 & & [y_1,y_2] = \frac{1-1}{0-1} = 0 & \qquad [y_0,y_1,y_2,y_3] = \frac{\frac{-1}{2}-\frac{-1}{2}}{0-3} = 0. \\
2 & [y_2] = 1 & & [y_1,y_2,y_3] = \frac{0+1}{1-3} = \frac{-1}{2} \\
 & & [y_2,y_3] = \frac{1-0}{0-1} = -1 & \\
3 & [y_3] = 0 & &
\end{array}
$$

If the data points are given as a function, i.e. organised as t_i and $f(t_i)$, the data point is $(t_i, f(t_i))$. For any function $f : \mathbb{R} \to \mathbb{R}^m$, $m \in \mathbb{Z}$, and for any two values $t_0 \neq t_1$, the first-order divided difference is

$$f[t_0,t_1] = \frac{f(t_0) - f(t_1)}{t_0 - t_1}.$$

Higher-order divided differences is then defined recursively,

$$f[t_0,\ldots,t_n] = \frac{f[t_0,\ldots,t_{n-1}] - f[t_1,\ldots,t_n]}{t_0 - t_n}, \quad \text{for all } \; n > 1.$$

Some important properties,

◇ **Linearity**

$$(f+g)[t_0,\ldots,t_n] = f[t_0,\ldots,t_n] + g[t_0,\ldots,t_n]$$
$$(\lambda \cdot f)[t_0,\ldots,t_n] = \lambda \cdot f[t_0,\ldots,t_n]$$

◇ **Leibniz rule**

$$(f \cdot g)[t_0,\ldots,t_n] = f[t_0] \cdot g[t_0,..,t_n] + f[t_0,t_1] \cdot g[t_1,..,t_n] + f[t_0,..,t_n] \cdot g[t_n],$$

◇ **Divided differences are symmetric**, If σ is a permutation of $\{0,\ldots,n\}$ then

$$f[t_0,\ldots,t_n] = f[t_{\sigma(0)},\ldots,t_{\sigma(n)}],$$

◇ **Divided differences and mean value theorem**

$f[t_0,\ldots,t_n] = \frac{f^{(n)}(\xi)}{n!}$ where ξ is in the open interval determined by the smallest and largest of the t_m's

The formula for divided differences can be expanded as we can see below,

$$
\begin{aligned}
f[t_0] &= f(t_0), \\
f[t_0,t_1] &= \tfrac{f(t_0)}{(t_0-t_1)} + \tfrac{f(t_1)}{(t_1-t_0)}, \\
f[t_0,t_1,t_2] &= \tfrac{f(t_0)}{(t_0-t_1)(t_0-t_2)} + \tfrac{f(t_1)}{(t_1-t_0)(t_1-t_2)} + \tfrac{f(t_2)}{(t_2-t_0)(t_2-t_1)}, \\
f[t_0,t_1,t_2,t_3] &= \tfrac{f(t_0)}{(t_0-t_1)(t_0-t_2)(t_0-t_3)} + \tfrac{f(t_1)}{(t_1-t_0)(t_1-t_2)(t_1-t_3)} + \tfrac{f(t_2)}{(t_2-t_0)(t_2-t_1)(t_2-t_3)} + \\
&\quad \tfrac{f(t_3)}{(t_3-t_0)(t_3-t_1)(t_3-t_2)}.
\end{aligned}
$$

We then get a direct formula for divided differences,

$$f[t_0,\ldots,t_n] = \sum_{i=0}^{n} \frac{f(t_i)}{\prod_{k=0,k\neq i}^{n}(t_i - t_k)},$$

which can also be formulated more compact,

$$f[t_0,...,t_n] = \sum_{i=0}^{n} \frac{f(t_i)}{\omega_n'(t_i)}, \tag{5.1}$$

where $\omega_n(t) = (t-t_0)\cdots(t-t_n)$, i.e.

$$\omega_n(t) = \prod_{k=0}^{n}(t-t_k), \quad \text{and it follows that} \quad \omega_n'(t_i) = \prod_{k=0,k\neq i}^{n}(t_i - t_k). \tag{5.2}$$

To extend divided differences to include equal values we has to look at derivatives. Remember that

$$f'(t_0) = \lim_{t_1\to t_0} \frac{f(t_0)-f(t_1)}{t_0-t_1}.$$

Thus we can define

$$f[t_i,t_i] = f'(t_i).$$

and if, in the expression for divided differences, t_i is repeated n times we get a connection between divided differences and the derivatives of a function in general, notice the inverse scaling $(n-1)!$,

$$f[t_i,t_i,\ldots,t_i] = \frac{f^{(n-1)}(t_i)}{(n-1)!} \quad \text{ie } f[t_i \text{ repeated } n \text{ times}]. \tag{5.3}$$

An important issue is that divided differences can be vector or point-valued. That is, if $f(t_i) \in \mathbb{R}^d$, $d > 0$ then all the associated divided differences $f[t_i,\ldots,t_{i+k}] \in \mathbb{R}^d$.

5.2 Newton polynomial

We start with a first order divided differences. Given a set of values $t_i \in I \subset \mathbb{R}$. Then for $t \in I$ but $t \neq t_0$ we get

$$f[t,t_0] = \frac{f(t)-f(t_0)}{t-t_0}.$$

It follows that the value of f **at any** t can be written as

$$f(t) = f(t_0) + (t-t_0)f[t,t_0]. \tag{5.4}$$

We continue with a second order divided differences,

$$f[t,t_0,t_1] = \frac{f[t,t_0]-f[t_0,t_1]}{t-t_1}.$$

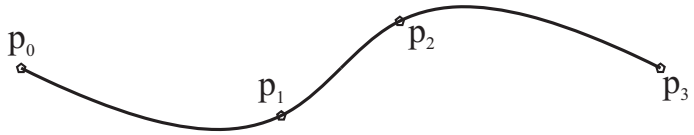


Figure 5.1: *The curve from the example below. The interpolation points are highlighted.*

Turning this we get

$$f[t,t_0] = f[t_0,t_1] + (t-t_1)f[t,t_0,t_1]. \tag{5.5}$$

we can now substitute $f[t,t_0]$ in (5.4) with expression (5.5). Then we can repeat this process with higher and higher order divided differences. The result is

$$\begin{aligned} f(t) = f[t_0] &+ (t-t_0)\, f[t_0,t_1] \\ &+ (t-t_0)(t-t_1)\, f[t_0,t_1,t_2] \\ &+ (t-t_0)(t-t_1)(t-t_2)\, f[t_0,t_1,t_2,t_3] + \ldots \end{aligned}$$

A compact notations is

$$f(t) = \sum_{j=0}^{k} a_j n_j(t), \tag{5.6}$$

where $a_j = f[t_0,\ldots,t_j]$, and $n_0(t) \equiv 1$ and $n_j(t) = \prod_{i=0}^{j-1}(t-t_i)$ when $j > 0$.

If we do not use a function notation but a set of points connected to each parameter value, i.e. $\{(t_i,p_i)\}$ we get $a_j = [p_0,\ldots,p_j]$, $p_i \in \mathbb{R}^d$, $i = 0,1,\ldots,j$, $d > 1$.

Newton polynomials are used for polynomial interpolation. For a given set of distinct parameter values $\{t_j\}_{j=0}^d$ and corresponding points $\{p_j\}_{j=0}^d$, the set gives the polynomial of the least degree that at each value t_j fit the corresponding point p_j, i.e. the function interpolate all points. As an example, let's use the curve in the expression (4.4) in Chapter 4, which is plotted in Figure 4.3. The parameter range of the curve is $[0,1]$. We divide it into 3 and get the following, $t_0 = 0$, $t_1 = \frac{1}{3}$, $t_2 = \frac{2}{3}$ and $t_3 = 1$. When we use these four parameter values in (4.4) we get the points, $p_0 = (0,0)$, $p_1 = (\frac{11}{9}, -\frac{2}{9})$, $p_2 = (\frac{16}{9}, \frac{2}{9})$ and $p_3 = (3,0)$. If we calculate the divided differences, we get,

$$f\left[0,\frac{1}{3}\right] = \frac{\begin{pmatrix}0\\0\end{pmatrix} - \begin{pmatrix}\frac{11}{9}\\ \frac{-2}{9}\end{pmatrix}}{0-\frac{1}{3}} = \begin{pmatrix}\frac{11}{3}\\ -\frac{2}{3}\end{pmatrix}, \quad f\left[\frac{1}{3},\frac{2}{3}\right] = \frac{\begin{pmatrix}\frac{11}{9}\\ \frac{-2}{9}\end{pmatrix} - \begin{pmatrix}\frac{16}{9}\\ \frac{2}{9}\end{pmatrix}}{\frac{1}{3}-\frac{2}{3}} = \begin{pmatrix}\frac{5}{3}\\ \frac{4}{3}\end{pmatrix}$$

$$f\left[\frac{2}{3},1\right] = \frac{\begin{pmatrix}\frac{19}{9}\\ \frac{2}{9}\end{pmatrix} - \begin{pmatrix}3\\0\end{pmatrix}}{\frac{2}{3}-1} = \begin{pmatrix}\frac{11}{3}\\ -\frac{2}{3}\end{pmatrix}, \quad f\left[0,\frac{1}{3},\frac{2}{3}\right] = \frac{\begin{pmatrix}\frac{11}{3}\\ \frac{-2}{3}\end{pmatrix} - \begin{pmatrix}\frac{5}{3}\\ \frac{4}{3}\end{pmatrix}}{0-\frac{2}{3}} = \begin{pmatrix}-3\\3\end{pmatrix}$$

$$f\left[\frac{1}{3},\frac{2}{3},1\right] = \frac{\begin{pmatrix}\frac{19}{9}\\ \frac{2}{9}\end{pmatrix} - \begin{pmatrix}3\\0\end{pmatrix}}{\frac{2}{3}-1} = \begin{pmatrix}3\\-3\end{pmatrix}, \quad f\left[0,\frac{1}{3},\frac{2}{3},1\right] = \frac{\begin{pmatrix}-3\\3\end{pmatrix} - \begin{pmatrix}3\\-3\end{pmatrix}}{0-1} = \begin{pmatrix}6\\-6\end{pmatrix}$$

The points marked in red are the coefficients a_1, a_2 and a_3, and together with $a_0 = p_0$ and $t_0 = 0$, $t_1 = \frac{1}{3}$, $t_2 = \frac{2}{3}$ and $t_3 = 1$ they define a curve using (5.6). The curve is the same as in Figure 4.3 and is in Figure 5.1 plotted along with the four marked interpolation points.

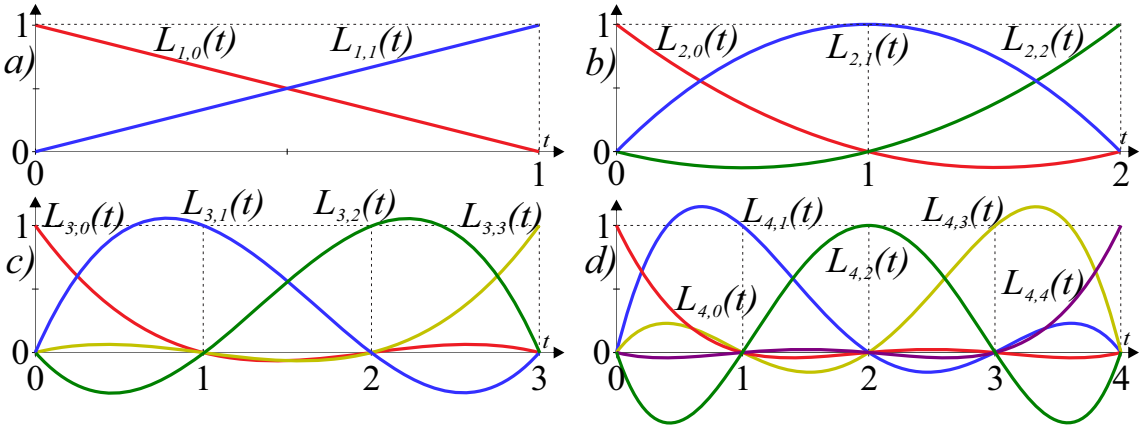


Figure 5.2: *A plot of four sets of Lagrange interpolation polynomials (5.8). The set* **a)** *is of polynomial degree one, forming first degree curves. The set* **b)** *is of polynomial degree two, forming second degree curves. The set* **c)** *is of polynomial degree three, forming third degree curves. The set* **d)** *is of polynomial degree four, forming forth degree curves. All Lagrange interpolation polynomials are named $L_{d,i}(t)$, where d is the polynomial degree and i is the index of the interpolation point the function is connected to.*

5.3 Lagrange polynomials

A very elegant alternative representation of Newton's general formula, ie (5.6), which does not require computation of divided differences, was published in 1779 by Waring [159], this formula is

$$\begin{aligned} f(t) = & f(t_0)\frac{(t-t_1)(t-t_2)(t-t_3)\cdots}{(t_0-t_1)(t_0-t_2)(t_0-t_3)\cdots} + \\ & f(t_1)\frac{(t-t_0)(t-t_2)(t-t_3)\cdots}{(t_1-t_0)(t_1-t_2)(t_1-t_3)\cdots} + \\ & f(t_2)\frac{(t-t_0)(t-t_1)(t-t_3)\cdots}{(t_2-t_0)(t_2-t_1)(t_2-t_3)\cdots} + \ldots . \end{aligned}$$

Notice the pattern of the t_i in the expression above. A compact notation of the formula is,

$$f(t) = \sum_{i=0}^{d} f(t_i)\, L_{d,i}(t), \tag{5.7}$$

where d is the polynomial degree, and where the basis functions, i.e the Lagrange polynomials are

$$L_{d,i}(t) = \prod_{j=0, j\neq i}^{d} \frac{(t-t_j)}{(t_i-t_j)}. \tag{5.8}$$

In Figure 5.2 are all Lagrange polynomials $L_{d,i}(t)$ up to degree four plotted. Note that $L_{d,i}(t_i) = 1$ and $L_{d,i}(t_j) = 0, \quad j \neq i$. Lagrange polynomials[1] are used for polynomial

[1]Today, it is common to credit these polynomials to Lagrange, which however published it 16 years after Waring [100]. According to Euler [61], the formula (5.7) can be linked to a related representation of Newton's formula, and according to Joffe [94], it was Gauss who first noticed the connection between the formulae by Newton, Euler, and Waring-Lagrange, as appears from his posthumous works [72], although Gauss did not refer to his predecessors.

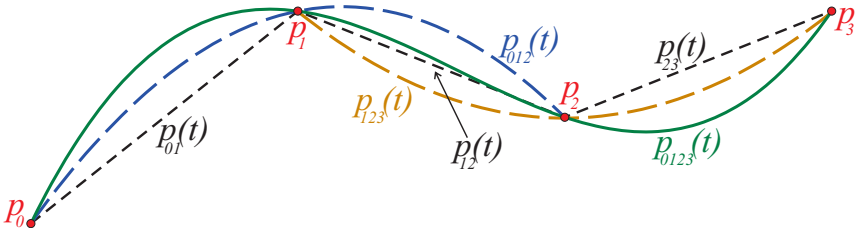


Figure 5.3: *Four points p_0, p_1, p_2 and p_3 are plotted in red. Three 1st degree curves $p_{01}(t), p_{12}(t)$, and $p_{23}(t)$ interpolate the points and are plotted in dashed black. Two 2nd degree curve $p_{012}(t)$ and $p_{123}(t)$ are plotted in dashed blue and dashed orange, and one 3rd degree curve $p_{0123}(t)$ in solid green interpolates all four points.*

interpolation. For a given set of distinct parameter values $\{t_j\}_{j=0}^{d}$ and corresponding points $\{p_j\}_{j=0}^{d}$, the set and Lagrange polynomials gives the polynomial of the least degree that at each value t_j fit the corresponding point p_j, i.e. the function interpolate all points.

Lemma 5.1. *The Newton and Lagrange polynomials are two different representations as expressions of the same polynomial.*

The proof for lemma 5.1 can be found in appendix C.1.

Although the Newton and Lagrange polynomials are the same polynomials, they have different advantages. Lagrange polynomials are the easiest and fastest to use because the coefficients are the interpolation points. The Newton polynomials have the advantage that one can add new interpolation points by adding a new term.

The Newton and Lagrange polynomials also show us a more basic result, namely that only one polynomial of degree greater or equal to n interpolates $n+1$ point at $n+1$ given values. This gives us the following theorem.

Theorem 5.1. *If $x_0, x_1, \ldots, x_n$ are distinct real numbers, then for arbitrary values $y_0, y_1, \ldots, y_n$, there is a unique polynomial p_n of degree at most n such that $p_n(x_i) = y_i, \quad 0 \leq i \leq n$.*

Proof. Suppose there were two such polynomials, p_n and q_n. Then the polynomial $p_n - q_n$ would have the property $(p_n - q_n)(x_i) = 0$ for $0 \leq i \leq n$. Since the degree of $p_n - q_n$ can be at most n, this polynomial can have at most n zeros if it is not the 0 polynomial. Since the x_i are distinct, $p_n - q_n$ should have $n+1$ zeros; it must therefore be 0. Hence, $p_n = q_n$. □

5.3.1 Neville's Algorithm

The algorithm is named after the English mathematician E.H. Neville (1889-1961). It is a geometric interpretation of Newton/Lagrange polynomials. We take (5.6), expand it, reorganise it, and introduce the Neville's notation of an interpolation polynomials $p_{\ldots}(t)$,

$$\begin{aligned}
f(t) &= p_0 + (t-t_0)f[t_0,t_1] + (t-t_0)(t-t_1)f[t_0,t_1,t_2] \\
&= p_0 + \tfrac{t-t_0}{t_0-t_1}(p_0-p_1) + \tfrac{(t-t_0)(t-t_1)}{t_0-t_2}(f[t_0,t_1]-f[t_1,t_2]) \\
&= p_0 + \tfrac{t-t_0}{t_0-t_1}(p_0-p_1) + \tfrac{(t-t_0)(t-t_1)}{(t_0-t_2)}(\tfrac{p_0-p_1}{t_0-t_1} - \tfrac{p_1-p_2}{t_1-t_2}) \\
&= \tfrac{t-t_1}{t_0-t_1}p_0 + \tfrac{t_0-t}{t_0-t_1}p_1 + \tfrac{(t-t_0)(t-t_1)}{(t_0-t_2)}(\tfrac{p_0-p_1}{t_0-t_1} - \tfrac{p_1-p_2}{t_1-t_2}) \\
&= p_{01}(t) + \tfrac{(t-t_0)(t-t_1)}{(t_0-t_2)}(\tfrac{p_0-p_1}{t_0-t_1} - \tfrac{p_1-p_2}{t_1-t_2}) \\
&= p_{01}(t) + \tfrac{t-t_0}{t_0-t_2}p_{01} + \tfrac{t_0-t}{t_0-t_2}p_{12} \\
&= \tfrac{t-t_2}{t_0-t_2}p_{01}(t) + \tfrac{t_0-t}{t_0-t_2}p_{12}(t) \\
&= p_{012}(t).
\end{aligned}$$

This procedure can be extended to higher degrees in the same way, and is essentially the same as used in the proof of Lemma 5.1. This means that we now have a procedure for interpolation of a point set $p_0, p_1, p_2, \ldots$ that actually gives Newton / Lagrange polynomials. The first step is to calculate,

$$p_{01}(t) = \frac{t-t_1}{t_0-t_1}p_0 + \frac{t_0-t}{t_0-t_1}p_1, \quad p_{12}(t) = \frac{t-t_2}{t_1-t_2}p_1 + \frac{t_1-t}{t_1-t_2}p_2,$$

and further $p_{23},\ p_{34}, \ldots$ which all give straight lines from p_0 to p_1, from p_1 to p_2 and so on. The next levels are

$$\begin{aligned}
&p_{012}(t) = \tfrac{t-t_2}{t_0-t_2}p_{01} + \tfrac{t_0-t}{t_0-t_2}p_{12}, \qquad p_{123}(t) = \tfrac{t-t_3}{t_1-t_3}p_{12} + \tfrac{t_1-t}{t_1-t_3}p_{23}, \\
&p_{0123}(t) = \tfrac{t-t_3}{t_0-t_3}p_{012} + \tfrac{t_0-t}{t_0-t_3}p_{123},
\end{aligned}$$

where in the first line there are 2^{nd}-degree curves interpolating p_0, p_1, p_2 and p_1, p_2, p_3, and in the last line there is a 3^{rd}-degree curve interpolating all 4 point. Furthermore, we can compute several levels and thus more interpolation points with the same procedure.

In Figure 5.3 there is an example of interpolating 4 points with Neville's algorithm. In his book *Pyramid Algorithms* [77], Ron Goldman use Neville's algorithm as one of the basic pyramid algorithms.

5.4 Hermite interpolation

Hermite interpolation is a method closely related to Newton's classical interpolation formulae defined with divided differences (5.6). Hermite interpolation allows us to interpolate points together with a given number of subsequent derivatives in each point. The result of the interpolation is a polynomial with a degree equal to the number of points + the number of derivatives used together in the points - 1 (can also be less if the result is a degenerate polynomial). In Hermite interpolation, we start by treating derivatives as extra points. In the divided difference table, the points are repeated corresponding to the number of derivatives. To avoid division by zero, we replace the expression with derivatives multiplied by a constant, depending on the order. For example, if a value t_i is repeated n times, from (5.3) we have

$$f[t_i, t_i, ..., t_i] = \frac{f^{(n-1)}(t_i)}{(n-1)!},$$

i.e. for $n = 4$ is, $f[t_i,t_i,t_i,t_i] = \frac{1}{6}f^{(3)}(t_i)$. (NB, divided differences can be vector valued.)

In Hermite interpolation we start with two points, $f(t_i)$ and $f(t_{i+1})$, $f:\mathbb{R}\to\mathbb{R}^d$, $d=1,2,3,...$ and two 1^{st}-derivatives $f'(t_i)$ and $f'(t_{i+1})$ connected to each point. We then make a 3^{rd}-degree polynomial using Hermite interpolation in the two points together with their respective 1^{st}-derivatives using (5.6)[2],

$$\begin{aligned} c_i(t) = f(t_i) &+ (t-t_i)\ f[t_i,t_i] \\ &+ (t-t_i)(t-t_i)\ f[t_i,t_i,t_{i+1}] \\ &+ (t-t_i)(t-t_i)(t-t_{i+1})\ f[t_i,t_i,t_{i+1},t_{i+1}]. \end{aligned} \tag{5.9}$$

If we "nest out" the divided differences, we get

$$\begin{aligned} c_i(t) = f(t_i) &+ (t-t_i)\ f'(t_i) \\ &+ (t-t_i)(t-t_i)\ \frac{f'(t_i)-f[t_i,t_{i+1}]}{t_i-t_{i+1}} \\ &+ (t-t_i)(t-t_i)(t-t_{i+1})\ \frac{\frac{f'(t_i)-f[t_i,t_{i+1}]}{t_i-t_{i+1}} - \frac{f[t_i,t_{i+1}]-f'(t_{i+1})}{t_i-t_{i+1}}}{t_i-t_{i+1}}. \end{aligned}$$

Since $(t-t_{i+1}) = (t-t_i)+(t_i-t_{i+1})$ and we use this to substitute the third factor in the third line, we get

$$\begin{aligned} c_i(t) = f(t_i) &+ (t-t_i)\ f'(t_i) \\ &+ (t-t_i)^2\ \frac{2f'(t_i)+f'(t_{i+1})-3f[t_i,t_{i+1}]}{t_i-t_{i+1}} \\ &+ (t-t_i)^3\ \frac{f'(t_i)+f'(t_{i+1})-2f[t_i,t_{i+1}]}{(t_i-t_{i+1})^2}. \end{aligned} \tag{5.10}$$

Note that by using Newton polynomials, we can add derivatives to the points or elsewhere (with some restrictions) to the formula (5.9). In Figure 5.4, a curve $f(t)$, based on a combination of trigonometric and polynomial formulas, is plotted in dashed green. In solid red, a 3^{rd} degree polynomial curve is made by Hermite interpolation of two points on the curve and their 1st derivatives. We see that the point should have been closer to get a better approximation.

However, (5.10) can be transformed into a more convenient and familiar format. Let's start by expanding the formula (5.10),

$$\begin{aligned} c_i(t) = f(t_i) &+ \frac{(t-t_i)}{(t_{i+1}-t_i)}(t_{i+1}-t_i)\,f'(t_i) \\ &+ \frac{(t-t_i)^2}{(t_{i+1}-t_i)^2}(t_{i+1}-t_i)\left(3\frac{f(t_i)-f(t_{i+1})}{(t_i-t_{i+1})} - 2f'(t_i) - f'(t_{i+1})\right) \\ &+ \frac{(t-t_i)^3}{(t_{i+1}-t_i)^3}(t_{i+1}-t_i)\left(f'(t_i)+f'(t_{i+1}) - 2\frac{f(t_i)-f(t_{i+1})}{(t_i-t_{i+1})}\right). \end{aligned}$$

[2]The reason for using an index i at the curve $c_i(t)$ in (5.9) and the following expressions will be explained on page 68.

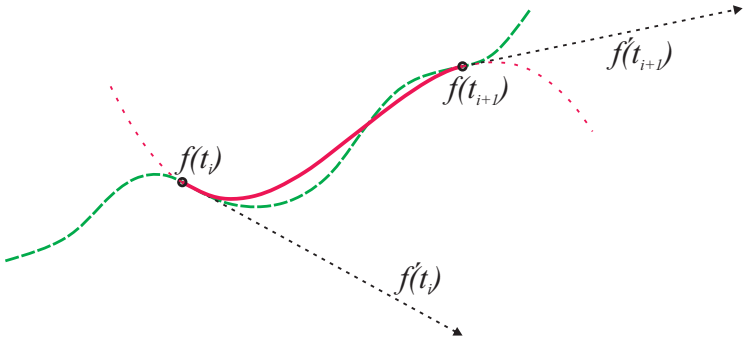


Figure 5.4: *A curve $f(t)$ (dashed green) and a 3^{rd} degree polynomial curve (solid red) that is an Hermite interpolation of f at t_i and t_{i+1}.*

We then reorganise it,

$$\begin{aligned} c_i(t) = &\left(1-3w_i(t)^2+2w_i(t)^3\right) f(t_i) \\ &+ \left(3w_i(t)^2-2w_i(t)^3\right) f(t_{i+1}) \\ &+ (t_{i+1}-t_i)\left(w_i(t)f'(t_i)-2w_i(t)^2f'(t_i)+w_i(t)^3f'(t_i)\right) \\ &+ (t_{i+1}-t_i)\left(w_i(t)^2f'(t_{i+1})+w_i(t)^3f'(t_{i+1})\right). \end{aligned}$$

Thus, the cubic polynomials is now reformulated to what we call the geometric form:

$$c_i(t) = f(t_i)H_{i,3,0}(t) + f(t_{i+1})H_{i,3,1}(t) + f'(t_i)H_{i,3,2}(t) + f'(t_{i+1})H_{i,3,3}(t) \tag{5.11}$$

where

$$\begin{aligned} H_{i,3,0}(t) &= 2w_i(t)^3-3w_i(t)^2+1, \\ H_{i,3,1}(t) &= -2w_i(t)^3+3w_i(t)^2, \\ H_{i,3,2}(t) &= \Delta t_i\left(w_i(t)^3-2w_i(t)^2+w_i(t)\right), \\ H_{i,3,3}(t) &= \Delta t_i\left(w_i(t)^3-w_i(t)^2\right), \end{aligned} \tag{5.12}$$

and where

$$w_i(t) = \frac{t-t_i}{t_{i+1}-t_i}, \quad \text{and} \quad \Delta t_i = t_{i+1}-t_i. \tag{5.13}$$

In (5.11) is the start and end points and their respective 1^{st}-derivatives, the coefficients. The basis functions (5.12) are defined using the "translation and scaling function" $w_i(t)$, (5.13), which you will also find (with an extra index) in the definition of B-splines, see (6.11). Note that the sum of the first two basis functions in (5.12) is 1, ie they are invariant under affine transformations. The last two basis functions are connected to vectors and therefore do not have to sum up to 1. These two coefficients are not affected by translation, but they are affected by any rotations.

Remark 2. *However, in a system of homogeneous coordinates, as we find in graphical systems such as OpenGL, both translation and rotation will work well for all four coefficients in (5.11) if the first two coefficients are recognized as points and the last two as vectors.*

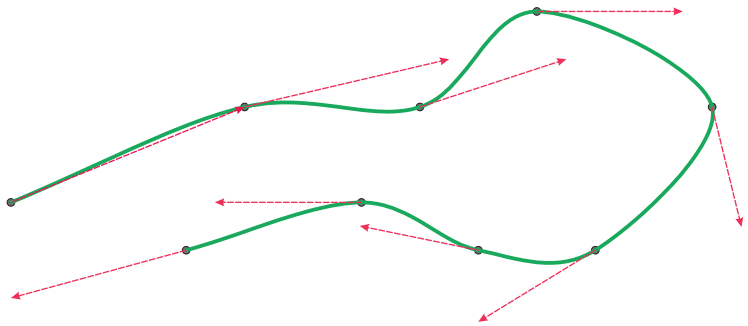


Figure 5.5: *A Catmull-Rom spline curve in green, that interpolates 9 points marked as grey circles. The tangent vectors appear as red arrows.*

5.5 Taylor expansions

This is a kind of Hermite interpolation, but was first introduced as a series by Brook Taylor in 1715. If we use a 3^{rd}-degree polynomial as an example and start with the the Newton polynomials (5.6), and then change the formula from interpolating four different points to interpolating the same point four times, ie

$$\begin{aligned} f(t) = f[t_0] &+ (t-t_0)\ f[t_0,t_0] \\ &+ (t-t_0)(t-t_0)\ f[t_0,t_0,t_0] \\ &+ (t-t_0)(t-t_0)(t-t_0)\ f[t_0,t_0,t_0,t_0]. \end{aligned}$$

If we replace the divided differences according to (5.3) we get

$$f(t) = f(t_0) + (t-t_0)\ f'(t_0) + (t-t_0)^2 \frac{f''(t_0)}{2} + (t-t_0)^3 \frac{f'''(t_0)}{6}.$$

The general expression for Taylor expansions is thus,

$$f(t) = \sum_{n=0}^{\infty} \frac{f^{(n)}(t_0)}{n!}(t-t_0)^n.$$

5.6 Hermite spline

The classical Hermite interpolation method using 5.11, from the reorganization of 5.6 together with 5.3, is suitable to interpolate several points together with derivatives of subsequent orders located at these points. The result is therefore a polynomial of very high degree. This has many drawbacks, so there is another way to do it which is more practical, namely using pieces of several polynomial-based curves that are glued together at their endpoints by using common points and derivatives of subsequent orders. This was earlier called "osculatory interpolation"[3] but nowadays it is mostly called Hermite splines. A Hermite spline is a set of pieces of polynomial curves of degree up to d (odd number) that is $C^{\frac{d-1}{2}}$-smooth on its domain. Cubic Hermite splines are mostly used, that is

[3]Repeated interpolation at a point was called "osculatory interpolation" since it produces higher than first order contact between the function and its interpolant ("osculari" means "to kiss" in Latin), p.7 in [37].

n polynomial pieces of curves, $\{c_i(t)\}_{i=0}^{n-1}$, where each curve piece is defined as in (5.11), (5.12) and (5.13). It follows that the spline is uniquely defined by $n+1$ knot values $\{t_i\}_{i=0}^{n}$, $n+1$ points $\{f(t_i)\}_{i=0}^{n}$ and $n+1$ vectors $\{f'(t_i)\}_{i=0}^{n}$, where $n > 0$.

Sometimes the tangents (derivatives) are not present, Therefor, several strategies for making smooth curves without given derivatives are developed:
♦ A cardinal spline[4] is a cubic Hermite spline whose tangents are defined by the points and a tension parameter. This spline creates a curve from one point to another taking into account the points before and after. By taking into account the points before and after the current curve, the curves appear to join together making one seamless curve. I.e. given n+1 points $p_0, ..., p_n$, to be interpolated with n cubic Hermite curve segments, for each curve we have a starting point p_i and an ending point p_{i+1} with starting tangent v_i and ending tangent v_{i+1} with the tangents defined by:

$$v_i = \frac{t_{i+1} - t_{i-1}}{2} (1-c) (p_{i+1} - p_{i-1})$$

where the first and last tangent v_0 and v_n are given (or one can use the endpoints only without dividing by 2) and c is a constant that modifies the length of the tangent (the tension parameter). The tension parameter c should be between 0 and 1.
♦ A Catmull-Rom spline (see [20]) is a cardinal spline with the tension parameter $c = 0$. Figure 5.5 shows an example of a Catmull-Rom spline curve made by 9 points.
♦ Another spline related to cubic Hermite spline is "cubic Bessel spline" (see [37]). Here one chooses the vectors (tangents) as the derivative at t_i of the polynomial of degree 2 which agrees with f at t_{i-1}, t_i, t_{i+1}. A short calculation gives the tangents

$$v_i = \frac{\Delta t_i \; f[t_{i-1}, t_i] + \Delta t_{i-1} \; f[t_i, t_{i+1}]}{\Delta t_{i-1} + \Delta t_i},$$

which shows that Bessel interpolation is also a local method.
♦ Yet another local method is Akima's interpolation method from 1970 (see [2]). Akima chose to compute the tangents by,

$$v_i = \frac{w_{i+1} \; f[t_{i-1}, t_i] + w_{i-1} \; f[t_i, t_{i+1}]}{w_{i-1} + w_{i+1}}.$$

where

$$w_j = \left| f\left[t_j, t_{j+1}\right] - f\left[t_{j-1}, t_j\right] \right|.$$

5.7 Cubic spline interpolation

There is, however, an other "classical" interpolation method, using piecewise polynomials, which has a higher degree of continuity, namely cubic spline interpolation (see [37]). This method is not local, so changes do not only have a local affect. The computational cost is, however, only slightly higher then the local methods because the matrix to be solved is a very narrow band matrix with only 3 diagonal elements.

[4] Schoenberg (see [140]) defines Cardinal splines as the class of polynomial splines on $\mathbb{R}$ with an infinite knot vector of simple integer knots and where all knot interval are 1.

So, if we want a C^2-smooth function we still can look at the cubic Hermite interpolation, but we now refrain from specifying derivatives at the internal knots. Instead we specify that:

$$c''_{i-1}(t_i) = c''_i(t_i) \quad \text{for} \quad i = 1,2,...,n-1, \tag{5.14}$$

and use these to solve a system of linear equations, i.e. involving an $(n-1 \times n-1)$ matrix, to find the derivatives in the internal knots,

$$c'_i(t_i), \quad i = 1,2,...,n-1.$$

To find the equations for cubic spline interpolation we just take (5.10) and compute the second derivative. Then we use the condition (5.14) in the internal knots.

The system of equations follows,

$$\mathbf{Ax} = \mathbf{b}, \tag{5.15}$$

i.e.

$$\begin{pmatrix} 2 & 1-\lambda_1 & 0 & \cdots & 0 \\ \lambda_2 & 2 & 1-\lambda_2 & \ddots & \vdots \\ 0 & \lambda_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1-\lambda_{n-2} \\ 0 & \cdots & 0 & \lambda_{n-1} & 2 \end{pmatrix} \begin{pmatrix} c'_1(t_1) \\ \vdots \\ \vdots \\ \vdots \\ c'_{n-1}(t_{n-1}) \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_{n-1} \end{pmatrix}.$$

Here is

$$\lambda_i = \frac{\Delta t_i}{\Delta t_{i-1} + \Delta t_i}, \quad \text{and} \quad b_i = 3\beta_i - \delta_i, \tag{5.16}$$

where

$$\beta_i = \lambda_i \, f[t_{i-1}, t_i] + (1-\lambda_i) \, f[t_i, t_{i+1}], \tag{5.17}$$

and

$$\delta_i = \begin{cases} \lambda_1 \, c'_0(t_0), & i = 1, \\ 0, & 1 < i < n-1, \\ (1-\lambda_{n-1}) \, c'_{n-1}(t_n), & i = n-1. \end{cases} \tag{5.18}$$

In (5.18) we can see that we need to know (or be able to estimate) the tangents at the start and end of the spline. This freedom leads to several types of boundary conditions. In [37] several boundary condition are listed.

Another way of organizing the equations that leads to another type of boundary conditions is to restrict the 2nd derivatives at the ends. The equation is the same as in (5.15), but the matrix dimension will now be $n+1 \times n+1$. The start and end tangents, and 2nd derivatives, are now included in a new first and last line of the matrix (computed from a

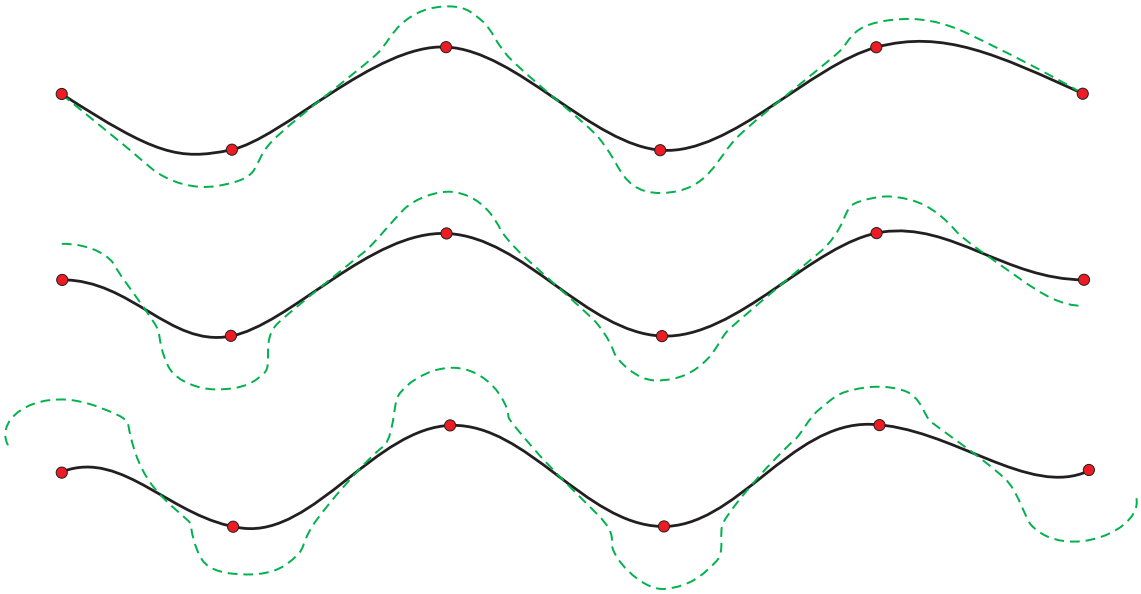


Figure 5.6: *Three cubic spline curves in solid black, interpolating the same six points marked in blue. The dotted green lines mark the curvature of the curves as an offset based on the direction of the 2nd derivative. Free end condition is used in the upper example, that is, 2nd derivatives are zero at start and end, this can be clearly seen in the figure. Horizontal tangent vectors used as end conditions are given in the middle example. Tangent vectors that are both* 45° *up to the right used as end conditions are given in the lower example.*

second derivative of (5.10). We now get

$$\begin{pmatrix} 2 & 1 & 0 & \cdots & \cdots & 0 \\ \lambda_1 & 2 & 1-\lambda_1 & \ddots & \ddots & \vdots \\ 0 & \lambda_2 & 2 & 1-\lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \lambda_{n-1} & 2 & 1-\lambda_{n-1} \\ 0 & \cdots & \cdots & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} c_0'(t_0) \\ \vdots \\ \vdots \\ \vdots \\ c_{n-1}'(t_{n-1}) \\ c_{n-1}'(t_n) \end{pmatrix} = \begin{pmatrix} b_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ b_n \end{pmatrix}.$$

Here λ is defined in (5.16) and β is defined in (5.17), but now

$$b_i = \begin{cases} 3f[t_0,t_1] - \frac{\Delta t_0}{2} c_0''(t_0), & i = 0, \\ 3\beta_i, & 0 < i < n, \\ 3f[t_{n-1},t_n] - \frac{\Delta t_{n-1}}{2} c_{n-1}''(t_n), & i = n. \end{cases} \tag{5.19}$$

If, in (5.19), $c_0''(t_0) = c_{n-1}''(t_n) = 0$ (or a zero vector if vector valued) we have so called *free ends*, and the spline is a "natural spline" with a behavior quite like a real spline device.[5]

[5]To draw curves, especially for shipbuilding, draftsmen often used long, thin, flexible strips of wood,

In Figure 5.6 is one example of cubic spline interpolation with free ends (the upper one). As we can see, the curvature zero is at the ends. In the other two examples end conditions are used with given tangents at both the start and end. The fact that the energy (or actually the approximation and simplification by using $c''(t)$ instead of the curvature) is minimized by a third degree spline function (in the Sobolev space $H^2(a,b)$), i.e. $\min \int_a^b |c''(t)|^2 dt$, is in a way illustrated in the figure. We can clearly see from the plots in Figure 5.6 that the free end condition has "the smallest curvature".

According to Schumaker [141] it was noticed already in the mid-1700s by Euler and the Bernoulli brothers that the shape of a spline device is approximately given by pieces of functions that are 3^{rd} degree polynomials.

5.8 Circle Splines

In 1996 Hans-Jörg Wenz published [163] an interpolation method based on blending circular arcs. Given a sequence of points $\{p_i\}_{i=0}^n$. In each internal point p_i, $i = 1,2,\ldots,n-1$ there is a circular arc $\vartheta_i(u)$, $u \in [0,1]$ starting in p_{i-1}, interpolating p_i and ending in p_{i+1} but parameterized from 0 to 1 in both segments, $[p_{i-1}, p_i]$ and $[p_i, p_{i+1}]$. To make a smooth curve segment between two point p_i and p_{i+1}, $i = 0,1,\ldots,n-1$, Wench used a simple blending scheme,

$$c_i(t) = \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} \vartheta_i(t) \\ \vartheta_{i+1}(t) \end{pmatrix}.$$

Note that, despite that they used a linear blending, the curve is G^1-smooth because every arcs interpolates three points and the expanded Hermite interpolation property, that will be explained in theorem 7.2 (on page 119) tells us that this raise the continuity by one.

Wenz was followed by Márta Szilvási-Nagy and Teréz P. Vendel, [153]. They improved the blending scheme, by replacing the linear interpolation function with a trigonometrically weighted blending function that also sums up to 1 and giving a G^2-smooth curve,

$$c_i(t) = \begin{pmatrix} \cos^2\left(\frac{\pi t}{2}\right) & \sin^2\left(\frac{\pi t}{2}\right) \end{pmatrix} \begin{pmatrix} \vartheta_i(t) \\ \vartheta_{i+1}(t) \end{pmatrix}.$$

Figure 5.7 shows an example of constructing a circle spline curve by blending two circular arcs using a trigonometrically weighted blending function.

To avoid loops and cusps Carlo Séquin, Jane Yen Kiha Lee introduced another blending technic (see [146, 145]). They used the trigonometrically weighted blending function to compute the directional angle $\tau(u)$ from τ_i and τ_{i+1},

$$\tau(u) = \begin{pmatrix} \cos^2\left(\frac{\pi u}{2}\right) & \sin^2\left(\frac{\pi u}{2}\right) \end{pmatrix} \begin{pmatrix} \tau_i \\ \tau_{i+1} \end{pmatrix}.$$

plastic, or metal called a spline. The strips were held in place with lead weights (called ducks because of their duck line shape). The elasticity of the material combined with the constraint of the control points, or knots, would cause the strip to take the shape which minimizes the energy required for bending it between the fixed points, and thus adopt "the smoothest possible" shape (according to elasticity theory).

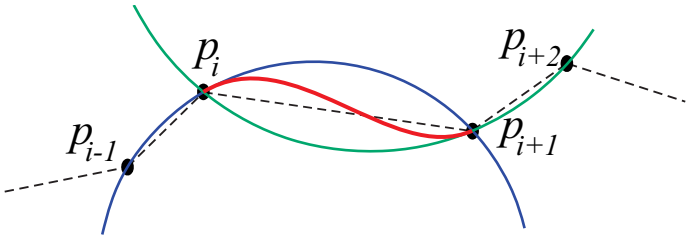


Figure 5.7: *Example of Circle splines. Between the point p_i and p_{i+1} are two circular arcs blended to one curve. As we can see, the curve segment between the points p_i and p_{i+1} is only defined by the four points: p_{i-1}, p_i, p_{i+1} and p_{i+2}.*

The angle τ_i is the angle between the line segment $p_{i+1} - p_i$ and the tangent $\vartheta_i'(0)$. They then compute the position on the curve $c_i(u)$ by computing the angle between the line segment $p_{i+1} - p_i$ and the line segment $c_i(u) - p_i$,

$$\phi(u) = (1-u)\tau(u),$$

and the distance $|c_i(u) - p_i|$ by

$$|c_i(u) - p_i| = \frac{\sin(u\ \tau(u))}{\sin(\tau(u))}|p_{i+1} - p_i|.$$

Chapter 6

B-spline Curves

Today, spline curves are synonymous with B-spline curves. B-splines are the de facto industrial standard for modelling in Computer Aided design. B-splines is a way of representing polynomial splines, it connect polynomial splines to corner cutting techniques, which induces many useful properties. In this chapter we will go deeper into polynomial B-splines, but we start with an overview to give an idea about what B-splines are.

B-splines is basically a set of "local" basis functions that together sums up to 1 over the entire domain. Each basis function is connected to one point. The formula for a curve is,

$$c(t) = \sum_{i=0}^{n-1} c_i \, b_{d,i}(t), \quad \text{where} \quad \sum_{i=0}^{n-1} b_{d,i}(t) = 1, \quad \text{for} \quad t_d \leq t \leq t_n,$$

and where c_i are n points that together form a control polygon, and $b_{d,i}(t)$ are the basis functions, d is the polynomial degree and $\tau = \{t_i\}_{i=0}^{n+d}$ is a vector of knot values. The basis functions together form a spline space, and is defined by a knot vector and a polynomial degree. The domain is restricted to $[t_d, t_n]$ because it is where the basis functions sums up to 1, and an important property is the C^{d-1}-continuity over non multiple knots.

An example, given a vector of knot values $\tau = \{0,0,0,0,1,2,3,4,5,6,7,7,7,7\}$. Together with a degree $d = 3$ we get 10 basis functions shown in Figure 6.1. In the figure we see that there are 7 intervals between the knots. In each interval there are 4 active basis functions. We now introduce a set of 10 points p_i, $i = 0,1,\ldots,9$. The result is shown in

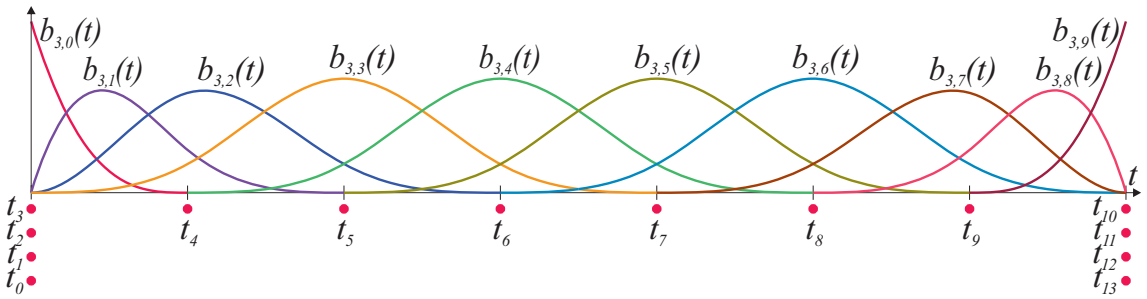


Figure 6.1: *A set of 10 3^{rd}-degree B-spline (basis) functions $b_{3,i}(t)$, $i = 0,1,\ldots,9$, defined by a vector of 14 knot values, marked with red bullets.*

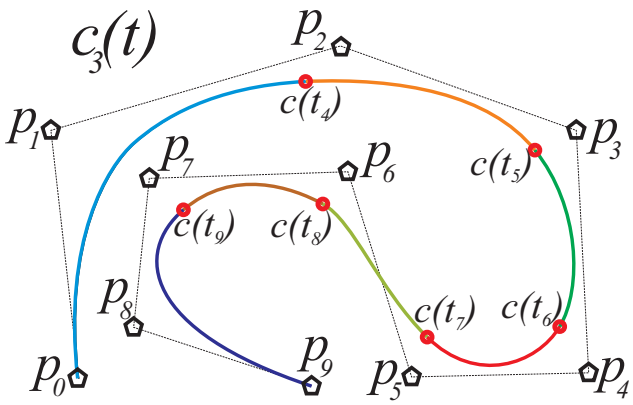


Figure 6.2: *We see a 3^{rd}-degree B-spline curve where the 7 parts have different colors. Control points and polygon are marked. The red dots mark position of knot values.*

Figure 6.2. We see the ten point p_i marked with black pentagons, we see the dotted black control polygon, the total C^2-smooth spline curve $c_3(t)$, and each piece of the curve in different colors. The red marks is where the the parameter value is equal the "internal" knot values t_i, $i = 4,5,\ldots,9$. Each of the 7 pieces is a polynomial curve and we can find the formula in the Bézier format. Because of the local support of the basis functions, we see that if we move p_2, only the blue, orange and green curve will be affected.

6.1 History of B-splines

We now go back more than a century. The need for smoother interpolants in some applications led in the late 1800s to the development of interpolation formulae based on piecewise polynomials, so-called *osculatory interpolation techniques*, most of which appeared in the actuarial literature, Greville [81]). An example of this is the formula proposed in 1899 by Karup [96]. The formula results in a piecewise 3^{rd}-degree polynomial interpolant that is continuous and continuously differentiable everywhere. By using this formula, it is possible to reproduce polynomials up to 2^{nd} degree. Another example is the formula proposed in 1906 by Henderson [89], which also yields a continuously differentiable, piecewise 3^{rd}-degree polynomial interpolant, and is capable of reproducing polynomials up to 3^{rd}-degree. A third example is the formula published in 1927 by Jenkins [93]. The resulting composite curve is a piecewise 3^{rd}-degree polynomial that is twice continuously differentiable. However, this curve is not an interpolant. (All these formulaes can be found in [81], and [124].)

The need for practically applicable methods for interpolation or smoothing of empirical data was the driving force for Schoenberg's first study of the subject. In his 1946 papers [137, 138] he noted that for every *osculatory interpolation* formula applied to equidistant data, where he assumed the distance to be unity, there exists an even function $\Phi : \mathbb{R} \to \mathbb{R}$, in terms of which the formula may be written as

$$f(x) = \sum_{j=-\infty}^{\infty} a_j \Phi(x-j), \tag{6.1}$$

where Φ, which he termed as the *basic function*, completely determines the properties of the resulting interpolant and reveals itself when applying the initial formula to the impulse sequence defined by $a_0 = 1$ and $a_k = 0,\ \forall k \neq 0$. By analogy with Whittaker's cardinal series [164], Schoenberg referred to the general expression (6.1) as a formula of the cardinal type, but noted that the basic function of the cardinal series, $\Phi(x) = sin(\pi x)/(\pi x)$, is inadequate for numerical purposes due to its excessively low damping rate. The basic functions involved in Waring-Lagrange interpolation, on the other hand, possess the limiting property of being at most continuous, but not continuously differentiable. He then pointed at the smooth curves obtained by the use of a mechanical spline,[1] argued that these are piecewise cubic arcs with a continuous 1^{st}- and 2^{nd}-order derivative, and then continued to introduce the notion of the mathematical spline:

Definition 6.1. *A real function f defined for all real x is called a spline function (curve) of order k and denoted by S_k if it have the following properties:*

i) it is composed of polynomial arcs of degree at most $d = k-1$,
ii) it is of class C^{k-2}, i.e., f has $k-2$ continuous derivatives,
iii) the only possible function points (knot values) of the various polynomial arcs are the values $x = k$ if k is even, or else $x = k + \frac{1}{2}$ if k is odd.

Note that these requirements are satisfied by the curves resulting from the aforementioned smoothing formula proposed by Jenkins, and also studied by Schoenberg [137], which constitutes one of the earliest examples of a spline generating formula. Note also that Hermite spline and related splines do not fulfill all the requirements.

After having defined the spline curve, Schoenberg continued to prove that any spline curve S_k may be represented uniquely in the form

$$S_k(x) = \sum_{j=-\infty}^{\infty} a_j M_k(x-j) \tag{6.2}$$

for appropriate values of the coefficients a_j. There are no convergence difficulties since, as we will se below, $M_k(x)$ vanishes for $|x| > k/2$. Thus (6.2) represents an S_k for arbitrary $\{a_j\}$ and represents the most general one.

Here, $M_k : \mathbb{R} \to \mathbb{R}$ denotes the so-called central *B-spline* of degree $d = k-1$, which Schoenberg defines as the inverse Fourier integral

$$M_k(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \left(\frac{\sin(\omega/2)}{\omega/2} \right)^k e^{i\omega x} d\omega, \tag{6.3}$$

and, when evaluating explicitly as[2]

[1] The use of the word "spline" for a flexible rod can be traced back to shipbuilding in the 18th century. It was originally an East Anglian dialect word. Flexible rods (spline devices) have been used by shipbuilders all over the world for a very long time. In Norway there are actually 3 different names for them, depending on the location of the shipyard. In the north (where this author has his experience) it is called "rei", pronounced and related to ray, as in "ray-tracing".

[2] Scoenberg noted that (6.3) had been evaluated explicitly for low values of k by various authors, i.e. S. Bochner in lectures of Fourier analysis in 1936. In an article by Butzer, Schmidt and Stark [18] sev-

$$M_k(x) = \frac{1}{(k-1)!}\delta^k x_+^{k-1},$$

where δ^p is the pth-order central difference operator [3] and x_+^n denotes the one-sided power function defined as

$$x_+^n = \begin{cases} x^n, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0. \end{cases}$$

Soon after this first paper [137, 138] was written, the possibilities of an extension to arbitrary spaced knots was implied by Curry in his review [31] of Schoenberg's paper. And in an abstract, [32], B-splines[4] for arbitrary spaced knots were introduced. The whole article was written in 1945-47 but was actually published 20 years later [33]. Curry and Schoenberg's definition of B-splines is as follows,

Definition 6.2. *A B-spline denoted $M_k(x)$ is defined by $k+1$ increasing real values $x_0,\ldots,x_k$, where $x_0 < x_k$, its explicit expression is*

$$M_k(x;x_0,\ldots,x_k) = k\sum_{\nu=0}^{k}\frac{(x_\nu - x)_+^{k-1}}{\omega'(x_\nu)} \tag{6.4}$$

where

$$\omega(x) = (x-x_0)\cdots(x-x_n) \quad \textit{see (5.2).} \tag{6.5}$$

Curry and Schoenberg stated the following properties for the B-splines:

1) They are shown to be bell-shaped.
2) They are also shown to be the projections onto the x-axis of the volumes of appropriate n-dimensional simplices.
3) This geometric interpretation allows us to conclude, by means of Brunn's theorem, that the B-spline function is logically concave.
4) They are also shown to form a basis for all spline functions of degree $n-1$ and given knots.
5) They can be defined on multiple knots.
6) The B-splines $M_k(x)$ (defined above) are frequency functions, which means that they are non-negative and their integral over $\mathbb{R}$ is 1.[5]

---

eral predecessor in B-splines are mentioned that where probably not known by Scoenberg. Among these are Maurer in 1896 [120] (discussing a function related to the central B-spline function of Schoenberg), Sommerfeld in 1904 and 1929 [149] (making a geometric interpretation, a Box-spline like method for constructing B-splines) and Brun in 1932 [17] (developing a recursive corner cutting method like de Casteljau).

[3] Often called Sheppard's central-difference operator δ (see [147]), defined by

$$\delta\ \phi(\xi) = \phi(\xi+\frac{1}{2}) - \phi(\xi-\frac{1}{2})$$

and

$$\delta^p\phi(\xi) = \delta^{p-1}\phi(\xi+\frac{1}{2}) - \delta^{p-1}\phi(\xi-\frac{1}{2}), \quad p>1,$$

for any function $\phi : \mathbb{R}\to\mathbb{R}$ at any ξ. Not to be mixed up with the right-sided difference operator.

[4] Curry and Schoenberg actually first called them the fundamental spline functions. The name B-spline refers to basis splines because they form a basis for spline functions, which was proved by Curry and Schoenberg in [32]. In [139], published in 1967, Schoenberg used the name B-spline.

[5] NB! Modern B-splines are scaled in a different way because in CAGD partition of unity is a more important property.

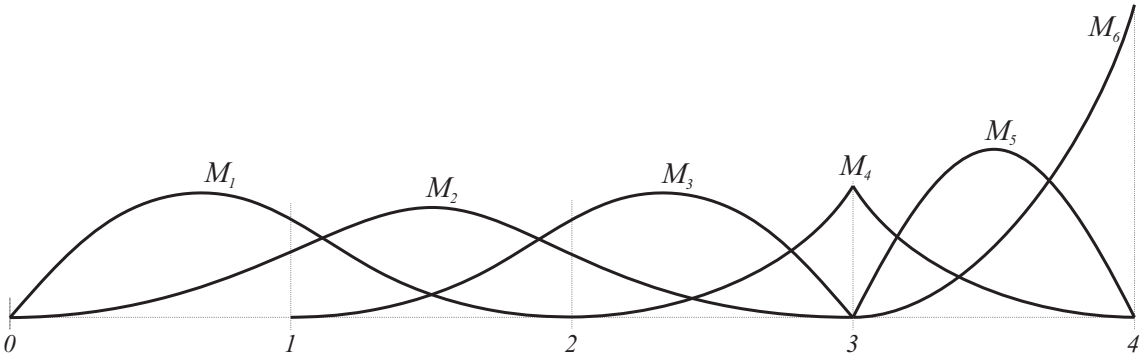


Figure 6.3: *A copy of a sketch from Curry and Schoenberg [33] of six quaddratic B-splines defined by 9 knots;* $x_1 = x_2 = 0$, $x_3 = 1$, $x_4 = 2$, $x_5 = x_6 = 3$ *and* $x_7 = x_8 = x_9 = 4$.

In Figure 6.3 there is a copy of a sketch from Curry and Schoenberg [33] of the six quaddratic B-splines $(k = 3)$ for the case when there are 9 knots, and where some of them are equal. The six B-spline functions are:

$$M_1(x) = M_3(x;0,0,1,2), \quad M_2(x) = M_3(x;0,1,2,3), \quad M_3(x) = M_3(x;1,2,3,3),$$
$$M_4(x) = M_3(x;2,3,3,4), \quad M_5(x) = M_3(x;3,3,4,4), \quad M_6(x) = M_3(x;3,4,4,4).$$

The classical expanded definition of divided differences (5.1), states the following

$$[y_o, y_1, \ldots, y_n] = \sum_{i=0}^{n} \frac{y_i}{\omega'(x_i)}, \tag{6.6}$$

where $\omega(x)$ is defined in (5.2), and is the same as in (6.5).

Expression (6.6) is in a way equal to (6.4), but because there is the function $(x_\nu - x)_+^{k-1}$ instead of scalars involved in (6.4) we have to clarify the notation. A divided difference version of a B-spline $M_k(x)$ is, therefore,

$$M_k(x;x_0,\ldots,x_k) = k\,(\cdot - x)_+^{k-1}\,[x_0,\ldots,x_k]\,. \tag{6.7}$$

The notation is called a "placeholder" notation and a more detailed description of this notation can be found on page 108 in [37].

Divided differences are defined recursive, so it is natural to try to develop a simple recursive algorithm for computing B-splines. This was done in 1972 by at least three different authors simultaneously. Cox [29] established it for simple knots. The result for general knots is credited to deBoor [36]. In his paper he mentioned that Louis Mansfield had also discovered the recursion. In the same paper deBoor also introduces the derivative formula for B-splines. Some other important algorithms in spline history are; knot insertion introduced simultaneously in 1980 by Boehm [15] for single knots and Cohen, Lyche and Riesenfield [23] for general knot insertion, and degree raising in 1985 by Cohen, Lyche and Schomaker [24]. It should of course be mentioned that Paul de Casteljau in 1959 developed an algorithm for the computation of Bézier curves (but it was only published in an internal Citroë report [38]) and that Pierre Étienne Bézier in 1966 published the construction of Bézier curves/surfaces [12, 13]. (Bézier curves are the simplest B-spline curves)

As a final historical remark in this section it can be mentioned that according to Farin [64] N. Lobachevsky was as early as the nineteenth century investigating B-splines as convolutions of certain probability distributions (over a very special knot sequence). Also Peano Kernel for 3^{rd} difference gives a second degree B-spline, for which there is a plot on page 73 in the Davis book from 1963, [35]. The name spline is refereing to a mechanical spline device that is minimizing the energy while bending. This can be expressed as an equation minimizing the square of the curvature, i.e.

$$\min \int_{s=x_0}^{x_n} \kappa(s)^2 ds. \tag{6.8}$$

A cubic spline function satisfies the formula, but where the second derivative replaces the curvature. Of course, this is not the same as (6.8), but if the speed is close to 1 above the hole curve, this is a good approximation. There are works on better approximations, e.g Mehlum in [122, 123]. How to use the term spline is therefore not evident. It should possibly in some sense approximate a spline device, whether it is an optimal or not an optimal solution according to some functional. today it is most common to think about splines as pieces that are glued together to one curve or surface with controlled smoothness.

This leads to the modern normalized B-splines, it's notation and algorithms which are the contents of the next section.

6.2 Modern B-splines

Modern B-spline theory has several branches. There is blossoming method proposed by Ramshaw [133] and, in a different form by de Casteljau [41], he called it polar form. However in this section we will first look at B-splines in what we now can call the classical way, we will give some examples and finally we will look at B-spline algorithms. But in order not to get lost in indices, we will use matrix - vector formulations.

There are several different B-spline notations in use. The most common are,

$$b_{d,i}(t) = b(t;t_i,\ldots,t_{i+d+1}), \quad \text{or} \quad b_{k,i}(t) = b(t;t_i,\ldots,t_{i+k}).$$

where d is the degree, $k = d+1$ is the order (dimension of the function space). Here, B-splines are synonymous with normalized B-splines, which means that all B-splines defined on an infinite knot sequence sum up to 1 (form the partition of unity), i.e.

$$\sum_{i=j-k}^{j} b(t;t_i,\ldots,t_{i+k}) = 1, \quad t_j \le t < t_{j+1}. \tag{6.9}$$

Compared to the original Curry-Schoenberg B-splines $M(t;t_i,\ldots,t_{i+k})$ we thus have the following relationship (to compare visually see also Figure 6.4)

$$b(t;t_i,\ldots,t_{i+k}) = \frac{t_{i+k}-t_i}{k} M(t;t_i,\ldots,t_{i+k}).$$

Note that for integer non multiple knots i.e 1,2,... then $b(t;t_i,\ldots,t_{i+k}) = M(t;t_i,\ldots,t_{i+k})$.

This leads to the following definition of the B-spline that usually is called the Cox-de'Boor recursion formula, sometimes also called Mansfield-Cox-de'Boor:

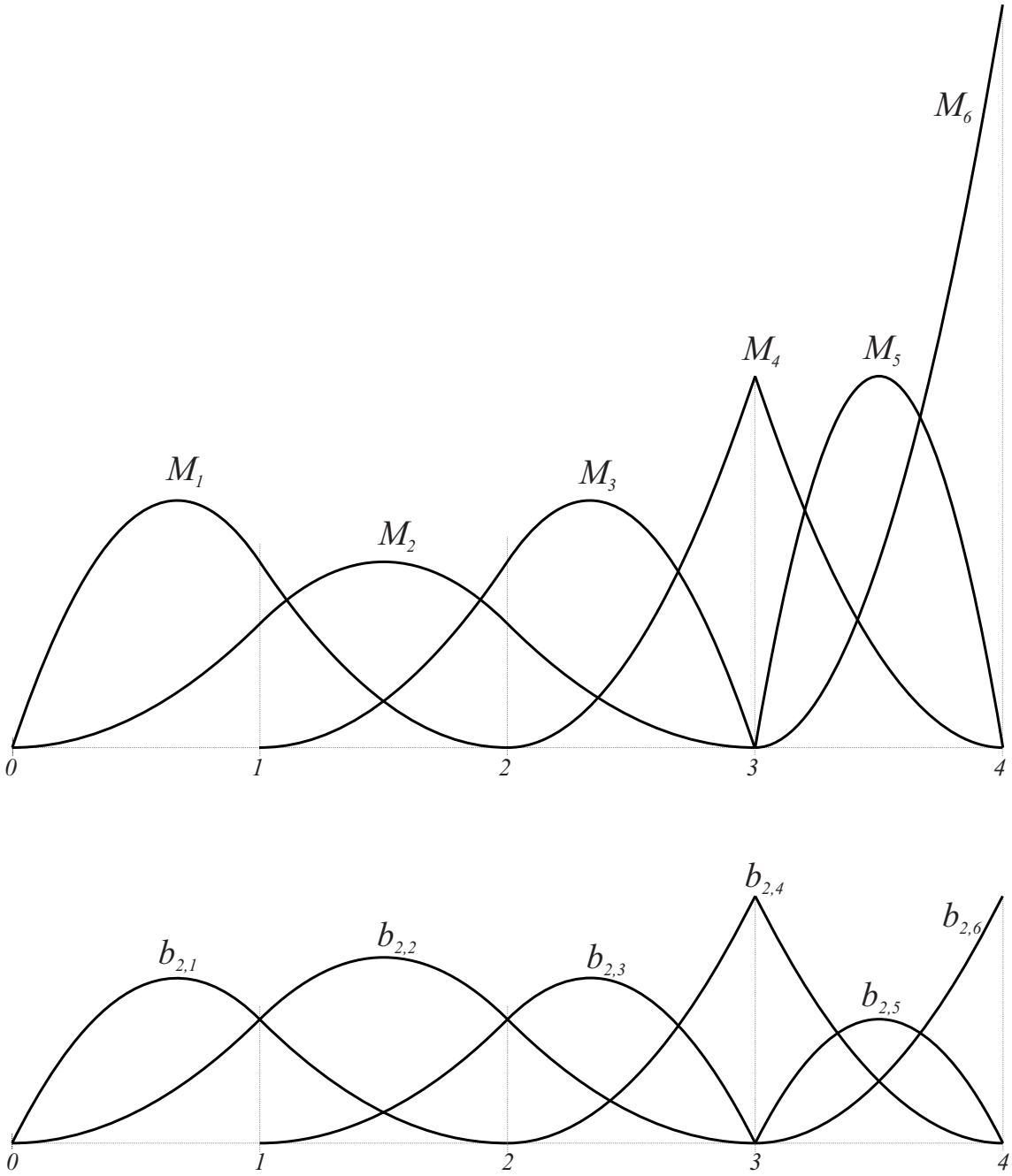


Figure 6.4: *Two plots of six 2^{nd}-degree B-splines that both are defined by the 9 knots used in Figure 6.3; $t_1 = t_2 = 0$, $t_3 = 1$, $t_4 = 2$, $t_5 = t_6 = 3$ and $t_7 = t_8 = t_9 = 4$. The upper ones are the original B-splines from Curry and Schoenberg [33], sketched in Figure 6.3 and where the integral of each basis function is 1, ie they are frequency functions. The lower ones are the modern B-splines that form the partition of unity, that is, they sums up to 1 wherever there is a complete set of basis functions, which means k basis functions different from zero where k is the order (the dimension of the function space). The scale is the same for both plots. Note that $M_2 = b_{2,2}$ because the knots are $\{0,1,2,3\}$, ie non multiple integer values.*

Definition 6.3. *For a polynomial degree d, and order $k = d+1$, and given $k+1$ increasing real numbers $\{t_i, t_{i+1}, \ldots, t_{i+k}\}$, where $t_{i+k} > t_i$ (they do not otherwise need to be strongly increasing), then a B-spline of degree d is defined by the recursion formula*

$$b(t;t_i,\ldots,t_{i+k}) = w_{d,i}(t)\, b(t;t_i,\ldots,t_{i+k-1}) + \big(1 - w_{d,i+1}(t)\big)\, b(t;t_{i+1},\ldots,t_{i+k}) \quad (6.10)$$

where the termination of the recursion is

$$b(t;t_j,t_{j+1}) = \begin{cases} 1, & t_j \le t < t_{j+1}, \\ 0, & \textit{otherwise}, \end{cases}$$

and where the linear translation and scaling function

$$w_{d,i}(t) = \frac{t - t_i}{t_{i+d} - t_i}. \quad (6.11)$$

A list of the basic properties of $b(t;t_i,\ldots,t_{i+k})$ follows.

P1. Every basis functions $b(t;t_i,\ldots,t_{i+k})$ is positive on (t_i,t_{i+k}) and zero otherwise.

P2. The set of basis functions $b(t;t_i,\ldots,t_{i+k})$ for $i = j-d,\ldots,j$ form a partition of unity on $[t_j,t_{j+1})$, i.e.

$$\sum_{i=j-d}^{j} b(t;t_i,\ldots,t_{i+k}) = 1, \quad t_j \le t < t_{j+1}.$$

P3. A B-spline $b(t;t_i,\ldots,t_{i+k})$ is $C^r(\mathbb{R})$ where $r = d-s$ and s is 'the number of maximum multiplicity of the knots $t_i,\ldots,t_{i+k}$', i.e. maximum number of equal knots.

P4. The B-spline $b(t;t_i,\ldots,t_{i+k})$ is, in case of simple knots "bell-shaped", i.e. the v^{th} derivative $b^{(v)}(t;t_i,\ldots,t_{i+k})$ (up to $v = d-1$) has exactly v zeros in the interval (t_i,t_{i+k}). If the knots are simple these zeros are distinct.

6.2.1 The knot vector

Knots have already been mentioned several times, first in section about Hermite spline 5.6 and Cubic spline interpolation 5.7 and also earlier in this chapter. The knot values actually describe the domain for each piece of polynomial functions. These functions are glued together at the knot values. This is clearly illustrated in Figure 6.2 and 6.7, where the red circles mark $c(t_i)$, ie where on the curve the knot values are, and thus where two curve parts are glued together.

The knot vector

is the set of knot values that together with a polynomial degree d defines the spline space, ie the set of B-splines (basis function). In [6.9] the requirement that the sum of the basic functions must be 1 is stated (the partition of the unity). It follows that there must be $d+1$ active basic functions at all knot intervals. That is for a given knot vector $\tau = \{t_0,t_1,\ldots,t_{n+d}\}$ is the domain of any B-spline function/curve of degree d based on this knot vector restricted to $[t_d,t_n]$.

A B-spline function / curve can have three states determined by the knot vector, open, clamped or closed/cyclic. These states will be described in more detail in the following.

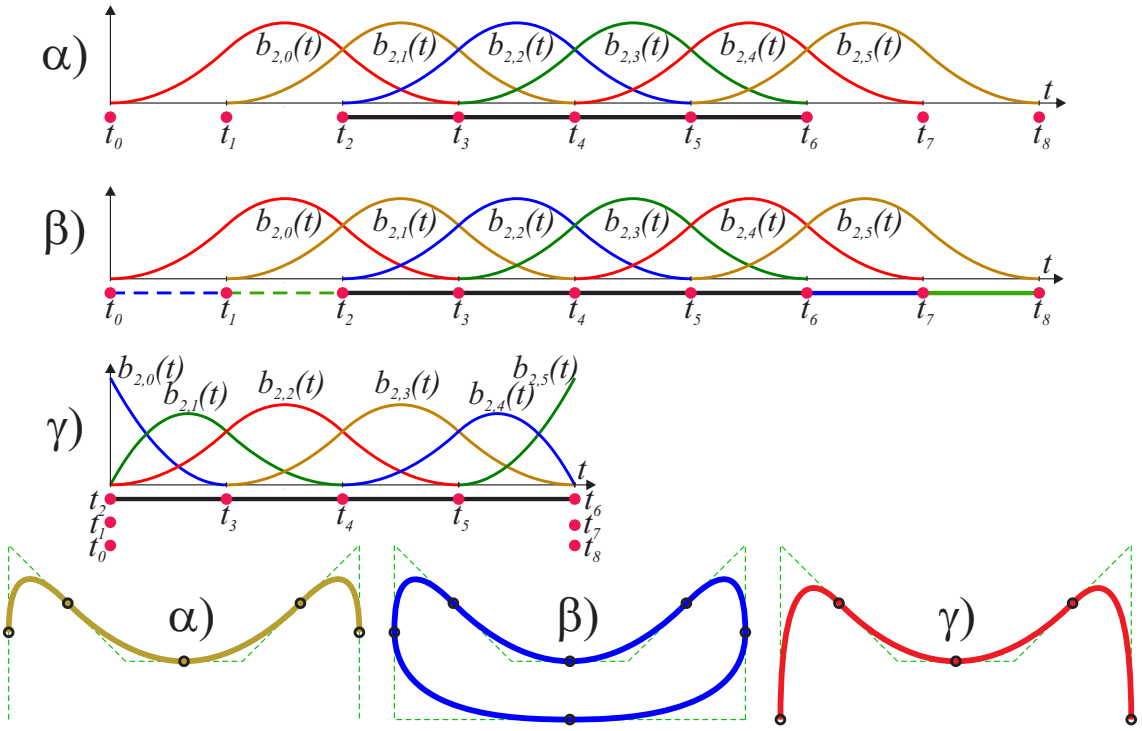


Figure 6.5: *Three B-spline curves are shown, α) is open, β) is closed and cyclic and γ) is open and clamped. Also the B-splines (basis functions and the knot vectors (red dots) are shown and the domains are marked with solid lines through the red knots.*

6.2.2 B-spline curves - Open, Clamped or Closed

B-spline curves share many properties with Bézier curves, because the former is a generalization of the later. In the following, we denote a B-spline curve $c(t)$. It is defined by a polynomial degree d, and thus order $k = d+1$, and by a knot vector $\tau = \{t_0, t_1, ..., t_{n+d}\}$ and n control points. The general formula is

$$c(t) = \sum_{i=0}^{n-1} c_i \, b_{k,i}(t), \quad \text{open / clamped - } \; t \in [t_d, \, t_n] \quad \text{closed - } \; t \in [t_d, \, t_{n+d}), \quad (6.12)$$

where $\{c_i\}_{i=0}^{n-1}$, is the vector of coefficients / control points. These control points define the control polygon of the curve. $b_{k,i}(t)$ is the set of B-spline basis functions defined by the knot vector τ, which has $n+k$ knot values.

As mentioned, a B-spline curve can have three states, open, open/clamped or closed/-cyclic. If a curve are constructed from n basis functions and thus n control points, then

Open - is a curve that is build on an ordinary set of knot values $\tau = \{t_0, t_1, ..., t_{n+d}\}$ where $t_{i+1} \geq t_i$ and where the domain of the curve is restricted to $[t_d, t_n]$.

Open/Clamped - is an open curve with a knot vector $\tau = \{t_0, t_1, ..., t_{n+d}\}$, but where the first k and last k knots are equal, i.e $t_0 = t_2 = ... = t_d$ and $t_n = t_{n+1} = ... = t_{n+d}$,

Closed/Cyclic - is a curve where the knot vector "bites itself in the tail", it has a knot vector $\tau = \{t_0, t_1, ..., t_{n+d}\}$, but the domain is extended to $[t_d, t_{n+d})$. In addition must $t_{n+i+1} - t_{n+i} = t_{i+1} - t_i, \quad i = 0, 1, \ldots, d-1$.

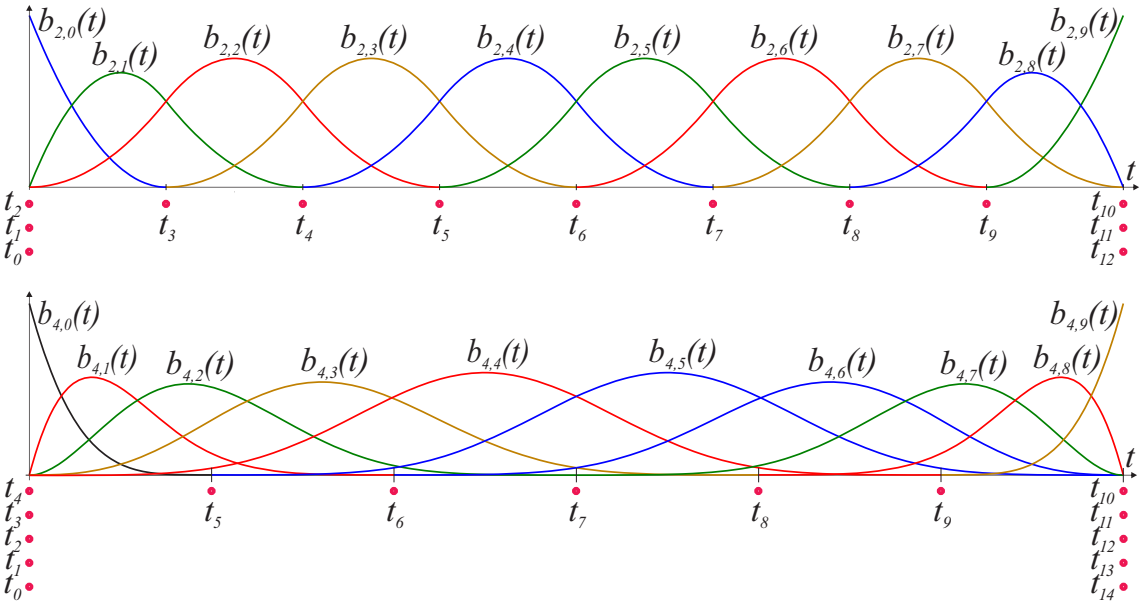


Figure 6.6: *At the top we see a set of 10 2^{nd}-degree B-spline (basis) functions $b_{2,i}(t)$, defined by a knot vector of 13 knots, marked with red bullets, and at the bottom we see a set of 10 4^{th}-degree B-spline (basis) functions $b_{4,i}(t)$, defined by a knot vector of 15 knots.*

In Figure 6.5, three 2^{nd}-degree B-spline curves are plotted together with their basis functions. They all are build on the same 6 control points. Curve α) is open and "not clamped", β) is closed/cyclic and γ) is open and clamped.

An open curve can be clamped or not clamped. An open not clamped B-spline curve is shown as curve α) in Figure 6.5. On the plot of the basis function the domain is marked and is $[t_2, t_6]$. In the plot of the curve we can see that it starts and ends at a knot value between two control points.

A clamped B-spline curve starts in the first coefficient, then goes out in the same direction as the control polygon. It ends in the last coefficient and enters in the same direction as the control polygon. An example is shown in curve γ) in Figure 6.5. The knot vector is clamped to the domain of the curve and the curve start and ends at control points. The 1^{st}-derivatives at the start end point also follows the control polygon. **This type of B-spline curves is the most common, and in general, if nothing is mentioned we are dealing with this type of curves.**

A closed/cyclic B-spline curve is shown as curve β) in Figure 6.5. As we see for the open curves α) and γ), there are 4 knot interval that form the domains of the curves, but for the closed/cyclic curve there are 6. This can clearly be seen in the plots of the curves at the bottom of Figure 6.5, ie curve β). We can also see that the open (not clamped) curve is equal the closed curve at the same 4 knot intervals. The two additional knot intervals are the parameter domain of the parts that is closing the curve. It follows that the domain is the half open interval $[t_2, t_8)$. However, the restriction on the knot vector is that the last two knot intervals are equal to the first two, ie $t_1 - t_0 = t_7 - t_6$ and $t_2 - t_1 = t_8 - t_7$ in the example in curve β) in Figure 6.5.

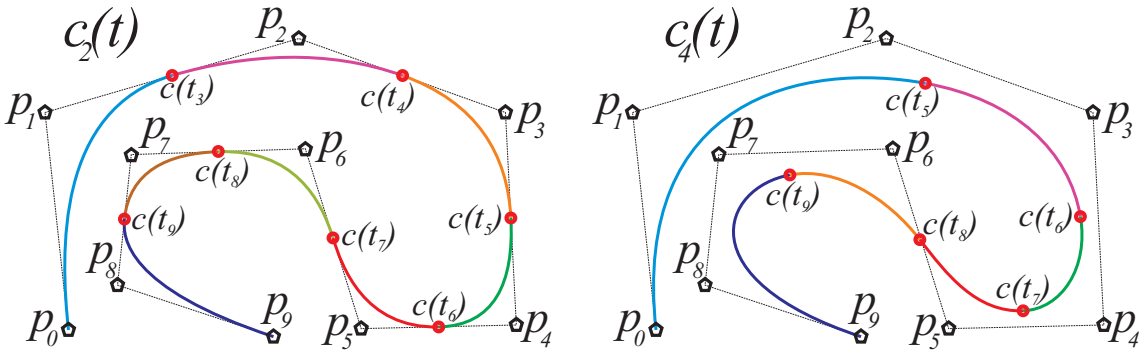


Figure 6.7: *A 2^{nd}-degree B-spline curves $c_2(t)$ and a 4^{th}-degree B-spline curve $c_4(t)$ based on the same control points. The red dots mark position of internal knot values.*

In Figure 6.1, a set of 3^{rd}-degree B-spline basis functions based on a knot vector $\tau = \{0,0,0,0,1,2,3,4,5,6,7,7,7,7\}$ is shown. The corresponding Figure 6.2 shows a B-spline curve based on these basis functions and a set of 10 points p_i, $i = 0,1,\ldots,9$.

From the same set of control points. We make a curve of polynomial degree 2 based on a knot vector $\{0,0,0,1,2,3,4,5,6,7,8,8,8\}$, and another curve with polynomial degree 4 and knot vector $\{0,0,0,0,0,1,2,3,4,5,6,6,6,6,6\}$. In Figure 6.6, we see the B-splines based on these two knot vectors. In Figure 6.7 is B-spline curves based on these two set of basis functions plotted. We see that the 2^{nd}-degree curve touches the control polygon due to that only two basis function are $\neq 0$ at the knot values (illustrated in Figure 6.6). Consequently the touch points are also the place where the pieces are glued together. In the 2^{nd}-degree curve we have 8 pieces, see Figure 6.7-left, while in the 3^{rd}-degree curve we have 7 pieces, Figure 6.2, and in the 4^{th}-degree curve 6 pieces, Figure 6.7-right.

A B-spline curve has many useful properties, some of which are listed below.

1. A B-spline curve $c(t)$ is a piecewise curve where each piece is a curve of degree at most d. Thus, the curve is the union of curve segments defined on each knot span.

2. A Clamped B-spline curve start at the first control point and end at the last control point. Further, the curve leave the first point tangential to the line between the first and the second control point and enter the last control point tangential to the line between the second last and the last control point.

3. B-spline curves has a strong ´´Convex Hull Property". That is, a B-spline curve is contained in the convex hull of its control points. More specifically, if $t \in [t_i,\ t_{i+1})$ then $c(t)$ is in the convex hull of the control points $c_{i-d}, c_{i-d+1}, ..., c_i$.

4. A B-spline curve has a local Modification scheme. That is, changing the position of the control point c_i only affects the curve $c(t)$ at the interval $[t_i, t_{i+k}]$.

5. A B-spline curve $c(t)$ is C^{d-j} continuous at a knot of multiplicity j. That is, if $t_i = t_{i+1} = ... = t_{i+j-1}$, then the curve is C^{d-j} continuous at $c(t_i)$.

6. A B-spline curve has a *variation diminishing property*. That is, in $\mathbb{R}^2$, no straight line intersects a B-spline curve more times than it intersects its control polygon.

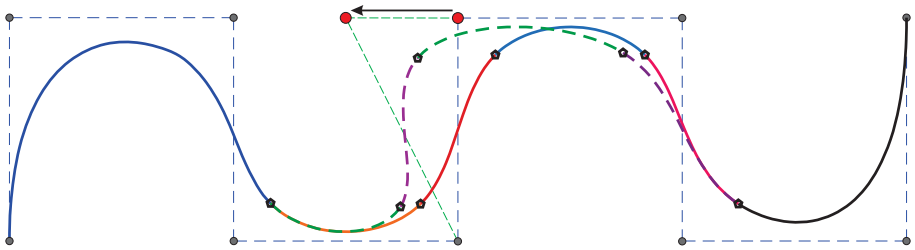


Figure 6.8: *A 3^{rd}-degree B-spline curve and its control polygon. One of the control points is moved to the left. To illustrate the local control, part of the curve that is changed is also plotted (dashed).*

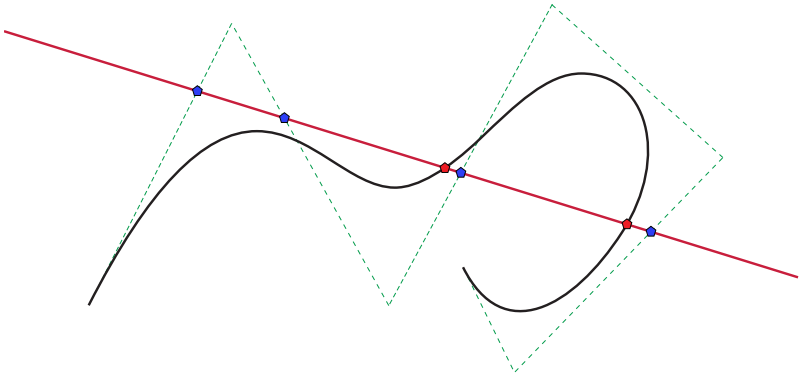


Figure 6.9: *A 3^{rd}-degree B-spline curve and its control polygon. A line is intersecting the curve 2 times and the control polygon 4 times.*

7. Bézier curves are special cases of B-spline curves. They are Clamped B-spline curves on the domain $[0, 1]$, and without internal knots (only knots at start and end).

8. The affine invariance property also applies to B-spline curves. If a B-spline curve is to be translated, rotated, scaled, ..., ie applied to an affine transform, see (2.4), this can be done by using the affine map only on the control points. This also applies for Bézier curves, section 4.4, and if the homogeneous matrix is used as an affine map, see (2.8), this also applies to Hermite curves, section 4.3.

9. The control polygon of a B-spline curve is converging towards the B-spline curve when the number of knots goes towards infinity. This will be clearly shown in the knot insertion algorithm shown in subsection 6.2.6. This is because the B-spline algorithm itself is basically corner cutting, see also section 6.7 about subdivision curves, especially Lane-Riesenfeld subdivision algorithm in subsection 6.7.3.

Figure 6.8 illustrate local modification. We see a 3^{rd}-degree B-spline curve, with different color at each interval. The knot vector that define the B-splines is as in Figure 6.1. 9 of the 10 control points are marked blue, while one is red. The red point is then moved to the left and we can see the part of the curve that is affected, ie 4 intervals because the degree is 3.

Figure 6.9 illustrate the variation diminishing property. A line intersects a 3^{rd}-degree B-spline curve 2 times, while it intersects the control polygon 4 times.

6.2.3 The B-spline factor matrix $T(t)$

In subsection 4.4.2, factor matrices for Bézier curve, $T_d(t)$, were defined. In (4.31) a 3^{rd}-degree Bézier curve is formulated in matrix notation. The algorithm for computing the Bernstein polynomials is described in subsection 4.4.1. A Comparison of the Bernstein recursion with the B-spline recursion, Definition 6.3, shows that they are equal, but where t in the Bernstein algorithm is replaced by the linear function $w_{d,i}(t)$, described in(6.11), in the B-spline algorithm. Recall that $w_{d,i}(t)$ maps $[t_i, t_{i+d}]$ to $[0,1]$.

A B-spline curve is polynomial-based curve pieces that are glued together at the knot values. Therefor, for a 2^{nd}-degree B-spline curve, for each piece with the domain $[t_i, t_{i+1})$, $i = d, d+1, \ldots, n$ we get the following formula (provided that $t_{i+1} > t_i$),

$$c(t) = \begin{pmatrix} 1-w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1-w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1-w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}.$$

We call these matrices for *factor matrices* because they actually factorize the B-splines over one knot interval. A general definition of a factor matrix follows.

Definition 6.4. *The B-spline factor matrix $T_d(t)$ is a $d \times (d+1)$ band limited matrix with two nonzero elements (based on $w_{d,i}(t)$ (6.11)) on each row. The matrix is as follows*

$$T_d(t) = \begin{pmatrix} 1-w_{d,i-d+1}(t) & w_{d,i-d+1}(t) & 0 & \cdots & \cdots & 0 \\ 0 & 1-w_{d,i-d+2}(t) & w_{d,i-d+2}(t) & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & 0 & 1-w_{d,i-1}(t) & w_{d,i-1}(t) & 0 \\ 0 & \cdots & \cdots & 0 & 1-w_{d,i}(t) & w_{d,i}(t) \end{pmatrix}$$

Note that this matrix $T_d(t)$, is for the knot interval $[t_i, t_{i+1})$.

We observe in the matrix $T_d(t)$ that the last index of w is decreased by 1 from one line to the next line. For the bottom line it is denoted i and for the line above $i-1$ and so on.

For $d = 3$ we have the following formula for a B-spline curve on $[t_i, t_{i+1})$,

$$c(t) = \begin{pmatrix} 1-w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1-w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1-w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \begin{pmatrix} 1-w_{3,i-2}(t) & w_{3,i-2}(t) & 0 & 0 \\ 0 & 1-w_{3,i-1}(t) & w_{3,i-1}(t) & 0 \\ 0 & 0 & 1-w_{3,i}(t) & w_{3,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix},$$

where the index i is determined by $t_i \le t < t_{i+1}$. Note that the indices of the $w(t)$ at the bottom of all the matrices $T_d(t)$ are (d,i). As for the Bézier curves (4.32), an expression for a 3^{rd}-degree B-spline curve is

$$c(t) = T_1(t)T_2(t)T_3(t)\,\mathbf{c} = T^3(t)\,\mathbf{c},$$

where $T^3(t)$ is a vector of 4 B-splines of degree 3 on the interval $[t_i, t_{i+1})$.

Theorem 6.1. *The matrices in Definition 6.4 ($T_q(t),\ q = 1,2,\ldots,d$) are together the factorisation of a set of B-splines on a knot interval, and they are the formula of the Algorithm described in Definition 6.36, the recursive B-spline algorithm.*

Proof. Recall that $T_1(t) = (1 - w_{1,i}(t) \quad w_{1,i}(t)) = (b(t;t_{i-1},t_i,t_{i+1}) \quad b(t;t_i,t_{i+1},t_{i+2}))$. We also see that $T^1(t) = T_1(t)$. We now assume that $T^{d-1}(t)$ is a vector of d B-spline functions of degree $d-1$ on the interval $[t_i,t_{i+1}]$. Computing this vector with each of the columns of $T_d(t)$ gives each of the B-splines of degree d and is exactly the same as expression (6.10) describes in Definition 6.3. □

To investigate the derivative of $T(t)$ we first look at the derivative of the linear translation and scaling function $w_{d,i}(t)$ (6.11), which we denote

$$\delta_{d,i} = \frac{1}{t_{i+d} - t_i}, \quad \text{where } d \text{ is the degree and } i \text{ is fixed by} \quad t_i \le t < t_{i+d} \tag{6.13}$$

$\delta_{d,i}$ is a constant, only reflecting the scaling, independent of the translation, and only depending on t to find the index i. In the Bézier case is $\delta_{d,i} = 1$ for all relevant i and d.

The derivative of the matrix $T(t)$ is then defined by the following.

Definition 6.5. *The B-spline derivative matrix T'_d is a $d \times (d+1)$ band limited matrix with two nonzero constant elements on each row (independent of t). The matrix is as follows*

$$T'_d = \begin{pmatrix} -\delta_{d,i-d+1} & \delta_{d,i-d+1} & 0 & \cdots & \cdots & 0 \\ 0 & -\delta_{d,i-d+2} & \delta_{d,i-d+2} & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & 0 & -\delta_{d,i-1} & \delta_{d,i-1} & 0 \\ 0 & \cdots & \cdots & 0 & -\delta_{d,i} & \delta_{d,i} \end{pmatrix}$$

Although T'_d in itself is independent of t, the index i in the definition is dependent of t and follows the rule $t_i \le t < t_{i+1}$ which is the same as for $T_d(t)$ from the definition 6.4.

6.2.4 B-splines on Matrix notations

First a concrete example, given a third degree B-spline curve

$$c(t) = T_1(t)\ T_2(t)\ T_3(t)\ \mathbf{c} = T^3(t)\ \mathbf{c}.$$

It follows from Theorem C.1 given in appendix C.2 that the derivatives are

$$\begin{aligned} c'(t) &= (T'_1\ T_2(t)\ T_3(t) + T_1(t)\ T'_2\ T_3(t) + T_1(t)\ T_2(t)\ T'_3)\ \mathbf{c} = 3\ T^2(t)\ T'_3\ \mathbf{c}, \\ c''(t) &= (T'_1\ T_2(t)\ T'_3 + T_1(t)\ T'_2\ T'_3)\ \mathbf{c} = 6\ T_1(t)\ T'_2\ T'_3\ \mathbf{c}. \end{aligned}$$

We are now ready to look at a general expression of a B-spline function/curve and its derivatives on matrix notation.

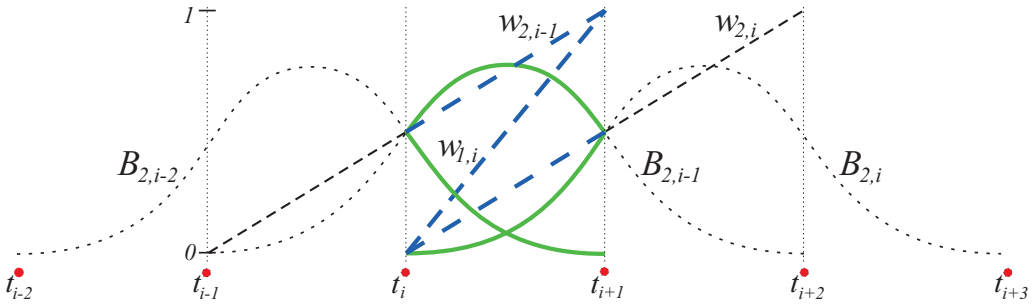


Figure 6.10: *The three 2^{nd}-degree B-splines in the knot interval $[t_i, t_{i+1}]$ (solid green) and the three linear translation and scaling functions (dashed blue) involved in the computations of these B-splines. The B-splines and w are dashed outside the actual interval.*

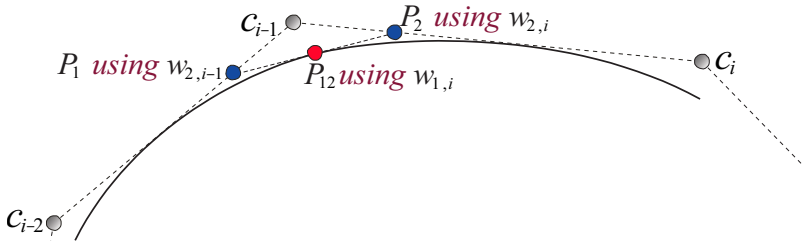


Figure 6.11: *The de Casteljau's corner cutting algorithm applied on a 2^{nd}-degree B-spline curve with knots $0,1,2,3$. In the example the position p_{12} is at $c\left(\frac{3}{2}\right)$.*

Definition 6.6. *A B-spline function/curve of degree d is on matrix notation as follows*

$$c(t) = T_1(t)\ T_2(t)\ \cdots\ T_d(t)\ \mathbf{c} = T^d(t)\ \mathbf{c}$$

where d, $T_1(t)\ T_2(t)\ \cdots\ T_d(t)$ is the factorization of the set of $d+1$ B-splines of degree on the interval $[t_i, t_{i+1}]$ (Definition 6.4), $\mathbf{c} = (c_{i-d},\ c_{i-d+1},\ \ldots,\ c_i)^T$ is the coefficient vector with $d+1$ elements, and where the index i satisfies $t_i \le t < t_{i+1}$.

The j^{th}-derivative, $j = 1,2,\ldots,d$, of a B-spline function/curve of degree d is a function of degree $d-j$ and is as follows

$$c^{(j)}(t) = \frac{d!}{(d-j)!}\ T_1(t)T_2(t)\cdots T_{d-j}(t)\ T'_{d-j+1}\ T'_{d-j+2}\ \cdots\ T'_d\ \mathbf{c} = \frac{d!}{(d-j)!}\ T^{d-j}(t)\ T'^{j}\ \mathbf{c}.$$

6.2.5 An example of B-splines and de Casteljau's algorithm

We will make it simple, i.e. use a 2^{nd}-degree B-spline curve,

$$\begin{aligned} c(t) &= T^2(t)\ \mathbf{c} = T_1(t)\ T_2(t)\ \mathbf{c} = (B_{2,i-2}(t),\ \ B_{2,i-1}(t),\ \ B_{2,i}(t))\,\mathbf{c} \\ &= \begin{pmatrix} 1-w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1-w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1-w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}. \end{aligned}$$

Figure 6.10 shows the three 2^{nd}-degree B-splines, $B_{2,i-2}(t), B_{2,i-1}(t)$ and $B_{2,i}(t)$, together with the three linear translation and scaling functions they are constructed from (remember that this is only in the interval $[t_i, t_{i+1})$). These 3 functions are $w_{1,i}$, in the left hand matrix $T_1(t)$, and $w_{2,i-1}$ and $w_{2,i}$ in the second matrix $T_2(t)$.

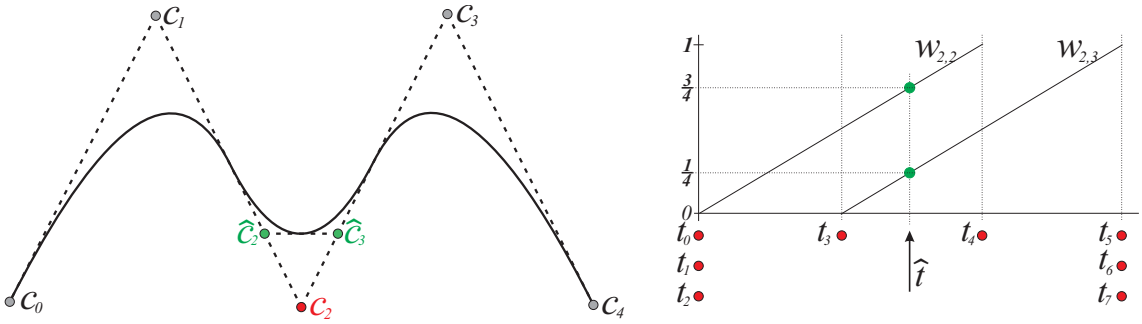


Figure 6.12: *On the left there is a 2^{nd}-degree B-spline curve and its control polygon. On the right is the knot vector shown. One new knot, $\widehat{t}$ is inserted. The result is, as we can see on the left, two new control points $\widehat{c}_2$ and $\widehat{c}_3$, which replaces the old control point c_2.*

Figure 6.11 shows de Casteljau's corner cutting algorithm applied on a 2^{nd}-degree B-spline curve with the knots $\{t_{i-1},\ldots,t_{i+2}\} = \{0,1,2,3\}$. $\widehat{t} = 3/2$ is used as parameter to evaluate in the example. This gives $w_{1,1}(\widehat{t}) = 1/2$ in the first matrix $T_1(t)$, and $w_{2,i-1}(\widehat{t}) = 3/4$ and $w_{2,i}(\widehat{t}) = 1/4$ in the second matrix $T_2(t)$. The first two new points p_1 and p_2 are calculated, and then the last point p_{12} (the one on the curve) is calculated below,

$$\begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} 1-\frac{3}{4} & \frac{3}{4} & 0 \\ 0 & 1-\frac{1}{4} & \frac{1}{4} \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} = \begin{pmatrix} \frac{1}{4}c_{i-2} + \frac{3}{4}c_{i-1} \\ \frac{3}{4}c_{i-1} + \frac{1}{4}c_i \end{pmatrix}.$$

$$p_{12} = \begin{pmatrix} 1-\frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}p_1 + \frac{1}{2}p_2 \end{pmatrix}$$

6.2.6 B-splines and knot insertion

Knot insertion is one of the most important algorithms on B-splines. It was introduced simultaneously in 1980 by Boehm [15] for single knots and Cohen, Lyche and Riesenfield [23] for general knot insertion, the Oslo Algorithm.

Knot insertion is basically corner cutting and can therefor also be expressed in the context of matrix notation. When a single knot is inserted, we have the following expression

$$\widehat{\mathbf{c}} = T_d(\widehat{t})\,\mathbf{c} \tag{6.14}$$

where $\widehat{t}$ is the value of the new knot and $\widehat{\mathbf{c}}$ is a vector of the d new control points replacing the $d-1$ control points that are in the interior of the vector $\mathbf{c}$ on the right hand side (except for the first and the last control points). Here, as usual, the index i is determined by $t_i \le \widehat{t} < t_{i+1}$. We shall look at some examples, first a second degree B-spline function

$$\begin{pmatrix} \widehat{c}_{i-1} \\ \widehat{c}_i \end{pmatrix} = \begin{pmatrix} 1-w_{2,i-1}(\widehat{t}) & w_{2,i-1}(\widehat{t}) & 0 \\ 0 & 1-w_{2,i}(\widehat{t}) & w_{2,i}(\widehat{t}) \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}. \tag{6.15}$$

In Figure 6.12 is there to the left, a 2^{nd}-degree B-spline curve and it's control polygon $\{c_0, c_1, c_2, c_3, c_4\}$ in $\mathbb{R}^2$. On the right, the knot vector and the w-functions are plotted. In

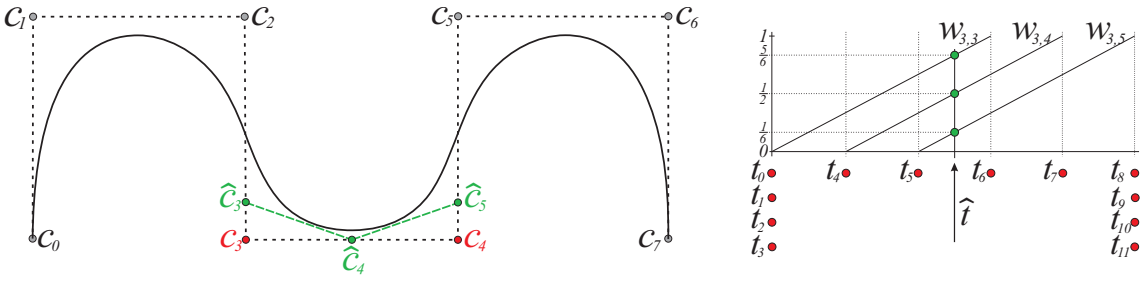


Figure 6.13: *On the left there is a 3^{rd}-degree B-spline curve and its control polygon. On the right is the knot vector shown. One new knot, $\widehat{t}$ is inserted. The result is, as we can see on the left, three new control points $\widehat{c}_3$, $\widehat{c}_4$ and $\widehat{c}_5$, which replaces the two old control points c_3 and c_4.*

the middle a new knot, $\widehat{t}$, is to be inserted. It follows from the value of $\widehat{t}$ that $i = 3$. The two "linear translation and scaling" functions involved in the matrix $T_2(\widehat{t})$ in expression (6.15), $w_{2,2}$ and $w_{2,3}$ are shown to the right in the figure. The result of the knot insertion is that the internal coefficient on the right of (6.15), c_2, is replaced by two new coefficients on the left of (6.15), $\widehat{c}_2$ and $\widehat{c}_3$. This can clearly be seen to the left in Figure 6.12.

Next is a 3^{rd}-degree B-spline function

$$\begin{pmatrix} \widehat{c}_{i-2} \\ \widehat{c}_{i-1} \\ \widehat{c}_i \end{pmatrix} = \begin{pmatrix} 1-w_{3,i-2}(\widehat{t}) & w_{3,i-2}(\widehat{t}) & 0 & 0 \\ 0 & 1-w_{3,i-1}(\widehat{t}) & w_{3,i-1}(\widehat{t}) & 0 \\ 0 & 0 & 1-w_{3,i}(\widehat{t}) & w_{3,i}(\widehat{t}) \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} \quad (6.16)$$

In Figure 6.13 is there, to the left, a third degree B-spline curve (in $\mathbb{R}^2$) and its control polygon $(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$. On the right side there is the knot vector illustrated (on $\mathbb{R}$, horizontal). In the middle a new knot, $\widehat{t}$, is to be inserted. It follows from the position of $\widehat{t}$ that $i = 5$. The three "linear translation and scaling" functions involved in the matrix $T_3(\widehat{t})$ in expression (6.16), $w_{3,3}$, $w_{3,4}$ and $w_{3,5}$ are shown to the right in Figure 6.13. The result of the knot insertion is that the internal coefficients on the right side of (6.16), c_3 and c_4 are replaced by the three new coefficients on the left side of (6.16), $\widehat{c}_3$, $\widehat{c}_4$ and $\widehat{c}_5$. This can clearly be seen on the left in Figure 6.13.

From subsection 4.4.2 and definition 6.4 it is clear that the matrix $T_d(t)$ is a corner cutting matrix. It is therefore obvious that knot insertion is corner cutting. From the two examples we can see that the new control points we get when we insert one new knot are on the old control polygon. This is related to discrete B-splines,[6] and it leads us to subdivision technics. Inserting more than one knot (Oslo algorithm) leads us to blossoming, see Goldman in [76].

[6]Discrete splines (on a uniform grid) were introduced by Mangasarian and Schumaker in [117] as solutions to certain variational problems. They were discussed in detail by Tom Lyche in his Ph.D. Thesis, "Discrete polynomial spline approximation methods", for which there is a summary in [115].

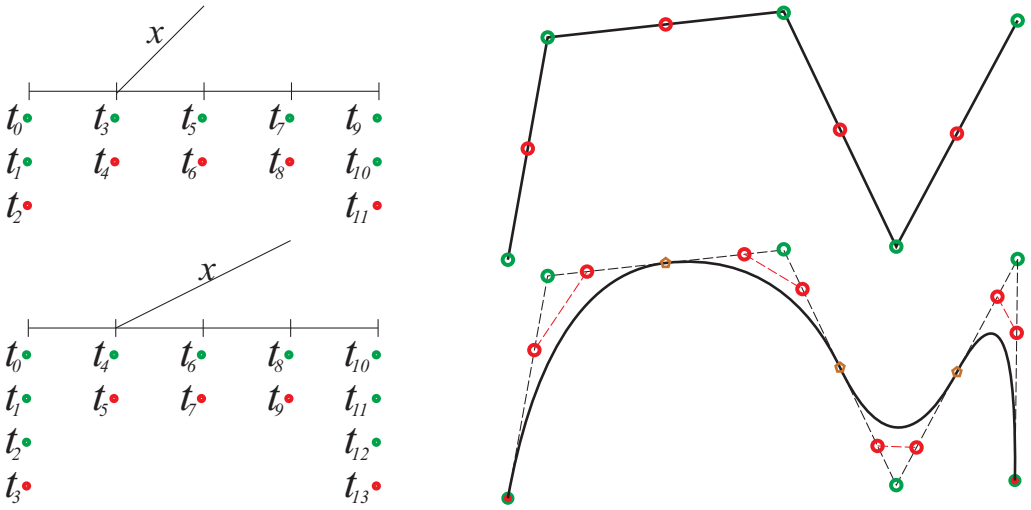


Figure 6.14: *On the top left, in green, we see the knot vector for a 1^{st}-degree B-spline curve. To the right we see the curve itself and its 5 (green) control points. After degree raising to a 2^{nd}-degree B-spline curve we see the same curve but we now get 9 control points, including 4 new marked red. In the lower part we see a 2^{nd}-degree curve, green knots and control points, raised to a 3^{rd}-degree curve, where red knots have been added where the internal green control points have been replaced with the red ones.*

6.2.7 Degree elevation of B-splines

In subsection 4.4.4 is degree elevation of Bézier Curves shown. At each parameter interval (between two knot clusters) there is a polynomial based curve segment of the given degree. If, at each parameter interval, we transform the format to power basis, and add one more term t^{d+1}, ie $c(t) = \sum_{i=0}^{d+1} a_i t^i$, where $a_{d+1} = 0$, we have raised the degree by 1. The challenge is to do this with the B-spline format.

Because of the continuity property for B-splines over knot values, it follows that to maintain the continuity as we increase the degree by 1, we must increasing the multiplicity of each cluster of knots with 1. Thus, the number of control points must increase with the number of knot intervals.

We first use a 1^{st}-degree, piecewise continuous linear curve as an example, it is quite obvious that, after the degree elevation, there will be one new control point between each of the old ones, and this is true for each knot interval. It is also obvious that the new control points will be on the original control polygon, in the middle between two old points.

To verify this observation we first distinguish between the 1^{st}-degree and the 2^{nd}-degree curve. We therefore use $\sim$ in the notation for the original 1^{st}-degree curve. Given a knot vector for a 1^{st}-degree B-spline space, $\tilde{\tau} = \{\tilde{t}_i\}_{i=0}^{6} = \{0,0,1,2,3,4,4\}$. After degree elevation we get $\tau = \{t_i\}_{i=0}^{11} = \{0,0,0,1,1,2,2,3,3,4,4,4\}$. Recall (6.11), ie $\widetilde{w}_{1,i}(t) = \frac{t-\tilde{t}_i}{\tilde{t}_{i+1}-\tilde{t}_i}$. To simplify, we set $x = \widetilde{w}_{1,i}(t)$ and $\bar{x} = 1-x$. If after the degree elevation, $t_{j-1} = t_j$ and $t_{j+1} = t_{j+2}$, then $x = \widetilde{w}_{1,i}(t) = w_{2,2i-1}(t) = w_{2,2i}(t)$. We thus get the following

expression for the knot elevation in each knot interval,

$$c(t) = \begin{pmatrix} \bar{x} & x \end{pmatrix} \begin{pmatrix} \widetilde{c}_{i-1} \\ \widetilde{c}_i \end{pmatrix} = \begin{pmatrix} \bar{x} & x \end{pmatrix} \begin{pmatrix} \bar{x} & x & 0 \\ 0 & \bar{x} & x \end{pmatrix} \begin{pmatrix} c_{2i-2} \\ c_{2i-1} \\ c_{2i} \end{pmatrix}$$

From $x = 0$ (start of the interval) it follows that $c_{2i-2} = \widetilde{c}_{i-1}$ and from $\bar{x} = 0$ (end of the interval) that $c_{2i} = \widetilde{c}_i$. By solving $\bar{x}^2\widetilde{c}_{i-1} + 2\bar{x}x\ c_{2i-1} + x^2\widetilde{c}_i = \bar{x}\widetilde{c}_{i-1} + x\widetilde{c}_i$ we get the new point $c_{2i-1} = \frac{1}{2}\widetilde{c}_{i-1} + \frac{1}{2}\widetilde{c}_i$. In the upper part of Figure 6.14 is this degree elevation illustrated. On the left side we see the knot vector (green), after inserting new (red) knots at each knot cluster. Also $x = \widetilde{w}_{1,2}(t) = w_{2,3}(t) = w_{2,4}(t)$ is shown. On the right side we see the piecewise linear 1^{st}-degree B-spline curve and its control points (green). The 2^{nd}-degree curve is the same curve, but where the red points are the ones we have added in the degree elevation. Together with the green points they form the new control polygon.

For a 2^{nd}-degree B-spline curve, the number of control points will increase with the number of knot intervals. In fact, if $\tilde{t}_i < \tilde{t}_{i+1}$, then $\tilde{c}_{i-1}$ must be replaced by two new control points. This is because we insert one new knot into each cluster of knots. To illustrate the process, we start with a knot vector $\widetilde{\tau} = \{\tilde{t}_i\}_{i=0}^{8} = \{0,0,0,1,2,3,4,4,4\}$. This will generate six 2^{nd}-degree B-splines, and the B-spline curve $c(t) = \sum_{i=0}^{5} b_{2,i}(t)\ \widetilde{c}_i$. On the lower part of Figure 6.14 there is an example of such a curve, illustrated with green knots and green control points. The new knot vector is $\tau = \{t_i\}_{i=0}^{13} = \{0,0,0,0,1,1,2,2,3,3,4,4,4,4\}$ illustrated on the left in the figure by added red dots. To the right we see the curve with both old and new control points as well as a brown pentagons at each cluster of internal knots. Now, to find the new set of control points, we do the following. For every knot interval, ie for $i = 0,1,2,\ldots,\tilde{n}-2$, (in Figure 6.14 is $\tilde{n} = 6$)

$q = (1 - w_{2,i+1}(\tilde{t}_{i+2}))\tilde{c}_i + w_{2,i+1}(\tilde{t}_{i+2})\tilde{c}_{i+1}$, position of a knot cluster on the curve.

We make two new control point for this knot interval, see algorithm (4.37),

$c_{2i} = \frac{2}{3}\tilde{c}_i + \frac{1}{3}q$ and $c_{2i+1} = \frac{1}{3}q + \frac{2}{3}\tilde{c}_{i+1}$,

which is equivalent to the Bézier-algorithm. Note that $w_{2,i}(t)$ uses the original knot vector $\widetilde{\tau}$. We shorten this and get the generally degree elevation algorithm from degree 2 to 3.

Degree elevation from 2 to 3

For $i = 0,1,2,\ldots,\tilde{n}-2$

$x = w_{2,i+1}(\tilde{t}_{i+2})$	// Relative distribution in this knot interval
$c_{2i} = \left(1 - \frac{x}{3}\right)\tilde{c}_i + \frac{x}{3}\tilde{c}_{i+1}$	// Two control points in this knot interval
$c_{2i+1} = \frac{1}{3}(1-x)\tilde{c}_i + \frac{1}{3}(2+x)\tilde{c}_{i+1}$	// for the elevated 3^{rd}-degree B-spline curve.

If there are multiple internal knots in the 2^{nd}-degree B-spline curve we will get one control points twice. For example, if $\tilde{t}_4 = \tilde{t}_5$, c_5 will be equal to c_6, and also the resulting knot vector will have 4 equal knots $t_6 = t_7 = t_8 = t_9$. But we can (and should) reduse this to just 3 equal knots, and we then have to skip c_6 and reduce the index of the next control points by 1 in the algorithm.

As a final example, we will look at a 3^{rd}-degree curve. In this example the knot vector is $\widetilde{\tau} = \{0,0,0,0,1.08,2,3.2,4,5.1,6.5,7,7,7,7\}$, ie nonuniform with only single internal knots. Thus the knot vector of the elevated 4^{th}-degree B-spline curve will be

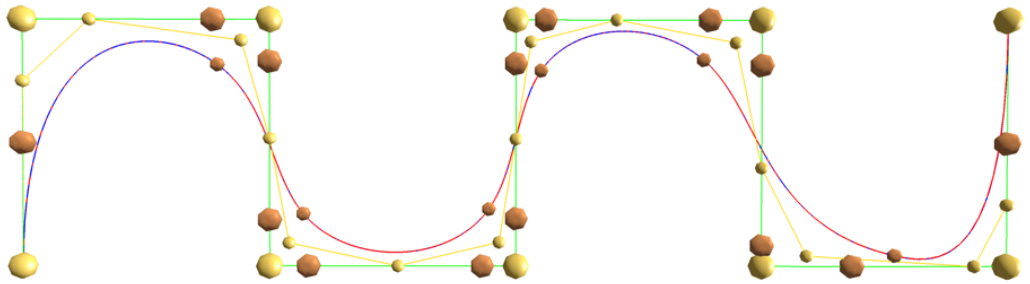


Figure 6.15: *We see a 3-degree B-spline curve in blue, and its 10 control points, large brass-colored spheres, along with the degree elevated (4^{th}-degree) B-spline curve in red and its 17 control points, small brass-colored spheres. The large copper-colored spheres are the points in the first corner cutting step, q_0 and q_1. The small copper-colored spheres are the position of the internal knot clusters.*

$\tau = \{0,0,0,0,0,1.08,1.08,2,2,3.2,3.2,4,4,5.1,5.1,6.5,6.5,7,7,7,7,7\}$. The example is shown in Figure 6.15. Note that the new basis functions b_0 and b_1 cover 1 knot interval, b_2 and b_3 cover 2 intervals, b_4 cover 3 intervals, b_5 cover 2 intervals, and so on with 2 and 3 knot intervals every other time, until the end mirrors the start. We initially have 10 control points and we get 17 after the degree elevation. It follows that the two first and the two last control points follows the Bézier algorithm because the related basic functions only cover 1 knot interval, but we do not need to treat them separately. Furthermore, we must distinguish between the control points associated with basic functions that cover 2 and those that cover 3 knot intervals. At start and end the cover is redused due to the knot multipisity, but this will not efect how we have to treat them. We only need to separate the algorithm into one for the new control points with an even number as an index, and one for the new control points with an odd number as an index. In our example, for the points with an even index we get, for $i = 0,1,\ldots,8$,

$$c_{2i} = (1-x)\,\tilde{c}_i + x\,\tilde{c}_{i+1}, \qquad \text{where} \quad x = \frac{w_{3,i+1}(\tilde{t}_{i+2}) + w_{3,i+1}(\tilde{t}_{i+3})}{2}.$$

In Figure 6.15, these points can be seen as small brass-colored spheres that lies on the original control polygon. Now, the points with an odd index are connected to corners, we get, for $i = 0,1,\ldots,7$,

$$\begin{aligned} q_0 &= x\tilde{c}_i + (1-x)\tilde{c}_{i+1}, \qquad \text{where} \quad x = \frac{1 - w_{3,i+1}(\tilde{t}_{i+3})}{2}, \\ q_1 &= (1-y)\tilde{c}_{i+1} + y\tilde{c}_{i+2}, \qquad \text{where} \quad y = \frac{w_{3,i+2}(\tilde{t}_{i+3})}{2}, \\ c_{2i+1} &= \frac{q_0+q_1}{2}, \end{aligned}$$

Note that $w_{3,i}(t)$ uses the original knot vector $\widetilde{\tau}$. In Figure 6.15, q_0 and q_1 are marked as large copper-colored spheres on either side of one original internal control points, exept for one that is coverd by the second original point and one that is covered by the second last original point. The new control points with odd indices (small brass-colored spheres) lies in the middle between their respective q_0 and q_1. We make the algorithm general, shorten it and get the degree elevation algorithm from degree 3 to 4.

Degree elevation from 3 to 4

For $i = 0,1,2,\ldots,\tilde{n}-2$
$x = \frac{w_{3,i+1}(\tilde{t}_{i+2})+w_{3,i+1}(\tilde{t}_{i+3})}{2}$ // Relative distribution in this knot interval
$c_{2i} = (1-x)\,\tilde{c}_i + x\,\tilde{c}_{i+1}$ // Control points with an even index number
For $i = 0,1,2,\ldots,\tilde{n}-3$
$x = w_{3,i+1}(\tilde{t}_{i+3})$ // First scale and translation mapping
$y = w_{3,i+2}(\tilde{t}_{i+3})$ // Second scale and translation mapping
$c_{2i+1} = \frac{1-x}{4}\tilde{c}_i + \frac{3+x-y}{4}\tilde{c}_{i+1} + \frac{y}{4}\tilde{c}_{i+2}$ // Control points with an odd index number

If there are multiple internal knots in the 3nd degree B-spline curve we will get "redundent" control points. For example, if $\tilde{t}_5 = \tilde{t}_6$, c_6 will lies on the line between c_5 and c_7 , and the resulting knot vector will have 4 equal knots $t_7 = t_8 = t_9 = t_{10}$. But we can (and should) reduse this to just 3 equal knots. Thus we have to skip c_6 by simply reducing the indices of the next "odd" control points by 1, and increasing the indices of the next "even" control points by 1, but in the "even" case we also need to increase the indices in the computation to $w_{3,i+2}(\tilde{t}_{i+3})$, $w_{3,i+2}(\tilde{t}_{i+4})$, $\tilde{c}_{i+1}$ and $\tilde{c}_{i+2}$.

6.2.8 Blossoming - Polar form

In 1987 L. Ramshaw introduced blossoming as a way to treat B-splines and corner cutting, [131], [132] and [133]. This was based on the work of P. de Casteljau from 1984, [40]. R. Goldman has also later worked with blossoming in [76] and [77].

The blossom or polar form of a polynomial of a single variable of degree d, $p_d(t)$, is the symmetric multiaffine polynomial $b_p(u_1,\ldots,u_d)$ such that $b_p(t,\ldots,t) = p_d(t)$. Blossoming means replacing a degree d polynomial in one variable by an equivalent symmetric polynomial in d variables where each new variable is only of power 1. Thus the blossom $b_p(t_1,\ldots,t_d)$ is characterized by three properties,

Diagonal $b_p(t,t,t) = p_3(t)$, where $d = 3$ can be replaced by any number > 0.

Symmetry $b_p(u_1,\ldots,u_i,\ldots,u_j,\ldots,u_d) = b_p(u_1,\ldots,u_j,\ldots,u_i,\ldots,u_d)$

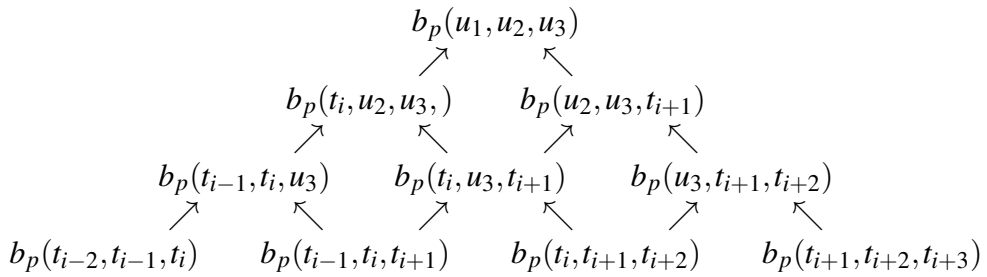
Multiaffine $b_p(\ldots,a\,u_i+(1-a)u_i,\ldots) = a\,b_p(\ldots,u_i,\ldots)+(1-a)b_p(\ldots,u_i,\ldots)$

It follows that blossoming is a kind of factorisation down to only d-linear functions, in a way the same as we see in matrix notation,

$$b_p(u_1,u_2,u_3) = \begin{pmatrix} 1-w_{1,i}(u_1) & w_{1,i}(u_1) \end{pmatrix} \begin{pmatrix} 1-w_{2,i-1}(u_2) & w_{2,i-1}(u_2) & 0 \\ 0 & 1-w_{2,i}(u_2) & w_{2,i}(u_2) \end{pmatrix}$$
$$\begin{pmatrix} 1-w_{3,i-2}(u_3) & w_{3,i-2}(u_3) & 0 & 0 \\ 0 & 1-w_{3,i-1}(u_3) & w_{3,i-1}(u_3) & 0 \\ 0 & 0 & 1-w_{3,i}(u_3) & w_{3,i}(u_3) \end{pmatrix} \begin{pmatrix} b_p(t_{i-2},t_{i-1},t_i) \\ b_p(t_{i-1},t_i,t_{i+1}) \\ b_p(t_i,t_{i+1},t_{i+2}) \\ b_p(t_{i+1},t_{i+2},t_{i+3}) \end{pmatrix},$$

where $[t_i,t_{i+1}]$ is the active domain, and $u_j, j = 1,2,3$ must be in the domain. If we

organize the sub-results as a pyramid we get,



6.2.9 Algorithms for B-splines

The recursion formula for B-splines was given i Definition 6.3 and clearly illustrated by the matrix notation in subsection 6.2.3. An algorithm would therefore be to compute the factor matrices from left to right, but skip the elements that are zero. In the following algorithm, we will fill in a vector with $d+1$ real numbers, one for each of the active B-splines (basic functions) at the current interval $[t_i, t_{i+1})$ when $t_i \le t < t_{i+1}$.

Algorithm 3. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes a vector $\mathbf{b}_d(t) \in \mathbb{R}^{d+1}$*, containing the values of the* $d+1$ *B-splines* $\{b_{d,j}(t)\}_{j=\varsigma-d}^{\varsigma}$*, where* ς *is determined by* $t_\varsigma \le t < t_{\varsigma+1}$*.The input variables are: the knot vector* τ*, the polynomial degree of the B-splines d, the index* ς*, which is derived from* $t_\varsigma < t \le t_{\varsigma+1}$*, and the parameter value* $t \in [t_d, t_n]$*.*

vector⟨double⟩ **bspline(** vector⟨double⟩ τ, int d, int ς , double t **)**
- vector⟨double⟩ *b(d+1);* — *// The return vector, dimension* $d+1$*.*
- vector⟨double⟩ *w(d);*
- $b_1 = W_{1,\varsigma}(t;\tau)$; — *// see (6.11)*
- $b_0 = 1 - b_1$; — *// The general Cox/deBoor algorithm for*
- for *(* int $i = 2$*;* $i \le d$*;* $i{+}{+}$*)* — *// - B-splines, computing the set*
  - for *(* int $j = 0$*;* $j < i$*; j++)* — *// - of all B-spline values of degree d at t*
    - $w_j = W_{i,\varsigma-j}(t;\tau)$; — *// - when* $\tau_\varsigma \le t < \tau_{\varsigma+1}$*.*
  - $b_i = (1 - w_0)\, b_{i-1}$*;*
  - for *(* int $j = i-1$*;* $j > 0$*;* $j{-}{-}$*)*
    - $b_j = w_{i-j}\, b_{j-1} + (1 - w_{i-j-1})\, b_j$*;*
  - $b_0 = w_{i-1}\, b_0$*;*
- return *b;*

The Algorithm 3 is a classical optimal Cox-deBoor algorithm for B-splines.

Often we will need derivatives of order 1, 2, etc. Therefore, an algorithm is needed that not only compute the values of the B-spline functions, but also derivatives of several orders. For a degree d polynomial function, there will be d derivatives of subsequent order that may have values different from zero. Therefor, we need a $d+1 \times d+1$ matrix $B_{d,\tau}(t)$ of real numbers to store all these values. We get $\mathbf{c}(t) = B_{d,\tau}(t)\ \mathbf{c}$, where $\mathbf{c}(t)$ is a vector of one value and d subsequent derivatives. Note that $\mathbf{c} = B_{d,\tau}(t)^{-1}\mathbf{g}$, ie Taylor expansion at t, where $\mathbf{g}$ is the position and d subsequent derivatives. To compute the position and d derivatives of a 3^{th}-degree B-spline curve, we have the following formulaes,

$$\begin{aligned} c(t) &= \mathbf{T}^3(t)\,\mathbf{c} &&= T_1(t)T_2(t)T_3(t)\,\mathbf{c}, \\ c'(t) &= 3\,\mathbf{T}^2(t)\mathbf{T}'\,\mathbf{c} &&= 3\,T_1(t)T_2(t)\,T_3'\,\mathbf{c}, \\ c''(t) &= 6\,\mathbf{T}^1(t)\mathbf{T}'^2\,\mathbf{c} &&= 6\,T_1(t)\,T_2'T_3'\,\mathbf{c}, \\ c'''(t) &= 6\,\mathbf{T}'^3\,\mathbf{c} &&= 6\,T_1'T_2'T_3'\,\mathbf{c}. \end{aligned} \tag{6.17}$$

In the first line of (6.17) we compute the upper left triangle of the matrix with values of the B-spline functions from degree 0 to d from the bottom and up. In the next step we compute T_1' on the bottom line the matrix, then we compute T_2' on the bottom line and the line above, and so on until all derivatives are computed according to (6.17)

The following algorithm creates the matrix $\mathbf{B}_{d,\tau}(t,\varsigma)$, where d is the polynomial degree, $\tau = \{t_0, t_1, \ldots, t_{n+d}\}$ is the knot vector, $t \in [t_d, t_n]$ is the parameter value and the index ς is derived from $t_\varsigma < t \le t_{\varsigma+1}$. The matrix is defined and made in the same way as the Bernstein-Hermite matrix, described in subsection 4.4.3, but where the Bernstein factor matrices, defined in section 4.4.3, are replaced by the matrices in definition 6.4 and 6.5.

Algorithm 4. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes the extended square matrix $\mathbf{B}_{d,\tau}(t,\varsigma) \in \mathbb{R}^{d+1\times d+1}$, which contains in the first row, the values of the $d+1$ B-splines $\{b_{d,i}(t)\}_{i=\varsigma}^{\varsigma+d}$, and in the following rows values for each of the d derivatives, $\{D^j b_{d,i}(t)\}_{i=\varsigma}^{\varsigma+d}$, $j = 1,2,\ldots,d$. The input variables are: the knot vector τ, the polynomial degree of the B-splines d, the index ς, which is derived from $t_\varsigma < t \le t_{\varsigma+1}$, and the parameter value $t \in [t_d, t_n]$.

```
matrix⟨double⟩ BSplineHermiteMat ( vector⟨double⟩ τ, int d, int ς , double t )
    matrix⟨double⟩ B(d+1,d+1);        // The return matrix, dimension (d+1)×(d+1).
    vector⟨double⟩ w(d);
    B_{d-1,1} = W_{1,ς}(t;τ);                    // see (6.11)
    B_{d-1,0} = 1 − B_{d-1,1};                   // The general Cox/deBoor algorithm for
    for ( int i=d-2, k=2; i ≥ 0; i−−, k++ )      // - B-splines, computing the triangle
        for ( int j=1; j < k; j++ )              // - of all values of B-splines of degree
            w_j = W_{k,ς−k+j+1}(t;τ);            // - 1 to d, respectively in each row.
        B_{i,0} = (1 − w_0) B_{i+1,0};
        for ( int j=1; j < d − i; j++ )
            B_{i,j} = w_{j−1} B_{i+1,j−1} + (1 − w_j) B_{i+1,j};
        B_{i,d−i} = w_{k−1} B_{i+1,d−i−1};
    B_{d,1} = δ_{1,ς}(τ);                        // see (6.13)
    B_{d,0} = −B_{d,1};                          // Multiply all rows except the upper one
    for ( int k=2; k ≤ d; k++ )                  // - with the derivative matrices in the
        for ( int j=0; j < k; j++ )              // - definition 6.5, so every row
            w_j = k δ_{k,ς−k+j+1}(τ);            // - extends the number of
        for ( int i = d; i > d − k; i−− )        // - nonzero elements to d+1.
            B_{i,k} = w_{k−1} B_{i,k−1};
            for ( int j = k − 1; j > 0; j−− )
                B_{i,j} = w_{j−1} B_{i,j−1} − w_j B_{i,j} ;
            B_{i,0} = −w_0 B_{i,0};
    return B;
```

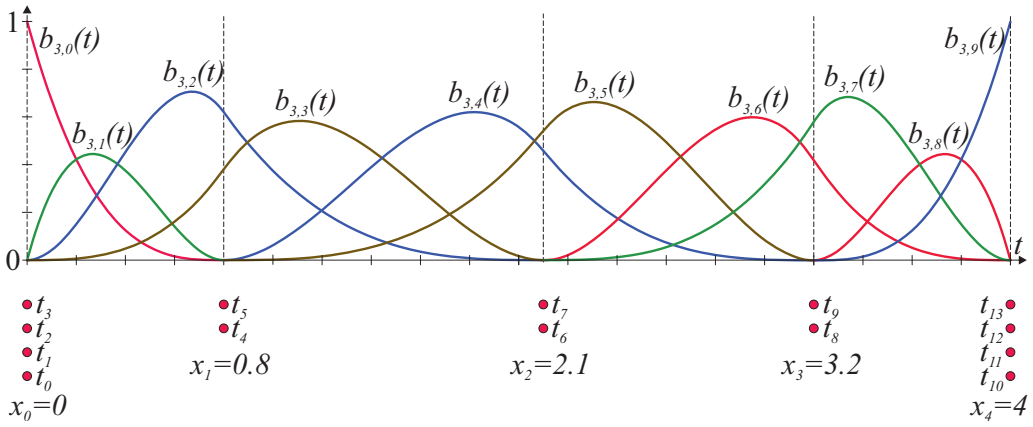


Figure 6.16: *The B-spline basis functions for Hermite splines. There are 5 interpolation points and 10 B-splines. The knot values are marked by dots. One can see that the multiplicity of the internal knots is 2.*

6.3 Hermite spline interpolation on B-spline form

In section 5.6, page 68–68, cubic Hermite splines were described in both algebraic and geometric form. We shall now look at cubic Hermite splines in B-spline form.

Given m strictly increasing real numbers $\{x_i\}_{i=1}^m$, m points $\{p_i\}_{i=1}^m$ and m respective vectors $\{v_i\}_{i=1}^m$. We can now define a cubic Hermite splines in B-spline form, i.e.

$$c(t) = T^3(t)\,\mathbf{c}.$$

This curve will interpolate the points and the vectors at the given real numbers, i.e. $c(x_i) = p_i$ and $c'(x_i) = v_i$. We first make a knot vector, $\{t_i\}_{i=0}^{n+3}$, where $n = 2m$, setting

$$\begin{array}{lll} \text{at the start:} & t_0 = t_1 = t_2 = t_3 = x_1 & \\ \text{at the end:} & t_n = t_{n+1} = t_{n+2} = t_{n+3} = x_m & \\ \text{and otherwise:} & t_i = t_{i+1} = x_j, \quad \text{for} \quad i = 4,6,8,\ldots,2(m-1) \quad \text{and} \quad j = \frac{i}{2}. & \end{array} \tag{6.18}$$

We then compute the control points, which we are construct by setting

$$c_0 = p_1 \quad \text{and} \quad c_{n-1} = p_m,$$

and

$$\begin{array}{llll} c_i = p_j + \frac{\Delta x_j}{6} v_j, & \text{for} \quad i = 1,3,\ldots,n-3 & \textit{where} & j = \frac{i+1}{2}, \\ c_i = p_{j+1} - \frac{\Delta x_j}{6} v_{j+1}, & \text{for} \quad i = 2,4,\ldots,n-2 & \textit{where} & j = \frac{i}{2}, \end{array} \tag{6.19}$$

where $\Delta x_j = x_{j+1} - x_j$.

We can of course generate the vectors $\{v_i\}_{i=1}^m$ using either Cardinal, or Catmull-Rom spline, or Bessel's interpolation, or Akima's interpolation method, as described in section 5.6. In Figure 6.16 is there an example of a set of B-splines (basis functions) that are constructed to interpolate $m = 5$ points. One can observe in the figure, and recognize

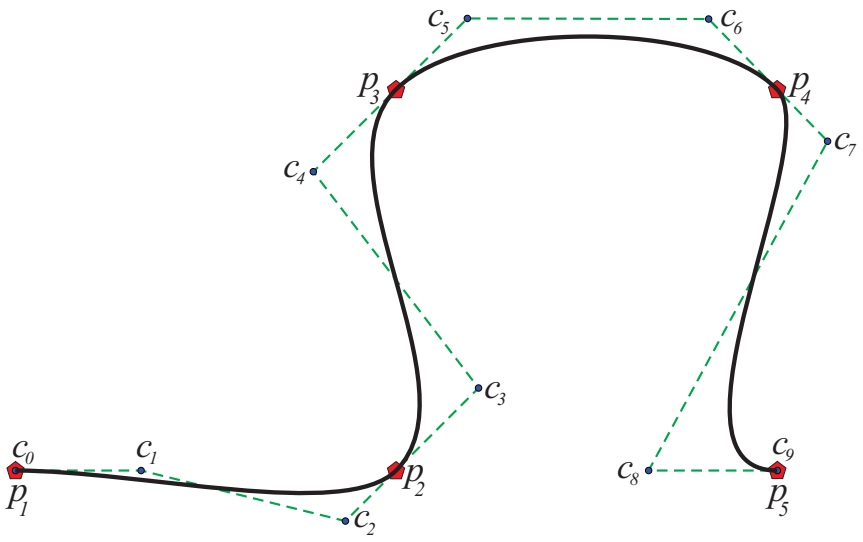


Figure 6.17: *We see an Hermite interpolation of 5 points $\{p_i\}_{i=1}^{5}$ and respective vectors $\{v_i\}_{i=1}^{5}$ by Hermite spline on B-spline form. The control polygon appears in dashed green. The control points $\{c_i\}_{i=0}^{9}$ are blue, and the interpolation points $\{p_i\}_{i=0}^{4}$ are marked red. The resulting curve is in solid black.*

from the expressions in (6.18), that all the inner knots are of multiplicity 2. This is because an Hermite spline is C^1-smooth. A 3^{rd}-degree B-spline is $C^2[t_d,t_n]$ with simple internal knots , so to be only $C_1[t_0,t_n]$ there must be 2 equal internal knots. This also secure the interpolation. In Figure 6.16 we can observe that there are 5 clusters of knot-values, with multiplicity 2 in the internal and 4 at the ends. The value of each of these clusters is the parameter value of the interpolation points, i.e. $c(x_i) = p_i$. We can see in Figure 6.16 that over the internal knots there are only two B-splines that are different from zero. This ensures that the interpolation points lie on the control polygon, the straight line between two control points, and is actually how we construct the control points in (6.19).

Figure 6.17 shows a Catmull-Rom spline where the vectors $v_i = \frac{1}{2}(p_{i+1} - p_{i-1})$ and the x values are $\{0, 0.8, 2.1, 3.2, 4\}$ as in Figure 6.16. In Figure 6.17 we can see a solid black B-spline curve together with the control polygon in dashed green. The 10 control points $\{c_i\}_{i=0}^{9}$ are marked as blue circles and the 5 interpolation points $\{p_i\}_{i=1}^{5}$ are marked with red stars. The tangent vectors are $v_0 = (p_2 - p_1)$, $v_1 = \frac{1}{2}(p_3 - p_1)$, $v_2 = \frac{1}{2}(p_4 - p_2)$, $v_3 = \frac{1}{2}(p_5 - p_3)$ and $v_4 = v_0$. If we introduce tension parameters, the example is also potentially a Cardinal spline, also called a Canonical spline.

If we generally introduce tension parameters $\{\rho_i\}_{i=1}^{5}$, we must change (6.19) to

$$\begin{aligned} c_i &= p_j + \tfrac{\Delta x_j\, \rho_j}{6}\, v_j, && \text{for} \quad i = 1,3,\ldots,n-3 \quad \textit{where} \quad j = \tfrac{i+1}{2}, \\ c_i &= p_{j+1} - \tfrac{\Delta x_j\, \rho_{j+1}}{6} v_{j+1}, && \text{for} \quad i = 2,4,\ldots,n-2 \quad \textit{where} \quad j = \tfrac{i}{2}. \end{aligned}$$

Also Bessels spline or Akima's method described in sections 5.6 can be used to generate the tangent vectors.

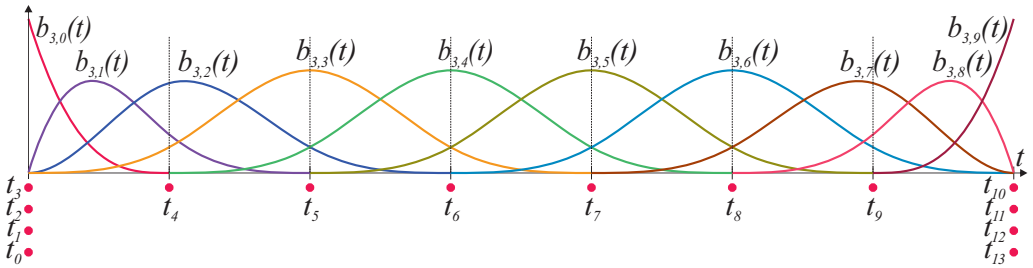


Figure 6.18: *The figure shows the 10 3^{rd}-degree B-splines that are generated by a knot vector of 14 elements, $\{t_i\}_{i=0}^{13}$, that is used for cubic spline interpolation of 8 points. As we can see, only 3 B-splines are different from zero at the internal knots. That is why the matrix A in the equation (6.20), is three diagonal.*

6.4 Cubic spline interpolation on B-spline form

In section 5.7 a cubic spline interpolation is described. We shall now look at cubic spline interpolation on B-spline form.

In section 5.7, we used Hermite interpolation as the basic form even though the curve was C^2-smooth. This is why the unknowns were the 1^{st}-derivatives in the interpolation points. When we have B-splines, it is natural that the unknowns are the coefficients, i.e. the control points.

Given m strictly increasing real numbers $\{x_i\}_{i=1}^m$ and m points $\{p_i\}_{i=1}^m$. We will now construct a 3^{rd}-degree B-spline curve,

$$c(t) = T^3(t)\,\mathbf{c}.$$

This curve interpolates the given points at the given real numbers, i.e. $c(x_i) = p_i$, $i = 1,2,\ldots,m$. Because we use 3^{rd}-degree B-splines, it follows that the number of knot values must be $m+6$. This is because we want single knots internally, since the curve is actually C^2-smooth, and four knots at start and end. Since the number of knots is the number of control points plus the order of the B-spline, that is 4, it follows that the number of control points is $n = m+2$. First, The knot vector must be

$$\begin{aligned}
&\text{at the start:} && t_0 = t_1 = t_2 = t_3 = x_1\\
&\text{at the end:} && t_n = t_{n+1} = t_{n+2} = t_{n+3} = x_m\\
&\text{and otherwise:} && t_i = x_{i-2}, \quad \text{for} \quad i = 4,5,\ldots,n-2,n-1.
\end{aligned}$$

Figure 6.18 shows the knot vector we get from a given x-vector and a plot of the B-splines it generates. When the degree is 3 then there will be $k = 4$ equal knots at start and end. It then follows that the control points at start and at the end are equal to the interpolation points, i.e.

$$c_0 = p_1 \quad \text{and} \quad c_{n-1} = p_m.$$

To compute the remaining control points we have to solve a system of linear equations similar to equation (5.15) in section 5.7, i.e.

$$\mathbf{A}\,\mathbf{c} = \mathbf{b} \tag{6.20}$$

where

$$\mathbf{A} = \begin{pmatrix} B''_{3,1}(t_3) & B''_{3,2}(t_3) & 0 & \cdots & \cdots & 0 \\ B_{3,1}(t_4) & B_{3,2}(t_4) & B_{3,3}(t_4) & 0 & \ddots & \vdots \\ 0 & B_{3,2}(t_5) & B_{3,3}(t_5) & B_{3,4}(t_5) & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & B_{3,n-4}(t_{n-1}) & B_{3,n-3}(t_{n-1}) & B_{3,n-2}(t_{n-1}) \\ 0 & \cdots & \cdots & 0 & B''_{3,n-3}(t_n) & B''_{3,n-2}(t_n) \end{pmatrix},$$

and

$$\mathbf{c} = \begin{pmatrix} c_1 \\ \vdots \\ \vdots \\ \vdots \\ c_{n-2} \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} -B''_{3,0}(t_3)\, c_0 \\ p_2 \\ \vdots \\ p_{m-1} \\ -B''_{3,n-1}(t_n)\, c_{n-1} \end{pmatrix}.$$

We have, in this equation, used the free end condition, which is that the second derivative is zero at start and end (see (5.19) in section 5.7). All rows in $\mathbf{A}$, except the first and last, can be created using Algorithm 3. The first and last rows can be created using Algorithm 4, and then 3^{rd} row from the result.

Figure 6.18 shows an example of B-splines in cubic spline interpolation where $m = 8$ (you can see that there are 8 clusters of knots). It follows that there are 10 B-splines, $n = 10$, and that there are 14 knots (marked with red dots). On the left side of Figure 6.18 we see that only 4 B-splines are active at start, $B_{3,0}$, $B_{3,1}$, $B_{3,2}$ and $B_{3,3}$. But since $B''_{3,3}(t_3) = 0$ it follows that we get an expression with 3 terms that describe the 2^{nd} derivative at start. At the end of the curve we can see something similar. Thus we get the following conditions,

$$\begin{aligned} c''(t_3) &= B''_{3,0}(t_3)\, c_0 + B''_{3,1}(t_3)\, c_1 + B''_{3,2}(t_3)\, c_2 = 0 \\ c''(t_{10}) &= B''_{3,7}(t_{10})\, c_7 + B''_{3,8}(t_{10})\, c_8 + B''_{3,9}(t_{10})\, c_9 = 0. \end{aligned}$$

If we reorganize these two expressions we get,

$$\begin{aligned} B''_{3,1}(t_3)\, c_1 + B''_{3,2}(t_3)\, c_2 &= -B''_{3,0}(t_3)\, c_0 \\ B''_{3,7}(t_{10})\, c_7 + B''_{3,8}(t_{10})\, c_8 &= -B''_{3,9}(t_{10})\, c_9 \end{aligned}$$

These two requirements are the first and the last line of matrix A in (6.20). As we see in Figure 6.18, only three B-splines are different from zero in all internal knots. Thus, to interpolate in these internal knots, it follows that (as we can see from the figure)

$$c(t_i) = B_{3,i-3}(t_i)\, c_{i-3} + B_{3,i-2}(t_i)\, c_{i-2} + B_{3,i-1}(t_i)\, c_{i-1} = p_{i-2},$$

which is actually what all the other lines in the matrix A in equation (6.20) show.

6.5 B-spline approximation and least squares

As we saw in cubic spline interpolation, the number of interpolation points is equal to the number of control points minus the start and end point. That is, the system is determined and thus can be solved. If, on the other hand, the number of interpolation points is greater than the degrees of freedom, we cannot interpolate, because then we do not get a square matrix. So what can be done? One possibility is to use the least squares method.

So, given m strictly increasing real numbers $\{x_i\}_{i=1}^m$ and m points $\{p_i\}_{i=1}^m$. We can construct a B-spline curve of degree d,

$$c(t) = T^d(t)\,\mathbf{c},$$

where the number of control points is $n < m$. Unlike cubic spline interpolation, we now have freedom to make a knot vector independent of the x_i values. Thus, we must make a knot vector such that there is an x_i value inside every B-splines (domain of the basis functions). We now have the following optimal solution for each of the m points,

$$c(x_i) = \sum_{j=0}^{n-1} c_j b_{d,j}(x_i) = p_i.$$

If we organize this in a matrix/vector equation, we get

$$\mathbf{A}\,\mathbf{c} = \mathbf{p}. \tag{6.21}$$

Here $\mathbf{A}$ is a $m \times n$-matrix where each row has a maximum of $d+1$ non-zero elements. We can look at an example, a 2^{nd} degree B-spline curve where we have made a knot vector where $t_0 = t_1 = t_2 = x_1$, $t_n = t_{n+1} = t_{n+2} = x_m$, $t_3 > x_2$ and $t_{n-1} < x_{m-1}$. The equation is,

$$\mathbf{A} = \begin{pmatrix} B_{2,0}(x_1) & 0 & \cdots & \cdots & \cdots & 0 \\ B_{2,0}(x_1) & B_{2,1}(x_1) & B_{2,2}(x_1) & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & B_{2,1}(x_2) & B_{2,2}(x_2) & B_{2,3}(x_2) & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & B_{2,n-4}(x_{m-1}) & B_{2,n-3}(x_{m-1}) & B_{2,n-2}(x_{m-1}) & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \cdots & 0 & B_{2,n-3}(x_m) & B_{2,n-2}(x_m) & B_{2,n-1}(x_m) \\ 0 & \cdots & \cdots & \cdots & 0 & B_{2,n-1}(x_m) \end{pmatrix}, \tag{6.22}$$

and

$$\mathbf{c} = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{pmatrix} \quad \text{and} \quad \mathbf{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_{m-1} \\ p_m \end{pmatrix}.$$

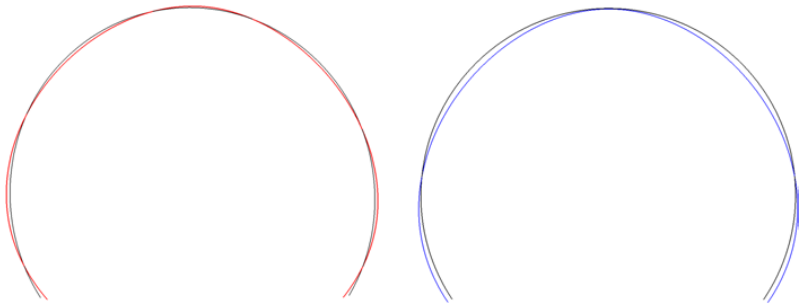


Figure 6.19: *To the left an arc curve in black and a 3^{rd}-degree B-spline curve, in red, made by 60 sample point on the arc and using least square to make the B-spline curve. To the right is the arc curve in black and a B-spline curve in blue. This B-spline curve is made by the least squares method including smoothing and where $\alpha = 1$.*

All rows in $\mathbf{A}$ can be created using Algorithm 3. Remember that one of the parameters in the algorithm is ς, and that in row number i in $\mathbf{A}$ we use x_i as parameter to the basis functions. It follows that ς comes from $t_\varsigma \leq x_i < t_{\varsigma+1}$. If $i = m$ then $\varsigma = n-1$.

Since the matrix $\mathbf{A}$ is not squared and thus not invertible, we can not solve it as it is. So what do we do? We take $\min |\mathbf{A}\ \mathbf{c} - \mathbf{p}|^2$, find the derivative with respect to $\mathbf{c}$, and find when the derivative is 0. That is when $\frac{d}{d\mathbf{c}}(|\mathbf{Ac}|^2 - 2\mathbf{pAc} + |\mathbf{p}|^2) = 0$, which give $2\mathbf{A}^T\mathbf{Ac} - 2\mathbf{A}^T\mathbf{p} = 0$. Thus, to solve the least square expression, we must solve

$$\mathbf{Bc} = \mathbf{y}, \quad \text{where} \quad \mathbf{B} = \mathbf{A}^T\mathbf{A} \quad \text{and} \quad \mathbf{y} = \mathbf{A}^T\mathbf{p}. \tag{6.23}$$

If $m = n$, (6.21) can be solve directly, it is an interpolation, but different from classical cubic spline interpolation. If $m > n$, and usual mush bigger, (6.23) will create a system with an $n \times n$ matrix $\mathbf{B} = \mathbf{A}^T\mathbf{A}$ and a vector with n points $\mathbf{y} = \mathbf{A}^T\mathbf{b}$ which is easy to solve. Especially because the matrix is symmetric around the main diagonal and also diagonal dominant because the most significant basic function is on the diagonal. The matrix is called a positive definite matrix (defined by $\mathbf{x}^T\mathbf{Ax} > 0$ for any nonzero vector $\mathbf{x}$), and can better and faster be solved by using Cholesky- or QR-decomposition instead of LU decomposition. See section about BLAS - B.1, or about Numerical Computations in [78].

In Figure 6.19 is an example given. It's a piece of a circular arc sampled in 60 points. These points together with a vector of parameter values used in the sampling are then used to create a 3^{rd}-degree B-spline curve. This curve is clamped and otherwise has a uniform knot vector. On left hand side, the B-spline curve is made by using least square method. It has 5 control points. In the figure, the arc-curve is black while the B-spline curve is red. A closer study of these two curves shows that the B-spline curve oscillate around the arc curve. This is typical of interpolation and also approximation using least squares method. It is a result of balancing the error.

The degree/order of a B-spline is limiting the shapes of a curve. To understand the problem we can just draw a freehand curve on a sheet of paper, and ask the question; is it possible to find the formula for this curve? The answer is yes, but there is a catch with it, the dimension of the function space must be infinite. If we reduce this dimension to 4,

ie a 3^{rd}-degree B-spline curve then we get an approximation. If we force the curve to go through a set of points or as close as possible, the curve will necessarily begin to oscillate.

Often we want to get a smooth curve that is close enough. This can be done by adding an extra term to the equation, ie try to minimize the square of the curvature, or what is easier, the square of the second derivative. So, in addition we get $c''(t) = \sum_{i=0}^{n-1} b''_{d,i}(t)c_i$, which give $\min|\mathbf{Dc}|^2$, where the matrix $\mathbf{D}$ is similar to matrix $\mathbf{A}$ except that the basis functions are replaced by the 2^{nd} derivatives of the basis functions. We thus get, $\mathbf{Bc} = \mathbf{b}$, where $\mathbf{B} = \mathbf{A}^T\mathbf{A} + \alpha\ \mathbf{D}^T\mathbf{D}$ and $\mathbf{b} = \mathbf{A}^T\mathbf{p}$. Here α is a scalar that determines the weight of the smoothing. On right hand side in Figure 6.19 can we see the same approximation in blue as the one to the left, but here a smoothing with $\alpha = 1$ is added.

Least Square and B-splines

Given m points $\mathbf{p}$, and m parameter values $\mathbf{x}$. Then we can make a B-spline curve of degree d, with $n < m$ control points and a knot vector τ, where all x-values are in the domain, and where the domain of each B-splines includes at least one x-value. We get

$$\mathbf{Bc} = \mathbf{y}, \quad \text{where} \quad \mathbf{B} = \mathbf{A}^T\mathbf{A} + \alpha\ \mathbf{D}^T\mathbf{D} \quad \text{and} \quad \mathbf{y} = \mathbf{A}^T\mathbf{p}.$$

The $m \times n$ matrix $\mathbf{A}$ is according to (6.22), which also applies to matrix $\mathbf{D}$, but where the B-splines are replaced by the 2^{nd} derivative of the B-splines. α is a scalar that determines the smoothing. If $\alpha = 0$ we have an ordinary least square method and can use Algorithm 3 to generate $\mathbf{A}$. Otherwise it is a smoothing least square method and we must use Algorithm 4, 1st row to generate $\mathbf{A}$ and 3rd row to generate $\mathbf{D}$.

6.6 NURBS

NURBS is short for non uniform rational B-splines. Non uniform B-splines are, as typical modern B-splines, defined by knots that can be arbitrary spaced. This is contrary to the uniform B-spline defined (on an implicit integer knot vector) by Schoenberg in [137, 138]. To describe rational B-splines it is necessary to know about homogeneous coordinates, frequently used in graphical systems like OpenGL. A homogeneous coordinate system is connected to a projective space, $\mathbb{P}^n$.[7] A concrete description is that $\mathbb{P}^n$ can be defined as the space of all infinite straight lines in $\mathbb{R}^{n+1}$ going through the origin. The main effect is that there is one extra coordinate compared to an equivalent Euclidian/affine[8] space. We have $q = (x, y, z, w)$, when q is an element in a 3D space. Using this description it follows that $q \in \mathbb{P}^3$ can be expressed by

$$q = (kx, ky, kz, kw),$$

where q is independent of k, i.e. k can be any nonzero real.

[7] Projective space is described in section 2.5 and homogeneous coordinates in section 2.6. For a more thorough study, see e.g. [11] or [30].

[8] Affine spaces are discussed in section 2.4. Affine spaces are spaces of points with associated vectors. The points in the affine space are independent of the origin, and the origin is just one of the points just like any other. See section 2.4. For more deeply studies see e.g. http://en.wikipedia.org/wiki/Affine_space.

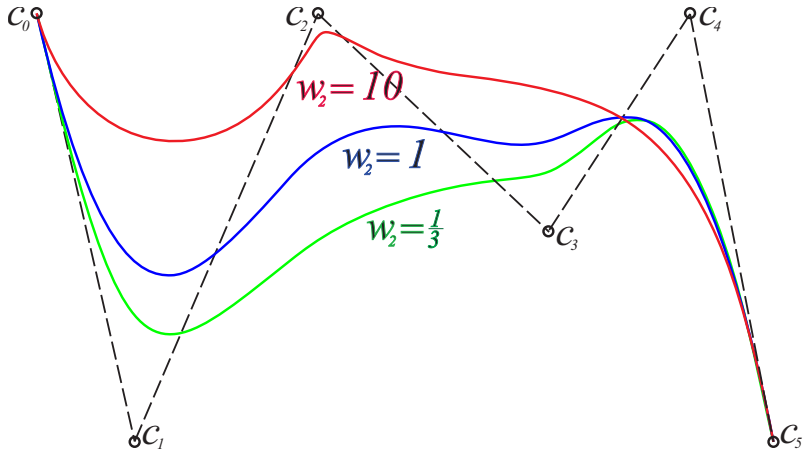


Figure 6.20: *Three versions of a Rational Bézier Curve (NURBS without internal knots) It is only the weight, w_2, of the control point c_2 that has different values in the three examples. The control polygon is the same for all three curves.*

There is a canonical injection of $\mathbb{R}^n$ into $\mathbb{P}^n$. This means that an affine space $\mathbb{R}^n$ can be embedded isomorphically in $\mathbb{P}^n$ by the standard injection

$$(x_1,\ldots,x_n) \mapsto (x_1,\ldots,x_n,1).$$

Affine points can be recovered from projective ones with the mapping

$$(x_1,\ldots,x_n,x_{n+1}) \sim \left(\frac{x_1}{x_{n+1}},\ldots,\frac{x_n}{x_{n+1}},1\right) \mapsto \left(\frac{x_1}{x_{n+1}},\ldots,\frac{x_n}{x_{n+1}}\right).$$

Definition 6.7. *Non uniform rational B-splines (NURBS) are B-splines, defined on a non uniform knot vector, in a projective space and mapped into an affine space. It follows that a B-spline in $\mathbb{P}^n$ is*

$$c(t) = T^d(t)\ \mathbf{c},$$

where each element $c_j,\ j = i-d, i-d+1,\ldots,i$ of $\mathbf{c}$ are given in homogeneous coordinates

$$c_j = w_j(x_{j,1},\ldots,x_{j,n},1).$$

Finally, preparing for mapping from the projective space $\mathbb{P}^n$ to an affine space $\mathbb{R}^n$ gives

$$c(t) = \frac{T^d(t)\ \mathbf{c}}{T^d(t)\ \mathbf{w}}, \tag{6.24}$$

where $\mathbf{w} = (w_{i-d}, w_{i-d+1},\ldots,w_i)^T$, which is actually the NURBS definition.

It follows from expression (6.24) that if $w_i = 1$ for all i, then the NURBS is an ordinary B-spline function in an affine space. To sketch the control polygon the coefficients $c_i,\ i = 0,\ldots,m-1$, where m is the number of coefficients, is mapped from $\mathbb{P}^n$ to $\mathbb{R}^n$, i.e.

$$w_i(x_{i,1},\ldots,x_{i,n},1) \mapsto (x_{i,1},\ldots,x_{i,n}).$$

Figure 6.20 shows a rational 3^{rd}-degree B-spline curve plotted with the weight of the third coefficient having the values $w_2 = 0.1$, $w_2 = 1$ and $w_2 = 10$. The effect is clearly demonstrated, as we can see, a small weight is pushing the curve from the control point, while a big weight is pulling it towards the control point.

6.7 Uniform B-splines and subdivision

Uniform integer-based B-splines, described in (6.2) and (6.3), were the first B-splines developed by Schoenberg. In particular, the symmetry and the fact that all basic functions are similar, only moved in relation to each other, means that knot insertion in the middle between the knots can be simplified as well as a subsequent re-parametrization to integer knots. This leads to subdivision curves.

Subdivision is usually corner cutting, where an initial point set are replaced by a new larger point set where the new points are on the lines between the previous set of points. But subdivision can also be interpolation, where the new points are added to the initial point set and where the new points not necessarily are inside the convex hull of the original point set. In the following subsections we will look at both types of subdivision and we start with interpolation.

6.7.1 Catmull-Rom Subdivision Splines

Catmull-Rom splines are discussed in section 5.6, page 69, and later shown in Figure 6.17, page 99. Catmull-Rom splines interpolates a set of points in such a way that the 1^{st} derivatives at each point are equal to the vector from the point before, to the point after this, scaled by $\frac{1}{2}$. If we consider a uniform and unit-based knot vector, then Catmull-Rom splines can be developed as an interpolating C^1-continues subdivision curve, [20].

Given the knots $\{-1,0,1,2\}$ and the points p_{i-1}, p_i, p_{i+1} and p_{i+2}, then we get, on Lagrange form,

$$\begin{aligned} c(t) &= L_{3,0}(t)p_{i-1} + L_{3,1}(t)p_i + L_{3,2}(t)p_{i+1} + L_{3,3}(t)p_{i+2} \\ &= \frac{t(t-1)(t-2)}{-1(-2)(-2)}p_{i-1} + \frac{(t+1)(t-1)(t-2)}{1(-1)(-2)}p_i \\ &\quad + \frac{(t+1)t(t-2)}{2(1)(-1)}p_{i+1} + \frac{(t+1)t(t-1)}{3(2)(1)}p_{i+2} \\ &= -\frac{1}{6}\left(t^3 - 3t^2 + 2t\right)p_{i-1} + \frac{1}{2}\left(t^3 - 2t^2 - t + 2\right)p_i \\ &\quad - \frac{1}{2}\left(t^3 - t^2 - 2t\right)p_{i+1} + \frac{1}{6}\left(t^3 - t\right)p_{i+2}. \end{aligned} \tag{6.25}$$

If we compute the formula for Catmull-Rom splines at $t = \frac{1}{2}$, halfway between the points p_i and p_{i+1}, we get the 4 points subdivision scheme for Catmull-Rom splines, also called the Dubuc-Deslaurier subdivision scheme,

$$\widehat{p_i} = -\frac{1}{16}p_{i-1} + \frac{9}{16}p_i + \frac{9}{16}p_{i+1} - \frac{1}{16}p_{i+2}, \tag{6.26}$$

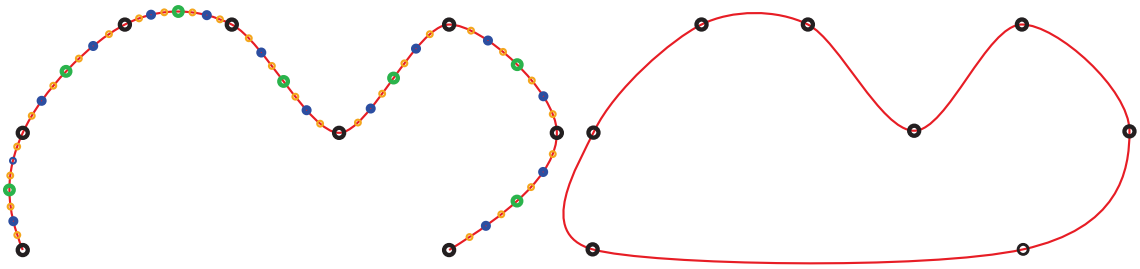


Figure 6.21: *To the left is an open Catmull-Rom subdivision curve, The black circles are the initial points, the green points are the 1^{st} level, the blue are the 2^{nd} level and the orange are the 3^{rd} level of new points. To the right is a closed Catmull-Rom subdivision curve made from the same set of points as the curve to the left.*

where the point $\widehat{p}_i$ is a new point that lies between p_i and p_{i+1}. The procedure is quite clear. Given a point set, we can expand this set by making new points between all the old ones, recursively using the Dubuc-Deslaurier subdivision scheme, until the point set is dense enough. Note that we only use the old point set to generate the new points that should be between the old points, and we keep the old points.

If the curve is closed, ie topological equal to a circle, we implement "the head to bite the tail", see Algorithm 5. However, if the curve is open, ie including a start and an end point, then the scheme does not work at those ends. We can either move away from the ends as we go deeper in the recursion, or changing the scheme at the ends. We use the same formula, (6.25). The index of the first four points are $0,1,2,3$. Because we now want a new point between p_0 and p_1 we change the t-value, ie we compute the formula (6.25) at $t=-\frac{1}{2}$, halfway between the points p_0 and p_1 and get the following scheme,

$$\widehat{p}_0 = \frac{5}{16}p_0 + \frac{15}{16}p_1 - \frac{5}{16}p_2 + \frac{1}{16}p_3, \tag{6.27}$$

where the point $\widehat{p}_0$ is a new point that lies between p_0 and p_1. At the end we can use the same scheme, but turned. If the last point has index n, ie p_n, then

$$\widehat{p}_{n-1} = \frac{5}{16}p_n + \frac{15}{16}p_{n-1} - \frac{5}{16}p_{n-2} + \frac{1}{16}p_{n-3}, \tag{6.28}$$

where the point $\widehat{p}_{n-1}$ is a new point that lies between p_{n-1} and p_n.

A tension parameter was launched in [56]. Introducing $\omega = \frac{1}{16}$, then we get:

$$\widehat{p}_i = \left(\frac{1}{2} + \omega\right)(p_i + p_{i+1}) - \omega\,(p_{i-1} + p_{i+2}), \tag{6.29}$$

that is a reformulation of (6.26). But ω can be changed, and Dyn at al showed in [56] that the curve is C^1-continues if $0 < \omega < \frac{\sqrt{5}-1}{8}$.

A description of an algorithm without a tension parameter follows. There will be some differences between an algorithm for making an open curve and an algorithm for a closed curve. In the algorithm description below, the differences are described after the algorithm description itself, which is basically for an open curve

Algorithm 5. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm calculates a vector of sample points of a curve based on Dubuc-Deslaurier subdivision scheme. There is an algorithm for open curves and changes for closed curves. The input variables are: the initial point set $\{P_i\}_{i=0}^{n}$ *and the level of refinement* d.

Vector⟨Point⟩ **catmulRom(** vector⟨Point⟩ *P,* int d **)**
int $n = P.size - 1$; // *The number of intervals*
int $m = 2^d n$; // *The final number of points*
vector⟨Point⟩ $\Phi(m+1)$; // *The return vector - $m+1$ points.*
for (int *i=0;* $i < P.size$; $i++$)
$\Phi_{2^d i} = P_i$; // *Inserting the initial points*
for (int *j=1;* $j \leq d$; $j++$) // *For each level of refinement*
int $h = 2^{d-j}$;
int $k = 2h$;
$\Phi_h = \frac{5}{16}p_0 + \frac{15}{16}p_k - \frac{5}{16}p_{2k} + \frac{1}{16}p_{3k}$; // *a) - from (6.27)*
for (int *i=1;* $i < n-1$; $i++$)
$\Phi_{ki+h} = -\frac{1}{16}p_{k(i-1)} + \frac{9}{16}p_{ki} + \frac{9}{16}p_{k(i+1)} - \frac{1}{16}p_{k(i+2)}$; // *b) - from (6.26)*
$\Phi_{kn-h} = \frac{5}{16}p_{kn} + \frac{15}{16}p_{k(n-1)} - \frac{5}{16}p_{k(n-2)} + \frac{1}{16}p_{k(n-3)}$ // *c) - from (6.28)*
return Φ;

If the curve is close, we change the number if interval, int $n = P.size$;
after inserting the initial points we add $\Phi_{2^d n} = P_0$;
we replace the line marked a) with $\Phi_h = -\frac{1}{16}p_{k(n-2)} + \frac{9}{16}p_k + \frac{9}{16}p_{2k} - \frac{1}{16}p_{3k}$;
and we replace the line marked c) with $\Phi_{kn-h} = -\frac{1}{16}p_{2k} + \frac{9}{16}p_{kn} + \frac{9}{16}p_{k(n-1)} - \frac{1}{16}p_{k(n-2)}$;

Remember, in programming language of the C-family, 2^d can be implemented as $1 << d$.

On left side of Figure 6.21 is shown an open Catmull-Rom subdivision curve made using the algorithmic description given above. We have $n+1$ ´´black" points where $n = 7$. These are the original points. We use 3 refinement levels, ie $d = 3$. We follow the description above and find that $m = 2^3 7 = 56$, so there will be 57 points when the process is complete. We copy the 8 original points to $q_0, q_8, q_{16}, q_{24}, q_{32}, q_{40}, q_{48}$ and q_{56}. Then at the 1^{st} level is $\widehat{k} = 4$ and $k = 8$. In accordance with the algorithm, 7 new 'green' points are added, with the indices $4, 12, 20, 28, 36, 44$ and 52. At the 2^{nd} level we add 14 new ´´blue" points, with indices $2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50$ and 54. Finally at 3^{rd} level we insert 28 new ´´orange" points with the indices $1, 3, 5, ..., 51, 53$ and 55. As we can see to the left in Figure 6.21, the total number of points is 57. If we have used 4 refinement levels, the total number would have been 113.

On right in Figure 6.21 is shown a closed Catmull-Rom subdivision curve made using the same set of points as the curve on left side in the figure.

6.7.2 Chaikin's algorithms, 2^{nd}-degree subdivision B-splines

We first look at the most simple corner cutting procedure. George Chaikin gave a lecture at the University of Utah in 1974, where he specified a new procedure for generating curves from a limited set of points [21]. This algorithm was among the first refinement

algorithms based on corner cutting. The algorithm refines a point set in such a way that it converges towards a smooth curve. The algorithm is derived from a 2^{nd} degree uniform B-splines [135]. Recall the matric from knot insertion, expression (6.15), and put a new knot in the middle between t_i and t_{i+1},

$$\begin{pmatrix} q_{2i-1} \\ q_{2i} \end{pmatrix} = \begin{pmatrix} 1-w_{2,i-1}(\frac{t_i+t_{i+1}}{2}) & w_{2,i-1}(\frac{t_i+t_{i+1}}{2}) & 0 \\ 0 & 1-w_{2,i}(\frac{t_i+t_{i+1}}{2}) & w_{2,i}(\frac{t_i+t_{i+1}}{2}) \end{pmatrix} \begin{pmatrix} p_{i-1} \\ p_i \\ p_{i+1} \end{pmatrix},$$

where the two new points lie on opposite sides of p_i. If we assume a uniform knot vector of integers, then

$$w_{2,i-1}\left(\frac{t_i+t_{i+1}}{2}\right) = \frac{\frac{1}{2}-(-1)}{2} = \frac{3}{4} \qquad \text{and} \qquad w_{2,i}\left(\frac{t_i+t_{i+1}}{2}\right) = \frac{\frac{1}{2}-0}{2} = \frac{1}{4}$$

and that the entire point set p_i is replaced by the following set of two new points in each of the intervals p_i, p_{i+1},

$$\begin{aligned} q_{2i} &= \frac{3}{4}p_i + \frac{1}{4}p_{i+1} \qquad \text{and} \\ q_{2i+1} &= \frac{1}{4}p_i + \frac{3}{4}p_{i+1}. \end{aligned} \tag{6.30}$$

If the number of intervals is n. It follows that the number of points for an open curve is $n+1$ and for a closed curve n. We set the level of refinement to d. The number of points we get after corner cutting is 2 point for each interval, ie $2n$. This means that for open curves is the number of intervals now $2n-1$, but for closed curves is it $2n$. For closed curves we add a copy of the first point to the end. For d levels of refinements we get,

$$m = 2^d(n-1)+2, \quad \text{for open curves and,} \qquad m = 2^d n + 1, \quad \text{for closed curves.} \tag{6.31}$$

Because of (6.30) we can observe that for an open curve the starting point will be moved from p_0 towards p_1, similarly the end point will be moved from p_n towards p_{n-1}. We can calculate how much the start and end points will move. From (6.30) we see that, with the first corner cutting, the start point will move $x = \frac{1}{4}(p_1 - p_0)$. In the next step, it will move $x = \frac{1}{4}\frac{1}{2}(p_1 - p_0)$ and so it continues so that for d levels of refinement we get,

$$x = \sum_{i=1}^{d} \frac{1}{2^{i+1}}, \quad \text{where } x \text{ is the factor the endpoints will move.} \tag{6.32}$$

A solution of this is to move the start and end points so that the first and the last point in the point set will be at start and end of the curve after the refinement. Note that this is only possible for a 2^{nd}-degree subdivision-curves. This is because a 2^{nd}-degree B-spline curve touch the control polygon at knot values. Thus, we get,

$$p_0 = p_0 + \frac{x}{1-x}(p_0 - p_1) \quad \text{and} \quad p_n = p_n + \frac{x}{1-x}(p_n - p_{n-1}) \tag{6.33}$$

This subdivision scheme is the same as we find in Doo-Sabin surface construction, [54, 55], and it is therefore natural to name the curve as a Doo-Sabin curve. Figure 6.22 shows an example of Doo-Sabin curves, an open curve on the left and a closed curve on the right. Now the algorithms follow.

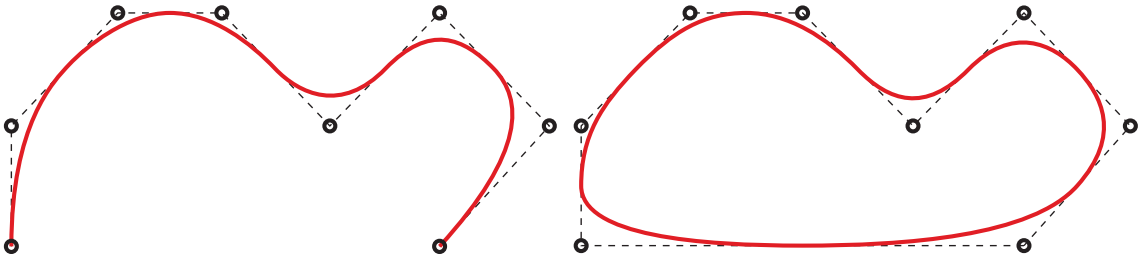


Figure 6.22: *Corner cutting in subdivision curves using Chaikin's algorithms, made from the same set of points as the curve in Figure 6.21. On the left side is an open curve and on the right side is a closed curve. The black circles are the initial points.*

Algorithm 6. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm calculates a vector of sample points of a curve based on Chaikin's algorithm. There is an algorithm for open curves and one for closed curves. The input variables are: the initial point set $\{P_i\}_{i=0}^{n}$ *and the level of refinement* d.

vector⟨Point⟩ **ChaikinOpen(** vector⟨Point⟩ P, int d **)**

```
int  n = P.size − 1;                    // The number of intervals
int  m = 2^d (n − 1) + 2;               // The final number of points, from (6.31)
vector⟨Point⟩  Φ(m);                    // The return vector - m points.
double  x = Σ_{i=1}^{d} 1/2^{i+1};      // the factor to move the endpoints, see (6.32)
Φ_0 = P_0 + x/(1−x) (P_0 − P_1);        // Moving the first point according to (6.33)
for ( int  i=1; i < n; i++ )
    Φ_i = P_i;                          // Inserting the initial points
Φ_n = P_n + x/(1−x) (P_n − P_{n−1});    // Moving the last point according to (6.33)
for ( int  j=1; j ≤ d; j++ )            // For each level of refinement
    for ( int  i=n-1; i ≥ 0; i−− )
        P_{2i} = 3/4 P_i + 1/4 P_{i+1};     // Making new points according to (6.30)
        P_{2i+1} = 1/4 P_i + 3/4 P_{i+1};
    n = 2n − 1;                         // The number of intervals in the next level
return Φ;
```

vector⟨Point⟩ **ChaikinClosed(** vector⟨Point⟩ P, int d **)**

```
int  n = P.size;                        // The number of intervals
int  m = 2^d n + 1;                     // The final number of points, from (6.31)
vector⟨Point⟩  Φ(m);                    // The return vector - m points.
for ( int  i=0; i < n; i++ )
    Φ_i = P_i;                          // Inserting the initial points
Φ_n = P_0;                              // Copy the first point to the end
for ( int  j=1; j ≤ d; j++ )            // For each level of refinement
    for ( int  i=n-1; i ≥ 0; i−− )
        P_{2i} = 3/4 P_i + 1/4 P_{i+1};     // Making new points according to (6.30)
        P_{2i+1} = 1/4 P_i + 3/4 P_{i+1};
    Φ_n = Φ_0;                          // Copy the first point to the end
    n = 2n;                             // The number of intervals in the next level
return Φ;
```

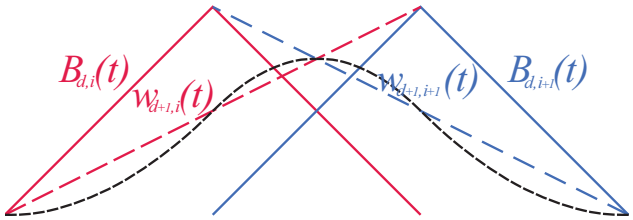


Figure 6.23: *A B-spline $B_{d+1,i}(t)$, dotted black, is made from a sum $w_{d+1,i}(t)B_{d,i}(t)$, in red, and the symmetric product of the neighboring B-spline of degree d, in blue.*

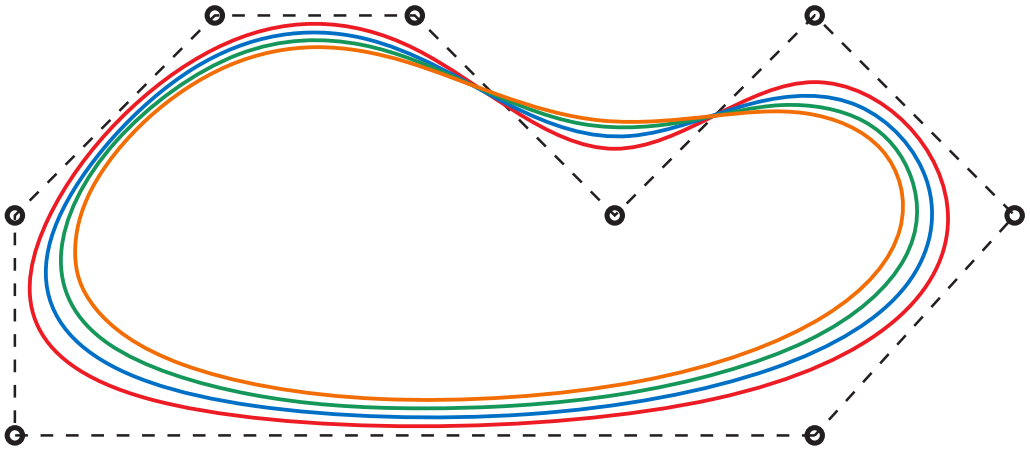


Figure 6.24: *We see four closed subdivision curves that are all generated from the same set of points as the curves in Figures 6.21 and 6.22. In red we see a 3^{rd}-degree curve, in blue a 4^{th}-degree curve, in green a 5^{th}-degree curve and in orange a 6^{th}-degree curve. The black circles are the initial points.*

6.7.3 Lane-Riesenfeld subdivision algorithm

However there is an easier and also more general way to develop subdivision schemes for Uniform B-splines. In 1980, the Lane-Riesenfeld subdivision algorithm for uniform B-splines was launched, [110]. This algorithm reproduce closed uniform B-spline curves and open curves that are not clamped. It is a compact and very simple and thus elegant algorithm, and follows from symmetry around the half knot values, and that $B_{d+1,i}(t) = w_{d+1,i}(t)B_{d,i}(t) + (1 - w_{d+1,i+1}(t))B_{d,i+1}(t)$ as illustrated in Figure 6.23. The algorithm is divided into two parts. First doubling the point set, which can be done by making two of each point or inserting new points in the midpoint between all the points. The next step is the smoothing/degree raising. This is done by replacing the points with new points in the midpoint between all the old points, ie $p_i = \frac{1}{2}(\tilde{p}_i + \tilde{p}_{i+1})$. This step is repeated to the desired degree. We have now doubled the number of points. The whole procedure of doubling and smoothing are now repeated until we have a sufficient number of points.

In Figure 6.24, four curves are made using Lane-Riesenfeld subdivision algorithm. The curves are closed uniform subdivision B-splines of polynomial degree 3, 4, 5 and 6 made of the same set of 8 points as in Figures 6.21 and 6.22. The Lane-Riesenfeld subdivision algorithm now follows.

Algorithm 7. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm calculates a vector of sample points of a curve based on Lane-Riesenfeld algorithm. The algorithm is divided in closed curves and open but not clamped curves. There are help functions for the two parts. The input variables are: the initial point set $\{P_i\}_{i=0}^{n-1}$ *and the level of refinement k and the degree d.*

```
vector⟨Point⟩ LaneRiesenfeldOpen( vector⟨Point⟩ P, int k, int d )
    int n = P.size;                          // The number of intervals
    int m = 2^k(n−d)+d;                      // The final number of points
    vector⟨Point⟩ Φ(m+d−1);                  // The return vector – m points.
    for ( int i=0; i < n; i++ )
        Φ_i = P_i;                           // Inserting the initial points
    for ( int i=0; i < k; i++ )              // For each level of refinement
        n = doublePart(Φ, n);
        smoothPartOpen(Φ, n, d);
    Φ.resize(m);
    return Φ;

vector⟨Point⟩ LaneRiesenfeldClosed( vector⟨Point⟩ P, int k, int d )
    int n = P.size;                          // The number of intervals
    int m = 2^k n+1;                         // The final number of points
    vector⟨Point⟩ Φ(m);                      // The return vector – m points.
    for ( int i=0; i < n; i++ )
        Φ_i = P_i;                           // Inserting the initial points
    Φ_n = P_0;                               // Closing the curve
    for ( int i=0; i < k; i++ )              // For each level of refinement
        n = doublePart(Φ, n);
        smoothPartClosed(Φ, n, d );
    return Φ;

int doublePart( vector⟨Point⟩& P, int n )
    for ( int i=n-1; i > 0; i−− )
        P_2i = P_i;
        P_2i−1 = 1/2(P_i + P_i−1);
    return 2n − 1;

void smoothPartOpen( vector⟨Point⟩& P, int& n, int d )
    for ( int j=1; j < d; j++, n−− )
        for ( int i=0; i < n − 1; i++ )
            P_i = 1/2(P_i + P_i+1);

void smoothPartClosed( vector⟨Point⟩& P, int n, int d )
    for ( int j=1; j < d; j++ )
        for ( int i=0; i < n − 1; i++ )
            P_i = 1/2(P_i + P_i+1);
        P_n−1 = P_0;
```

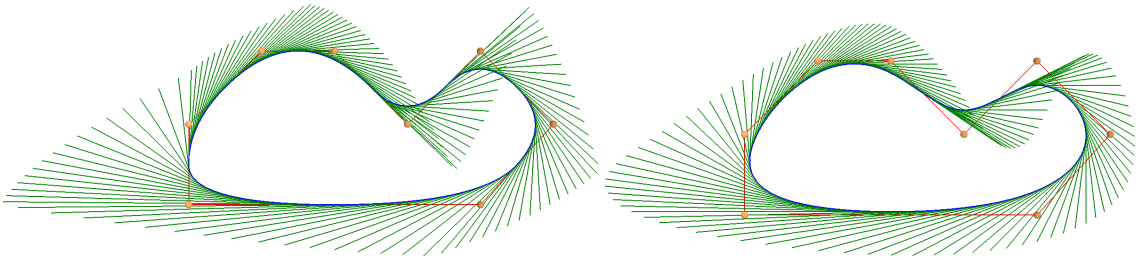


Figure 6.25: *To the left is a Doo-Sabin curve and it's 1^{st} derivative in green. To the right is a Catmull-Clak curve and it's 1^{st} derivative. They are both made from the same point set.*

As the reader probably now recognize, it is possible to replace the nested loops in the smooth-part with a scheme that is produced by Pascals triangle, ie if the degree is 3 we can use $P_i = \frac{1}{4}(p_i + 2p_{i+1} + p_{i+2})$, or if the degree is 4, $P_i = \frac{1}{8}(p_i + 3p_{i+1} + 3p_{i+2} + p_{i+3}))$. It is also possible to optimize by merging the two parts and develop specific schemes for given degrees. If we look at a degree 3 curve, we see that we have two levels. Due to the doubling of points, we have 3 and 3 points that lie on a straight line. Therefore, due to 2 levels, every second point will be equal to the midpoints that were generated. The other points will then be $Q_i + 2Q_{i+1} + Q_{i+2}$ where Q_i and Q_{i+2} will be midpoints. We then get $P_j = \frac{1}{4}\left(\frac{1}{2}(P_{i-1} + P_i) + 2P_i + \frac{1}{2}(P_i + P_{i+1})\right)$. This leads to Catmull-Clark curves.

Catmull-Clark curves, named after Catmull-Clark surfaces, [19], curves are based on 3^{rd}-degree uniform B-splines. The algorithm is an optimized Lane-Riesenfeld algorithm where both the doublePart() and smoothPartOpen() functions are replaced by,

int **CatmullClarkOpen(** *vector⟨Point⟩& P, int n* **)**
for (*int* $i = n-2; i > 1; i--$)
$P_{2i} = \frac{1}{2}(P_i + P_{i+1})$;
$P_{2i-1} = \frac{1}{8}(P_{i-1} + 6P_i + P_{i+1})$;
Point q = $\frac{1}{8}(P_0 + 6P_1 + P_2)$;
$P_2 = \frac{1}{2}(P_1 + P_2)$;
$P_0 = \frac{1}{2}(P_0 + P_1)$;
$P_1 = q$;
return $2n - d$;

or if closed where both the doublePart() and smoothPartClosed() functions are replaced by,

int **CatmullClarkClosed(** *vector⟨Point⟩& P, int n* **)**
$P_{2n} = \frac{1}{8}(P_{n-1} + 6P_0 + P_1)$;
for (*int* $i = n-1; i > 0; i--$)
$P_{2i+1} = \frac{1}{2}(P_i + P_{i+1})$;
$P_{2i} = \frac{1}{8}(P_{i-1} + 6P_i + P_{i+1})$;
$P_1 = \frac{1}{2}(P_0 + P_1)$;
$P_0 = P_{2n}$;
return $2n$;

In Figure 6.25, to the left is there a plot of a Doo-Sabin curve and to the right a Catmull-Clark curve. Also the 1^{st} derivatives are plotted for both curves. The derivatives are generated using divided differences. We can clearly see that the Doo-Sabin curve on the left is $C^1 - smooth$, and the Catmull-Clark curve on the right is at least $C^2 - smooth$. This is of course in accordance with the continuity properties of B-splines.

Subdivision is widely used, especially in computer graphics. Here is just a small excerpt of articles in the field that go beyond what we have gone through; [56, 58, 87, 70, 57, 112, 154].

Chapter 7

Blending

Bézier and B-splines are based on blending of points, Hermite curves are based on blending of points and vectors, Coons Patch [26] are actually blending of Surfaces, and Gordon Surfaces [79] is also using blending of curves and surfaces. There is a lot of work done on blending, examples are [160], [86], [148]. It is therefore of interest to investigate blending in more detail, and especially look at blending technics to blend functions in general. We start with defining the B-function.

7.1 B-functions

B-function is an abbreviation for blending function. It is for blending functions, whether they are based on scalar-, vector- or point-values, such as points, curves, tensor product or triangular surfaces etc. In the following, we restrict B functions to be monotonous and we will look especially at symmetric B-functions and what it means. The definition is:

Definition 7.1. *A B-function is:*

– D1	*a homeomorphism ("permutation function")*	$B : I \to I \quad (I = [0,1] \subset \mathbb{R})$,
– D2	*and thus is*	$B(0) = 0$,
– D3	*and*	$B(1) = 1$,
– D4	*and that is monotone, i.e.*	$B'(t) \geq 0, \quad t \in I$.
D5	*A B-function is called symmetric if,*	$B(t) + B(1-t) = 1, \quad t \in I$.

This symmetry is a point symmetry, around the point $(0.5\ 0.5)$. Other types of symmetry will be introduced in sections 7.8. A more general definition of a B-function is given i Definition 13.1. To give an idea of what a B-function is, we will look at four simple examples of symmetric B-functions:

a)	linear function	$B(t) = t$
b)	trigonometric function	$B(t) = \sin^2 \frac{\pi t}{2}$
c)	polynomial function of first order	$B(t) = 3t^2 - 2t^3$
d)	rational function of first order	$B(t) = \frac{t^2}{t^2 + (1-t)^2}$

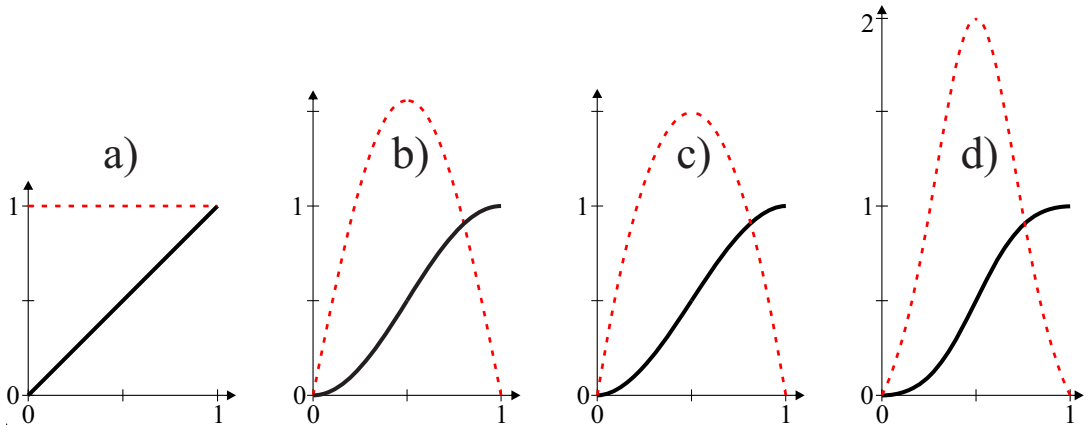


Figure 7.1: *Four B-functions (solid black) and their derivatives (dashed red). From left is a) $B(t) = t$, b) $B(t) = \sin^2 \frac{\pi t}{2}$, c) $B(t) = 3t^2 - 2t^3$ and d) $B(t) = \frac{t^2}{(1-t)^2+t^2}$.*

Figure 7.1 shows the four examples of B-functions plotted together with their 1^{st} derivatives. We can clearly see from the figure that these four functions all starts at 0 and ends at 1 and that they are all monotone. That they are symmetrical follows from that,

a) $B(t) + B(1-t) = t + 1 - t = 1$,

b) $B(t) + B(1-t) = \sin^2 \frac{\pi t}{2} + \sin^2 \frac{\pi(1-t)}{2} = 1$,

c) $B(t) + B(1-t) = 3t^2 - 2t^3 + 3(1-t)^2 - 2(1-t)^3 = 1$,

d) $B(t) + B(1-t) = \frac{t^2}{t^2+(1-t)^2} + \frac{(1-t)^2}{(1-t)^2+t^2} = 1$.

B-functions can be organized in groups. All of the above examples are members of groups of B-functions. Later, we will conduct a thorough investigation of some of the groups and their special properties. But first let's look at the definition of an important property.

Definition 7.2. *A property that plays an important role is the number of subsequent derivatives that is zero at start and end. We call this*

The order of B-functions

The order of a B-function, short for the Hermite order of a B-function, denoted S, is for a symmetric B-function determined by

$$B^{(j)}(0) = B^{(j)}(1) = 0, \quad j = 1, 2, ..., S. \tag{7.1}$$

For a non-symmetric B-function we have to differ between the start and the end,

$$\begin{aligned} B^{(j)}(0) &= 0, \quad j = 1, 2, ..., S_0. \\ B^{(j)}(1) &= 0, \quad j = 1, 2, ..., S_1. \end{aligned} \tag{7.2}$$

This is the Hermite property to a B-function, explained further in Theorem 7.1.

For the examples b), c) and d) are $B'(0) = B'(1) = 0$, i.e. the first derivative is zero at start and end, but the second derivative is not. It follows that these are 1^{st}-order B-functions.

Later in this chapter we will look at higher order B-functions, and also complete B-functions where all derivatives are zero at both start and end.

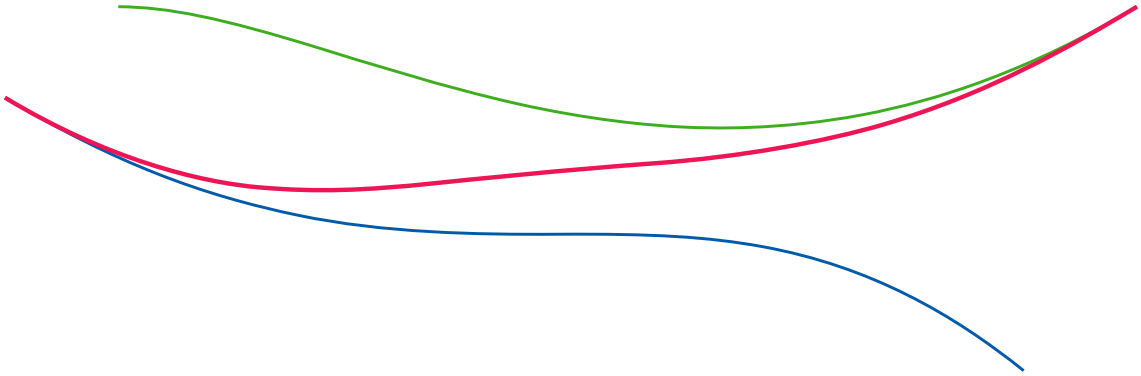


Figure 7.2: *In the figure is two Bézier-curves blended to one curve using a B-function. The original Bézier-curves are shown in blue and green and the resulting curve is shown in red.*

7.2 Blending of two functions

The simplest blending is blending of two functions/curves, $g_1(t)$ and $g_2(t)$, organized in a sequence where $g_1(t)$ is the first function and $g_2(t)$ is the last function, and both with domain $[0,1]$. In Figure 7.2 is an example shown. Two Bézier-curves are plotted, $g_1(t)$ in blue and $g_2(t)$ in green. The result of the blending is shown as a red curve in the figure. The B-function used here is $B(t) = 3t^2 - 2t^3$. What the result curve look like follows from the sequence of the two initial curves together with the order of the B-function used in the blending. This will be further discussed later. First we look at the formulas.

The formulas for a two functions blending

The formulas for a two functions blending is

$$\begin{aligned} f(t) &= (1 - B(t))\, g_1(t) + B(t)\, g_2(t) \\ &= g_1(t) + B(t)\, (g_2(t) - g_1(t)). \end{aligned} \tag{7.3}$$

where $B(t)$ is a B-function.
If we denote the difference function for $h(t) = g_2(t) - g_1(t)$, we get the formula

$$f(t) = g_1(t) + B(t)\, h(t)). \tag{7.4}$$

The 1^{st} order derivative is

$$f'(t) = g_1'(t) + B(t)\, h'(t) + B'(t)\, h(t), \tag{7.5}$$

and the formula for derivatives of all orders is

$$f^{(j)}(t) = g_1^{(j)}(t) + \sum_{i=0}^{j} \binom{j}{i} B^{(i)}(t)\, h^{(j-i)}(t). \tag{7.6}$$

In Figure 7.3, the speed of the curves from Figure 7.2 is plotted. There we can see that the speed of the resulting curve is the same as the speed of the first curve at start, ie $|f'(0)| = |g_1'(0)|$, and the speed of the resulting curve is the same as the speed of the

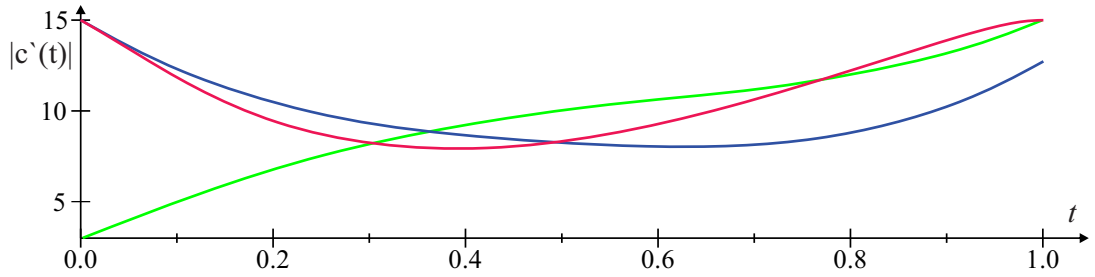


Figure 7.3: *The functions shows the speeds, $|c'(t)|$, to the two Bézier-curves and the resulting blended curve from Figure 7.2. The speed-function plots also have the same colors as the corresponding curves in Figure 7.2.*

second curve at end, ie $|f'(1)| = |g_2'(1)|$.

The relationship between the two initial functions called local functions and the resulting function called the global function, and then especially what happens at the start and at the end of the functions is of special interest. The following theorem provides a description of this.

Theorem 7.1. *A two-functions blending by B-function has the following property*

The Hermite interpolation property

In a two-functions blending using a B-function, $f(t) = g_1(t) + B(t)\ (g_2(t) - g_1(t))$, it follows that the global function interpolate the first local function at the start point with position and subsequently derivatives up to order S_0, i.e.

$$f^{(j)}(0) = g_1^{(j)}(0), \quad j = 0, 1, ..., S_0, \tag{7.7}$$

and the global function interpolate the last local function at the end point with position and subsequently derivatives up to order S_1, i.e.

$$f^{(j)}(1) = g_2^{(j)}(1), \quad j = 0, 1, ..., S_1, \tag{7.8}$$

where S_0 and S_1 are the (Hermite) orders at start and end of the current B-function.

Proof. First we look at (7.7). ie at the start of the functions where $t = 0$. From **D2** in Definition 7.1 and from (7.2) we see that $B^j(0) = 0$ for $j = 0, 1, ..., S$. Thus it follows from (7.4) and (7.6) that $f^{(j)}(0) = g_1^{(j)}(0)$ for $j = 1, ..., S_0$.

Proving (7.8) is based on a similar reasoning as above due to the symmetry. At the end is $t = 1$. Recall from **D3** in Definition 7.1 that $B(1) = 1$. From (7.3) it follows that $f(1) = g_2(1)$. If we rewrite (7.6) by separating out the first term of the sum we get

$$f^{(j)}(t) = g_1^{(j)}(t) + B(t)\ (g_2^{(j)}(t) - g_1^{(j)}(t)) + \sum_{i=0}^{j} \binom{j}{i} B^{(i)}(t)\ h^{(j-i)}(t).$$

Since $B(1) = 1$ and $B^{(j)}(1) = 0$ is $f^{(j)}(1) = g_2^{(j)}(1)$ for $j = 1, ..., S_1$. □

The Hermite interpolation property is also influenced by the behavior of the local functions. This can be summarized in the following theorem.

Theorem 7.2. *The Hermite interpolation property is influenced by*

The extended Hermite interpolation property

In a two-function blending, if the two local functions are equal at the start, i.e. $g_1(0) = g_2(0)$*, then the order of the Hermite interpolation increase by 1 at the start, ie*

$$f^{(j)}(0) = g_1^{(j)}(0), \quad j = 0, 1, ..., S_0 + 1, \tag{7.9}$$

and if the two local functions is equal at end, i.e. $g_1(1) = g_2(1)$*, then the order of the Hermite interpolation increases by 1 at end, ie*

$$f^{(j)}(1) = g_2^{(j)}(1), \quad j = 0, 1, ..., S_1 + 1, \tag{7.10}$$

where S_0 *and* S_1 *are the (Hermite) orders at start and end of the current B-function.*

In general, if the two local functions have position and subsequently derivatives equal at start, i.e. $g_1^{(j)}(0) = g_2^{(j)}(0),\ j = 0, 1, ...d_0$*, then the order of the Hermite interpolation increase by* $d_0 + 1$ *at start, i.e.*

$$f^{(j)}(0) = g_1^{(j)}(0), \quad j = 0, 1, ..., S_0 + d_0 + 1, \tag{7.11}$$

and if the two local functions have position and subsequently derivatives equal at end, i.e. $g_1^{(j)}(1) = g_2^{(j)}(1),\ j = 0, 1, ...d_1$*, then the order of the Hermite interpolation increase by* $d_1 + 1$ *at end, i.e.*

$$f^{(j)}(1) = g_2^{(j)}(1), \quad j = 0, 1, ..., S_1 + d_1 + 1, \tag{7.12}$$

where S_0 *and* S_1 *are the (Hermite) orders at start and end of the current B-function.*

Proof. If we rewrite (7.6) by taking out the last term of the sum in the expression we get

$$f^{(j)}(t) = g_1^{(j)}(t) + \sum_{i=0}^{j-1} \binom{j}{i} B^{(i)}(t)\ h^{(j-i)}(t) + B^{(j)}(t)\ (g_2(t) - g_1(t)).$$

Since $g_2(0) - g_1(0) = 0$, then it follows that the sum and the last term is also zero for $j = 0, 1, \ldots, S_0 + 1$ because $B^{(i)}(0) = 0$ for $j = 0, 1, \ldots, S_0$.

The same type of argument is also valid for the end of the curve, which concludes the proof of the first part of the theorem.

If we rewrite (7.6) by taking out the d last terms of the sum in the expression we get

$$f^{(j)}(t) = g_1^{(j)}(t) + \sum_{i=0}^{j-d-1} \binom{j}{i} B^{(i)}(t)\ h^{(j-i)}(t) + \sum_{i=j-d}^{j} \binom{j}{i} B^{(i)}(t)\ (g_2^{(j-i)}(t) - g_1^{(j-i)}(t)).$$

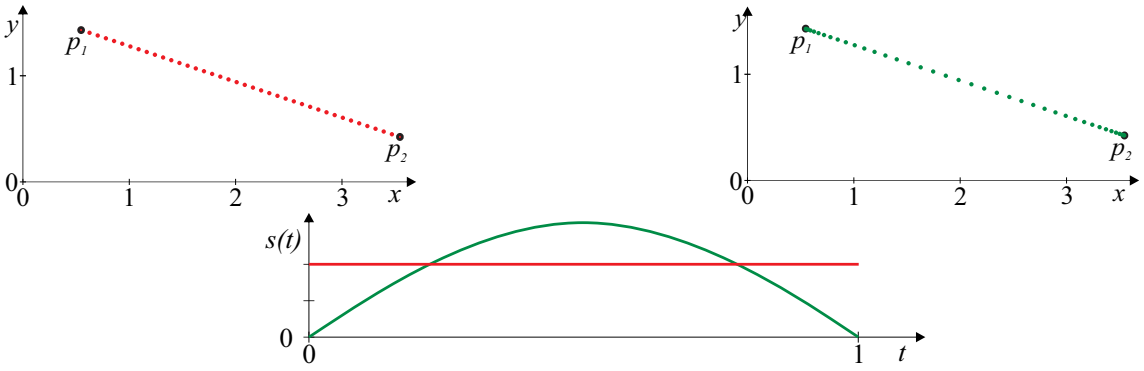


Figure 7.4: *The figure at the top left shows a curve with linear blending of the two points p_1 and p_2. The red curve is plotted using equal spaced (in the parameter line) dots to show the constant speed. On the right side is the trigonometric B-function used in the blending (the green curve) and we can clearly see that the speed is not constant. Below, the speed is plotted. The red is the constant speed, the green starts and ends at zero. The integral of the two functions is the same because the curve length is the same.*

Because $B^{(i)}(0) = 0$, the penultimate sum is also zero for $i = 0,1,...,S_0$. And, since $j-i = d,d-1,\ldots,0$ when $i = j-d,j-d+1,\ldots,j$ is also $g_1^{(j-i)}(0) - g_2^{(j-i)}(0) = 0$ when $j-i = 0,1,...d$. Thus it follows that the order of the derivatives that is zero goes up to S_0+d+1.

The same argument is valid also for $t = 1$, the end of the curve, which concludes the proof of the second part of the theorem. □

7.2.1 Examples, blending of order zero and order one

Example with blending of two points (Figure 7.4):
The most classical B-function is the linear blending function, $B(t) = t$, best known from linear interpolation. On left hand side in Figure 7.1 this, order zero B-function is plotted together with its 1^{st} derivative, and on the left in Figure 7.4 we can see a curve which is a linear blending of two points, p_1 and p_2. From formula (7.4) we get

$$\begin{aligned} c(t) &= p_1 + t\,(p_2 - p_1), \\ c'(t) &= p_2 - p_1. \end{aligned}$$

As we can see, the derivative $c'(t)$ is a constant, and the velocity/speed of the curve is thus the same everywhere. This can be clearly seen in Figure 7.4 where the red curve is plotted with an evenly distributed point sequence at the top left, and the the red line which is a plot of the velocity/speed at the bottom of the figure.

On the upper right side in Figure 7.4 there is a linear curve which is made from formula (7.4) and where the B-function is the trigonometric $B(t) = \sin^2 \frac{\pi}{2}t$ of order 1. This gives

$$\begin{aligned} c(t) &= p_1 + \left(\sin^2 \tfrac{\pi}{2}t\right)(p_2 - p_1) \\ c'(t) &= \left(\pi \cos \tfrac{\pi}{2}t \sin \tfrac{\pi}{2}t\right)(p_2 - p_1) \end{aligned}$$

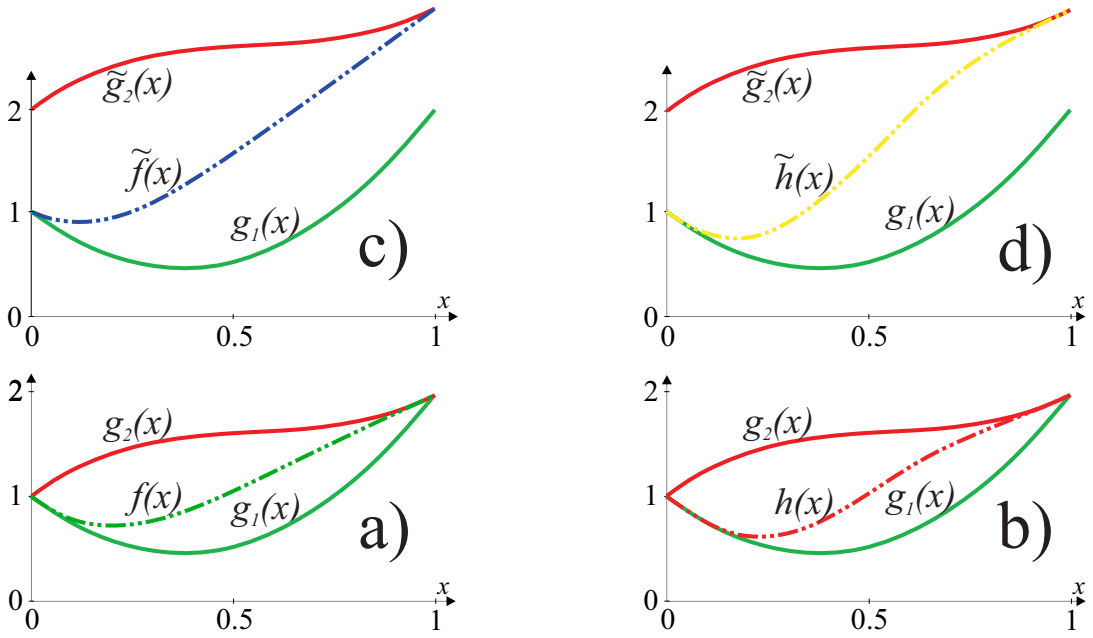


Figure 7.5: *All green curves are $g_1(x)$, expression (7.13). The red curves in the two lower plots are $g_2(x)$, expression (7.14). The red curves in the two upper plots are $\widetilde{g}_2(x)$, expression 7.15. The blending on left hand side are using a linear B-function of order 0, while the blending on right hand side are using a trigonometric B-function of order 1.*

From the formula it is clear that the speed is not constant. The speed is zero at start and end because $\sin 0 = 0$ and $\cos \frac{\pi}{2} = 0$. In Figure 7.4 on the upper right side, the dots in the green curve are not uniformly distributed, the density of the dots is greatest at both ends. This is confirmed in the plot at the bottom of Figure 7.4. There we see a green function describing the speed of the curve over the domain.

Example with blending of two functions (Figure 7.5):
In the next example we blend two functions. The first function is

$$g_1(x) = 4x^2 - 3x + 1 \tag{7.13}$$

which is shown as a green curve in Figure 7.5. The second function is

$$g_2(x) = 3x^3 - 5x^2 + 3x + 1, \tag{7.14}$$

which is shown as a red curve in plot a) and plot b) in Figure 7.5. In plot c) and d), g_2 is moved 1 upwards, which gives

$$\widetilde{g}_2(x) = 3x^3 - 5x^2 + 3x + 2. \tag{7.15}$$

In the examples we use two different B-function, the zero order linear B-function and the first order trigonometric B-function. Using the zero order linear B-function we get

$$\begin{aligned} f(x) &= g_1(x) + x\,(g_2(x) - g_1(x)), \\ f'(x) &= g_1'(x) + x\,(g_2'(x) - g_1'(x)) + g_2(x) - g_1(x), \\ f''(x) &= g_1''(x) + x\,(g_2''(x) - g_1''(x)) + 2(g_2'(x) - g_1'(x)). \end{aligned}$$

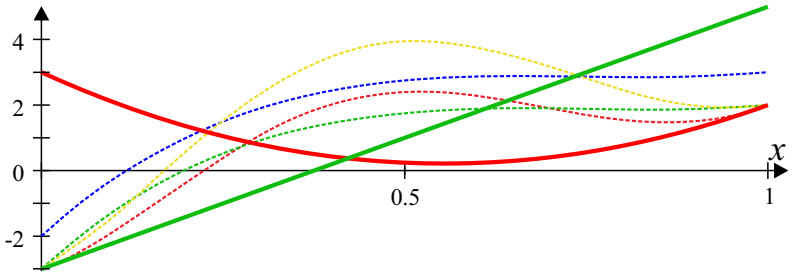


Figure 7.6: *The first derivative, $g_1'(x)$-solid green, $g_2'(x)$-solid red and the resulting curve $f'(x)$-dashed green , $h'(x)$-dashed red, $\widetilde{f}'(x)$-dashed blue and $\widetilde{h}'(x)$-dashed yellow.*

Using the first order trigonometric B-function we get

$$\begin{aligned}
h(x) &= g_1(x) + \sin^2 \tfrac{\pi}{2}x(g_2(x) - g_1(x)) \\
&= \tfrac{1}{2}(g_1 + g_2 + (g_1 - g_2)\cos \pi x), \\
h'(x) &= \tfrac{1}{2}\left((g_1' + g_2') + (g_1' - g_2')\cos \pi x - \pi\,(g_1 - g_2)\sin \pi x\right), \\
h''(x) &= \tfrac{1}{2}\left((g_1'' + g_2'') + \left((g_1'' - g_2'') - \pi^2\,(g_1 - g_2)\right)\cos \pi x - 2\pi\,(g_1' - g_2')\sin \pi x\right).
\end{aligned}$$

Since we use two blending functions and two second functions g_2 and $\widetilde{g_2}$, we get 4 resulting curves $f(x)$, $\widetilde{f}(x)$, $h(x)$ and $\widetilde{h}(x)$. These four curves are shown in Figure 7.5 as dashed curves. Observe that the curve in plot c) only interpolates the position of g_1 at start and g_2 at end (order 0). The curves in plot a) and d) also interpolate the first derivative (order 1), and the curve in plot b) also interpolates the second derivative (order 2).

To verify this observation, the first and second derivatives are calculated in the table below. Note that all derivatives are equal for $\widetilde{g_2}$ and g_2. The first and second derivatives at start of g_1 and end of g_2, and at start and end of the resulting curves f, h, $\widetilde{f}$ and $\widetilde{h}$ are calculated:

g_1-green, g_2-red	$g_1'(0) = -3$	$g_2'(1) = 2$	$g_1''(0) = 8$	$g_2''(1) = 8$	order
a) $f(x)$-green	$f'(0) = -3$	$f'(1) = 2$	$f''(0) = 20$	$f''(1) = 2$	1
b) $h(x)$-red	$h'(0) = -3$	$h'(1) = 2$	$h''(0) = 8$	$h''(1) = 8$	2
c) $\widetilde{f}(x)$-blue	$\widetilde{f}'(0) = -2$	$\widetilde{f}'(1) = 3$	$\widetilde{f}''(0) = 20$	$\widetilde{f}''(1) = 2$	0
d) $\widetilde{h}(x)$-yellow	$\widetilde{h}'(0) = -3$	$\widetilde{h}'(1) = 2$	$\widetilde{h}''(0) = 8 + \frac{\pi^2}{2}$	$\widetilde{h}''(1) = 8 - \frac{\pi^2}{2}$	1

The letter and the color in first column in the table is refereing to Figure 7.5 and also to the color of the first derivatives in Figure 7.6. The red numbers in the table indicate match to g_1 at $x = 0$ and to g_2 at $x = 1$. The order in the last column is Hermite order of the B-function. In Figure 7.6 is $g_1'(x)$ solid green, $g_2'(x) = \widetilde{g}_2'(x)$ is solid red. $f'(x)$ is dashed green, $h'(x)$ is dashed red, $\widetilde{f}'(x)$ is dashed blue and $\widetilde{h}'(x)$ is dashed yellow.

7.2.2 Examples, connecting two curves by using a B-function

Now we look at an example where two separate curves are smoothly connected using a B-function. There are a number of articles on this topic and the topic in the previous subsection, such as [166], [121] and [86]. We shall here use a spline concept and thus a knot vector. We start with two 3^{rd}-degree Bézier-curves embedded in $\mathbb{R}^2$,

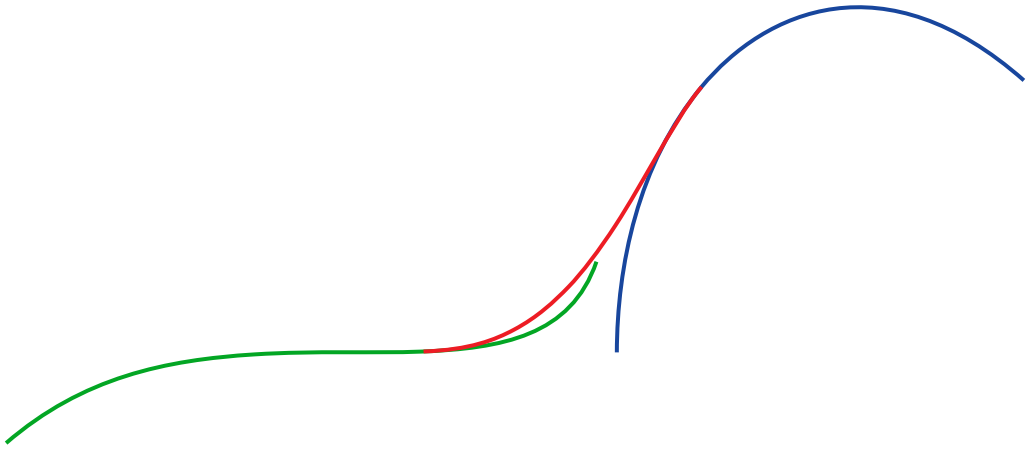


Figure 7.7: *In the figure, two curves are connected using a B-function. The green curve $c_1(t)$ is connected to the blue curve $c_3(t)$ with the red curve $c_2(t)$ which connects the two original curves in a smooth way, C^1 continuity.*

$$c_1(t) = (1-t)^3 \begin{pmatrix} 1 \\ 1 \end{pmatrix} + 3t(1-t)^2 \begin{pmatrix} 2 \\ 2 \end{pmatrix} + 3t^2(1-t) \begin{pmatrix} 3.6 \\ 1 \end{pmatrix} + t^3 \begin{pmatrix} 3.9 \\ 2 \end{pmatrix}, \quad t \in [0,1],$$

and

$$c_3(t) = (1-t)^3 \begin{pmatrix} 4 \\ 1.5 \end{pmatrix} + 3t(1-t)^2 \begin{pmatrix} 4 \\ 3 \end{pmatrix} + 3t^2(1-t) \begin{pmatrix} 5 \\ 4 \end{pmatrix} + t^3 \begin{pmatrix} 6 \\ 3 \end{pmatrix}, \quad t \in [0,1].$$

These two curves are plotted in Figure 7.7, $c_1(t)$ in green on left hand side, and $c_3(t)$ in blue on right hand side. If we use 40% of each of the original curves for the "joining curve", we get the knot vector $\{t_i\}_{i=0}^3 = \{0,\ 0.6,\ 1,\ 1.6\}$, and we get the following formula for the new piece of curve,

$$c_2(t) = c_1(t) + B \circ w_{1,1}(t)(c_3(t-t_1) - c_1(t)), \quad t_1 \le t < t_2.$$

where $B(t)$ is a B-function, and $w_{1,i}(t) = \frac{t-t_i}{t_{i+1}-t_i}$, see (6.11). In this example we use $B(t) = 3t^2 - 2t^3$. This new curve segment $c_2(t)$ is shown as the red curve in Figure 7.7.

The total curve is now,

$$f(t) = \begin{cases} c_1(t), & \text{if } \ t_0 \le t < t_1, \\ c_2(t), & \text{if } \ t_1 \le t < t_2, \\ c_3(t-t_1), & \text{if } \ t_2 \le t \le t_3. \end{cases} \tag{7.16}$$

The domain of $f(t)$ defined in (7.16) is $[t_0, t_3]$. The curve is C^1-smooth because we used a B-function of order 1, see Theorem 7.1. The resulting curve can be seen in Figure 7.7 as a subsequent green, red and blue curve.

7.3 Beta-functions, the group of polynomial B-functions

The Euler beta function, also called the Euler integral of the first kind, is a special function defined by

$$\mathscr{B}(a,b) = \int_0^1 x^a (1-x)^b dx$$

for $a,b \geq 0$, see [1]. The Euler beta function is symmetric, i.e. $\mathscr{B}(a,b) = \mathscr{B}(b,a)$. If a and b are positive integers the expression can be simplified to:

$$\mathscr{B}(a,b) = \frac{a!\ b!}{(a+b+1)!}. \tag{7.17}$$

The Euler beta function was first studied by Euler and Legendre and was given its name by Jacques Binet.

The incomplete beta function is a generalization of the Euler beta function that replaces the definite integral of the Euler beta function with an indefinite integral, ie

$$\mathscr{B}(t;a,b) = \int_0^t x^a (1-x)^b dx. \tag{7.18}$$

The regularized incomplete beta function is defined in terms of the incomplete beta function and the complete beta function, ie

$$I_t(a,b) = \frac{\mathscr{B}(t;a,b)}{\mathscr{B}(a,b)}, \quad t \in [0,1]. \tag{7.19}$$

Working out the integral for integer values of a and b, one finds (computet in [130])

$$I_t(a,b) = \sum_{j=a+1}^{a+b+1} \frac{(a+b+1)!}{j!\ (a+b+1-j)!}\ t^j (1-t)^{a+b+1-j}. \tag{7.20}$$

Lemma 7.1. *The regularized incomplete beta function, $I_t(a,b)$, has the properties:*

I	*Zero at $t=0$*	$I_0(a,b) = 0$,	
II	*One at $t=1$*	$I_1(a,b) = 1$,	
III	*Hermite order*	$\frac{d^j}{dt^j} I_0(a,b) = 0, \quad j = 1,2,...,a,$	*i.e. order a at start*
		$\frac{d^j}{dt^j} I_1(a,b) = 0, \quad j = 1,2,...,b,$	*i.e. order b at end*
IV	*Antisymmetric*	$I_t(a,b) = 1 - I_{1-t}(b,a),$	*i.e. symmetric if a=b*
V	*monotone*	$\frac{d}{dt} I_t(a,b) > 0, \quad 0 < t < 1 \quad$ *and* $\quad \frac{d}{dt} I_t(a,b) = 0, \quad t = \{0,1\}.$	
VI	*Recursive*	$I_t(a,b) = t\ I_t(a-1,b) + (1-t)\ I_t(a,b-1).$	
		$I_t(a,b) = I_t(a-1,b) - \frac{t^a (1-t)^{b+1}}{a\ \mathscr{B}(a-1,b)} = I_t(a,b-1) + \frac{t^{a+1}(1-t)^b}{b\ \mathscr{B}(a,b-1)},$	

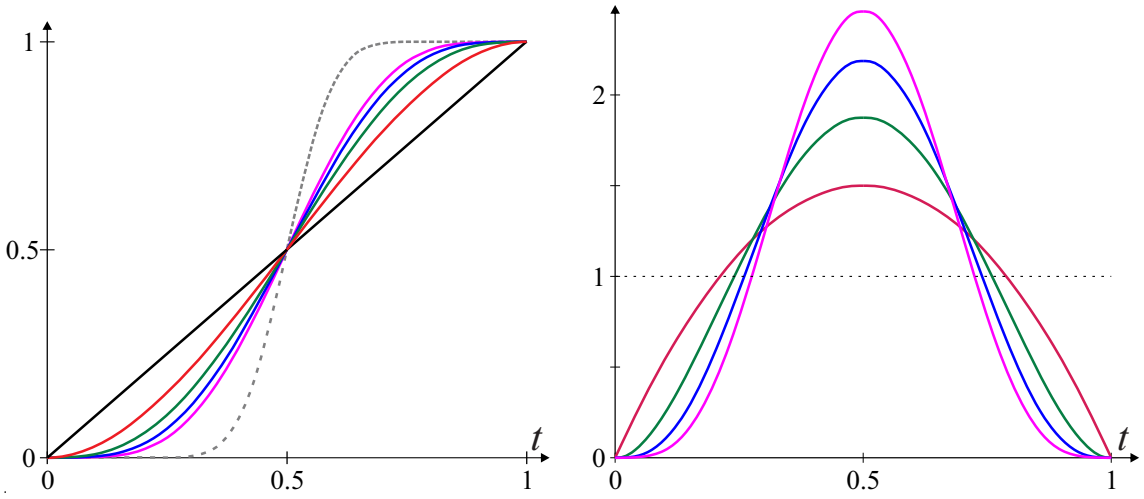


Figure 7.8: *On left side we see six symmetric Beta-functions, black-order 0, red-order 1, green-order 2, blue-order 3, purple-order 4 and dashed gray-order 20. On right side is the first derivative for five of the Beta-functions shown in the same color.*

The proof of lemma 7.1 can be found in appendix C.3.

Theorem 7.3. *The regularized incomplete beta function is a B-function when the shape-values a,b are integers. For short we name it Beta-function (see [43]), expressed by*

$$B_{a,b}(t) = I_t(a,b), \quad \text{for} \quad t \in [0,1],$$

and for $a,b \geq 0$, $a,b \in \mathbb{Z}$, and where $a = S_0$ and $b = S_1$ are the (Hermite) orders of the Beta-function. If $a = b$ we use the notation $B_S(t) = I_t(S,S)$

Proof. The theorem follows from Definition 7.1 and Definition 7.2 and lemma 7.1. □

Note that the Beta-function of lowest order is linear and the series of the group converges towards step-functions. Beta-function in blending was first introduced by L. Dechevsky together with Gancheva and Delistoyanova in 2006 and published in [71] and [47]. We will now look at the symmetric series of Beta-functions.

Symmetric Beta-functions

A symmetric Beta-function $B_s(t)$ is symmetric because of property **IV** in lemma 7.1. The order of the Beta-function is S. The first five Beta-functions of order 0 to 4 are:

$$B_0(t) = t \qquad \text{order } 0$$
$$B_1(t) = -2t^3 + 3t^2 \qquad \text{order } 1$$
$$B_2(t) = 6t^5 - 15t^4 + 10t^3 \qquad \text{order } 2$$
$$B_3(t) = -20t^7 + 70t^6 - 84t^5 + 35t^4 \qquad \text{order } 3$$
$$B_4(t) = 70t^9 - 315t^8 + 540t^7 - 420t^6 + 126t^5 \qquad \text{order } 4$$

Calculated from expression (7.20)

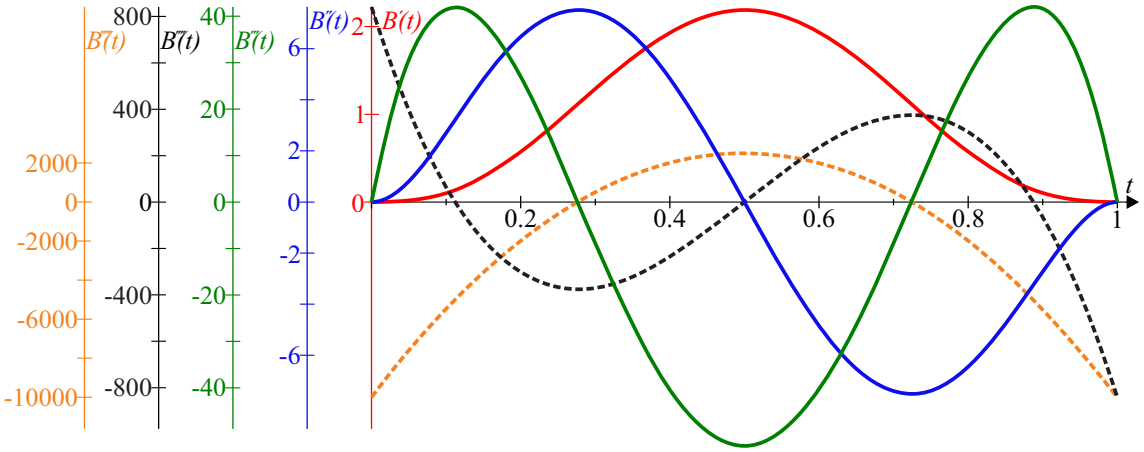


Figure 7.9: *The figure shows the five first subsequent derivatives of a 3^{rd} order symmetric Beta-functions; 1^{st} derivative red, 2^{nd} derivative blue, 3^{rd} derivative green, 4^{th} derivative dashed black and 5^{th} derivative dashed orange. The functions are connected to their respective axis on the left side with their color. The 4^{th} and 5^{th} derivative are dashed because they are not zero at $t = 0, 1$.*

In Figure 7.8 is the five first symmetric Beta-functions plotted together with the order 20 Beta-function. Note that the series appears to converge towards a step function at $t = 0.5$. Otherwise, the symmetric Beta-functions are the same functions we find as basis functions for Hermite curves of degree $3, 5, 7, ...$, see the basis function number 2 in (4.17) and (4.24).

7.3.1 Beta-functions, differentiation

The derivatives of a general Beta-function are quite complex especially on the high order derivatives. The three first derivatives for the general Beta-function are:

$$
\begin{aligned}
B'_{a,b}(t) &= \tfrac{(a+b+1)!}{a!b!}\, t^a(1-t)^b, \\
B''_{a,b}(t) &= \tfrac{(a+b+1)!}{a!b!}\, (a(1-t)-bt)\, t^{a-1}(1-t)^{b-1}, \\
B'''_{a,b}(t) &= \tfrac{(a+b+1)!}{a!b!}\, (a(a-1)-(a+b-1)(2a-(a+b)t)t)\, t^{a-2}(1-t)^{b-2}.
\end{aligned}
$$

For a concrete Beta-function is the derivatives much more easy. We use the symmetric 3^{rd} order Beta-functions as example. The formula and the five first derivatives are

$$
\begin{aligned}
B_3(t) &= -20t^7 + 70t^6 - 84t^5 + 35t^4, \\
B'_3(t) &= 140t^3(1-t)^3, \\
B''_3(t) &= 420t^2(1-2t)(1-t)^2, \\
B'''_3(t) &= 840t(1-t)(5t^2-5t+1), \\
B^{(4)}_3(t) &= 840(1-2t)(10t^2-10t+1), \\
B^{(5)}_3(t) &= 10080(-5t^2+5t-1).
\end{aligned}
$$

In Figure 7.9 shows the first five subsequent derivatives of the symmetric 3^{rd} order Beta-function. To show them in the same figure, we must use a separate axis for each of them. The color of the axis is the same as the color of the respective function. The red is the first derivative, the blue is the second derivative, the green is the third derivative. Note that the values of all three are zero at start and end. This is because the order of the Beta-function is three. The fourth derivatives are plotted in dashed black and the fifth derivatives are plotted in dashed orange.

7.4 The group of rational B-functions

In 2001, parametric G^n blending was discussed by E. Hartmann in [86]. He introduced a rational blending function, abbreviated RB-function.

$$b_n(t;\mu) = \frac{(1-\mu)t^{n+1}}{(1-\mu)t^{n+1} + \mu(1-t)^{n+1}},\ t \in [0,1],\ 0 < \mu < 1,\ n \geq 0.$$

If $\mu = 0.5$ the curves are point symmetric about the point (0.5, 0.5). For other μ the curves are asymmetric. Therefor he called μ the balance of the blending function $b_n(t;\mu)$. The number n determine the number of subsequent derivatives to be zero at start and end.

To use a balance variable μ can be useful and we will look into it later. However, if we want to limit the number of shape parameters to only the order parameters we can slightly modify the formula to get a more convenient B-function.

Theorem 7.4. *The rational function is a B-function. For short we name it an RB-function. It is expressed by*

$$B_{a,b}(t) = \frac{t^{a+1}}{t^{a+1} + (1-t)^{b+1}}, \quad \textit{for} \quad t \in [0,1], \tag{7.21}$$

and for $a,b \geq 0$, $a,b \in \mathbb{Z}$, where $a = S_0$ is the left side (Hermite) order and where $b = S_1$ is the right side (Hermite) order. If $a = b$ then the RB-function is symmetric according to **D5** *in Definition 7.1.*

The proof of theorem 7.4 can be found in appendix C.4.

Note that the RB-function of lowest order is linear,

$$B_{0,0}(t) = \frac{t}{t + (1-t)} = t,$$

and that the series of the group seems to converges towards a step-function at $t = 0.5$. Except for the linear function they all have a kind of S-shape.

We first look at the symmetric series of RB-functions.

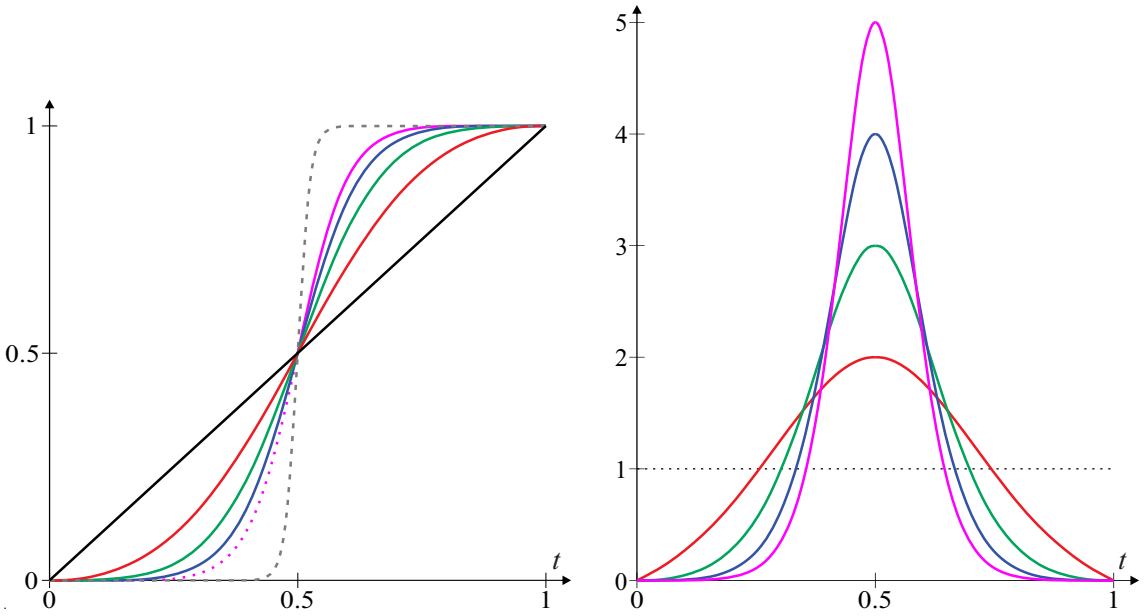


Figure 7.10: *On left side we see six symmetric RB-functions; black-order 0, red-order 1, green-order 2, blue-order, purple-order 4 and dashed gray-order 20. On right side is the first derivative for five of the RB-functions shown in the same color.*

Symmetric RB-functions

In Theorem 7.4 is shown that a symmetric RB-function $B_s(t)$ is symmetric, and the order is shown to be S in Theorem 7.4. The general formula for the series of symmetric RB-functions is:

$$B_s(t) = \frac{t^{s+1}}{t^{s+1} + (1-t)^{s+1}}$$

The four first symmetric RB-function in the series (of order 0,1,2,3) are:

$$B_0(t) = \frac{t^1}{t^1+(1-t)^1} = t$$

$$B_1(t) = \frac{t^2}{t^2+(1-t)^2} = \frac{t^2}{2t^2-2t+1}$$

$$B_2(t) = \frac{t^3}{t^3+(1-t)^3} = \frac{t^3}{3t^2-3t+1}$$

$$B_3(t) = \frac{t^4}{t^4+(1-t)^4} = \frac{t^4}{2t^4-4t^3+6t^2-4t+1}$$

$$B_4(t) = \frac{t^5}{t^5+(1-t)^5} = \frac{t^5}{2t^5-5t^4+10t^3-10t^2+5t-1}$$

In Figure 7.10 are the five first symmetric RB-functions plotted together with the order 20 symmetric RB-function. Here we see that the series seems to converge towards a step-function at $t = 0.5$. Compared to the symmetric Beta-functions from Figure 7.8, we see from the plot of first derivatives in Figure 7.10 that the RB-functions are steeper.

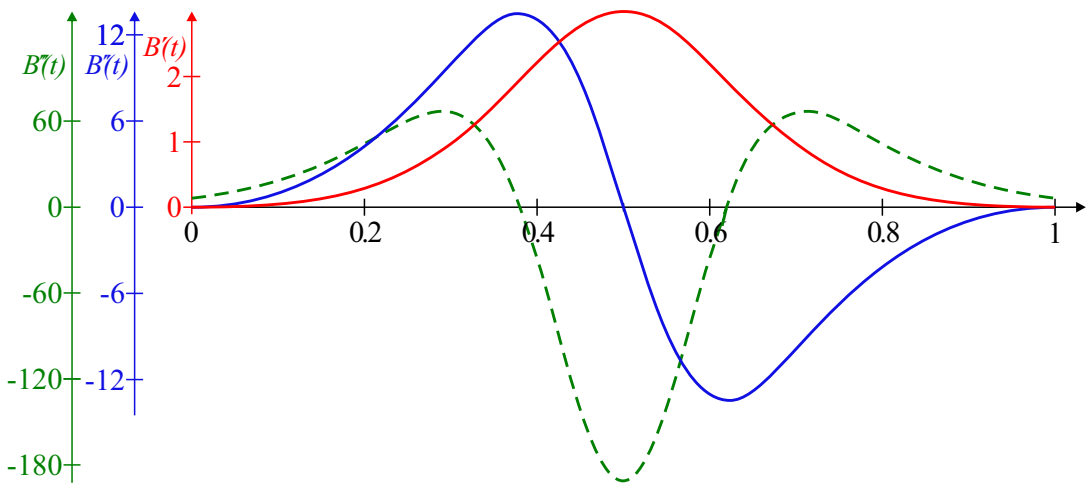


Figure 7.11: *The figure shows the three first subsequent derivatives of a 2th order symmetric RB-functions; 1^{st} derivative - bold red, 2^{nd} derivative - bold blue and 3^{rd} derivative - dashed green. Each vertical axis is connected to the plot with the same color.*

7.4.1 RB-functions, differentiation

The derivatives of a general RB-function are quite complex especially on the high order derivatives. The two first derivatives for the general RB-function are:

$$B'_{a,b}(t) = (a(1-t)+bt+1)\frac{t^a(1-t)^b}{(t^{a+1}+(1-t)^{b+1})^2},$$

$$B''_{a,b}(t) = \Big[\left[t^2\left(a\left(a+1\right)-b\left(b+1\right)\right)+a\left(a+1\right)\left(1-2t\right)\right]\left(t^{a+1}+\left(1-t\right)^{b+1}\right)-$$
$$2\left(a(1-t)+bt+1\right)t\left(1-t\right)\left(\left(a+1\right)t^a-\left(b+1\right)\left(1-t\right)^b\right)\Big]\frac{t^{a-1}(1-t)^{b-1}}{\left(\left(1-t\right)^{b+1}+t^{a+1}\right)^3}.$$

For a concrete RB-function is the derivatives much easier. We use the symmetric 2nd order RB-functions as example. The formula and the three first derivatives are

$$\begin{aligned}
B_2(t) &= \frac{t^3}{t^3+(1-t)^3},\\
B'_2(t) &= 3\frac{t^2(1-t)^2}{(t^3+(1-t)^3)^2},\\
B''_2(t) &= -6\frac{(2t-1)t(1-t)}{(t^3+(1-t)^3)^3},\\
B'''_2(t) &= -6\frac{18t^2(1-t)^2-1}{(t^3+(1-t)^3)^4}.
\end{aligned}$$

In Figure 7.11, the three first derivatives of the symmetric 2nd order RB-function are plotted. To be able to show them in the same figure we have to scale them. The bold-red is the first derivative (not scaled), the bold-blue is the second derivative (divided by 6) and the dashed-green is the third derivative (divided by 30). Note that the value of the first and second derivatives are zero at start and end. It follows that the order of this Beta-function is two. The third derivatives is not zero at start and end as we can see in the Figure 7.11.

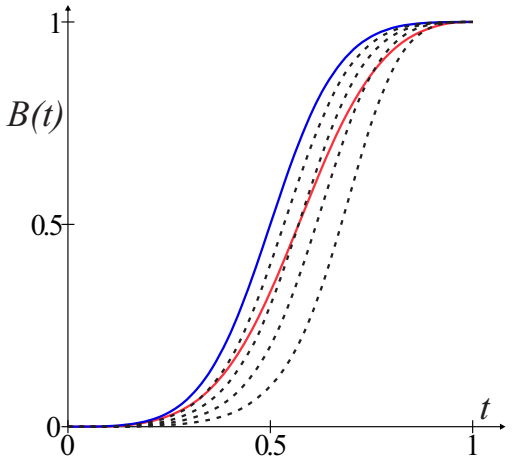


Figure 7.12: *Examples of Rμ-functions. The blue curve is the symmetric B-function* $B_{2,2}(t;0.5) = B_2(t)$. *The four black dashed curves are Rμ-functions where* μ *is* $0.6, 0.7, 0.8$ *and* 0.9. *The red solid curve is an asymmetric RB-function,* $B_{2,1}(t)$.

7.4.2 RB-functions with a balance parameter

If we include the balance parameter to the rational B-function we get:

Corollary 7.1. *A Rμ-function is a B-function defined by*

$$B_{a,b}(t;\mu) = \frac{(1-\mu)t^{a+1}}{(1-\mu)t^{a+1} + \mu(1-t)^{b+1}}, \quad \text{for } t \in [0,1], \tag{7.22}$$

and where $a, b \in \mathbb{N}$, $a, b > 0$ *and* $0 < \mu < 1$, $\mu \in \mathbb{R}$.

It follows that a Rμ-function is a B-function because of Theorem 7.4.

As for RB-functions, the left side order is $a = S_0$, *and the right side order is* $b = S_1$. *If* $a = b$ *and* $\mu = 0.5$ *then the Rμ-function is symmetric according to* **D5** *in Definition 7.1.*

To compare Rμ-functions with RB-functions we first look at a second order symmetric RB-function $B_2(t)$. We then expand this RB-function to a Rμ-functions where $a = b = 2$ and $\mu = 0.5$, i.e. $B_{2,2}(t;0.5)$. Then we change μ to see what happens.

In Figure 7.12 is the function, $B_{2,2}(t;0.5) = B_2(t)$ plotted in solid blue. We can also see four functions plotted in dashed black. These are $B_{2,2}(t;0.6)$, $B_{2,2}(t;0.7)$, $B_{2,2}(t;0.8)$ and $B_{2,2}(t;0.9)$. As we can see, they are pushed to the right according to the value of μ. Remember that all five function are of order 2 on both sides, but only the solid blue is symmetric.

To see the effect of the balance parameter μ we also shows an asymmetric RB-function $B_{2,1}(t)$ plotted in solid red in Figure 7.12. The balance parameter μ does not influence the steepness of the function, while the order affects the steepness. Later, in section 7.8, we will investigate what we call a balance-symmetry of B-functions.

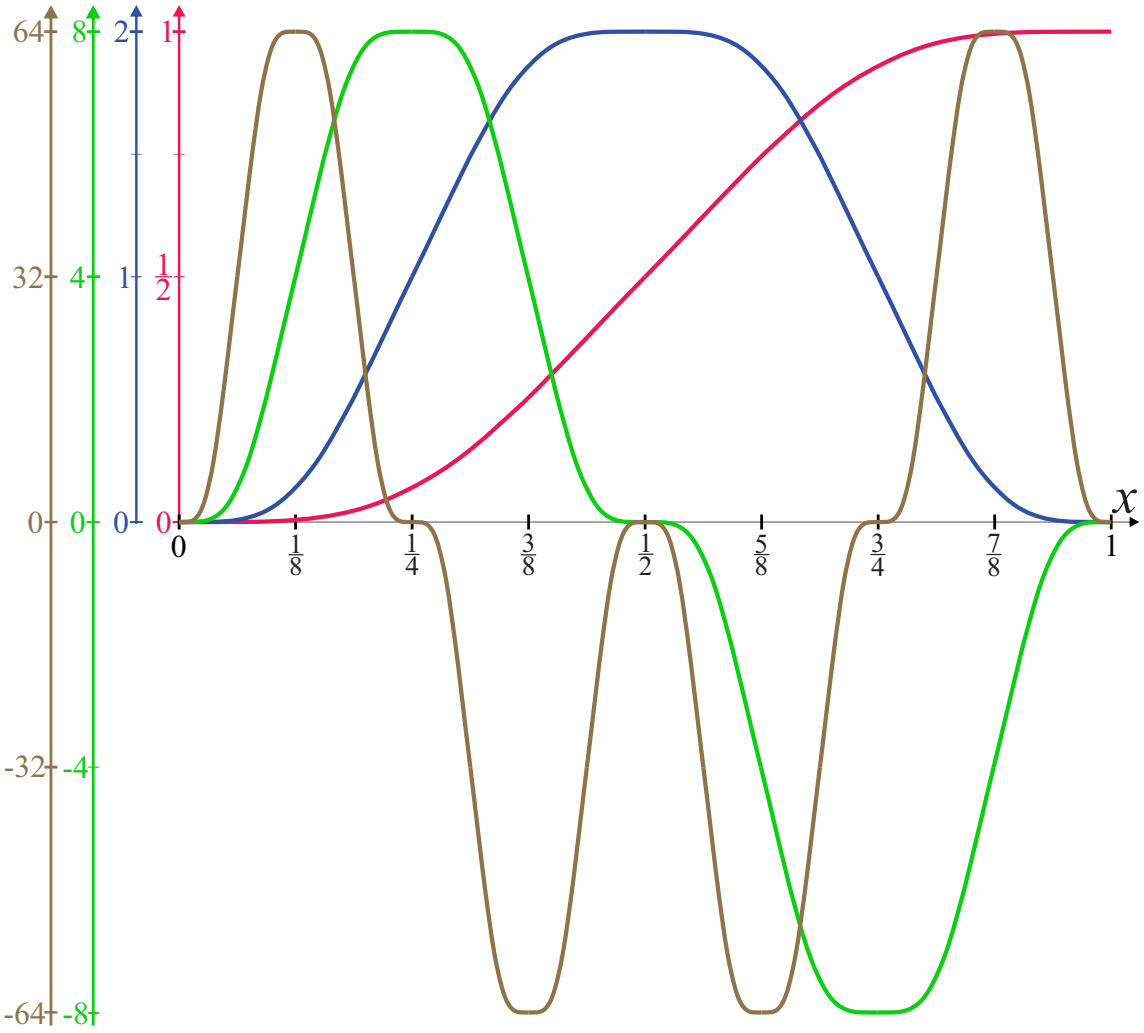


Figure 7.13: *A plot of the Fabius function (red) and its 1^{st} derivative (blue), 2^{nd} derivative (green) and 3^{rd} derivative (brown). The vertical axis on the left have the same colors as the plots to which they are attached.*

7.5 Fabius function, the complete B-function

H. Olofsen introduced the Fabius function as a B-function in [126]. Fabius function introduced by Jaap Fabius [62] is an infinitely differentiable, but nowhere analytic function that is self-differential. The Fabius functional satisfies the following differential equation:

$$\mathfrak{F}'(x) = 2\mathfrak{F}(2x), \quad x \in \mathbb{R}+ \quad (\text{that is } [0,+\infty)). \tag{7.23}$$

It follows that $\mathfrak{F}(x+2^n) = -\mathfrak{F}(x)$ for $0 \leq x \leq 2^n$ where n is a positive integer. The sequence of intervals within which this function is positive or negative follows the same pattern as the Thue-Morse sequence [3]. In Figure 7.13 there is a plot of the Fabius function and its 1^{st}, 2^{nd} and 3^{rd} derivatives on $0 \leq t \leq 1$. We can see that the derivatives are squeezed and scaled (plus or minus) copies of the function itself. The scaling of the derivatives are reflected in that $\max|\mathfrak{F}^{(j)}(x)| = 2^{\Sigma_{i=1}^{j} i}$. i.e. $2,8,64,1024,\ldots$, due to (7.23) and this can also be observed in Figure 7.13.

If we limit the domain to $[0,1]$, we observe the following, the Fabius function satisfies the initial condition $\mathfrak{F}(0)=0$, the symmetry condition $\mathfrak{F}(x)+\mathfrak{F}(1-x)=1$ for $0\leq x\leq 1$, i.e. it is point symmetric about $\left(\frac{1}{2},\frac{1}{2}\right)$. It follows that $\mathfrak{F}(x)$ is monotone increasing for $0\leq x\leq 1$, with $\mathfrak{F}(\frac{1}{2})=\frac{1}{2}$ and $\mathfrak{F}(1)=1$. An expression of the Fabius functon, if we restrict the domain to $x\in[0,1]$, is

$$\mathfrak{F}(x)=\begin{cases}\displaystyle\int_0^{2x}\mathfrak{F}(s)ds, & \text{if } x\in\left[0,\tfrac{1}{2}\right],\\[2ex] \displaystyle\int_{2x-1}^{1}\mathfrak{F}(s)ds+2x-1, & \text{if } x\in\left(\tfrac{1}{2},1\right].\end{cases}\tag{7.24}$$

The Fabius function clearly meets all the requirements for being a symmetric B-function, All the 5 points from Definition 7.1 are fulfilled. However, there is one extra property that makes the Fabius function special. Because all derivatives are scaled versions of the function itself where it is scaled by 2^n in the x-axis, as follows from (7.23) and (7.24) and as we see in Figure 7.13, all derivatives are zero at $t=0$ and $t=1$. It follows that the Hermite order is ∞, and therefore it leads to the following definition:

The complete B-function

Definition 7.3. *A complete B-function is a function that fulfill the requirement in Definition 7.1, and where all derivatives at $t=0$ and $t=1$ are zero. Thus, the order of a complete B-function is ∞.*

- A complete B-function are not analytic at $t=0$ and $t=1$. It follows that Taylor expansion does not work at these two points.
- The Fabius function is a complete B-function, but in addition to be a complete function it is nowhere analytic.

In addition to be nowhere analytic there is one more practical problem using the Fabius function as a B-function, we cannot directly calculate the value anywhere. It is only possible to calculate at dyadic rational numbers in the form $\frac{m}{2^n}$, where n is a positive integer $1,2,3,\ldots$ and m is a positive odd number $1,3,...,2^n-1$, as shown by Jan K. Haugland in 2016 [88]. Calculating (7.24) for n up to 5 gives us,

$$\mathfrak{F}\left(\tfrac{1}{2}\right)=\tfrac{1}{2}$$
$$\mathfrak{F}\left\{\tfrac{1}{4},\tfrac{3}{4}\right\}=\left\{\tfrac{5}{72},\tfrac{67}{72}\right\}$$
$$\mathfrak{F}\left\{\tfrac{1}{8},\tfrac{3}{8},\tfrac{5}{8},\tfrac{7}{8}\right\}=\left\{\tfrac{1}{288},\tfrac{73}{288},\tfrac{215}{288},\tfrac{287}{288}\right\}$$
$$\mathfrak{F}\left\{\tfrac{1}{16},\tfrac{3}{16},\ldots,\tfrac{15}{16}\right\}=\left\{\tfrac{143}{2073600},\tfrac{46657}{2073600},\tfrac{305857}{2073600},\tfrac{777743}{2073600},\tfrac{1295857}{2073600},\tfrac{1767743}{2073600},\tfrac{2026943}{2073600},\tfrac{2073457}{2073600}\right\}$$
$$\mathfrak{F}\left\{\tfrac{1}{32},\tfrac{3}{32},\ldots,\tfrac{31}{32}\right\}=\left\{\tfrac{19}{33177600},\tfrac{25219}{33177600},\tfrac{334781}{33177600},\tfrac{1396781}{33177600},\tfrac{3470381}{33177600},\tfrac{6555581}{33177600},\tfrac{10393219}{33177600},\right.$$
$$\left.\tfrac{14515219}{33177600},\tfrac{18662381}{33177600},\tfrac{22784381}{33177600},\tfrac{26622019}{33177600},\tfrac{29707219}{33177600},\tfrac{31780819}{33177600},\tfrac{32842819}{33177600},\tfrac{33152381}{33177600},\tfrac{33177581}{33177600}\right\}.$$

To calculate the Fabius function in general, one can use interpolation between these dyadic rational numbers or/and to use polynomial approximations of the Fabius function, see appendix A or [126].

7.6 The group of trigonometric B-functions

TB-function is short for trigonometric B-functions. In 2019 Hans Olofsen introduced blending functions based on trigonometric and polynomial approximations of the Fabius function [126]. He showed us how to construct trigonometric B-functions of all order from approximating the Fabius function. A sum of weighted cosines can approximate the Fabius function on the domain $[0,2]$, see [42, eq.(30)]. From this we get

$$F_M(t) = \frac{1}{2} - \sum_{m=1}^{M} c_m \cos(m\pi t), \quad t \in [0,1]. \tag{7.25}$$

In order to be able to develop trigonometric B functions that are point symmetric, the following three points are important;

- If m is odd, $\cos(m\pi t)$ is point symmetric around $\left(\frac{1}{2},\frac{1}{2}\right)$. If m is even, $\cos(m\pi t)$ is symmetrical about the vertical axis $x=\frac{1}{2}$. Therefore, to ensure point symmetry, m must only be odd ie $\{1,3,5,\ldots\}$, and consequently M must also be odd.
- To ensure that $F(0)=0$ and $F(1)=1$ must $\sum c_m = \frac{1}{2}$ for $m=1,3,...,M$.
- The order of the B-function is connected to that all 1^{st}-, 2^{nd}-,..., M^{th}-derivatives are zero at $t=0,1$. This will be fulfilled if $\sum m^j c_m = 0$ for $m=1,3,...,M$ and $j=2,4,...,M-1$.

Two examples of how to construct symmetrical B-functions of odd order will be given. In (7.25), to find c_m when $m=1,3$ and c_m when $m=1,3,5$ we make a $\frac{M+1}{2}\times\frac{M+1}{2}$ matrix and then compute c_m for $M=3$ and for $M=5$ in the following way,

$$M=3 \;\Rightarrow\; \begin{pmatrix} 1 & 1 \\ 1^2 & 3^2 \end{pmatrix}\begin{pmatrix} c_1 \\ c_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix} \;\Rightarrow\; \begin{pmatrix} c_1 \\ c_3 \end{pmatrix} = \begin{pmatrix} \frac{9}{16} \\ -\frac{1}{16} \end{pmatrix},$$

$$M=5 \;\Rightarrow\; \begin{pmatrix} 1 & 1 & 1 \\ 1^2 & 3^2 & 5^2 \\ 1^4 & 3^4 & 5^4 \end{pmatrix}\begin{pmatrix} c_1 \\ c_3 \\ c_5 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \end{pmatrix} \;\Rightarrow\; \begin{pmatrix} c_1 \\ c_3 \\ c_5 \end{pmatrix} = \begin{pmatrix} \frac{75}{128} \\ -\frac{25}{256} \\ \frac{3}{256} \end{pmatrix}.$$

The result is trigonometric B-functions of odd order, order 1 with coordinates $c_1=\frac{1}{2}$, order 3 with coordinates $c_1=\frac{9}{16}$ and $c_3=-\frac{1}{16}$, and order 5 with coordinates $c_1=\frac{75}{128}$, $c_3=-\frac{25}{256}$ and $c_5=\frac{3}{256}$.

To obtain trigonometric B-functions of Hermite order of even numbers, we use B-functions of odd Hermite order $M=1,3,5,\ldots$ as the first derivative of the new function. This can be done because $\cos m\pi t$ is symmetric about the vertical axis $x=1$ for $m=1,3,5,\ldots$. It follows that the derivatives at $t=2$ are also zero for M subsequent derivatives. Therefore, it is possible to obtain a B-function of order $m+1$, $B_{m+1}(t)$ from a B-function of order m, ie. $B_m(t)$ when $m=1,3,5,\ldots$.

The method is to use the antiderivative of $B_m(t),\ m=1,3,...$ scaled by 2, because (7.23) states that the maximum value of the first derivative of the Fabius function is 2, and we use it on the domain $[0,2]$ because the parameter is scaled by 2 in (7.23). The result is $B_n(t),\ n=2,4,...$ on the domain $[0,1]$.

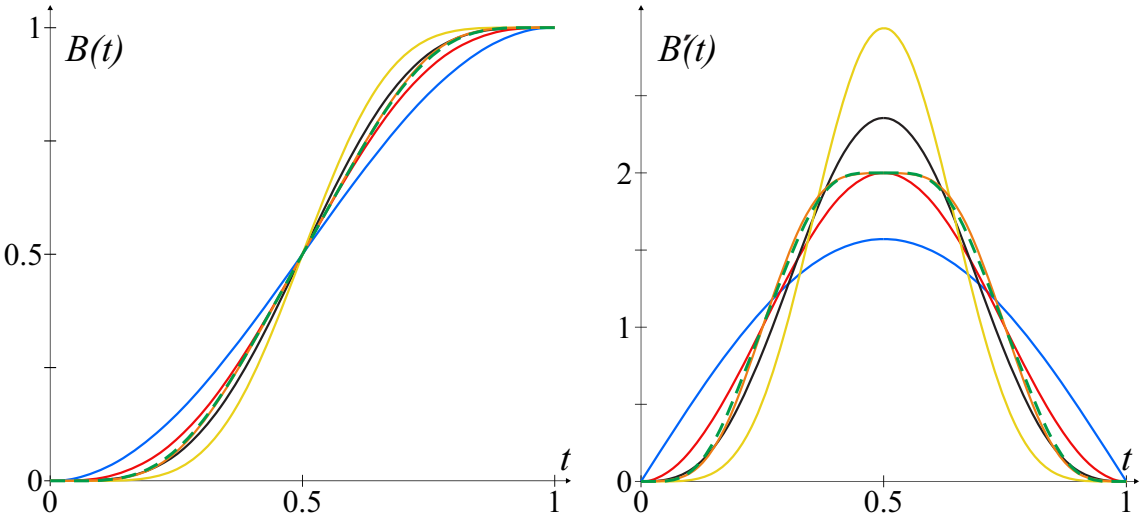


Figure 7.14: *On left side, the plot of the Fabius function, dashed green and five trigonometric B-functions, order 1 blue, 2 red, 3 black, 4 orange and 5 yellow. On right side are the plots of the* 1^{st} *derivatives, in the same color as their respective functions.*

One example is; we start with $B_1(t) = \frac{1}{2} - \frac{1}{2}\cos \pi t$. We then replace t with $2t$, and we multiply the function by 2. The result is

$$B_2'(t) = 1 - \cos 2\pi t.$$

We then use the antiderivative, and get,

$$B_2(t) = t - \frac{1}{2\pi}\sin 2\pi t, \quad t \in [0,1].$$

In Figure 7.14 on the left side there is a plot of 5 trigonometric B-functions (of order 1,2,3,4,5) together with the Fabius function, on the right side is their 1^{st} derivatives.

Symmetric TB-functions

A series of symmetric TB-functions obtained by approximating the Fabius function. The formula for the first five symmetric TB-functions are

$$\begin{aligned}
B_1(t) &= \tfrac{1}{2} - \tfrac{1}{2}\cos \pi t && \text{order 1} \\
B_2(t) &= t - \tfrac{1}{2\pi}\sin 2\pi t && \text{order 2} \\
B_3(t) &= \tfrac{1}{2} - \tfrac{1}{16}(9\cos \pi t - \cos 3\pi t) && \text{order 3} \\
B_4(t) &= t - \tfrac{1}{48\pi}(27\sin 2\pi t - \sin 6\pi t) && \text{order 4} \\
B_5(t) &= \tfrac{1}{2} - \tfrac{1}{256}(150\cos \pi t - 25\cos 3\pi t + 3\cos 5\pi t) && \text{order 5}
\end{aligned} \tag{7.26}$$

Note that $B_1(t)$ is the same function as the one plotted in b) in Figure 7.1 and used in circle splines [153], i.e.

$$\sin^2 \frac{\pi t}{2} = \frac{1}{2} - \frac{1}{2}\cos \pi t.$$

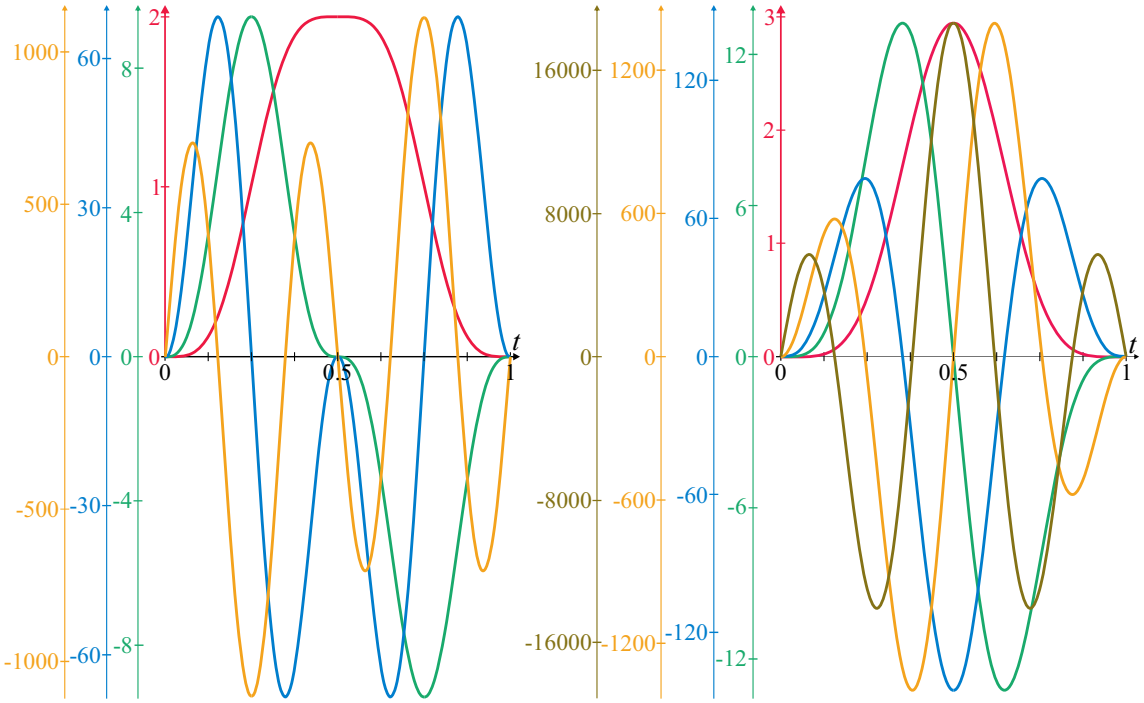


Figure 7.15: *On left side, the plot of the four first derivatives for $B_4(t)$, $B_4'(t)$-red, $B_4''(t)$-green, $B_4'''(t)$-blue and $B_4^{(4)}(t)$-orange. On right side are the plots of five derivatives for $B_5(t)$, $B_5'(t)$-red, $B_5''(t)$-green, $B_5'''(t)$-blue, $B_5^{(4)}(t)$-orange and $B_5^{(5)}(t)$-brown.*

The differentiation of the trigonometric B-functions, (7.26), is simple and faste to calculate both on a CPU and a GPU. In Figure 7.15, the first four successive derivatives of $B_4(t)$ are plotted on the left. On the right, the five derivatives from 1^{st} to 5^{th} of $B_5(t)$ are plotted. Each plot of a derivative function has a given color. The different vertical axis on the left is connected to the plots that have the same color.

The plot in Figure 7.15 clearly verifies the Hermite order of the two functions because all derivatives in the plot are zero at $t = 0, 1$. Another observation is that $B_4(t)$ is the function that seems to be the best approximation of the Fabius function, an observation from combining Figure 7.13, 7.14 and 7.15. We also see that the maximum absolute value of the derivatives for $B_4(t)$ is increasing slightly more than the Fabius function as the order increase. For $B_5(t)$ the value increase significant more.

One open question raised by Olofsen; is the Fabius function the complete B-function where the maximum absolute value of the derivatives increases at least as the order of the derivatives increases? ie

$$\max|\mathfrak{F}^{(j)}(t)| < \max|B_c^{(j)}(t)|, \quad 0 \le t \le 1,$$

when $j \to \infty$, and for all complete B-functions $B_c(t)$.

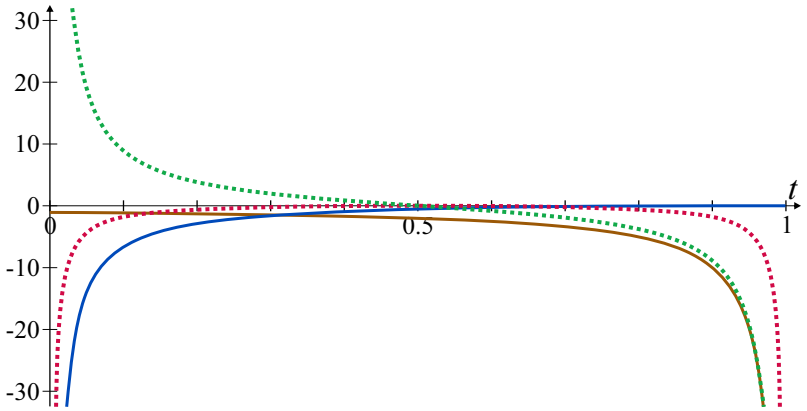


Figure 7.16: *Plot of the rational exponents of the expo-rational functions, (7.27) dashed red, (7.28) brown and blue, and (7.30) dotted green.*

7.7 The group of Expo-Rational B-functions

The properties of the natural exponential function are useful when building a B function. It is always positive and it is its own derivative, and it always goes faster towards $+\infty$ than any power of x when $x \to +\infty$, thus it also overrides any rational polynomial function when it goes toward zero[1]. If we add a rational exponent or, a hierarchical model together with a rational exponent, we get the basis for several Expo-Rational B functions. Of particular interest are the expo-rational functions and their 1^{st} derivative below,

$$\phi(t) = e^{-\frac{\left(t-\frac{1}{2}\right)^2}{t(1-t)}}, \qquad \phi'(t) = \frac{1-2t}{4(t(1-t))^2}\phi(t), \tag{7.27}$$

$$\psi(t) = e^{\frac{-2}{t}e^{\frac{-1}{1-t}}}, \qquad \psi'(t) = 2\frac{t^2-t+1}{t^2(1-t)^2}e^{\frac{-1}{1-t}}\psi(t), \tag{7.28}$$

$$\theta(t) = e^{\frac{1}{t}}, \qquad \theta'(t) = -\frac{1}{t^2}\theta(t), \tag{7.29}$$

$$\varphi(t) = \frac{\theta(t)}{\theta(1-t)} = e^{\frac{1}{t}-\frac{1}{1-t}}, \qquad \varphi'(t) = -\left(\frac{1}{t^2}+\frac{1}{(1-t)^2}\right)\varphi(t). \tag{7.30}$$

An exponential function maps $\mathbb{R}$ to $\mathbb{R}$+. What we see is that (7.27), (7.28) and (7.30) contract the domain to the open interval $(0,1)$. In Figure 7.16 there are plots of the rational exponents from the three functions above. The dashed red plot is the exponent in (7.27), mapping $(0,1)$ to $\mathbb{R}$-, it is symmetric about the axis $t = \frac{1}{2}$. The result is that $\phi(t)$ is a hat function with max value 1, and that the exponent at $t = 0,1$ is $-\infty$, a number that is not $\in \mathbb{R}$ but it is in the extended real number system. The result is that $\phi(t)$ is non analytic at $t = 0,1$. It is the same for (7.28). In Figure 7.16 is the brown and the blue plot the upper and the total exponent of $\psi(t)$. The blue comes from $-\infty$ and the brown goes to $-\infty$ and they are both only negative. So $\psi(t)$ is also non analytic at $t = 0,1$. The dotted green plot in Figure 7.16 is the exponent of (7.30), it maps $(0,1)$ to $\mathbb{R}$ turned, and is point symmetric around $(0.5,0)$. Consequently, $\varphi(t)$ is also non analytic at $t = 0,1$.

[1] $\lim_{x\to+\infty} x^n e^{-x} = 0$ for every n and $e^0 = 1$, $e^{a+b} = e^a e^b$, $e^a = \frac{1}{e^{-a}}$ and $\mathbb{R}$+ is $[0,+\infty)$, $\mathbb{R}$- is $(-\infty,0]$.

In 2004 Expo-Rational B-splines were presented by Dechevsky, Lakså and Bang at the conference of Mathematical methods for Curves and Surfaces, and later published in [104],[44],[45] and [102]. They initially used an expo-rationale function with several intrinsic parameters, but the default exponential function they used was $\phi(t)$ from (7.27). This function is related to the Gaussian bell function and the cumulative distribution function, see [74]. In 2015 the Expo-Rational B-splines was put into the frame of B-functions [103]. The expression of the default B-function, ie type 1 ERB-function, is:

$$B_d(t) = \mathbb{S} \int_0^t \phi(s)ds, \quad t \in [0,1], \tag{7.31}$$

where the expo-rational function $\phi(t)$ as given in (7.27) and the scaling

$$\mathbb{S} = \left[\int_0^1 \phi(t)dt \right]^{-1} \approx 1.6571376797382103. \tag{7.32}$$

Because there is no analytical solution to the integral in (7.31), this is a computationally demanding formula. But in 2007 an approximative calculation algorithm with tolerances better than $3.4e-13$ ($L^\infty[0,1]$ and double precision) was given and where the number of multiplications is 6 for the function value, 4 for the 1^{st} derivative and 5 for the 2^{nd} derivative. The method and algorithm are described in Appendix A.4 and in more detail in [102].

In 2004 Ying and Zorin published a work on constructing surfaces of arbitrary smoothness [168]. The work was based on Grimm and Hughes work from 1995 [82], and Navau and Garcia in 2000 [125] on the same topic. Navau and Garcia used a basis function (B-function) that was not point-symmetric according to Definition 7.1. This function was (7.28). Ying and Zorin made a symmetric B-function by combining this non-symmetric function with the rational construction of Hartmann given in 7.21, ie

$$B_y(t) = \frac{\psi(t)}{\psi(t) + \psi(1-t)}. \tag{7.33}$$

However, there is a standard way to create a symmetric B-function from a non-symmetric B-function. It is to sum the function with its antisymmetric twin and divide the result by two. The Hermite order of the resulting B-function will be equal the lowest order of the initial non symmetric B-function. If we use this method on (7.30), we get the following symmetric ERB-function,

$$B_x(t) = \frac{1}{2}\left(1 - \psi(1-t) + \psi(t)\right). \tag{7.34}$$

In 2013 Dechevsky and Zanaty introduced a Logistic ERB-function [46], which included the expo-rational functions $\theta(t)$, (7.29) or $\varphi(t)$, (7.30). The following fractions gives the LERB-function,

$$B_z(t) = \frac{\theta(1-t)}{\theta(1-t) + \theta(t)} = \frac{1}{\varphi(t)+1}, \tag{7.35}$$

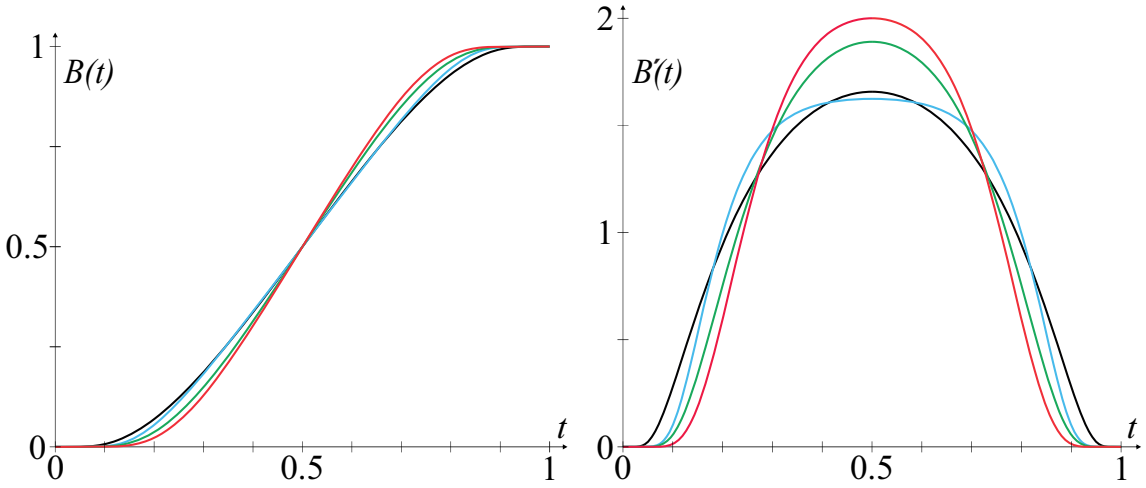


Figure 7.17: *On the left side in the figure is four Expo-Rational B-functions, $B_d(t)$ in black (7.31), $B_y(t)$ in blue (7.33), $B_x(t)$ in green (7.34) and the Logistic ERB-function $B_z(t)$ in red (7.35). On the right side is a plot of the 1^{st} derivatives of these four ERB-functions in the same color as their respective functions on the right side.*

which is actually the standard logistic function with a contraction of the domain from $\mathbb{R}$ to the open interval $(0,1)$. Following the definitions of the ERB-functions, the 1^{st} derivative of (7.31) $B_d(t)$, (7.33) $B_y(t)$, (7.34) $B_x(t)$ and (7.35) $B_z(t)$ are:

$$B'_d(t) = \mathbb{S}\,\phi(t), \qquad B'_y(t) = \frac{\psi'(t)B_y(1-t) + \psi'(1-t)B_y(t)}{\psi(t)+\psi(1-t)}, \tag{7.36}$$

$$B'_x(t) = \frac{1}{2}\left(\psi'(1-t) + \psi'(t)\right), \qquad B'_z(t) = -\varphi'(t)B_z(t)^2. \tag{7.37}$$

On the left side in Figure 7.17, there is a plot of the four functions $B_d(t)$ in black, $B_y(t)$ in blue, $B_x(t)$ in green and $B_z(t)$ in red together with their 1^{st} derivatives on the right side.

Expo-Rational B-functions

Definition 7.4. *An Expo-Rational B-function (ERB-function) is:*

- *a B-function according to Definition 7.1,*
- *it is a complete B-function according to Definition 7.3 (order ∞),*
- *is not analytic at $t = 0$ and $t = 1$,*
- *and is constructed using exponential functions with rational exponents.*

Theorem 7.5. *$B_d(t)$ defined in (7.31), $B_y(t)$ defined in (7.33), $B_x(t)$ defined in (7.34) and $B_z(t)$ defined in (7.35) are all symmetric Expo-Rational B-functions (ERB-functions).*
Thus, they are complete B-functions, with order infinite and they are not analytic at $t = 0$ and $t = 1$.

The proof of theorem 7.5 can be found in appendix C.5.

In Figure 7.18 are four derivatives of the four ERB-functions plotted. On upper left hand side we see four successive derivatives of $B_d(t)$, (7.31), on upper right hand side we see

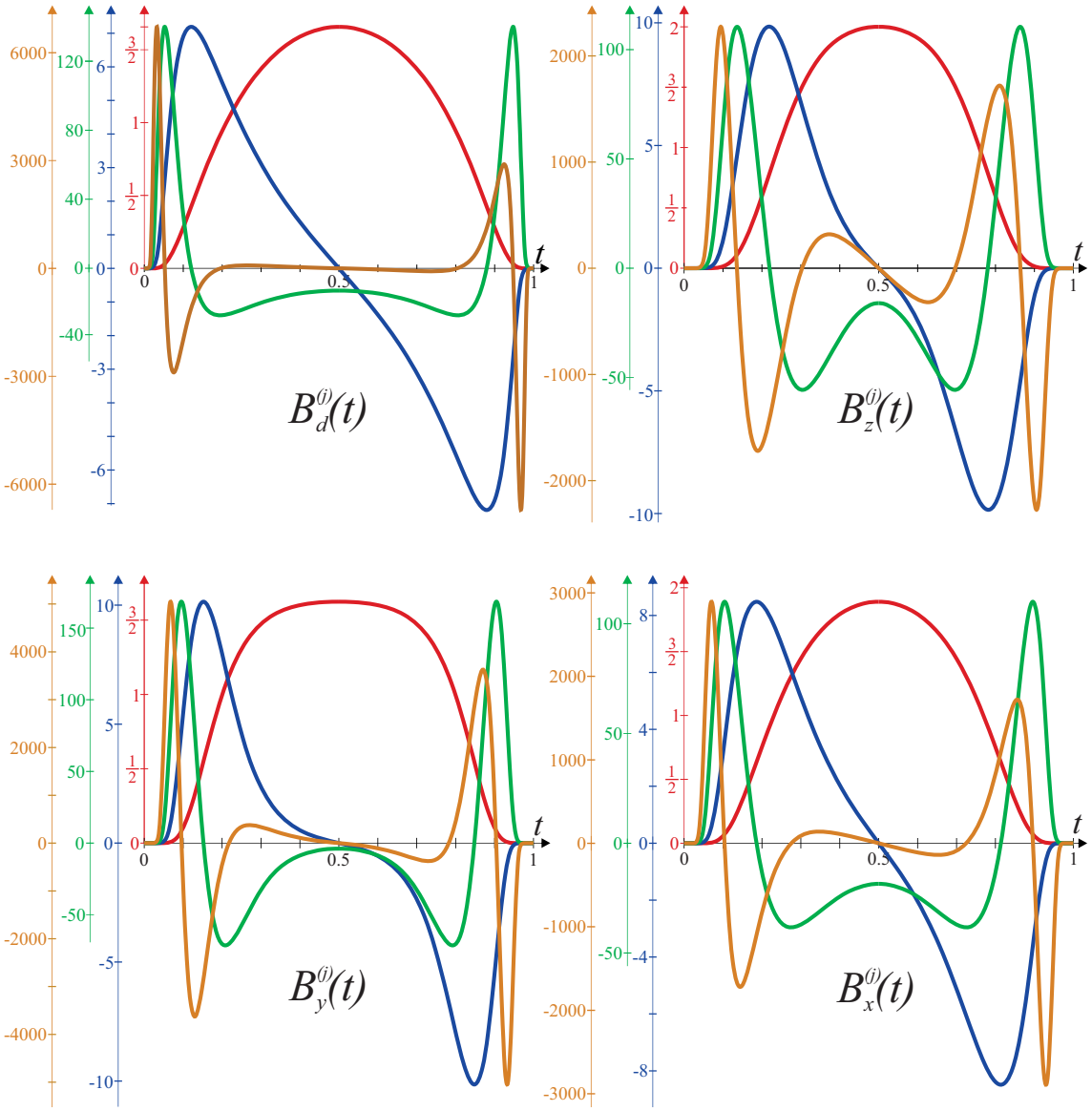


Figure 7.18: *In the figure, there are four derivatives of four ERB-functions. On upper left hand side we see four successive derivatives of $B_d(t)$, (7.31), on upper right hand side we see four successive derivatives of $B_z(t)$, (7.35), on lower left hand side we see four successive derivatives of $B_y(t)$, (7.33) and on lower right hand side we see four successive derivatives of $B_x(t)$, (7.34). The 1^{st} derivatives are plotted in red, the 2nd derivatives are plotted in blue, the 3^{rd} derivatives are plotted in green and the 4^{th} derivatives are plotted in orange. The axis on left hand side of each derivative plots has the same color as their respective functions in the derivative plots.*

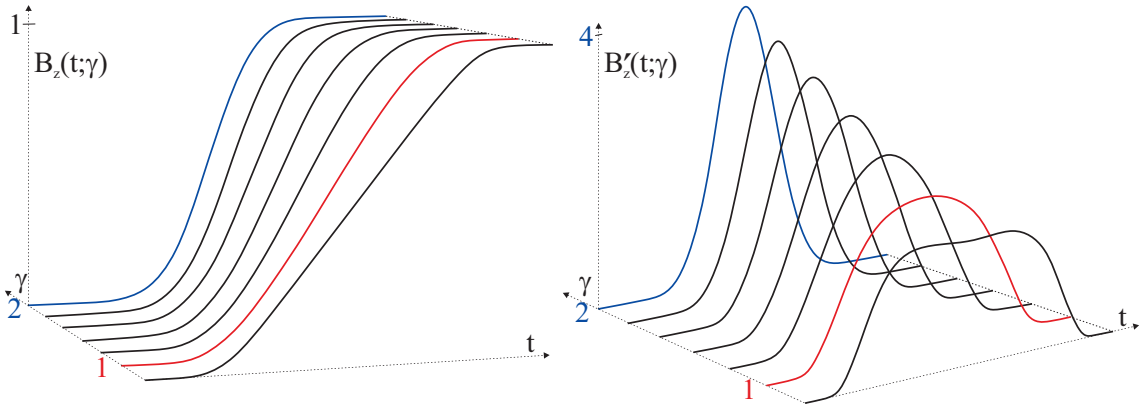


Figure 7.19: *On left hand side is a plot of $B_z(t;\gamma)$ for $\gamma = \{0.8, 1, 1.2, 1.4, 1.6, 1.8, 2\}$. $B_z(t;\gamma = 1) = B_z(t)$ is plotted in red and $B_z(t;2)$ is plotted in blue. On right hand side, the respective 1^{st} derivatives are plotted.*

four successive derivatives of $B_z(t)$, (7.35), on lower left hand side we see four successive derivatives of $B_y(t)$, (7.33) and on lower right hand side we see four successive derivatives of $B_x(t)$, (7.34). The 1^{st} derivatives are plotted in red, the 2^{nd} derivatives in blue, the 3^{rd} derivatives in green and the 4^{th} derivatives are plotted in orange. In general, the derivatives have the same behaviour for all four ERB-functions. But there are some differences. For $B_d(t)$ the changes of the derivatives occur closer to $t = 0$ and $t = 1$ than for the others, and also, the maximum value of the derivatives grows faster for increasing order of the derivatives. On the opposite side is $B_z(t)$ which seems to be closest to the Fabius function.

In the following sections we will introduce four intrinsic parameters to the ERB-functions.

7.7.1 The slope parameter γ

The first intrinsic parameter is the slope parameter γ. We just add the slope parameter to (7.27), (7.28) and (7.30) and get,

$$\phi(t;\gamma) = e^{-\gamma\frac{\left(t-\frac{1}{2}\right)^2}{t(1-t)}}, \quad \psi(t;\gamma) = e^{\frac{-2\gamma}{t}e^{\frac{-\gamma}{1-t}}}, \quad \varphi(t;\gamma) = e^{\gamma\left(\frac{1}{t-1}+\frac{1}{t}\right)}, \quad \gamma \in \mathbb{R},\ \gamma > 0. \quad (7.38)$$

We can now adjust the slope of the ERB-functions. Note that the ERB-functions are still point symmetric. Figure 7.19 illustrates the effect of the slope parameter. Here $B_z(t;\gamma)$ is used as an example. The function and 1^{st} derivative are shown for a set of slope parameters. The functions can be seen on the left side of the figure and the 1^{st} derivatives on the right side. The least steep ERB-function is the function with slope parameter $\gamma = 0.8$. The function plotted in red is the default function with $\gamma = 1$. Then follows functions with $\gamma = 1.2$, $\gamma = 1.4$, $\gamma = 1.6$ and $\gamma = 1.8$, all plotted in black. Finally, in blue, is a function with $\gamma = 2$. Looking at the 1^{st} derivatives we see that the slope becomes steeper and steeper as γ grows. For $\gamma = 1$, the maximum value is 2, while for $\gamma = 2$, the maximum value is 4. For all four ERB-functions, the behaviour is quite similar. Note that for $B_d(t,\gamma)$, the constant $\mathbb{S}$ will also be affected by γ, ie $\mathbb{S}_\gamma = \left[\int_0^1 \phi(t;\gamma)dt\right]^{-1}$.

For $\psi(t;\gamma)$ (7.38), it is possible to replace γ with two slope parameters, i.e.

$$\psi(t;\gamma_1,\gamma_2) = e^{\frac{-\gamma_1}{t}e^{\frac{-\gamma_2}{1-t}}}.$$

The default connection between them is then $\gamma_1 = 2\gamma_2$. It is also possible to link them i.e. $\gamma_1 = 1+\gamma$ and $\gamma_2 = 1-\gamma$.

The slope parameter has the same effect on the shape of the B-function as the order parameter of the symmetric Beta-functions and the symmetric RB-functions.

7.7.2 The balance parameter μ

The next intrinsic parameter is the balance parameter μ. This parameter is analogous to the balance parameter in the Rμ-function, see Corollary 7.1 in subsection 7.4.2. We now also add a balance parameter to (7.27), (7.28) and (7.30) and get,

$$\phi(t;\gamma,\mu) = e^{-\gamma\frac{(t-\mu)^2}{t(1-t)}}, \qquad \gamma,\mu\in\mathbb{R}, \quad \gamma>0 \quad \text{and} \quad 0<\mu<1, \tag{7.39}$$

$$\psi(t;\gamma,\mu) = (1-\mu)\, e^{\frac{-2\gamma}{t}e^{\frac{-\gamma}{1-t}}}, \qquad \gamma,\mu\in\mathbb{R}, \quad \gamma>0 \quad \text{and} \quad 0<\mu<1, \tag{7.40}$$

$$\varphi(t;\gamma,\mu) = e^{2\gamma\left(\frac{1-\mu}{t}-\frac{\mu}{1-t}\right)}, \qquad \gamma,\mu\in\mathbb{R}, \quad \gamma>0 \quad \text{and} \quad 0<\mu<1. \tag{7.41}$$

Note that the balance parameter will give the following formulas to (7.33) and (7.34),

$$B_y(t;\mu) = \frac{(1-\mu)\psi(t)}{(1-\mu)\psi(t)+\mu\ \psi(1-t)}, \tag{7.42}$$

$$B_x(t;\mu) = \mu\,(1-\psi(1-t)) + (1-\mu)\psi(t), \tag{7.43}$$

For $B_d(t;\mu)$ the formula is straight forward, except that we have to calculate $\mathbb{S}_\mu$. For $B_z(t;\mu)$ the formula is just straight forward.

The effect is expected to be similar to that of the Rμ-function. The ERB-functions (7.39), (7.40) and (7.40) with a balance parameter $\mu = 0.5$ are point symmetric as the default definition in (7.27),(7.28) and (7.30), while all other values gives asymmetric functions.

In Figure 7.20 is $B_x(t;\mu)$ used as an example because the result for $B_x(t;\mu)$ is similar to $B_z(t;\mu)$ and $B_y(t;\mu)$. For $B_d(t;\mu)$ the behaviour is more similar to the RB-function with balance parameter (7.22), more like a parallel shift as we can observe in Figure 7.12.

In section 7.8 we will investigate what is called a balance-symmetry of B-functions.

7.7.3 The asymmetric tightening parameters α and β

The next intrinsic parameters are probably most of theoretical interest. We call them the asymmetric tightening parameters α and β. They are, in terms of shape, analogous to the order parameter a and b in the Beta- and RB-function, see section 7.3, Theorem 7.3 and section 7.4, Theorem 7.4. Remember that, unlike Beta- and RB-functions, these parameters do not effect the order of the ERB-function, because the order is infinite. The shape of the function is of cause effected.

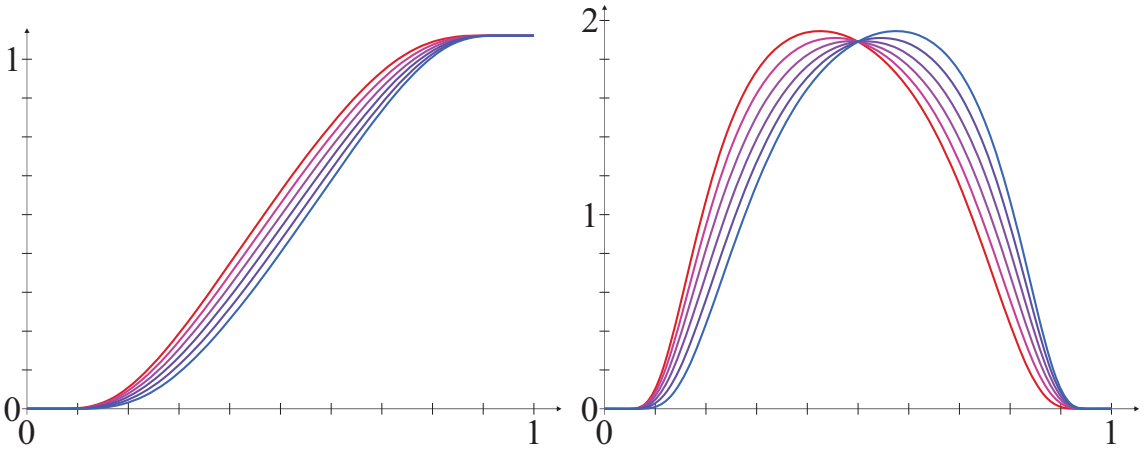


Figure 7.20: *On left hand side we see $B_x(t;\mu)$ for $\mu = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. $B_x(t;\mu = 0)$ is plotted in red and $B_x(t;\mu = 1)$ is plotted in blue. The other four are plotted in colors that gradually go from red to blue. On right hand side, the respective 1^{st} derivatives are plotted in the same color as their respective functions.*

We add this asymmetric tightening parameters to (7.39), (7.40) and (7.41). Including all intrinsic parameters we get the following new definition,

$$\phi(t;\gamma,\mu,\alpha,\beta) = e^{-\gamma\frac{|t-\mu|^{\alpha+\beta}}{t^{\alpha}(1-t)^{\beta}}}, \qquad \gamma,\mu,\alpha,\beta \in \mathbb{R}, \quad \gamma,\alpha,\beta > 0 \quad \text{and} \quad 0 < \mu < 1, \tag{7.44}$$

$$\psi(t;\gamma,\mu,\alpha,\beta) = \mu e^{\frac{-2\gamma}{t^{\alpha}}e^{\frac{-\gamma}{(1-t)^{\beta}}}}, \qquad \gamma,\mu,\alpha,\beta \in \mathbb{R}, \quad \gamma,\alpha,\beta > 0 \quad \text{and} \quad 0 < \mu < 1, \tag{7.45}$$

$$\varphi(t;\gamma,\mu,\alpha,\beta) = e^{2\gamma\left(\frac{1-\mu}{t^{\alpha}} - \frac{\mu}{(1-t)^{\beta}}\right)}, \quad \gamma,\mu,\alpha,\beta \in \mathbb{R}, \quad \gamma,\alpha,\beta > 0 \quad \text{and} \quad 0 < \mu < 1. \tag{7.46}$$

In [102, Theorem 2.1, page 18–24], a complete proof is given of that $B_d(t;\gamma,\mu,\alpha,\beta)$ is a complete B-function. The same proof will also be valid for the other ERB-functions.

The different ERB-functions will behave differently on these intrinsic parameters. For $B_d(t,\gamma,\mu,\alpha,\beta)$, increasing α and β, the plots tends toward a linear function on a part of the domain, see Figure 7.21 - **d)**. This is confirmed by Figure 7.21 - **a)**, where we see that the 1^{st} derivative goes towards something similar to "a doorway". However, the ERB-function is still C^{∞}-smooth, which can be seen in Figure 7.21 -**b)**.

If α and β are different, the curve will be offset parallel to the side with the lowest number and at the same time a little less steep. This can be observed in Figure 7.21 - **c)**. Remember that the integral of $B'_d(t)$ is always 1, so in the figure, if the "box" is lower it is also wider.

In Figure 7.22 - **a)**, $B_z(t;\alpha,\beta)$ is plotted for $(\alpha+\beta = 8)$ where α and β are positive integers. The function plots are almost vertical and according to the difference between α and β they are horizontally offset relative to one another.

In $B_d(t;\alpha,\beta)$ and $B_z(t;\alpha,\beta)$, α and β are antisymmetric, where α is associated to $t = 0$

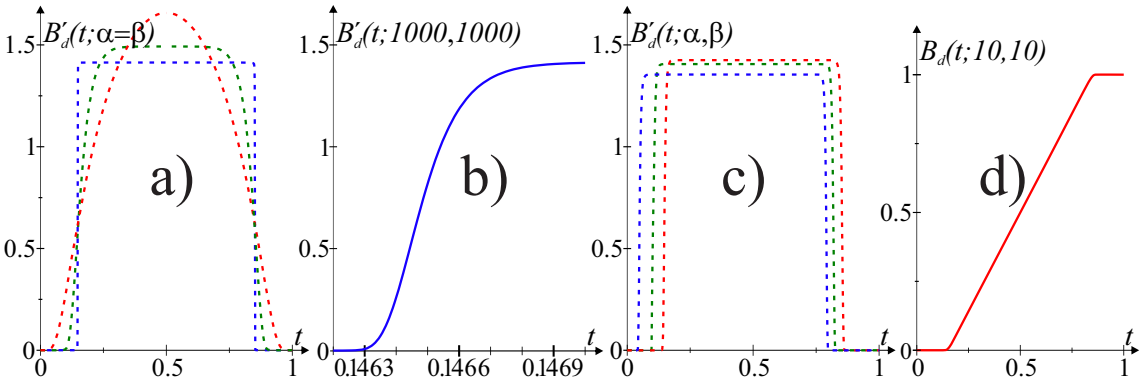


Figure 7.21: *In plot a) is $B'_d(t;\alpha=\beta=1)$ in dashed red, $B'_d(t;\alpha=\beta=3)$ in dashed green and $B'_d(t;\alpha=\beta=1000)$ in dashed blue. In plot b) is the t-axis of plot a) sharply scaled, and we can see a part of $B'_d(t;\alpha=\beta=1000)$. In plot c) is $B'_d(t;\alpha=\beta=20)$ in dashed red, $B'_d(t;\alpha=15,\beta=25)$ in dashed green and $B'_d(t;\alpha=10,\beta=30)$ in dashed blue. In plot d) is $B_d(t;\alpha=\beta=10)$ plotted in solid red.*

and β is associated to $t=1$. This unlike $B_x(t;\alpha,\beta)$ and $B_y(t;\alpha,\beta)$ where α and β act on two levels. In Figure 7.22 - **b)** is a plot of $B_x(t;\alpha,\beta)$ where $\alpha=1$ and $\beta=\{1,2,3,4,5,1000\}$. It look like that $\lim_{\beta\to\infty} B_x(t,1,\beta)=\frac{1}{2}$. A plot of $B_y(t;\alpha,\beta)$ with the same values for α and β gives a similar result.

In Figure 7.22 - **c)** there is a plot of $B_x(t;\alpha,\beta)$ where $\beta=1$ and $\alpha=\{1,2,3,4,5,1000\}$. From $\beta=1$ and up, the function starts to get steeper, but from $\beta=2$ it turns and seems to converge towards $\frac{1}{2}$, as in **b)** where β was varies. In Figure 7.22 - **d)** is a plot of $B_y(t;\alpha,\beta)$ where $\beta=1$ and $\alpha=\{1,2,3,4,5,10\}$. It behaves different and seems to converges towards a step function at $t=\frac{1}{2}$.

7.7.4 ERB-functions, differentiation

As an example, let's look at derivatives for $B_d(t)$. For the other ERB-functions, there will be similar procedure.

Let's call the exponent in (7.27), or the one with intrinsic parameters, for $g(t)$. In (7.36) is the 1^{st} derivatives given. The other derivatives can be expressed as,

$$B_d^{(j)}(t) = f_j(t)\, B'_d(t),$$

where $f_1(t)=1$. $f_j(t)$ can be expressed with $g(t)$. For the default version of $B_d(t)$ is

$$g(t)=\frac{h(t)}{s(t)}, \quad \text{where} \quad h(t)=1-2t \quad \text{and} \quad s(t)=(2t(1-t))^2. \tag{7.47}$$

Using derivatives of a product, we get the following expression for $f_j(t)$ for j up to 4,

$$\begin{aligned} f_2(t) &= g(t),\\ f_3(t) &= g'(t)+g(t)^2,\\ f_4(t) &= g''(t)+3g(t)g'(t)+g(t)^3. \end{aligned}$$

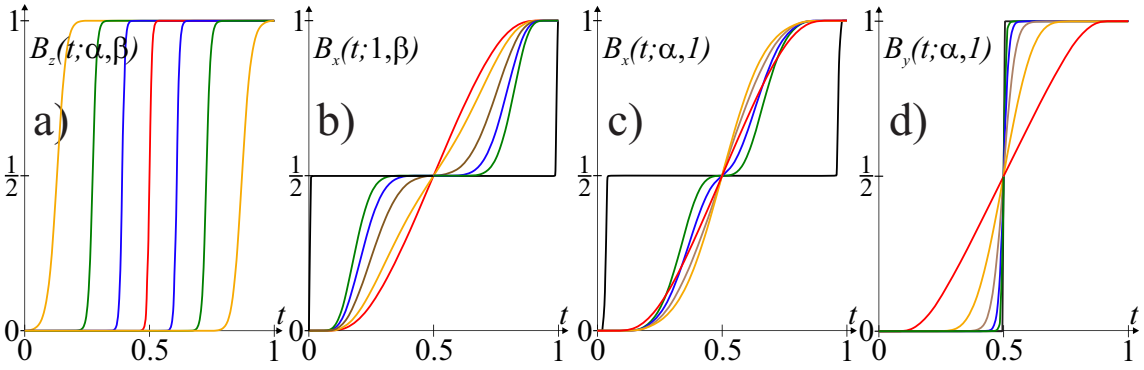


Figure 7.22: *In plot* **a)** *is from left* $B_z(t;\alpha=1,\beta=7)$ *in orange,* $B_z(t;2,6)$ *in green,* $B_z(t;3,5)$ *in blue,* $B_z(t;4,4)$ *in red,* $B_z(t;5,3)$ *in blue,* $B_z(t;6,2)$ *in green and* $B_z(t;7,1)$ *in orange. In plot* **b)** *is* $B_x(t;\alpha=\beta=1)$ *in red,* $B_x(t;1,2)$ *in orange,* $B_x(t;1,3)$ *in brown,* $B_x(t;1,4)$ *in blue,* $B_x(t;1,5)$ *in green and* $B_x(t;1,1000)$ *in black. In plot* **c)** *is* $B_x(t;\alpha=\beta=1)$ *in red,* $B_x(t;2,1)$ *in orange,* $B_x(t;3,1)$ *in brown,* $B_x(t;4,1)$ *in blue,* $B_x(t;5,1)$ *in green and* $B_x(t;1000,1)$ *in black. In plot* **d)** *is* $B_y(t;1,1)$ *in red,* $B_y(t;2,1)$ *in orange,* $B_y(t;3,1)$ *in brown,* $B_y(t;4,1)$ *in blue,* $B_y(t;5,1)$ *in green and* $B_y(t;10,1)$ *in black.*

For the default version of $B_d(t)$, we get the following expressions for $f_j(t)$ for j up to 4,

$$f_2(t) = \frac{h(t)}{s(t)},$$
$$f_3(t) = \frac{\frac{3}{2}h(t)^4-\frac{1}{2}}{s(t)^2},$$
$$f_4(t) = \frac{\left(3h(t)^6+\frac{3}{2}h(t)^4-5h(t)^2+\frac{3}{2}\right)h(t)}{s(t)^3}.$$

where $h(t)$ and $s(t)$ is given in 7.47.

7.8 Point-, Order- and Balance-symmetry of B-functions

The symmetry properties are important for the B-functions. It tells us how the local functions influence the result. in section 7.1, the point-symmetry of a B-function was introduced. It is that a B-function is symmetric about the point $(0.5,\ 0.5)$.

The Point-symmetry of a B-function

A B-function is symmetric or Point-symmetric if

$$B(t)+B(1-t)=1, \quad t\in[0,1]\subset\mathbb{R}.$$

This is the most important type of symmetry and tells us that the influence on the result of both local functions are the same. We have shown that for all groups and series of B-functions there exist Point-symmetric functions.

The next type of symmetry is Order-symmetry. This type of symmetry is opening for different orders, connected to knots in a spline-function based on blending.

The Order-symmetry of a B-function

A B-function is called Order-symmetric if $a,b \geq 0$, $a,b \in \mathbb{Z}$, i.e.

$$B_{a,b}(t) + B_{b,a}(1-t) = 1, \quad t \in [0,1] \subset \mathbb{R}, \tag{7.48}$$

where a and b are the (Hermite) orders that are changing side in the two functions.

Theorem 7.6. *The following B-functions are Order-symmetric:*

- *Beta-functions are Order-symmetric.*
- *RB-functions are Order-symmetric.*
- *The ERB-functions $B_d(t;\alpha,\beta)$ and $B_z(t;\alpha,\beta)$ are "Order"-symmetric in the sense that it is symmetric according to α and β.*

The proof of Theorem 7.6 can be found in appendix C.6.

The last type of symmetry is Balance-symmetry. This type of symmetry is opening for different balance and thus weight, connected to knots in a spline-function based on blending.

The Balance-symmetry of a B-function

A B-function is called Balance-symmetric if

$$B(t;\mu) + B(1-t;1-\mu) = 1. \tag{7.49}$$

Theorem 7.7. *The following B-functions are Balance-symmetric:*

- *Rμ-functions are Balance-symmetric.*
- *ERB-functions are Balance-symmetric.*

The proof of Theorem 7.7 can be found in appendix C.7.

As a conclusion of the section we will show that some B-functions can be Order-symmetric and Balance-symmetric simultaneously.

Simultaneous order and balance symmetry of B-functions

Theorem 7.8. *The following B-functions are Order-symmetric and Balance-symmetric simultaneously:*
The Rμ-functions fulfills $B_{a,b}(t;\mu) + B_{b,a}(1-t;1-\mu) = 1,$
The ERB-function B_d fulfills $B_d(t;\mu,\alpha,\beta) + B_d(1-t;1-\mu,\beta,\alpha) = 1.$
The ERB-function B_z fulfills $B_z(t;\mu,\alpha,\beta) + B_z(1-t;1-\mu,\beta,\alpha) = 1.$

The proof of Theorem 7.8 can be found in appendix C.8.

7.9 Implementing B-functions

For most B-functions and their derivatives, algorithms can easily be created directly from their formulas. An algorithm for calculating B-functions, including their derivatives is called an *evaluator*. It is easy to implement efficient evaluators for Beta-functions, Rational B-functions and Trigonometric B-functions for given intrinsic parameters. However,

it is more complicated to create a general evaluator for B-functions, and an evaluator for ERB-functions of type 1 is not possible to make directly, because it require numerical integrations and/or approximations.

In Appendix A is a numerical evaluator described. The evaluator was initially made for ERB-functions of type 1, but is later modified to other B-functions.[2] The evaluator is based on approximation of the B-function by piecewise 3^{rd}-degree Hermite polynomials, where the coefficients for all of the Hermite functions in all subareas are pre-computed and stored. The default partitioning is 1024 and this partitioning requires $1024 \times 6 \times 8 = 48k$-byte memory.

The error bounds are important and for ERB-functions is the error bound: 10^{-13} for the function value, 10^{-9} for the first derivative and 10^{-6} for the second derivative.

The efficiency of the evaluator is 6 multiplications for the function value, 4 multiplications for the first derivative, 5 multiplications for the second derivative ... C++ codes for the evaluator are present at https://source.coderefinery.org/gmlib/gmlib1/gmlib/-/tree/master/modules/parametrics/evaluators.

[2]The evaluator was introduced in [102], made for Expo-Rational B-splines. It was made for the scalable subset, which is practically identical with the ERB-function described here. Later Gancheva and Delistoyanova expanded the evaluator in [71], to also include Beta-functions.

Chapter 8

Blending splines

Recall the formula for a polynomial B-spline curve in (6.12), ie

$$c(t) = \sum_{j=0}^{n-1} c_j \, b_{d,j}(t).$$

where $\{c_j\}_{j=1}^{n}$, are the control points, ie the points that define the control polygon of the curve. The set of B-splines (basis functions) $\{b_{d,j}(t)\}_{j=0}^{n-1}$ is defined by the knot vector $\bar{t} = \{t_0, t_2, ..., t_{n+d}\}$ and the polynomial degree d. The order of the B-spline is $k = d+1$ and is the dimension of the function space.

Because $b_{d,j}(t)$ has local support, ie is different from zero only at the interval $t \in [t_j, \, t_{j+k})$, we can reformulate the formula for the B-spline curve as follows,

$$c(t) = \sum_{j=i-k}^{i} c_j \, b_{d,j}(t).$$

where the index i is determined by $t_i \leq t < t_{i+1}$. That is a specific formula for each knot interval $[t_i, t_{i+1})$.

In subsection 6.2.3, matrix formulation of B-splines was introduced. Using it, a third degree B-spline curve will have the following formula for each knot interval $[t_i, t_{i+1})$,

$$\begin{aligned} c(t) = \begin{pmatrix} 1 - w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} & \begin{pmatrix} 1 - w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1 - w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \\ & \begin{pmatrix} 1 - w_{3,i-2}(t) & w_{3,i-2}(t) & 0 & 0 \\ 0 & 1 - w_{3,i-1}(t) & w_{3,i-1}(t) & 0 \\ 0 & 0 & 1 - w_{3,i}(t) & w_{3,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}, \end{aligned} \tag{8.1}$$

where

$$w_{d,i}(t) = \frac{t - t_i}{t_{i+d} - t_i}, \tag{8.2}$$

and the index i is determined by $t_i \leq t < t_{i+1}$.

The matrix formulation illustrate that B-spline curves formula is based on corner cutting. Some important properties of B-spline curves comes from this corner cutting algorithm illustrated in the matrix form in (8.1). As described in section 6.2.2 the properties connected to the corner cutting are:

- The strong convex hull property,
- the variation diminishing property.

In addition can the following properties easily be seen from the matrix formula,

- the local modification scheme,
- the affine invariance property.

Local modification follows from that the formula (for degree 3) only depend on four points, and the affine invariance follows from that each row in all matrices sums up to 1.

The continuity property can be directly derived from (8.2), i.e. $w_{d,i}(t) : [t_i, t_{i+d}] \to [0,1]$. Recall the Hermite-order of the B-function in definition 7.2. The linear B-function $B(t) = t$ has Hermite-order zero, it further follows that t^2 has left side Hermite-order 1, t^3 has left side Hermite-order 2 etc. It thus follows that the Hermite-order of the B-spline basis is $d-1$ at the ends because it start with t^d, where t is translated and scaled, and ends with $(1-t)^d$, with a translated and scaled t. At the internal knots, only one linear function ends and starts, which then reduce the continuity from d to $d-1$. This is the explanation for the B-spline continuity at a simple knot, the continuity is the polynomial degree minus one. By the same arguments it follows that the continuity is reduced by 1 if two knots has the same value and that the continuity in general is the degree minus the multiplicity of the knots.

8.1 B-splines with B-function

Recall the matrices in (8.1). Most elements in the matrices are zero. At each row of the matrices, the non-zero elements have the values $w_{d,j}(t)$ and $1 - w_{d,j}(t)$. Since $w_{d,j}(t)$ is a linear function mapping t from $[t_j, t_{j+d}]$ to $[0,1]$, each row is a linear interpolation between to points. It is still a linear interpolation if instead of linear functions we use higher order symmetric B-function, Definition 7.1, as elements of the B-spline matrices. We then get the following expression for B-spline curves:

$$c(t) = \begin{pmatrix} 1 - B \circ w_{1,i}(t) & B \circ w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1 - B \circ w_{2,i-1}(t) & B \circ w_{2,i-1}(t) & 0 \\ 0 & 1 - B \circ w_{2,i}(t) & B \circ w_{2,i}(t) \end{pmatrix}$$
$$\begin{pmatrix} 1 - B \circ w_{3,i-2}(t) & B \circ w_{3,i-2}(t) & 0 & 0 \\ 0 & 1 - B \circ w_{3,i-1}(t) & B \circ w_{3,i-1}(t) & 0 \\ 0 & 0 & 1 - B \circ w_{3,i}(t) & B \circ w_{3,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}, \tag{8.3}$$

where the knot interval and thus index i is determined by $t_i \leq t < t_{i+1}$.

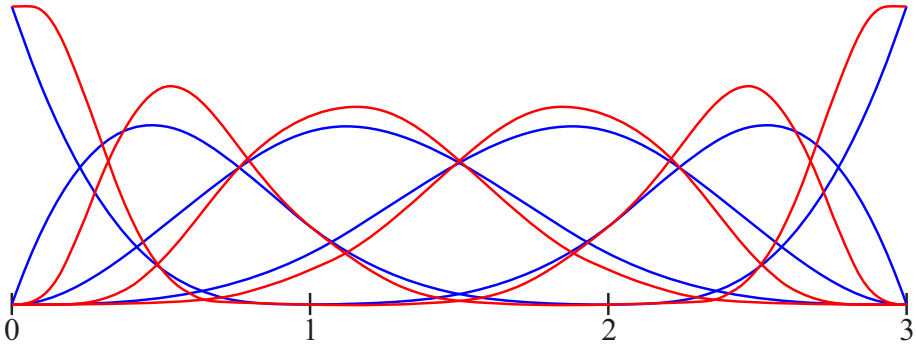


Figure 8.1: *The blue basis functions are the polynomial B-splines on a given knot vector* $\vec{t} = \{0,0,0,0,1,2,3,3,3,3\}$. *The red basis functions are the B-spline with a* 2^{nd}*-order B-function on the same knot vector.*

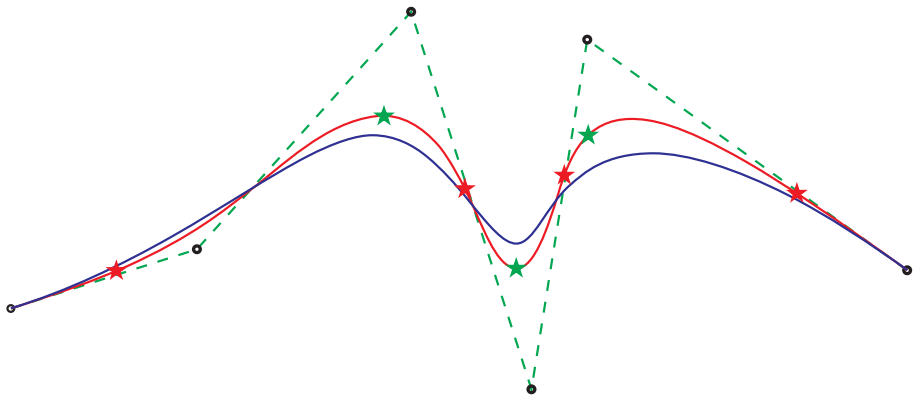


Figure 8.2: *A plot of a* 3^{rd}*-degree polynomial B-spline curve in blue. The dashed green lines are the control polygon.The red curve is a B-spline with B-functions curve with the same knot vector and control points as the polynomial B-spline. The red and green stars marks extreme values of the speed of the red-curve.*

The extension do not change the main properties of the B-spline curve. We still have the corner cutting algorithm inducing all the properties of convex hull, local modification scheme, variation diminishing etc, and the linear interpolation scheme induce the affine invariant property.

The main difference compared to polynomial B-splines is the continuity properties. The continuity at the knots will increase with the order of the B-function. If the B-function has order infinity, then the curve will be C^{∞}-smooth.

In Figure 8.1 is the B-splines (the basis functions), plotted for both polynomial B-splines in blue, and for B-splines with a B-function in red. The main difference we can observe, just by looking at Figure 8.1, is that the derivatives of all basis functions with a B-function are zero at the start and at the end of their support. Most clearly can this be seen for the first two basis function at $t = 0$, and for the last two basis function at $t = 3$. At the internal knots we can observe that the basis functions are closer to zero over a larger area.

In Figure 8.2 is a third degree polynomial B-spline curve plotted in blue. The knot vector

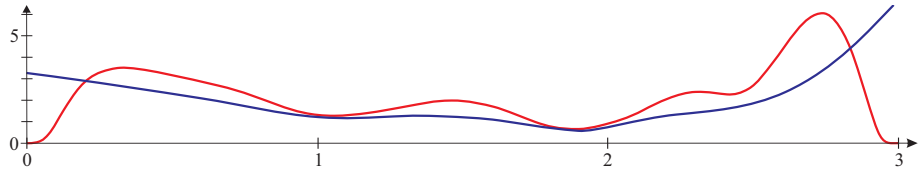


Figure 8.3: *The speed of the curves in Figure 8.2. The blue function is the speed of the polynomial B-spline and the red function is the speed of B-spline with B-functions.*

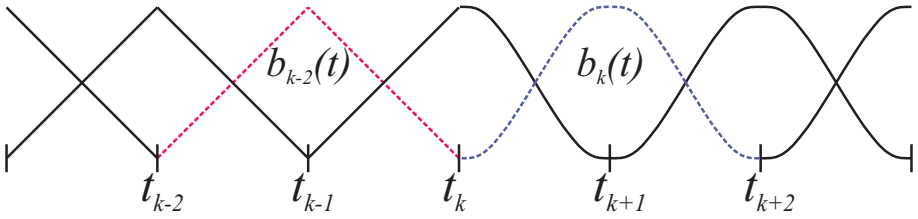


Figure 8.4: *A plot of 2^{nd} order B-splines. To the left polynomial 2^{nd} order (1^{st}-degree) B-splines, to the right 2^{nd} order Expo-Rational B-splines.*

(spline space) is the same as in Figure 8.1. The 6 control points are marked with circles. The red curve is the B-spline with B-functions curve based on the same knot vector and control points, but where we have used the basis function plotted in red in Figure 8.1.

The examples in Figure 8.2 and also the plot of the respective velocities in Figure 8.3 show one special property of a B-spline with B-functions curve. The red stars in Figure 8.2 mark the points where the speed has a local maximum, and the green stars mark the points where the speed has local minimum. We can clearly see that the stars correlate with the curvature, where the curvature is smallest the speed is highest, and where the curvature is highest the speed is lowest. It can be reminiscent of driving a car over a given distance, you start with zero for speed, acceleration, ..., ie all derivatives, and you regulate the speed with respect to the road bends and you end by stopping, and where then all "derivatives" are zero.

8.1.1 2^{nd} order B-splines with B-function

The formula for a 2^{nd} order B-spline curve is quite simple also if we include B-functions, the formula for a knot interval $[t_i, t_{i+1})$ is

$$c(t) = \begin{pmatrix} 1 - B \circ w_{1,i}(t) & B \circ w_{1,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-1} \\ c_i \end{pmatrix}, \tag{8.4}$$

which obviously is a curve of strait lines between the subsequent control points due to the linear interpolation in the formula. But if the B-function is an ERB- or Fabius-function then the curve is actually C^∞-smooth, which is quite strange since the curve is piecewise linear. The explanation for this is that all derivatives are zero at all control points as will be shown later.

In Figure 8.4 the basis functions are plotted, to the left polynomial 2^{nd} order (1^{st}-degree) B-splines, to the right 2^{nd} order Expo-Rational B-splines. The formulas follow from

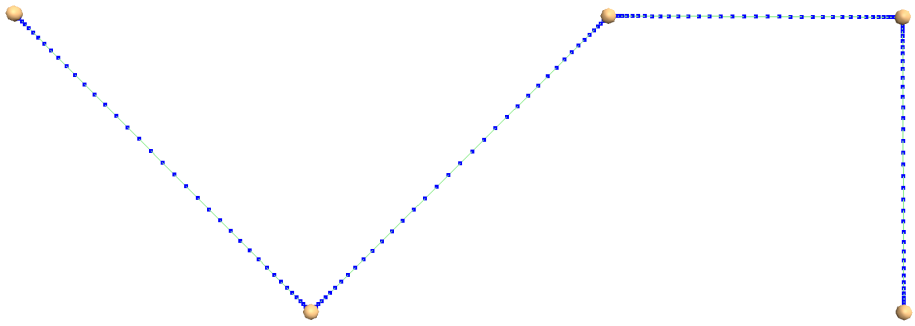


Figure 8.5: *A 2^{nd} order Expo-Rational B-spline curve is plotted in dotted blue. The points are plotted uniformly according to parameter values. Thus the density of points will describe the velocity. Note that the curve is C^∞-smooth, but only G^0.*

(6.10) and (8.4) and are

$$b_{1,i}(t) = \begin{cases} B \circ w_{1,i}(t), & t_i < t \leq t_{i+1}, \\ 1 - B \circ w_{1,i+1}(t), & t_{i+1} < t < t_{i+2}, \\ 0, & \text{otherwise.} \end{cases}$$

For a polynomial B-spline is $B(t) = t$, otherwise $B(t)$ can be any symmetric B-function as described in Chapter 7.

In Figure 8.5, a 2^{nd} order Expo-Rational B-spline curve is plotted. The points are plotted uniformly according to parameter values. Thus the density of points will describe the velocity. In the figure we see that the points become denser and denser the closer to the control points we are.

To verify the statement at the beginning of the section that all derivatives are zero in the control points and the observation in Figure 5, we now derive (8.4) and get

$$\begin{aligned} c'(t) &= \begin{pmatrix} -\delta_{d,i} B' \circ w_{1,i}(t) & \delta_{d,i} B' \circ w_{1,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-1} \\ c_i \end{pmatrix}, \\ &= \delta_{d,i} B' \circ w_{1,i}(t) \ (c_i - c_{i-1}) \end{aligned}$$

where $\delta_{d,i}$, defined in (6.13), is the derivative of $w_{1,i}(t)$, defined in (6.11). $\delta_{d,i}(c_i - c_{i-1})$ is the vector from control point c_{i-1} to c_i, scaled with the parameter interval. It thus follows that

$$c^{(j)}(t) = \delta_{d,i}^j B^{(j)} \circ w_{1,i}(t) \ (c_i - c_{i-1}), \quad \text{for} \quad j = 1, 2, \ldots. \tag{8.5}$$

As we see in 8.5 the derivatives are dependent on the derivatives of the selected B-function. If the B-function is an ERB-function or the Fabius function, then all derivatives will be zero at all control points. This also corresponds to Theorem 7.1 even though we only have points here.

Here, too, the car driving analogy can be advantageous to use, you start with zero speed, acceleration,..., then you start to increase the speed to midway between two points, then you slow down until you stop at the point. There you turn before you drive on. In this way you drive in a straight line between all the points in a mathematically C^∞-smooth way while geometrically it is only G^0-smooth.

8.2 2^{nd} order B-splines as blending splines

2^{nd} order B-splines with B-functions have some very special properties, such as

- the basis functions have minimal support, ie over two knot intervals,
- the curve interpolates the control points, this follow from order 2
- the curve is C^k-smooth where k is the order of the B-function,

Together with the fact that the construction is very simple, these properties are the reason for using 2^{nd}-order B-splines with B-functions as a basis for blending-splines.

The formula of a 2^{nd}-order B-spline curve, where the control points are replaced by control curves (also called local curves), and where the basis functions are generated from a knot vector $\tau = \{t_i\}_{i=0}^{n+1}$ combined with B-functions, is

$$c(t) = \sum_{i=0}^{n-1} c_i(t)\, b_{1,i}(t), \quad \text{for} \quad t \in [t_1, t_n], \tag{8.6}$$

where $c_i(t),\ i = 0,1,...,n-1$ are local curves each defined over the two knot intervals (t_i, t_{i+2}), and where $b_{1,i}(t)$ are 2^{nd}-order B-splines with a symmetric B-function according to Definition 7.1, as shown in (8.4). We call this construction for blending splines. You will also find them under the name ERBS or GERBS in several papers[1]. Due to the 2^{nd} order and thus minimal support, the formula (8.6) can, on the knot interval $t \in [t_i, t_{i+1}]$, be simplified to

$$\begin{aligned} c(t) &= \begin{pmatrix} 1 - B\circ w_{1,i}(t) & B\circ w_{1,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-1}(t) \\ c_i(t) \end{pmatrix} \\ &= c_{i-1}(t) + B\circ w_{1,i}(t)\, \Delta c_i(t), \end{aligned} \tag{8.7}$$

where $\Delta c_i(t) = c_i(t) - c_{i-1}(t)$. This is according to two function blending described in Section 7.2 and is the same as expression (7.3).

Definition 8.1. *A 2^{nd} order B-splines with B-functions, and with control curves, also called local curves is*

Blending splines

It follows from (7.6) that for $j = 0,1,2,\ldots,S$ (the order of the B-functions used) is

$$c^{(j)}(t_i) = c_{i-1}^{(j)}(t_i), \qquad \text{for} \quad i = 1,2,\ldots,n, \tag{8.8}$$

$$c^{(j)}(t) = c_{i-1}^{(j)}(t) + \sum_{k=0}^{j} \binom{j}{k} \delta_{1,i}^{k} B^{(k)} \circ w_{1,i}(t)\, \Delta c_i^{(j-k)}(t) \quad \text{when} \quad t_i < t < t_{i+1}, \tag{8.9}$$

where $\delta_{1,i}$ is defined in (6.13). (8.8) is according to the Hermite interpolation property described in Theorem 7.1. Expression (8.9) looks complicated but has the same structure as the Bernstein polynomials where the numbers follows Pascals triangle.

[1] ERBS and later GERBS were first presented at "Mathematical Methods for Curves and Surfaces-conference" in Tromsø 2004 and later published several places, [104], [44], [45], [102], [47], [6],...

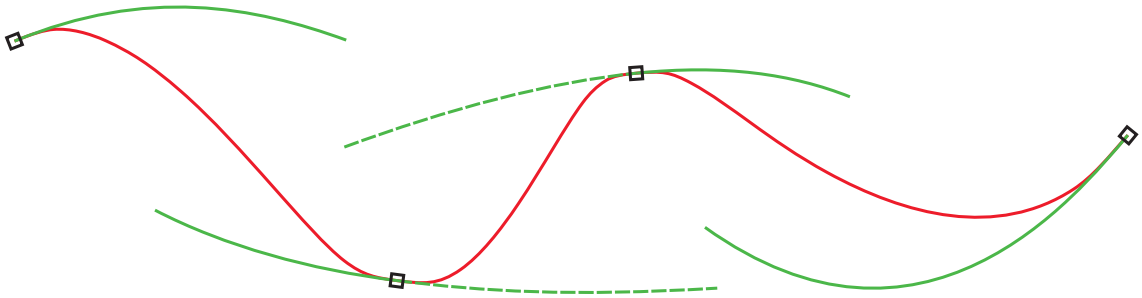


Figure 8.6: *A blending spline, ie a 2^{nd} order B-spline curve in red, with four local curves in green. The resulting curve is a blending of its local curves, where B-functions are the blending functions. The resulting curve also completely interpolates all existing derivatives to each of the adjacent local curves at the "middle" knot.*

2^{nd} order B-spline with B-function

To simplify the notation of the basis functions including its derivatives, we use

$$B_i^{(k)}(t) = \delta_{1,i}^k B^{(k)} \circ w_{1,i}(t), \quad \textit{for} \quad k = 0,1,\ldots \quad \textit{and} \quad t \in [t_i, t_{i+1}]. \tag{8.10}$$

Then we call the second term of (8.9) for: $g(t) = \sum_{k=0}^{j} \binom{j}{k} \delta_{1,i}^k B^{(k)} \circ w_{1,i}(t)\, \Delta c_i^{(j-k)}(t)$. The function value and the 1^{st}, 2^{nd} and 3^{rd} derivatives for this second term of (8.9) are,

$$\begin{aligned}
g_i(t) &= B_i(t)\, \Delta c_i(t), \\
g_i'(t) &= B_i'(t)\, \Delta c_i(t) + B_i(t)\, \Delta c_i'(t), \\
g_i''(t) &= B_i''(t)\, \Delta c_i(t) + 2B_i'(t)\, \Delta c_i'(t) + B_i(t)\, \Delta c_i''(t), \\
g_i'''(t) &= B_i'''(t)\, \Delta c_i(t) + 3B_i''(t)\, \Delta c_i'(t) + 3B_i'(t)\, \Delta c_i''(t) + B_i(t)\, \Delta c_i'''(t).
\end{aligned} \tag{8.11}$$

We now summarize one of the main property of blending splines.

The Hermite interpolation property of Blending splines

A blending splines defined in Definition 8.1 interpolates each local curve c_i in the knot value t_{i+1}, not only the value but with all subsequent derivatives up to the order of the B-function used, as shown in expression 8.8. This is called the Hermite-interpolation property of the blending spline.

Are there any restrictions on the local curves? The answer is yes, but for all practical use it is almost impossible to find examples of curves that cannot be used as local curves. Of theoretical interest, these restrictions are described in [102], Section 2.6. The most common types of local curves are Bézier curves, sub-curves, circular arcs, blending splines itself, B-splines and curves made by Taylor expansion. Of course, the continuity of the resulting curve depends on the continuity of the local curves used.

Figure 8.6 shows an example of a blending spline curve and its local curves. There are 4 local curves (green). The knot vector is $\tau = \{t_i\}_{i=0}^5$, where $t_0 = t_1$ and $t_4 = t_5$ are the multiple start and end knots, and t_2 and t_3 are non-multiple internal knots. Each basis function and thus also local curves spans two knot intervals, and each local curve

interpolates the global curve at its middle knot of its span. Because the first two knots are equal and t_1 is the middle knot, the first local curve, which spans $[t_0, t_2]$, will interpolate the start of the global curve. The same reasoning also explains why the last local curve interpolates the end of the global curve. The curve in Figure 8.6 can be divided into three parts, and the "division points" are where the local curves touch the global curve, ie the internal knots t_2 and t_3. Each part of the curve is a blending of parts of two local curves. The first part is a blending of the whole first local curve and the first half (until knot t_2) of the second local curve. The second part is a blending of the second part (from knot t_2) of the second local curve and the first part (until knot t_3) of the third local curve (both parts dashed green). The third part is a blending of the second part (from knot t_3) of the third local curve and the whole fourth local curve. This shows the extreme local support, ie if we change the first local curve, only the first third of the global curve will be changed.

A general algorithm for blending splines is as follows:

Algorithm 8. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes $\{c^{(j)}(t)\}_{j=0}^{d}$ *for a blending spline curve. The algorithm assumes that evaluators for the local curves and for the B-function are present. The knot vector* $\{t_i\}_{i=0}^{n+1}$ *is also assumed to be present. The input variables are:* $t \in [t_1, t_n]$, *and* d *- which is the number of derivatives to compute. The return is a* vector⟨Vector⟩, *where the first element contains* $c(t)$, *and then* $c'(t), \ldots, c^{(d)}(t)$.

vector⟨Vector⟩ **eval** (double t, int d)
- int $i = i : \backslash ;\ t_i \le t < t_{i+1}$; — *// Index for the current knot-interval.*
- vector⟨Vector⟩ $c_0 = \{c_{i-1}^{(j)}(t)\}_{j=0}^{d}$; — *// Result evaluating local curve* $c_{i-1}(t)$.
- if $(t == t_i)$ return c_0; — *// Return only local curve* $c_{i-1}(t)$, *see (8.8).*
- vector⟨Vector⟩ $c_1 = \{c_i^{(j)}(t)\}_{j=0}^{d}$; — *// Result evaluating local curve* $c_i(t)$.
- vector⟨double⟩ $a(d+1)$; — *// Vector to store "Pascals triangles nr".*
- vector⟨double⟩ $B = \{B_i^{(j)}(t)\}_{j=0}^{d}$; — *// Computing B-function, see (8.10).*
- $c_1 -= c_0$; — *//* c_1 *now becomes* Δc_i, *see (8.9).*
- for (int j=0; $j \le d$; j++)
  - $a_j = 1$;
  - for (int k=j-1; $k > 0$; $k--$)
    - $a_k += a_{k-1}$; — *// Computing "Pascals triangle"-numbers.*
  - for (int k=0; $k \le i$; k++)
    - $c_{0,j} += (a_k\, B_k) c_{1,j-k}$; — *// "vector += scalar*vector", from (8.9).*
- return c_0;

In the Algorithm, the computation of the local curves is marked in red.

8.2.1 Affine transformations of local curves

Affine maps are described in (2.4) on page 16. These are typically scaling, rotation, share and translation, and in general taken on the familiar form

$$Ap + v,$$

where p is a point and v is a associated vector in an affine space. This affine space might typical be 3D, where A is a 3×3 matrix

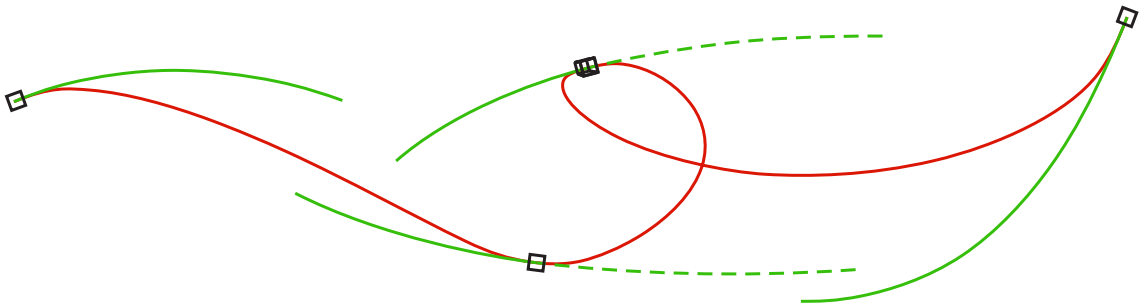


Figure 8.7: *The blending spline curve from Figure 8.6, but where three of the local green curves are translated and two are also rotated. The curve is in the 3D-space.*

However, if we use homogeneous coordinates, see Section 2.6 and 3.2, everything will be gathered in one 4×4 matrix (in 3D) for all affine maps. This matrix then looks like this;

$$A_i = \begin{bmatrix} x_{i,x} & y_{i,x} & z_{i,x} & p_{i,x} \\ x_{i,y} & y_{i,y} & z_{i,y} & p_{i,y} \\ x_{i,z} & y_{i,z} & z_{i,z} & p_{i,z} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{8.12}$$

The first three columns of the matrix are vectors where the homogeneous coordinate is 0, the last column is a point where the homogeneous coordinate is 1. The matrix represents the position and orientation of a local curve, and describes a local coordinate system where the point p_i is the origin and the three column vectors are the coordinate axes x_i (red), y_i (green) and z_i (blue). If we restrict the maps to rotation, scaling and translation, we keep an orthogonal coordinate system.

- To translate with a vector d - we compute $p_i = p_i + d$,
- To rotate an angle β around an axis r - we compute $x_i = R_{\beta,r}\, x_i$ and $y_i = R_{\beta,r}\, y_i$ and $z_i = R_{\beta,r}\, z_i$ where $R_{\beta,r}$ is a rotation matrix described in (2.5).
- To scale with a factor s, we scale each vector $x_i = s\, x_i$ and $y_i = s\, y_i$ and $z_i = s\, z_i$.

In Algorithm 8, we see two places marked in red, these are the places where the local curves are calculated. To implement affine transformations of local curves, we make these computations

$$c_i^{(j)}(t) = A_i\, \alpha_i^{(j)}(t), \quad j = 1, 2, ...,$$

where A_i is a matrix as (8.12) associated with the local curve $\alpha_i(t)$. Note that $\alpha_i(t)$ and thus $c_i(t)$ are points where the homogeneous coordinate is 1, and for $j > 0$, $\alpha_i^{(j)}(t)$ and thus $c_i^{(j)}(t)$ are derivatives and thus vectors where the homogeneous coordinate is 0.

In Figure 8.7, the curve in Figure 8.6 (embedded in $\mathbb{R}^3$) is changed by moving and rotating some of the local curves. The second curve is only moved, the third curve is moved and rotated about a vertical axis and the fourth curve is moved upwards and rotated slightly around an axis out of the page.

As we see in Figure 8.7, the rotations of the local curves are made around the interpolation points of the curves $c_i(t_i)$. Another observation / possibility is that rotation axis often

is connected to the tangent or cotangent of the curve at the interpolation points. This provides some guidance on what local coordinate systems should look like.

✓ Therefore, in general it is preferable that the local origin of a local curve is at the point where the blending curve interpolates the local curve. The way to do this will depend on the type of local curves, but in general we find the position of the interpolation point, $p_i = c_i(t_{i+1})$, then p_i is inserted as the last column in the matrix A_i and we correct the intrinsic data in the local curve with $-p_i$.

✓ It can be useful to make the **Frenet frame**, the matrix $F(t_{i+1})$, also called a TNB frame, see subsection 4.1.2. Usually it is a row matrix, but in our case we want a column matrix. It is a matrix where the first column is $T = c_i'(t_{i+1})$ normalized, the second column is $N = c_i''(t_{i+1}) - \langle c_i''(t_{i+1}), T\rangle T$ normalized, and the third column is $B = T \wedge N$. Since the matrix is orthogonal $F^{-1} = F^T$. Thus, we correct the intrinsic data in $c_i(t)$ with $F^{-1}(t_{i+1})$ and set $F(t_{i+1})$ as the upper left submatrix in A_i.

8.2.2 Bézier-curves as local curves

Bézier curves are very convenient to use as local curves. Recall from Section 4.4 expression (4.29) that Bézier curves are defined by

$$\alpha(t) = \sum_{j=0}^{d} c_j \, b_{d,j}(t), \quad \text{for } t \in [0,1],$$

where d is the polynomial degree, the basis functions $b_{d,j}(t)$ are the Bernstein polynomials described in Subsection 4.4.1, $c_j \in \mathbb{R}^s$, $j = 0,1,...d$ are the coefficients and thus the control polygon, and where s usually is 2 or 3 (in the plane or the 3D-space).

From Definition 8.1 we can see that the domain of each local curve $c_i(t)$ must be $[t_i, t_{i+2}]$. This means that each Bézier curve $c_i(t)$ must be re-parameterized from $[0,1] \to [t_i, t_{i+2}]$. To do this, the linear translation and scaling functions and its derivative are used,

$$w_{2,i}(t) = \frac{t - t_i}{t_{i+2} - t_i} \quad \text{and} \quad \delta_{2,i} = w_{2,i}' = \frac{1}{t_{i+2} - t_i}. \tag{8.13}$$

So for local Bézier curves we get the following evaluator, which includes position and d derivatives, which reflects (8.8) and (8.9), and which can be used where it is marked red in Algorithm 8.

$$c_i^{(j)}(t) = \delta_{2,i}^j \alpha_i^{(j)} \circ w_{2,i}(t), \quad j = 0,1,\ldots,d \quad \text{and} \quad t \in [t_i, t_{i+2}]. \tag{8.14}$$

In section 4.4.3 is the Bernstein-Hermite matrix $\mathbf{B}_d(t,\delta)$ described in (4.36). This is the matrix we use for computing the position and all derivatives at a given t-value, ie

$$\begin{pmatrix} c_i(t) \\ c_i'(t) \\ \vdots \\ c_i^{(d)}(t) \end{pmatrix} = \begin{pmatrix} b_{0,d}(w_{2,i}(t)) & b_{1,d}(w_{2,i}(t)) & \cdots & b_{d,d}(w_{2,i}(t)) \\ \delta_{2,i}\, Db_{0,d}(w_{2,i}(t)) & \delta_{2,i}\, Db_{1,d}(w_{2,i}(t)) & \cdots & \delta_{2,i}\, Db_{d,d}(w_{2,i}(t)) \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{2,i}^d D^d b_{0,d}(w_{2,i}(t)) & \delta_{2,i}^d D^d b_{1,d}(w_{2,i}(t)) & \cdots & \delta_{2,i}^d D^d b_{d,d}(w_{2,i}(t)) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_d \end{pmatrix}$$

and in vector/matrix notation

$$\{c_i^{(j)}(t)\}_{j=0}^{d} = \mathbf{B}_d(w_{2,i}(t), \delta_{2,i})\ \mathbf{c}. \tag{8.15}$$

To compute (8.15) and thus (8.14) we can use Algorithm 2 which creates the matrix $\mathbf{B}_d(w_{2,i}(t), \delta_{2,i})$, described in section 4.4.3, and multiply this matrix by a vector of the control points $\mathbf{c}$. If we want to compute fewer than d derivatives, Algorithm 2 can be modified to reduce the number of rows in the matrix. In both Figure 8.6 and Figure 8.7, 2^{nd}-degree Bézier curves are actually used as local curves of the blending spline.

In order for all local Bézier curves to have their local coordinate system, where the local origin is where the blending curve interpolates the local curve, we must correct the control points. We do this by finding the local origin, $p_i = \alpha_i \circ w_{2,i}(t)$, and then going through all the control points to correct them to $c_j = c_j - p_i$, for $j = 0, 1, \ldots, d$. As a counterweight, the last column in the homogeneous matrix A_i must be set to p_i.

In Figure 8.7 is the local Bézier curves rotated around their local origin.

8.2.3 Making a blending spline approximation of a curve

Given a curve $\varphi(t)$, with a domain $t \in [a,b]$ for an open curve or $t \in [a,b)$ for a closed/-cyclic curve. To make an approximate "copy" of this curve we need

- the number n and the polynomial degree d of the local Bézier curves to make,
- to create a knot vector, $\{t_i\}_{i=0}^{n+1}$ spanning the domain of $\varphi(t)$,
- to find the control points $\{\mathbf{c}_{i,j}\}_{i=0}^{d}$ of all local Bézier curves $c_j(t),\ j = 0, 1, \ldots, n-1$.

The simplest way to create a knot vector is to use uniform distance between the internal knots $t_1, \ldots, t_n$, where $[t_1, t_n]$ is the domain of $\varphi(t)$. The two end knots is then set according to that $\varphi(t)$ is open or closed (see section 6.2.2). Then recall expression (8.15). We must use as many derivatives as the polynomial degree we have chosen, therefor the matrix $\mathbf{B}_d(w_{2,i}(t), \delta_{2,i})$ is a $d+1 \times d+1$ matrix (made by Algorithm 2). From the original curve $\varphi(t)$ we therefore need the position and d subsequent derivatives in each knot value $t_1, t_2, \ldots, t_n$. We now reverse expression (8.15) and get the following procedure to make a blending spline copy of a curve.

Copying with Hermite interpolation

This method of "copying" curves is actually an **Hermite interpolation**. We get

$$\mathbf{c}_{i-1} = \mathbf{B}_d(w_{2,i}(t), \delta_{2,i})^{-1}\ \overline{\varphi}(t_i), \quad \text{for} \quad i = 1, 2, \ldots, n,$$

where $\mathbf{c}_{i-1}$ are the control points of the local Bézier curve c_{i-1}, and $\overline{\varphi}(t_i)$ are the position and d subsequence derivatives to the original curve in the parameter value t_i. Thus, at every internal knot value, the position and d derivatives of the blending spline $c(t)$ will be equal the position and d derivatives of the original curve $\varphi(t)$, ie

$$c^{(j)}(t_i) = \varphi^{(j)}(t_i), \quad j = 0, 1, \ldots, d \quad \text{and} \quad i = 1, 2, \ldots, n.$$

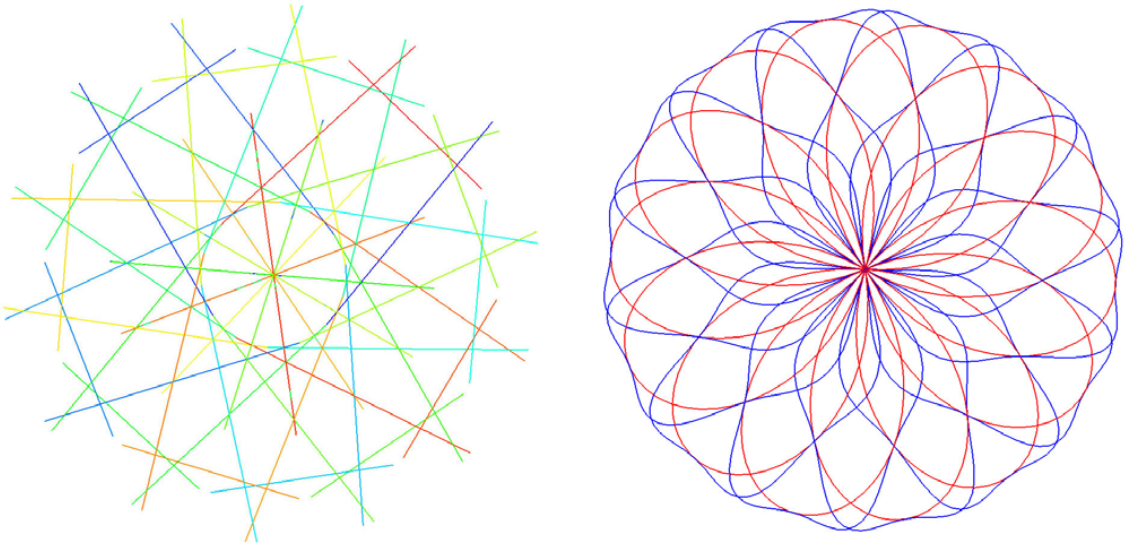


Figure 8.8: *To the right, a blending spline curve (blue) approximating a "Rose-curve" (red) by using 56 interpolation points; only the position and the first derivative in each point are used. This approximation seems to make the curve too long, so that the curve is buckling between the interpolation points. To the left, the 56 local Bézier curves (lines) of the blending splines plotted in different colors (all 1^{st}-degree). Because the "Rose-curve" has 14 petals, and there are 14*4=56 interpolation points, the local curves (lines) are "symmetrical around the rose center". In addition, every pair of subsequent lines intersect.*

8.2.4 Examples

We will now see examples of Hermite-interpolation using three different original curves. These three curves are approximated by blending spline curves using local Bézier curves of different degrees. The purpose is to show some of the properties and possibilities of blending spline curves using local Bézier curves. Most of the examples are closed/cyclic curves, but there is also one example of an open curve. The examples clearly show that Hermite-interpolation is not an "optimal" approximation (a well known fact), and that it is possible to improve the solution by scaling the local curves, or the local domain in the input of the interpolation process.

The first example is a so called "Rose-curve" described in [162], and defined by the formula,

$$g(t) = \begin{pmatrix} \cos t \, \cos\left(\frac{7}{4}t\right) \\ \sin t \, \cos\left(\frac{7}{4}t\right) \\ 0 \end{pmatrix} \quad \text{for} \quad t \in [0, 8\pi). \tag{8.16}$$

A plot will look like a rose with 14 petals. The number of petals is 2 times the numerator in the fraction in the cosine in equation (8.16). The speed is oscillating between 1 and 1.75, slowest at the center of the rose and the fastest at the tip of each petal.

In Figure 8.8, the "Rose-curve" is interpolated by a blending spline using 4 interpolation points on each petal. In total, there are 4*14=56 interpolation points uniformly distributed

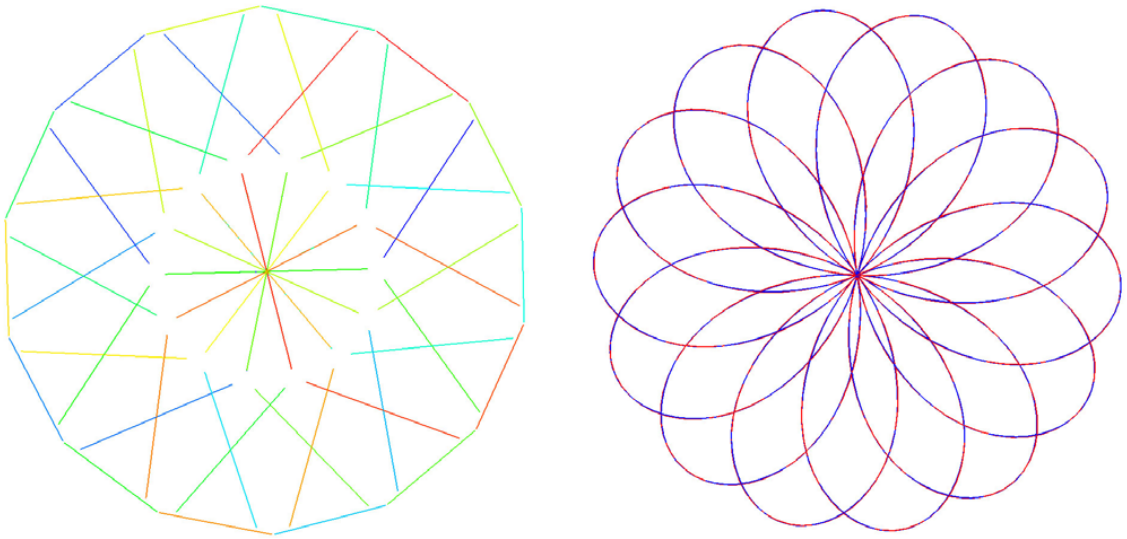


Figure 8.9: *To the right a blending spline curve (blue) approximating a "Rose-curve" (red) by using 56 interpolation points; only the position and the first derivative in each point are used, but the local curves are scaled by 0.5 after the interpolation. This approximation seems to be, geometrically, very close to the original curve. To the left, the 56 local Bézier curves (lines) of the blending splines (all 1^{st}-degree) are plotted in different colors. Because the "Rose-curve" has 14 petals, and there are 14*4=56 interpolation points, the local lines are "symmetrical around the rose center". In addition, each line does not intersect with the previous and next line in the sequence.*

on the parameter interval. The position and first derivative in each of the interpolation points are used, and we can easily see the effect of using only 1^{st}-derivatives. Since we do not specify 2^{nd}-derivatives, the curvature is thus assumed to be zero at all intersection points. But the curvature here is greater than zero, and the result is that the curve becomes too long. This can clearly be seen to the right in Figure 8.8. To the left in the figure, the local curves are plotted. Because all local curves are of degree 1, they are straight lines. The length of the lines, and the fact that they intersect each other, also indicates that the resulting curve becomes too long and thus will bulge out.

The curve in Figure 8.9 is made with the same interpolation as in the previous example (Figure 8.8). But the local curves are now scaled by 0.5, as can be seen on the left in Figure 8.9. When scaling, it is important that the interpolation point is the origin of the local coordinate system of each local curve, so that the interpolation points do not moving. The resulting curve is geometrically very similar to the original "Rose-curve". However, the speed will oscillate more strongly. It is possible to get the same result by using another method. You can scale the input derivatives instead of scaling the resulting curve, and still get the same result. As can be seen in this example, the result is geometrically very good. However, an even greater potential for improving the result is if the scaling factor differs from local curve to local curve.

In Figure 8.10, the "Rose-curve" is, as in the two previous examples, using 56 interpolation points, but now the position and two derivatives in each interpolating point are used.

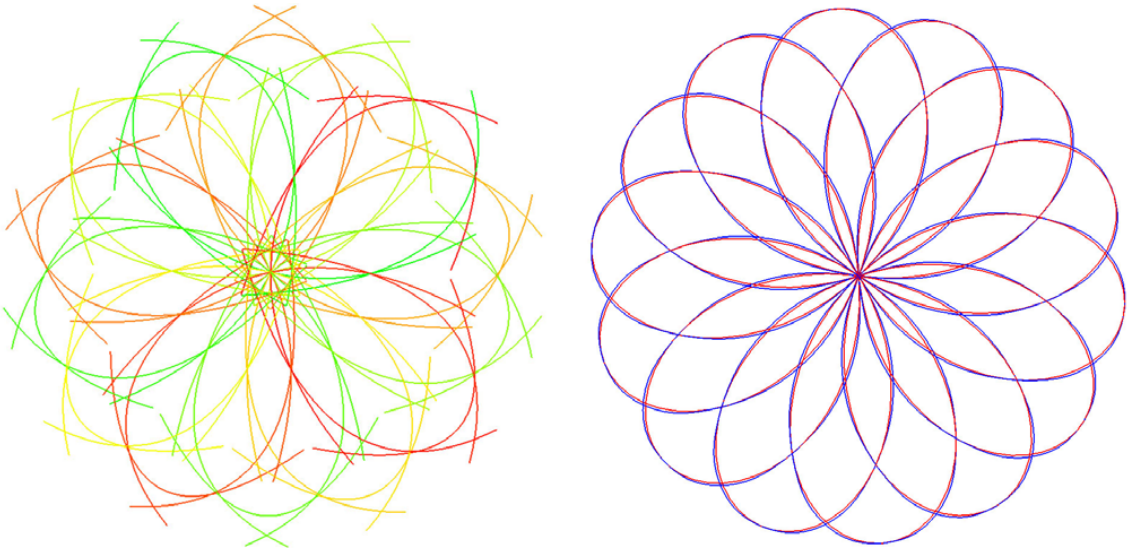


Figure 8.10: *To the right, two curves partly covering each other. The red curve is the original "Rose-curve", the blue curve is the approximating blending spline curve. The approximation is made by 56 interpolation points, the position and the first and second derivatives in each point are used. To the left, the 56 local Bézier curves of the blending splines are plotted gradually from green to red. They are all 2nd degree Bézier curves, forming the original curve around each interpolating point. The local curves are in a way "symmetrical around the rose center".*

The result is quite good, but not as geometrically good as in the previous example. The speed however is quite equal to the original curve, and is thus much better than in the previous example. To the left in the figure, the local curves are plotted. They are all of degree 2, and "symmetrical", in the sense that on every petal there is a set of local curves that have the same form on each petal. One can also noticeably improve the result by changing the local curves in this example, using a similar argument as in the previous example.

The next curve example is a so called "Cardioid curve" described in [80], and defined by the formula,

$$g(t) = \begin{pmatrix} 2\cos t\,(1+\cos t) \\ 2\sin t\,(1+\cos t) \\ 0 \end{pmatrix}, \quad \text{for } t \in [0, 2\pi). \tag{8.17}$$

A plot of expression (8.17) will look like an apple, with a cusp at the top. To the right in Figure 8.11, the "Cardioid curve" is approximated by a blending spline using 7 interpolation points uniformly distributed on the parameter interval. In each of the interpolation points are the positions, the first and the second derivatives used in the Hermite interpolation process. No special effort is done to reshape the cusp. There is one interpolation point at the bottom, and three on each side, where the distance (in the plane) is getting shorter on the way up from the bottom intersection point. The third point on each side is relatively close to the cusp. To the left in Figure 8.11, all seven local curves are plotted in different colors. They are all 3^{rd}-degree Bézier curves, modeling the shape of the original

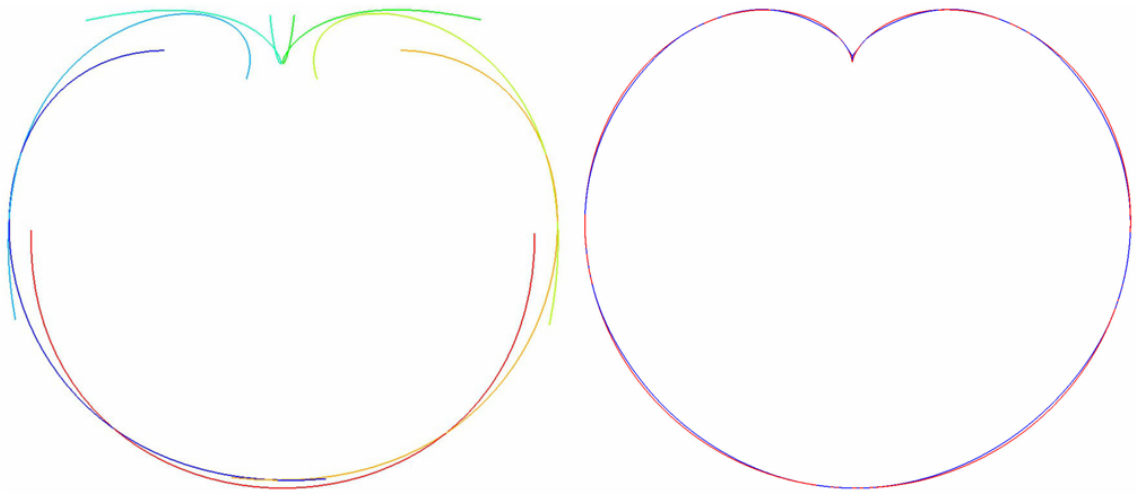


Figure 8.11: *To the right, two curves partly covering each other. The red curve is the original "Cardioid curve" . The blue curve is the approximating blending splines. The approximation is made by 7 interpolation points; the position and the three first derivatives in each point are used. To the left, the seven local Bézier curves of the blending splines are plotted in different colors. All seven curves are 3^{rd}-degree Bézier curves with 4 control points.*

curve quite well, but they seem to be too long because not only do they overlap half of the next curve, but they also overlap some of the following curve. The logical reason for this is, of course, the same as for the previous example, that the algorithm in this case assumes that the fourth derivatives are zero.

The last examples are some curiosities. There are three different approximations of circles and circular-arcs. To the left of Figure 8.12 there is an approximation of a circle using only one interpolating point, the position, first and second derivatives. The local curve is, therefore, only one 2nd degree Bézier curve. In the figure the original circle is red, the approximating curve is blue, and the local Bézier curve is dashed green.

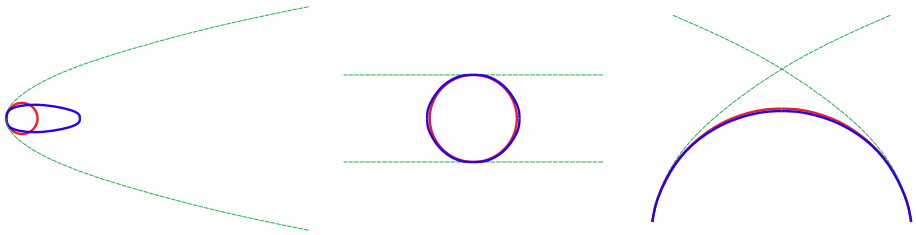


Figure 8.12: *Three examples of approximations of circles/circular arcs by blending spline using local Bézier curves. The original curves are red, the blending spline are blue, and the local Bézier curves are green. On the left side only one 2^{nd}-degrees Bézier-curves is used as local curve. In the middle, two local curves of 1^{st}-degree (lines) are used. On the right hand side there is a circular arc approximated by an blending spline using two 2^{nd}-degrees Bézier-curves as local curves.*

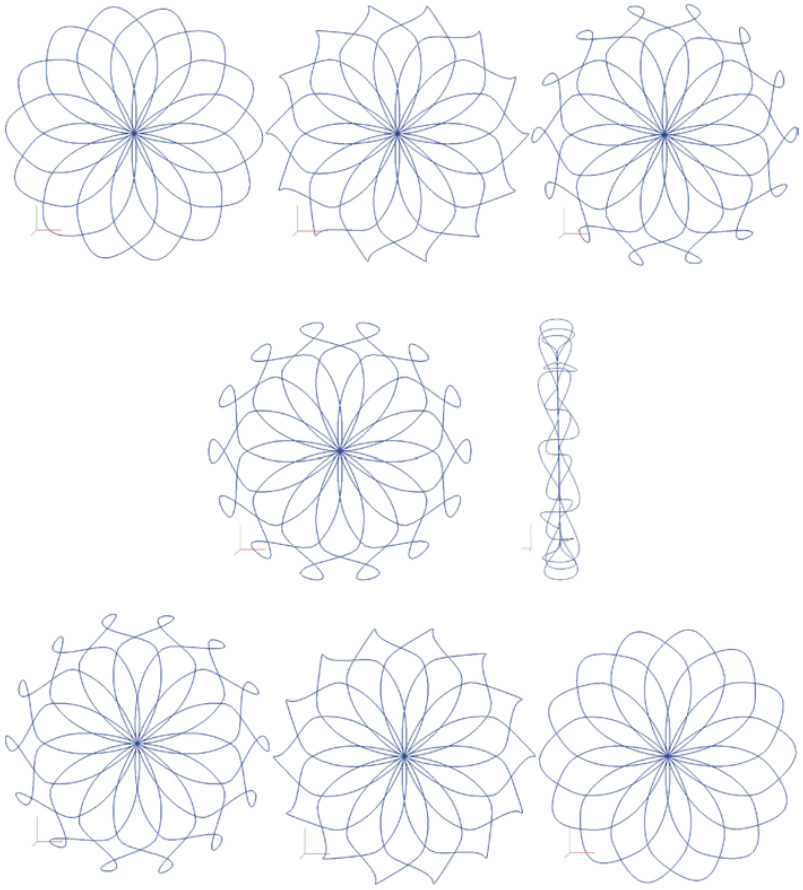


Figure 8.13: *The "Rose-curve" approximated by a blending spline using 56 interpolation points and local 2^{nd}-degree Bézier curves. The local curves at the tip of the petals are rotated (7 times around their local y-axis which coincide with the direction of the radius of the rose curve). In the figure, 7 "rotated" "Rose-curves" are plotted, each rotated $22.5°$ more than the previous one. The fourth curve, where the local curves at the tip of the petals are rotated a total of $90°$, is also plotted from the side.*

In the middle of Figure 8.12, a circle is approximated by only two interpolating points, the position and the first derivatives in each of the two points. The local lines are scaled by 0.92 after the interpolation. The result is visually quite good.

To the right of Figure 8.12, an arc, slightly smaller than a half circle, is approximated by a blending spline. Two local curves are created using an interpolation point with position, first and second derivatives, for each of them. In this case, another type of correction is showed, the domain interval is scaled by 0.635. The result is very accurate. One can see almost only the blue blending spline curve which is covering the red original circular arc.

In Figure 8.13, animation by rotating some of the local curves is shown. On a copied rose curve, all the local curves at the tip of each petal are continuously rotated 7 times about the radius of the rose curve. You will find more about all the examples shown here and also other examples, about approximation errors and how you can improve the result in [102], which can be loaded from http://urn.nb.no/URN:NBN:no-15022

8.3 The sub-curve construction

Extending any parametric curve to a blending spline curve

We can convert any curve to a spline curve by adding a knot vector. The result is that we get a set of overlapping sub-curves, each of which is linked to one of the internal knot values and with a domain that covers two knot intervals, one on each side of the corresponding knot. Thus, a sub-curve is only a restriction of the domain of a curve. To use sub-curves as local curves means that a blending spline copy of a curve initially is identical to the curve itself. This is because blending of a curve with itself gives the curve itself. When adding affine transformations (subsection 8.2.1), a sub-curve algorithm will basically be similar to the algorithm for local Bézier curves. From modifying (8.7), we get for $i = 1, 1, \ldots, n$

$$c(t) = \begin{pmatrix} 1 - B \circ w_{1,i}(t) & B \circ w_{1,i}(t) \end{pmatrix} \begin{pmatrix} A_{i-1}\ \varphi(t) \\ A_i\ \varphi(t) \end{pmatrix}, \quad t \in [t_i, t_{i+1}),$$

where $\varphi(t)$ is the curve from which we get the sub-curves, and A_i is a homogeneous matrix described in subsection 8.2.1 and expression (8.12). The formula can be further simplified to,

$$c(t) = A_{i-1}\ \varphi(t) + B \circ w_{1,i}(t)(A_i - A_{i-1})\varphi(t)$$

which gives us the following method.

Making any curve editable by converting to blending splines

Given a curve $\varphi(t)$. To converted $\varphi(t)$ to a blending spline curve we do the following
- determine the number n of editing cubes we want to use,
- create a knot vector adapted to this number and to a 2^{nd}-order B-spline, ie $\{t_i\}_{i=0}^{n+1}$,
- to each internal knot we assign a homogeneous matrix, initially identity matrices.

Then we have the following formula for the knot interval $[t_i, t_{i+1})$

$$c(t) = (A_{i-1} + B \circ w_{1,i}(t)\ \Delta A_i)\ \varphi(t) \tag{8.18}$$

where A_i is the homogeneous matrix connected to knot t_{i+1}, $\Delta A_i = A_i - A_{i-1}$, B is a B-function and $w_{1,i}(t)$ is defined in (6.11). Note that $\varphi(t)$ is a point with the last homogeneous coordinate equal to 1. For the derivatives we will have the same structure as in (8.9) and thus (8.11), ie we use $B_i^{(k)}(t) = \delta_{1,i}^k B^{(k)} \circ w_{1,i}(t)$, defined in (8.10). Thus the first 3 derivatives are

$$\begin{aligned}
c'(t) &= A_{i-1}\ \varphi'(t) + \Delta A_i \left(\underline{B_i'(t)\ \varphi(t) + B_i(t)\ \varphi'(t)} \right), \\
c''(t) &= A_{i-1}\ \varphi''(t) + \Delta A_i \left(\underline{B_i''(t)\ \varphi(t) + 2B_i'(t)\ \varphi'(t) + B_i(t)\ \varphi''(t)} \right), \\
c'''(t) &= A_{i-1}\ \varphi'''(t) + \Delta A_i \left(\underline{B_i'''(t)\ \varphi(t) + 3B_i''(t)\ \varphi'(t) + 3B_i'(t)\ \varphi''(t) + B_i(t)\ \varphi'''(t)} \right).
\end{aligned}$$

Note that in the first part, A_{i-1} is multiplied by a vector where the last homogeneous coordinate is 0, and in the second part, ΔA_i is multiplied by a point (underlined) where the last homogeneous coordinate is 1.

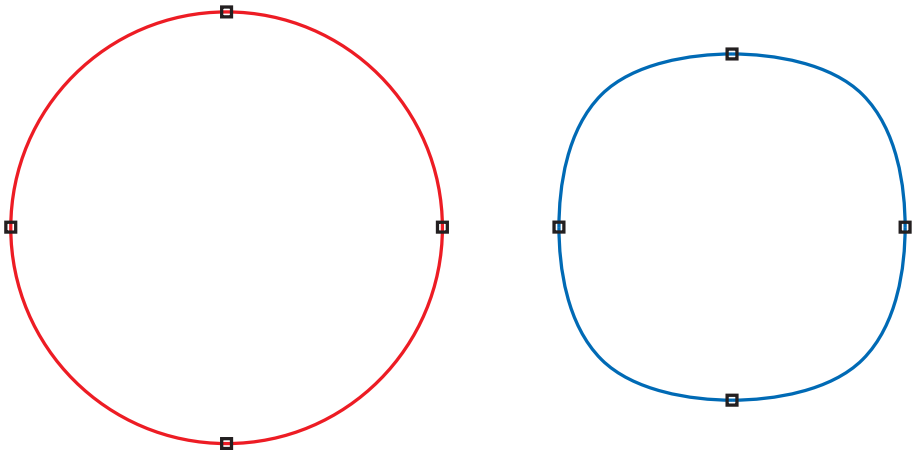


Figure 8.14: *A blending spline curve made from a circle, formula (4.1). On the left side we see it initially, a perfect circle where the four editing points are marked, on right side the curve is changed by moving the four editing points towards the center.*

In subsection 8.2.1 is the homogeneous matrices A_i introduced. The matrix represent a frame, a local coordinate system located at a point p_i with the coordinate axes x_i, y_i and z_i in $\mathbb{R}^3$. Thus the matrix A_i, ie the point and the axis can be visualized as a cube. This can be seen in Figure 8.6, Figure 8.7 and in Figure 8.14.

We will look at a simple example. A circle is given by the formula (4.1). The domain is $[0, 2\pi)$. The only changes we can make are to change the radius, otherwise the curve is static. The first decision is to use 4 editing points. The curve is closed, and from subsection 6.2.2 we see that the domain is $[t_1, t_5)$ and the knot vector $\tau = \{-\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi\}$. The four matrices $\{A_i\}_{i=0}^{3}$ are initially identity matrices.

Figure 8.14 shows this example. The curve is initially a perfect circle, as we can see on the left in the figure. To the right of the figure we see the curve after all four editing points have been moved towards the center of the circle. The editing points can now be moved in different directions, rotated and scaled. The curve will totaly change shape. but it will always be closed and thus topologically similar to a circle.

One problem in this example, and also in the recipe on the previous page, is that the local coordinate systems for the editing points are the global coordinate system. To make it easier to change/edit the curve we should follow the advise from subsection 8.2.1 on page 156, we must adjust the local curves, and insert the corresponding rotation and translation into the matrices A_i, $i = 0, 1, 2, 3$.

Therefore, at each editing point we add one extra matrix, $\mathfrak{T}_i$. Now, for each editing points,

- compute $A_i = F(t_{i+1})$, ie the Frenet frame at every point $\varphi(t_{i+1})$, $i = 0, 1, \ldots, n-1$, see about Frenet frame on page 156 and section 4.1.1,

- then we compute $\mathfrak{T}_i = A_i^{-1}$, $i = 0, 1, \ldots, n-1$.

The consequence is that the evaluation of the original curve now depend on the knot

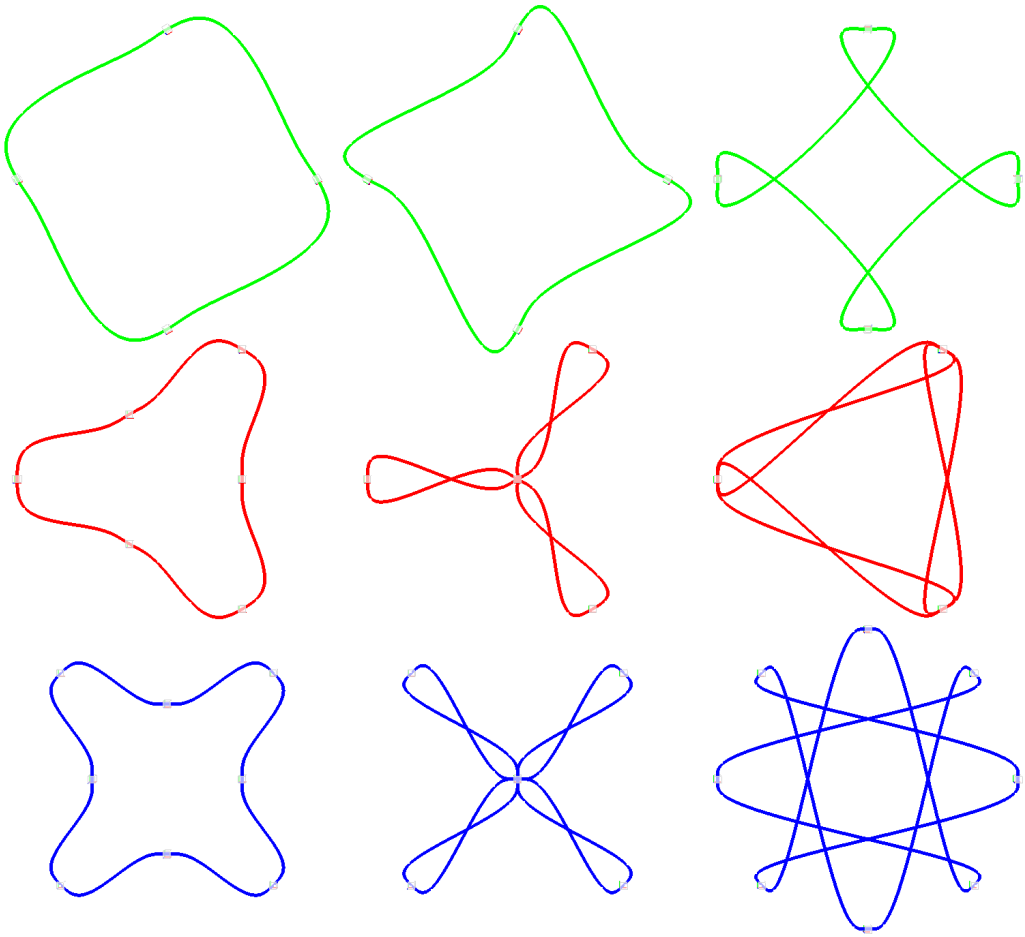


Figure 8.15: *Blending spline curve made from a circle, formula (4.1). The green curves has 4 knot intervals. At each knot, all the sub-curves are rotated,* 30°, 60° *and* 180°, *respectively, in the curves seen from the left side. In the red curves with 6 knot-intervals and the blue ones with 8 knot-intervals, every other sub-curve is moved towards the center of the circle, respectively* $\frac{1}{2}$, *1, 2 times the distance to the center.*

interval, ie expression (8.18) now becomes

$$c(t) = (A_{i-1} + B \circ w_{1,i}(t)\ \Delta A_i)\,(\mathfrak{T}_i\ \varphi(t))$$

and a corresponding change for the derivatives. When editing is completed, A_i can be updated by multiplying A_i by $\mathfrak{T}_i$, and then remove $\mathfrak{T}_i$.

Figure 8.15 shows more examples of the sub-curve construction. The green curves show the use of rotation and the red and blue curves show the use of translation. Note that only the matrices A_i are changed, for rotation the rotation in the 3×3 sub-matrix is at the top left, see page 16. For translation, only the column vector on the far right is changed, as described in (8.12).

Figure 8.16 and 8.17 shows how we can interactively shape figures using rotation and translation at the knots. It can be compared to pencil sketching, and it is done in seconds.

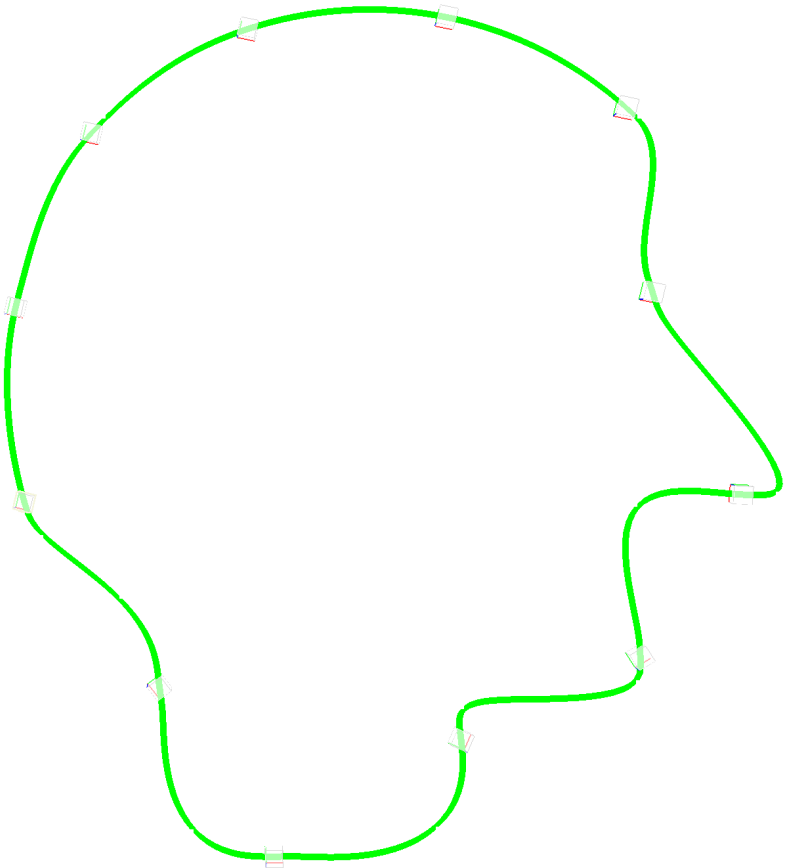


Figure 8.16: *A silhouette of a face modeled from a circle using 12 knots.*

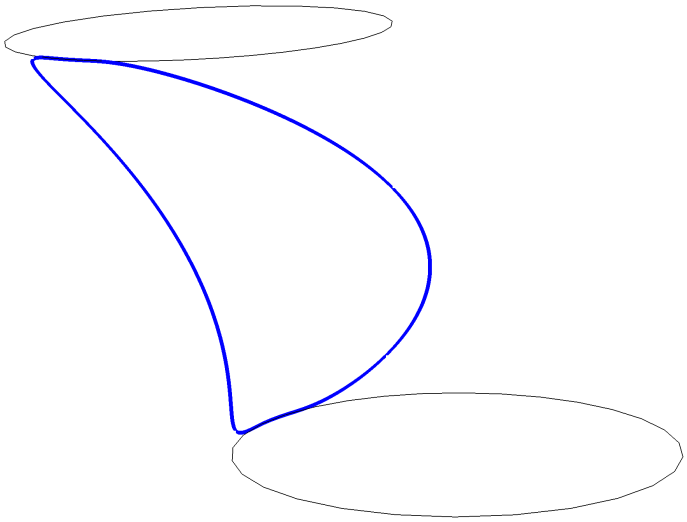


Figure 8.17: *Only 2 knots are used, both are rotated and one is translated. Two circles are plotted to illustrate the blending.*

Part II

Surfaces

Chapter 9

Parametric Surfaces

To give an idea of what a parametric surface is, imagine $\mathbb{R}^2$ as an infinite plane. To make a surface, we select a part of this plane, which we then call a parameter plane. Then we insert it into the intended space, which is usually the Euclidian space $\mathbb{R}^3$, deform it by either stretching or pressing it and bending it in several ways. Finally, we put it in the desired position and orientation. Note that to cut and glue is *not* an option here. Figure 9.2 give an example of this concept. A surface is a 2-dimensional object. If a surface is neither degenerated nor self intersecting we also call it a 2-dimensional manifold.

This was an attempt to illustrate the concept of parametric Surfaces. Usually is the output to be embedded in $\mathbb{R}^3$ but it can be in another Euclidian space or manifold with dimension > 0. A more formal definition is:

Definition 9.1. *A parameterized differentiable surface is a differentiable map $S : U \subset \mathbb{R}^2 \longrightarrow \mathbb{M}^n$, $n > 0$ of an open set $U \subset \mathbb{R}^2$ into an Euclidian space or manifold. Note that a half open or closed set is just a restriction of an open set.*

One often imagines a surface as the outer boundary of a three-dimensional object in space. Usually a surface is closed and topologically similar to a sphere or torus, or a torus-like object with several holes. It is not always as easy to parameterize such surfaces. Only the torus can be easily parameterized, the other types must be covered by several parameterizations, where the transition between them must be consistent (homeomorphic).

For the rest of the chapter we will assume that surfaces are embedded in $\mathbb{R}^3$. A map is usually notated by a letter, eg. $S : U \subset \mathbb{R}^2 \to \mathbb{R}^3$ where U is called the domain / the parameter plane, where the coordinates usually are notated by the letters (u, v). A surface can have a boundary as in the monkey saddle example or partly boundary as in a cylinder or no boundary as in a torus. The first example is a monkey saddle defined as following,

$$S(u,v) = \begin{pmatrix} u \\ v \\ uv^2 \end{pmatrix}, \quad u \in [-1,1], \quad v \in [-1,1]. \tag{9.1}$$

Here the domain $U = [-1,1] \times [-1,1]$ and the parameter plane is reflecting the xy-plane in $\mathbb{R}^3$ ($x = u$ and $y = v$). This means that the surface is stretched in the mapping process

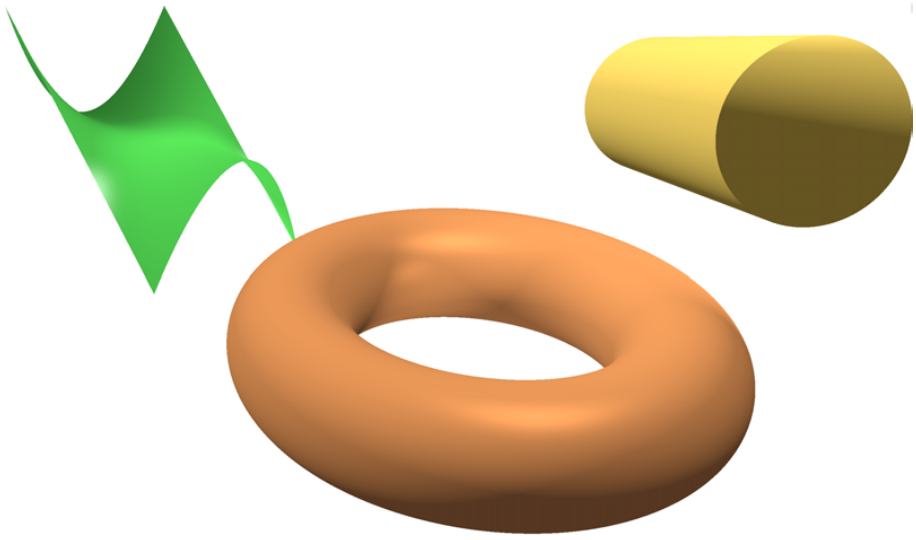


Figure 9.1: *Three example surfaces, a so called monkey saddle from expression (9.1), a cylinder from expression (9.2) and a torus from expression (9.3).*

and that the area must be bigger than the parameter plane, $2^2 = 4$. A plot of the surface is shown in green in figure 9.1.

The next example is a cylinder,

$$S(u,v) = \begin{pmatrix} \cos v \\ \sin v \\ u \end{pmatrix}, \quad u \in [0,6], \quad v \in [0,2\pi). \tag{9.2}$$

Here the domain $U = [0,6] \times [0,2\pi)$. This mapping is more like taking a piece of paper and roll it. The surface is not stretched in the mapping process and the area is the same as in the parameter plane, 10π. A plot of the surface is shown in yellow in figure 9.1.

Example three is a torus,

$$S(u,v) = \begin{pmatrix} (\cos u + 3)\cos v \\ (\cos u + 3)\sin v \\ \sin u \end{pmatrix}, \quad u \in [0,2\pi), \quad v \in [0,2\pi). \tag{9.3}$$

Here the domain is $U = [0,2\pi) \times [0,2\pi)$. A plot of the surface is shown, copper-colored, in figure 9.1.

Figure 9.2 is an attempt to provide a picture of what a parametric surface is. The figure shows the parameter plane at bottom left. It is called U and its coordinates are named u and v. The map is called S and is as a dotted curved arrow. The surface, upper right in the figure, is thus called as $S(U)$. In the figure, S maps a point p in the parameter plane to a point q on the surface. Summing up:

- $S(U)$ is the entire surface embedded in R^3.
- $p = (u,v)$ is a point in the parameter plane $U \subset R^2$, (u,v) are the coordinates.
- $q = S(p) = S(u,v)$ is a point on the surface in $\mathbb{R}^3$.
- S maps points in the parameter plane to points on the surface (in R^3).

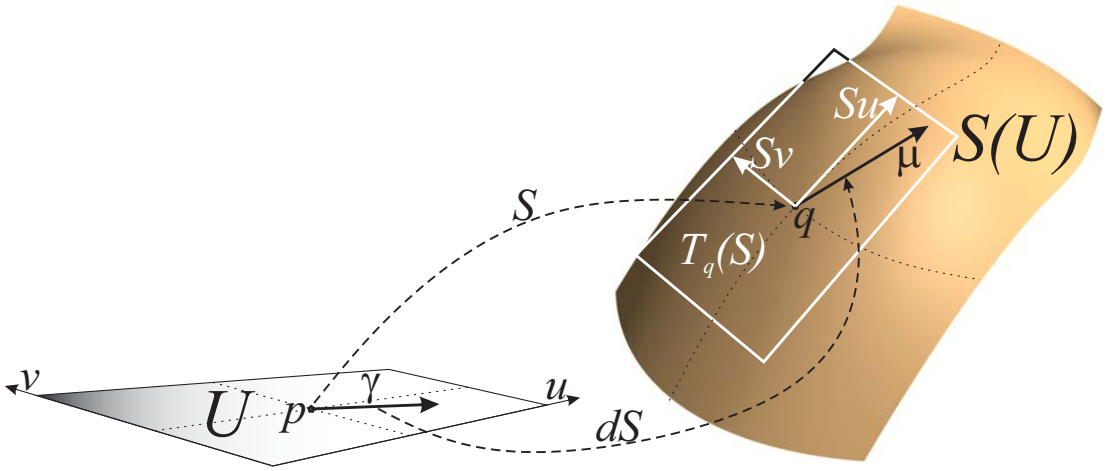


Figure 9.2: *On lover left hand side is the parameter plane $U \subset \mathbb{R}^2$ shown. On upper right hand side is there the surface, $S(U)$, embedded in $\mathbb{R}^3$. A point p in the parameter plane is mapped by S to a point $q = S(p)$ on the surface. A vector γ from the point p in the parameter plane is mapped by diferential map dS to a vector $\mu = dS_p(\eta)$ laying in the tangent plane $T_q(S)$ at the point q on the surface.*

9.1 Differentiation

The next step, which is also illustrated in figure 9.2, is differentiation. Recall that S maps a point p from the parameter plane to a point q on the surface. We will now find the direction we have to move in if we are in q and want to move in the direction where only one of the two parameter values changes. We call these directional vectors the partial derivatives. A surface has two parameter values and thus two partial derivatives. There are several notations used for partial derivatives. In the following we will see 3 different notations that are commonly used.

- The partial derivative in the u-direction, denoted in three different ways $D_u S = S_u = \frac{\partial S}{\partial u}$, is a 3-dimensional vector if $S(U) \subset \mathbb{R}^3$ and a tangent vector to the surface at the given position $q = S(p)$.
- The partial derivative in the v direction, denoted in three different ways $D_v S = S_v = \frac{\partial S}{\partial v}$, is a 3-dimensional vector if $S(U) \subset \mathbb{R}^3$ and a tangent vector to the surface at the given position $q = S(p)$.

The two tangent vectors S_u and S_v span a tangent plane at the point $q = S(p)$. The tangent plane, denoted $T_q(S)$, is also shown in figure 9.2 with a white border. Remember that the dimension of the tangent vectors, the partial derivatives, is equal the dimension of the space for which the surface is embedded. As an example of partial derivatives we can look at the monkey saddle in expression (9.1). We then get the following partial derivatives (using all three notations),

$$D_u S = S_u = \frac{\partial S}{\partial u} = \begin{pmatrix} 1 \\ 0 \\ v^2 \end{pmatrix}, \qquad D_v S = S_v = \frac{\partial S}{\partial v} = \begin{pmatrix} 0 \\ 1 \\ 2uv \end{pmatrix}, \tag{9.4}$$

9.1.1 The differential dS_p

The next map is the differential dS, a matrix where the two partial derivatives are the columns,

$$dS = [S_u \; S_v], \quad \in \mathbb{M}(s,2),$$

where s is the dimension of the space the surface is embedded in. This means that dS has two columns and the number of rows is equal the dimension of the space the surface is embedded into (i.e. 3 in $\mathbb{R}^3$). It maps a vector at the point p in the parameter plane, to a vector in the tangent plane $T_q(S)$ at the point $q = S(p)$.

In figure 9.2, there is an example where a vector γ is mapped to a vector μ by dS_p. i.e. $dS_p(\gamma) = \mu$. We call μ the directional derivative in direction γ. If the surface is embedded in $\mathbb{R}^3$ then dS_p is a 2×3 matrix, i.e.

$$dS_p : \mathbb{R}^2 \to \mathbb{R}^3, \quad p \in U \subset \mathbb{R}^2. \tag{9.5}$$

It also follows that d is a differential operator making a map to a differential map, we therefor get

$$d(S_u) = [S_{uu} \; , \; S_{uv}],$$
$$d(S_v) = [S_{vu} \; , \; S_{vv}],$$

and so on.

9.1.2 Curves on surfaces

We will describe the construction of how curves defined in the parameter plane of a surface is mapped into $\mathbb{R}^3$.

Given a curve $h : I \subset \mathbb{R} \to \mathbb{R}^2$,

$$h(t) = \sum_{i=1}^{n} c_i b_i(t).$$

where $c_i, \;\; i = 1,...,n$ are points or vectors in $\mathbb{R}^2$ and $b_i(t)$ are basis functions spanning a finite dimensional function space.

The question is, what curve will we get when the curve in the parameter space is mapped into 3D-space. Given a surface $S : U \subset \mathbb{R}^2 \to \mathbb{R}^3$. The curve definition then is,

$$c(t) = S(h(t)) = S \circ h(t), \quad t \in I \subset \mathbb{R}. \tag{9.6}$$

Using the chain rule on (9.6) we get the first, second and third derivative,

$$\begin{aligned} c'(t) &= dS_h(h'), \\ c''(t) &= d(dS_h(h'))_h(h') + dS_h(h''), \\ c'''(t) &= d(d(dS_h(h'))_h(h'))_h(h') + d(dS_h(h'))_h(h'') + d(dS_h(h''))_h(h') + dS_h(h'''), \end{aligned} \tag{9.7}$$

where the differentials matrices, there are used in the derivatives above are (to simplify we skip the position index in the rest of this subsection),

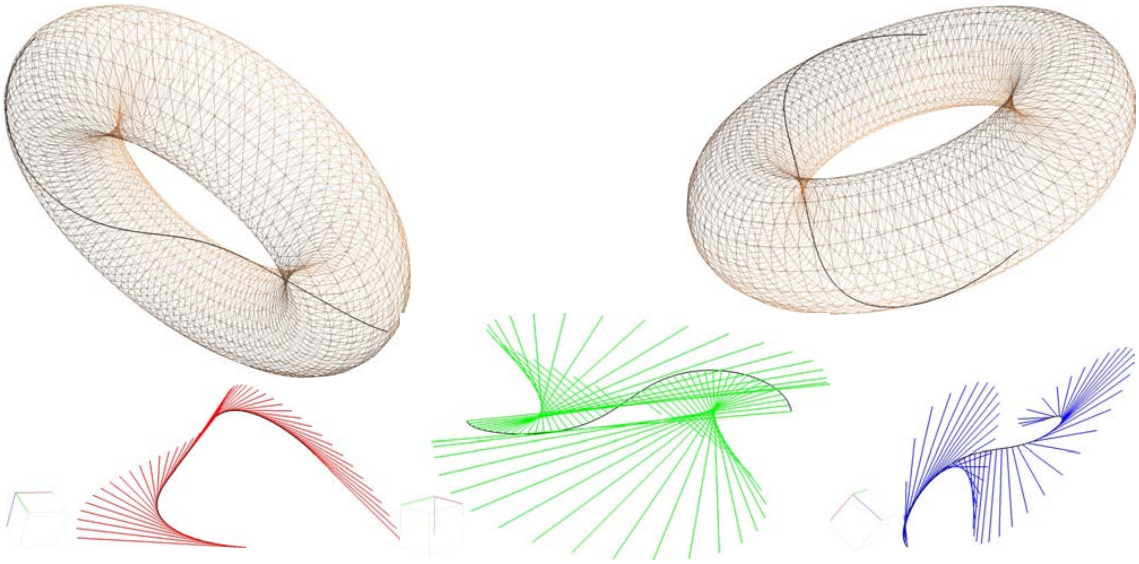


Figure 9.3: *On top, two views of a curve on a torus are displayed along with the torus itself. On the lower part there are three different views of the same curve. But now the first derivative is also shown in red, the second derivative is shown in green and the third derivative is shown in blue.*

$$\begin{aligned} d(dS(h')) &= d\left([S_u\ ,\ S_v]\ h'\right) \\ &= \left[\left([S_u\ ,\ S_v]\ h'\right)_u\ ,\ \left([S_u\ ,\ S_v]\ h'\right)_v\right] \\ &= \left[[S_{uu}\ ,\ S_{vu}]\ h'\ ,\ [S_{uv}\ ,\ S_{vv}]\ h'\right], \end{aligned}$$

and

$$\begin{aligned} d(d(dS(h'))(h')) &= d\left(\left[[S_{uu}\ ,\ S_{vu}]h'\ ,\ [S_{uv}\ ,\ S_{vv}]h'\right]\ h'\right) \\ &= \left[\left(\left[[S_{uu}\ ,\ S_{vu}]h'\ ,\ [S_{uv}\ ,\ S_{vv}]h'\right]\ h'\right)_u\ ,\ \left(\left[[S_{uu}\ ,\ S_{vu}]h'\ ,\ [S_{uv}\ ,\ S_{vv}]h'\right]\ h'\right)_v\right] \\ &= \left[\left[[S_{uuu}\ ,\ S_{vuu}]h'\ ,\ [S_{uvu}\ ,\ S_{vvu}]h'\right]\ h'\ ,\ \left[[S_{uuv}\ ,\ S_{vuv}]h'\ ,\ [S_{uvv}\ S_{vvv}]h'\right]\ h'\right]. \end{aligned}$$

In Figure 9.3 is a first degree Bézier curve, a straight line, expression (4.29), in the parameter plane of a torus, expression (9.3), mapped into $\mathbb{R}^3$ by expression (9.6). On top in Figure 9.3 is two different views of the curve together with the torus shown, on the lower part is there three different views of the curve and its first derivative (red), second derivative (blue) and third derivative (green) given in expression (9.7).

Another example of "curves" on a surface is a vector valued function describing the directional derivatives along a curve, not in the curve direction, but perpendicular to the curve direction. In this case we need two vector valued functions in the parameter plane, one vector valued function describing the position in the parameter plane depending on the parameter, another vector valued function describing the vector in $\mathbb{R}^2$ to find the directional derivatives on the surface. Vi therefore must have the two vector valued functions,

$$h(t) = \sum_{i=1}^{n_1} c_i b_i(t), \quad \text{and} \quad r(t) = \sum_{i=1}^{n_2} d_i b_i(t), \tag{9.8}$$

both for $t \in I \subset \mathbb{R}$. The map from parameter space of the surface to 3D-space is then

$$\tilde{c}(t) = dS_{h(t)}\left(r(t)\right). \tag{9.9}$$

The derivatives are,

$$\begin{aligned} \tilde{c}'(t) &= d(dS(r))(h') + dS(r'), \\ \tilde{c}''(t) &= d(d(dS(r))(h'))(h') + d(dS(h'))(r') + d(dS(r'))(h') + dS(r''), \end{aligned} \tag{9.10}$$

where the differentials matrices are,

$$\begin{aligned} d(dS(r)) &= d\left([S_u\ ,\ S_v]\ r\right) \\ &= \left[\left([S_u\ ,\ S_v]\ r\right)_u\ ,\ \left([S_u\ ,\ S_v]\ r\right)_v\right] \\ &= \left[[S_{uu}\ ,\ S_{vu}]\ r\ ,\ [S_{uv}\ ,\ S_{vv}]\ r\right], \end{aligned}$$

and

$$\begin{aligned} d(d(dS(r))(h')) &= d\left(\left[[S_{uu}\ ,\ S_{vu}]\, r\ ,\ [S_{uv}\ ,\ S_{vv}]\, r\right]\ h'\right) \\ &= \left[\left(\left[[S_{uu}\ ,\ S_{vu}]\, r\ ,\ [S_{uv}\ ,\ S_{vv}]\, r\right]\ h'\right)_u\ ,\ \left(\left[[S_{uu}\ ,\ S_{vu}]\, r\ ,\ [S_{uv}\ ,\ S_{vv}]\, r\right]\ h'\right)_v\right] \\ &= \left[\left[[S_{uuu}\ ,\ S_{vuu}]\, r'\ ,\ [S_{uvu}\ ,\ S_{vvu}]\, r\right]\ h'\ ,\ \left[[S_{uuv}\ ,\ S_{vuv}]\, r\ ,\ [S_{uvv}\ S_{vvv}]\, r\right]\ h'\right]. \end{aligned}$$

An example of this type of vector valued function, where the expressions (9.8), (9.9) and (9.10) are used is given in section 9.3.

9.1.3 The tangent plane $T_q(S)$

Consider any curve in a surface passing a point $q = S(p)$ on the surface. The derivatives of these curves at the common point q is, from (9.7),

$$c' = dS_p(h') = [S_u\ ,\ S_v] \begin{pmatrix} h'_u \\ h'_v \end{pmatrix} = h'_u\, S_u + h'_v\, S_v.$$

We can see that the first derivative, the tangent vector, of every curve on the surface passing q can be described as a linear combination of the two vectors S_u and S_v. This shows that they all lays in a plane spanned by S_u and S_v. This plane is therefore a tangent plane of surface $S(U)$ at the point q, and we denote it

$$T_q(S) = dS_q(\mathbb{R}^2).$$

It is obvious that the tangent plane still is the same even if we change parametrization. It is an intrinsic property independent of the parametrization.

In $\mathbb{R}^3$ we denote the unit vector orthogonal to the tangent plane to be the normal of the surface. The notation and orientation is in the following way. We first define the normal with a small letter,

$$n_q = S_u \wedge S_v(p), \tag{9.11}$$

where $\wedge$ denote the 3D vector product. We then define the unit normal to be denoted

$$N_q = \frac{n_q}{|n_q|}. \tag{9.12}$$

The unit normal is also obvious independent of the parametrization and thus an intrinsic property.

9.1.4 First fundamental form

Here we shall study geometric structures carried by the surface and that is connected to the tangent plane. The first fundamental form is the inner product on the tangent plane of a surface embedded in a 3D Euclidean space, ie it is the inner product of tangent vectors. It permits the calculation of curvature and metric properties of a surface such as length and area in a manner consistent with the ambient space. The first fundamental form is denoted by the Roman numeral I,

$$I_q(r,s) = \langle r,s \rangle.$$

Let $S(p),\ p=(u,v) \in \mathbb{R}^2$ be a parametrization of a surface. Then the inner product of two tangent vectors $r = dS_p(\hat{r})$ and $s = dS_p(\hat{s})$, $\hat{r} = (a,b)$ and $\hat{s} = (c,d)$, is

$$\begin{aligned} I_q(r,s) &= \langle dS(\hat{r}), dS(\hat{s}) \rangle_{S(p)} \\ &= \langle aS_u + bS_v, cS_u + dS_v \rangle \\ &= ac\langle S_u, S_u \rangle + (ad+bc)\langle S_u, S_v \rangle + bd\langle S_v, S_v \rangle \\ &= ac\,E + (ad+bc)\,F + bd\,G, \end{aligned}$$

where E, F and G are the coefficients of the first fundamental form, and gives us -

The first fundamental form

The coefficients of the first fundamental form are

$$\begin{aligned} E &= \langle S_u, S_u \rangle \\ F &= \langle S_u, S_v \rangle \\ G &= \langle S_v, S_v \rangle \end{aligned} \tag{9.13}$$

The first fundamental form, represented as a symmetric matrix.

$$I_q(r,s) = \hat{r}^T \begin{pmatrix} E & F \\ F & G \end{pmatrix} \hat{s}.$$

where $r,\ s \in T_q(S)$, the tangent plane of S in the point $q = S(p)$, and $\hat{r},\ \hat{s} \in \mathbb{R}^2$, connected to the point p in the parameter plane.
If the vectors r and s are the same vector the first fundamental form becomes

$$I_q(\mu) = \hat{\mu}^T \begin{pmatrix} E & F \\ F & G \end{pmatrix} \hat{\mu}.$$

where $\mu = dS_p(\hat{\mu}),\ \hat{\mu} \in \mathbb{R}^2$.

The first fundamental form completely describes the metric properties of a surface. Thus, it enables one to calculate the lengths of curves on the surface and the areas of regions on the surface. In the following, the line element and the area element will be developed and shown, and afterwards an example of the use of each of them will be shown .

The line element ds may be expressed in terms of the coefficients of the first fundamental form as

$$ds^2 = I_q\,(d\hat{s})$$

where $d\hat{s} = \begin{pmatrix} du \\ dv \end{pmatrix}$, the elements in $\mathbb{R}^2$. Computing this gives

$$ds^2 = (du\ ,\ dv)\begin{pmatrix} E & F \\ F & G \end{pmatrix}\begin{pmatrix} du \\ dv \end{pmatrix} = E\ du^2 + 2F\ dudv + G\ dv^2. \tag{9.14}$$

The classical area element given by $dA = |S_u \wedge S_v|$ can be expressed in terms of the first fundamental form with the assistance of Lagrange's identity,

$$dA = |S_u \wedge S_v|\ dudv = \sqrt{EG - F^2}\ dudv. \tag{9.15}$$

✓ First example is computation of length of curve on surfaces, using (9.14),

$$l(c) = \int_a^b \frac{ds}{dt}\,dt = \int_a^b \sqrt{E\ u'^2 + 2F\ u'v' + G\ v'^2}\,dt$$

Using the curve on a torus, equation (9.3), shown in Figure 9.3, gives,

$$S_u = \begin{pmatrix} -\sin u\cos v \\ -\sin u\sin v \\ \cos u \end{pmatrix} \quad \text{and} \quad S_v = \begin{pmatrix} -(\cos u + 3)\sin v \\ (\cos u + 3)\cos v \\ 0 \end{pmatrix},$$

and than,

$$E = 1, \quad F = 0, \quad G = (\cos u + 3)^2$$

The curve in Figure 9.3 defined in the parameter plane of the torus is

$$h(t) = \begin{pmatrix} 2.3 \\ 0.3 \end{pmatrix} + \begin{pmatrix} 3.7 \\ 5.7 \end{pmatrix} t, \quad \text{and the derivative} \quad h'(t) = \begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} 3.7 \\ 5.7 \end{pmatrix},$$

which gives

$$l(c) = \int_0^1 \sqrt{13.69 + 32.49(\cos(2.3 + 3.7t) + 3)^2}\,dt \approx 16,$$

where the approximated solution is computed using composite Simpson's rule.

✓ The second example is computing the area of the torus defined in (9.3) using (9.15)

$$\begin{aligned} A(S) &= \int_U dA \\ &= \int_0^{2\pi}\int_0^{2\pi} (\cos u + 3)\ dudv \\ &= 2\pi \int_0^{2\pi} (\cos u + 3)\ du \\ &= 12\pi^2 \end{aligned}$$

9.1.5 Second fundamental form

The second fundamental form (or shape tensor) is a quadratic form on the tangent plane of a smooth surface in the three dimensional Euclidean space, usually denoted by II (read "two"). Together with the first fundamental form, it serves to define extrinsic invariants of the surface, its principal curvatures.

Recall the normal N_q. There is a close relation between a unit sphere and a unit normal. There is a map

$$N : S \to S^2,$$

where S^2 is the unit 2D sphere. This is called the Gauss map of S. It follows that the differential of the Gauss map N is

$$dN_q : T_q(S) \to T_{N_q}(S^2).$$

But since $T_q(S)$ and $T_{N_q}(S^2)$ are parallel planes, it follows that

$$dN_q : T_q(S) \to T_q(S).$$

The linear operator dN_q is clearly related to changes of directional derivatives because the normal and all tangents are orthogonal. It follows that $|dN_q(r)|$ is the curvature in direction of a unit vector r in the tangent plane.

Second fundamental form is a quadratic form on the tangent plane. It is also an inner product. Given a vector $\mu = dS_p(\hat{\mu}) \in T_p(S)$, where $\widehat{\mu} = (a,b)$. The second fundamental form is

$$\begin{aligned} II_p(\mu) &= -\langle dN_q(\mu), \mu \rangle \\ &= -\langle N_u a + N_v b, S_u a + S_v b \rangle \\ &= -a^2 \langle N_u, S_u \rangle - ab\left(\langle N_u, S_v \rangle - \langle N_v, S_u \rangle\right) - b^2 \langle N_v, S_v \rangle \\ &= a^2\ e + 2ab\ f + b^2\ g, \end{aligned}$$

where e, f and g are the coefficients of the second fundamental form. From the definition of the normal it follows that $\langle N, S_u \rangle = \langle N, S_v \rangle = 0$. The derivatives both with respect to u and v are therefore also zero. It then follows that $\langle N, S_u \rangle_u = \langle N, S_{uu} \rangle + \langle N_u, S_u \rangle = 0$, and so on. This gives us -

Second fundamental form

The coefficients of the second fundamental form are

$$\begin{aligned} e &= \langle N, S_{uu} \rangle = -\langle N_u, S_u \rangle, \\ f &= \langle N, S_{uv} \rangle = -\langle N_u, S_v \rangle = -\langle N_v, S_u \rangle, \\ g &= \langle N, S_{vv} \rangle = -\langle N_v, S_v \rangle. \end{aligned} \tag{9.16}$$

The second fundamental form, represented as a symmetric matrix is

$$II_p(\mu) = \mu^T \begin{pmatrix} e & f \\ f & g \end{pmatrix} \mu.$$

where $\mu \in T_q(S)$, i.e is in the the tangent plane of S in the point $q = S(p)$.

The following definition is related to curvature:

- The Gaussian curvature $K = det(dN_q)$ of S, ie $K = \frac{det(II)}{det(I)}$.
- The maximum normal curvature k_1 and the minimum normal curvature k_2 are called the principal curvatures of S at the point q. It follows that $dN_q(e_1) = k_1 e_1$ and $dN_q(e_2) = k_2 e_2$. e_1 and e_2 are orthogonal to each other, they are called principal directions and are eigenvectors to dN_q. k_1 and k_2 are thus eigenvalues of dN_q.
- The mean curvature of S at the point q is $H = \frac{k_1+k_2}{2}$.
- It follows that the Gaussian curvature $K = k_1 k_2$.

By computations (see [49], page 154-156) the following formulas follows:

$$
\begin{aligned}
K &= \frac{eg - f^2}{EG - F^2} \\
H &= \frac{1}{2}\frac{eG - 2fF + gE}{EG - F^2} \\
k &= H \pm \sqrt{H^2 - K} \\
dN &= \frac{-1}{EG-F^2}\begin{pmatrix} e & f \\ f & g \end{pmatrix}\begin{pmatrix} G & -F \\ -F & E \end{pmatrix} = \frac{-1}{EG-F^2}\begin{pmatrix} fF - eG & eF - fE \\ gF - fG & fF - gE \end{pmatrix}.
\end{aligned}
$$

9.2 Surface of revolution

This includes most of the classical surfaces as cone, cylinder, sphere and torus. The basic construction is:

- We start with a curve $c(u)$ in $\mathbb{R}^2$,

$$
c(u) = \begin{pmatrix} c_x(u) \\ c_y(u) \end{pmatrix} \quad \begin{cases} u \in [s,e], & \text{if open,} \\ u \in [s,e), & \text{if closed.} \end{cases} \tag{9.17}
$$

- Then we decide the axis that the curve should rotate around. If we choose the z-axis we get

$$
S(u,v) = \begin{pmatrix} c_x(u)\cos v \\ c_x(u)\sin v \\ c_y(u) \end{pmatrix} \tag{9.18}
$$

- Finally we choose the domain. In u-direction we must use an interval in the domain of the curve $c(u)$, but in v-direction it has to be $[a,b]$ if $0 < b-a < 2\pi$ (not complete rotation) or $[a,b)$ if $b-a = 2\pi$ (complete rotation).

We have already seen two surfaces of revolution, a cylinder, equation (9.2), and a torus, equation (9.3. The curves (9.17) for the cylinder $\hat{c}(u)$ and the torus $\bar{c}(u)$ are

$$
\hat{c}(u) = \begin{pmatrix} 1 \\ u \end{pmatrix} \quad u \in [0,5] \quad \text{and} \quad \bar{c}(u) = \begin{pmatrix} \cos u \\ \sin u \end{pmatrix} + \begin{pmatrix} 3 \\ 0 \end{pmatrix} \quad u \in [0, 2\pi).
$$

Both fits their respective formulas (9.2) and (9.3) when replacing c_x and c_y in (9.18).

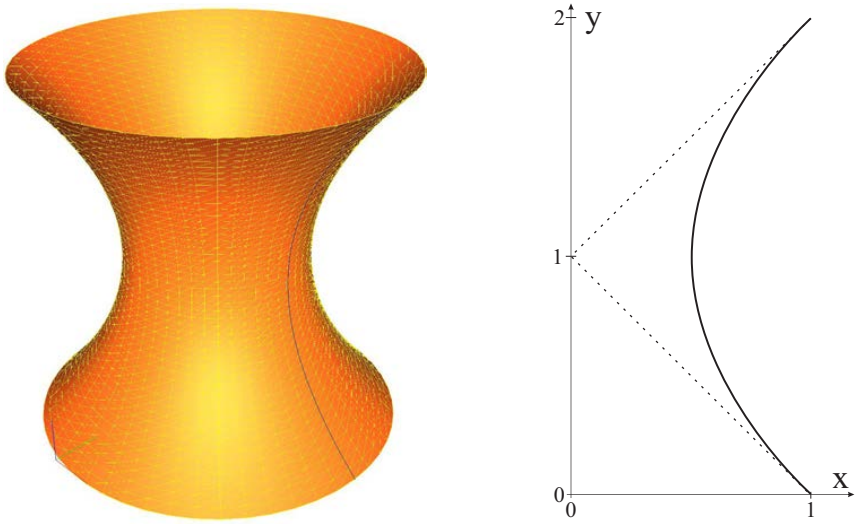


Figure 9.4: *An example of a surface of revolution, The curve on right hand side is rotated* 360° *around it's y-axis to get the surface shown on left hand side.*

Another example is shown in figure 9.4. In the figure is the curve $c(u)$ shown on right hand side. The curve is a second degree Bézier curve and we can see the control polygon and thus the coefficients (control points) in the figure. It follows that the curve is

$$c(u) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} (1-u)^2 + \begin{pmatrix} 0 \\ 1 \end{pmatrix} 2u(1-u) + \begin{pmatrix} 1 \\ 2 \end{pmatrix} u^2, \quad u \in [0,1],$$

and the surface

$$S(u,v) = \begin{pmatrix} (1-2u+2u^2)\cos v \\ (1-2u+2u^2)\sin v \\ 2u \end{pmatrix} \quad u \in [0,1], \quad v \in [0,2\pi).$$

9.3 Surface by sweeping

Imagine two curves, and that one of the curves, g - called a cross section or profile curve, is sweeping along the other curve, h - called a spine curve, ie

$$s(u,v) = h(v) + A(v)\left(g(u) - h(v_0)\right), \tag{9.19}$$

where $A(v)$ is an orthonormal 3×3 matrix,

$$A(v) = \begin{bmatrix} t(v) & f_2(v) & f_3(v) \end{bmatrix}, \tag{9.20}$$

where $t(v)$ is the unit tangent vector to the spine curve h,

$$t(v) = \frac{h'(v)}{|h'(v)|}, \tag{9.21}$$

and $f_2(v)$ and $f_3(v)$ are unit vectors normal to each other and to $t(v)$.

The matrix $A(v)$ is a rotation matrix describing how to rotate the profile curve when we move along the spine curve. We want the profile curve to keep the orientation according to the spine curve. We therefor need the first column vector to be the tangent vector of the spine curve. There are however several choices of rotation around the tangent vector, and we shall look at two different choices.

Frenet frame (also called TNB frame, see subsection 4.1.1) is one choice. Here we denote the column of the matrix by

$$A(v) = [T \quad N \quad B]. \tag{9.22}$$

where

$$T = t(v) = \frac{h'}{|h'|} \quad \text{and} \quad B = T \wedge N = \frac{h' \wedge h''}{|h' \wedge h''|} \quad \text{and} \quad N = \frac{T'}{|T'|} = B \wedge T.$$

The partial derivatives of a surface by sweeping are

$$S_u = A(v)g'(u) \quad \text{and} \quad S_v = h'(v) + A'(v)(g(u) - h(v_0)). \tag{9.23}$$

To compute $A'(v)$ using Frenet frame, is using the Frenet-Serret formulas,

$$\frac{d}{dv}[T \quad N \quad B] = |h'| \ [T \quad N \quad B] \begin{bmatrix} 0 & -\kappa & 0 \\ \kappa & 0 & -\tau \\ 0 & \tau & 0 \end{bmatrix} = |h'| \ [\kappa N \quad \tau B - \kappa T \quad -\tau N].$$

where κ is the curvature of the spine curve h, and τ is the torsion.

The formulas for computing the curvature and the torsion are

$$\kappa = \frac{|h' \wedge h''|}{|h'|^3}, \quad \tau = \frac{\langle (h' \wedge h''), h''' \rangle}{|h' \wedge h''|^2}.$$

Proofs of all formulas connected to Frenet frames can be found from page 130 in [80].

There is a problem connected to the use of Frenet frame. If h'' is zero or parallel to h' at a point on the spine curve h, then we do not have an orhonormal matrix at that point and we also might get a jump in the rotation over the point. The surface might even look strange if h'' is close to zero or to be parallel to h'.

It is not easy to avoid singularities, but it is possible to make the rotation smoother. Another choice is therefor using rotation minimizing frames, RMF, developed and discussed in [97], [83], [119] and [158].

In relation to the Frenet frame (9.22), the RMF (9.20) can be represented as

$$\widetilde{A}(v) = A(v) \ R(v). \tag{9.24}$$

where $A(v)$ is defined in(9.22) and the orthonormal matrix

$$R(v) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\omega & -\sin\omega \\ 0 & \sin\omega & \cos\omega \end{bmatrix}. \tag{9.25}$$

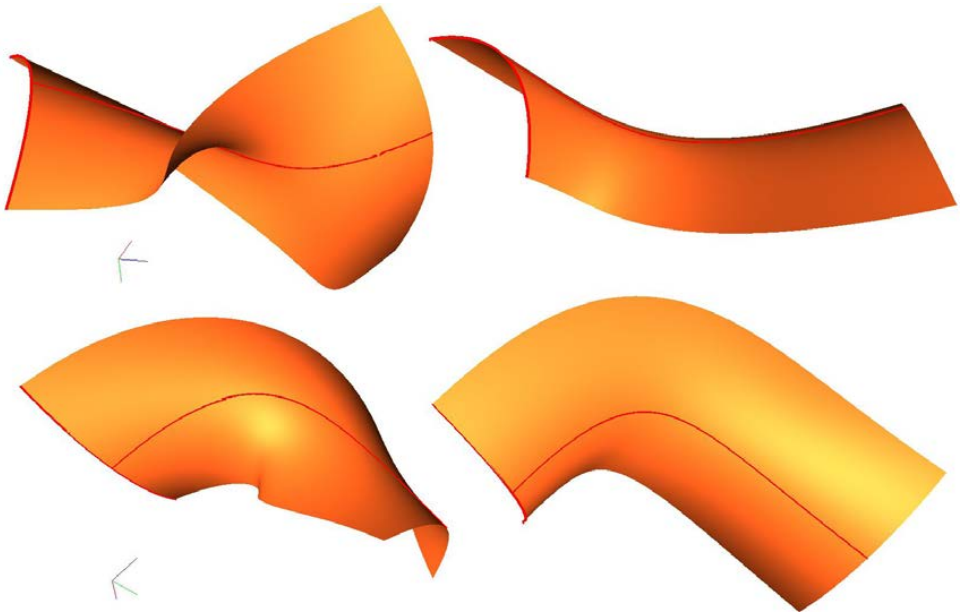


Figure 9.5: *On left hand side is there two different views of a surface by sweeping using TNB-frame. On right hand side is there two views of a surface by sweeping using RMF-frame. The same profile and spine curves are used for both surfaces.*

with a rotation angle $\omega = \omega(v)$ specifying the rotation around the T-axis, i.e. the difference between TNB and RMF. The angle ω can be computed from the integral formula

$$\omega(v) = \omega_0 - \int_{v_0}^{v} \tau(t)|h'(t)|dt, \tag{9.26}$$

with an integration constant ω_0, giving an initial rotation.

To compute the derivative, $\widetilde{A}'(v)$, we get

$$\widetilde{A}'(v) = A'(v)R(v) + A(v)R'(v). \tag{9.27}$$

where

$$R'(v) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\omega' \sin\omega & -\omega' \cos\omega \\ 0 & \omega' \cos\omega & -\omega' \sin\omega \end{bmatrix}. \tag{9.28}$$

and

$$\omega'(v) = \tau(v)|h'(v)|. \tag{9.29}$$

In figure 9.5 is one profile curve, a curve along one edge of the surfaces, and one spine curve, a curve from one edge to the opposite edge in the middle of the surfaces, marked with red. Two different surfaces are made by the same profile and spine curve, one on left hand side and one on right hand side. The surface on left hand side is made using a TNB-frame (Frenet frame) along the spine curve, and the surface on right hand side is made by using a RMF frame (rotation minimizing frame) along the spine curve. In this case is $\omega_0 = 0$ chosen, see (9.26). Both surfaces are shown in two different views to give a better impression of them. The surface on right hand side is obvious much smoother than the other one.

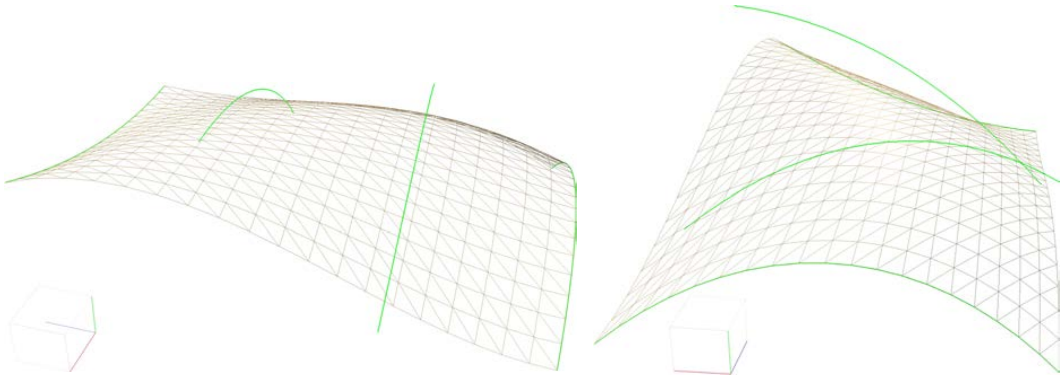


Figure 9.6: *Two views of a surfaces made by blending 4 2^{nd} degree Bézier curves plotted in green, and where the blending functions are the 3^{rd} degree Bernstein polynomials.*

9.4 Surfaces from blending curves

Recall the Curve constructions, Hermite, Bézier, B-splines etc. All these constructions are based on blending points or points and vectors by using different types of blending functions. It is possible to make a surface in the same way by just replacing the points (and vectors) by curves.

The following general construction can be used for surfaces, similar to the general curve construction in (4.11),

$$S(u,v) = \sum_{i=1}^{n} b_i(u)\, c_i(v). \tag{9.30}$$

Here all curves must have a common domain,

$$c_i(v) : I \subset \mathbb{R}^2 \to \mathbb{R}^3, \quad i = 1,2,...,n,$$

and the set of basis functions $b_i(u),\ \ i = 1,2,...,n$, must be linearly independent of each other and span an n-dimensional function space. An example of a surface made from blending curves is given in Figure 9.6. Note that there are few limits for which blending functions to be use, for example Lagrange, Bernstein, Hermite, B-splines. The choice will depend on which interpolation / approximation properties you want to use in the modeling.

The differentiation is straight forward, and the partial derivatives are

$$S_u = \sum_{i=1}^{n} b_i'(u)\, c_i(v), \quad S_v = \sum_{i=1}^{n} b_i(u)\, c_i'(v), \quad S_{uv} = \sum_{i=1}^{n} b_i'(u)\, c_i'(v).$$

Alternatively we can swap the parameters,

$$\bar{S}(u,v) = \sum_{i=1}^{n} b_i(v) c_i(u). \tag{9.31}$$

In the rest of the chapter will a bar over the map of a surface based on curve construction denote a swap of the parameters between the curves and the basis functions.

9.5 Tensor product surfaces

The name tensor product refers to a product of two vector spaces, in this case finite dimensional function spaces.

Recall from Section 4.2 how a function space is spanned by a set of basis functions. In Section 4.2 the initial example was $\mathscr{P}_d(I)$, the space of polynomial functions of degree d on the domain $I \subset \mathbb{R}$. In the following, the tensor product of two polynomial-based function rooms is the theme. From section 4.2 we know that $\mathscr{P}_d(I)$ for a given d and I can be expressed by different set of basis functions, ie

- Monomial basis functions $\{t^i\}_{i=0}^d$, where d is finite
- Hermite basis functions $\{H_i\}_{i=0}^d$, especially if $d = 3$ or $d = 5$, see Section 4.3
- Lagrange interpolation functions $\{L_{d,i}\}_{i=0}^d$, where d is finite, see Section 5.3
- Bernstein polynomials $\{b_{i,d}\}_{i=0}^d$, where d is finite, see Section 4.4.1

For the Hermite basis functions and set of Bernstein polynomials, the domain $I = [0,1]$.

A tensor product is actually an outer product between two vectors. As an example, given two vectors of monomial basis functions, one 2^{nd}-degree, $\mathbf{m}(u) = (1,u,u^2)$, and one 3^{rd}-degree, $\mathbf{m}(v) = (1,v,v^2,v^3)$ Than the tensor product will be a matrix of basis functions,

$$M(u,v) = \mathbf{m}(u)^T \ (\mathbf{m}(v) = \begin{pmatrix} 1 \\ u \\ u^2 \end{pmatrix} \begin{pmatrix} 1 & v & v^2 & v^3 \end{pmatrix} = \begin{bmatrix} 1 & v & v^2 & v^3 \\ u & uv & uv^2 & uv^3 \\ u^2 & u^2v & u^2v^2 & u^2v^3 \end{bmatrix}.$$

To each of these 12 basis functions is a coefficient awarded. In order to get a parametric surface, one can do it in the following way,

$$S(u,v) = \begin{pmatrix} 1 & u & u^2 \end{pmatrix} \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} \begin{pmatrix} 1 \\ v \\ v^2 \\ v^3 \end{pmatrix}$$

A surface will typically be embedded in $\mathbb{R}^3$, where the coefficients are elements of $\mathbb{R}^3$, ie points or vectors. A general formula for a tensor product surface based on polynomials is

$$S(u,v) = \sum_{i=0}^{d_u} \sum_{j=0}^{d_v} a_{i,j}\, b_j(v)\, b_i(u), \quad u \in I_u,\ v \in I_v, \tag{9.32}$$

where d_u is the polynomial degree and I_u is the parameter domain in u-direction, d_v is the polynomial degree and I_v is the parameter domain in v-direction. For implementation, the expression can be divided into two curve like expressions (see 9.30),

$$S(u,v) = \sum_{i=0}^{d_u} c_i(v) B_i(u), \quad \text{where} \quad c_i(v) = \sum_{j=0}^{d_v} a_{i,j} B_j(v), \quad \text{for} \quad i = 0,1,...,d_v. \tag{9.33}$$

This shows that this is just a "surface from blending curves" (from Section 9.4), and that all algorithms from curves can thus be used for tensor product surfaces.

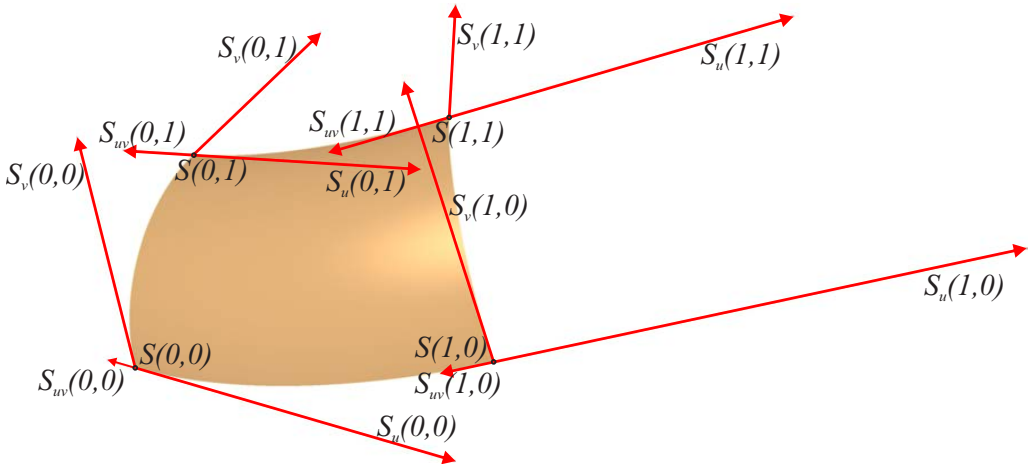


Figure 9.7: *A 3^{rd}-degree tensor product Hermite surface $S(u,v)$ is shown. At each corners is the position S and the partial derivatives S_u and S_v and the cross derivative S_{uv} plotted. These 4 points and 12 vectors are the content of the 4×4 - matrix in the formula.*

9.5.1 Tensor product Hermite surfaces

In Section 4.3, the basis functions for 3^{rd}-degree Hermite interpolation are shown in (4.17), ie

$$\mathbf{H}(t) = \begin{pmatrix} H_1(t) \\ H_2(t) \\ H_3(t) \\ H_4(t) \end{pmatrix} = \begin{pmatrix} 1-3t^2+2t^3 \\ 3t^2-2t^3 \\ t-2t^2+t^3 \\ -t^2+t^3 \end{pmatrix}.$$

It follows that according to (9.32), a tensor product Hermite surface will be as follows:

$$s(u,v) = \mathbf{H}(u)^T M\, \mathbf{H}(v) = \sum_{i=1}^{4}\sum_{j=1}^{4} M_{i,j}\, H_j(v)\, H_i(u), \quad u\in[0,1],\ v\in[0,1],$$

which expanded gives,

$$s(u,v) = (H_1(u)\ \ H_2(u)\ \ H_3(u)\ \ H_4(u)) \begin{bmatrix} s(0,0) & s(0,1) & s_v(0,0) & s_v(0,1) \\ s(1,0) & s(1,1) & s_v(1,0) & s_v(1,1) \\ s_u(0,0) & s_u(0,1) & s_{uv}(0,0) & s_{uv}(0,1) \\ s_u(1,0) & s_u(1,1) & s_{uv}(1,0) & s_{uv}(1,1) \end{bmatrix} \begin{pmatrix} H_1(v) \\ H_2(v) \\ H_3(v) \\ H_4(v) \end{pmatrix}.$$

The 4×4 matrix M is the data describing the shape of the surface. It can be divided into four 2×2 sub-matrices. The upper left sub-matrix consist of 4 points that describe the location of the 4 corners. The lower left sub-matrix consists of the partial derivatives in the u-direction in the 4 corners. The upper right sub-matrix consist of the partial derivatives in the v-direction in the 4 corners and the lower left sub-matrix consist of the cross-derivatives in in the 4 corners.

Figure 9.7 shows an example of a tensor product Hermite surface. It is a copy of a part of a torus, see (9.3). The parameter interval on the torus is $[[0.4,0.9]\times[0.4,1.6]]$ and the 4 partial derivatives s_u from the torus are scaled with 0.5, the 4 partial derivatives s_v are scaled with 1.2 and the 4 cross derivatives s_{uv} are scaled with $0.5*1.2=0.6$. All derivatives (the scaled ones) are visualized as red arrows in Figure 9.7.

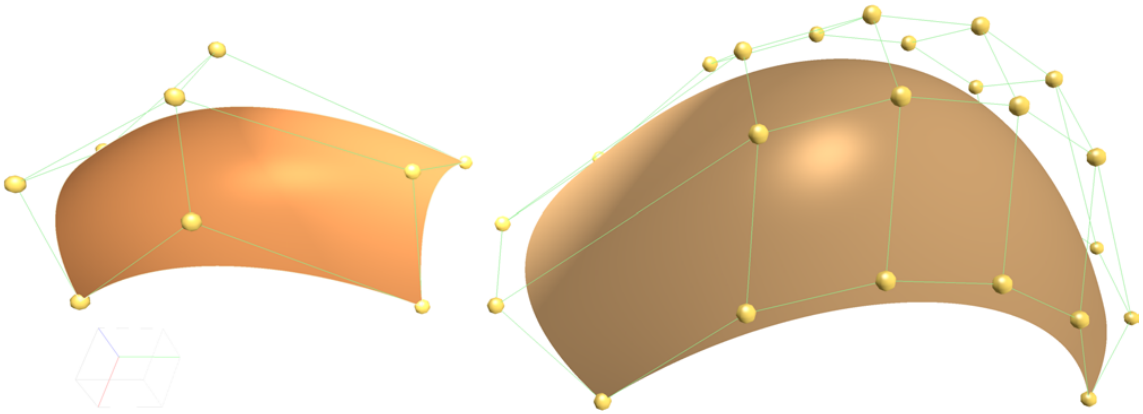


Figure 9.8: *We see two Bézier surfaces including their control points and control polygons. The surface on the left is of degree 2 in both directions, the surface on the right is of degree 5 in one direction and degree 3 in the other direction.*

9.5.2 Tensor product Bézier surfaces

Recall Bézier curves from Section 4.4. Given a degree d and a control polygon consisting of $d+1$ control points. If we then replace the control points with curves, ie Bézier curves, we get a tensor product Bézier surface,

$$s(u,v) = \sum_{i=0}^{d_u} \sum_{j=0}^{d_v} p_{i,j}\, b_{j,d_v}(v)\, b_{i,d_u}(u), \quad u,\, v \in [0,1], \tag{9.34}$$

The polynomial function spaces in the Bézier case are of degree d_u in the u-parameter direction and of degree d_v in the v-parameter direction. The two functions spaces are are spanned by the Bernstein polynomials $b_{i,d_u}(u),\ i = 0,1,\ldots,d_u$ and by the Bernstein polynomials $b_{j,d_v}(v),\ j = 0,1,\ldots,d_v$. The control points $p_{i,j},\ i = 0,1,\ldots,d_u,\ j = 0,1,\ldots,d_v$ define a control net that in a way outlines the surface.

If we use the 2^{nd} degree example shown on the left side in Figure 9.8, and look at the matrix formulation of (9.34), we get,

$$s(u,v) = \begin{pmatrix} b_{0,2}(u) & b_{1,2}(u) & b_{2,2}(u) \end{pmatrix} \begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} \\ C_{1,0} & C_{1,1} & C_{1,2} \\ C_{2,0} & C_{2,1} & C_{2,2} \end{bmatrix} \begin{pmatrix} b_{0,2}(v) \\ b_{1,2}(v) \\ b_{2,2}(v) \end{pmatrix}.$$

Recall the Bernstein/Hermite matrix from section 4.4.3, as well as Algorithm 2 to compute it. By using this matrix, the 2^{nd} degree example from Figure 9.8, and thus the expression above, can be expanded to

$$\begin{bmatrix} s & s_u & s_{uu} \\ s_v & s_{uv} & s_{uuv} \\ s_{vv} & s_{uvv} & s_{uuvv} \end{bmatrix} = \begin{bmatrix} (1-u)^2 & 2u(1-u) & u^2 \\ 2u-2 & 2-4u & 2u \\ 2 & -4 & 2 \end{bmatrix} \begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} \\ C_{1,0} & C_{1,1} & C_{1,2} \\ C_{2,0} & C_{2,1} & C_{2,2} \end{bmatrix} \begin{bmatrix} (1-v)^2 & 2v-2 & 2 \\ 2v(1-v) & 2-4v & -4 \\ v^2 & 2v & 2 \end{bmatrix}$$

In Figure 9.8 is there two examples of tensor product Bézier surfaces. To the left is a 2^{nd} degree Bézier surface including its 9 control points. To the right is a $5^{th} \times 3^{rd}$-degree Bézier surface including its 24 control points plotted.

Figure 9.9: *We see 4 tensor product B-spline surfaces including their control points and control polygons. The control points and polynomial degree are the same for all four. The only difference is that the upper left surface is open/clamped in both directions, the upper right is closed in one direction, the lower left is closed in the other direction and the lower right is closed in both directions.*

9.5.3 Tensor product B-spline surfaces

Recall B-spline curves from Chapter 6, Section 6.2. The definition includes a knot vector $\tau = \{t_0, t_1, \ldots, t_{n+d}\}$, a degree d, and a set of n control points $c_0, c_1, \ldots, c_{n-1}$. The general formula, (6.12), is similar to the formula of Bézier curves.

If we, as in the Bézier case, replace the control points, but now with B-spline curves defined over the same spline space (ie common knot vector and polynomial degree), then we get a tensor product B-spline surface,

$$s(u,v) = \sum_{i=0}^{n_u-1} \sum_{j=0}^{n_v-1} c_{i,j}\, b_{d_v,j}(v)\, b_{d_u,i}(u), \tag{9.35}$$

The parametric domain depends on the following (illustrated in Figure 9.9):

Open - u	$u \in [u_{d_u}, u_{n_u}]$	Open - v	$v \in [v_{d_v}, v_{n_v}]$	Figure 9.9 - upper left
Open - u	$u \in [u_{d_u}, u_{n_u}]$	Closed - v	$v \in [v_{d_v}, v_{n_v+d_v})$	Figure 9.9 - upper right
Closed - u	$u \in [u_{d_u}, u_{n_u+d_u})$	Open - v	$v \in [v_{d_v}, v_{n_v}]$	Figure 9.9 - lower left
Closed - u	$u \in [u_{d_u}, u_{n_u+d_u})$	Closed - v	$v \in [v_{d_v}, v_{n_v+d_v})$	Figure 9.9 - lower right

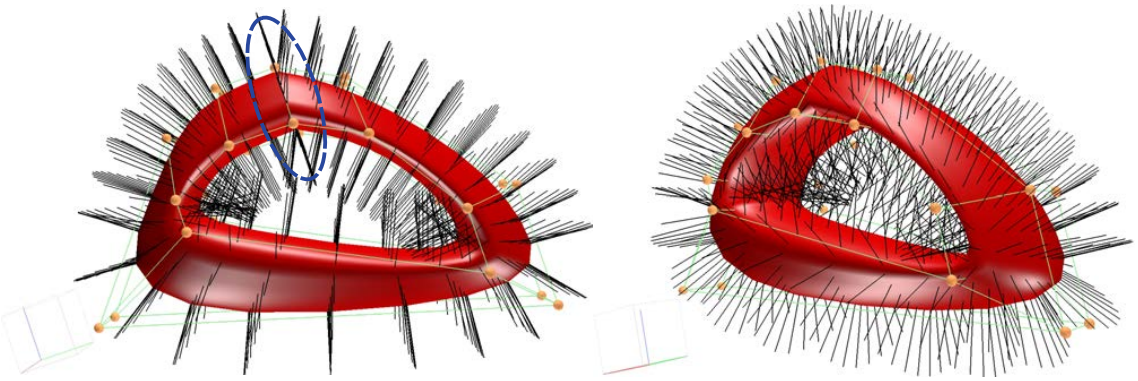


Figure 9.10: *We see two different views of the surface from Figure 9.9 that is closed in both directions. Here the number of sample points is smaller and the unit normals $N_{u,v}$ at each sample point are also shown. On the left side of the figure there is a dashed blue ellipse, which marks that there are two normals at the same point in all the points along a given constant v-parameter value, $v = 2$.*

The surfaces in Figure 9.9 are of 2^{nd}degree in both directions. For open surface, the knot vectors are $\mathbf{u} = \{0,0,0,1,2,2,2\}$, $\mathbf{v} = \{0,0,0,1,2,2,3,4,4,4\}$ and domain $[0,2] \times [0,4]$. For closed surfaces are $\mathbf{u} = \{-2,-1,0,1,2,3,4\}$, $\mathbf{v} = \{-2,-1,0,1,2,2,3,4,5,6\}$ and domain $[0,4) \times [0,6)$. The control points $c_{i,j},\ i = 0,1,2,3,\ j = 0,1,\ldots,6$ define a control net sketching the surface better than a Bézier surface. In Figure 9.9, there are $4 \times 7 = 28$ control points, which can be seen as copper colored balls connected with green lines.

To implement closed B-spline surfaces, it is best to use a modified version of Algorithm 4 which we find in Section 6.2.9, where we reduce the number of derivatives as needed. Then, for practical reasons, it will be an advantage to make extended knot vectors, where we add d more knots at the end of the knot vectors, where the distance between them reflect the distance between the knots from knot t_d to knot t_{d+d}. In Figure 9.9 and 9.10, this means $\mathbf{u} = \{-2,-1,0,1,2,3,4,5,6\}$ and $\mathbf{v} = \{-2,-1,0,1,2,2,3,4,5,6,7,8\}$.

Note that the knot vector $\mathbf{v}$ from the example in Figure 9.9 has two equal knots in the interior, $v_4 = v_5 = 2$. Since the polynomial degree is 2 in the v-direction, the continuity is reduced by 1 over the parameter value $v = 2$, cf. Definition 6.3, property **P3** and point 5 in the list of nice properties in Section 6.2.2. A 2^{nd}-degree B-spline are C^1-smooth over a single knot-value. So over 2 equal knots the surface will have a kink. This is what Figure 9.10 shows. The lower right surface from Figure 9.9 (closed in both direction), is shown in Figure 9.10 also, but now the unit normal at each sample points is also plotted. In the figure is there a dashed blue ellipse which marks that we can see two, not equal, normals at the same point for all sample points along the constant parameter value $v = 2$. It is also possible to see in the figures that there is a king on the surface at this curve, $s(u,2)$.

On matrix notation is the computation for tensor product B-spline surfaces as follows,

$$\bar{s}(u,v) = \mathbf{B}_{d,\mathbf{u}}(u,i)\ \mathbf{C}\ \mathbf{B}_{d,\mathbf{v}}(v,j)^T, \quad \text{when} \quad u_i \le u < u_{i+1} \quad \text{and} \quad v_i \le v < v_{i+1},$$

which is the B-spline version of the last expression in section 9.5.2. The B-spline-Hermite matrices $\mathbf{B}$ in both u- and v-direction can be computed using Algorithm 4 on page 97.

9.6 Boolean sum surface

Boolean sum surface is based on equating a surfaces to a Boolean sum set. If $A(F)$ and $B(F)$ are two set operators acting on F, then the Boolean sum $(A \oplus B)(F)$ is defined as the union of the two sets, which contains the intersection set, $A(B(F)) = B(A(F)) = AB(F)$, only once. Hence

$$(A \oplus B)(F) = A(F) + B(F) - AB(F).$$

Imagine that the boundary of a surface, represented by 4 connected curves, is know. To make a surface with this boundary, then Coons patch - bilinear blends can be used.

When modeling shapes, we often want smooth surfaces. If several surfaces are put together, we want, not only the surfaces to be connected, but that the derivatives across an edge are the same on two adjacent surfaces. Coons Patch - bicubic blends, is a solution.

Another possibilities is that a net of curves describing a surface is known. The curves in one direction intersect all curves in the other direction. A surface made by these curves is called a Gordon surface (see [79]).

The construction is the same for these three types of blending and can be adapted to any other type of boolean sum surfaces. The fundamental construction is:

- First a surface from blending curves is made. The curves are oriented in the first parameter direction. This surface is $S_1(u,v)$ and the set of basis functions is $\{b_i\}$.
- Another surface from blending curves is made. The curves are oriented in the second parameter direction. This surface is $\bar{S}_2(u,v)$ and the set of basis functions is $\{\bar{b}_i\}$.
- Finally a tensor product surface is made, where $\{\bar{b}_i\}$ and $\{b_i\}$ are the basis functions. The coefficients depend of the type of basis functions. That might be values and derivatives at corner points, or a point "net". This surface is $S_3(u,v)$.
- A Boolean sum surface is defined as,

$$S(u,v) = S_1(u,v) + \bar{S}_2(u,v) - S_3(u,v). \tag{9.36}$$

9.6.1 Coons patch, bilinear blending

Steven Anson Coons [1] introduced the Coons patch in 1964, [26]. The Coons patch approach is based on the premise that a patch can be described in terms of four distinct boundary curves. A simple approach is the bilinearly blended Coons patch. It is based on four boundary curves, i.e. two sets. The two sets are (see also Figure 9.11)

$$\text{the u-set: } g_0(u) \text{ and } g_1(u),\ u \in [0,1], \quad \text{the v-set: } c_0(v) \text{ and } c_1(v),\ v \in [0,1].$$

The curves must be connected, and it is therefore an assumption that

$$c_0(0) = g_0(0),\ \ c_0(1) = g_1(0),\ \ c_1(0) = g_0(1),\ \ c_1(1) = g_1(1).$$

[1] Steven Anson Coons (1900 – 1979) was professor at the Massachusetts Institute of Technology in mechanical engineering. During world war II, he worked with aircraft surfaces, developing the mathematics for generalized "surface patches". Later(1960is) he published works on what today is known as Coons patch.

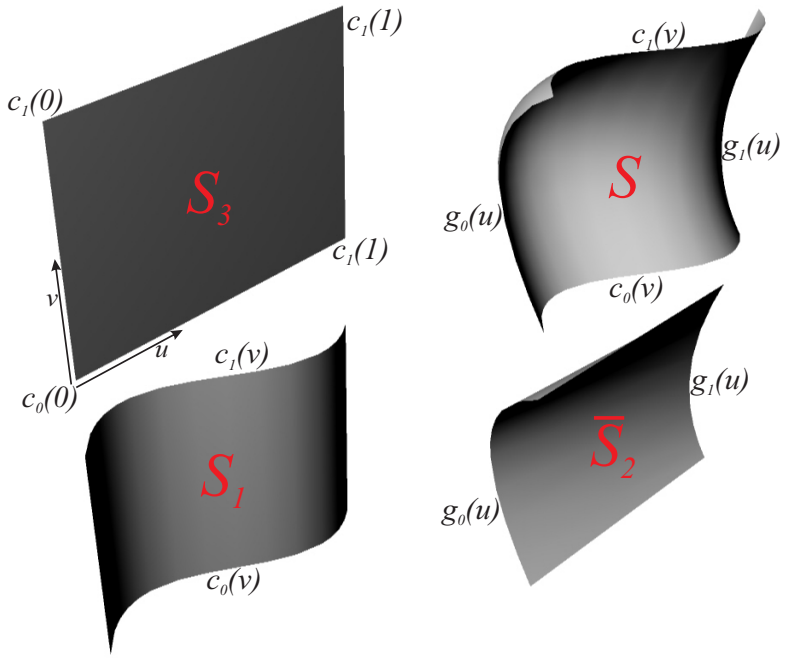


Figure 9.11: *Example of Coons patch. The surface S is a Boolean sum of the three surfaces S_1, $\bar{S}_2$ and S_3, i.e. $S = S_1 + \bar{S}_2 - S_3$.*

We first makes three surfaces by linear interpolation in u direction, S_1, in v direction, $\bar{S}_2$ and finally (tensor product) bilinear interpolation that interpolates the corner points, S_3, ie

$$
\begin{aligned}
S_1(u,v) &= \begin{pmatrix} c_0(v) & c_1(v) \end{pmatrix} \begin{pmatrix} 1-u \\ u \end{pmatrix} && \text{blending curves,} \\
\bar{S}_2(u,v) &= \begin{pmatrix} 1-v & v \end{pmatrix} \begin{pmatrix} g_0(u) \\ g_1(u) \end{pmatrix} && \text{turned blending curves,} \\
S_3(u,v) &= \begin{pmatrix} 1-v & v \end{pmatrix} \begin{pmatrix} c_0(0) & c_1(0) \\ c_0(1) & c_1(1) \end{pmatrix} \begin{pmatrix} 1-u \\ u \end{pmatrix} && \text{tensor product.}
\end{aligned}
\tag{9.37}
$$

To make the final surface (S) we just use the boolean sum of the three surfaces, i.e.

$$S(u,v) = S_1(u,v) + \bar{S}_2(u,v) - S_3(u,v),$$

where S_1, $\bar{S}_2$ and S_3 are defined in (9.37). The process is clearly illustrated in Figure 9.11. As a control, we calculate the edges of the resulting surface S,

$$
\begin{aligned}
S(u,0) &= S_1(u,0) + \bar{S}_2(u,0) + S_3(u,0) \\
&= (1-u)\, c_0(0) + u\, c_1(0) + g_0(u) - (1-u)\, c_0(0) - u\, c_1(0) = g_0(u), \\
S(u,1) &= S_1(u,1) + \bar{S}_2(u,1) + S_3(u,1) \\
&= (1-u)\, c_0(1) + u\, c_1(1) + g_1(u) - (1-u)\, c_0(1) - u\, c_1(1) = g_1(u), \\
S(0,v) &= S_1(0,v) + \bar{S}_2(0,v) + S_3(0,v) \\
&= c_0(v) + (1-v)\, g_0(0) + v\, g_1(0) - (1-v)\, c_0(0) - v\, c_0(1) = c_0(v), \\
S(1,v) &= S_1(1,v) + \bar{S}_2(1,v) + S_3(1,v) \\
&= c_1(v) + (1-v)\, g_0(1) + v\, g_1(1) - (1-v)\, c_1(0) - v\, c_1(1) = c_1(v),
\end{aligned}
$$

showing that the edges are as originally provided.

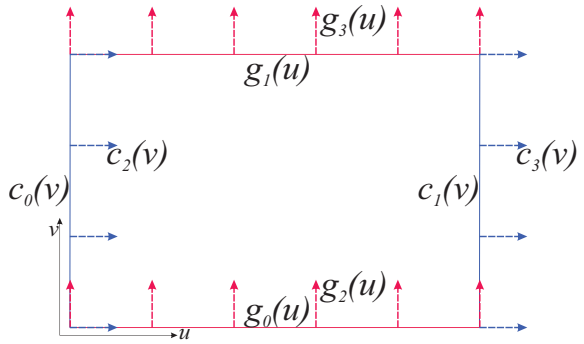


Figure 9.12: *Four boundary curves $c_0(v)$, $c_1(v)$, $g_0(u)$ and $g_1(u)$, and four vector valued functions $c_2(v)$, $c_3(v)$, $g_2(u)$ and $g_3(u)$, describing the derivatives orthogonal to the boundary curves.*

9.6.2 Coons patch, bicubic blending

It is also possible to construct Coons patch which give a higher degrees of control of the boundary, i.e. which also give the derivative in v direction on the boundary where u varies, and the derivative in u direction on the boundary where v varies. This is called a bicubically blended Coons patch, and is constructed in the same way as the bilinearly blended Coons patch. The difference is that we now have to use a Hermite interpolation rule instead of linear interpolation.

We need four boundary curves, i.e. two sets, notated as

$$\begin{aligned} c_0(v) \quad &\text{and} \quad c_1(v), \quad v \in [0,1], \\ g_0(u) \quad &\text{and} \quad g_1(u), \quad u \in [0,1]. \end{aligned}$$

Then it is two vector valued functions that gives the derivatives in u direction along the boundary curves in v direction, and the two vector valued functions that gives the derivatives in v direction along the boundary curves in u direction,

$$\begin{aligned} c_2(v) \quad &\text{and} \quad c_3(v), \quad v \in [0,1], \\ g_2(u) \quad &\text{and} \quad g_3(u), \quad u \in [0,1]. \end{aligned}$$

The curves must be connected and consistent with each other, therefore it is required that

$$\begin{aligned} c_0(0) &= g_0(0), & c_0(1) &= g_1(0), \\ c_1(0) &= g_0(1), & c_1(1) &= g_1(1). \\ c_0'(0) &= g_2(0), & c_0'(1) &= g_3(0), \\ c_1'(0) &= g_2(1), & c_1'(1) &= g_3(1), \\ g_0'(0) &= c_2(0), & g_0'(1) &= c_3(0), \\ g_1'(0) &= c_2(1), & g_1'(1) &= c_3(1). \end{aligned}$$

The connection between the curves and vector valued functions is shown in Figure 9.12. Before we start constructing Coons patch bicubic blending, we define the vectors

$$\begin{aligned} \mathbf{H}(t) &= [\ H_1(t), \quad H_2(t), \quad H_3(t), \quad H_4(t)\ ], \\ \mathbf{c}(t) &= [\ c_1(t), \quad c_2(t), \quad c_3(t), \quad c_4(t)\ ], \\ \mathbf{g}(t) &= [\ g_1(t), \quad g_2(t), \quad g_3(t), \quad g_4(t)\ ]. \end{aligned}$$

Two surfaces made by Hermite interpolation of curves and vector valued function, one in u direction (surface S_1), one in v direction (surface $\bar{S}_2$), and one tensor product Hermite surface using the position, the partial derivatives in u and v and the cross derivatives, in all four corners (surface S_3), must be made. Ie

$$\begin{aligned} S_1(u,v) &= \langle \mathbf{c}(v),\, \mathbf{H}(u) \rangle && \text{blending curves,} \\ \bar{S}_2(u,v) &= \langle \mathbf{H}(v),\, \mathbf{g}(u) \rangle && \text{turned blending curves,} \\ S_3(u,v) &= \mathbf{H}(v)\, \mathbf{M}\, \mathbf{H}(u)^T && \text{tensor product,} \end{aligned} \tag{9.38}$$

where the matrix $\mathbf{M}$ is

$$\mathbf{M} = \begin{pmatrix} c_0(0) & c_1(0) & c_2(0) & c_3(0) \\ c_0(1) & c_1(1) & c_2(1) & c_3(1) \\ c_0'(0) & c_1'(0) & a_{11} & a_{12} \\ c_0'(1) & c_1'(1) & a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} g_0(0) & g_0(1) & g_0'(0) & g_0'(1) \\ g_1(0) & g_1(1) & g_1'(0) & g_1'(1) \\ g_2(0) & g_2(1) & a_{11} & a_{12} \\ g_3(0) & g_3(1) & a_{21} & a_{22} \end{pmatrix}.$$

The four numbers on the lower right square are the cross derivatives to S_3 in the four corners. We will later show what these four values must be if the requirements at the edges shall be fulfilled. But the pattern also indicates what these values should be. To make the final surface S we use the boolean sum of (9.38), i.e.

$$S(u,v) = S_1(u,v) + \bar{S}_2(u,v) - S_3(u,v),$$

Recall the Hermite basis property described in (4.18),

$$\mathbf{H}(0) = [1,\, 0,\, 0,\, 0], \quad \mathbf{H}(1) = [0,\, 1,\, 0,\, 0], \quad \mathbf{H}'(0) = [0,\, 0,\, 1,\, 0], \quad \mathbf{H}'(1) = [0,\, 0,\, 0,\, 1].$$

Using this when calculating the boundaries of the surface gives us

$$\begin{aligned} S(u,0) &= S_1(u,0) + \bar{S}_2(u,0) - S_3(u,0) = \langle \mathbf{c}(0),\, \mathbf{H}(u) \rangle + g_0(u) - \langle \mathbf{c}(0),\, \mathbf{H}(u) \rangle = g_0(u), \\ S(u,1) &= S_1(u,1) + \bar{S}_2(u,1) - S_3(u,1) = \langle \mathbf{c}(1),\, \mathbf{H}(u) \rangle + g_1(u) - \langle \mathbf{c}(1),\, \mathbf{H}(u) \rangle = g_1(u), \\ S(0,v) &= S_1(0,v) + \bar{S}_2(0,v) - S_3(0,v) = c_0(v) + \langle \mathbf{H}(v),\, \mathbf{g}(0) \rangle - \langle \mathbf{H}(v),\, \mathbf{g}(0) \rangle = c_0(v), \\ S(1,v) &= S_1(1,v) + \bar{S}_2(1,v) - S_3(1,v) = c_1(v) + \langle \mathbf{H}(v),\, \mathbf{g}(1) \rangle - \langle \mathbf{H}(v),\, \mathbf{g}(1) \rangle = c_1(v), \end{aligned}$$

which shows that the edges are as expected. The derivative in the opposite direction on the four edges must be equal to the four given vector valued functions, thus

$$S_v(u,0) = \langle \mathbf{c}'(0),\, \mathbf{H}(u) \rangle + g_2(u) - \langle (g_2(0),\, g_2(1),\, a_{11},\, a_{12}),\, \mathbf{H}(u) \rangle = g_2(u),$$

which requires that $a_{11} = c_2'(0)$ and $a_{12} = c_3'(0)$. Further is

$$S_v(u,1) = \langle \mathbf{c}'(1),\, \mathbf{H}(u) \rangle + g_3(u) - \langle (g_3(0),\, g_3(1),\, a_{21},\, a_{22}),\, \mathbf{H}(u) \rangle = g_3(u),$$

which requires that $a_{21} = c_2'(1)$ and $a_{22} = c_3'(1)$. Further is

$$S_u(0,v) = c_2(v) + \langle \mathbf{H}(v),\, \mathbf{g}'(0) \rangle - \langle \mathbf{H}(v),\, (c_2(0),\, c_2(1),\, a_{11},\, a_{21}) \rangle = c_2(v),$$

which requires that $a_{11} = g_2'(0)$ and $a_{21} = g_3'(0)$. Further is

$$S_u(1,v) = c_3(v) + \langle \mathbf{H}(v),\, \mathbf{g}'(1) \rangle - \langle \mathbf{H}(v),\, (c_3(0),\, c_3(1),\, a_{12},\, a_{22}) \rangle = c_3(v),$$

which requires that $a_{12} = g_2'(1)$ and $a_{22} = g_3'(1)$.

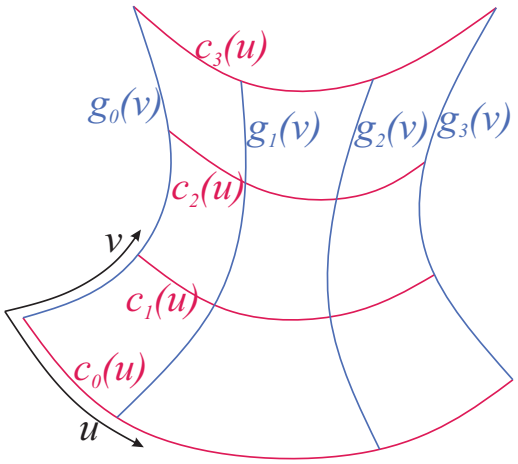


Figure 9.13: *A net of curves, the red in the u-direction and the blue in the v-direction. Every red curves intersect all blue curves and vice versa. A Gordon surface will interpolate all of these curves.*

Thus, if the final surface shall fulfill the requirements, that the four edges and the "orthogonal" derivatives to the four edges shall be equal the given boundary curves and vector valued functions, the matrix $\mathbf{M}$ must be,

$$\mathbf{M} = \begin{pmatrix} c_0(0) & c_1(0) & c_2(0) & c_3(0) \\ c_0(1) & c_1(1) & c_2(1) & c_3(1) \\ c_0'(0) & c_1'(0) & c_2'(0) & c_3'(0) \\ c_0'(1) & c_1'(1) & c_2'(1) & c_3'(1) \end{pmatrix} = \begin{pmatrix} g_0(0) & g_0(1) & g_0'(0) & g_0'(1) \\ g_1(0) & g_1(1) & g_1'(0) & g_1'(1) \\ g_2(0) & g_2(1) & g_2'(0) & g_2'(1) \\ g_3(0) & g_3(1) & g_3'(0) & g_3'(1) \end{pmatrix}. \quad (9.39)$$

9.6.3 Gordon surface

Gordon surfaces where developed in the late 1960s by W. Gordon, [79], who then worked for the General Motors Research labs. He called the method for "transfinite interpolation".

The method is basically the same as the method for Coons patch, ie a Boolean sum operation. The difference is that Hermite interpolation is replaced by ordinary interpolation, ie that the basis functions here will be Lagrange polynomials. The input data of a Gordon surface is a net of curves, one set of curves $c_i(u)$ in the u-parameter direction going in "the same" direction, and another set of curves $g_j(v)$ in the v-parameter direction, not necessarily orthogonal to the first set of curves, but in an "opposite" direction as can be seen in Figure 9.13. Every curve $c_i(u)$ in the first set of curves must intersect all the curves in the second set and vice versa, which also is clearly illustrated in Figure 9.13.

Recall the Langrange interpolation polynomials from Section 5.3, and formula (5.8), $L_{d,i}(t) = \prod_{j=0, j\neq i}^{d} \frac{(t-t_j)}{(t_i-t_j)}$, where $i = 0, 1, \ldots, d$ and d is the polynomial degree, and further we denote the set of Lagrange polynomials for $\mathbf{L}_d(t) = \begin{pmatrix} L_{d,0}(t) & L_{d,1}(t) & \ldots & L_{d,d}(t) \end{pmatrix}$.

The input to an algorithm must be a set of $d_v + 1$ curves $\{g_i(u)\}_{i=0}^{d_v}$ in the u-parameter direction. They must have a common parameter domain I_u, and there must be a set of

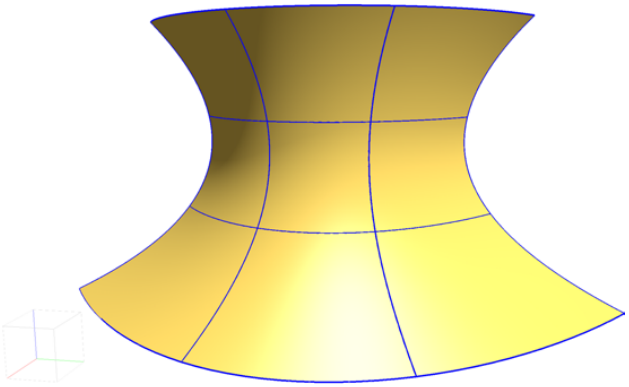


Figure 9.14: *We see an implementation of a Gordon surface that interpolates a net of* 4×4 *curves drawn in blue.*

$d_u + 1$ parameter values $u_i \in I_u$ where this set of curves intersect the other set. Likewise it must be a set of $d_u + 1$ curves $\{c_i(v)\}_{i=0}^{d_u}$ in the v-parameter direction. They must have a common parameter domain I_v, and there must be a set of $d_v + 1$ parameter values $v_i \in I_v$ where this second set of curves intersect the first set. Summing up:

✓ First set of curves $\mathbf{g}(u) = \{g_i(u)\}_{i=0}^{d_v}$,

✓ second set of curves $\mathbf{c}(v) = \{c_j(v)\}_{j=0}^{d_u}$,

✓ the constrains are $g_i(u_j) = c_j(v_i)$, for $i = 0, 1, \ldots, d_v$ and $j = 0, 1, \ldots, d_u$,

✓ and in case of open curves, $[u_0, u_{d_u}] = I_u$, and $[v_0, v_{d_v}] = I_v$.

The partial formulas for Gordon surfaces are

$$\begin{aligned} S_1(u,v) &= \langle \mathbf{c}(v),\ \mathbf{L}_{d_u}(u) \rangle && \text{blending curves,} \\ \bar{S}_2(u,v) &= \langle \mathbf{L}_{d_v}(v),\ \mathbf{g}(u) \rangle && \text{turned blending curves,} \\ S_3(u,v) &= \mathbf{L}_{d_v}(v)\ \mathbf{M}\ \mathbf{L}_{d_u}(u)^T && \text{tensor product,} \end{aligned} \tag{9.40}$$

where the matrix $\mathbf{M}$ is

$$\mathbf{M} = \begin{bmatrix} c_0(u_0) & c_0(u_1) & \cdots & c_0(u_{d_u}) \\ c_1(u_0) & \ddots & \ddots & c_1(u_{d_u}) \\ \vdots & \ddots & \ddots & \vdots \\ c_{d_v}(u_0) & c_{d_v}(u_1) & \cdots & c_{d_v}(u_{d_u}) \end{bmatrix} = \begin{bmatrix} g_0(v_0) & g_1(v_0) & \cdots & g_{d_u}(v_0) \\ g_0(v_1) & \ddots & \ddots & g_{d_u}(v_1) \\ \vdots & \ddots & \ddots & \vdots \\ g_0(v_{d_v}) & g_1(v_{d_v}) & \cdots & g_{d_u}(v_{d_v}) \end{bmatrix}.$$

Finally, the total formula for Gordon surfaces is

$$S(u,v) = S_1(u,v) + \bar{S}_2(u,v) - S_3(u,v).$$

In Figure 9.14 is shown a Gordon surface made by 4 curves in each parameter directions. The construction started with 16 points in a net. 4 and 4 curves were then made by interpolation, which were then used in the surface construction. The parameter domain is $[0,3] \times [0,3]$. The polynomial degree is 3 in both directions. The surface can be converted to a tensor product Bézier surface, but then the domain must be changed to $[0,1] \times [0,1]$ by reparametrization, see Section 4.1.2.

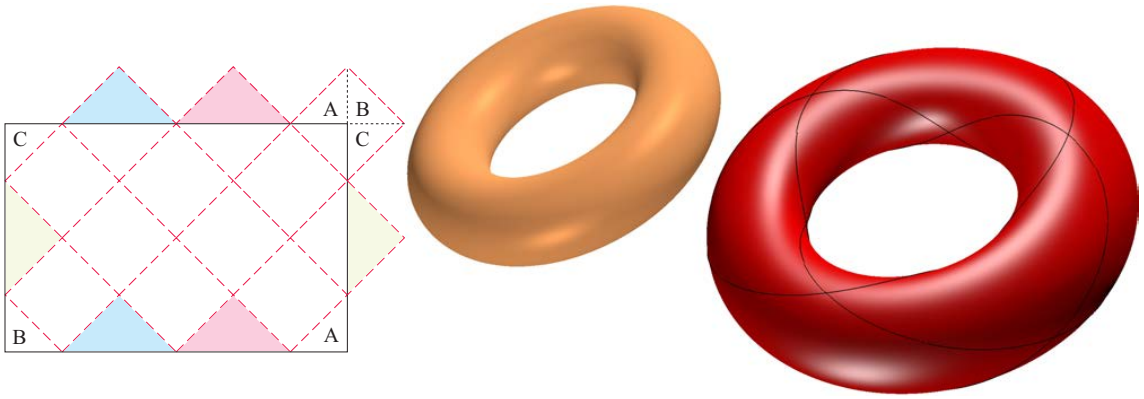


Figure 9.15: *A torus plotted in "bronze" is approximated with 12 Coons patch bicubic blending surfaces, in red. The edges are also drawn in black. To the left we see the parameter plane, to show how the 12 patches are placed.*

9.6.4 Example, Coons patch

Coon Patch bicubic blending is a boolean sum of three surfaces, Two surfaces made by blending curves using Hermite interpolation of curves and vector valued functions, and one tensor product Hermite surface.

We will now see an example of approximating a torus with 12 Coons patch bicubic blending surfaces using curves on surfaces. To the left in Figure 9.15 is the parameter domain of a torus, $\Omega = [0, 2\pi) \times [0, 2\pi)$. The domain is divided into 12 parts as illustrated in the figure. Each part is a rotated square. To show that all parts are squares, parts of some of them are apparently outside the domain, they are marked either red, blue, green or with the letters A, B or C. All these parts are also marked inside the domain with the corresponding color or letter.

In the parameter domain $\Omega \subset \mathbb{R}^2$ we have 17 points, $p_i, i = 0, 1, \ldots, 16$ which are all corners of the squares. The point are organized regularly so that the distance between them is $\frac{2}{3}\pi$ in the u-direction and π in the v-direction. There are a total of 24 edges, lines between two points, $\sigma_i(t) = (1-t)p_r + tp_s$ and where $\sigma_i'(t) = p_s - p_r$, ie $\left(\frac{\pi}{3}\ \frac{\pi}{2}\right)$ for the lines that go up to the right, and $\left(-\frac{\pi}{3}\ \frac{\pi}{2}\right)$ for the lines that go up to the left. We denote the torus for $\theta(p), p = (u, v)$. The curves (edges) in $\mathbb{R}^3$ are thus

$$c_i(t) = \theta \circ \sigma_i(t),$$

and if we denote the derivative functions "orthogonal" to the edges for

$$g_i(t) = [\theta_u\ \theta_v]_{\sigma_i(t)}\, \sigma_i'(t) = \frac{\pi}{2}\theta_v \circ \sigma_i(t) \pm \frac{\pi}{3}\theta_u \circ \sigma_i(t).$$

we get the following surfaces by blending curves and vector valued functions

$$\begin{aligned} S_1(u,v) &= H_1(u)\ c_i(v) + H_2(u)\ c_j(v) + H_3(u)\ g_i(v) + H_4(u)\ g_j(v), \\ \overline{S}_2(u,v) &= H_1(v)\ c_r(u) + H_2(v)\ c_s(u) + H_3(v)\ g_r(u) + H_4(u)\ g_s(v). \end{aligned}$$

For the tensor product surface we need, in each corner, the position, the partial derivatives in the two directions and the cross derivative. Let us call the to direction vectors in the parameter plane $a = \begin{pmatrix} \frac{\pi}{3} & \frac{\pi}{2} \end{pmatrix}$ and $b = \begin{pmatrix} -\frac{\pi}{3} & \frac{\pi}{2} \end{pmatrix}$. Then, at a point p_i, we get the position, the two partial derivatives and the cross derivative

$$\begin{aligned}
&\theta(p_i) \\
&\theta_a(p_i) = [\theta_u \ \theta_v]_{p_i} a = \frac{\pi}{2}\theta_v(p_i) + \frac{\pi}{3}\theta_u(p_i), \\
&\theta_b(p_i) = [\theta_u \ \theta_v]_{p_i} b = \frac{\pi}{2}\theta_v(p_i) - \frac{\pi}{3}\theta_u(p_i), \\
&\theta_{ab}(p_i) = \left[[\theta_{uu} \ \theta_{uv}]_{p_i} a \ \ [\theta_{vu} \ \theta_{vv}]_{p_i} a \right] b = \frac{\pi^2}{4}\theta_{vv}(p_i) + \frac{\pi^2}{9}\theta_{uu}(p_i).
\end{aligned}$$

Now, for the four corners of each square, we put these values into the matrix $\mathbf{M}$ in (9.39).

In Figure 9.15 we see, in the middle, a torus made by formula (9.3) and to the right a set of 12 red Coons patch bicubic blending surfaces that approximates the torus. the curves between the surfaces are also plotted, in black. Using sampling, we find that the largest deviation from the torus is approximately 0.0003, which is very close since the tube radius is 1.

Chapter 10

Subdivision Surfaces

Parametric surfaces, as described in the previous chapter, are based on formulas for calculating position and partial derivatives, and thus the surface normal, for given parameter values. To render a surface, the surface must be tessellated. That is, the surface is been divided into a collection of connected small quad or triangular patches. The vertices in these patches are sample points, usually created by computing values and partial derivatives and thus normals. In the case of tensor product B-splines, this can also be done by using knot insertions that are sufficiently refined.

Subdivision surfaces, similar to subdivision curves described in Section 6.7, are initially based on uniform B-splines. They where first described in 1978 by Edwin Catmull and Jim Clark, ie Catmull-Clark subdivision surface, and by Daniel Doo and Malcom Sabin, ie Doo-Sabin subdivision surface. The first one was based on bi-cubic uniform B-splines and the second was based on bi-quadratic uniform B-splines. Due to the flexibility, and thus the ability to use triangular patches, Box-splines have been widely used for schemes/algorithms that have been developed later.

In the following, we will not explain the subdivision schemes, only describe them. The same applies to analyzes of convergence and continuity. To be a valid scheme, it must converge to a continuous surface, a tensor product B-spline surface or the like, ie Box-splines. Typically surfaces that are C^1 or C^2-smooth. However, schemes for **arbitrary** topology meshes are modifications of spline based schemes, and therefore there will be vertices called extraordinary vertices[1]. These vertices change the regularity of a mesh. However, it is these vertices that make subdivision surfaces so applicable. Ulrich Reif described the behaviour near extraordinary vertices in 1995, [134]. At these points, the continuity is typically one order lower than in the rest of the surface.

Figure 10.1 illustrates 6 different sub-division schemes with stencils. In the figure, the blue points are the initial vertices and the red point are the vertices after one subdivision. The red points are either new or modified old vertices. Green points are temporary points

[1]Extraordinary vertices can be compared to Star junction, described in section 12.6.4. Sometimes they are called irregular or singular vertices. From Graph theory, the valence of a vertex is the number of edges connected to the vertex. An extraordinary vertex is one that has a valence different from its neighbors.

used in the calculation of the new vertices. Note the scheme for Doo-Sabin and Catmull-Clark. Doo-Sabin is made from 2^{nd}-degree tensor product B-spline surfaces. Figure 6.12 shows the knot insertion for 2^{nd}-degree B-splines. Here, one control point is replaced by two. We see the same in the Doo-Sabin scheme where a point is replaced by four (ie. 2×2). Catmull-Clark is made from 3^{rd}-degree tensor product B-spline surfaces. Figure 6.13 shows the knot insertion for 3^{rd}-degree B-splines. Here, two control points are replaced by three. We see the same in the Catmull-Clark subdivision scheme. Since subdivision is based on uniform B-splines, it should be possible to evaluate accurately. Jos Stam provided a method for accurate evaluation for Catmull-Clark subdivision surfaces under arbitrary parameter values, [151].

The schemes can be classified in several ways. For example, they can be classified as Primal or Dual, based on face split or vertex split. The classifications based on geometric properties are mainly:

1. whether they are corner cutting (ie approximating) or interpolating schemes,
2. the continuity, ie C^1-smooth or C^2-smooth,
3. whether they mainly are based on triangles or quads.

Note the extraordinary vertices, those with valance different from most of the others, and that converge towards a point where the smoothness is 1 order of magnitude smaller. We do not find them in for example Doo-Sabin and Mid-Edge. In Catmull-Clark, after first refinement, all the polygons are quads, and in Loop all the polygons are triangles. For Dual Doo-Sabin and Mid-Edge schemes the valance are always the same, e.g. all vertices have valance 4. Most polygons are quads, but those that are not will be of the same type in every refinements, and will converge toward a point that has 1 order of magnitude lower continuity, just like extraordinary vertices.

The starting point for a subdivision algorithm is a set of points connected in a network that covers and defines an underlying surface. The data must be arranged so that the inside and outside can be determined locally in each polygon. There are ready-made data structures for programming subdivision, such as OpenMesh, [14], but it is also possible to use a simple self-maid data structure. For example, arrays of three types of objects, a Vertex (a point and a surface normal), a Face (a polygon - ie an ordered set of Vertex indices, organized counterclockwise seen from the outside) and an edge (two Vertex indices plus two Face indices). NB! When implementing, it is important, and also possible, to make all parts of an algorithm so that they are of $O(n)$. If some parts are of $O(n \log n)$ or $O(n^2)$ the subdivision will be very slow when the number of vertices becomes large.

10.1 A selection of subdivision schemes

Figure 10.2 shows 6 subdivision surfaces, all made from a box, shown as blue lines in the figure. For quads we see 8 vertices, 6 faces and 12 edges, and for triangles 8 vertices, 12 faces and 18 edges. Stencils of schemes of these surfaces are shown in Figure 10.1. In the following, we will take a closer look at those schemes. Remember that schemes are always affine combinations of points, ie weights that sums up to 1.

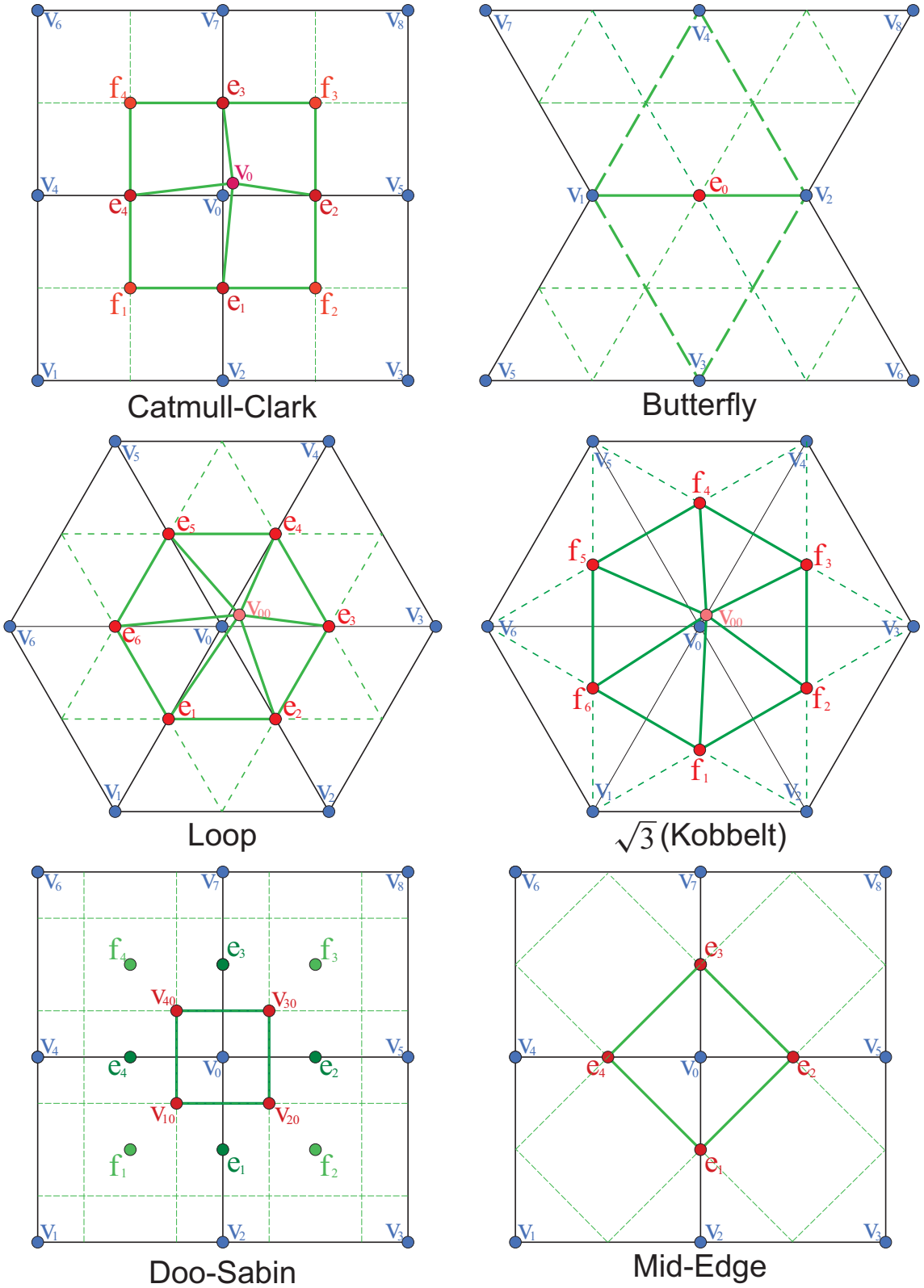


Figure 10.1: *Stencils of 6 different subdivision schemes for surfaces. Top left is Catmull-Clark, based on cubic B-splines. To the right is butterfly, an interpolation scheme based on triangles. In the middle are Loop and $\sqrt{3}$-Kobbelt, both based on triangles and box-splines. At the bottom are Mid-Edge and Doo-Sabin, both based on 2^{nd}-degree B-splines.*

10.1.1 Catmull-Clark

This is a Quad based scheme based on bi-cubic uniform B-spline knot insertion, first described in [19]. For arbitrary initial meshes, this scheme generates limit surfaces that are C^2-continuous everywhere except at extraordinary vertices where they are C^1-continuous. A surface is shown at the top of Figure 10.2. At the top left of Figure 10.1 we see a stencil of the scheme. The blue dots are the old vertices and the red are the new ones generated around one old vertex v_i. There are 3 types of new vertices, edge points, face points and correction of the old vertices. The scheme is as follows. For each refinement step and for all faces, edges and vertices, we create the following new **vertices** (points);

Face point is a point in the center of a face: $f_i = \frac{1}{m}\sum_j v_j$, where m is the number of vertices defining the face. After the first subdivision is $m = 4$.

Edge point is a point in the middle of an edge: $e_i = \frac{1}{4}(v_a + v_b + f_h + f_g)$, where v_a and v_b are the old vertices defining the current edge, and f_g and f_h are the new face points from the connected faces.

Vertex point is updating of an old vertex value: $v_i = \frac{1}{n}(F + 2E + (n-3)v_i)$, where n is the valance of the vertex, $F = \frac{1}{n}\sum_j f_j$ - is the average of all connected Face Points and $E = \frac{1}{n}\sum_j e_j$ - is the average of all connected Edge Points.

When all new vertices are made, we denote the index in the vertex array of the first Face point to be m_f and of the first Edge point to be m_e. Then there is a simple map between the index of a face and a Face point, and between the index of an edge and an Edge point. Now we can make a new array of **faces**.

✓ For each old face F_i (which has n vertices the first time, 4 later),
where $F_i = \{v_1, v_2, \ldots, v_n\}$, the Face Point is: $f_i = m_f + i$,
and the Edge Points are $\{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n\} = \{e_1 + m_e,\ e_2 + m_e,\ \ldots,\ e_n + m_e\}$,
we make n new Faces that replace F_i, ie
$F_{i1} = \{f_i, \varepsilon_n, v_1, \varepsilon_1\}$, $F_{i2} = \{f_i, \varepsilon_1, v_2, \varepsilon_2\}$, $\ldots$, $F_{in} = \{f_i, \varepsilon_{n-1}, v_n, \varepsilon_n\}$.

In the algorithm above, we use the indices of the edges of a face, $\{e_1,\ e_2,\ \ldots,\ e_n\}$. They can be stored in either a separate array or in each individual face, ie a face is a set of indices for vertices and a corresponding set of indices for edges. If there are still subdivision steps to be performed, the array of **edges** must be created and the indices of the edges in the faces must be set. As an auxiliary tool, we create an array Ψ with size equal the new Vertex array, and where each element is a list of pairs of indices to a vertex and an edge.

✓ For each new face $F_i = \{v_1, v_2, v_3, v_4\}$ (which always has 4 vertices),
we make 4 edges: $\mathbb{E}_1 = \{v_4, v_1, i\}$, $\mathbb{E}_2 = \{v_1, v_2, i\}$, $\mathbb{E}_3 = \{v_2, v_3, i\}$, $\mathbb{E}_4 = \{v_3, v_4, i\}$.
Then for each $\mathbb{E}_j$, $j = 1,\ 2,\ 3,\ 4$
If $\Psi_{\mathbb{E}_j[0]}$ do not contain $\mathbb{E}_j[1]$, the edge is not made already, then
$\Psi_{\mathbb{E}_j[1]}$.insert $\{\mathbb{E}_j[0], s\}$, where s is the size of the Edge vector E,
E.push_back($\mathbb{E}_j$) (we put the new edge in the end of the edge vector).
If the edge is made already we get the edge index from $\Psi_{\mathbb{E}_j[0]}$ and then update F_i.
and we update the edge with this face index i.

Note that if the input faces have a correct orientation, then the orientation is correct after a subdivision as well.

After computing all subdivision steps we need the surface normal at each vertex. Since the orientation is to be found in the faces, we use them to create the surface normals. The vertices/edges of a face is organized counter clockwise seen from outside. There are several works on calculating surface normals, for example [73]. However, here we will use a simple technique based on the sum of the normals of the adjacent triangular part of the faces of a Vertex. This converges towards the correct surface normal. We use the same technic as for the edges, ie we use an auxiliary tool, we create an array Ψ with size equal the new Vertex array, and where each element is a list of indices to faces. To make this array Ψ we go through all faces, and for each Face we insert its Face index in $\Psi[j]$, where j is the indices of all the vertices of the Face. The next step is to go through each element of Ψ, ie

✓ For each element of Ψ, that is Ψ_i.
  For each Face of Ψ_i, in the list of indices to vertices,
    find the index (b)efore and (a)fter index i,
      compute $n = n + (v_a - v_i) \wedge (v_b - v_i)$, (remember $\wedge$ is the vector product)
    update v_i with its unit normal $\frac{n}{|n|}$.

The surface **a)** in Figure 10.2 is a Catmull-Clark subdivision surface made from a cube (plotted in blue). 5 steps are used, and we got the following number of elements,

Subdivision step	0	1	2	3	4	5	formula
Number of vertices	8	26	98	386	1538	6146	$v+e+f$
Number of edges	12	48	192	768	3072	12288	$2v+f$
Number of faces	6	24	96	384	1536	6144	$4f$

10.1.2 Doo-Sabin and Mid-Edge

Doo-Sabin subdivision scheme is also quad based. The scheme was launched in [55], and is Chaikin's corner cutting method extended from curves to surfaces, see subsection 6.7.2. It is based on the bi-quadratic uniform tensor product B-spline surface and produce C^1-smooth limit surfaces with arbitrary topology for arbitrary initial meshes[2]. After one subdivision, all vertices have valence 4.

A stencil of the Doo-Sabin scheme is displayed at the bottom left of Figure 10.1. There we see that the Face and Edge points to be made are only help points and not new vertices. The new vertices are 4 points that replace each of the old vertices. Then, for each refinement step and for all faces and edges, we create the following two arrays of help points;

Face Point is a point in the center of a face: $f_i = \frac{1}{m}\sum_j v_j$, where m is the number of vertices defining the face. For most faces are $m = 4$.

[2] An auxiliary point can improve the shape of Doo-Sabin subdivision, see [22].

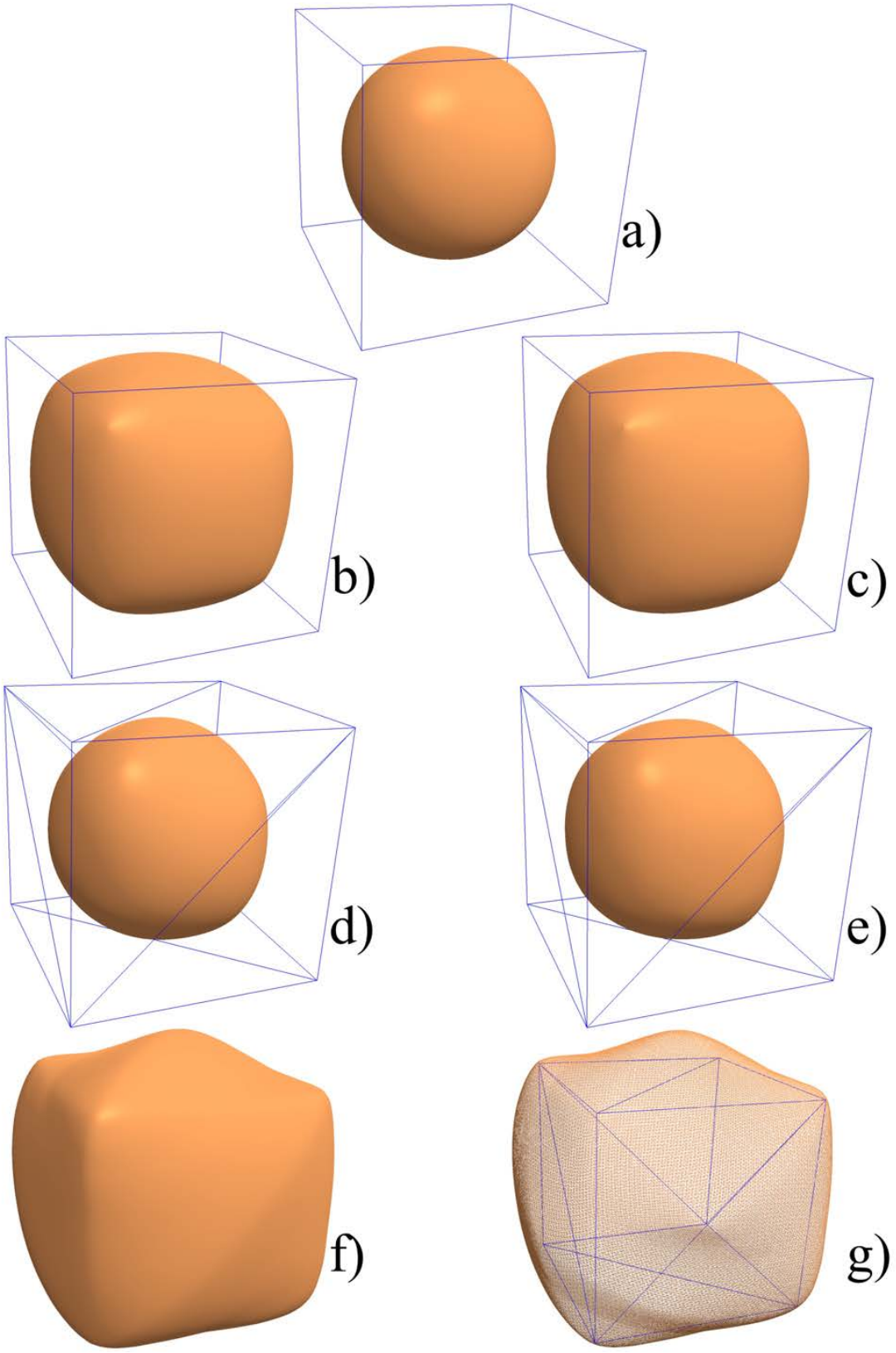


Figure 10.2: *Catmull-Clark is shown in a), Doo-Sabin in b), Mid-Edge in c), Loop in d), $\sqrt{3}$-Kobbelt in e) and the interpolation scheme butterfly is shown in f) and g). We see that Doo-Sabin and Mid-Edge gives the same surface, and so do Loop and $\sqrt{3}$-Kobbelt.*

Edge Point is a point in the middle of an edge: $e_i = \frac{1}{2}(v_a + v_b)$, where v_a and v_b are the old vertices defining the current edge.

Furthermore, we can either use auxiliary arrays or extended data structure to connect faces and edges to each vertex. Next step is to make a new array of vertices replacing the old ones.

✓ For each old vertex v_i, (which has n faces and edges the first time, 4 later)
For each face F_j connected to the vertex v_i we make a new vertex v_{ji},
here f_j is the face point, e_a and e_b are the connected Edge points,
$v_{ji} = \frac{1}{4}(v_i + f_j + e_a + e_b)$.
Finally we create a face based on the n vertices we have now created.

After this we make a face inside each old face and we make a face around each old edge. The edges and connection to the faces and the surface normals can be made in the same way as in the Catmull-Clark scheme.

The surface **b)** in Figure 10.2 is a Doo-Sabin subdivision surface made from a cube that is plotted in blue. 5 steps are used, and we got the following number of elements,

Subdivision step	0	1	2	3	4	5	formula
Number of vertices	8	24	96	384	1536	6144	$4v$
Number of edges	12	48	192	768	3072	12288	$8v$
Number of faces	6	26	98	386	1538	6146	$v+e+f$

Note that the formula is not correct for the first step because of the vertex valance.

Mid-Edge subdivision scheme was proposed independently by Jörg Peters and Ulrich Reif [129] and Ayman Habib and Joe Warren [85]. The former used the mid-point of each edge to build the new mesh. The latter used a four-directional box spline to build the scheme. This scheme generates the same surface as Doo-Sabin, but converge noticeably slower, in fact 2 to 1 as we can see in the tables.

The scheme is quite simple, we replace the old vertices with new ones. The new ones are the center point of each old edge. After making the new vertices we make the faces inside each old face and around each old vertex. The edges and surface normals are made in the same way as for Doo-Sabin scheme.

The surface **c)** in Figure 10.2 is a Mid-Edge subdivision surface made from a cube. 8 steps are used, and we got the following number of elements,

Subdivision step	0	1	2	3	4	5	6	7	8	formula
Number of vertices	8	12	24	48	96	192	384	768	1536	e
Number of edges	12	24	48	96	192	384	768	1536	3072	$2e$
Number of faces	6	14	26	50	98	194	386	770	1538	$v+f$

10.1.3 Loop and $\sqrt{3}$

Loop subdivision scheme, based on triangles, was introduced in 1987 by Charles Loop, [113] and [114]. He proposed a subdivision scheme based on a quartic box-splines of six

direction vectors to provide a rule to generate C^2-continuous limit surfaces everywhere except at extraordinary vertices where they are C^1. A stencil of the scheme is shown in the middle left of Figure 10.1, and a surface generated from a "box" is shown in Figure 10.2 as surface **d)**. The "box" is shown as triangles drawn in blue lines in Figure 10.2. The lack of symmetry that we see in the plot is due to the triangulation of the cube. Normally the valance should be 6 for most of the vertices, but in the example in Figure 10.1 half of the initial vertices have valance 4 and the rest have valance 5, and they will all keep their valance after all the subdivisions. However, all new vertices will have valance 6.

Loop scheme are quite simple, it is basically to divide each edge in two, and then split a triangle in 4. In each refinement step, be sure that edges and faces are connect, then for all edges and for all vertices, update the old vertices and create the new **vertices** as follows:

Vertex point is an update of the position of an old vertex: $v_i = (1-\alpha_n)\, v_i + \frac{\alpha_n}{n}\sum_{j\in\beta} v_j$, where n is the valance of the vertex to update, β is the set of indices of the neighboring vertices, and α_n is a valance value described in (10.1). The example from Figure 10.1 is: $v_{00} = (1-\alpha_6)\; v_0 + \frac{\alpha_6}{6}\sum_{j=1}^{6} v_j \approx 0.635 v_0 + 0.0625\sum_{j=1}^{6} v_j$.

Edge point is a point in the middle of an edge: $e_i = \frac{1}{8}\left(3v_a + 3v_b + v_f + v_g\right)$, where v_a and v_b are the old vertices defining the current edge, and v_f and v_g are the missing vertices from the two connected faces/triangles. An example from Figure 10.1 is $e_3 = \frac{1}{8}\left(3v_0 + 3v_3 + v_2 + v_4\right)$.

All vertices are now updated or made. We denote the index in the vertex array of the first Edge point to be m_e. Then there is a simple map between the index of an edge and an Edge point. Now we update and make new faces/triangles.

✓ For each old Face/triangle F_i (which has 3 vertices),
where $F_i = \{v_1, v_2, v_3\}$ and
the Edge Points are $\{\varepsilon_1, \varepsilon_2, \varepsilon_3\} = \{e_1 + m_e,\ e_2 + m_e,\ e_3 + m_e\}$,
we updating the old Face F_i, and make 3 new Faces F_{i2}, F_{i3}, F_{i4}, ie
$F_i = \{v_1, \varepsilon_1, \varepsilon_3\}$, $F_{i2} = \{\varepsilon_1, v_2, \varepsilon_2\}$, $F_{i3} = \{\varepsilon_1,\ \varepsilon_2,\ \varepsilon_3\}$ and $F_{i4} = \{\varepsilon_2, v_3, \varepsilon_3\}$.

Finally, edges and surface normals can be made in the same way as in Catmull-Clark.

The valance values α_n for Loop subdivision used in updating old vertices are

$$\alpha_n = \frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2. \tag{10.1}$$

Calculated values for some valance numbers can be seen i Table 10.1.

The surface **d)** in Figure 10.2 is a Loop subdivision surface made from a triangulated cube that is plotted in blue. 5 steps are used, and we got the following number of elements,

Subdivision step	0	1	2	3	4	5	formula
Number of vertices	8	26	98	386	1538	6146	$v+e$
Number of edges	18	72	288	1152	4608	18432	$2e+3f$
Number of faces	12	48	192	768	3072	12288	$4f$

Valance - $n =$	4	5	6	7	8
Loop: $\alpha_n =$	0.4844	0.4205	0.375	0.3432	0.3205
$\sqrt{3}$-Kobbelt: $\alpha_n =$	0.4444	0.3758	0.3333	0.3059	0.2873

Table 10.1: *The weighting values α_4, α_5, α_6, α_7, and α_8.*

If we compare this numbers with the Catmull-Clark scheme on page 201 we see that the number of vertices are the same, the number of faces are twice as large (a quad divided in two triangles), and the number of edges for Loop is equal the number of edges plus the number of faces in Catmull-Clark.

$\sqrt{3}$-Kobbelt scheme is also based on triangles - It was developed by Leif Kobbelt, [99], and it handles arbitrary triangular meshes, it is C^2 continuous everywhere except at extraordinary vertices where it is C^1 continuous and it offers a natural adaptive refinement when required. And also, it is a "dual" scheme for triangle meshes and it has a slower refinement rate than Loop, but it converges toward the same surface.

A $\sqrt{3}$-Kobbelt scheme is basically to make a new vertices in the center of each face/triangle, update old vertices and divide the old triangles in 3 and swap all old edges, see the stencil in the middle right of Figure 10.1. Thus we get

Vertex point is an update of the position of an old vertex: $v_i = (1-\alpha_n)\, v_i + \frac{\alpha_n}{n} \sum_{j\in\beta} v_j$, where n is the valance of the vertex to update, β is the set of indices of the neighboring vertices, and α_n is a valance value described in (10.2). The example from Figure 10.1 is: $v_{00} = (1-\alpha_6)\; v_0 + \frac{\alpha_6}{6} \sum_{j=1}^{6} v_j$.

Face point is a point in the middle of a face: $f_i = \frac{1}{3}(v_a + v_b + v_c)$, where v_a, v_ab and v_c are the old vertices defining the current face/triangle. An example from Figure 10.1 is $f_3 = \frac{1}{3}(v_0 + v_3 + v_4)$.

All vertices are now updated or made. We denote the index in the vertex array of the first Face point to be m_f. Then there is a simple map between the index of an face and an Face point. Now we update and make new faces/triangles.

✓ For each old Edge E_i,
shared by two faces/triangles with indices f_a and f_b
and the indices of the Face Points become $\phi_a = f_a + m_f$ and $\phi_b = f_b + m_f$
we make two new Faces $F_{2i} = \{E_i[0],\ \phi_a,\ \phi_b\}$ and $F_{2i+1} = \{E_i[1],\ \phi_b,\ \phi_a\}$

Note that it is important that the two faces of an edge are organized so that f_a is on the right side of the edge seen from the outside of the surface. Now, edges and surface normals can be made in the same way as in Catmull-Clark.

The valance values α_n for $\sqrt{3}$-Kobbelt subdivision used in updating old vertices are

$$\alpha_n = \frac{4 - 2\cos\frac{2\pi}{n}}{9}, \tag{10.2}$$

and calculated values for some valance numbers can be seen i Table 10.1.

The surface **e)** in Figure 10.2 is a $\sqrt{3}$-Kobbelt subdivision surface made from a triangulated cube that is plotted in blue. 6 steps are used, and we got the following number of elements,

Subdivision step	0	1	2	3	4	5	6	formula
Number of vertices	8	20	56	164	488	1460	4376	$v+f$
Number of edges	18	54	162	486	1458	4374	13122	$e+3f$
Number of faces	12	36	108	324	972	2916	8748	$2e$

10.1.4 Butterfly

Catmull-Rom subdivision splines, discussed in section 6.7.1, and generalized to "A 4-point interpolatory subdivision scheme" in [56], interpolate all points, both starting points and points generated by subdivision steps.

In 1990 Nira Dyn at al provided a subdivision scheme for surfaces based on triangles, [59]. The scheme is an extension of (6.29), where the tension parameter ω was introduced. Recall that if $\omega = \frac{1}{16}$ than (6.29) is an ordinary Catmull-Rom subdivision spline. If we use the indices of the vertices from the stencil on the upper right side in Figure 10.1, Dyn et al started with $e_0 = u(v_1+v_2)+v(v_3+v_4)-\omega(v_5+v_6+v_7+v_8)$ and showed that to be C^0 then $2u+2v-4\omega = 1$ and if $u = \frac{1}{2}$ it is C^1, and it follows that $v = 2\omega$.

The scheme is called Butterfly because of its shape. It is based on triangles and the valance is thus normally 6. It also work with valance 5, 7 and 8. In out example is the valance initially 4 at some vertices. We have then defined v_6 and v_8 to be the same point. Then, the scheme is as follows. For each refinement step and for all faces, edges and vertices, we create the following new vertices (points),

Vertex point is the set of the old vertices and will remain unchanged.

Edge point is a point in the middle of an edge. Using indices from the sketch on the upper right side in Figure 10.1: $e_0 = \frac{1}{2}(v_1+v_2)+2\omega(v_3+v_4)-\omega(v_5+v_6+v_7+v_8)$.

The tension parameter ω should not be too far from $\frac{1}{16}$, and it is possible to have different value on each individual vertex. All vertices are now updated or ready for reuse. We denote the index in the vertex array of the first Edge point to be m_e. Then there is a simple map between the index of an edge and an Edge point. To update and make new faces/triangles, edges and finally normals we use the same procedure as for Loop.

Surface **f)** in Figure 10.2 is made using the Butterfly scheme. It is made from the triangulated cube which is plotted in blue in **g)** in Figure 10.2. 6 steps are used, and we got the following number of elements,

Subdivision step	0	1	2	3	4	5	6	formula
Number of vertices	8	26	98	386	1538	6146	24578	$v+e$
Number of edges	18	72	288	1152	4608	18432	73728	$2e+3f$
Number of faces	12	48	192	768	3072	12288	49152	$4f$

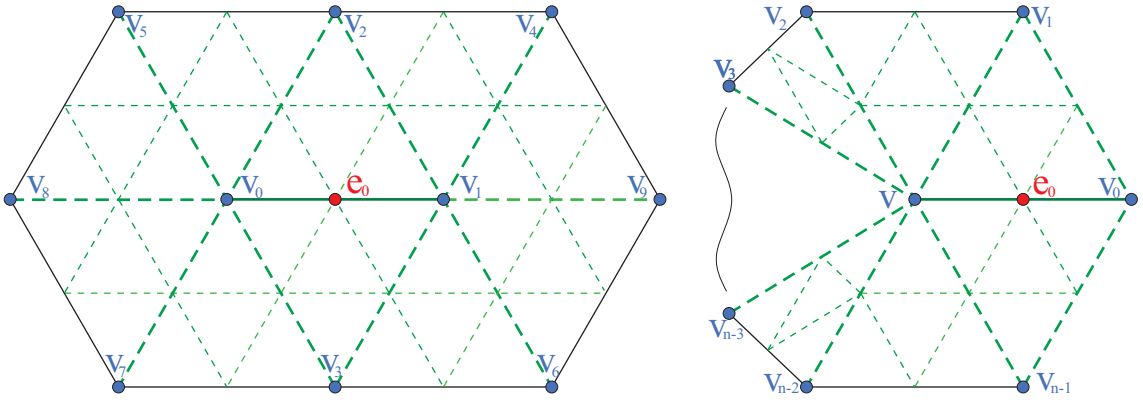


Figure 10.3: *Modified Butterfly scheme. To the left is a stencil of an ordinary 10 point scheme, where both vertices of an edge has valance 6. To the right is a stencil of a scheme where one vertex v has a valance different from 6.*

The Butterfly scheme is an 8-point interpolation scheme, as we can observe in the Butterfly stencil in Figure 10.1. There is an extension, a 10-point scheme also called Modified Butterfly, described in [169]. Here, vertices with valance different from 6 have a modified scheme using only the extraordinary vertex. This scheme is illustrated on the right side in Figure 10.3. To the left in the figure there is an ordinary 10 points scheme. It require that both vertices of the edge have valance 6.

The Modified Butterfly slightly change the way of making new edge, which gives,

Edge point , where both vertices have valance 6:
Using index names from the stencil on the left side in Figure 10.3:
$e_0 = a(v_0+v_1)+b(v_2+v_3)+c(v_4+v_5+v_6+v_7)+d(v_9+v_9)$
where $a = \frac{1}{2}-w$, $b = \frac{1}{8}+2w$, $c = -\frac{1}{16}-w$ and $d = w$.

Edge point , where one vertex have valance $\neq 6$:
Using index names from the sketch on the right side in Figure 10.3:
If the valance $n = 3$, $e_0 = \frac{3}{4}v + \frac{5}{12}v_0 - \frac{1}{12}v_1 - \frac{1}{12}v_2$.
If the valance $n = 4$, $e_0 = \frac{3}{4}v + \frac{3}{8}v_0 + \cancel{0v_1} - \frac{1}{8}v_2 + \cancel{0v_3}$.
If the valance $n > 4$, $e_0 = \frac{3}{4}v + \sum_{j=0}^{n-1}\frac{1}{n}\left(\frac{1}{4}+\cos\frac{2\pi j}{n}\frac{1}{2}+\cos\frac{4\pi j}{n}\right)v_j$.

Edge point , where both vertices have valance $\neq 6$:
Use the formula above for both v and v_0, and use the middle, ie $e_0 = \frac{e_0(v)+e_0(v_0)}{2}$.

Note that the w used for valance 6 is not the same as the previous ω used in ordinary Butterfly. In fact, if $w = 0$ we get an ordinary 8 point Butterfly scheme with $\omega = \frac{1}{16}$. Also note that since $\sum_{j=0}^{n-1}\frac{1}{n}\left(\frac{1}{4}+\cos\frac{2\pi j}{n}\frac{1}{2}+\cos\frac{4\pi j}{n}\right) = \frac{1}{4}$, the weights in all parts of the Modified Butterfly scheme sums up to 1.

In Figure 10.4 there is on the left a surface made by ordinary Butterfly and on the right a surface made by Modified Butterfly. They both started from a cube withe 8 corner points plus one point in the middle of each plane of the cube, a total of 14 points defining 24 triangles. In the figure we see a hint of the initial triangles in blue. As we can see, the

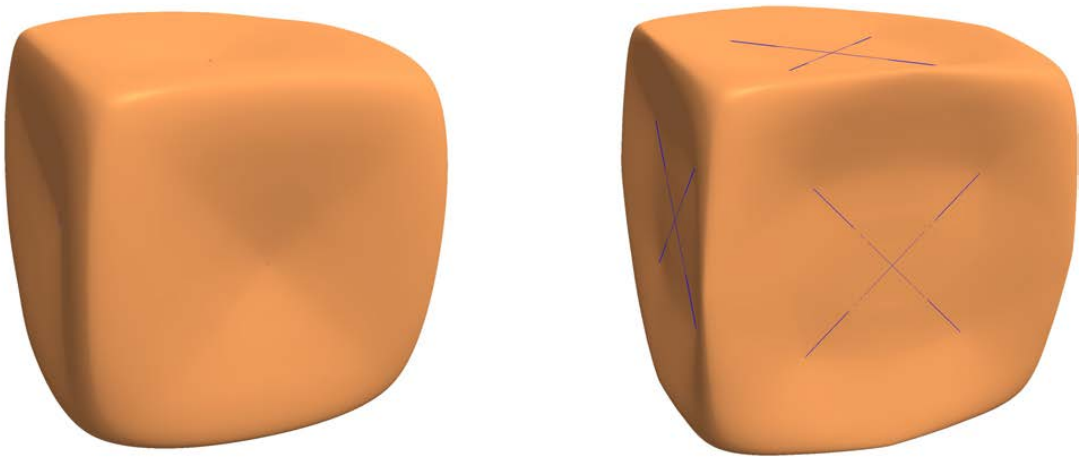


Figure 10.4: *To the left is a surface from ordinary Butterfly and to the right is one from Modified Butterfly. The starting point for both is 8 points that form a cube and one point in the center of each of the 6 sides of the cube, ie a total of 14 points.*

vertices from the points in the middle of the planes have valance 4. The ordinary Butterfly do not necessarily handle valance 4 smooth as we can observe on the left in Figure 10.4.

10.1.5 Interpolatory Quad - Kobbelt

As we have seen for curves, The "4-point interpolatory subdivision scheme", (6.29), interpolates all points, both starting points and points generated by subdivision steps. If we extend this to be used in two directions, as tensor product B-splines we get a quad-based interpolatory subdivision scheme. This was recognized and tested by Leif Kobbelt in [98], and he also introduced a possible solution for extraordinary vertices.

In section 9.5.3 we showed that a Tensor product surface is just a surface from blending curves. It follows that in the case of interpolation the scheme is simply the curve scheme in two directions. If we have regular vertices then for all edges we just make edge points with the curve scheme (6.29). For face points it is also easy, we can make them from the 4×4 vertices that surround the face using a tensor product (outer product) of the curve scheme. But since we have already made the edge points, and due to the symmetry of the tensor product, we can instead only use the curve scheme on these new edge points, and the result will be the same no matter which direction we choose.

Thus the scheme is to make one new vertex in the "middle" of each edge, and one new vertex in the "middle" of each face. For regular vertices, with valance 4, we get

Edge point is a point in the "middle" of an edge. If both vertices of an edge is regular, we use the scheme (10.3), where p_i and p_{i+1} are the vertices of the edge and p_{i-1} and p_{i+2} are the next vertices on the extension of the edge.

Face point is a point in the "middle" of a face. If all vertices of a face is regular, we use the scheme (10.3), where p_i and p_{i+1} are new edge points of two opposite edges of the face, and p_{i-1} and p_{i+2} are the new edge points on the opposite edges of the

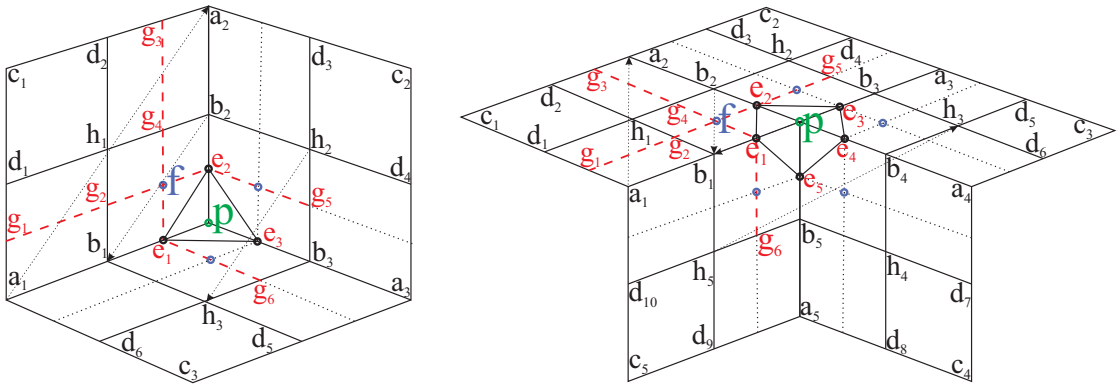


Figure 10.5: *An illustration of how to generate a face point f when the valence at the vertex p is 3 (the figure on the left) and 5 (the figure on the right). g_3, g_4, e_1, g_6 and g_1, g_2, e_2, g_5 are edge points, e_1, e_2, e_3 and e_4, e_5 are the edge points closest to p.*

opposite faces of the first two edges.

To simplify the implementation of edge and face points near extraordinary vertices, it is useful to keep the symmetry of the face points. Therefore, to calculate an edge point when one or both vertices of the edge are extraordinary, **virtual vertices** can be introduced. Remember (6.29), the scheme can be reformulated to

$$\widehat{p_i} = \mu(p_i + p_{i+1}) - \omega(p_{i-1} + p_{i+2}), \quad \text{where} \quad \mu = \frac{1}{2} + \omega \tag{10.3}$$

First we look at valence 3, left side of Figure 10.5. We calculate the face point f from g_3, g_4, e_1, g_6 and set this equal to f calculated from g_1, g_2, e_2, g_5, ie

$$\begin{Bmatrix} \omega^2 c_1 & -\omega\mu\, d_2 & -\omega\mu a_2 & +\omega^2 d_3 \\ -\omega\mu\, d_1 & +\mu^2 h_1 & +\mu^2 b_2 & -\omega\mu\, h_2 \\ & +\mu\, e_1 & & \\ +\omega^2 d_6 & -\omega\mu\, h_3 & -\omega\mu b_3 & +\omega^2 h_2 \end{Bmatrix} = \begin{Bmatrix} \omega^2 c_1 & -\omega\mu\, d_1 & -\omega\mu\, a_1 & +\omega^2 d_6 \\ -\omega\mu\, d_2 & +\mu^2 h_1 & +\mu^2 b_1 & -\omega\mu\, h_3 \\ & +\mu\, e_2 & & \\ +\omega^2 d_3 & -\omega\mu\, h_2 & -\omega\mu\, b_3 & +\omega^2 h_3 \end{Bmatrix}.$$

Summing and reorganizing this we get

$$e_1 - e_2 = \omega(a_2 - a_1) + \mu(b_1 - b_2) + \frac{\omega^2}{\mu}(h_3 - h_2). \tag{10.4}$$

We now do the same for valence 5, ie the right side of Figure 10.5 we get

$$\begin{Bmatrix} \omega^2 c_1 & -\omega\mu\, d_2 & -\omega\mu a_2 & +\omega^2 d_3 \\ -\omega\mu\, d_1 & +\mu^2 h_1 & +\mu^2 b_2 & -\omega\mu\, h_2 \\ & +\mu\, e_1 & & \\ +\omega^2 d_{10} & -\omega\mu\, h_5 & -\omega\mu b_5 & +\omega^2 h_4 \end{Bmatrix} = \begin{Bmatrix} \omega^2 c_1 & -\omega\mu\, d_1 & -\omega\mu\, a_1 & +\omega^2 d_{10} \\ -\omega\mu\, d_2 & +\mu^2 h_1 & +\mu^2 b_1 & -\omega\mu\, h_5 \\ & +\mu\, e_2 & & \\ +\omega^2 d_3 & -\omega\mu\, h_2 & -\omega\mu\, b_3 & +\omega^2 h_3 \end{Bmatrix},$$

$$e_1 - e_2 = \omega(a_2 - a_1) + \mu(b_1 - b_2) + \omega(b_5 - b_3) + \frac{\omega^2}{\mu}(h_3 - h_4). \tag{10.5}$$

Note that one edge point is used to create face points in two adjacent faces. It follows that in all faces connected to an extraordinary vertex, there is a locked vector between the

two edge points on the adjacent edges. This means that for all extraordinary vertices with valance n, the nearest edge points lie in the corners of an n-edged polygon. For $n = 3$ this is a triangle and for $n = 5$ it is a pentagon, both of which can see in Figure 10.5.

The shape and orientation of the polygon is given. However, the position of the polygon is a freedom we can use, but under the restriction that the barycenter of the edge points must converge smoothly towards the extraordinary vertex. One way to do this is to compar with ordinary vertices. The barycenter X for ordinary vertices, $n = 4$ is

$$X = \frac{1}{n}\sum_{i=1}^{n} e_i = \mu\ p + \frac{1}{n}\left(\frac{1}{2}\sum_{i=1}^{n} b_i - \omega\sum_{i=1}^{n} a_i\right) \tag{10.6}$$

Because we, for extraordinary vertices, can calculate the distance vector between each pair of edge points, eg (10.4) and (10.5), we can compute the barycenter only depending on one of the edge points,

$$X = \frac{1}{n}\sum_{i=1}^{n} e_i = e_1 + \frac{1}{n}\sum_{i=1}^{n-1}(n-i)(e_{i+1} - e_i) \tag{10.7}$$

We then set (10.6) equal (10.7) and reorganize it. The result is a general formula for calculating edge points connected to one extraordinary vertex:

$$e_1 = \mu\ p + \frac{1}{n}\left(\frac{1}{2}\sum_{i=1}^{n} b_i - \omega\sum_{i=1}^{n} a_i + \sum_{i=1}^{n-1}(n-i)(e_i - e_{i+1})\right). \tag{10.8}$$

Finally we reformulate this to

$$e_1 = -\omega\ a_1 + \mu\ b_1 + \mu\ p - \omega\ v, \tag{10.9}$$

where v is a virtual vertex that can be expressed by combining (10.8) with either (10.4) if $n = 3$ or (10.5) if $n = 5$, or a related expression if $n > 5$. Note that if b_1 also is an extraordinary vertex, we replace a_1 with another virtual vertex.

If $n = 3$ we replace the vectors $e_1 - e_2$ in (10.8) with (10.4), and $e_2 - e_3$ with a rotated (10.4), and get the virtual vertex

$$v = \frac{1}{3}\sum_{i=1}^{3} b_i + \frac{\omega}{\mu}\left(h_2 - \frac{1}{3}\sum_{i=1}^{3} h_i\right). \tag{10.10}$$

If $n = 5$ we replace the vectors $e_1 - e_2$ with (10.5), and $e_2 - e_3$, $e_3 - e_4$ and $e_4 - e_5$ with a a 1 times, 2 times and 3 times rotated (10.5), and get the virtual vertex

$$v = b_3 + b_4 - \left[\frac{1}{5}\sum_{i=1}^{5} b_i + \frac{\omega}{\mu}\left(h_3 - \frac{1}{5}\sum_{i=1}^{5} h_i\right)\right]. \tag{10.11}$$

In Figure 10.6 is an example of an interpolatory subdivision surface based on quads shown. The algorithm are using formula (10.3), (10.9), (10.10) and (10.11). The left

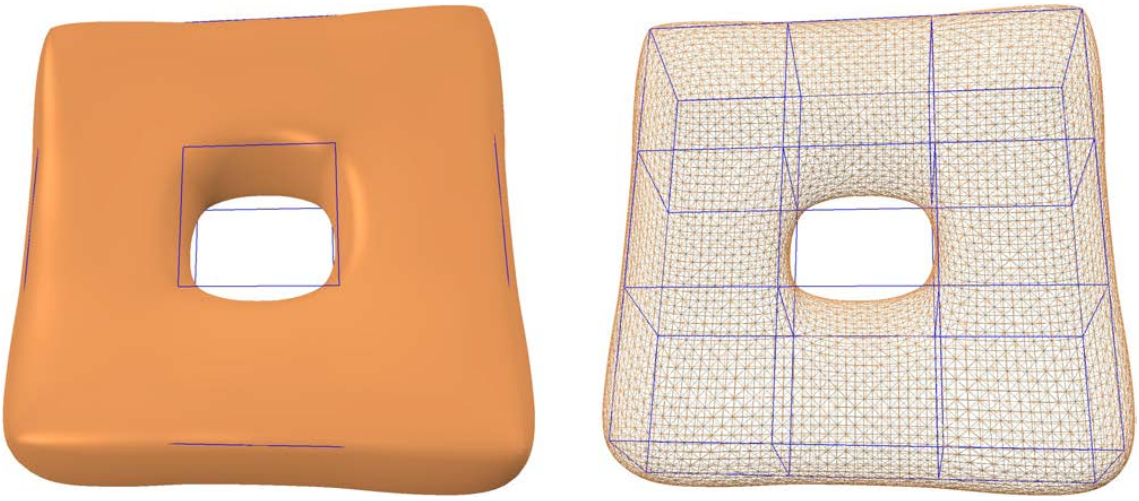


Figure 10.6: *An interpolating Quad subdivision surface. To the right, the initial points and quads are shown in blue. There are 8 corner points with valance 3 and 8 inside corners with valance 5. The remaining 16 points are ordinary vertices with valance 4.*

side of Figure 10.6 shows the initial points and quads in blue. There are 8 corner points with valance 3 and 8 inner corners with valance 5. The remaining 16 points are ordinary vertices with valance 4. Four refinement steps were run. From the initial 32 points, 64 edges and 32 faces, this generated 8192 vertices, and the program used 95 milliseconds on the computation. In total, including the generation of normals and triangles for the graphics, the program used 161 milliseconds. The program was run on Intel Core i9-9900 CPU, 3.6 GHz and is made of simple but $O(n)$-algorithms and without optimization.

Chapter 11

Two surface blending

In Boolean sum surfaces, three surfaces are blended together into one surface. We will now look at a method where we only use the first two surfaces, the surfaces that are made by blending curves.

From the expression (9.38), Coons patch - bicubic blending, we have the first two surfaces $S_1(u,v)$ and $\overline{S}_2(u,v)$, both made from blending curves. These two surfaces are defined on a common domain $U = [0,1] \times [0,1]$. The difference surface is

$$\widetilde{S}(u,v) = \overline{S}_2(u,v) - S_1(u,v). \tag{11.1}$$

Using this we get a blending surface

$$S(u,v) = S_1(u,v) + B(u,v)\, \widetilde{S}(u,v), \tag{11.2}$$

where $B(u,v)$ is a blending function in two variables (2-p). Recall Chapter 7, Definition 7.1, where the B-function in one variable (1-p) is defined. In the following we will look at how we can construct a 2-p B-function, $B(u,v)$, with similar properties as a 1-p B-function, ie internal C^d-smooth where d is the order of the 1-p B-function used to build $B(u,v)$, and that all derivatives up to order d are zero at the boundaries of the domain.

11.1 2-parameter B-function

To construct a 2-p B-function $B(u,v)$ of at least order 1 and preferably larger than 1, cf. Definition 7.1, we start with a help function

$$g(u) = \begin{cases} 1 - \frac{1}{2}B(2u), & \text{if } \ u \le \frac{1}{2}, \\ 1 - \frac{1}{2}B(2-2u), & \text{otherwise}, \end{cases} \tag{11.3}$$

with smoothness in accordance with the order of the B function, and which is symmetric about $u = \frac{1}{2}$. It follows that

$$g'(u) \ = \begin{cases} -B'(2u), & \text{if } \ u \le \frac{1}{2} \\ B'(2-2u), & \text{otherwise} \end{cases} \qquad g''(u) \ = \begin{cases} -2\,B''(2u), & \text{if } \ u \le \frac{1}{2}, \\ -2\,B''(2-2u), & \text{otherwise.} \end{cases} \tag{11.4}$$

The next two functions are

$$a(u) = 2u(1-u), \quad \text{where} \quad a' = 2(1-2u), \quad \text{and} \quad a'' = -4, \tag{11.5}$$

and

$$t(u,v) = \begin{cases} \frac{v}{a(u)}, & \text{if} \quad v < a(u) \\ \frac{1-v}{a(u)}, & \text{if} \quad v > 1-a(u) \\ 1 & \text{otherwise} \end{cases} \tag{11.6}$$

We can see that $t(u,v)$ is continues. It follows that

$$t_u = \begin{cases} \frac{-v\,a'}{a^2}, & \text{if} \quad v < a \\ \frac{-(1-v)\,a'}{a^2}, & \text{if} \quad v > 1-a \\ 0 & \text{otherwise} \end{cases}, \qquad t_v = \begin{cases} \frac{1}{a}, & \text{if} \quad v < a \\ \frac{-1}{a}, & \text{if} \quad v > 1-a \\ 0, & \text{otherwise} \end{cases}$$

$$t_{uu} = \begin{cases} \frac{v(2(a')^2-a''a)}{a^3}, & \text{if} \quad v < a \\ \frac{(1-v)(2(a')^2-a''a)}{a^3}, & \text{if} \quad v > 1-a \\ 0, & \text{otherwise} \end{cases}, \quad t_{uv} = \begin{cases} \frac{-a'}{a^2}, & \text{if} \quad v < a \\ \frac{a'}{a^2}, & \text{if} \quad v > 1-a \\ 0, & \text{otherwise} \end{cases} \tag{11.7}$$

and $t_{vv} = 0$.

Finally a blending function can be constructed,

$$\hat{B}(u,v) = g(u)\; B \circ t(u,v), \tag{11.8}$$

and where the partial derivatives are

$$\begin{aligned} \hat{B}_u &= g'\,B + g\,B'\,t_u, \\ \hat{B}_v &= g\,B'\,t_v, \\ \hat{B}_{uu} &= g''\,B + 2g'\,B'\,t_u + g\,(B''\,t_u^2 + B'\,t_{uu}), \\ \hat{B}_{uv} &= g'\,B'\,t_v + g\,(B''\,t_u\,t_v + B'\,t_{uv}), \\ \hat{B}_{vv} &= g\,B''\,t_v^2. \end{aligned} \tag{11.9}$$

One problem with the blending function $B(u,v)$ is that it does not behave symmetric in the sense that

$$B(u,v) + B(v,u) = 1 \tag{11.10}$$

One would expect that a blending function will behave the same way in the two parameter directions. To modify a blending function to behave like this we can define,

$$B(u,v) = \frac{1}{2}(1 + \hat{B}(u,v) - \hat{B}(v,u)). \tag{11.11}$$

A plot of the blending functions $B(u,v)$) as a surface can be seen in Figure 11.1. To get a better impression of the function, it is displayed from two sides. If the surface is turned upside down and rotated 90°, it will be exactly the same.

In addition to being symmetrical as described above, $B(u,v)$ is designed to have some special properties, ie for a given d, to be internal C^d-smooth and that all derivatives up to

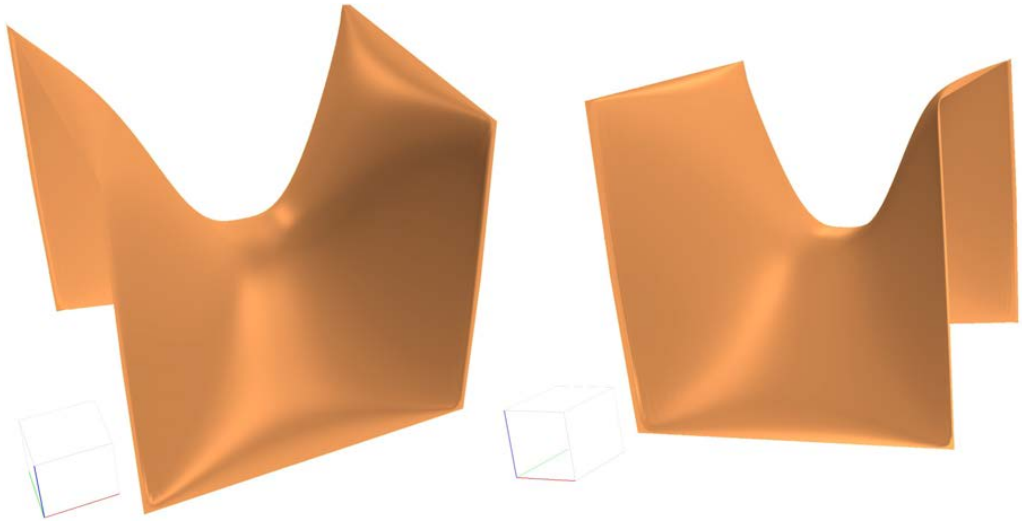


Figure 11.1: *We see two different view of the blending function $B(u,v)$ defined in expression (11.11). It appears as a surface. We can see that it seems that the derivatives along the edges are zero and that there is a kind of discontinuity at the corners.*

order d are zero at the boundaries of the domain. Note that B and $\hat{B}$ will have the same properties, so all propositions that apply to $\hat{B}$ will also apply to B, so we only call it B. A summary of the properties of $B(u,v)$ is

- that the value is 0 "internally" on two opposite edges (with constant v value),
- that the value is 1 over all the other two edges (with constant u value),
- it is symmetric, i.e. $B(u,v)+B(v,u)=1$,
- it is "internally" C^d-smooth,
- that all derivatives up to order d are 0 on the edges,
- and it is discontinuous in all the corners in the "u direction"

In appendix C.9 there is a set of proofs showing that the 2-p B function $B(u,v)$ satisfies all these properties.

This 2-p B-function essentially reflects the properties of a 1-p B-function, in addition to the fact that all derivatives along the edges are zero in **all directions**. However, this 2-p B-function is special because although all derivatives are continuous, the function itself is discontinuous at the corners, i.e. it jumps from 0 to 1 in the "u direction", as pointed out by the last item in the list of properties.

The last point in the summary of the properties is special, because it means that we can only use $B(u,v)$ (11.11) to blend two surfaces when the two surfaces are equal in the four corners - if the result should be C^d-continuous. We will look more closely at applications of this in the next section.

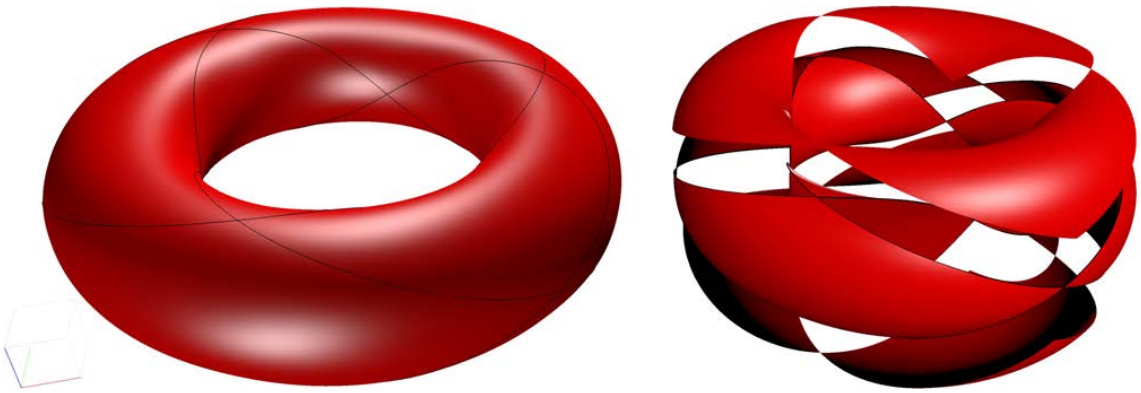


Figure 11.2: *On left hand side is there a plot of 12 surfaces that together form a C^1-smooth surface, which is approximately a torus. We can see the boundary curves located on the "torus". The derivatives that are "orthogonal" to this boundary curves are also according to a torus. The 12 surfaces are made by using Hermite blending surfaces. On the right side are also the 12 surfaces that together are almost a torus, but here they are moved apart so that we better can see them separately.*

11.2 Hermite 2-p blending surface

With "Coons patch" - bicubic blending it is possible to fill a square hole in a surface in a smooth way, see section 9.6.2. However, the blending of two surfaces is an alternative method. We start by creating two surfaces using Hermite blending of curves in two directions, as in "Coons patch". We call these $S_1(u,v)$ and $\overline{S}_2(u,v)$, and the difference surface for $\widetilde{S}(u,v) = \overline{S}_2(u,v) - S_1(u,v)$ (11.1). We then use these in accordance with (11.2), i.e.

$$S(u,v) = S_1(u,v) + B(u,v)\,\widetilde{S}(u,v),$$

where $\widetilde{S}(u,v)$ is defined in (11.1). The partial derivatives then become

$$\begin{aligned}
S_u &= S_{1u} + B_u\,\widetilde{S} + B\,\widetilde{S}_u,\\
S_v &= S_{1v} + B_v\,\widetilde{S} + B\,\widetilde{S}_v,\\
S_{uu} &= S_{1uu} + B_{uu}\,\widetilde{S} + 2\,B_u\,\widetilde{S}_u + B\,\widetilde{S}_{uu},\\
S_{uv} &= S_{1uv} + B_{uv}\,\widetilde{S} + B_u\,\widetilde{S}_v + B_v\,\widetilde{S}_u + B\,\widetilde{S}_{uv},\\
S_{vv} &= S_{1vv} + B_{vv}\,\widetilde{S} + 2\,B_v\,\widetilde{S}_v + B\,\widetilde{S}_{vv}.
\end{aligned} \tag{11.12}$$

In figure 11.2 is there to the left a plot of a torus, approximated by 12 surfaces made by Hermite 2-p blending surfaces, in the same way as in the Coons patch example shown in Section 9.6.4. To the right, the surfaces have been moved apart so that we can more easily see them individually. Figure 11.2 can be compared with figure 9.15 as the partial surfaces in both surfaces are made on the same domains.

The method is quite simple to implement, easier than for Coons patch, but the approximation gives a deviation that is slightly larger than for the Coons patch example in Section 9.6.4. In the appendix C.10 there is a theorem that shows that it is possible to fill a square hole with a surface so that the overall result has a desired degree of continuity, which thus means that Hermite 2-p blending surfaces can replace "Coons patch".

Chapter 12

Tensor Product Blending spline Surface

Similar to blending spline curves, tensor product blending spline surfaces are 2^{nd} order tensor product B-spline surfaces with B-functions, but where the control points are replaced with local surfaces. We have the following general formula,

$$S(u,v) = \sum_{i=0}^{n_u-1} \sum_{j=0}^{n_v-1} s_{ij}(u,v) B \circ w_{1,i}(u) B \circ w_{1,j}(v), \tag{12.1}$$

where $s_{ij}(u,v),\ i = 0,..,n_u-1,\ j = 0,..n_v-1,$ are $n_u \times n_v$ local surfaces, and $\{B \circ w_{1,i}(u)\}_{i=0}^{n_u-1}$, $\{B \circ w_{1,j}(v)\}_{j=0}^{n_v-1}$ are the respective B-spline basis functions with B-functions, see (8.10). The formula resembles the ordinary polynomial B-spline tensor product surface, except that $s_{ij}(u,v)$ are surfaces, not points. A blending spline tensor product surface can therefor be regarded as a blending of "local" surfaces. An open blending spline surface with $n_u \times n_v$ local surfaces can thus be divided into $(n_u-1) \times (n_v-1)$ "quadrilateral" patches, each of which is a blending of parts of 4 local surfaces. These are the 4 surfaces connected to their respective interpolation points that are in each of the four corners of the patch.

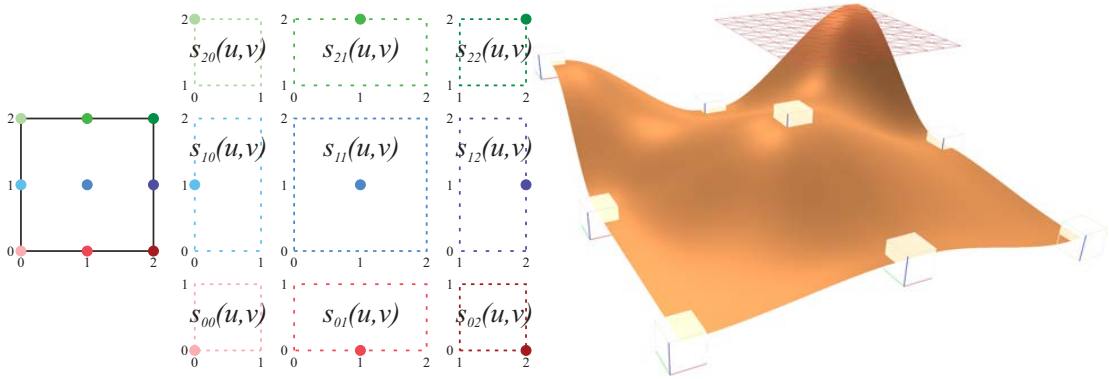


Figure 12.1: *A tensor product Blending spline surface with 9 local surfaces (interpolation points - colored in parameter domain, cubes in $\mathbb{R}^3$). To the right we see the domain of the local surfaces. All local surfaces are flat, one of them is shown on the left side.*

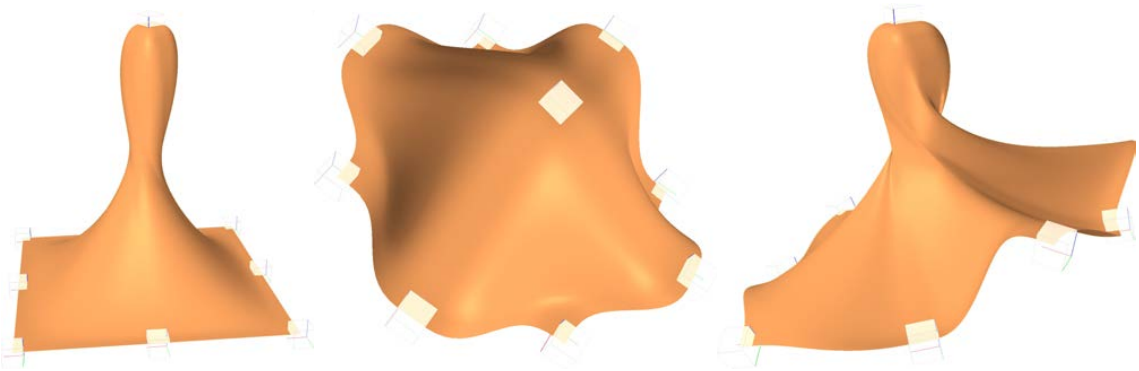


Figure 12.2: *Three tensor product blending spline surface with the same domain, knot vectors and local surfaces as the surface in Figure 12.1. The only difference is that the local surfaces are moved and/or rotated.*

An example is given in Figure 12.1. On the left side we see the parameter domain of a tensor product blending spline surface with $3 \times 3 = 9$ local surfaces. The surface is a 2^{nd} order B-spline surface with two clamped knot vectors, u and v. The domain of each local surface $s_{i,j}$ is shown in the middle of Figure 12.1. The interpolation points, the points where the blending spline surface (later also called the global surface) completely interpolates (including derivatives up to the order of the B-function used) the local surfaces, are marked as cubes. All local surfaces in this example are planar and parallel to the xy-plane, and on the right hand side in Figure 12.1, one of the local surfaces is shown. The knot vector $\mathbf{u}$ is $\{u_i\}_{i=0}^{4} = \{0,0,1,2,2\}$ and the knot vector $\mathbf{v}$ is $\{v_i\}_{i=0}^{4} = \{0,0,1,2,2\}$. As can be seen on the left in Figure 12.1, the local surfaces at the corners are only covering $\frac{1}{4}$ of the global surface, and for this example, the local surfaces connected to the points on the sides are covering half of the global surface, and the surface connected to the point in the middle of the global surface is covering the whole global surface. In Figure 12.2, there are three surfaces with the same domain, knot vectors and local surfaces as the surface in Figure 12.1. The only difference is that the local surfaces are moved and/or rotated. Apparently we now have a spline surface with control points that interpolate the surface and which also have an orientation, not just position.

As we see in the example, the domain is $[u_1,u_3] \times [v_1,v_3]$. This is in accordance with B-splines, see section 6.2.2, where the domain for an open B-spline curve is $[t_d,t_n]$, where $d = k-1$ is the polynomial degree and n is the number of control points. In the example is $k-1 = 1$ and $n = 3$. In Section 8.2, the local curves $c_i(t)$ in blending spline curves are defined with domains (t_i,t_{i+2}). It follows that for surfaces, the domain of the local surfaces $s_{i,j}(u,v)$ will be $(u_i,u_{i+2}) \times (v_j,v_{j+2})$. This can be seen in the example in the middle of Figure 12.1.

12.1 Implementation of Blending spline Surfaces

A tensor product blending spline surface is a 2^{nd}-order B-spline surface in both u and v direction, and is, like ordinary tensor product B-spline surfaces, to be considered as a curve (se Section 8.2) where the coefficients are curves, as described in Section 9.5.3. But

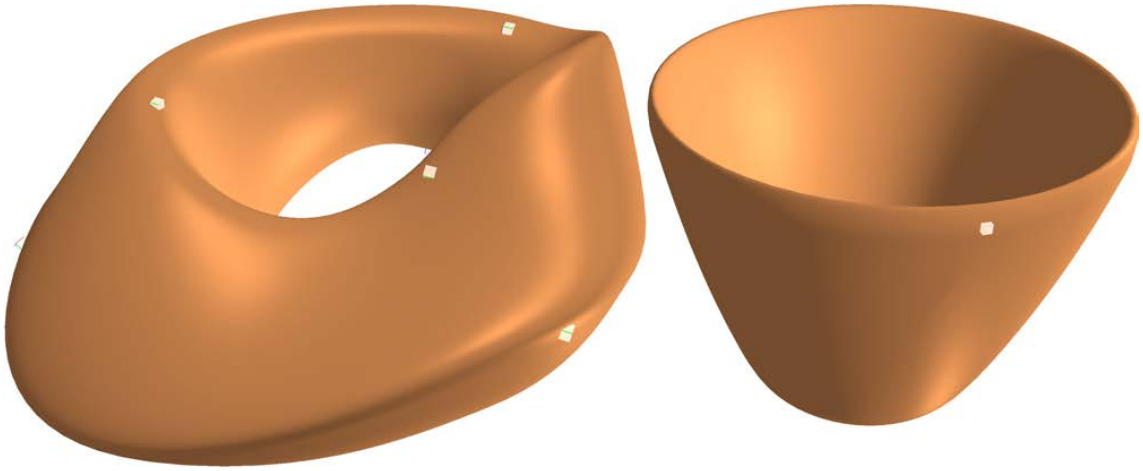


Figure 12.3: *On the left side we see a tensor product blending spline surface which initially is a copy of a torus. The surface has* $3 \times 3 = 9$ *local surfaces where some of them have been moved and rotated. To the right there is also a copy of a torus, but here with* $1 \times 2 = 2$ *local surfaces, the two local surfaces have just been moved apart.*

here the inner curves has coefficients that are surfaces.

Recall from Section 6.2.2 that B-spline curves and surfaces can be either open or closed in each of the parameter directions. Normally we mean clamped when we say open. Remember that for a 2^{nd}-order B-spline is $k = 2$ and analogous to the polynomial degree is $k - 1 = 1$. Below is a table where the domain for open, half-open and closed surfaces are shown. By half-open we mean open in one direction and closed in the other. On the right side of the table, the number of knot intervals, ie "quadrilateral" patches, is shown.

u - direction	v - direction	The parameter domain	Number of patches
Open	Open	$(u,v) \in [u_1, u_{n_u}] \times [v_1, v_{n_v}]$	$(n_u - 1) \times (n_v - 1)$
Open	Closed	$(u,v) \in [u_1, u_{n_u}] \times [v_1, v_{n_v+1})$	$(n_u - 1) \times n_v$
Closed	Open	$(u,v) \in [u_1, u_{n_u+1}) \times [v_1, v_{n_v}]$	$n_u \times (n_v - 1)$
Closed	Closed	$(u,v) \in [u_1, u_{n_u+1}) \times [v_1, v_{n_v+1})$	$n_u \times n_v$

Remember that a closed knot vector is a "cyclic" knot vector, it follows that the continuity is the same over the "closing edge" as over any other knot value. If we use the u-parameter as example, it follows that

$$\begin{aligned} \lim_{u \to u_{n_u+1}-} S_u^{(j)}(u,v) &= S_u^{(j)}(u_1,v), \quad \text{for} \quad j = 0,1,2,...,\mathscr{S}_u, \\ \lim_{u \to u_{n_u+1}-} S_v^{(j)}(u,v) &= S_v^{(j)}(u_1,v), \quad \text{for} \quad j = 0,1,2,...,\mathscr{S}_v, \end{aligned} \quad \text{and} \quad v \in [v_1, v_{n_v}],$$

where $\mathscr{S}_u$ and $\mathscr{S}_v$ are the order of the respective B-function used.

In Figure 12.3 we see two blending spline surfaces, both of which were originally copies of a torus. They are both closed in both directions. After the creation, the local surfaces were moved and/or rotated and we see the results in the figure. In Figure 12.4, a cylinder and a sphere are initially copied and then modified by moving and rotating local surfaces. These are half open surfaces. Note that a sphere is either implemented topologically

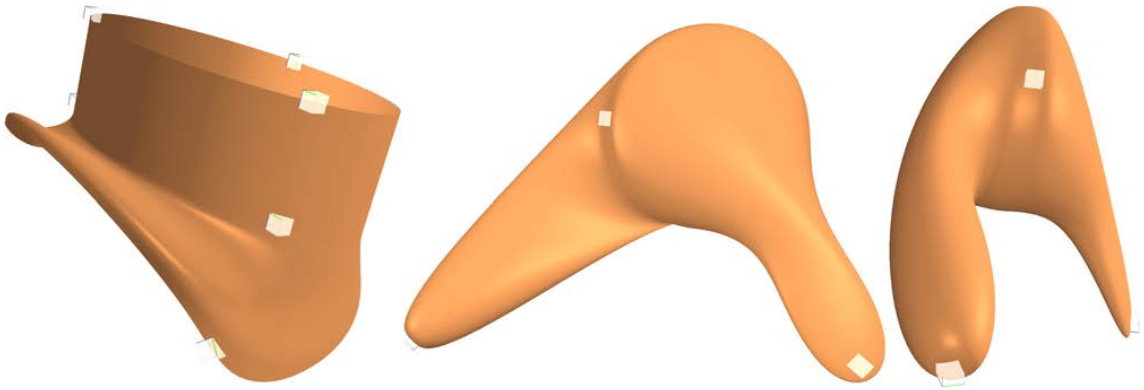


Figure 12.4: *On the left side we see a tensor product blending spline surface that initial is a copy of a cylinder, but where 3 of the 9 local surfaces has been moved and rotated. To the right is there two plots of the same surface, that initially is a copy of a sphere using* $1 \times 3 = 3$ *local surfaces that has been moved and rotated. The implementation we see is actually a sphere that is a modified cylinder, ie. where the two boundaries of the cylinder each are pinched together into two points/poles.*

similar to a cylinder, where the north and south poles are added with hard-coded normals, or it is topologically similar to a plane where the boundaries collapse towards one point. The first is most common, and a typical implementation is as a cylinder with a contraction in both of the "open" ends, and it is, therefore, degenerated at the poles. It is actually possible to introduce "contraction" as a state on its own, to handle degenerated poles, prevent holes and secure a kind of "continuous" unit normal (a single map for a sphere is, as known, not possible without degenerations). "Contraction" is actually not a state for a parameter direction, but a "contraction" of all function values along a curve in the parameter plane and where the directional derivatives in the direction of the derivative of the curve is a zero vector. However, a practical implementations is a straight line restricted to be parallel with one of the coordinate axes in the parameter plane. For a sphere there typically is a "contraction" at the start value and end value of one of the parameters (u or v), while the other parameter is "closed", ie cyclic.

For a "closed" parameter in a blending spline surface, it is basically only necessary to have the same number of knot intervals as the number of local surfaces. That is $n+1$ knot values, where n is the number of local surfaces. However, to make it consistent to "closed" parameter knot vectors where we have two equal knot values at start and end, we put an extra knot value first in the knot vector so that the knot interval at the end is "mirrored" at the beginning of the knot vector.

One question is, what is the minimum number of local surfaces we can have in a parameter direction? The answer is given in the table below.

Type of parameter	Minimum number of local surfaces	size of knot vector
Open	$n = 2$	$n+2$
Closed	$n = 1$	$n+2$

In order to implement / program blending spline surfaces, a number of specific problems

must be solved. this is dealt with in the following sections.

12.2 Evaluation - computing value and derivatives

The general expression for a tensor product blending spline surface was given in (12.1), ie

$$S(u,v) = \sum_{j=0}^{n_v-1} \sum_{i=0}^{n_u-1} s_{i,j}(u,v)\; B\circ w_{1,i}(u)\; B\circ w_{1,j}(v), \tag{12.2}$$

We can simplify. Due to the double sum, and the tensor product standard computation method, the natural choice is first to split the function into an inner part and then apply an outer part, see (9.33). The inner part is, for $u \in [u_i, u_{i+1}]$

$$c_{i,j}(u,v) = \begin{pmatrix} 1-B\circ w_{1,i}(u) & B\circ w_{1,i}(u) \end{pmatrix} \begin{pmatrix} s_{i-1,j}(u,v) \\ s_{i,j}(u,v) \end{pmatrix}, \quad j = 0,1,...,n_v-1. \tag{12.3}$$

It follows that the equation (12.2) can be reformulated by using (12.3). We therefor get a new formulation that replace (12.2), for $(u,v) \in [u_i, u_{i+1}] \times [v_j, v_{j+1}]$ is

$$S(u,v) = \begin{pmatrix} 1-B\circ w_{1,j}(v) & B\circ w_{1,j}(v) \end{pmatrix} \begin{pmatrix} c_{i,j-1}(u,v) \\ c_{i,j}(u,v) \end{pmatrix}. \tag{12.4}$$

Both equations, (12.3) and (12.4) are comparable with the curve formula (8.6) combined with (8.10). Therefore, when developing an algorithm, we can, with some modifications, use the same algorithm as that used for curves.

First, how to compute (12.3):
As in the curve example, we can simplify. From (8.7) and definition 8.1 we get; For every knot interval in the v-direction, $\Delta v_j = [v_j, v_{j+1}),\ j = 0,1,...,n$, where $n = n_v - 2$ for open surfaces and $n = n_v - 1$ for closed surfaces:

$$\begin{aligned} &c_{i,j}(u_i,v) = s_{i-1,j}(u_i,v), && \text{for} \quad i = 1,2,\ldots,n_u \\ &c_{i,j}(u,v) = s_{i-1,j}(u,v) + \widehat{s}_{i,j}(u,v)B_i(u), && \text{when} \quad u_i < u < u_{i+1}, \end{aligned} \tag{12.5}$$

where $B_i(u) = B\circ w_{1,i}(u)$ and where

$$\widehat{s}_{i,j}(u,v) = s_{i,j}(u,v) - s_{i-1,j}(u,v). \tag{12.6}$$

To computing partial derivatives only according to u of (12.5), we see that for $u = u_i,\ i = 1,2,...,n_u$, are all partial derivatives with respect to u equal to the respective derivatives of the local surface, i.e.

$$D_u^{d_u} c_j(u_i,v) = D_u^{d_u} s_{i-1,j}(u_i,v), \quad \text{for } j = 0,...,n_v-1, \text{ and } d_u = 0,1,...,\mathscr{S}_u. \tag{12.7}$$

Then for $u_i < u < u_{i+1}$, we get the following equation for the function value and the two first partial derivatives in the u direction,

$$\begin{aligned} c_{i,j}(u,v) &= s_{i-1,j}(u,v) + \widehat{s}_{i,j}(u,v)B_i(u), \\ D_u c_{i,j}(u,v) &= D_u s_{i-1,j}(u,v) + D_u\widehat{s}_{i,j}(u,v)B_i(u) + \widehat{s}_{i,j}(u,v)DB_i(u), \\ D_u^2 c_{i,j}(u,v) &= D_u^2 s_{i-1,j}(u,v) + D_u^2\widehat{s}_{i,j}(u,v)B_i(u) + 2D_u\widehat{s}_{i,j}(u,v)DB_i(u) + \widehat{s}_{i,j}(u,v)D^2B_i(u). \end{aligned} \tag{12.8}$$

The expressions are essentially the same as those we used in the curve case (8.9). However, this is only a part of the computation in the inner loop. We must also compute all the partial derivatives/mixed derivatives for $c_j(u,v)$ in the v-direction. However, the derivatives in the v-direction are straightforward to compute, because the basis function in (12.8) is independent of v, and therefore, only one of the factors in all terms is dependent on v. The derivatives of the total lines are thus only the derivatives of each term. This can, as we will see, be used to expand formula (12.8) to a vector of vectors instead of only vectors. To expand formula (12.8) to also include derivatives in the v direction we have to first look at the first line of (12.8). It follows that

$$D_v^d c_{i,j}(u,v) = D_v^d s_{i+1,j}(u,v) + D_v^d \widehat{s}_{i,j}(u,v) B_i(u), \quad \text{for } d = 1,2,...,$$

and that in general for computing

$$D_u^{d_u} D_v^{d_v} c_{i,j}(u,v), \quad \text{for } d_u = 1,2,... \text{ and } d_v = 1,2,...,$$

on the right hand side in expression 12.8, we just have to replace

$$\begin{aligned} &D_u^{d_u} s_{i+1,j}(u,v) \quad \text{with} \quad D_u^{d_u} D_v^{d_v} s_{i+1,j}(u,v) \quad \text{and} \\ &D_u^{d_u} \widehat{s}_{i,j}(u,v) \quad \text{with} \quad D_u^{d_u} D_v^{d_v} \widehat{s}_{i,j}(u,v). \end{aligned}$$

Actually, the total result of the inner loop must be a matrix of vectors, where both the matrix and the vectors must have the same dimension as the evaluators from the local surfaces returns, and the matrix must contain the value and all partial derivatives. We, therefore, clarify this by introducing the notation of the inner loop matrix,

$$\mathbf{C}_{i,j,d_u,d_v}(u,v), \tag{12.9}$$

where each element of the matrix is a vector $\in \mathbb{R}^n$, where n usually is 3. The index j is related to the knot interval, and d_u denotes the number of derivatives there are in the u direction, and d_v denotes the number of derivatives there are in the v direction. An example of this matrix, where both $d_u = 2$ and $d_v = 2$, is

$$\mathbf{C}_{i,j,2,2}(u,v) = \begin{bmatrix} c_j(u,v) & D_v c_j(u,v) & D_v^2 c_j(u,v) \\ D_u c_j(u,v) & D_u D_v c_j(u,v) & D_u D_v^2 c_j(u,v) \\ D_u^2 c_j(u,v) & D_u^2 D_v c_j(u,v) & D_u^2 D_v^2 c_j(u,v) \end{bmatrix}. \tag{12.10}$$

The notation of the equivalent matrix from the local patches is

$$\widetilde{\mathbf{S}}_{i,j,d_u,d_v}(u,v), \tag{12.11}$$

where i and j are in the indices of the local patch, and d_u and d_v denote the number of derivatives to compute in the respective u and v directions. These matrices are also organized as (12.10).

To sum up before constructing an algorithm: The first column in (12.9–12.10) is equal to the left hand side of (12.8), and only elements from the first column in (12.11) are used on

the right hand side of (12.8). To compute the other columns we have to replace elements from the first column in (12.11) with respective elements from the other columns. The conclusion is, therefore, that we only have to replace individual elements from the first column of (12.11), that is, on right hand side of (12.8), with the respective rows of (12.11), to expand (12.8) to return (12.9).

Remark 3. *It is, of course, possible and usual, to construct an evaluator which only returns the upper left half of the matrix (12.9). The advantage of a construction like this is to optimize speed, because one often only needs the upper left part of the matrix. The algorithm in this section, however, will focus on computing the whole matrix, but it is fairly simple to modify the algorithm to only compute the upper left part of the matrix.*

The algorithm now becomes quite similar to algorithm 8 used for curves. It depends on the B-function evaluator and evaluators for local surfaces. As for curves, we assume that the surfaces are embedded in an Euclidian space, where the dimension normally is 3, but it could also be something else, so the vector type is, therefore, denoted Vector, although one of them is a point (the upper left ones that is the position). However, the big difference from curves is that the algorithm returns a matrix (of Vectors) instead of a vector (of Vectors), and that the evaluator for local surfaces also returns matrices. We now introduce an algorithm for the inner loop of the tensor product surface evaluator, computing (12.3):

Algorithm 9. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes the matrix $\mathbf{C}_{i,j,d_u,d_v}(u,v)$ *defined in (12.9). Computation of B-function must be present. This also applies to local surfaces where the evaluators must return a matrix* $\widetilde{\mathbf{S}}_{i,j,d_u,d_v}(u,v)$ *defined in (12.11). The knot vectors* $\{u_i\}_{i=0}^{n_u+1}$ *and* $\{v_i\}_{i=0}^{n_v+1}$ *are also supposed to be present. The input variables are:* $u \in [u_1, u_{n_u}]$, $v \in [v_1, v_{n_v}]$, *and* i_u *:\;* $u_{i_u} \le u < u_{i_u+1}$, i_v *:\;* $v_{i_v} \le v < v_{i_v+1}$, *and* $d_u \in \{0,1,2,...,\mathscr{S}_u\}$ *(the number of derivatives in u direction) and* $d_v \in \{0,1,2,...,\mathscr{S}_v\}$ *(the number of derivatives in v direction), where* $\mathscr{S}_u$ *and* $\mathscr{S}_v$ *depends on the B-function. The return is a* "matrix⟨Vector⟩", *ie a set of vectors typically in* $\mathbb{R}^3$, *remember,however, that* $matrix[0][0]$ *is actually a Point.*

matrix⟨Vector⟩ **C**(double u, double v, int i_u, int i_v, int d_u, int d_v)
- matrix⟨Vector⟩ $C_0 = \widetilde{\mathbf{S}}_{i_u,i_v,d_u,d_v}(u,v)$; // *Result evaluating first local surface*
- if $(u == u_{i_u})$ return C_0; // *Return only local surface, see (12.5)*
- Matrix⟨Vector⟩ $C_1 = \widetilde{\mathbf{S}}_{i_u+1,i_v,d_u,d_v}(u,v)$; // *Result evaluating second local surface*
- vector⟨double⟩ $a(d_u+1)$; // *For "Pascals triangle"-numbers*
- vector⟨double⟩ $B = \{B_i^{(j)}\}_{j=0}^{d_u}$; // *Computing B-function, see (8.10).*
- $C_1 -= C_0$; // *The matrix* c_0 *is now* $\widehat{c}_0$, *see (12.6)*
- for (int *i=0;* $i \le d_u$; *i++*)
  - $a_i = 1$;
  - for (int *j=i-1;* $j > 0$; $j--$)
    - $a_j += a_{j-1}$; // *Computing "Pascals triangle"-numbers*
  - for (int *j=0;* $j \le i$; *j++*)
    - $C_{0,i} += (a_j\, B_j) C_{1,i-j}$; // *"row += scalar*row", see (12.8)*
- return C_0;

Note that the algorithm updates entire rows of the matrix C_0 by summing to each row scaled rows of matrix $\widehat{C}_1$. This follows directly from the fact that equation (12.8) is expanded to compute not only the first column in (12.10), but all columns, and that the B-spline (with the B-function) only depend on u.

Next, how to compute (12.4):
This is the final computation of tensor product blending spline surfaces. The equation (12.4) is similar to equation (12.3). Thus, we can simplify the first part of formula (12.4) in the same way as we did for (12.3). It follows that at every knot interval in the u-direction, ie $u \in [u_i, u_{i+1}),\ i = 0, 1, ..., n$, where $n = n_u - 2$ for open surfaces and $n = n_u - 1$ for closed surfaces, we get:

$$\begin{aligned} S(u, v_j) &= c_{i,j-1}(u, v_j), && \text{for} \quad j = 1, 2, \ldots, n_v, \\ S(u, v) &= c_{i,j-1}(u, v) + \widehat{c}_{i,j}(u, v) B_j(v), && \text{when} \quad v_j < v < v_{j+1}. \end{aligned} \tag{12.12}$$

where $B_j(v) = B \circ w_{1,j}(v)$ and where

$$\widehat{c}_{i,j}(u, v) = c_{i,j}(u, v) - c_{i,j-1}(u, v),$$

Computing the partial derivatives of (12.12) only according to v, we can see that for $v = v_j,\ j = 1, ..., n$, all partial derivatives of the spline surface with respect to v are equal to the respective derivatives from the inner loop, i.e.

$$D_v^{d_v} S(u, v) = D_v^{d_v} c_{i,j}(u, v), \qquad \text{for } j = 1, ..., n_u, \text{ and } d_v = 0, 1, 2, ...$$

then for $v_j < v < v_{j+1}$ we get the following equation for the function value and the derivatives in the v direction,

$$\begin{aligned} S(u, v) &= c_{i,j+1}(u, v) + \widehat{c}_{i,j}(u, v) B_i(v) \\ D_v S(u, v) &= D_v c_{i,j+1}(u, v) + \widehat{c}_{i,j}(u, v) D B_j(v) + D_v \widehat{c}_{i,j}(u, v) B_j(v) \\ D_v^2 S(u, v) &= D_v^2 c_{i,j+1}(u, v) + \widehat{c}_{i,j}(u, v) D^2 B_j(v) + 2 D_v \widehat{c}_{i,j}(u, v) D B_j(v) + D_v^2 \widehat{c}_{i,j}(u, v) B_j(v). \end{aligned} \tag{12.13}$$

Studying (12.13) we can see that it has the same structure as (12.8). But as for the inner loop, the total result of the outer loop must be a matrix of vectors, where both the matrix and the vectors must have the same dimension as the algorithm for computing the inner loop returns, and the matrix must contain the value and all partial derivatives. We, therefore, clarify this by introducing the notation of the matrix of the surface evaluator,

$$\mathbf{S}_{d_u, d_v}(u, v), \tag{12.14}$$

where each element of the matrix is a vector $\in \mathbb{R}^n$, where n is the dimension of the Euclidian space the surface is embedded in. The index d_u denotes the number of derivatives there are in u direction, and d_v denotes the number of derivatives there are in v direction. An example of this matrix, where $d_u = 2$ and $d_v = 2$, is:

$$\mathbf{S}_{2,2}(u, v) = \begin{bmatrix} S(u, v) & D_v S(u, v) & D_v^2 S(u, v) \\ D_u S(u, v) & D_u D_v S(u, v) & D_u D_v^2 S(u, v) \\ D_u^2 S(u, v) & D_u^2 D_v S(u, v) & D_u^2 D_v^2 S(u, v) \end{bmatrix}. \tag{12.15}$$

Finally, when preparing for the main algorithm of the tensor product blending spline surface, and thus the outer loop, note that on the left side of (12.13) we have the first *row*, not the first column in the resulting matrix (12.14–12.15). In addition, on the right side of (12.13) we only find elements from the first *row* of the matrix $C_{j,d_u,dv}(u,v)$ (12.9–12.10).

The algorithm becomes quite similar to algorithm 8 used for curves, and algorithm 9 for the inner loop. It depends on the B-function evaluator and on the inner loop. As it was for the inner loop algorithm, we assume that the surfaces are embedded in an Euclidian space, normally $\mathbb{R}^3$, but can also be somewhere else, so the type is, therefore, denoted Vector, usually a 3D-vector. The big difference from the inner loop algorithm is that we, in the penultimate line of the algorithm, have to sum up matrix columns instead of rows.

Algorithm 10. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes the matrix $\mathbf{S}_{d_u,d_v}(u,v)$ *described in (12.14) and (12.15). The algorithm assumes that Algorithm 9 and computation of a B-function are present. The knot vectors* $\{u_i\}_{i=0}^{n_u+1}$ *and* $\{v_i\}_{i=0}^{n_v+1}$ *are supposed to be present. The input variables are:* $u \in [u_1, u_{n_u}]$, $v \in [v_1, v_{n_v}]$, *and* $d_u \in \{0,1,2,...,\mathscr{S}_u\}$ *(the number of derivatives in u direction), and* $d_v \in \{0,1,2,...,\mathscr{S}_v\}$ *(the number of derivatives in v direction), limited by the order of the B-function used. The return is a* "matrix⟨Vector⟩"*, where Vector typically is a 3D-vector matching* $S(u,v)$*, and where the elements in the matrix are matching the elements in the matrix* $C_{i,j,d_u,d_v}(u,v)$ *described in (12.10).*

matrix⟨Vector⟩ **eval (** double *u,* double *v,* int d_u, int d_v **)**

int $i_u = i_u :\backslash$; $u_{i_u} \le u < u_{i_u+1}$;	*// Index for the current knot-interval for u.*
int $i_v = i_v :\backslash$; $v_{i_v} \le v < v_{i_v+1}$;	*// Index for the current knot-interval for v.*
matrix⟨Vector⟩ $S_0 = \mathbf{C}_{i_u,i_v,d_u,d_v}(u,v)$;	*// Result from inner loop - i_v.*
if $(v == v_{i_v})$ return S_0;	*// Return only inner loop, see (12.12).*
matrix⟨Vector⟩ $S_1 = \mathbf{C}_{i_u,i_v+1,d_u,d_v}(u,v)$;	*// Result from inner loop - $i_v + 1$.*
vector⟨double⟩ *a(d+1);*	*// For numbers - "Pascals triangle".*
vector⟨double⟩ $B = \{B_{i_v}^{(j)}\}_{j=0}^{d_v}$;	*// Computing B-function, see (8.10).*
$S_0 -= S_1$;	*// C_0 is now $\widehat{C}_0$, the whole matrix, see (12.7).*
for *(* int *i=0; $i \le d_v$; i++)*	
$a_i = 1$;	
for *(* int *j=i-1; $j > 0$; $j--$)*	
$a_j += a_{j-1}$;	*// Computing "Pascals triangle"-numbers.*
for *(* int *j=0; $j \le i$; j++)*	
$(S_1^T)_i += (a_j\, B_j)(S_0^T)_{i-j}$;	*// "column += scalar×column", (12.13).*
return S_1;	

The computational cost of evaluating a tensor product blending spline-surface is, as we can see, to evaluate two B-functions, four local surfaces, and pass a total of three times through the summing loop in the last half of both the inner and outer loop. The most expensive part of the computation is to evaluate the four local surfaces. This can take more than $\frac{9}{10}$ of the time, depending on the type of local surface.

12.3 Bézier surfaces as local surfaces

In general, Bézier surfaces are very convenient to use as local surfaces. Bézier surfaces are defined in subsection 9.5.2, expression (9.34). It follows from the Bernstein polynomials that the parameter domain for Bézier tensor product surfaces is $[0,1] \times [0,1]$. Blending splines are 2^{nd}-order B-splines. Therefore, the domain of each local surface must span two associated knot intervals in each parameter direction. For a local surface $s_{i,j}(u,v)$, the parameter domain must be $[u_i, u_{i+2}] \times [v_j, v_{j+2}]$. In (8.13) is the map $w_{2,i}$ and its derivative $\delta_{2,i}$ described. For example, $w_{2,i}(u)$ will map $u \in [u_i, u_{i+2}]$ to $\overline{u} \in [0,1]$. Thus we get the following formula for local Bézier surfaces,

$$s_{i,j}(u,v) = \sum_{r=0}^{d_u} \sum_{s=0}^{d_v} c_{r,s} \;\; b_{d_v,s} \circ w_{2,j}(v) \;\; b_{d_u,r} \circ w_{2,i}(u), \tag{12.16}$$

when $(u,v) \in [u_i, u_{i+2}] \times [v_j, v_{j+2}]$, and where the basis functions are the Bernstein polynomials, and $c_{r,s} \in \mathbb{R}^n$, $n > 0$ and usually 3, are the coefficients, ie. control points.

All types of tools for computing Bézier curves are of course also available for Bézier tensor product surfaces. In subsection 4.4.3 the Bernstein/Hermite matrix was introduced, and in Algorithm 2 an algorithm to make the same matrix is developed and described. In the Bernstein/Hermite matrix, each row is scaled by δ^j, where the power j is the row number (starting with 0). This is $\delta_{2,i}$, the derivative of the the translation and scaling function $w_{2,i}$. Thus the matrix looks like this:

$$\mathbf{B}_d(w_{2,i}(t), \delta_{2,i}) = \begin{pmatrix} \delta_{2,i}^0 D^0 b_{d,0} \circ w_{2,i}(t) & \cdots & \delta_{2,i}^0 D^0 b_{d,d} \circ w_{2,i}(t) \\ \vdots & \ddots & \vdots \\ \delta_{2,i}^d D^d b_{d,0} \circ w_{2,i}(t) & \cdots & \delta_{2,i}^d D^d b_{d,d} \circ w_{2,i}(t) \end{pmatrix}. \tag{12.17}$$

In the curve case, the matrix was used both for evaluation (pre-evaluation) and Hermite interpolation. It is therefore natural to do this for tensor product blending spline surfaces.

Hence, using this matrix, (12.17), we first make an expanded matrix version of equation (12.16), not only returning the value, but also all the partial derivatives from 1^{st} to d^{th}, where d is the degree. We now get:

$$\widetilde{\mathbf{S}}_{d_u,d_v}(u,v) = \mathbf{B}_{d_u}(w_{2,i}(u), \delta_{2,i}) \;\; C \;\; \mathbf{B}_{d_v}(w_{2,j}(v), \delta_{2,j})^T \tag{12.18}$$

for $u_i \leq u \leq u_{i+2}$ and $v_j \leq v \leq v_{j+2}$, and where C is the control net (matrix), and $w_{2,i}(u)$, $w_{2,j}(v)$, $\delta_{2,i}$ and $\delta_{2,j}$ are from (8.13), described in subsection 8.2.2. If $d_u = d_v = 2$ the matrix $\widetilde{\mathbf{S}}_{d_u,d_v}(u,v)$ will be,

$$\widetilde{\mathbf{S}}_{2,2}(u,v) = \begin{bmatrix} s(u,v) & D_v s(u,v) & D_v^2 s(u,v) \\ D_u s(u,v) & D_u D_v s(u,v) & D_u D_v^2 s(u,v) \\ D_u^2 s(u,v) & D_u^2 D_v s(u,v) & D_u^2 D_v^2 s(u,v) \end{bmatrix}. \tag{12.19}$$

As one can see, this matrix (12.19) contains the position (upper-left element) and all partial derivatives at the parameter value (u,v) on the surface. It also follows that this matrix completely describes the local surface.

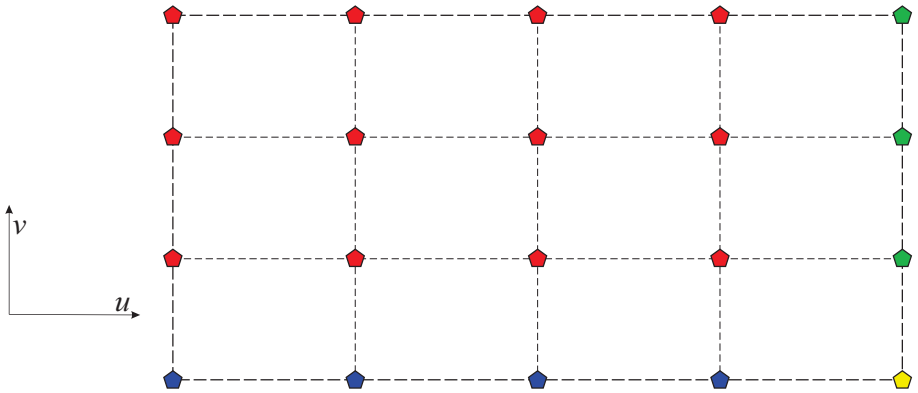


Figure 12.5: *The parameter plane of a surface $g(u,v)$, and two knot vectors $\{u_i\}_{i=0}^{6}$ and $\{v_j\}_{j=0}^{5}$ which divide the domain into 12 pieces. At all internal knot-values, (u_i, v_j) for $i = 1,\ldots,5$, $j = 1,\ldots,4$, there are polygons marked for Hermite interpolation to create local surfaces. The colors of the polygons are for explaining the differences between open and closed surfaces.*

12.3.1 Local Bézier surfaces and Hermite interpolation

We start by recalling the settings from subsection 8.2.3, and adapting them to tensor product blending spline surfaces.

1. Given is a surface $g(u,v)$, $g : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^k$, where $\Omega = [u_s, u_e] \times [v_s, v_e]$, and where $k > 0$ and usually 3.

2. Given is also the numbers of interpolation points $n_u > 1$ and $n_v > 1$ and the number of partial derivatives $d_u > 0$ and $d_v > 0$ to be used in the Hermite interpolation. Note that it is possible to specify the number of partial derivatives to be used for each interpolation point individually.

3. Generate the knot vectors $\mathbf{u} = \{u_i\}_{i=0}^{n_u+1}$ and $\mathbf{v} = \{v_j\}_{j=0}^{n_v+1}$ by:
   – first to set $u_1 = u_s$ and $v_1 = v_s$, i.e. the start of the domain of g in both u and v direction,
   – then to set $u_{n_u} = u_e$ and $v_{n_v} = v_e$, i.e. the end of the domain of g in both u and v direction.
   – Then for $i = 2,3,\ldots,n_u - 1$, generate u_i so that $u_{i-1} < u_i$ (typically uniformly distributed), and for $j = 2,3,\ldots,n_v - 1$, generate v_j so that $v_{j-1} < v_j$.
   – Finally, u_0 and u_{n_u+1} and also v_0 and v_{n_v+1} must be set according to the rules for "open/closed" parameters for 2^{nd}-order tensor product B-spline surfaces.

4. Make the local Bézier surfaces. We do this by Hermite interpolating g in all pairs of (internal) knot values, ie. (u_i, v_j), $i = 1,2,\ldots,n_u$, $j = 1,2,\ldots,n_v$. In Figure 12.5 is the interpolation points marked with colored polygons. If $g(u,v)$ is
   – open in both parameter directions, local surfaces must be created at all points,
   – open in u-direction and closed in the v-direction, local surfaces must be created at all red and green points while the blue and yellow must reuse the the local surfaces made on the top line,

– closed in u-direction and open in the v-direction, local surfaces must be created at all red and blue points while the green and yellow must reuse the the local surfaces made on the line on the left side,
– closed in in both parameter directions, local surfaces must be created at all red points while the blue must reuse the the local surfaces made on the top line, the green must reuse the the local surfaces made on the line on the left side, and the yellow must reuse the one on the upper left corner.

Item 4 above tells us to create local surfaces by Hermite interpolation. Recall that for B-functions of sufficiently high order and for $s_u = 0, ..., d_u$ and $s_v = 0, ..., d_v$ is

$$D_u^{s_u} D_v^{s_v} S(u_i, v_j) = \delta_{2,i-1,\bar{u}}^{s_u} \delta_{2,j-1,\bar{v}}^{s_v} \, D_u^{s_u} D_v^{s_v} s_{i,j} \circ (w_{2,i-1}(u_i), w_{2,j-1}(v_j)),$$

where $S(u,v)$ is the tensor product blending spline surface, $s_{i,j}(u,v)$ are for all i, j local Bézier surfaces, and

$$w_{2,i-1}(u_i) = \frac{u_i - u_{i-1}}{u_{i+1} - u_{i-1}} \quad \text{and} \quad w_{2,j-1}(v_j) = \frac{v_j - v_{j-1}}{v_{j+1} - v_{j-1}},$$

are the linear translation and scaling function from definition 6.11, and

$$\delta_{2,i-1,\bar{u}} = \frac{1}{u_{i+1} - u_{i-1}} \quad \text{and} \quad \delta_{2,j-1,\bar{v}} = \frac{1}{v_{j+1} - v_{j-1}}$$

are the derivative of the linear translation and scaling function, see (6.13).

The equation for the Hermite interpolations for a local Bézier surface with the pair of indices (i,j) for $s_u = 0, ..., d_u$ and $s_v = 0, ..., d_v$ is,

$$\begin{aligned} D_u^{s_u} D_v^{s_v} g(u_i, v_j) &= D_u^{s_u} D_v^{s_v} S(u_i, v_j) \\ &= \delta_{2,i-1,\bar{u}}^{s_u} \delta_{2,j-1,\bar{v}}^{s_v} \, D_u^{s_u} D_v^{s_v} s_{i,j} \circ (w_{2,i-1}(u_i), w_{2,j-1}(v_j)) \\ &= \sum_{r=0}^{d_1} \sum_{s=0}^{d_2} c_{i,j,r,s} \; \delta_{2,i-1,\bar{u}}^{s_u} \, D^{s_u} b_{d_u,r} \circ w_{2,i-1}(u_i) \delta_{2,j-1,\bar{v}}^{s_v} \, D^{s_v} b_{d_v,s} \circ w_{2,j-1}(v_j). \end{aligned}$$

This is nearly the same as the formulation in (12.18), but we now have included the parameter mapping. The matrix form now is, for $i = 1, ..., m_u$, and $j = 1, ..., m_v$,

$$\mathbf{g}_{d_u,d_v}(u_i, v_j) = \mathbf{B}_{d_u}(w_{2,i-1}(u_i), \delta_{2,i-1,\bar{u}}) \; C_{i,j} \; \mathbf{B}_{d_v}(w_{2,j-1}(v_j), \delta_{2,j-1,\bar{v}})^T,$$

where

$$\mathbf{g}_{d_u,d_v}(u_i, v_j) = \begin{pmatrix} g(u_i, v_j) & \dots & D_v^{d_v} g(u_i, v_j) \\ \vdots & \ddots & \vdots \\ D_u^{d_u} g(u_i, v_j) & \cdots & D_u^{d_u} D_v^{d_v} g(u_i, v_j) \end{pmatrix}.$$

Bézier-surface and Hermite interpolation

The final step in generating the local Bézier surfaces is to turn the equation according to the Bézier coefficients $\mathbf{C}_{i,j}$, the control polygon of the local surface $S_{i,j}(u,v)$, i.e.

$$C_{i,j} = \mathbf{B}_{d_u}(w_{2,i-1}(u_i), \delta_{2,i-1,\bar{u}})^{-1} \; \mathbf{g}_{d_u,d_v}(u_i, v_j) \; \mathbf{B}_{d_v}(w_{2,j-1}(v_j), \delta_{2,j-1,\bar{v}})^{-T}. \quad (12.20)$$

The conclusion is that, in order to compute the coefficient to the local Bézier surfaces, ie solve (12.20), one has to compute the expanded Bernstein/Hermite matrix using algorithm 2, and then invert this matrix and multiply the inverted matrix with the "evaluation"-matrix from the original surface. The matrix inversion will not be dealt with further here, but there are a lot of available programming libraries including optimized algorithms for matrix inversions, see, e.g., [165].

Remark 4. *Note that the three matrices on the right hand side in 12.20 are not of the same "type". The middle one, $\mathbf{g}_{d_u,d_v}(u_i,v_j)$ is a matrix of point/vectors in $\mathbb{R}^n$, where n usually is 3. The Bernstein/Hermite matrix is a standard matrix where each element is a scalar. For both a surface evaluator and the Hermite interpolation, it is ,therefore, of great interest to implement a matrix template type that has overloaded matrix multiplication including multiplication between a matrix of scalars and a matrix of vectors/points.*

As in the curve case, there are several reasons why it is advantageous to translate all coefficients so that the interpolation point is the local origin. Then it follows that we have to subtract the point $g(u_i,v_j)$ from all the coefficients in the control polygon $C_{i,j}$ of the local Bézier surfaces, and that we have to cancel this by inserting the opposite movement to the graphical homogeneous matrix system. The premise is, of course, that this homogeneous matrix system is involved in the total evaluator.

12.3.2 Examples of Hermite interpolations

In this subsection we will look at examples of Hermite interpolation of well-known parametric surfaces by tensor product Blending spline surfaces with local Bézier surfaces. In some of the examples we will see some of the local Bézier surfaces, and take a closer look at how they are constructed by the Hermite interpolation.

The first example is based on a surface called "Trianguloid Trefoil". This surface was constructed by Roger Bagula and can be found at [4]. The formula is

$$s(u,v)=\begin{pmatrix} 2\frac{\sin(3u)}{2+\cos v} \\ 2\frac{\sin u+2\sin(2u)}{2+\cos\left(v+\frac{2}{3}\pi\right)} \\ \frac{(\cos u-2\cos(2u))(2+\cos v)\left(2+\cos\left(v+\frac{2}{3}\pi\right)\right)}{4} \end{pmatrix} \quad \begin{matrix} \text{for} \quad u\in(-\pi,\pi], \\ \text{and} \quad v\in(-\pi,\pi]. \end{matrix} \tag{12.21}$$

In Figure 12.6 there is a plot of a tensor product blending spline surface interpolating a "Trianguloid Trefoil" surface (12.21) in 5×5 points. At each point the position and 8 partial derivatives are used, i.e. the matrix $\mathbf{g}_{d_u,d_v}(u_i,v_j)$ defined in (12.17) is 3×3 dimensional. In Figure 12.6 the interpolation points are marked with blue cubes, and most of them are visible. The surface is "closed" in both parameter directions, and is quite complex in shape, but the reconstruction has kept the structure and form fairly well.

The second example is based on a surface called "Bent Horns", also constructed by Roger

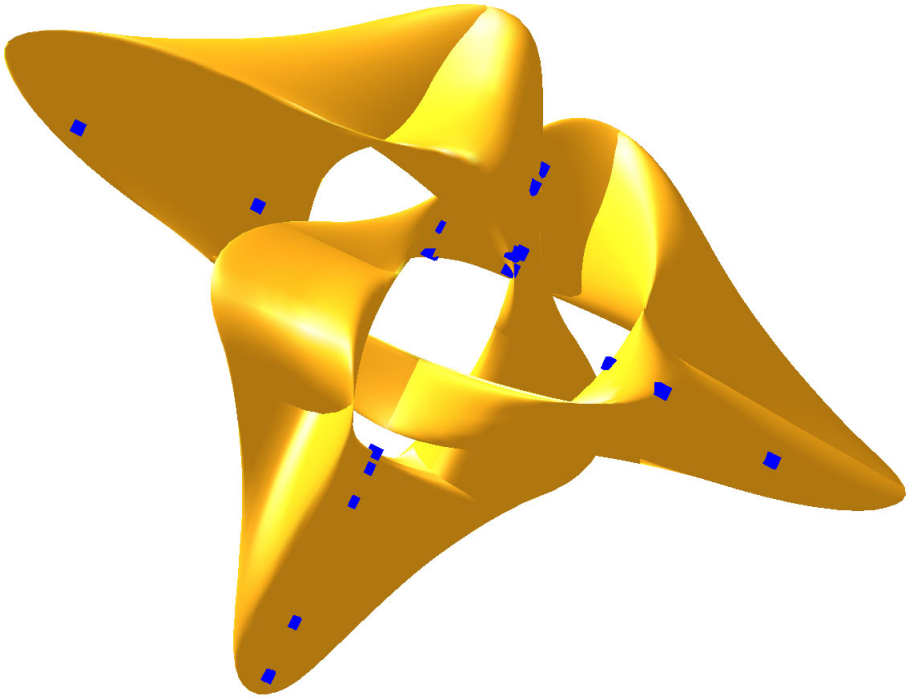


Figure 12.6: *This Blending spline tensor product surface is made by Hermite interpolation of a "Trianguloid Trefoil" surface [4] at* 5×5 *points. The interpolating points are seen as blue cubes.*

Bagula and can be found at [5]. The formula for this surface is

$$s(u,v) = \begin{pmatrix} (2+\cos u)\left(\frac{v}{3} - \sin v\right) \\ \left(2+\cos\left(u-\frac{2}{3}\pi\right)\right)(\cos v - 1) \\ \left(2+\cos\left(u+\frac{2}{3}\pi\right)\right)(\cos v - 1) \end{pmatrix} \quad \begin{array}{l} \text{for} \quad u \in (-\pi, \pi], \\ \text{and} \quad v \in (-2\pi, 2\pi]. \end{array} \tag{12.22}$$

In Figure 12.7 there is a plot of a tensor product blending spline surface interpolating a 'Bent Horns" surface (12.22) at 5×5 position. Also in this example, the position and 8 partial derivatives are used at each point, i.e. the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (12.17) has the dimension 3×3. In the Figure 12.7 the interpolation points are marked as blue cubes, and some of them can be seen clearly. The surface is "closed" in one parameter, but "open" in the other parameter directions. This cannot actually be seen, because the two "open" ends are squeezed into two separate edges. They can be seen at the front on either side in the figure, and each of them are marked by three blue cubes. These cubes are, in fact 5 cubes (one on the tip towards the center, and two coincident cubes on each of the other two). The surface is also irregular in the center, where it collapses to a point. The blending spline surface has managed to keep this irregularity intact because there is an interpolation point (actually, 5 points) at this position.

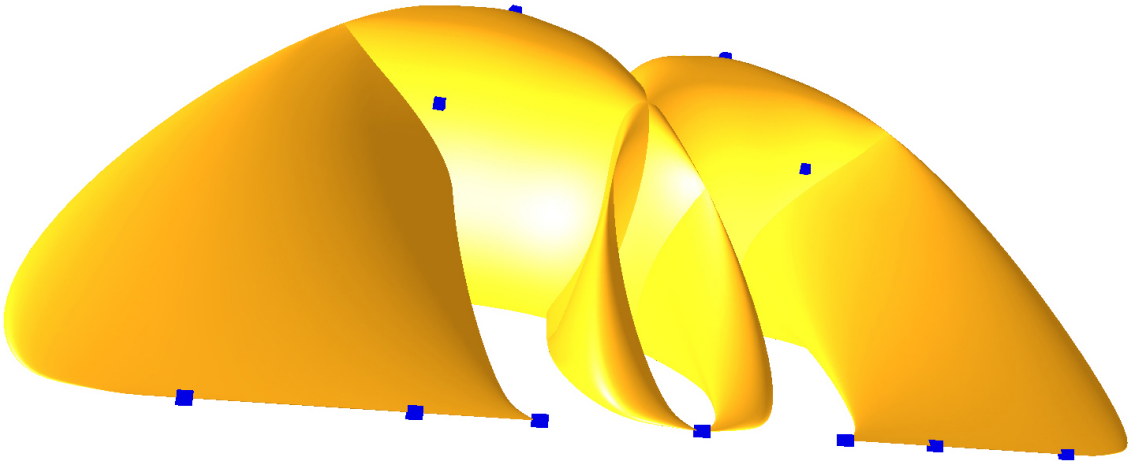


Figure 12.7: *This Blending spline tensor product surface is made by interpolating (position, first and second derivative) a "Bent Horns" surface [5] at* 5×5 *points. The positions of the interpolating points are seen as cubes.*

The third example is based on a sphere, where the formula is,

$$s(u,v) = \begin{pmatrix} r\cos u\ \cos v \\ r\sin u\ \cos v \\ r\sin v \end{pmatrix} \quad \begin{array}{ll} \text{for} & u \in (0, 2\pi], \\ \text{and} & v \in (-\frac{\pi}{2}, \frac{\pi}{2}]. \end{array} \tag{12.23}$$

In the equation, r is the radius of the sphere. Figure 12.8 shows 3 plots of a tensor product blending spline surface interpolating a sphere (12.23) at 4×4 points. Also in this example, the position and 8 partial derivatives are used at each point, i.e., the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (12.17) has the dimension 3×3. In Figure 12.8 the interpolation points are marked as blue cubes. At the top of the "sphere" we can see one cube, but there are actually 4 cubes in the same position. The surface is, therefore, irregular, and actually collapses to a point at both poles. In the upper part of Figure 12.8 there is plots that includes one of the local Bézier surfaces located at the "north pole". There is one shaded picture on the left hand side, and one wireframe picture on the right hand side. As we can see, especially in the wireframe picture, the local surface only has two corners. The other two corners have collapsed to one point, and are lying on the apparently "smooth" edge at the "north pole". Also the edge between the two corners has completely collapsed to the same point. The tensor product blending spline surface has 8 local Bézier surfaces, which can be found at the two poles. They are all equal in form compared to the Bézier surface shown in the figure (but rotated and/or translated). On the lower part of Figure 12.8 there is another plot of the approximated "sphere", this time including one of the local Bézier surfaces that is not located at the poles. The figure is rotated to the left, so that the "north pole" is on the left hand side. The local surface looks quite complex, and probably unlike what can be expected. There are a total of 16 local Bézier surfaces attached to the tensor product blending spline surface, and 8 of these local surfaces have the same shape as this one.

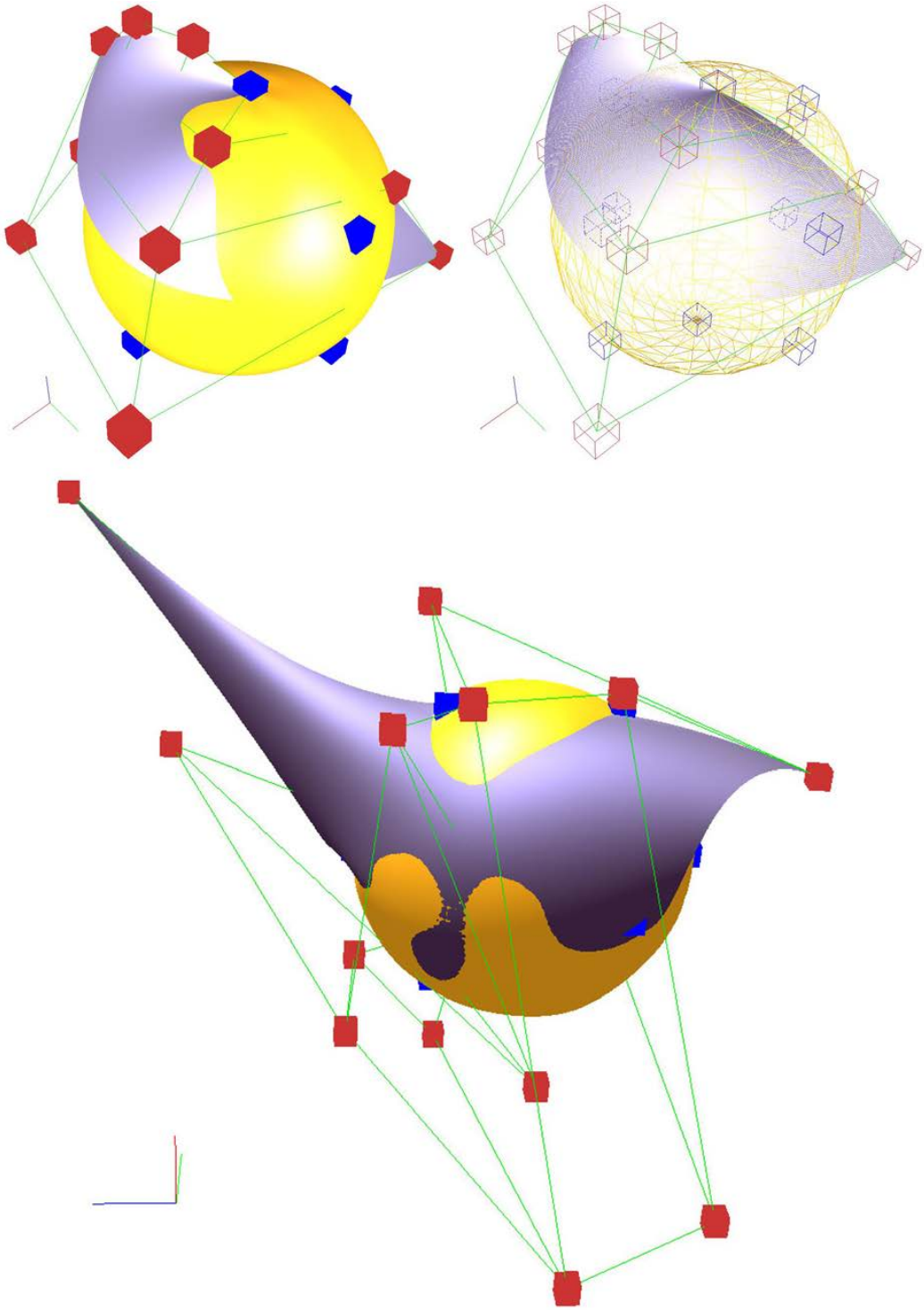


Figure 12.8: *Three plots of the approximated sphere from expression (12.23), The upper part with one of the local Bézier surfaces located at the "north pole". The control polygon of the Bézier surface is marked in green, and the nodes in the control polygon are marked as red cubes. On the right hand side there is a wireframe version of the figures that we can see on the left hand side. The lower part shows another local Bézier surface. This is one of the surfaces not located at the poles. All 8 local surfaces that is not located at north and south pole will have this shape, but translated and rotated.*

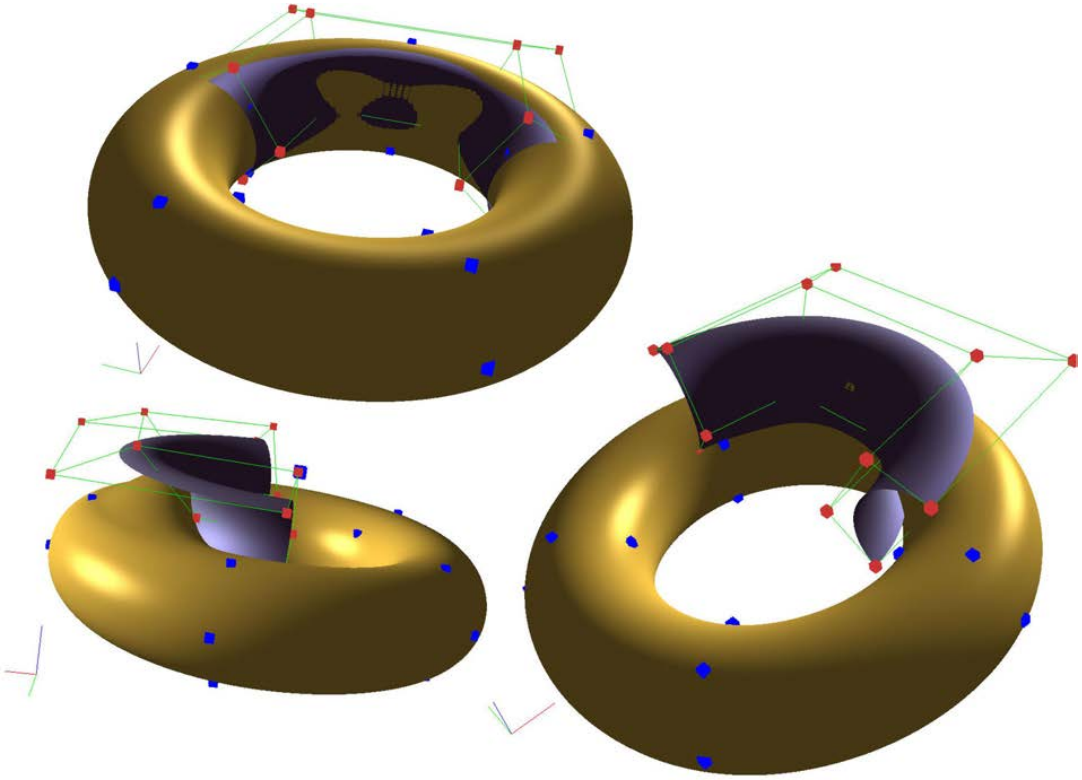


Figure 12.9: *A plot of the approximated torus from (12.24), where one of the local Bézier surfaces is also plotted. Also the control polygon of the local Bézier surface is plotted in green, while the control points are plotted as red cubes. At the bottom of the figure the local Bézier surface is moved slightly upwards.*

The fourth example is based on a Torus, where the equation is

$$s(u,v) = \begin{pmatrix} \cos u(R + r\cos v) \\ \sin u(R + r\cos v) \\ r\sin v \end{pmatrix} \quad \begin{array}{l} \text{for} \quad u \in (0, 2\pi], \\ \text{and} \quad v \in (0, 2\pi]. \end{array} \tag{12.24}$$

Here r is the small radius, the radius of the tube, and R is the big radius, the distance from the center of the tube to the center of the torus. The upper part of Figure 12.9 shows a tensor product blending spline surface interpolating a torus (12.24) at 5×5 points. As in the previous examples, the position and 8 partial derivatives are used at each point, ie the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (12.17) has dimension 3×3. In Figure 12.9 the interpolation points are marked as blue cubes. One of the local Bézier surfaces is also plotted. This surface models a part of a torus quite well. At the bottom of the figure the local Bézier surface is moved slightly upwards, so we can see it more clearly. There the surfaces are plotted in two different views, so it is possible for us to see the local surface better. As we can see on the left hand side, the blending spline surface is actually following the local surface when it is moved, and thus changing shape of the torus.

The last example shows us a surface called “Sea Shell”, described amongst other by Paul

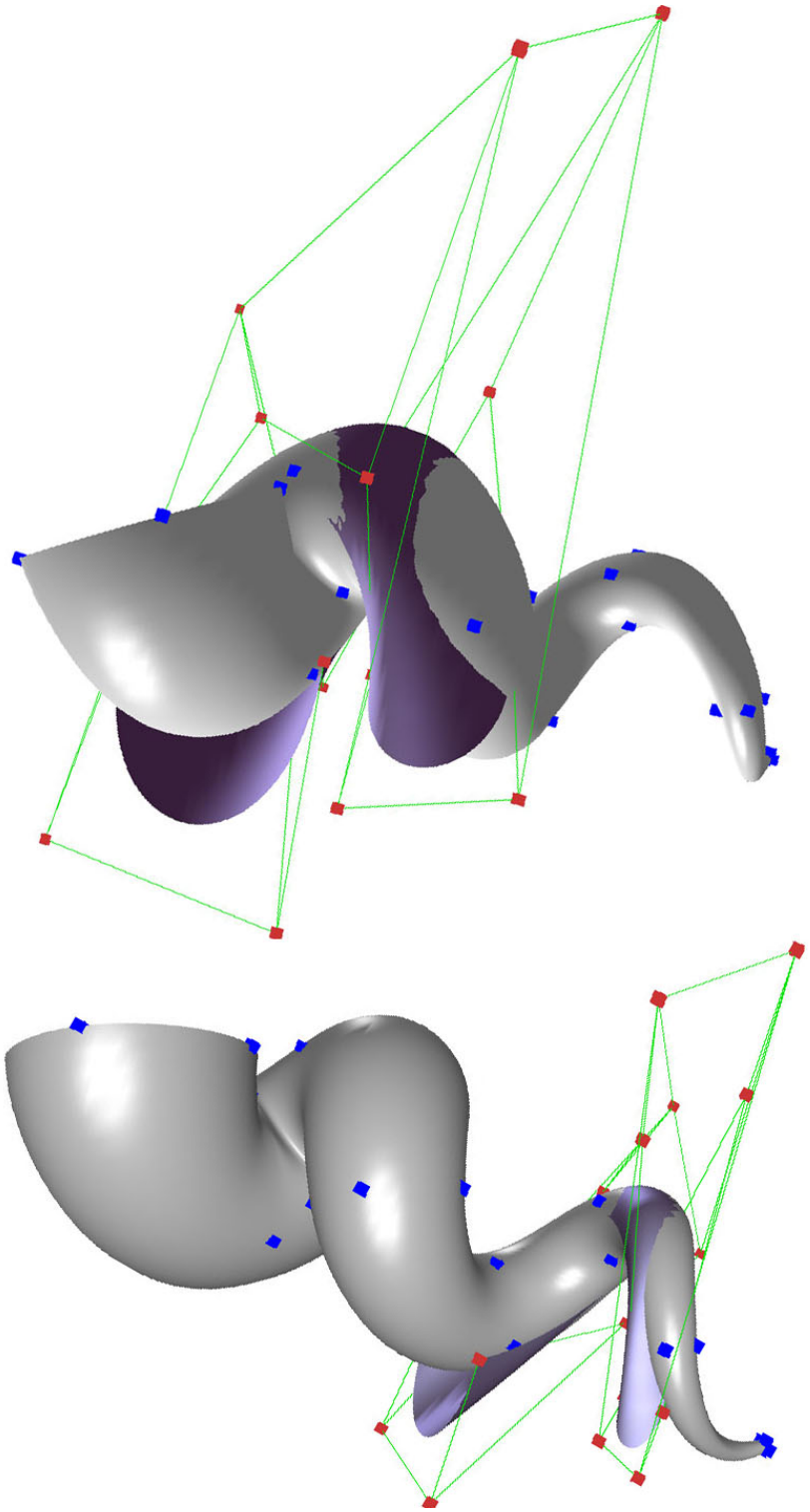


Figure 12.10: *Two plots of a tensor product blending Spline surface made by interpolating a "Sea Shell" surface expressed in (12.25). The two plots are seen from different angles. In each of the two plots one of the local Bézier surfaces is also plotted. In the upper plot one of the bigger local patches is plotted. In the lower one a smaller patch is plotted. We can also see the control polygon for each of the local Bézier surfaces. The local patches model the global surface quite well locally.*

Bourke at [16]. The formula for this surface is

$$s(u,v) = \begin{pmatrix} \cos v + \frac{v}{10}\left(\cos u \cos v + a \cos u \sin v\right) \\ \sin v + \frac{v}{10}\left(\cos u \sin v - a \cos u \cos v\right) \\ \left(b \sin u + \frac{6}{10}\right) v \end{pmatrix} \quad \begin{array}{l} \text{for} \quad u \in (0, 2\pi], \\ \text{and} \quad v \in (\frac{\pi}{4}, 5\pi]. \end{array} \tag{12.25}$$

In Figure 12.10 there is a plot of a tensor product blending spline surface interpolating a "Sea Shell" surface (12.25) at 4×8 points. At each point the position and 8 partial derivatives are used, ie the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (12.17) has the dimension 3×3. The interpolation points are marked as blue cubes, and most of them are visible. As can be seen, the surface is "closed" in one parameter direction, but "open" in the other parameter direction. In the figure, the surface can be seen from two different angles. In both plots one of the local Bézier surfaces is also plotted, and it can be seen colored in purple. The control polygons of the local Bézier surfaces are also plotted. They can be seen as a green net, and their control points are drawn as red cubes.

12.4 The sub-surface construction

As in the curve case, we can convert any parametric surface $\varphi(u,v)$ to a blending spline surface by adding two vectors of knot values, $\{u_i\}_{i=0}^{n_u+1}$ and $\{v_j\}_{j=0}^{n_v+1}$. With that we get a set of overlapping sub-surfaces, each of which is the original surface only limited by a reduction of parametric domain that covers 2×2 knot intervals. Using sub-surfaces as local surfaces means that a blending spline copy is initially identical to the surface itself. This is because blending of a surfaces with itself gives the surface itself. When adding affine transformations (subsection 8.2.1), a sub-surface algorithm will basically be similar to the algorithm for local Bézier surfaces. Similar to section 12.2, and expression (12.3), we get for $i = 1, 1, \ldots, n_u$ and $j = 1, 1, \ldots, n_v$ and $(u,v) \in [u_i, u_{i+1}] \times [v_j, v_{j+1}]$

$$\begin{aligned} C_{i,j}(u,v) &= \begin{pmatrix} 1 - B_i(u) & B_i(u) \end{pmatrix} \begin{pmatrix} A_{i-1,j}\ \varphi(u,v) \\ A_{i,j}\ \varphi(u,v) \end{pmatrix} \\ &= \left(A_{i-1,j} + B_i(u)\ \Delta^i A_{i-1,j}\right) \varphi(u,v), \end{aligned} \tag{12.26}$$

where $B_i(u) = B \circ w_{1,i}(u)$, $\varphi(u,v)$ is the initial surface, $A_{i,j}$ are homogeneous matrices described in subsection 8.2.1 and $\Delta^i A_{i-1,j} = A_{i,j} - A_{i-1,j}$. Next step is to reformulate (12.4),

$$\begin{aligned} S(u,v) &= \begin{pmatrix} 1 - B_j(v) & B_j(v) \end{pmatrix} \begin{pmatrix} C_{i,j-1}(u,v) \\ C_{i,j}(u,v) \end{pmatrix} \\ &= C_{i,j-1}(u,v) + B_j(v)\ \Delta^j C_{i,j-1}(u,v) \end{aligned} \tag{12.27}$$

where $\Delta^j C_{i,j-1}(u,v) = C_{i,j}(u,v) - C_{i,j-1}(u,v)$. If we put (12.26) in (12.27) we get:

The sub-surface construction

$$S(u,v) = \mathbb{A}_{i-1,j-1}(u,v)\ \ \varphi(u,v) \tag{12.28}$$

where

$$\mathbb{A}_{i,j}(u,v) = A_{i,j} + B_{i-1}(u)\ \Delta^i A_{i,j} + B_{j-1}(v)\ \left(\Delta^j A_{i,j} + B_{i-1}(u)\ \Delta^{ij} A_{i,j}\right) \tag{12.29}$$

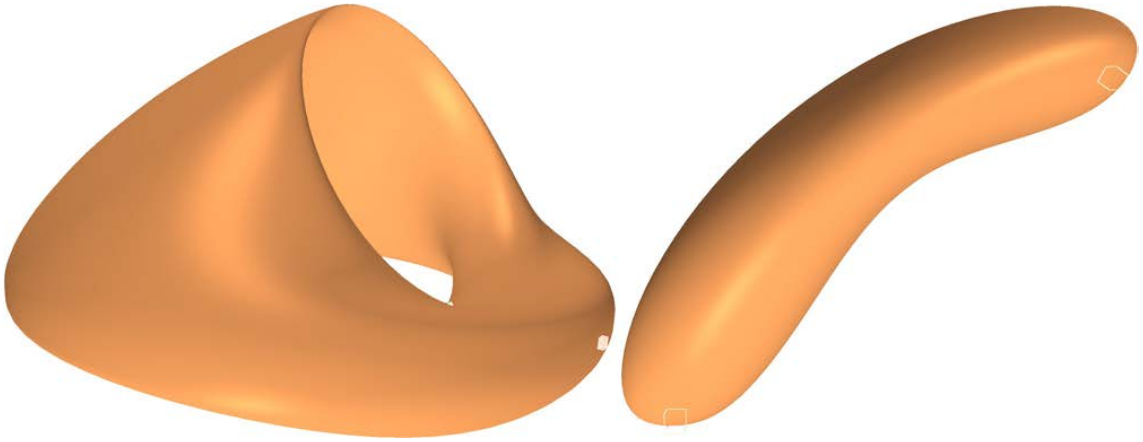


Figure 12.11: *Two examples of blending spline surfaces from the sub-surface construction. On the left side a torus is copied using only 2 local surfaces, the local surfaces associated with the interpolation point inside the torus is rotated* 60°*. To the right of the figure, a sphere is copied. Also here with only using 2 local surfaces. These 2 surfaces are then moved apart and rotated slightly.*

The delta notation in (12.29) can be expanded as follows,

$$\begin{aligned}\Delta^i A_{i,j} &= A_{i+1,j} - A_{i,j}\\ \Delta^j A_{i,j} &= A_{i,j+1} - A_{i,j}\\ \Delta^{ij} A_{i,j} &= A_{i+1,j+1} - A_{i,j+1} - A_{i+1,j} + A_{i,j}.\end{aligned}$$

All surfaces shown in Figures 12.1, 12.2, 12.3, 12.4 and 12.11 are blending spline surfaces based on sub-surfaces and thus computed by using expression (12.28) and (12.29).

12.5 Examples, free form sculpturing using tensor product blending splines

In Computer Aided Geometric Design, object (surface) sculpturing is a big issue both for constructions for real world (products), and design for virtual worlds (movies, computer games, VR/AR, etc.). As regards sculpturing, Barr introduced as early as in 1984 operations for twisting, stretching, bending and tapering surfaces around a central axis [8]. This was followed by Sederberg and Perry who in 1996 introduced a more general technique [144], called the Free Form deformation method, FFD. This method embeds objects to be deformed in the 3D lattice of control points that define a trivariate Bézier volume. Deformations can thus be done by deforming this 3D control polygon and evaluating the Bézier volume to find the new position for the embedded objects. In the following years a tremendous amount of work has been done in this area, using mechanical technics, multilevel representations etc. (Examples of articles are [27] [9] [84] [167]). In general, all works so far have struggled with the geometric representation. Blending splines offer an improved "shaping" functionality because it is a B-spline where all type of affine maps can be applied to the "control points".

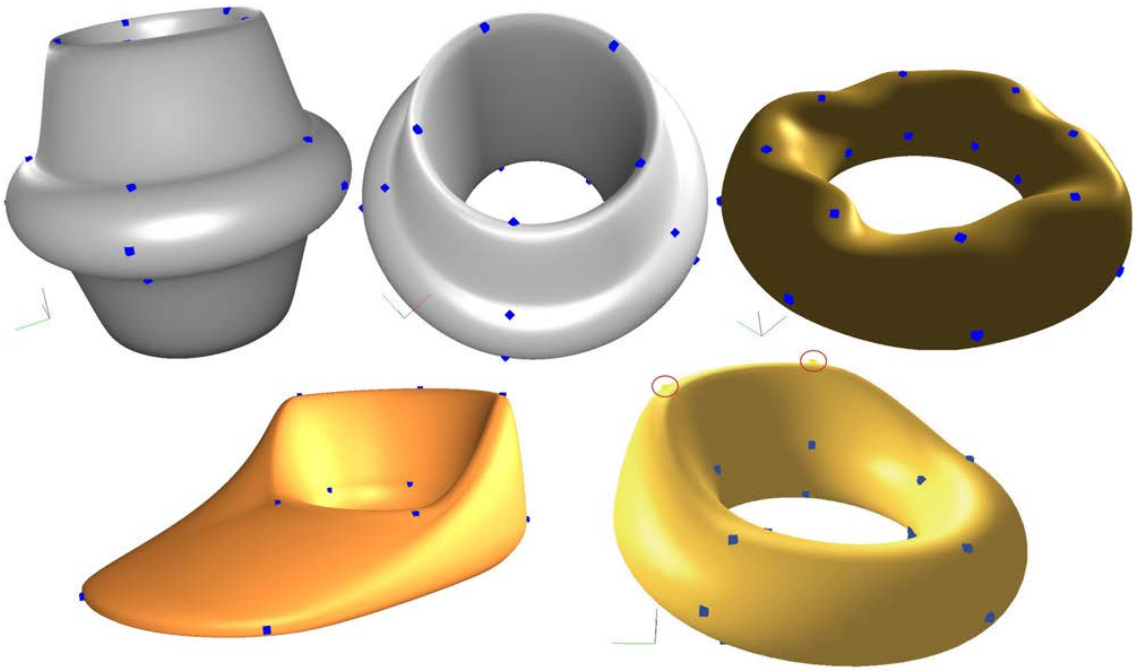


Figure 12.12: *Foure sculpted objects, all of which were initially a torus. They are edited by moving interpolation points, except for the object on the upper right side, where the points are only rotated. All surfaces are tensor product blending splines with local Bézier surfaces. The one on the upper right side has* 8×4 *local surfaces, the other has* 5×5.

In this chapter we have many figures that illustrate the possibilities for creative design. Early in this chapter we find 4 figures where all the examples/plots are based on the sub-surface construction. In Figure 12.1 and 12.2, a blending spline "plain" is deformed in different ways. They all have 3×3 interpolation points. In Figure 12.3, there is one torus with 9 local surfaces and one with 2. Both are deformed, and we can see that the number of interpolation points are important for the shape possibilities. In Figure 12.4 there is a deformed cylinder and two plots of a deformed sphere seen from two different positions.

Figure 12.11 is also based on sub-surfaces, and shows a torus and a sphere, both with only 2 local surfaces. And in Figure 12.12 is a torus changed in four different ways. The upper left one is plotted from 2 different positions. The local surfaces are Bézier surfaces. In Figure 12.13 is a sphere changed to a mug and a head.

12.6 T-junction and Star-junction

In product development and design, solid modeling is mainly used as a tool. Boundary representation, abbreviated B-rep, is a method of representing 3D object. The surface and the boundary between different materials in an object are then a collection of continuous surface elements. Thus, it is important to be able to model complex surfaces, ie to model surfaces of different genus, and with varying degrees of complexity. This means that we must be able to handle "irregular" surface geometry. Traditionally, this has been done mainly with trimmed surfaces with trimming curves that are calculated with Boolean

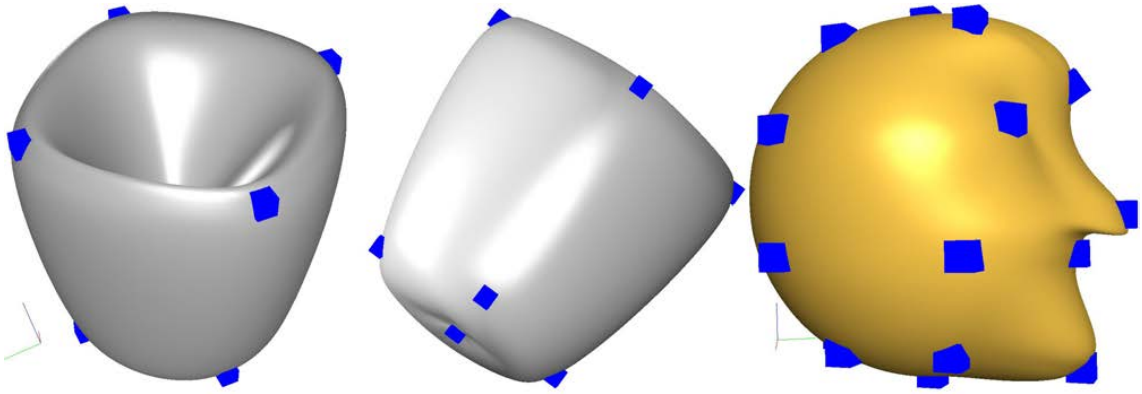


Figure 12.13: *To the left of the figure, a sphere is changed to a mug just by moving the interpolation points. The object is displayed from 2 different positions. To the right of the figure you can see a "head". It is a tensor product blending spline surface made with local Bézier surfaces by first interpolating a sphere at* 6×6 *points (position, first and second derivative). Then some of the interpolation points (five of the blue cubes) are moved, and some of them, three points at the eyes and nose, are also rotated.*

operations and thus surface intersection, see [118] and [90].

An important reason to handle irregularities without using trimming is the introduction of isogeometric analysis that is a replacement of the traditional Finite Element function-spaces with spline-spaces, to provide the same function space (set of basis functions) for both computation and shape description, see [28].

Both, T-splines, [143] and [142], LR-splines, [51], [95] and [127], and PHT-splines [157], are surface descriptions that modify B-splines to handle some kind of irregularities. Therefor, a related solution for dealing with irregularities using tensor product blending spline surfaces was described in [109]. This imply looking at T-junctions and Star-junctions, together with special local surfaces which in turn are divided into sub-surfaces and sometime with re-parameterization. This is to ensure continuity (and smoothness) in the splicing between different tensor product surfaces.

12.6.1 Dependencies on vertices and "internal edges"

The parameter domain of a tensor product blending spline is partitioned by the knot vectors, see Figure 12.14. At the intersection of the parameter lines are the vertices. These are the points where the local surfaces interpolate the global surface. It follows that the support of the local surfaces is the sub-partitions in the domains that is surrounding the related vertex. For all internal vertices is this four squared partitions, for vertices on the edges is it two partitions, and for vertices in the corners is it one partition. All this is clearly illustrated in Figure 12.14, but also outlined in Figure 12.1.

On each partition we have sub-surfaces of the four local surfaces that cover this partition. To simplify, we consider the local domain of each sub-surface as the unit square $[0,1] \times [0,1]$. The formula for a surface with this domain, made by blending parts of four local

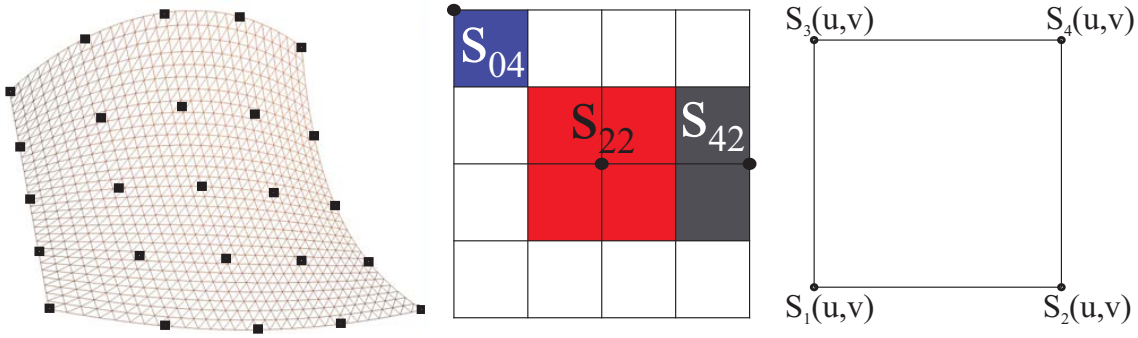


Figure 12.14: *To the left is a tensor product blending spline surface made of a network of 5×5 "local surfaces". In the middle is the parameter plane of the surface where the grid is shown. Three examples of the domain of the local surfaces are illustrated. In a corner, the surface S_{04} is in blue, in the center is S_{22} in red, and on the left edge is S_{42} in black. To the right is one partition shown and the four local surfaces, connected to each corner are marked.*

surfaces, each connected to one of the four corners, right side in Figure 12.14, is:

$$\begin{aligned} S(u,v) &= (1-B(v))((1-B(u))s_1(u,v)+B(u)s_2(u,v)) \\ &\quad +B(v)((1-B(u))s_3(u,v)+B(u)s_4(u,v)), \\ &= s_1(u,v)+B(u)(s_2(u,v)-s_1(u,v))+B(v)(s_3(u,v)-s_1(u,v)) \\ &\quad +B(u)B(v)(s_4(u,v)-s_3(u,v)-s_2(u,v)+s_1(u,v)). \end{aligned}$$

We skip the parameters (u,v) for the surfaces and the simplified expressions is

$$S = s_1 + B(u)(s_2 - s_1) + B(v)(s_3 - s_1) + B(u)B(v)(s_4 - s_3 - s_2 + s_1) \tag{12.30}$$

To investigate the behavior on the edges, we just look at the edge on the left side. The other edges we will have similar behavior. Thus, we get

$$S(0,v) = s_1 + B(v)(s_3 - s_1), \tag{12.31}$$

and the first and second order partial derivatives are

$$\begin{aligned} S_u(0,v) &= s_{1u} + B(v)(s_{3u} - s_{1u}), \\ S_v(0,v) &= s_{1v} + B(v)(s_{3v} - s_{1v}) + B'(v)(s_3 - s_1), \\ S_{uu}(0,v) &= s_{1uu} + B(v)(s_{3uu} - s_{1uu}), \\ S_{uv}(0,v) &= s_{1uv} + B'(v)(s_{3u} - s_{1u}) + B(v)(s_{3uv} - s_{1uv}), \\ S_{vv}(0,v) &= s_{1vv} + 2B'(v)(s_{3v} - s_{1v}) + B(v)(s_{3vv} - s_{1vv}). \end{aligned} \tag{12.32}$$

The following lemma states the interpolation properties on the boundary of the blending surfaces made by blending four surfaces connected to each corners. Thus, the blended surface inherits some of its behavior from its "local surfaces".

Lemma 12.1. *At the four corners we get the following properties*

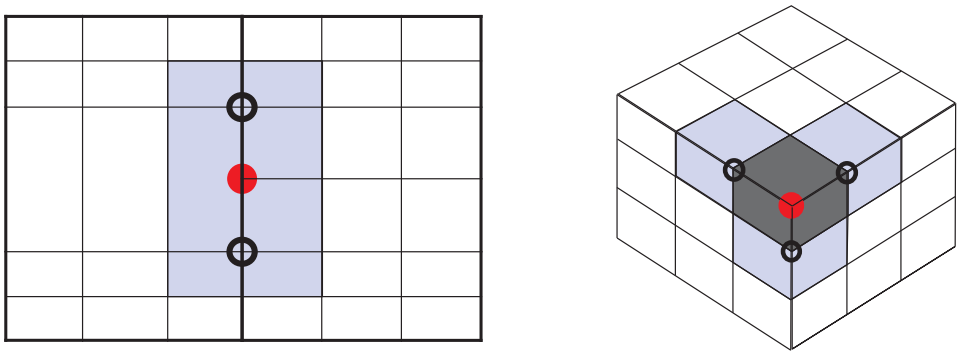


Figure 12.15: *To the left is a T-junction (solid red) and the two terminating points marked. The seven sub-patces involved in the irregular blending are marked light blue. To the right is a Star-junction (solid red) and three terminating points marked. The sub-surfaces involved are, three marked dark gray and six marked light blue.*

lower left corner	$S(0,0)$	$S \equiv s_1$ *including all its derivatives*
lower right corner	$S(1,0)$	$S \equiv s_2$ *including all its derivatives*
upper left corner	$S(0,1)$	$S \equiv s_3$ *including all its derivatives*
upper right corner	$S(1,1)$	$S \equiv s_4$ *including all its derivatives*

At the four edges we get the following properties

Left edge	$S(0,v) = s_1 + B(v)(s_3 - s_1)$	*only depend on* s_1 *and* s_3
Right edge	$S(1,v) = s_2 + B(v)(s_4 - s_2)$	*only depend on* s_2 *and* s_4
Lower edge	$S(u,0) = s_1 + B(u)(s_2 - s_1)$	*only depend on* s_1 *and* s_2
Upper edge	$S(u,1) = s_3 + B(u)(s_4 - s_3)$	*only depend on* s_3 *and* s_4

Proof. It follows from B-function properties and (12.30), (12.31) and (12.32). □

12.6.2 Tensor product Surfaces and irregular grids

Irregular grids can be quite complex. What we will discuss, however, is a collection of regular tensor product surfaces that are connected in an irregular way. This involves handling T-junctions and Star-junctions in the connections. To connect several blending surfaces to one surface, we must use special local surfaces just for the blending between neighboring surfaces. These local surfaces must cover the nearest neighborhood (according to the grid) to all surfaces to be connected.

To the left in Figure 12.15 there is an example of a T-junction, and to the right an example of a Star-junctions. We also call points that end irregular areas “terminating points”.

12.6.3 T-junctions

A T-junction is a grid line ending in an orthogonal grid line. In Figure 12.15 is there an examples of a T-junction. T-junction occurs either when the knot-vector in a surface is not the same as the knot-vector in the neighboring surface or if a knot value disappear when

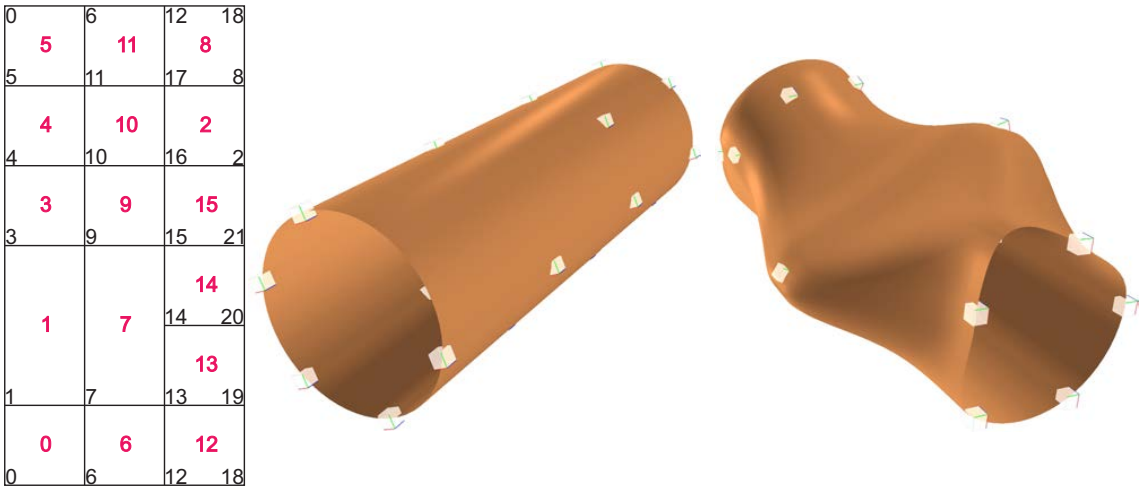


Figure 12.16: *To the left of the figure we see the parameter plane of a cylinder that is partitioned by the knot vectors* $\mathbf{u}$ *and* $\mathbf{v}$ *described in (12.33). The parameter patches are marked with red numbers in the figure, and the interpolation points are marked with black numbers. Two points have been removed. In the figure we see that points and parameter surfaces at the upper right part have taken over the indices (numbers) of the removed points and patches. To the right of the figure, we first see a blending spline cylinder with the specified T-junction. The points around the area where the interpolation points have been removed are then moved and rotated. We see the result to the right in the figure.*

passing a grid line. How to handle T-junction to achieve smooth surfaces, is given by the following theorem.

Theorem 12.1. *If there is a T-junction or a collection of connected T-junction then the local surfaces connected to these T-junction and to the terminating points of this collection must be parts of a common surface to get a smooth and even* C^∞*-smooth blending.*

Proof. From Lemma 12.1 it follows that, at the vertices, the surface is identical with the local surfaces connected to the respective vertex. If there is a T-junction, the T-junction is a vertex on two neighboring sub-surfaces on a common local surface. On the other side of the T-junction is another sub-surface from the same local surface. On this sub-surfaces is the T-junction not a vertex. To interpolate a local surface at an internal point on an edge (with all its derivatives), it follows from Lemma 12.1 that the two surfaces connected to the two vertices defining the edge must be parts of the same surface, and that the surface connected to the T-junction also must be part of the same surface. □

To the left in Figure 12.15 is a T-junction highlighted in solid grey and two terminating points marked with a black ring. The three local surfaces connected to the marked vertices and covering the irregularities, is marked light blue and is divided into seven sub-surfaces.

Figure 12.16 shows an implementation. To the left of the figure we see the parameter

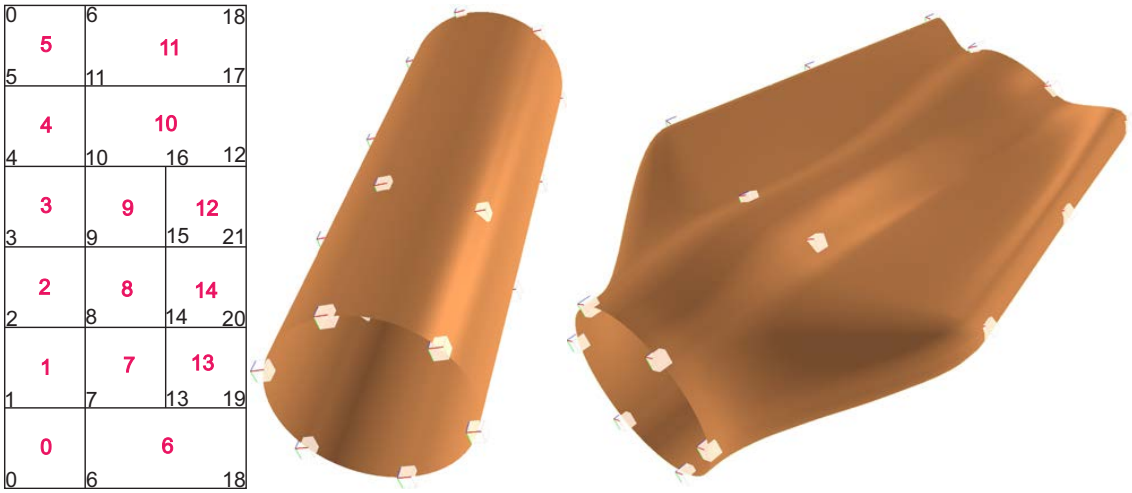


Figure 12.17: *As in Figure 12.16, it is a matter of removing interpolation points. To the left of the figure we see the parameter plane which is partitioned by the knot vectors* **u** *and* **v** *described in (12.33). The parameter patches are marked with red numbers in the figure, and the interpolation points are marked with black numbers. Two points have been removed. The placement of the numbers is a result of the algorithm for removing the points and patches. To the right of the figure, we first see a blending spline cylinder with the specified T-junctions. The points around the area where the interpolation points have been removed are then moved and rotated. We see the result to the right in the figure.*

plane of a cylinder which is partitioned by the knot vectors

$$\mathbf{u} = \left\{ -\frac{1}{2}, -\frac{1}{2}, -\frac{1}{6}, \frac{1}{6}, \frac{1}{2}, \frac{1}{2} \right\}, \quad \text{and} \tag{12.33}$$
$$\mathbf{v} = \left\{ -\frac{4}{3}\pi, -\pi, -\frac{2}{3}\pi, -\frac{1}{3}\pi, 0, \frac{1}{3}\pi, \frac{2}{3}\pi, \pi, \frac{4}{3}\pi \right\}.$$

The implementation is based on parameter patches marked with red numbers in the figure, and interpolation points including their local surfaces which are marked with black numbers in the figure. They are both placed in separate vectors where the numbers are indices in their respective vectors. The interpolation points can be removed interactively by selecting one or more points and then removing them. In Figure 12.16, this is done with two points. This is clearly illustrated in the figure because the points and patches in the upper right part of the parameter plane have taken over the numbers of the removed points and patches. The interpolation point number 14 in the figure is now a T-junction, and consequently the local surfaces in points 13, 14 and 15 must be connected in accordance with Theorem 12.1. If we use the sub-surface construction, the three points must have a common matrix. To the right in Figure 12.16, we first see a blending-spline cylinder with the current T-junction. The points around the patches where the interpolation points have been removed are then moved and rotated. The result can be seen to the right in Figure 12.16 and is still as smooth as the cylinder itself.

In Figure 12.17, the removed points are on each side of the "cyclic boundary", on a closed surface, and they are therefor internal. Two points have been removed. In Figure 12.17

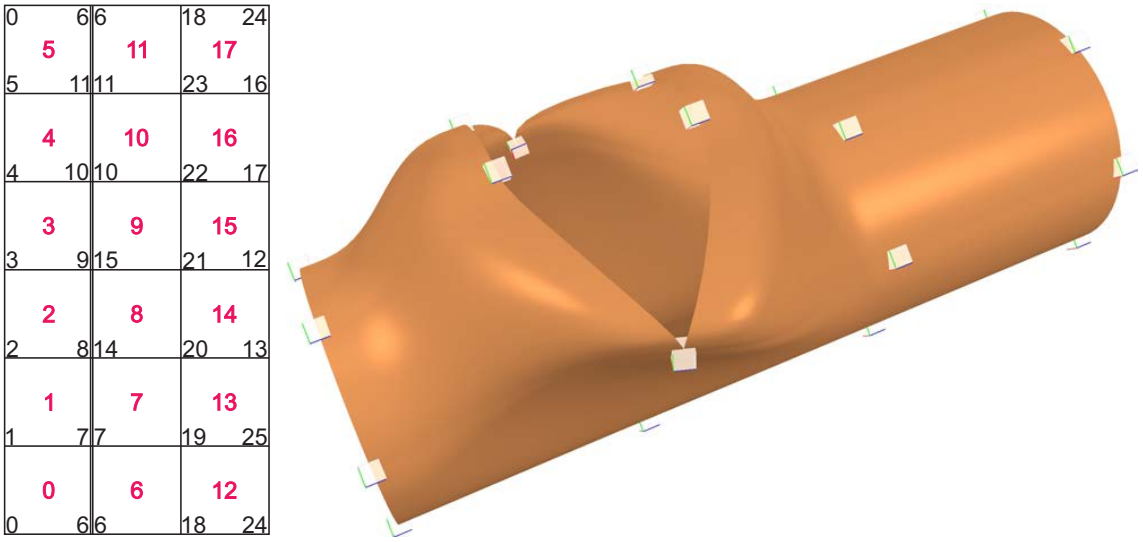


Figure 12.18: *In this example, the* **u***-vector, (12.34), has an inner multiple value, To the left of the figure we see the parameter plane and the double knot can be seen as two almost coincident vertical lines. Then 4 interpolation points with associated local surfaces are removed. These are the points to the right of 6,7,10 and 11. To the right of the figure is the surface shown in* $\mathbb{R}^3$*. The result is that we can model holes in the surface without trimming the surface, ie there is a one to one map between the parameter plane of the surface.*

we see that it is the point between patch 10 and 11, and the point between the patch 11 and 6. The figure to the right shows the result in the parameter plane after the two points have been removed. The parameter patches, marked with red numbers, keep track of the indices of the interpolation points marked with black numbers. The interpolation points also keep track of the parameter patches that surround them. The number of parameter patches is reduced by three. All points around the removed points have been moved and rotated after the two points have been removed.

If we insert a multiple knot-value, ie

$$\mathbf{u} = \left\{ -\frac{1}{2}, -\frac{1}{2}, -\frac{1}{6}, -\frac{1}{6}, \frac{1}{6}, \frac{1}{2}, \frac{1}{2} \right\} \tag{12.34}$$

we get a geometrically possible discontinuous surface. In figure 12.18 this is done. At the same time, using T-junctions, we have removed 4 interpolation points/local surfaces, these are the points with the indices 12, 13, 16 and 17. The result can be seen to the right in Figure 12.18. Where we have not removed the points along the multiple knot value, we now have a hole and where we have removed the points, the surface is continuous and smooth. Note that there is still is a one to one mapping between the parameter plane and the surface. In Pedersen at al. [128], more sophisticated holes are shown together with possible applications, especially related to Isogeometric analysis.

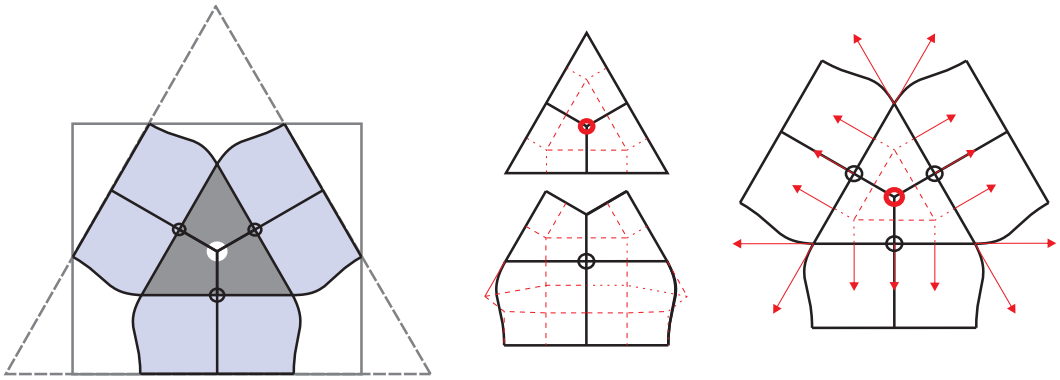


Figure 12.19: *To the left is the parameter plane of a surface (either tensor product surface or Bézier-triangle) for a Star-junction and how the sub-surfaces are. In the middle is shown the control polygon of the Bézier patches used in the reparametrization. To the right is a Bézier triangle used. It is divided in 3 by reparametrization using Bézier patches. In addition is 3 surfaces based on Hermite blended curves connected to each edges.*

12.6.4 Star-junctions

A Star-junction is a point where several grid lines meets at a non-orthogonal way. One example of a Star-junction is given to the right in Figure 12.15, where three grid lines meet at one point. The problems related to Star-junction appears clearly when we look at Figure 12.15. Lines in the parameter plane where one parameter is constant gets a kink when they pass one of the edges going out of a Star-junction. In the following lemma we show how to handle Star-junction in blending.

Theorem 12.2. *To get a smooth blending when a Star-junction occur, then:*
The local surface connected to the Star-junction and the local surfaces connected to the terminating points must be sub-surfaces of one common surface, but the local surfaces can be individually translated (not rotated, scaled, ..).
The smoothness over the edges from the Star-junction to its terminating points will be C^S, but the smoothness over the edge orthogonal to the terminating points (see Figure 12.19) will depend on the reparametrization that is used.

Proof. Because the local surfaces are either parts of the same surface or only translated, all derivatives of all orders are equal on the two local surfaces that share an edge. It follows from Lemma 12.1 that the value and all derivatives on the resulting surface must go towards the same when we go towards the edge from both sides. Thus, the edge between the Star junction and its termination points becomes C^S-smooth. At the surrounding edges, the triangle we see in Figure 12.19, the smoothness will only depend on the smoothness of reparametrization to the composite local surface. Thus, the resulting surface can only be as smooth as the reparametrization, cf. the example in Figure 12.19. □

On the left side of Figure 12.19 there is an example of a parameter plane of a surface that covers the irregular Star-junction area. There are nine sub-surfaces, marked gray and light blue, which together define four local surfaces. The top in the middle of the figure shows the local surface attached to the Star-junction itself, below we see one of the three local

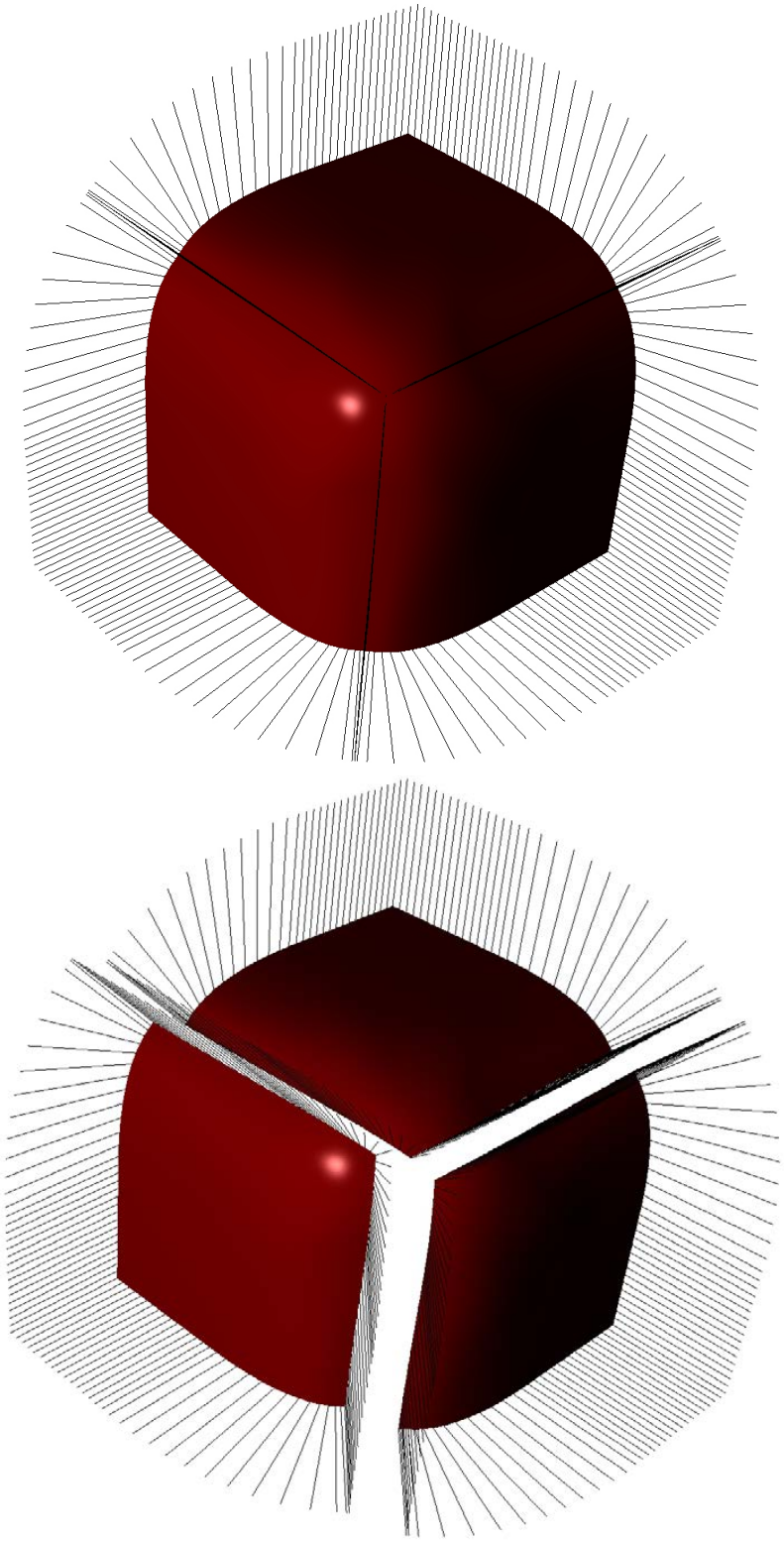


Figure 12.20: *Example of a smooth surface with a Star-junction. The surface is made by blending of planar surfaces and one Bézier triangle expanded by using Hermite blended curves from the three edges.*

surfaces that are connected to the terminating points. Note that the upper part of the lower surface is the same as the lower part of the upper surface.

At the upper middle triangle in Figure 12.19, the reparametrization is done in each of the three sub-surfaces using a Bézier map, $\omega : \mathbb{R}^2 \to \mathbb{R}^2$. Each of the three sub-surfaces is reparameterized in both directions using a 2^{nd}-degree Bézier map with a 3×3 control polygon, in a "symmetric" way. The lower surface in the middle has 4 parts. The two upper parts are equal to the two lower parts in the surface above. In the two lower parts we see in dashed red a Bézier image with 3×4 control points, ie 2^{nd} degree horizontally and 3^{rd} degree vertically.

In the lower middle surface in Figure 12.19, we name the control points on the left side from top to bottom for $c_{i,j},\ i = 1,2,3$ and $j = 1,2,...,6$. The upper patch then use $c_{i,j},\ i = 1,2,3$ and $j = 1,2,3$ and the lower patch $c_{i,j},\ i = 1,2,3$ and $j = 3,4,5,6$ then the local surface is continuous. If in addition $c_{i,4} - c_{i,3} = c_{i,3} - c_{i,2}$ then the local surface is C^1-smooth over the edge.

On the right side of Figure 12.19 is used a Béziertriangle map expanded by Hermite blended curves and vector valued functions connected to each of the edges of the triangle. The curves on the other side can be arbitrary chosen. The number of derivative functions used in the Hermite blending determines the continuity.

In Figure 12.20 a surface with a Star-junction is shown, using the map explained on the right side in Figure 12.19. The Surface is C^1-smooth because only one derivative function is used. Using the two maps described on the left side in Figure 12.19, the result was similar to the first example. To the left in Figure 12.20 the surface is shown with normals drawn at the boundary and the three edges connected to the Star-junction. The figures illustrates the continuity. To the right, the three parts are moved apart.

Chapter 13

Triangular Surfaces

A triangle is a simplex, and thus one of the basic shapes in geometry. It is a polygon with 3 edges and 3 vertices. A triangle can be seen as the boundary of a surface, typically embedded in $\mathbb{R}^3$. Often a triangular surface is planar, but here we will concentrate on curved triangular surfaces. Figure 13.1 gives an example of this.

Modelling of 3D objects in $\mathbb{R}^3$ is usually done by modeling the outer boundary. These surfaces of are bounded compact and connected and of different topological genus g, number of holes/handles[1]. Tessellation of surfaces to 3D objects is often done by triangulation. Triangular representation is more simple than general polygons. A short investigation of these triangulations tells us that $\varkappa(S)$ is actually a number independent of a particular triangulation, and will be the same for all triangulations of the given surface S.

In the first part of this chapter we will concentrate on the basic properties of simple triangular surfaces (a simplex). Furthermore, we will look at triangulated surfaces, ie surfaces that consist of a set of connected triangles that are more or less smooth in the connection.

[1]The Euler-Poincaré characteristic for a compact and connected surface S is $\varkappa(S) = F - E + V$, where F is the number of triangles, E is the number of edges and V is the number of vertices. It is related to the genus, g, of the surface in the following way, $\varkappa(S) = 2(1-g)$. See the Gauss-Bonnet Theorem in [49].

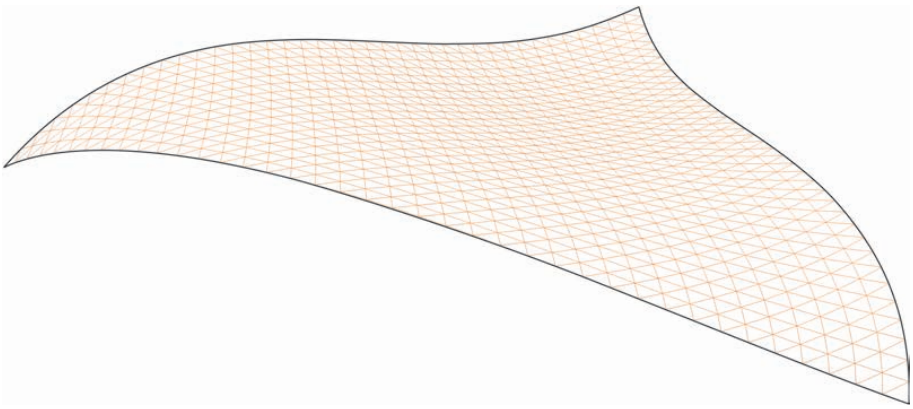


Figure 13.1: *A smooth and non-planar triangular surface embedded in* $\mathbb{R}^3$.

The best known curved triangular surface is the Bézier triangle. Bézier triangles were (according to Farin [64]) first introduced early in the 1960s by de Casteljau in two internal Citroën technical reports [38] and [39]. The basic constructions of a Bézier triangle will shortly be repeated in section 13.1.

However, the main goal of the first part of this chapter is the introduction of triangles based on blending, also defined on a domain described by homogeneous barycentric coordinates.

13.1 Bézier triangles

A convex set defined by homogeneous barycentric coordinates (a Δ_2 simplex) defines the domain for the Bézier triangle, where we can construct basis functions of arbitrary degrees d in the following way and still keep it in Δ_2,

$$(u+v+w)^d = 1, \quad \text{where} \quad u,v,w \geq 0 \quad \text{and} \quad d > 0.$$

The expression of these basis functions for Bézier triangles are thus the Bernstein polynomials of degree d, where for $u+v+w=1$,

$$b_{d,i,j,k}(u,v,w) = \binom{d}{ijk} u^i v^j w^k, \quad \text{where} \quad i+j+k=d \quad \text{and} \quad i,j,k \geq 0.$$

For Bézier triangular surfaces we, therefor, have the following general formula,

$$S(u,v,w) = \sum_{\substack{i+j+k=d,\\ i,j,k\geq 0}} c_{i,j,k}\, b_{d,i,j,k}(u,v,w) \quad \text{for} \quad u+v+w=1,$$

where $c_{i,j,k} \in \mathbb{R}^n$, are the coefficients, and n usually is 3.

Recall the Bernstein factor matrices $T_d(t)$ described in definition 4.5 in section 4.4.2. Here, each line in these matrices sums to 1, so they are ready for barycentric coordinates. We just need to replace $u = 1-t$ and $v = t$. Rune Dalmo has in [34] made this extension for higher dimensions. He showed that $T_d(u_0,\ldots,u_k)$ is a $\binom{d+k-1}{k} \times \binom{d+k}{k}$ band limited matrix with $k+1$ nonzero elements on each row. The matrix is defined recursively as

$$T_d(u_0,\ldots,u_k) = \left(\begin{array}{c|c} T_{d-1}(u_0,\ldots,u_k) & 0 \\ \hline T_{d^{diag}}(u_0) & T_d(u_1,\ldots,u_k) \end{array}\right).$$

A 3^{rd}-degree Bézier triangle is then

$$s(u,v,w) = T_1(u,v,w)\; T_2(u,v,w)\; T_3(u,v,w)\; \mathbf{C}$$

where

$$T_1(u,v,w) = \begin{pmatrix} u & v & w \end{pmatrix}, \qquad T_2(u,v,w) = \left(\begin{array}{ccc|ccc} u & v & w & 0 & 0 & 0 \\ \hline 0 & u & 0 & v & w & 0 \\ 0 & 0 & u & 0 & v & w \end{array}\right)$$

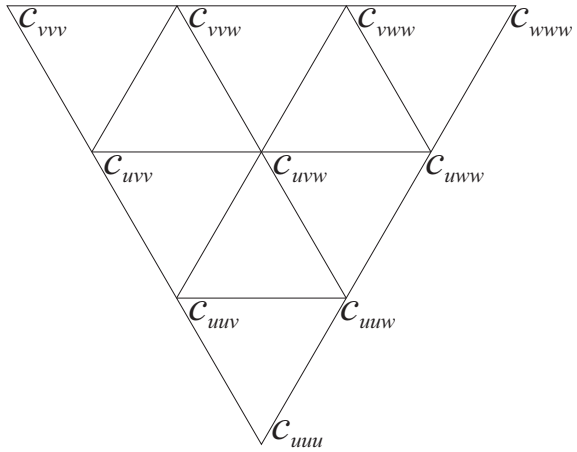


Figure 13.2: *The notation, distribution and connection of the control points in a 3^{rd}-degree Bézier triangle.*

$$T_3(u,v,w) = \left(\begin{array}{cccccc|cccc} u & v & w & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & u & 0 & v & w & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & u & 0 & v & w & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & u & 0 & 0 & v & w & 0 & 0 \\ 0 & 0 & 0 & 0 & u & 0 & 0 & v & w & 0 \\ 0 & 0 & 0 & 0 & 0 & u & 0 & 0 & v & w \end{array}\right)$$

and

$$\mathbf{C} = (c_{uuu}, c_{uuv}, c_{uuw}, c_{uvv}, c_{uvw}, c_{uww}, c_{vvv}, c_{vvw}, c_{vww}, c_{www})^T$$

Figure 13.2 shows the notation and distribution/interconnection of the control points $\mathbf{C}$. Note that the organization of the points in the vector is from bottom to top and from left to right in the figure. If we now compute the 3^{rd}-degree Bézier triangle we get

$$\begin{aligned} s(u,v,w) = {} & u^3 c_{uuu} + 3u^2 v c_{uuv} + 3u^2 w c_{uuw} + 3uv^2 c_{uvv} + uvw c_{uvw} \\ & + 3uw^2 c_{uww} + v^3 c_{vvv} + 3v^2 w c_{vvw} + 3vw^2 c_{vww} + w^3 c_{www} \end{aligned}$$

In homogeneous barycentric coordinates, we have both partial derivatives, S_u, S_v, S_w and directional derivatives, $S_{\mathbf{d}}$. From (4.33) we saw that we can compute the derivative by differentiating the last matrix, and that the rows of this matrix sum up to 0, and that the derivatives are thus vectors. Here in barycentric coordinates, the lines in the derivative matrix will sum up to 1, and thus the partial derivatives will be points. The appropriate derivatives are thus direction derivatives, where the vector $\mathbf{d} = p_1 - p_2$ (the distance vector between two points) is a vector in homogeneous barycentric coordinates $\mathbf{d} = (r,s,t)$ where $r+s+t = 0$. The directional derivative is thus a vector and can be computed in the following way

$$S_{\mathbf{d}}(u,v,w) = r\, S_u(u,v,w) + s\, S_v(u,v,w) + t\, S_w(u,v,w), \tag{13.1}$$

where $u+v+w = 1$ and $r+s+t = 0$.

Note that the parameters at the corners are $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$. We can thus use $d_1 = (-1,1,0)$ and $d_2 = (-1,0,1)$ to compute directional derivatives,

$$s_{d_1}(u,v,w) = s_v(u,v,w) - s_u(u,v,w) \quad \text{and} \quad s_{d_2}(u,v,w) = s_w(u,v,w) - s_u(u,v,w),$$

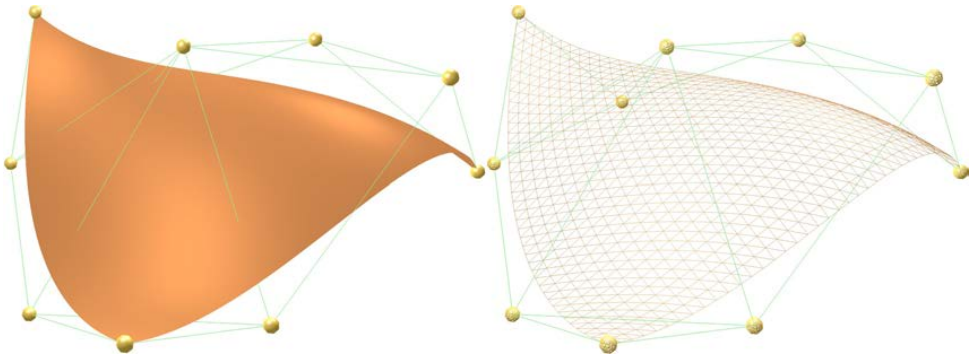


Figure 13.3: *A 3^{rd}-degree Bézier triangle including the control points and the control polygon is shown. The surface is plotted both shaded and in wire-frame mode so that all control points can be seen.*

and the normal is

$$n = s_{d_1}(u,v,w) \wedge s_{d_2}(u,v,w).$$

Figure 13.3 shows a shaded Bézier triangle, where normals are computed. More about the general theory of Bernstein-Bézier triangles, including the algorithm for evaluation and derivations, can be found in [63] or [64].

13.2 B-function in homogeneous barycentric coordinates

In section 7.1 is the univariate B-function defined. In **D5** in Definition 7.1 is the point symmetry defined, that is $B(t)+B(1-t)=1$, which in 1-dimensional homogeneous barycentric coordinates becomes $B(u)+B(v)=1$ or $B(u,v)=(x_1,x_2)$ where (x_1,x_2) are homogeneous barycentric coordinates.

If we now extend the B-function to homogeneous barycentric coordinates we get:

Definition 13.1. *A B-function $B(u_0,u_1,\ldots,u_n)$, where $\sum_{i=0}^{n} u_i = 1$, is:*

D1 *a permutation function* $\quad B:\Delta_n \to \Delta_n, \quad$ *where Δ_n is an n-simplex*

– D2 *thus is* $\quad B(\ldots,u_{i-1},0,\ldots)=\{\ldots,x_{i-1},0,\ldots\}, \quad i=0,1,\ldots,n$

– D3 *and* $\quad B(\ldots,u_{i-1},1,\ldots)=\{\ldots,x_{i-1},1,\ldots\}, \quad i=0,1,\ldots,n$

– D4 *and that is monotone, i.e.* $\quad \sum_{j=0}^{n} u_j \frac{\partial}{\partial u_i} B(\bar{u}) - \frac{\partial}{\partial u_i} B(\bar{u}) = \bar{x}, \quad x_i \geq 0, \quad i=0,1,\ldots,n$

D5 *A B-function is symmetric if* $\quad B(\bar{u})=\bar{x}$, *for all coincident permutations of $\bar{u}$ and $\bar{x}$*

*Note that to show the monotony, **D4**, it follows that for each coordinate, the directional derivative from any point, in the direction of the top point of the current coordinate must be ≥ 0 (the top point is where the current coordinate value is 1, see figure 13.4).*

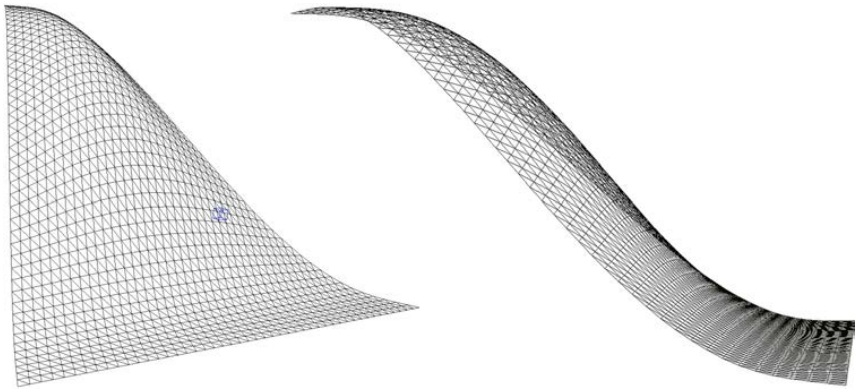


Figure 13.4: *The B-function $\mathscr{B}_1(u_1,u_2,u_3)$ for triangles seen from two different view points.*

A B-function maps $n+1$ coordinates. With triangles we get 3 coordinates such that

$$B(u,v,w) = (x_0,x_1,x_2)\,,$$

and it follows that we can denote each element with an index, ie.

$$B_i(u,v,w) = x_i, \quad i = 0,1,2.$$

As we learned from univariate B-functions in section 7.1, we have many ways of constructing B-functions in homogeneous barycentric coordinates.

We will look at two examples, first using the univariate B-function.

Definition 13.2. *Given a point $(u_0,u_1,...,u_n) \in \Delta_n$ (in homogeneous barycentric coordinates, fulfilling the convexity property). A B-function in Δ_n can be defined as follows,*

$$B\,(u_0,u_1,\ldots,u_n) = \frac{1}{\sum_{i=0}^{n} B(u_i)}\,(B(u_0),B(u_1),\ldots,B(u_n)) \tag{13.2}$$

where $B(u)$ is a univariate B-function as defined in Definition 7.1.

For $n = 2$, ie. a triangle, each component of the B-function will be

$$B_i\,(u_0,u_1,u_2) = \frac{B(u_i)}{B(u_0)+B(u_1)+B(u_2)} \quad \text{for} \quad i = 0,1,2.$$

We can see that Definition 13.2 fulfill all 5 points in Definition 13.1.

In Figure 13.4 is a 2-dimensional B-function for one coordinate shown. In this plot is the Expo-Rational B-function (7.31) used as the univariate B-function in Definition 13.2.

As in the Bézier triangle, the directional derivatives must be used, $\mathbf{d} \in \Upsilon_n$. Thus, the partial derivatives are necessary. Therefore, for $n > 0$ we have for the following 6 equations. First, the first-order partial derivatives, where for $i = 0,1,...,n$ is

$$D_{u_i}B_i(u_0,\ldots,u_n) = B'(u_i)\frac{\sum_{j=0}^{n} B(u_j) - B(u_i)}{\left(\sum_{j=0}^{n} B(u_j)\right)^2}, \tag{13.3}$$

and, for $j = 0, 2, ..., n$, but where $j \neq i$,

$$D_{u_j} B_i(u_0, \ldots, u_n) = B'(u_j) \frac{-B(u_i)}{\left(\sum_{j=0}^{n} B(u_j)\right)^2}. \tag{13.4}$$

For second order partial derivatives we have,

$$D^2_{u_i} B_i(u_0, \ldots, u_n) = \left(B''(u_i) - \frac{2\left(B'(u_i)\right)^2}{\sum_{j=0}^{n} B(u_j)} \right) \frac{\sum_{j=0}^{n} B(u_j) - B(u_i)}{\left(\sum_{j=0}^{n} B(u_j)\right)^2}, \tag{13.5}$$

and for $j = 0, 1, ..., n$, but where $j \neq i$ for both

$$D^2_{u_j} B_i(u_0, \ldots, u_n) = \left(B''(u_j) - \frac{2\left(B'(u_j)\right)^2}{\sum_{j=0}^{n} B(u_j)} \right) \frac{-B(u_i)}{\left(\sum_{j=0}^{n} B(u_j)\right)^2}, \tag{13.6}$$

and

$$D_{u_i} D_{u_j} B_i(u_0, \ldots, u_n) = B'(u_i)\, B'(u_j) \frac{2B(u_i) - \sum_{j=0}^{n} B(u_j)}{\left(\sum_{j=0}^{n} B(u_j)\right)^3}, \tag{13.7}$$

and for $j = 0, 1, ..., n$, but where $j \neq i$, and for $h = 0, 1, ..., n$, but where $h \neq i, j$, we have

$$D_{u_h} D_{u_j} B_i(u_0, \ldots, u_n) = B'(u_h)\, B'(u_j) \frac{2B(u_i)}{\left(\sum_{j=0}^{n} B(u_j)\right)^3}. \tag{13.8}$$

Note that all these 6 partial derivatives has a factor with the chosen univariate B-function with the same order of derivatives. Thus the order, Definition 7.2, of the homogeneous barycentric B-function will be the same as the order of the chosen univariate B-function.

Now, given a vector $\mathbf{d} \in \Upsilon_n$, i.e.,

$$\mathbf{d} = \mathbf{u_1} - \mathbf{u_2} = (d_0, d_1, ..., d_n), \quad \text{where} \quad \mathbf{u_1} \text{ and } \mathbf{u_2} \in \Delta_n.$$

Remember that partial derivatives in homogeneous barycentric coordinates gives points, To compute normals we need vectors, ie. directional derivatives for the B-function in homogeneous barycentric coordinates. That is

$$D_{\mathbf{d}} B_i(u_0, u_1, ..., u_n) = \sum_{j=0}^{n} d_j\, D_{u_j} B_i((u_0, u_1, ..., u_n)). \tag{13.9}$$

It is convenient to define some main directions for derivatives. For triangles we can use,

$$\mathbf{d}_1 = (-1, 1, 0), \quad \text{and} \quad \mathbf{d}_2 = (-1, 0, 1). \tag{13.10}$$

In these main directions are the derivatives up to second order for $B_i(u, v, w)$, $i = 0, 1, 2$, as follows,

$$\begin{aligned}
D_{\mathbf{d}_1} B_i(u, v, w) &= D_v B_i(u, v, w) - D_u B_i(u, v, w), \\
D_{\mathbf{d}_2} B_i(u, v, w) &= D_w B_i(u, v, w) - D_u B_i(u, v, w), \\
D^2_{\mathbf{d}_1} B_i(u, v, w) &= D^2_v B_i(u, v, w) - D_u D_v B_i(u, v, w) + D^2_u B_i(u, v, w), \\
D^2_{\mathbf{d}_2} B_i(u, v, w) &= D^2_w B_i(u, v, w) - D_u D_w B_i(u, v, w) + D^2_u B_i(u, v, w), \\
D_{\mathbf{d}_1} D_{\mathbf{d}_2} B_i(u, v, w) &= D_v D_w B_i(u, v, w) - D_u D_v B_i(u, v, w) - D_u D_w B_i(u, v, w) + D^2_u B_i(u, v, w).
\end{aligned}$$

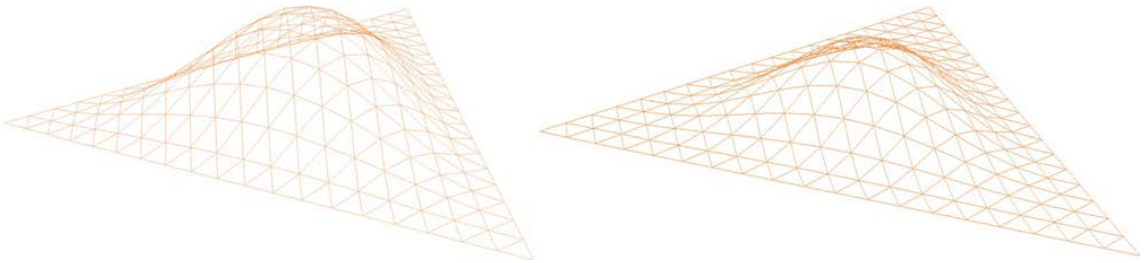


Figure 13.5: *Plot of density functions based on the Beta-function. To the left is an order 1 function $\Phi(u,v,w) = u^2v^2w^2$, and to the right is an order 2 function $\Phi(u,v,w) = u^3v^3w^3$.*

The next example is based on density of mass distribution. Based on observations in the univariate B-functions, Beta-functions (7.18) and an Expo-Rational B-function (7.31) who both are constructed with this technic, is it naturally to expand this to B-functions in homogeneous barycentric coordinates.

Definition 13.3. *Given a point $p = (u_0, u_1, ..., u_n) \in \Delta_n$ (in homogeneous barycentric coordinates, fulfilling the convexity property). A B-function in Δ_n can be defined as follows,*

$$B(u_0,u_1,\ldots,u_n) = \frac{1}{mass(\Delta_n)}\left(mass(p,u_0), mass(p,u_1),\ldots,mass(p,u_n)\right) \qquad (13.11)$$

where $mass(\Delta_n)$ is the mass of the hole simplex, $mass(p,u_i)$, $i = 0,1,\ldots,n$ is the mass of the simplex defined by the point p and the sub-simplex Δ_{n-1} where $u_i = 0$.

For $n = 2$, ie. the triangular case follows two examples of density functions,

$$\Phi(u,v,w) = e^{-\frac{a}{uvw}}, \quad a > 0, \quad \text{and} \quad \Phi(u,v,w) = (uvw)^k, \quad k > 0.$$

The last example, the Beta function, we will now go ahead with and develop B functions. In Figure 13.5 is an order 1 and an order 2 density function of type Beta-function plotted. the order means, like univariate B-functions, the order of the derivatives that are zero along the edges. Following Definition 13.3 expression (13.11) we get

$$B(u,v,w) = \frac{1}{\int_\Omega \Phi(u,v,w)d\Omega}\left(\int_{\Omega_0}\Phi(u,v,w)d\Omega_0, \int_{\Omega_1}\Phi(u,v,w)d\Omega_1, \int_{\Omega_2}\Phi(u,v,w)d\Omega_2\right)$$

where the domain Ω and the sub-domains Ω_0, Ω_1 and Ω_2 are shown in Figure 13.6. The point $p = (u,v,w)$, the ´´evaluation"-point, and the coordinates of the two vectors in the figure are $r = p - p_1 = (u, v-1, w)$ and $s = p - p_2 = (u,v,w-1)$. The integrals are thus

$$\int_\Omega \Phi(u,v,w)d\Omega = \int_0^1 \int_0^{1-\mu} \phi(\mu,\nu,1-\mu-\nu)\, d\nu d\mu$$

$$\int_{\Omega_0} \Phi(u,v,w)d\Omega_0 = \int_0^u \int_{\frac{v}{u}\mu}^{1-\frac{1-v}{u}\mu} \phi(\mu,\nu,1-\mu-\nu)\, d\nu d\mu$$

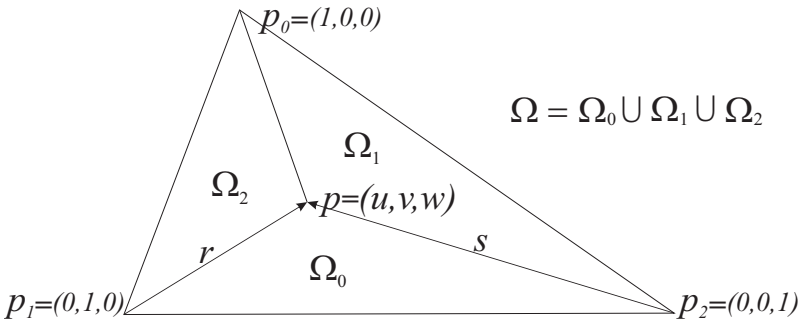


Figure 13.6: *We see a Triangle domain Ω and its three vertices p_0, p_1 and p_2. A point p divide it into three sub-domains Ω_0, Ω_1 and Ω_2. Also two vectors r and s are shown.*

Most of the limits in the integrals are self explanatory. What needs explanation is the lower and upper boundaries of the last integral. They follow because we use the vertices p, p_1 and p_2 and the vectors r and s scaled by $\frac{\mu}{u}$ (see Figure 13.6). Thus we get

$$p_1 + \frac{\mu}{u}(p - p_1) = ((0,1,0) + \frac{\mu}{u}((u,v,w) - (0,1,0)) = \left(\mu,\ 1 - \frac{1-v}{u}\mu,\ \frac{w}{u}\mu\right)$$

where the second component is the upper limit, and

$$p_2 + \frac{\mu}{u}(p - p_2) = ((0,0,1) + \frac{\mu}{u}((u,v,w) - (0,0,1)) = \left(\mu,\ \frac{v}{u}\mu,\ 1 - \frac{1-w}{u}\mu\right)$$

where the second component is the lower limit. $\int_{\Omega_1} \Phi(u,v,w)d\Omega_1$ and $\int_{\Omega_2} \Phi(u,v,w)d\Omega_2$ can then be computed by cyclic shifting of the barycentric variables u, v, w. Then -

For 1^{st}-order Beta-functions we get the following B-function

$$B(u,v,w) = \big((6vw - 2u + 3)u^2,\ (6uw - 2v + 3)v^2,\ (6uv - 2w + 3)w^2,\big)$$

and for 2^{nd}-order Beta-functions we get

$$\begin{aligned} B(u,v,w) = \big(&(30vw(3vw - u + 1) + 6u^2 - 15u + 10)u^3, \\ &(30uw(3uw - v + 1) + 6v^2 - 15v + 10)v^3, \\ &(30uv(3uv - w + 1) + 6w^2 - 15w + 10)w^3\big) \end{aligned}$$

The partial derivatives for both 1^{st} and 2^{nd}-order Beta-functions are easy to compute. And we can use the same technique as used in the first example, Definition 13.2, to compute the directional derivatives of all order. We also see that the Hermite order S, Definition 7.2, for Beta-functions will be the degree of the density functions minus 1, and that the order generally follow what we see for the univariate B-function of the same type.

Figure 13.7 shows the components of two Beta functions in homogeneous barycentric coordinates, the three of 1^{st}-order on the left and the three of 2^{nd}-order on the right. Remember that since they are elements of homogeneous barycentric coordinates, they always sum up to 1. The order S works in the same way as for univariate B-functions, but the start and end are where all directional derivatives are zero, ie where $u_i = 1$ and $u_i = 0$. The former is a point, while the latter is a sub-simplex Δ_{n-1}, i.e. an edge.

Figure 13.7: *we see the three components of a Beta function in homogeneous barycentric coordinates, three 1^{st}-order functions on the left, three 2^{nd}-order functions on the right.*

13.3 Blending triangles

The general formula for a Blending triangle is

$$S(u,v,w) = \sum_{i=0}^{2} s_i(u,v,w)\, B_i(u,v,w), \quad \text{where } u+v+w=1, \quad u,v,w \geq 0,$$

where $s_i(u,v,w),\ i=0,1,2$ are local triangles and B(u,v,w) is a B-function in homogeneous barycentric coordinates. In Figures 13.8, there are two plots of Blending triangles and their respective 3 local triangles. All the local triangles are actually planar in both figures, and we can see their boundary/control polygon marked as green lines. In the figure to the left we can see that they are also parallel. We can also see that the global Blending triangle interpolates each of the local triangles in one of the corners, and we know from the properties of B-functions that the interpolation not only involves the position, it actually includes interpolation of a certain order of derivatives depending on the order of the B-function used. The Blending surface to the right in Figure 13.8 is initially the same as the one to the left. The local triangles have just been moved and rotated slightly. The partial derivatives are computed using the product rule and the directional derivatives are done by using (13.9) and the direction we find in (13.10).

It is preferable for several reasons that the local surfaces are oriented in such a way that the full support of the first parameter (i.e. $u=1$) is in the interpolation point to the global Blending triangle. This simplifies the construction and organization of the local triangles, but it introduces the need for a new type of mapping of the parameters between the global triangle and the local triangles for each of the vertices. This mapping is actually only a cyclic use of the parameters. This cyclic mapping is shown in the following expression,

$$S(u,v,w) = s_0(u,v,w)\, B_0(u,v,w) + s_1(v,w,u)\, B_1(u,v,w) + s_2(w,u,v)\, B_2(u,v,w), \quad (13.12)$$

where $u+v+w=1$, $u,v,w \geq 0$, and $s_i,\ i=0,1,2$ are local triangles, orientated in such a way that the full support of the first parameter (i.e. the "local $u=1$") is in the interpolation point with the global Blending triangle. Therefore, we have the rotation of the parameters of the local triangles in (13.12). However, the expression is straightforward to implement.

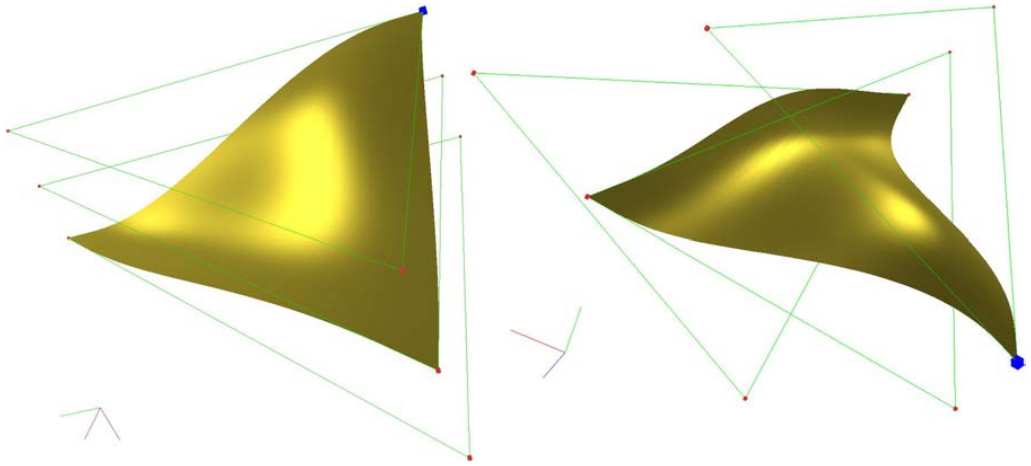


Figure 13.8: *Two blending spline triangle surfaces with their three local triangles marked with green lines. The local triangles are all 1^{st}-degree Bézier triangles. To the left they are parallel to each other but are rotated and moved in the surface to the right.*

13.4 Local Bézier triangles and Hermite interpolation

Figure 13.9 shows three Blending triangles, where the local triangles are 2^{nd}-degree Bézier triangles. This provides a small insight into the shaping possibilities where both the control points of the Bézier triangles and also the position and orientation of the Bézier triangles themselves can be changed.

To make a Blending triangle by ”copying” a part of another surface, the local triangles can be made by Hermite interpolating each of the vertices with one Bézier triangle. In section 12.3.1, local tensor product Bézier surfaces are made with Hermite interpolation at one point. The technique for Bézier triangles is quite similar, but must be adapted to triangles and thus barycentric coordinates. The method is:

√ Given a surface $g : \Omega_g \subset \mathbb{R}^2 \to \mathbb{R}^n$, where usually $n = 3$.

√ Given three points in the parameter plane Ω_g; p_0, p_1 and p_2, each associated with one local triangle. The polynomial degree of each of the local Bézier triangles is also required.

√ Make local Bézier triangles $s_i(u,v,w)$ of degree d_i using position and derivative information at the parameter values p_i, $i = 0,1,2$. Note the cyclic shift in the expressions below, cf. (13.12). The hermite interpolation is based on the fact that the following requirements must be met to make the three Bézier triangles, i.e.

$$
\begin{aligned}
D^i_{(-1,1,0)} D^j_{(-1,0,1)} s_0(1,0,0) &= D^i_{(-1,1,0)} D^j_{(-1,0,1)} S(1,0,0) = D^i_{p_1-p_0} D^j_{p_2-p_0} g(p_1), \\
D^i_{(-1,1,0)} D^j_{(-1,0,1)} s_1(1,0,0) &= D^i_{(0,-1,1)} D^j_{(1,-1,0)} S(0,1,0) = D^i_{p_2-p_1} D^j_{p_0-p_1} g(p_2), \\
D^i_{(-1,1,0)} D^j_{(-1,0,1)} s_2(1,0,0) &= D^i_{(1,0,-1)} D^j_{(0,1,-1)} S(0,0,1) = D^i_{p_0-p_2} D^j_{p_1-p_2} g(p_3),
\end{aligned}
$$

for all combinations of i, j where $0 \le (i+j) \le d_k$, and $i, j \ge 0$, for $k = 0,1,2$.

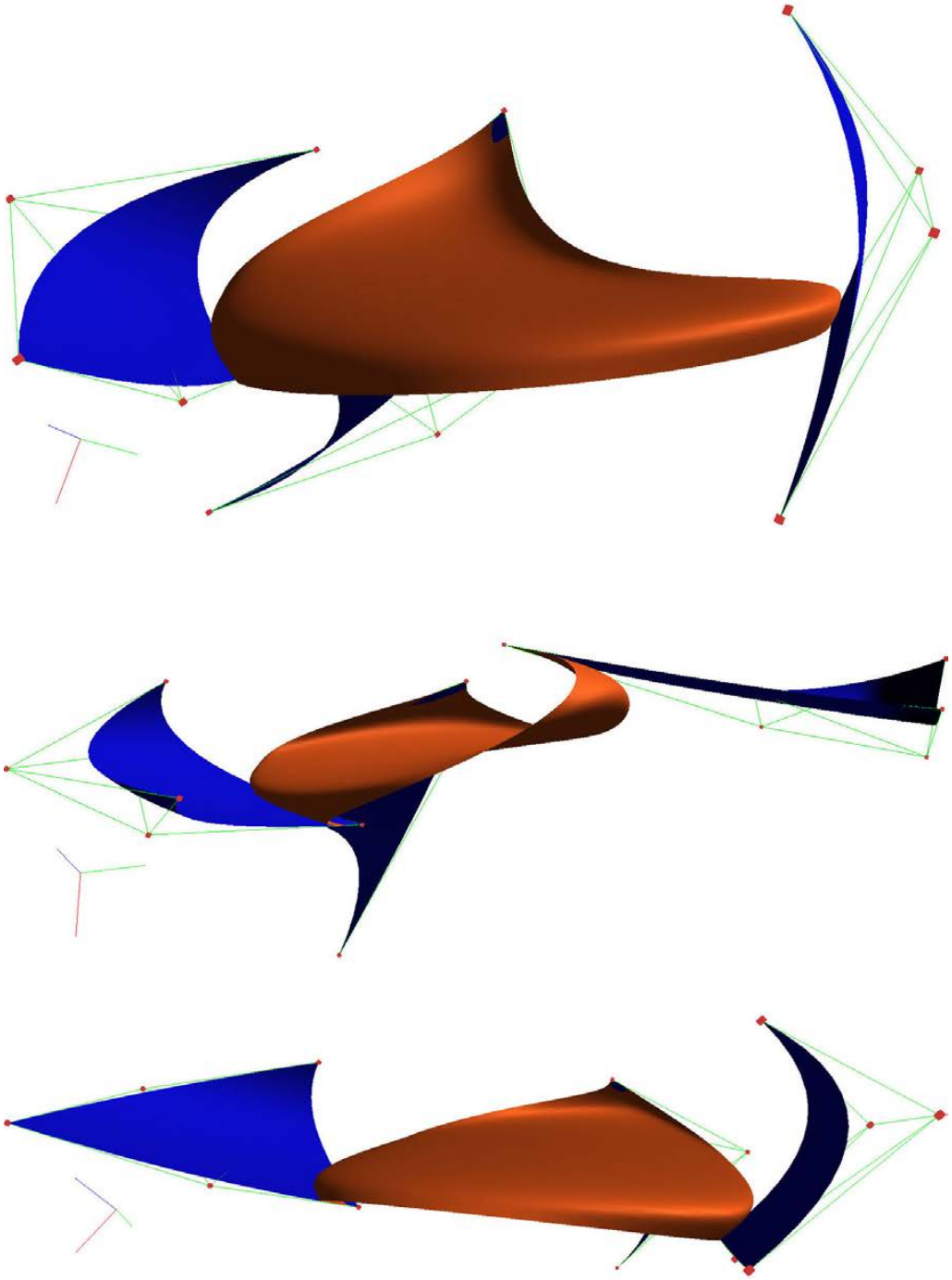


Figure 13.9: *Three different Blending triangles are shown. The local triangles are 2^{nd}-degree Bézier triangles (not planar any more), and we can see them colored blue. The control polygons of these local Bézier triangles can be seen as green lines, and their control points as red cubes.*

Blending triangles interpolates their local triangles in the vertices (corners), with the position and all derivatives up to the Hermite order S. This can be clearly seen in Figures 13.9. This is why Hermite interpolation of the local surfaces only must be done at one point for each local triangular surface, where the position and the derivatives from the original surface g uniquely determine the local surface.

To make Hermite interpolation of Bézier triangles, we start from the requirements on the previous page. We take the expression on the left which is the expression of position and derivatives to Bézier triangles, and the expression on the right which is the same derivatives in the same point on the original surface,ie

$$D^i_{(-1,1,0)} D^j_{(-1,0,1)} s(1,0,0) = D^i_{\mathbf{d}_1} D^j_{\mathbf{d}_2} g(\mathbf{p}), \quad \text{for} \quad 0 \leq (i+j) \leq d, \quad i,j \geq 0, \quad (13.13)$$

where d is the polynomial degree of the Bézier triangle, $\mathbf{p}$ is the point in Ω_g, and $\mathbf{d}_1, \mathbf{d}_2$ are vectors $\in \mathbb{R}^2$ at the point $\mathbf{p}$. Recall that the number of equations in (13.13) is

$$n_b = \sum_{i=1}^{d+1} i,$$

which follows from that the sum of the possible variants, for $i+j=d$ is $d+1$ when $i,j \geq 0$. For $d=1$, we get $n_b = 3$, for $d=2$, we get $n_b = 6$, and for $d=3$, we get $n_b = 10$. This matches the relationship between the degree and the number of control points/basis functions of a Bézier triangle. Equation (13.13) is, therefore, uniquely determining the Bézier triangle from the position and derivatives of the original surface.

To compute the equations for the Hermite interpolation we use the expression for the directional derivatives for the Bézier triangle. The 1^{st}-order (directional) derivatives are given in (13.1). All these computations are done in [102], page 157–161.

For 3 points $\mathbf{p}, \mathbf{p}_1, \mathbf{p}_2 \in \Omega_g$, and two vectors $\mathbf{q}_1 = \mathbf{p}_1 - \mathbf{p}$ and $\mathbf{q}_2 = \mathbf{p}_2 - \mathbf{p}$, the results for the Hermite interpolation of a parametric surface by the 1^{st}-degree Bézier triangle is

$$\begin{aligned} c_0 &= g(\mathbf{p}), \\ c_1 &= c_0 + dg_{\mathbf{p}}(\mathbf{q}_1), \\ c_2 &= c_0 + dg_{\mathbf{p}}(\mathbf{q}_2). \end{aligned}$$

The Hermite interpolation of a 2^{nd} degree Bézier triangle is a little bit more complex, the control points will then be

$$\begin{aligned} \mathbf{c}_1 &= g(\mathbf{p}), \\ \mathbf{c}_2 &= \mathbf{c}_1 + \frac{1}{2} dg_{\mathbf{p}}(\mathbf{q}_1), \\ \mathbf{c}_3 &= \mathbf{c}_1 + \frac{1}{2} dg_{\mathbf{p}}(\mathbf{q}_2), \\ \mathbf{c}_4 &= -\mathbf{c}_1 + 2\mathbf{c}_2 + \frac{1}{2} d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_1), \\ \mathbf{c}_5 &= -\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 + \frac{1}{2} d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_2), \\ \mathbf{c}_6 &= -\mathbf{c}_1 + 2\mathbf{c}_3 + \frac{1}{2} d(dg(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2). \end{aligned}$$

The Hermite interpolation of a 3^{rd}-degree Bézier triangle gives the control points

$$\begin{aligned}
\mathbf{c}_1 &= g(\mathbf{p}),\\
\mathbf{c}_2 &= \mathbf{c}_1 + \frac{1}{3} dg_{\mathbf{p}}(\mathbf{q}_1),\\
\mathbf{c}_3 &= \mathbf{c}_1 + \frac{1}{3} dg_{\mathbf{p}}(\mathbf{q}_2),\\
\mathbf{c}_4 &= -\mathbf{c}_1 + 2\mathbf{c}_2 + \frac{1}{6} d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_1),\\
\mathbf{c}_5 &= -\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 + \frac{1}{6} d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_2),\\
\mathbf{c}_6 &= -\mathbf{c}_1 + 2\mathbf{c}_3 + \frac{1}{6} d(dg(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2),\\
\mathbf{c}_7 &= \mathbf{c}_1 - 3\mathbf{c}_2 + 3\mathbf{c}_4 + \frac{1}{6} d(d(dg(\mathbf{q}_1))(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_1),\\
\mathbf{c}_8 &= \mathbf{c}_1 - 2\mathbf{c}_2 - \mathbf{c}_3 + \mathbf{c}_4 + 2\mathbf{c}_5 + \frac{1}{6} d(d(dg(\mathbf{q}_1))(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_2),\\
\mathbf{c}_9 &= \mathbf{c}_1 - \mathbf{c}_2 - 2\mathbf{c}_3 + 2\mathbf{c}_5 + \mathbf{c}_6 + \frac{1}{6} d(d(dg(\mathbf{q}_1))(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2),\\
\mathbf{c}_{10} &= \mathbf{c}_1 - 3\mathbf{c}_3 + 3\mathbf{c}_6 + \frac{1}{6} d(d(dg(\mathbf{q}_2))(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2).
\end{aligned}$$

As we see above, to find the control points of the Bézier triangles, we must calculate position and directional derivatives on the original surface $g(u,v)$, both first, second, third, and the cross derivatives between two directions. Therefore we need a position $\mathbf{p} = (u_0, v_0)$ and two direction vectors, $\mathbf{q}_1 = (u_1, v_1)$ and $\mathbf{q}_2 = (u_2, v_2)$, in the parameter plane of the original surface g. So, the 1^{st} order derivatives in the two directions are

$$\begin{aligned}
dg_{\mathbf{p}}(\mathbf{q}_1) &= u_1 g_u(\mathbf{p}) + v_1 g_v(\mathbf{p}),\\
dg_{\mathbf{p}}(\mathbf{q}_2) &= u_2 g_u(\mathbf{p}) + v_2 g_v(\mathbf{p}).
\end{aligned}$$

The 2^{nd}-order derivatives are

$$\begin{aligned}
d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_1) &= u_1 d(g_u)_{\mathbf{p}}(\mathbf{q}_1) + v_1 d(g_v)_{\mathbf{p}}(\mathbf{q}_1) = u_1^2 g_{uu}(\mathbf{p}) + 2u_1 v_1 g_{uv}(\mathbf{p}) + v_1^2 g_{vv}(\mathbf{p}),\\
d(dg(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2) &= u_2 d(g_u)_{\mathbf{p}}(\mathbf{q}_2) + v_2 d(g_v)_{\mathbf{p}}(\mathbf{q}_2) = u_2^2 g_{uu}(\mathbf{p}) + 2u_2 v_2 g_{uv}(\mathbf{p}) + v_2^2 g_{vv}(\mathbf{p}),\\
d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_2) &= u_1 d(g_u)_{\mathbf{p}}(\mathbf{q}_2) + v_1 d(g_v)_{\mathbf{p}}(\mathbf{q}_2)\\
&= u_1 u_2 g_{uu}(\mathbf{p}) + (u_1 v_2 + u_2 v_1) g_{uv}(\mathbf{p}) + v_1 v_2 g_{vv}(\mathbf{p}),
\end{aligned}$$

and the 3^{rd}-order derivatives are

$$\begin{aligned}
d(d(dg(\mathbf{q}_1))(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_1) &= u_1^3 g_{uuu}(\mathbf{p}) + (3u_1^2 v_1) g_{uuv}(\mathbf{p}) + (3u_1 v_1^2) g_{uvv}(\mathbf{p}) + v_1^3 g_{vvv}(\mathbf{p}),\\
d(d(dg(\mathbf{q}_2))(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2) &= u_2^3 g_{uuu}(\mathbf{p}) + (3u_2^2 v_2) g_{uuv}(\mathbf{p}) + (3u_2 v_2^2) g_{uvv}(\mathbf{p}) + u_2^3 g_{vvv}(\mathbf{p}),\\
d(d(dg(\mathbf{q}_1))(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2) &= u_1 u_2^2 g_{uuu}(\mathbf{p}) + a_1 g_{uuv}(\mathbf{p}) + b_1 g_{uvv}(\mathbf{p}) + v_1 v_2^2 g_{vvv}(\mathbf{p}),\\
d(d(dg(\mathbf{q}_1))(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_2) &= u_1^2 u_2 g_{uuu}(\mathbf{p}) + a_2 g_{uuv}(\mathbf{p}) + b_2 g_{uvv}(\mathbf{p}) + v_1^2 v_2 g_{vvv}(\mathbf{p}),
\end{aligned}$$

$a_1 = u_2^2 v_1 + 2u_1 u_2 v_2$, $b_1 = u_1 v_2^2 + 2u_2 v_1 v_2$, $a_2 = u_1^2 v_2 + 2u_1 u_2 v_1$, $b_2 = u_2 v_1^2 + 2u_1 v_1 v_2$.

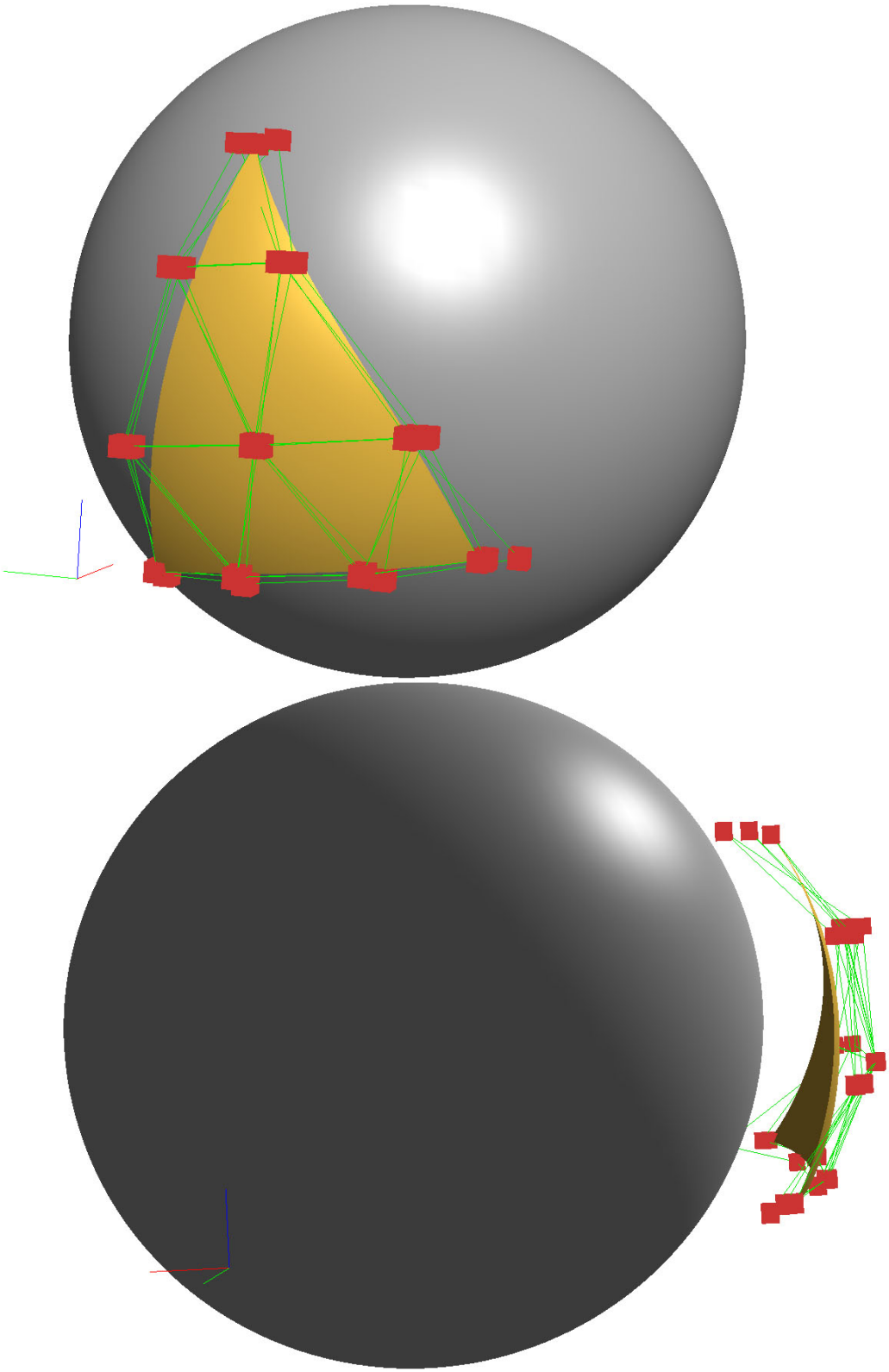


Figure 13.10: *Two views of a sphere and one Blending triangular surface interpolating a part of the sphere. The Blending triangle is slightly translated away from the sphere. The local triangles are 3^{rd}-degree Bézier triangles, and we can see all the control polygons of the local Bézier triangles as green lines, and the control points as red cubes.*

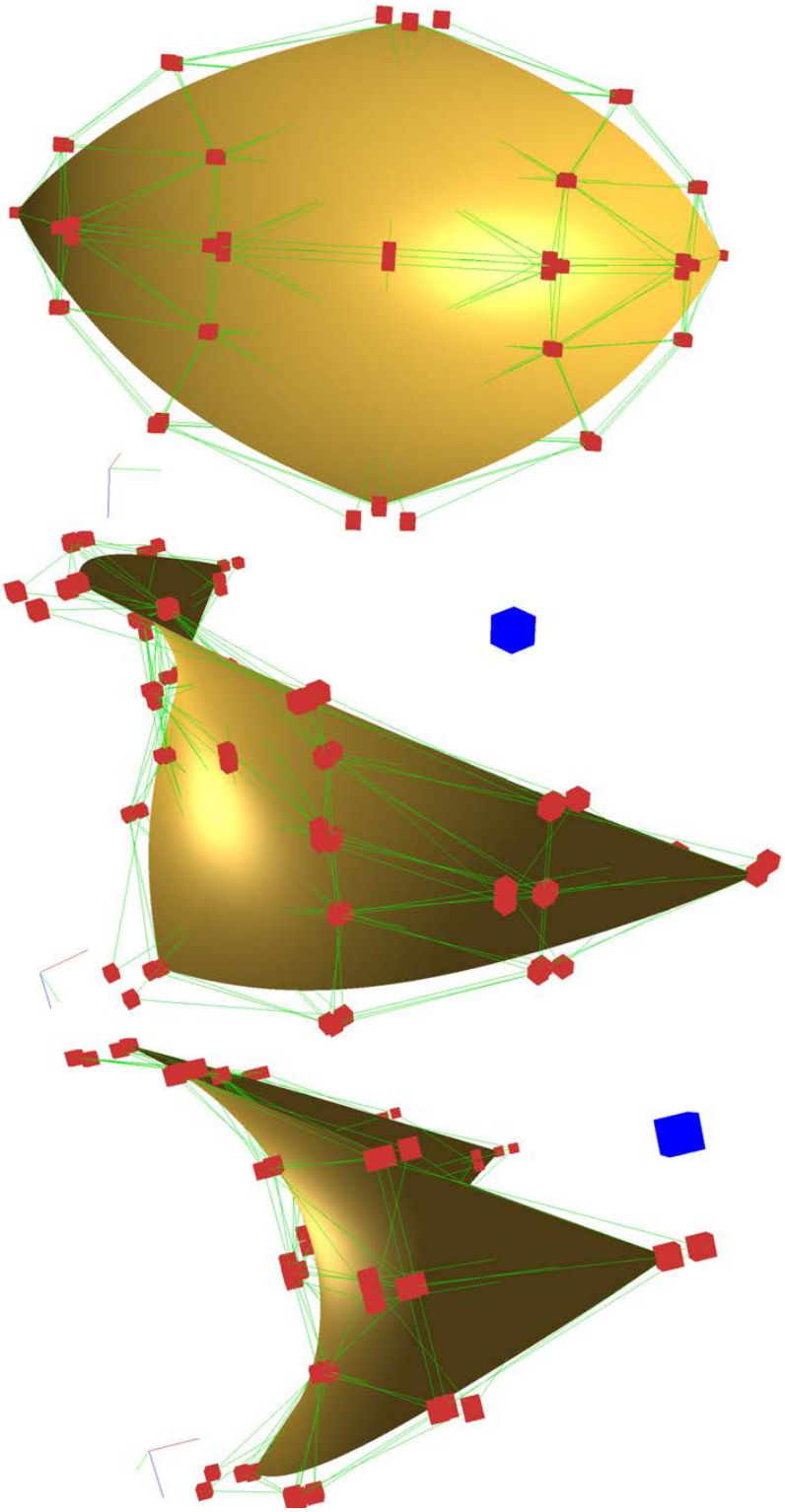


Figure 13.11: *Three different views of a surface composed of four Blending triangles that Hermite-interpolates a part of a torus, equation (12.24), in 5 points. There are* 4×3 *local* 3^{rd}*-degree Bézier triangles. We can see their control polygons as green lines, and the control points as red cubes. The blue cube is the center of the original torus.*

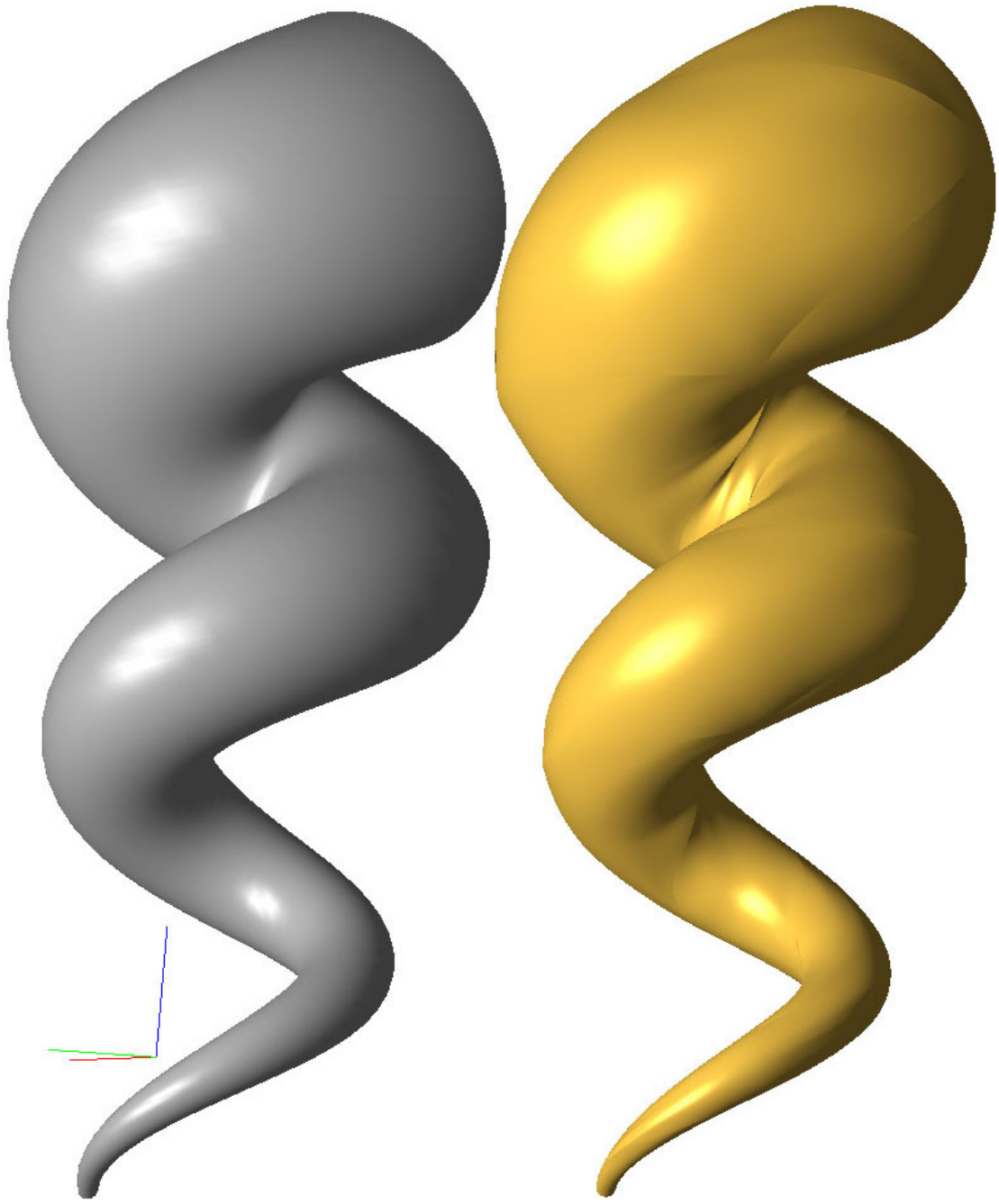


Figure 13.12: *On the left hand side there is a plot of a Seashell, equation (12.25), and on the right hand side there is a plot of a set of 80 Blending triangles, Hermite interpolating a Seashell at 44 different points. The 80 Blending triangles are all independent of each other, but are "continuously connected"; this means that there are no holes in the composite surface after the interpolation.*

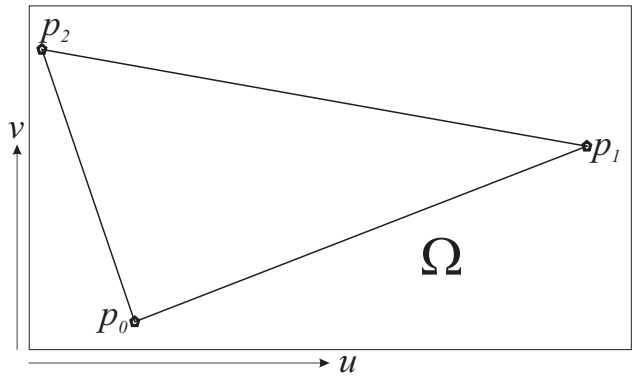


Figure 13.13: *The parameter plane Ω of a surface $\mathbb{S} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^n,\ n > 0$. The three points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2 \in \Omega$ describe a triangle in the parameter plane.*

In Figure 13.10, a Blending triangle is made by interpolating a sphere at three points,One 2^{nd}*degree*Bézier triangle is made at each of the points. In the figure, the Blending triangle is slightly removed from the sphere so that we can see it better. We can also see the control polygons for each of the local Bézier triangles as green lines, and the control points as red cubes. The three interpolation points are all in the parameter plane of the sphere, and the directional derivatives are the vector between these points in the parameter plane. The parametrization of the sphere affects the shape of the triangle. It can clearly be seen that two of the edges form a "slight S". The center points of the three control polygons almost coincides, and the three control polygons are quite equal.

In the previous example and the next two examples, an Expo-rational B-function with an ∞ order is used. The next example is based on a torus, equation (12.24). In Figure 13.11, we can see three different views of four Blending triangles computed by Hermite interpolation of a torus, expression (12.24), at 5 different points. The composition of the four Blending triangles is clearly continuous. Although this cannot be easily seen, the composition is actually based on four different triangles. But even if they together look as being G^∞ (geometrically infinitely smooth), they are not. This can clearly be seen in the next example, which is a Hermite interpolation of a "Sea Shell" surface, equation (12.25). On the left hand side of Figure 13.12 there is a plot of a "Sea Shell" surface, (12.25), and on the right hand side there is a plot of 80 Blending triangles interpolating the whole "Sea Shell" surface. One can clearly see that the composition is continuous, but it does not seem to be G^∞. It is actually G^∞ at all 44 interpolation points, but at the 124 edges it seems to be only continuous, G^0, although the result is quite good. An observation is that it seems to be smooth at a point in the middle of each edges. All this follows from the properties of the B-function and that if an object is covered by several parameterisations, the map from one to another must be continuous and smooth.

13.5 Sub-triangles from any parametric surface

It is possible to extract a triangular surface $S(u,v,w)$ from an ordinary surface in Cartesian coordinates ie. $\mathbb{S} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^n$, where n usually is 3. $S(u,v,w)$ can, thus, be defined

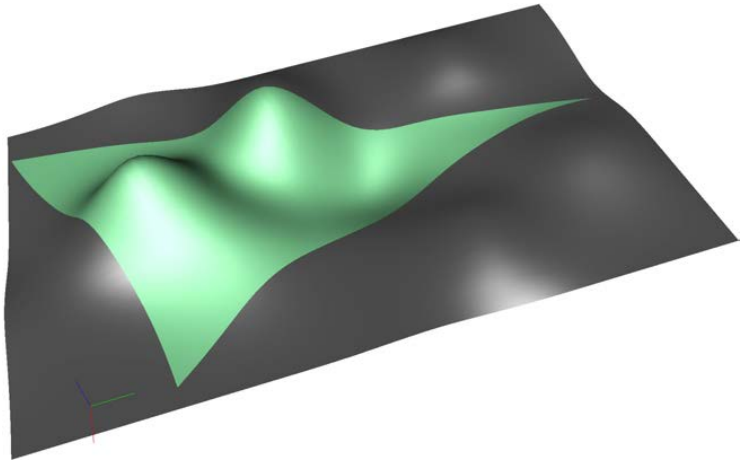


Figure 13.14: *The grey surface is a B-spline tensor product surface. The triangular green surface is defined by three points in the parameter plane of the B-spline tensor product surface, and its domain is the minimum convex set including these three points (that is, a triangle in the parametric domain).*

by the three points $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2 \in \Omega$, in the parameter plane of the parametric surface $\mathbb{S}$, see Figure 13.13.

To clarify the notation, we have an ´´ordinary" parametric surface $\mathbb{S}(\mathbf{p}) \in \mathbb{R}^n,\ \mathbf{p} \in \Omega \subset \mathbb{R}^2$ and the differential, $d\mathbb{S}_{\mathbf{u}} = [\mathbb{S}_u\ \ \mathbb{S}_v]_{(}\,\mathbf{p}) \in \mathbb{R}^{n\times 2}$. A triangular surface S is then

$$S(u,v,w) = \mathbb{S}(\mathbf{u}), \quad \text{where} \quad \mathbf{u} = u\mathbf{p}_0 + v\mathbf{p}_1 + w\mathbf{p}_2,$$

where the three points are $\mathbf{p}_i = \begin{pmatrix} u_i \\ v_i \end{pmatrix},\ i = 0,1,2$. The first order partial derivatives are,

$$\begin{aligned}
S_u(u,v,w) &= d\mathbb{S}_{\mathbf{u}}(\mathbf{p}_0) = u_0\ \mathbb{S}_u(\mathbf{u}) + v_0\ \mathbb{S}_v(\mathbf{u}),\\
S_v(u,v,w) &= d\mathbb{S}_{\mathbf{u}}(\mathbf{p}_1) = u_1\ \mathbb{S}_u(\mathbf{u}) + v_1\ \mathbb{S}_v(\mathbf{u}),\\
S_w(u,v,w) &= d\mathbb{S}_{\mathbf{u}}(\mathbf{p}_2) = u_2\ \mathbb{S}_u(\mathbf{u}) + v_2\ \mathbb{S}_v(\mathbf{u}),
\end{aligned}$$

and the second order partial derivatives are,

$$\begin{aligned}
S_{uu}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_0))_{\mathbf{u}}(\mathbf{p}_0) = u_0^2\ \mathbb{S}_{uu}(\mathbf{u}) + 2u_0v_0\ \mathbb{S}_{uv}(\mathbf{u}) + v_0^2\ \mathbb{S}_{vv}(\mathbf{u}),\\
S_{vv}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_1))_{\mathbf{u}}(\mathbf{p}_1) = u_1^2\ \mathbb{S}_{uu}(\mathbf{u}) + 2u_1v_1\ \mathbb{S}_{uv}(\mathbf{u}) + v_1^2\ \mathbb{S}_{vv}(\mathbf{u}),\\
S_{ww}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_2))_{\mathbf{u}}(\mathbf{p}_2) = u_2^2\ \mathbb{S}_{uu}(\mathbf{u}) + 2u_2v_2\ \mathbb{S}_{uu}(\mathbf{u}) + v_2^2\ \mathbb{S}_{vv}(\mathbf{u}),\\
S_{uv}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_0))_{\mathbf{u}}(\mathbf{p}_1) = u_0u_1\ \mathbb{S}_{uu}(\mathbf{u}) + (u_0v_1 + v_0u_1)\mathbb{S}_{uv}(\mathbf{u}) + v_0v_1\ \mathbb{S}_{vv}(\mathbf{u}),\\
S_{uw}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_0))_{\mathbf{u}}(\mathbf{p}_2) = u_0u_2\ \mathbb{S}_{uu}(\mathbf{u}) + (u_0v_2 + v_0u_2)\mathbb{S}_{uv}(\mathbf{u}) + v_0v_2\ \mathbb{S}_{vv}(\mathbf{u}),\\
S_{vw}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_1))_{\mathbf{u}}(\mathbf{p}_2) = u_1u_2\ \mathbb{S}_{uu}(\mathbf{u}) + (u_1v_2 + v_1u_2)\mathbb{S}_{uv}(\mathbf{u}) + v_1v_2\ \mathbb{S}_{vv}(\mathbf{u}).
\end{aligned}$$

To calculate directional derivatives and normals, we follow the same procedure as for Bézier triangles, see section 13.1. In Figure 13.14, a triangular surface extracted from a tensor product B-spline surface is shown. The domain of this triangular surface is shown in Figure 13.13.

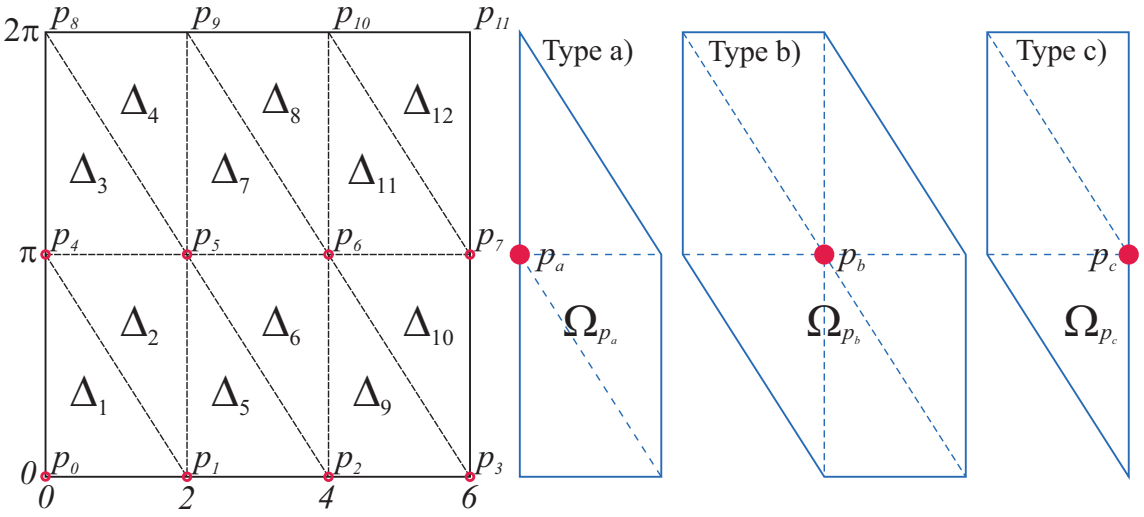


Figure 13.15: *To the left is the parameter domain of a cylinder,* $[0,6] \times [0,2\pi)$. *We see 8 points marked in red, and 4 points without marks because they are equal to the 4 points at the bottom. The parameter domain is triangulated. To the right we see the 3 different types of sub-domains we get, the red marks are the points they are connected to.*

We can copy any parametric surface by triangulating the domain and using these sub-triangles as local triangles, Be aware that we then must have 3 equal triangles to blend. If all triangles that are connected to one vertex can be moved and/or rotated together, then we have a tool for shaping objects. This is the subject for the next section.

13.6 Surface approximation by triangulation.

Any parametric surface can be triangulated. If we, in the parameter plane of a surface, have a set of points (vertices) then we can triangulate by connecting these points with edges. If the surface is closed/cyclic in one or both directions, the triangulation must also act on cyclic domains. The goal now is to make a copy of a surface with a set of connected triangular blending surfaces. The concept is as follows,

1. We start with a point set in the surface parameter domain.
2. We triangulate by connecting points with edges.
3. To each point we assign a sub-surface, ie a parameter area that covers all triangles where the point is one of the corners of the triangle.
4. To each point/sub-surface we assign an homogeneous matrix.
5. We now find each triangle in 3 different sub-surfaces. These three triangles, each from its own sub-surface, can now be used in the construction of a blending triangle.

Initially we get an exact copy of a given surface, but the shape can be changed by moving/rotating the points (sub-surfaces) by the homogeneous matrix assign to the point.

We will now look at two examples. The first example is a cylinder, equation (9.2). To the

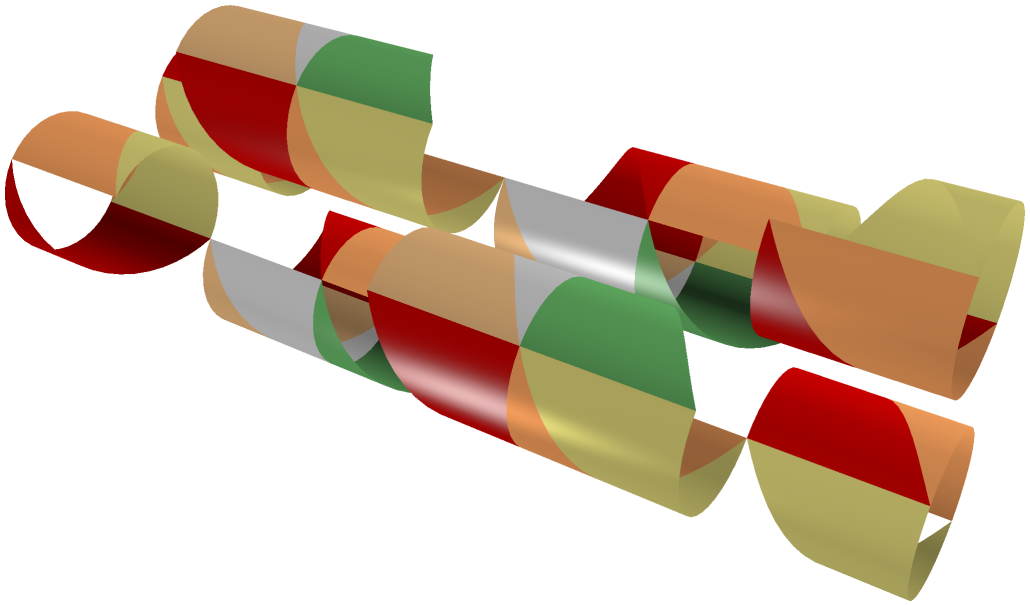


Figure 13.16: *The 8 sub-surfaces of the cylinder with the partition given in Figure 13.15 and the table below. Each triangle of the sub-surface are colored separately.*

left in Figure 13.15 we see the parameter domain with 8 points which is marked with red. These are the vertices for triangulating the domain. The 4 points at the top are the same as the 4 points at the bottom due of the cyclic structure. The domain is triangulated and each of the 12 triangles is marked as Δ_i, $i = 1,2,\ldots,12$. A sub-domain Ω_{p_i} is also assigned to each point p_i, $i = 0,1,\ldots,7$, see to the right in Figure 13.15. These sub-domains can be of three different types, and in the figure these are marked as type a), type b) and type c).

The parameter domain of a sub-surfaces is the union of all triangles that have one of their corners as the defining point of the parameter domain of the sub-surface. The table below describes the parameter domains of the 8 sub-surfaces of the cylinder example.

sub-surface domain	The triangles that form the domain	Type
Ω_{p_0}	$\Delta_1,\ \Delta_3,\ \Delta_4$	a)
Ω_{p_1}	$\Delta_1,\ \Delta_2,\ \Delta_5,\ \Delta_4,\ \Delta_7,\ \Delta_8$	b)
Ω_{p_2}	$\Delta_5,\ \Delta_6,\ \Delta_9,\ \Delta_8,\ \Delta_{11},\ \Delta_{12}$	b)
Ω_{p_3}	$\Delta_9,\ \Delta_{10},\ \Delta_{12}$	c)
Ω_{p_4}	$\Delta_3,\ \Delta_1,\ \Delta_2$	a)
Ω_{p_5}	$\Delta_3,\ \Delta_4,\ \Delta_7,\ \Delta_2,\ \Delta_5,\ \Delta_6$	b)
Ω_{p_6}	$\Delta_7,\ \Delta_8,\ \Delta_{11},\ \Delta_6,\ \Delta_9,\ \Delta_{10}$	b)
Ω_{p_7}	$\Delta_{11},\ \Delta_{12},\ \Delta_{10}$	c)

In the table, in the column triangles, there are a total of 36 triangles, and each specific triangle is always found at exact three different sub-surfaces. For example, triangle Δ_1 is in sub-surface Ω_{p_0}, Ω_{p_1} and Ω_{p_4}, triangle Δ_7 is in sub-surface Ω_{p_1}, Ω_{p_5} and Ω_{p_6}.

Figure 13.16 shows the 8 sub-surfaces of the cylinder example. Accordingly, they are parts of a cylinder and each of them is divided into triangles which are shown by each having its own color. Together, the sub-surfaces will cover the cylinder 3 times. To each

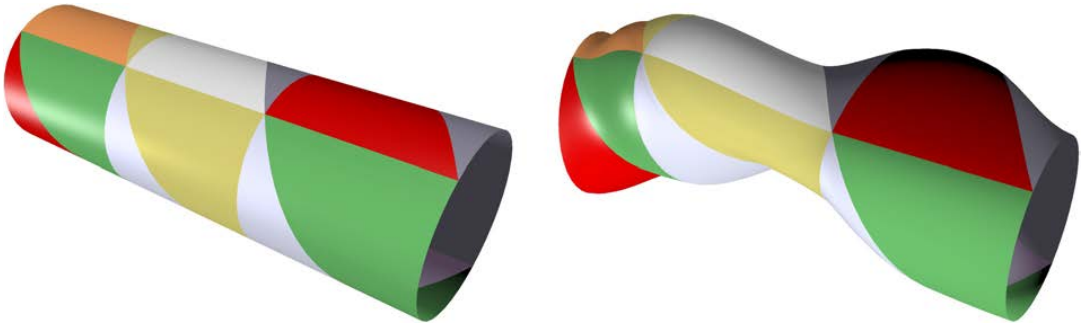


Figure 13.17: *To the left we see a cylinder composed of Blending-triangles. To the right we see the same set of triangular surfaces, but now they are deformed. This is because some sub-surfaces are moved apart in either vertical or horizontal directions.*

sub-surface is assigned a homogeneous matrix (see page 155). In practical implementation, the sub-surfaces are only represented as the definition point and a set of triangles in the parameter domain as well as the homogeneous matrix. Each of the sub-surfaces can then be moved, rotated, scaled, ... (affine maps, see page 16) with the homogeneous matrix so that for each blending triangle we get, in the same way as expression (13.12) but without the cyclic mapping, the following expression

$$S_i(\mathbf{u}) = B_0(\mathbf{u})\ H_0\ s_i(\mathbf{u}) + B_1(\mathbf{u}) H_1\ s_i(\mathbf{u}) + B_2(\mathbf{u}) H_2\ s_i(\mathbf{u})$$

where $\mathbf{u} = (u, v, w)$. This can be reorganized to

$$S_i(\mathbf{u}) = H(\mathbf{u})\ s_i(\mathbf{u}) \tag{13.14}$$

where

$$H(\mathbf{u}) = B_0(\mathbf{u})\ H_0 + B_1(\mathbf{u}) H_1 + B_2(\mathbf{u}) H_2. \tag{13.15}$$

and $s_i(\mathbf{u})$ is the expression for the sub-triangle $\mathbb{S}(\Delta_i)$ of the cylinder $\mathbb{S}$. Here is the H_0 matrix connected to the point in the corner of the triangle where $u = 1$, H_1 is the matrix connected to the point in the corner of the triangle where $v = 1$, and H_2 is the matrix connected to the point in the corner of the triangle where $w = 1$.

The expressions (13.14) and (13.15) are the general expression for surface approximation by triangulation. The construction is similar to the construction in Section 12.4 for tensor product sub-surface construction. In Figure 13.17, on the left side, there is a cylinder made by surface approximation by triangulation. The object is an exact copy of the initial cylinder. However, the construction is designed for shape changing. To the right in Figure 13.17, the homogeneous matrices are no longer identity matrices. Here the sub-surfaces are moved apart in alternating vertical and horizontal directions. But we can still recognize each individual triangle and clearly see the deformation of these.

The derivatives are straight forward to compute,

$$\begin{aligned}
D_u S_i(\mathbf{u}) &= H_u(\mathbf{u})\ s_i((\mathbf{u})) + H(\mathbf{u})\ D_u s_i((\mathbf{u})) \\
D_v S_i(\mathbf{u}) &= H_v(\mathbf{u})\ s_i((\mathbf{u})) + H(\mathbf{u})\ D_v s_i((\mathbf{u})) \\
D_w S_i(\mathbf{u}) &= H_w(\mathbf{u})\ s_i((\mathbf{u})) + H(\mathbf{u})\ D_w s_i((\mathbf{u}))
\end{aligned}$$

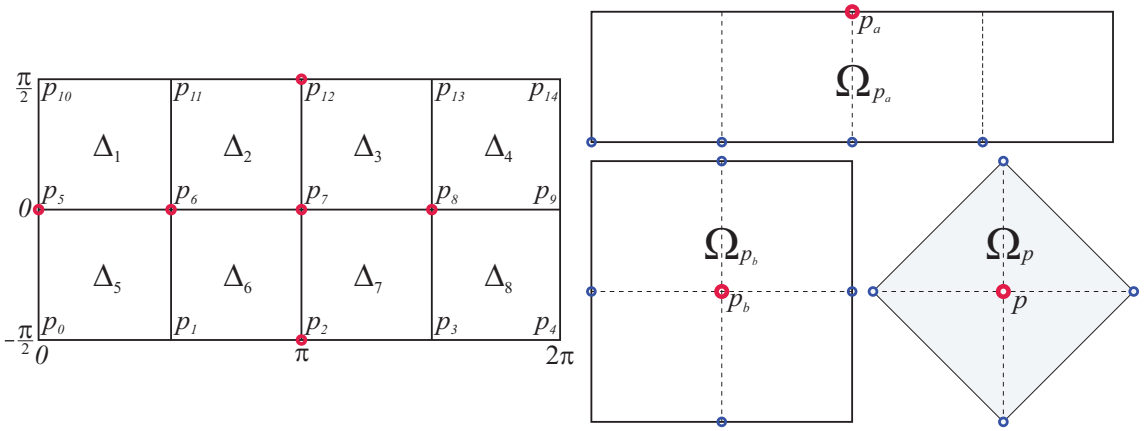


Figure 13.18: *On the left side we see the parameter plane of a sphere. It is divided into 8 parts which apparently look like squares but are related to triangles because one edge (top or bottom) is actually one point. In the parameter plane there are 6 points marked in red, 4 around "equator", one on the top and one at the bottom. To the right we see the parameter area of the sub-surfaces Ω_{p_a} and Ω_{p_b}, and the transformation of both of these to a union of triangles, shown as a light blue rotated square composed of 4 triangles.*

where

$$H_u(\mathbf{u}) = D_u B_0(\mathbf{u})\, H_0 + D_u B_1(\mathbf{u}) H_1 + D_u B_2(\mathbf{u}) H_2,$$

and where the other derivatives can be computed in the same way.

The next example is a sphere where we use the following expression,

$$\mathbb{S}(u,v) = \begin{pmatrix} \cos u \, \cos v \\ \sin u \, \cos v \\ \sin v \end{pmatrix}, \quad u \in [0, 2\pi), \quad v \in (-\pi, \pi). \tag{13.16}$$

Note that the points on both poles are not included in the formula. They must be added separately because at these points there is no one-to-one mapping from the parameter plane to the 3D space, and that they are irregular since we cannot span a tangent plane at these points with the formula.

In Figure 13.18, on the left side, we see the parameter plane of a sphere, as expressed in (13.16). This is divided into 8 "apparent" squares, "apparent" because each of these squares is mapped to triangular surfaces in $\mathbb{R}^3$. The top line and the bottom line are each actually one point as showed in Figure 13.18 with the points p_2 and p_{12}. The right side of Figure 13.18 shows two types of parameter domains for sub-surfaces, Ω_{p_a} and Ω_{p_b}. The defining point of the sub-surface is marked in red and there are 4 other points (in blue) that define the triangles that together form the domain. The transition from squares to triangles is illustrated by the light blue rotated square in Figure 13.18. The map is

$$g_i(\mathbf{u}) = \begin{cases} p_k, & \text{if } u = 1 \\ w\, p_i + v\, p_j + u\left(\frac{w}{v+w} p_k + \frac{v}{v+w} p_h\right), & \text{otherwise} \end{cases} \tag{13.17}$$

where $\mathbf{u} = (u,v,w)$, $u+v+w = 1$, and where the indices i, j, k, h depends on the partition, for example if Δ_1 then $(i, j, k, h) = (5, 6, 10, 11)$ and if Δ_5 then $(i, j, k, h) = (6, 5, 1, 0)$.

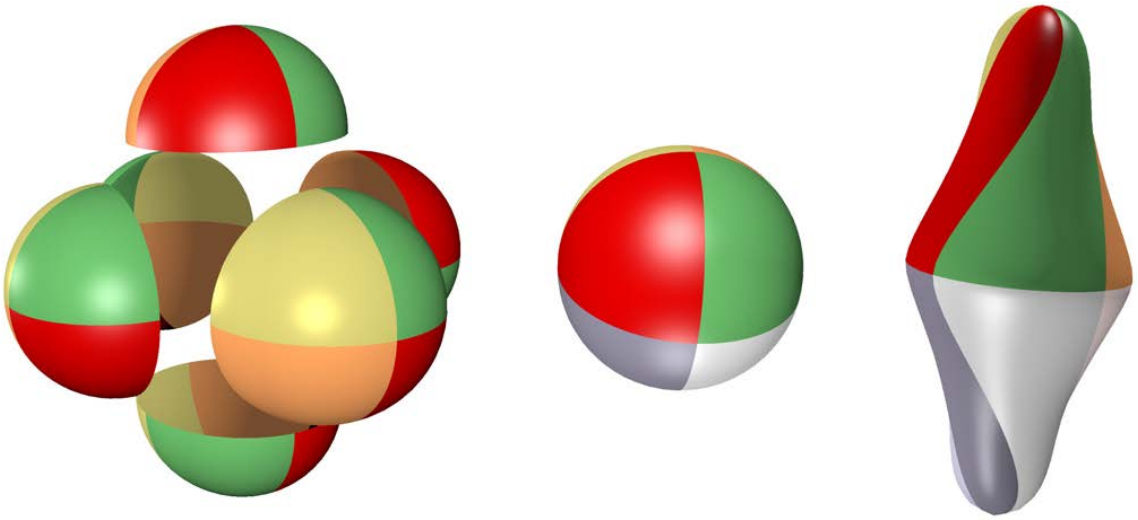


Figure 13.19: *On the left side, there are 6 sub-surfaces of a sphere. They are moved slightly apart so that they can be more easily seen. In the middle is a surface that is a collection of 8 blending triangles and an exact copy of a sphere. To the right we see the same surface, but where the sub-surfaces at the north and south poles are moved apart and then rotated around the vertical center axis.*

Although according to Figure 12.19 we have two types of parameter domains for the sub-surfaces, Ω_{p_a} and Ω_{p_b} , these will be mapped to the same, ie the light blue parameter domain Ω_p. The sub-surfaces are just limitations of the domain. If we plot them, we get 6 equal surfaces (hemispheres) that are only oriented differently, as we can see on the left in Figure 13.19. The surfaces there are correctly oriented, but have been moved slightly apart so that they can be separated from each other. Each small square in the domain is colored separately and appears as triangles.

The formula for sub-triangles to be used in the blending is

$$\begin{aligned} s_i(\mathbf{u}) &= \mathbb{S} \circ g_i(\mathbf{u}) \\ D_u s_i(\mathbf{u}) &= d\mathbb{S}_{\mathbf{u}}\left(\widetilde{w} p_k + \widetilde{v} p_h\right) \\ D_v s_i(\mathbf{u}) &= d\mathbb{S}_{\mathbf{u}}\left(p_j + \widetilde{u}\widetilde{w}\left(p_h - p_k\right)\right) \\ D_w s_i(\mathbf{u}) &= d\mathbb{S}_{\mathbf{u}}\left(p_i - \widetilde{u}\widetilde{v}\left(p_h - p_k\right)\right) \end{aligned}$$

where

$$\begin{array}{ll} \widetilde{u} = 0, \quad \widetilde{v} = 0, \quad \widetilde{w} = 1 & \text{if } u = 1, \\ \widetilde{u} = \frac{u}{v+w}, \quad \widetilde{v} = \frac{v}{v+w}, \quad \widetilde{w} = \frac{w}{v+w} & \text{otherwise.} \end{array}$$

Note that $S_u(p_i) = (0,0,0)^T$, $i = 0,1,2,3,4,10,11,12,13,14$, ie at the north and south poles. Since the "pole point" in each triangle corresponds to two points in the parameter plane, we use S_v in both points, thus we can set $S_u(p_k) = S_v(p_h)$. To make a blending triangle we use (13.14) and (13.15) where the homogeneous matrices are connected to 3 of the 6 points in the center of each hemisphere that appears as corners of the triangle. One example, in the middle of Figure 13.19, there are 8 blending triangles that together form an exact copy of a sphere. On the right side of Figure 13.19, the hemispheres at the north and south poles are moved apart and rotated 90° around the vertical center axis.

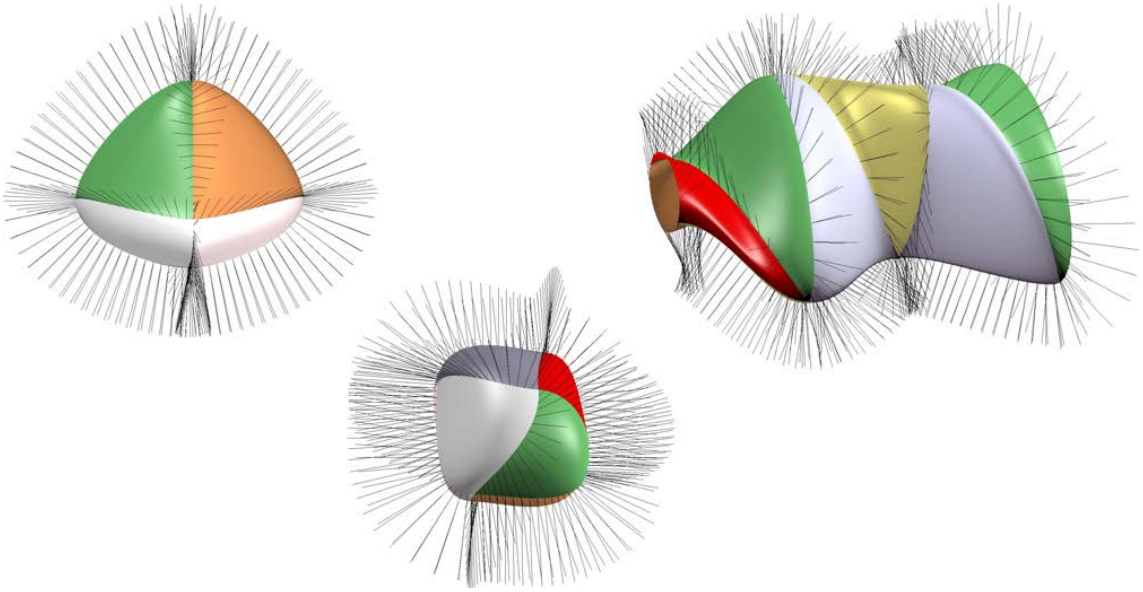


Figure 13.20: *Three blending surfaces by triangulation are shown. Unit normals along the edges of each of the triangles in the surfaces are also displayed. Two of the surfaces are based on a sphere and one is based on a cylinder. This shows continuity properties.*

One important property is continuity. The construction is obviously continuous, but when it comes to derivatives the construction only guarantees continuity up to the Hermite-order of the B-function, and this only in the vertices (the points to which the sub-surfaces and the matrices are connected). This follows from properties of the B-function. In Figure 13.4 we see a triangular B-function. At the vertex where the value is 1, and at the edge opposite to this top point, where the value is 0 everywhere, all the derivatives up to the Hermite order S are 0. The question is actually how the parametrisation act close to the edges. Because when triangles are deformed, directional derivatives can change direction differently on each side of en edge, and we will see that a kink occurs. The only way to prevent this is to get a parametrization that is such that the parameter lines (when the ratio between two parameters are locked) coincide and are preferably perpendicular to the edge.[2] Figure 13.20 shows three blending surfaces by triangulation. Unit normals along the edges of each of the triangles are also displayed. Two of the surfaces are based on a sphere where the vertices are moved and / or rotated (but different in the two examples). The third object is a deformed cylinder. A closer examination of the cylinder example shows us that the normals coincide in the vertices and sometimes at one point on the edges between the vertices. This is where the "parameter line" direction coincides on each side of the edge. The most interesting, however, are the two sphere examples. They show that on the edges between the northern and southern hemispheres, the normals always coincide. This is actually a result of the reparametrization done in (13.17).

Figure 13.21 shows three plots of the same triangular surface (1/4 of a hemisphere). The 3 "parameter lines" (where one parameter varies and the ratio between the other two is constant) are displayed in blue. The surfaces including the parameter lines are rotated

[2]This was first communicated in response to a question from Malcolm Sabin in may 2007.

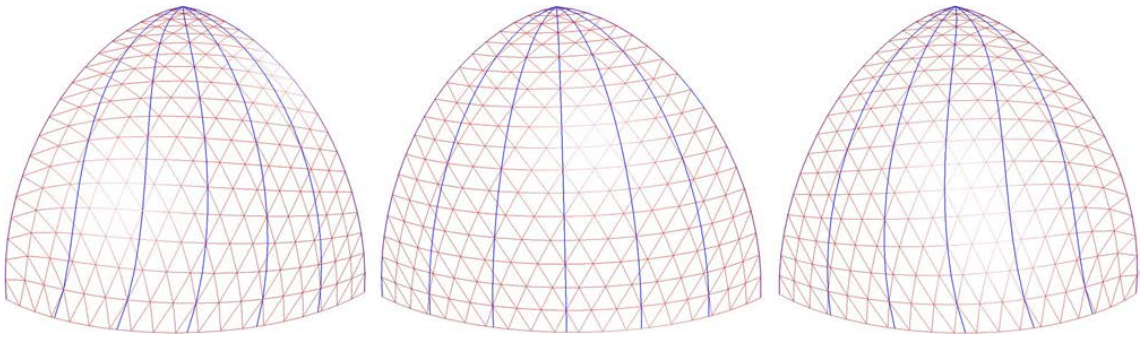


Figure 13.21: *Three plots of the same triangular surface (1/4 of a hemisphere) are displayed. From left and to the right they are rotated clockwise relative to each other. Seven "parameter lines" are displayed in each of the plots. We see that it is only in the middle surface that the "parameter lines" come perpendicular to the edge.*

clockwise from left to right in the figure relative to each other. To the left of the figure we see the w-lines, in the middle the u-lines , and to the right are the v-lines. We see that only the u-lines are perpendicular to the edge. The surface in Figure 13.21 is actually the sphere from Figure 13.19 and 13.20. The u-lines are those that come towards the "equator", and what we see is actually the reason why the surface is smooth over the equatorial edges.

A lot of work has gone into trying to find a general method for reparametrization that provides full continuity along all the edges. So far, the results have not been particularly successful. However, it may be possible to find this for specific triangular surfaces in a sub-surface concept, but this is still an open task, and it is a questionable whether it is theoretically possible, but this author does not know of any proof of this. Surface approach with triangles is possibly most useful to use in connection with free-form modeling in an artistic context and / or more spectacular product design. One method is to start with a topological object that matches the desired end result and then decide on the number of triangles / degree of fineness. Then we can create a composite surface and then change shape interactively by manipulating the interpolation points. From now on, there are two possibilities, either to make the result smooth by introducing the dual set of surfaces, see Chapter 14, or to tessellate / generate a refined triangle structure which then goes into a subdivision algorithm, ie use the Loop method for subdivision surfaces, see Section 12.3.

Figure 13.22 shows more examples of modeling with surface approximation with triangles. All examples are based on the sphere approximation defined in this section. The 5 surfaces at the top right are mainly based on rotations of the points (sub-surfaces), i.e. rotation has been added to the homogeneous matrices H_i, described in (13.14) and (13.15). The gray surface at the bottom left is only based on the points being moved apart. The three copper-colored surfaces are made with larger movements and rotations. The surface at the bottom left is self intersecting. This is how we get the special effect on the "nose".

There has been some focus on putting together Bézier triangles in a smooth way. The problem is that it reduces of degrees of freedom very much, so that the construction becomes rigid and not very formable. Some basic theory about this can be found in [101]. Recently, work has been done on this in connection with an isogeometric approach.

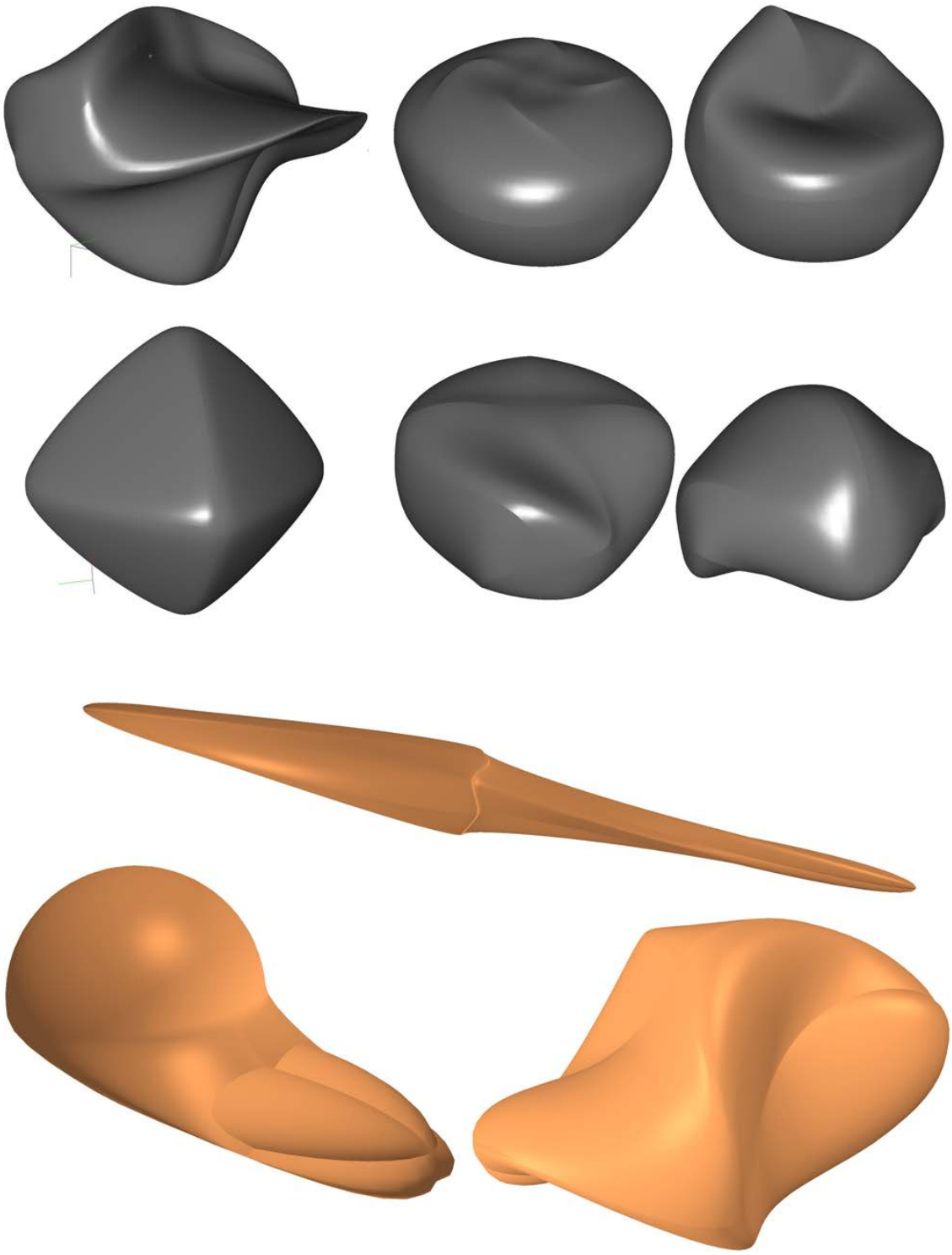


Figure 13.22: *All surfaces we see in the figure are based on the sphere shown in figure 13.19. The only changes are that the interpolation points are moved and rotated. Note that all rotations are performed around the corresponding interpolation point.*

Chapter 14

A Dual Surface Construction

In the previous chapter, Section 13.6, we were introduced to *surface approximation by triangulations*. A concept that is well suited for modeling and design, but which has one disadvantage. The surfaces are continuous, but there is no guarantee of higher order continuity over the edges. In the vertices, on the other hand, the surface is continuously up to the Hermite order of the B function used. However, if we want a higher degree of continuity over the edges, this can be done by introducing a dual set of square surfaces over the edges that we want to make smooth (introduced in [108]).

If, on surfaces composed of a set of triangular surfaces, we have an edge we want to make smooth, then we can insert a point in the middle of the parameter domain of each of the two associated triangular surfaces, ie $p_c = \frac{1}{3}(p_u + p_v + p_w)$, where p_u is the vertex where $u = 1$, p_v is the vertex where $v = 1$ and p_w is the vertex where $w = 1$. In each of the two original triangles we then get three smaller triangles; $\Delta(p_u, p_v, p_c)$, $\Delta(p_c, p_v, p_w)$ and $\Delta(p_u, p_c, p_w)$, and where the formula for each of them becomes:

$$
\begin{aligned}
s_{uvc}(u,v,w) &= s\left(u + \frac{w}{3}, v + \frac{w}{3}, \frac{w}{3}\right), \\
s_{cvw}(u,v,w) &= s\left(\frac{u}{3}, v + \frac{u}{3}, w + \frac{u}{3}\right), \\
s_{ucw}(u,,w) &= s\left(u + \frac{v}{3}, \frac{v}{3}, w + \frac{v}{3}\right).
\end{aligned}
\tag{14.1}
$$

Here s is the original triangular surface and s_{uvc}, s_{cvw} and s_{ucw} are the triangular surfaces from the subdivision. Remember to use the kernel rule for the partial derivatives.

The subdivisions are illustrated in Figure 14.1. We have the parameter domain of two neighboring triangles that share an edge to be smoothed. In both triangles, a new point is inserted in the center. Both triangles are then divided into 3 smaller triangles. In the center of the figure we see a red dashed square $\square(p_4, p_6, p_2, p_3)$. The diagonal is displayed in solid red, and is the edge to be smoothed.

Recall that in the triangular construction, the surface has a continuity in the vertices that is similar to the Hermite order of the B-function used. This means that all directional derivatives of order up to the Hermite order of the B-function are equal in all triangular

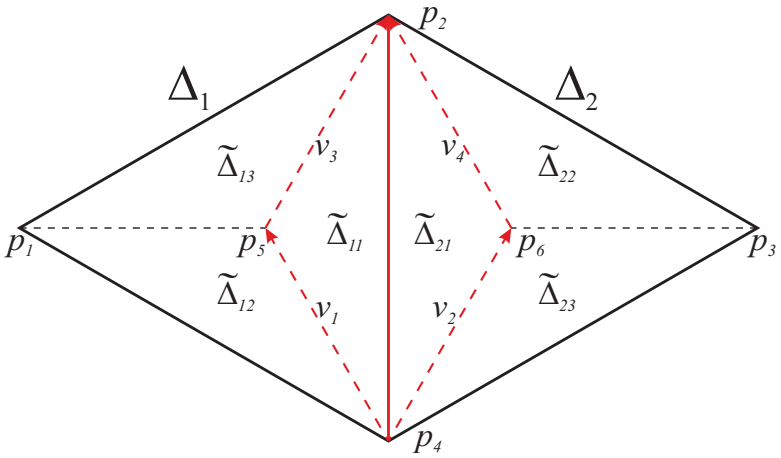


Figure 14.1: *The figure shows two triangles, Δ_1 with the vertices p_2, p_4, p_1 and Δ_2 with the vertices p_2, p_3, p_4. The center points $p_5 = \frac{1}{3}(p_2+p_4+p_1)$ and $p_6 = \frac{1}{3}(p_2+p_3+p_4)$ are marked, as are the vectors v_1, v_2, v_3, v_4. The barycentric coordinates of the points are $p_1 = (0,0,1)$ in Δ_1, $p_2 = (1,0,0)$ in both Δ_1 and Δ_2, $p_3 = (0,1,0)$ in Δ_2, $p_4 = (0,1,0)$ in Δ_1 and $p_4 = (0,0,1)$ in Δ_2. All six sub-triangles are marked.*

patches that share a vertex, in the vertex. Thus, we can fill the square area, marked with red dashed lines in Figure 14.1, with a surface, using either "Coons patch - bicubic blending" shown in Section 9.6.2, if G^1-smoothness is accepted, or using "Two surface blending" shown in Chapter 11 if a higher order of smoothness is required. If there are one or two edges in a triangle that are not to be smoothed, we only use the sub-triangles defined in (14.1).

Regardless of the choice of surfaces,, Coons patch or Two surface blending, we need boundary curves and vector functions that describe the directional derivatives across the boundaries along the boundaries. In Figure 14.1 is both the sub-triangles $\widetilde{\Delta}_{11}$ and $\widetilde{\Delta}_{21}$ and the vectors v_1, v_2, v_3 and v_4 highlighted.

14.1 Curves and vector fields on triangular surfaces

Figure 14.1 shows two triangles that share an edge, Δ_1 and Δ_2. These two triangles are the domains of two triangular surfaces, S_1 and S_2. In triangle Δ_1 we see two vectors, $v_1 = p_5 - p_4$ and $v_3 = p_2 - p_5$. Furthermore, we define two curves,

$$h_1(t) = p_4 + t\, v_1 \quad \text{and} \quad h_3(t) = p_5 + t\, v_3, \quad t \in [0,1]. \tag{14.2}$$

In triangle Δ_2 we see two vectors, $v_2 = p_6 - p_4$ and $v_3 = p_2 - p_6$. Furthermore, we define two curves,

$$h_2(t) = p_4 + t\, v_2 \quad \text{and} \quad h_4(t) = p_6 + t\, v_4, \quad t \in [0,1]. \tag{14.3}$$

Remember that in the parameter domain of the triangles, both the points and the vectors are in barycentric coordinates that relate to the vertices. Therefor, note that in this example is $u = 1$ in p_2 in both triangles, which will not always be the case. Using this location will

$p_4 = (0,1,0)$ in Δ_1, and $p_4 = (0,0,1)$ in Δ_2. This corresponds to the sphere examples in section 13.6. If we now use barycentric coordinates in the curve equations we get

$$h_1(t) = \frac{1}{3}\begin{pmatrix} t \\ 3-2t \\ t \end{pmatrix}, \quad h_2(t) = \frac{1}{3}\begin{pmatrix} t \\ t \\ 3-2t \end{pmatrix}, \quad h_3(t) = h_4(t) = \frac{1}{3}\begin{pmatrix} 1+2t \\ 1-t \\ 1-t \end{pmatrix},$$

and from (14.2) and (14.3) it follows that $h_1' = v_1$, $h_2' = v_2$, $h_3' = v_3$, and $h_4' = v_4$.

The next step is the mapping into $\mathbb{R}^3$. We have the triangular surfaces $S_j : \Delta_j \subset \mathbb{R}^2 \to \mathbb{R}^3$, j=1,2. The curve definition will be,

$$c_i(t) = S_j \circ h_i(t), \quad j = 1 \text{ and } i = 1,3 \quad \text{or} \quad j = 2 \text{ and } i = 2,4, \tag{14.4}$$

and it follows that the 1^{st} and 2^{nd} derivatives for all four curves are

$$\begin{aligned} c_i'(t) &= d(S_j)_{h_i(t)}(h_i'(t)), \\ c_i''(t) &= d(d(S_j)(h_i'(t)))_{h_i(t)}(h_i'(t)) + d(S_j)_{h_i(t)}(h_i''), \\ &= d(d(S_j)(h_i(t)'))_{h_i(t)}(h_i'(t)), \end{aligned} \tag{14.5}$$

because $h'' = 0$, and where

$$\begin{aligned} dS &= [S_u\ ,\ S_v,\ S_w]\,, \\ d(dS(h')) &= d\left([S_u\ ,\ S_v,\ S_w]\,(h')\right) \\ &= \left[[S_{uu}\ ,\ S_{uv},\ S_{uw}]\ h'\ ,\ [S_{vu}\ ,\ S_{vv}, S_{vw}]\ h',\ [S_{wu}\ ,\ S_{wv}, S_{ww}]\ h'\right]. \end{aligned}$$

Vector functions on triangular surfaces are the next topic. The vector valued functions in the parameter planes related to the four boundary curves are (for $t \in [0,1]$) ,

$$\begin{aligned} g_1(t) &= \tilde{v}_2(t) + B(t)\,(v_3(t) - \tilde{v}_2(t))\,, \quad \text{in } \Delta_1, \\ g_2(t) &= \tilde{v}_1(t) + B(t)\,(v_4(t) - \tilde{v}_1(t))\,, \quad \text{in } \Delta_2, \\ g_3(t) &= v_1(t) + B(t)\,(\tilde{v}_4(t) - v_1(t))\,, \quad \text{in } \Delta_1, \\ g_4(t) &= v_2(t) + B(t)\,(\tilde{v}_3(t) - v_2(t))\,, \quad \text{in } \Delta_2, \end{aligned} \tag{14.6}$$

where $B(t)$ is a unary B-function of Hermite order equal to the one used in the blending triangles. $\tilde{v}$ means that the vector must be expressed in coordinates connected to the domain of the neighbouring triangle.

To find the coordinates of a point in a triangle that lies in another triangle, we use the coordinates of the points p_i, $i = 1,2,3,4,5,6$ in the parameter domain of the underlying sub-surface. In Δ_2 we have $p_6 = \frac{1}{3}(p_2 + p_3 + p_4)$, but in Δ_1 we have

$$\begin{aligned} u\ p_2 + v\ p_4 + (1-u-v)p_1 &= p_6, \\ (p_2 - p_1)u + (p_4 - p_1)v &= p_6 - p_1, \end{aligned}$$

and it follows that the barycentric coordinates of p_6 with respect to Δ_1 are

$$u = \frac{(p_6 - p_1) \wedge (p_4 - p_1)}{(p_2 - p_1) \wedge (p_4 - p_1)}, \quad v = \frac{(p_2 - p_1) \wedge (p_6 - p_1)}{(p_2 - p_1) \wedge (p_4 - p_1)} \quad \text{and} \quad w = 1 - u - v,$$

where $a \wedge b$ is the wedge product in $\mathbb{R}^2$, described at the end of Section 2.1.

The four vector valued (derivative) functions in $\mathbb{R}^3$ are thus for $j = 1$ and $i = 1,3$ or $j = 2$ and $i = 2,4$

$$\begin{aligned} e_i(t) &= d(S_j)_{h_i(t)}(g_i(t)), \\ e_i'(t) &= d(d(S_j)(h_i'(t)))_{h_i(t)}(g_i(t)) + d(S_j)_{h_i(t)}(g_i'(t)), \end{aligned} \tag{14.7}$$

where $g(t)$ is defined in (14.6), $h(t)$ in (14.2) and (14.3). Note that compared to the 2^{nd}-derivative in (14.5) is the second term in the 1^{st}-derivative in (14.7) included because $g_i(t)$ is not a linear function due to the B-function.

14.2 The fill-in patch

The next step is to create a surface that fits in the squared area shown as dashed red lines in Figure 14.1. We then need the edge curves and functions that describe derivatives "orthogonally" across the edges.

In Figure 14.1 and from the previous section we see that the curves are organized so that $c_1(t)$ and $c_4(t)$ are on opposite edges of the square surface. Together with the vector functions $e_1(t)$ and $e_4(t)$, described in (14.7)), they can be used to create a surface by blending curves, as described in Section 9.4. We then use Hermite interpolation as shown in (4.16) and (4.17). The result is

$$S_1(u,v) = c_1(v)\, H_1(u) + c_4(v)\, H_2(u) + e_1(v)\, H_3(u) + e_4(v)\, H_3(u). \tag{14.8}$$

In the other direction, $c_2(t)$ and $c_3(t)$ are on opposite edges, and together with $e_2(t)$ and $e_3(t)$ they can also be used to create a surface by blending curves using Hermite interpolation, and the result is

$$S_2(u,v) = c_2(u)\, H_1(v) + c_3(u)\, H_2(v) + e_2(u)\, H_3(v) + e_3(u)\, H_3(v). \tag{14.9}$$

There are now two possible ways to make the final surface, either by using Coons Patch - bicubic blending, described in Section 9.6.2, or by using Two surface blending, described in Chapter 11. In both methods, it is assumed that the corners are consistent from both sides, and this can be seen by comparing (14.5) with (14.7). Remember that the boundary curves follows the vectors v_1, v_2, v_3 and v_4 shown in Figure 14.1 and they also have the same direction. Also remember the properties of the B-function, $B(0) = 0$, $B(1) = 1$ and $B'(0) = B'(1) = 0$. An investigation at each vertex shows that:

At the vertex p_2, where $c_3(1) = c_4(1)$ we see by comparing (14.5) and (14.7) that,

a) $c_3'(1) = e_4(1)$ because $h_3' = v_3$ and $g_4(1) = \tilde{v}_3$,

b) $c_3''(1) = e_4'(1)$ for the same reason as the point above and that $B'(1) = 0$,

c) $c_4'(1) = e_3(1)$ because $h_4' = v_4$ and $g_3(1) = \tilde{v}_4$

d) $c_4''(1) = e_3'(1)$ for the same reason as the point above and that $B'(1) = 0$,

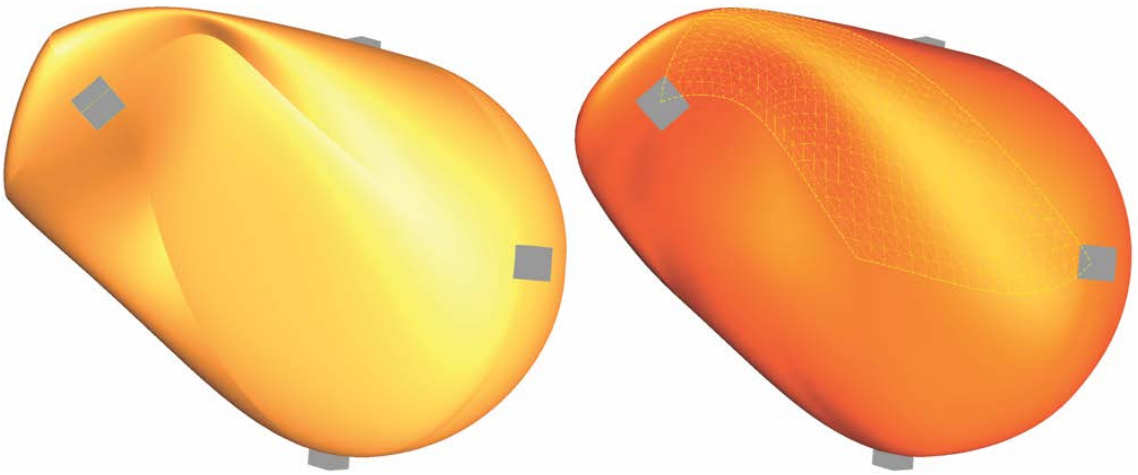


Figure 14.2: *We see two fairly similar surfaces. The one on the left is a surface approximation by triangulation. The surface is initially a sphere (13.16), with a triangulation as described in (13.17). After the generation, the interpolation points are moved and rotated. The interpolation points are displayed as grey cubes. In the surface on the right, the edges are smoothed with the technique described in this chapter. A yellow grid pattern outlines a squere surface between two of the interpolation points, but it is faintly visible.*

At the vertex p_6, where $c_2(1) = c_4(0)$ we see that

a) $c_2'(1) = e_4(0)$ because $h_2' = v_2$ and $g_4(0) = v_2$,

b) $c_2''(1) = e_4'(0)$ for the same reason as the point above and that $B'(0) = 0$,

c) $c_4'(0) = e_2(1)$ because $h_4' = v_4$ and $g_2(1) = v_4$

d) $c_4''(0) = e_2'(1)$ for the same reason as the point above and that $B'(1) = 0$,

At the other two vertices, p_4 where $c_1(0) = c_2(0)$, and p_5 where $c_1(1) = c_3(0)$, we get a similar result Another thing is that the 1^{st}-derivative in (14.7) is actually the twist-derivative at the corner, ie changing the 1^{st}-derivative when we move in the opposite direction. Therefore, in the vertices, we must make sure that these derivatives are equal from both sides. In p_4, $e_1'(0) = e_2'(0)$. By using (14.7) and finding the actual vectors we get $d(d(S_1)(v_1))_{p_4}(\tilde{v}_2) = d(d(S_2)(v_2))_{p_4}(\tilde{v}_1)$. Remember that it is a common underlaying surface that makes that all directional derivatives of S_1 and S_2 are equal, and that the differential is symmetry, ie $S_{uv} = S_{vu}$. The same we will observe in all vertices. This shows that the system is consistent in the vertices.

This little calculation stunt, to show the consistency of the vertices, really only shows the property of the construction "Surface approximation by triangulation" which is that the total surface interpolates the vertices with position and all derivatives up to the Hermite order of the B-function used.

Now the two methods. First Coons patch - bicubic blending that is $S(u,v) = S_1(u,v) + S_2(u,v) - S_3(u,v)$ where S_1 is (14.8) and S_2 is (14.9) and S_3 is a tensor product Hermite

surface, see Section 9.5.1, where we have a 4×4 matrix to fill,ie

$$M = \begin{bmatrix} c_1(0) & c_1(1) & c_1'(0) & c_1'(1) \\ c_4(0) & c_4(1) & c_4'(0) & c_4'(1) \\ c_2'(0) & c_3'(0) & e_1'(0) & e_1'(1) \\ c_2'(1) & c_3'(1) & e_4'(0) & e_4'(1) \end{bmatrix}.$$

The other method is the Two surface blending where we only use $S_1(u,v)$ and $S_2(u,v)$ together with a two variable B-function $B(u,v)$ described in Section 11.1.

An example is given in Figure 14.2. This is the sphere example from Section 13.6. Some of the 6 vertices have been moved and rotated. To the left in Figure 14.2 we see the result. We also see edges that are not smooth. To the right of the figure we see the result after smoothing the edges. Quite faintly we can also see an example of a square patch over an edge made by two surface blending. The patch is marked with a yellow grid pattern. The interpolation points (vertices) are shown as gray cubes.

Appendices

Appendix A

Computing ERB-function type 1

An "evaluator" for an Expo-Rational B-function of type 1 is a computation of $B(t)$, and thus the integrals in (7.31). The integral is of the exponential function $\phi(t)$ defines in (7.27) or $\phi(t;\gamma)$ defined in (7.38) or $\phi(t;\gamma,\mu)$ defined in (7.39) or $\phi(t;\gamma,\mu,\alpha,\beta)$ defined in (7.44).

In addition, the computation of $B^{(j)}(t)$ for $j = 1,2,...,d$ for some d, described in subsection 7.7.4, is required, which includes computation of $f_j(t)$. In practical computations, d ranges between 0 and 4. All this involves handling overflow and underflow and thus division by zero in the fractions, stable and precise numerical integrations and methods for speeding up the computations.

You will find a more detailed description of this in [102], Chapter 3. page 47 which can be loaded from http://urn.nb.no/URN:NBN:no-15022

This appendix considers the implementation, programming, and problems related to the number system of the computer. In the first section we will investigate the requirements for a reliable algorithm. The questions are overflow, underflow and division by zero, and we will investigate these using "IEEE binary floating point" standardized devices. The second section is treating algorithms for the derivatives. Then in the third section, A.3, we will investigate an algorithm for numerical integration of the integral

$$\int_{s=0}^{t} \phi(t;\gamma,\mu,\alpha,\beta)ds, \quad \text{where} \quad 0 < t \leq 1,$$

In [44], section 6, a sequence of numerical methods for computing "ERBS" were considered. The type 1 expo-rational B-function is just the same, and here we study only the simplest of the methods, because we will use an algorithm with fine control of the precision and a simple implementation.

In the fourth section we will show an implementation and a test of a precise and extremely fast evaluator. This evaluator is based on preevaluation and Hermite interpolation.

A.1 Reliability in computations

The reliability of an algorithm depends on possibilities for overflow, underflow and division by zero. We therefore start by looking at what these three phenomena are and when they occur. The IEEE standard for Binary Floating-Point Arithmetic [152] describes the formats of floating-point numbers. According to the current standard, binary floating point numbers should be on the form

$$(-1)^s \, 2^E \, (b_0.b_1b_2\ldots b_{p-1}), \tag{A.1}$$

where p is the precision and,

$$\begin{array}{ll} s \in \{0,1\} & \text{(binary)}, \\ E \in \{E_{min},\ldots,E_{max}\} & \text{(signed integer)}, \\ b_i \in \{0,1\} & \text{(binaries)}. \end{array}$$

The following table describes how the bits in the numbers are distributed.

type	sign	significant bits (precision)	bits for exponent	sum bits
single precision	1	23	8	32
double precision	1	52	11	64

(A.1) is defining the so-called normal values. In addition, the IEEE standard specifies the following special values for numbers; ± 0 (signed zero), subnormal values, $\pm\infty$ and signaled and quiet NaN (Not a Number). Usually the first significant bit b_0 is 1, because if it is not, one can always, for normal values, obtain it by reducing the exponent E. For subnormal values, this is not the case, because we are now already using $E_{min} - 1$. Therefore, for numbers with subnormal values, the first significant bit is always 0. This fact makes it possible to skip this first bit, and thus raise the precision (number of significant bits), because the separation between normal and subnormal values is well defined without the first significant bit (see the table below). The following table (see [75]) shows us how the 5 different types of values can be separated.

Type	values	implemented exponent	implemented precision
Special values	± 0	$E = E_{min} - 1$	$b = 0$
Subnormal values	$\pm 0.b \times 2^{E_{min}}$	$E = E_{min} - 1$	$b \neq 0$
Normal values	$\pm 1.b \times 2^{E}$	$E_{min} \leq E \leq E_{max}$	
Special values	$\pm\infty$	$E = E_{max} + 1$	$b = 0$
Special values	s/q NaN	$E = E_{max} + 1$	$b \neq 0$

Using this improved precision (skipping the first bit), we get the following number of significant digits in the decimal number system. For normal values, the biggest value for single precision (float) is $2^{24} - 1 = 16777215$, i.e. more than 7 significant digits, and for double precision it is $2^{53} - 1 = 9007199254740991$, i.e. close to 16 significant digits. For subnormal values the number of significant digits is reduced, and is gradually decreasing until there is only 1 binary digit.

To describe overflow, and how it occurs, we first look at the maximum normal value,

$$\text{single - } 1.11\ldots11\times 2^{2^{8-1}-1} = \left(2-2^{-23}\right)2^{127} \approx 3.4028237\, e+38.$$
$$\text{double - } 1.11\ldots11\times 2^{2^{11-1}-1} = \left(2-2^{-52}\right)2^{1023} \approx 1.7976931348623159\, e+308$$

Numbers that becomes larger than this will be set to $\pm\infty$ depending on the sign bit. To describe underflow, and how it occurs, we first look at the minimum normal value.[1]

$$\text{single - } 1.00\ldots00\times 2^{2-2^{8-1}} = 2^{-126} \approx 1.1754944\, e-38$$
$$\text{double - } 1.00\ldots00\times 2^{2-2^{11-1}} = 2^{-1022} \approx 2.22507385850720138\, e-308$$

Values smaller than this are subnormal values with lower precision. Notice that the significant bit will be $0.11\ldots11$ for the first subnormal value, and $0.00\ldots01$ for the last subnormal value. Therefore, the minimum (unsigned) subnormal value is,

$$\text{single - } 2^{2^{-126-23}} \approx 1.4012984\, e-45,$$
$$\text{double - } 2^{2^{-1022-52}} \approx 4.9406564584124654\, e-324.$$

Numbers smaller than this will be set to ± 0 depending on the sign bit. Looking at the numbers above we can see that for both and double precision then

$$\frac{1}{\text{min normal value}} < \text{max normal value}.$$

It follows from this that if the denominator in a fraction has a normal value, and the numerator is ≤ 1 then the fraction will not produce an overflow.

For a closer study of how the number system affects algorithms, we recommend [75]. An edited reprint can be found at https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html.

Summing up:

- Overflow produces a signal and $\pm\infty$, which cannot be legally used further in most computations.
- Underflow first produces subnormal values and then ± 0.
- Fractions might produce a signaled overflow.
  - Division by zero clearly produces a signaled overflow.
  - If the numerator in a fraction is ≤ 1 there will never be an overflow if the denominator is a normal value.

Now, back to a reliable algorithm for an ERB-evaluator. The first critical part is the computation of the fraction in the expression (7.27),...,(7.44), namely,

$$-\gamma\frac{|t-\mu|^{\alpha+\beta}}{t^{\alpha}(1-t)^{\beta}}, \quad \text{where} \quad \gamma,\alpha,\beta>0, \quad \text{and} \quad 0<\mu<1 \quad \text{and} \quad t\in[0,1], \qquad \text{(A.2)}$$

[1] The reason for using a total of 3 less numbers in the exponent than available is that one is used for zero and 2 are used for special and subnormal values as showed in the previous table.

and later also the fraction $f_{j,k}(t)$, $j = 2,3,...$, described in subsection 7.7.4. When we look at the numerator of the fraction in (A.2), we can see that

$$|t-\mu| < 1, \quad \text{because } 0 < \mu < 1 \text{ and } t \in [0,1],$$

and thus

$$|t-\mu|^{\alpha+\beta} < 1.$$

We, therefore, get the following remark and algorithm.

Remark 5. *It follows that it is not possible to get overflow even if the denominator is as small as possible, but still a normal number. The only critical part in the algorithm for computing $\phi(t)$ and expression (A.2) is the underflow of the denominator, i.e., that the number in the denominator is not a normal value.*

In the following we give an algorithm first for the set with all intrinsic parameters, and then for the default set, ie $\gamma = \alpha = \beta = 1$ and $\mu = \frac{1}{2}$.

Algorithm 11. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes $\phi(t;\gamma,\mu,\alpha,\beta)$ from expression (7.44), where γ, μ, α and β are present as states, and $t \in [0,1]$ is the input parameter. First follows a general function, then there is an optimized function for the default set of intrinsic parameters, (7.27).

double ϕ(double t)
    double $d = t^{\alpha}(1-t)^{\beta}$;
    if ($d \neq$ *normal value*) return 0.0; *// Test if underflow*
    else return $e^{-\gamma\frac{|t-\mu|^{\alpha+\beta}}{d}}$;

double ϕ(double t)
    double $d = t(1-t)$;
    if ($d \neq$ *normal value*) return 0.0; *// Test if underflow*
    else return $e^{-\frac{(t-\frac{1}{2})(t-\frac{1}{2})}{d}}$;

Remember that $\phi(t)$ is the 1^{st} derivatives if $B_d(t)$, the Expo-Rational B-function of type 1. The 2^{nd} derivative is shown in (7.27). In [102], page 51 it is shown that it is possible to compute up to 49 derivatives of the Expo-Rational B-function of type 1 with the default set of intrinsic parameters and only test for underflow and still get a valid and reliable result.

A general function that calculates the value and derivatives up to order 7 follows in the next section. For practical use, however, we need a much faster algorithm. This can be done with pre-evaluation, numerical integration and hermite interpolation, and will be given in section A.4.

A.2 ERB-evaluation, computing value and derivatives

Below is a reliable algorithm that calculates the value and up to 6 derivatives of the Expo-Rational B-function that has the default set of intrinsic parameters.

Algorithm 12. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes $D^jB(t),\ j=0,1,...,d,$ *where* $\phi(t)$ *is defined in (7.27). It is an implementation based on section 7.7.4. The algorithm assumes that there are functions to compute both* $\phi(t)$, *and* $\int_{s=0}^{t}\phi(s)ds,\ t\in[0,1]$. *The input variables are:* $t\in[0,1]$ *and* $d\in\{0,1,\ldots,6\}$ *(the number of derivatives to compute). The return is a* "vector⟨double⟩", *where the first element contains* $B(t)$, *and then* $DB(t),\ldots,D^dB(t)$.

vector⟨double⟩ **B (** double *t,* int *d* **)**
vector⟨double⟩ $R(d+1)=0.0$*; // Make a vector for return, size d+1, all elements 0.*
double $\phi=\phi(t)$*; // Call to Algorithm 11, part 2.*
if *(*$\phi\neq$ *normal value)* return *R; // Test if underflow.*
$R=\mathbb{S}$*; // Each of the* $d+1$ *elements equal to* $\mathbb{S}$*, (7.32).*
double $\widetilde{t}=(1-2t)^2$*;*
switch *(d)*
case *6:* $R_6*=\big(\big(\big(\big(\big(\frac{45}{2}\widetilde{t}+\frac{135}{2}\big)\widetilde{t}-\frac{765}{4}\big)\widetilde{t}+75\big)\widetilde{t}+66\big)\widetilde{t}-\frac{85}{2}\big)\widetilde{t}+\frac{15}{4}$*; // see*[2]
case *5:* $R_5*=\big(\big(\big(\big(\frac{15}{2}\widetilde{t}+\frac{45}{4}\big)\widetilde{t}-33\big)\widetilde{t}+\frac{29}{2}\big)\widetilde{t}+\frac{3}{2}\big)\widetilde{t}-\frac{3}{4}$*;*
case *4:* $R_4*=\big(\big(3\widetilde{t}+\frac{3}{2}\big)\widetilde{t}-5\big)\widetilde{t}+\frac{3}{2}$*; // see section 7.7.4*
case *3:* $R_3*=\frac{3}{2}\widetilde{t}^2-\frac{1}{2}$*;*
$R_0*=\int_{s=0}^{t}\phi(s)ds$*; // See algorithm 14 in the next section.*
$R_1*=\phi$;
for *(* int *i=2;* $i\leq d$*; i++)* $R_i*=\frac{\phi}{(2t(1-t))^{2(i-1)}}$*;*
for *(* int *i=2;* $i\leq d$*; i+=2)* $R_i*=(1-2t)$*;*
return *R;*

This algorithm is for the default intrinsic parameters, that is $\phi(t)$ defined in (7.27). If we want to freely choose the set of intrinsic parameters, we must use $\phi(t;\gamma,\mu,\alpha,\beta)$ defined in (7.44). The algorithm is then a little more complex. In Algorithm 12, $f_j(t)$, $j=1,2,...$, defined in section 7.7.4, are dissected, and the numerator and denominator in the fractions are calculated separately. In a general algorithm with non-default intrinsic parameters, this will not be the case. Pages 52-57 in [102] shows algorithms for evaluating $f_2(t)$ and $f_3(t)$ and $B_d(t,\gamma,\mu,\alpha,\beta)$ with 3 derivatives. In the same section, restrictions on the intrinsic parameters to obtain a continuous and thus legal B-function is discussed, and also to show combinations of values on the intrinsic parameters where asymptotic behaviors on the derivatives occurs.

In the following we will look at $f_2(t)$. In [102] it is shown to be continuous if $\alpha+\beta>1$ and that the algorithms can be able to handle asymptotes. Remember also the initial restrictions on the intrinsic parameters from (7.44). Combining this with the formula on

[2]The formulas for case 2, 3 and 4 is showed in section 7.7.4 and formulas for case 5 and 6 can be found in [102], page 33, which can be loaded from http://urn.nb.no/URN:NBN:no-15022.

the pages 28, 29 and 53 in [102] we get

$$f_2(t) = \begin{cases} 0, & \text{if } t = \mu, \\ \mathbb{S}\, x_2(t)\zeta(t), & \text{otherwise.} \end{cases} \tag{A.3}$$

where $\zeta(t)$ is the exponent in (7.44), and

$$x_2(t)\zeta(t) = -\gamma \frac{\alpha+\beta}{t^\alpha(1-t)^\beta}\left(\frac{t-\frac{\alpha}{\alpha+\beta}}{t(1-t)}|t-\mu| + \text{sign}(t-\mu)1\right)|t-\mu|^{\alpha+\beta-1}. \tag{A.4}$$

Remember that $\mathbb{S} = \left[\int_0^1 \phi(t;\gamma,\mu,\alpha,\beta)dt\right]^{-1}$. Now an algorithm computing $f_2(t)$ follows.

Algorithm 13. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes $f_2(t)$ for the ERB-function with non-default intrinsic parameters. γ, μ, α and β are supposed to be present, and $\mathbb{S} = \mathbb{S}(\gamma,\mu,\alpha,\beta)$ must be pre-evaluated. The algorithm is the implementation of (A.3) and (A.4). The input variable is supposed to be $t \in [0,1]$, and it is such that the second line in the algorithm shall guarantee that t in the computation of (A.4) has a normal value on the open segment $(0,1)$.[3]

```
double f2( double t )
    if ( t < 2.3e-308 || t == μ || t == 1 )    // See (A.3), upper part.
        return 0.0;
    double h = (t - α/(α+β)) / (t(1-t)) |t-μ|;   // First part of the second factor from (A.4).
    if ( t < μ )    h -= 1;                    // Last part of second factor (A.4).
    else            h += 1;
    h* = -S γ α+β / (t^α (1-t)^β);             // Inserting S and the first factor of (A.4).
    if ( α+β < 1 )                             // Asymptote at t = μ is present.
        double g = |t-μ|^(1-α-β) / h;          // The inverse of (A.4).
        if ( g ≠ normal value )
            return 0.0;
        else
            return 1/g;
    else if (α+β > 1)                          // Ordinary solution.
        return h |t-μ|^(α+β-1);
    else                                       // Discontinuity at t = μ.
        return h;
```

A comment on the foregoing algorithm; as previously mentioned, what do we do with a possible asymptote when $\alpha+\beta < 1$. The convention of what to do at $t = \mu$ when there is no value to return, is that it returns 0 as it will do for all cases when there is a value.

[3] The guarantee is that we have to introduce practical restrictions on the intrinsic parameters because of the binary number system.

A.3 Using Romberg integration in evaluation

The main part of the evaluation of the ERB-function $B_d(t)$ defined in (7.31) is the integration of $\phi(t)$ defined in (7.27). Here we will examine a reliable and controllable numerical integration, namely the Romberg integration, see [136] and [25]. Romberg integration is based on repeated Richardson extrapolations to eliminate error terms, see [92]. The background for the algorithm is the Euler-MacLaurin integration formula, see [156]. It says, given a function f that is $C^\infty[a,b]$, then the error from a trapezoidal approximation $T_n(f)$ according to the integral $I(f)$ is

$$I(f) - T_n(f) = \frac{1}{n^2}\sum_{j=0}^{\infty} A_j^{(0)} \frac{1}{n^{2j}} = \frac{1}{n^2}A_0^{(0)} + \frac{1}{n^4}A_1^{(0)} + \frac{1}{n^6}A_2^{(0)} + \ldots, \tag{A.5}$$

where $A_j^{(0)}$ are constants. For example, the error formula for the Trapezoidal and Simpson method gives

$$A_0^{(0)} = \tfrac{-(b-a)^2}{12}(f'(b) - f'(a)),$$
$$A_1^{(0)} = \tfrac{(b-a)^4}{180}(f^{(3)}(b) - f^{(3)}(a)).$$

Richardson extrapolation can be used in conjunction with (A.5) to iteratively eliminate terms in the error formula. Richardson extrapolation is in general a method to improve an approximation by combining two equations using different step sizes. If we make two simplified versions of (A.5), using step size h instead of the number of steps n, and using half step size (h/2) on the second one, we get

$$I(f) = T_h(f) + A_0 h^{2j} + O(h^{2j+1}),$$
$$I(f) = T_{h/2}(f) + A_0 \left(\tfrac{h}{2}\right)^{2j} + O(h^{2j+1}).$$

If we put this into order and subtract the second from the first equation we get

$$B(h) = \frac{2^{2j}T_{h/2}(f) - T_h(f)}{2^{2j} - 1}, \tag{A.6}$$

where

$$I(f) = B(h) + O(h^{2j+1}).$$

If we also compute $B(h/2)$ we can use $B(h)$ and $B(h/2)$ in a next step (in (A.6)) to reduce the error terms. This can be repeated in an iterative process until the "removed error term" is smaller than a given tolerance.

There are three questions appearing when using Romberg integration to numerically integrate $\phi(t)$,

1. reliability,
2. efficiency,
3. precision.

curve name		**a**	**b**	**c**	**d**	**e**	**f**
upper limit		0.5	0.25	0.125	0.0625	0.03125	0.015625
1	1	$7.2e-2$	$1.4e-2$	$8.3e-3$	$7.7e-4$	$7.3e-6$	$1.2e-9$
2	2	$2.0e-2$	$2.6e-3$	$7.9e-4$	$1.1e-5$	$1.4e-6$	$3.2e-10$
3	4	$1.3e-3$	$9.3e-4$	$7.0e-5$	$1.6e-6$	$1.8e-8$	$8.4e-11$
4	8	$9.5e-4$	$8.5e-5$	$2.9e-6$	$6.9e-8$	$1.6e-9$	$4.0e-12$
5	16	$8.9e-5$	$3.3e-6$	$8.0e-8$	$1.7e-9$	$1.9e-11$	$1.6e-14$
6	32	$3.4e-6$	$8.3e-8$	$1.8e-9$	$2.0e-11$	$4.8e-14$	$2.2e-17$
7	64	$8.4e-8$	$1.8e-9$	$2.0e-11$	$5.3e-14$	$2.8e-17$	
8	128	$1.9e-9$	$2.0e-11$	$5.4e-14$	$2.4e-17$		
9	256	$2.0e-11$	$5.5e-14$	$1.9e-17$			
10	512	$5.5e-14$	$2.8e-17$				
11	1024	$1.7e-16$					

Table A.1: *The table shows the connection between the number of steps/evaluations and the precision in the Romberg-integration algorithm. There are 6 "graphs" computed. Each graph is an integration from 0 to upper limit (second row). The column on the left hand side shows the number of steps in the integration, the next column shows the number of new computations of $\phi(t)$ that have to be done on the current step. All the other numbers are the last correction, indicating the level of tolerance (remaining errors).*

No attempt is made here to compare Romberg integration with other methods, but an investigation of how suitable Romberg integration is for integrations of $\phi(t)$ has been made, and some tests have been done on several integration intervals to find out:
- how fast the quadrature process converges,
- the number of calculations of $\phi(t)$ used,
- and the remaining errors.
The result can be seen both in Table A.1 and in Figure A.1.

First the reliability here depends not only on the fact that $\phi(t)$ is bounded on $[0,1]$, but also on the fact that we do not get an overflow when we computing $\phi(t)$ using algorithm 11, Remark 5 state this. Since also the integration interval is within $[0,1)$, the computation of the integral will always give a normal value, and is in this sense reliable. The degree of reliability, the error, will be discussed below.

In Table A.1 and in Figure A.1 we can see the connection between the number of steps and then evaluations in the Romberg-integration and the "last removed error term". Six evaluations, ie integral, are computed. In the computation, the default set of intrinsic parameters is used, and six different values are computed. The computations are:

$$\int_{s=0}^{t} \phi(s)ds, \quad \text{for} \quad t = \left\{ \frac{1}{2^k} \right\}_{k=1}^{6}.$$

Each computation is named by a letter, "**a**" means that the integration interval is $\left[0, \frac{1}{2}\right]$. This is the computation over the biggest interval. The interval is then halved from compu-

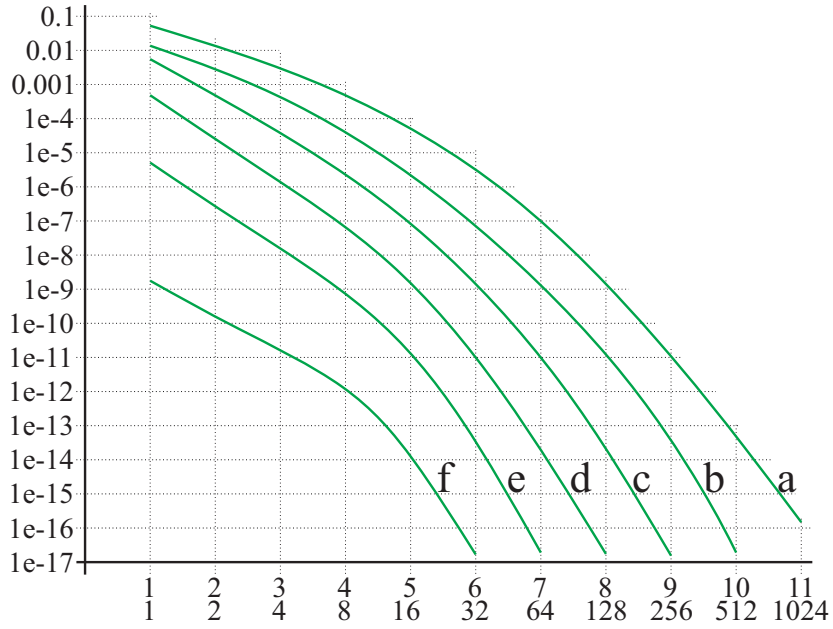


Figure A.1: *The figure is a plot of Table A.1. There are 6 integrals,* **a, b, c, d, e** *and* **f**. *The integrals is from $t = 0$ to t =upper limit, ie $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{32}$, $\frac{1}{64}$. The vertical axis indicates the error and the vertical axis is the number of steps,the upper number, and the nuber of new computations, the lower number.*

tation to computation until the last computation, named "**f**", where the integration interval is $\left[0, \frac{1}{64}\right]$. In Table A.1 we will find the value of all "remaining errors" until they are "out of significant bits". The values are actually the differences between the result from this step in the iteration and the result from the previous step. As we can see in Table A.1, a value in the table is clearly bigger than the sum of all values below in the same column. So the values are actually better than the "remaining errors". We can also see that the last values (on the bottom of the table) are numbers "outside the edge of significant bits", when the results are close to 1. In Figure A.1 is the values from Table A.1 plotted as 6 smooth graphs. On the horizontal axes is the number of new computations of $\phi(t)$ on a logarithmic scale. This tells us that the computational costs are doubled for each mark on the horizontal axes.

The depth of the iteration depends on the size of the integration interval. Evaluating an interval of 0.5 requires up to 11 iterations, which is the same as $2 \times 1024 = 2048$ computations of $\phi(t)$. This is indeed a time consuming process. In this case it is important to be careful; one should not use higher tolerance than necessary. When evaluating numbers bigger than 0.5, one should use mirroring and then instead integrate on a subinterval of the right part of the domain, $\left[\frac{1}{2}, 1\right]$.

The following algorithm is only for integration from 0 to t. To make an algorithm for the general case, integrating from a to b, changes must only be made on line 3, as it must be $\frac{\phi(a)+\phi(b)}{2}$, and in line 9, as it must be s += $\phi(a + j * t)$. In addition, of course, a new declaration introducing "double $t = b - a$;" must be included. The algorithm is optimal in that the number of evaluations of $\phi(t)$ is minimized. In some cases the evaluation on the

boundaries are already known, in these cases the algorithm can easily be adapted either by using state variables or by using parameters.

Algorithm 14. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes the integral $\widetilde{B} = \int_{s=0}^{t} \phi(s)ds,\ t \in (0,1]$, *where* $\widetilde{B}$ *is inside the given tolerance* ε. *The value of the input "tolerance" variable* ε *is recommended to be in* $[1e-2, 1e-15]$ *(according to Table A.1).*

```
double integrate ( double t, double ε )
    double M[16][16];
    double sum = φ(t)/2;                    // Call to Algorithm 11
    M_{0,0} = t * sum;
    for ( int i=1; i < 16; i++ )
        double s = 0;
        int k = 2^i;                        // C++ implementation: k = 1 ≪ i
        t /= 2;
        for ( int j=1; j < k; j+=2 )  s += φ(j*t);    // Call to Algorithm 11
        M_{0,i} = t * (sum += s);
        for ( int j=1; j ≤ i; j++ )
            double c = 4^j;                 // C++ implementation: c = 1 ≪ (j ≪ 1)
            M_{j,i-j} = (c*M_{j-1,i-j+1} - M_{j-1,i-j}) / (c-1);   // Richardson extrapolation scheme
        if ( |M_{i,0} - M_{i-1,0}| < ε )  return M_{i,0};
    return M_{15,0};
```

A.4 Fast ERB-evaluator based on approximations

A fast evaluator for ERB-functions of type 1 is absolutely necessary for the ERB-function to be used in the construction of curves and surfaces. In practical use, it is very important to have a fast but at the same time simple algorithm and also with a simple interface. This because blending splines are especially suitable for interactive design in graphics mode, for simulations where shape change is included and in situations where Hermite interpolation is necessary or preferable.

While it must be fast and easy to use, it must also be reliable and precise. In this section, we shall take a closer look at a method using preevaluation. We divide the domain into, for example, 1024 intervals and then make a 3^{rd}-degree polynomial in each interval using Hermite interpolation.

In practical implementation the whole evaluation system should be wrapped in an "object" (C++ class or equivalent). The advantage of this is that it takes care of all states such as the intrinsic parameters $\phi(t;\alpha,\beta,\gamma,\lambda)$ ie (7.44), the scaling factor $S(\gamma,\mu,\alpha,\beta)$ and, of course, the number of sample intervals, below denoted by m, and all sampled values, more than $6m$ in total, as will be described below. Therefore, in the following, an evaluation object type will be defined, it will be called the "ERB-evaluator" containing the following:

"ERB-evaluator"

The following state variables are present:

$\gamma,\ \mu,\ \alpha,\ \beta$	// Intrinsic parameters (the state identifiers, together with m).
m	// The number of sample intervals, number of samples is $m+1$
$\Delta t = \frac{1}{m}$	// The interval between each sample (also the scaling factor),
	// the sampling vector $\{t_i\}_{i=0}^{m}$ defined via $t_i = i * \Delta t$.
$S = \mathbb{S}(\alpha,\beta,\gamma,\lambda)$	// Scaling factor, $\mathbb{S}(\gamma,\mu,\alpha,\beta) = \int_0^1 \phi(s;\gamma,\mu,\alpha,\beta)ds$.
$\mathbf{b} = \left\{\int_0^{t_i} \phi(s)ds\right\}_{i=0}^{m}$	// Vector storing of the integral $\int_0^{t_i} \phi(s)ds$, where $t_i = i * \Delta t$.
$\mathbf{a}$	// A matrix with dimension $m \times 5$ (actually m vectors).
	// Each of the vectors$\{\mathbf{a}_i\}_{i=0}^{m}$ stores the Hermite coefficients
	// a_0, a_1, a_2, a_3 and a_4 for each sample interval (see A.12).

It will, of course, be necessary to have functions for construction, destruction, settings etc. They will not be handled here, but the important "public" functions are:

$initiate(\gamma,\mu,\alpha,\beta,m)$	// Changing states and thus computing S, $\mathbf{b}$, $\mathbf{a}$ and Δt.
$B(t,d)$	// For a given $t \in [0,1]$, d - number of derivatives,
	// analogous to Algorithm 12, computing $D^j B(t),\ j = 0,\ldots,d$.

For "internal" use (used only by initiate( )) we need the following functions (the three last functions are defined in previous sections):

$interpolate(i,\phi_0,\phi_1,\phi_0',\phi_1')$	// Computing $\mathbf{a}_{i,0},\ \mathbf{a}_{i,1},\ \mathbf{a}_{i,2},\ \mathbf{a}_{i,3}$ and $\mathbf{a}_{i,4}$ (using (A.12)).
$\phi(t)$	// Defined in Algorithm 11.
$f_2(t)$	// Defined in Algorithm 13.
$integrate(t_0,t_1,\phi_0,\phi_1,\varepsilon)$	// Modified version of Algorithm 14. This version uses
	// both the start and end value of the interval, and $\phi(start)$
	// and $\phi(end)$, to minimize the number of evaluations.

A timely question is, how should we use the "ERB-evaluator"? A possible set of answers to this is therefore listed below.

- ✓ For a given set of intrinsic parameters and a given "acceptable error", make an instance of an "ERB-evaluator". All curves and surfaces using this set of parameters can now use this object for evaluation.
- ✓ The parameters or the "acceptable tolerance" can, at any time, be changed for one or more specific curves/surfaces by making and using a new "ERB-evaluator", or by resetting the parameters in the "old" one.
- ✓ It is possible to have more than one "ERB-evaluator" available at the same time.
- ✓ It is possible to have several instances of a curve/surface, the "ERB-evaluator" needs only to be altered.

There is an obvious cost to using "ERBS-evaluators", related to the use of memory. This concerns, however, small numbers, from approximately 1.8 to 48 kbytes for each instance of an evaluator object, depending on the number of samples, and thus the desired errors tolerance. The sample data consist of the vector **b**, storing the incrementing integral sample data, and the matrix **a**, storing, in each line, the Hermite interpolation coefficients at each sampling interval. Only the first four values in each line, $\mathbf{a}_{i,0}$, $\mathbf{a}_{i,1}$, $\mathbf{a}_{i,2}$, $\mathbf{a}_{i,3}$, are actually the Hermite interpolation coefficients. The last one $\mathbf{a}_{i,4}$ is the integral from 0 to 1 of the Hermite interpolant. The reason for having this last element is to either integrate from 0 to ≤ 0.5 or from > 0.5 to 1, so as to reduce the error. The equations for Hermite interpolation between 0 and 1 will now briefly be discussed (we will later look at the problems of the scaling of the derivatives because of domain scaling). We start with a general 3^{rd}-degree polynomial equation,

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3, \tag{A.7}$$

and its first derivative,

$$f'(x) = a_1 + 2a_2 x + 3a_3 x^2. \tag{A.8}$$

Integrating (A.7) from 0 to a given value $\widehat{t} \in (0,1)$ gives the following result

$$\int_{x=0}^{\widehat{t}} f(x) = \widehat{t}\left(a_0 + \widehat{t}\left(\frac{a_1}{2} + \widehat{t}\left(\frac{a_2}{3} + \widehat{t}\left(\frac{a_3}{4}\right)\right)\right)\right), \tag{A.9}$$

and evaluating (A.7) at $\widehat{t}$ yields

$$f(\widehat{t}) = a_0 + \widehat{t}\left(a_1 + \widehat{t}\left(a_2 + \widehat{t}(a_3)\right)\right). \tag{A.10}$$

Then evaluating (A.8) gives

$$f'(\widehat{t}) = a_1 + \widehat{t}\left(2a_2 + \widehat{t}(3a_3)\right). \tag{A.11}$$

If we introduce $f(0)$, $f'(0)$, $f(1)$ and $f'(1)$ as known, we can solve the system according to $\{a_i\}_{i=0}^{3}$. If we, in addition, also include a_4 (the integral on the whole interval, i.e. $\int_0^1 f(x)dx$), the solution is

$$\begin{aligned} a_0 &= f(0), \\ a_1 &= f'(0), \\ a_2 &= 3\left(f(1) - f(0)\right) - f'(1) - 2f'(0), \\ a_3 &= -2\left(f(1) - f(0)\right) + f'(1) + f'(0), \\ a_4 &= \tfrac{f(0)+f(1)}{2} + \tfrac{f'(0)-f'(1)}{12}. \end{aligned} \tag{A.12}$$

The next step is to use these four expressions (A.9), (A.10), (A.11) and (A.12) in the evaluator. Recall that scaling of the domain influences the derivatives and the antiderivatives (integrals). Assume that the real interval is Δt while the expression above are made with an interval of 1. Due to the general rule about scaling the domain the derivatives / antiderivatives must be scaled / inversely-scaled respectively,see [49]). Thus, we need to make the following three adjustments:

- The input derivatives $f'(0)$ and $f'(1)$ must both be scaled by Δt,

- The output integral, $B(t)$, must be scaled by Δt.
- The output derivative (actual second derivative $D^2B(t)$) must be inversely scaled by Δt.

To complete the description of the "ERB-evaluator" there are still three algorithms that have to be described. The first algorithm is initialization.

Algorithm 15. *(For notation, see section "Algorithmic Language", page 6.) The algorithm computes the internal state variables* Δt, $S = \mathbb{S}(\alpha,\beta,\gamma,\lambda)$, $\mathbf{b} = \{B(t_i)\}_{i=0}^{m}$ *and the matrix* $\mathbf{a}$ *(by using an interpolating function). The algorithm assumes that there are algorithms present to compute* $\phi(t)$ *and* $\phi'(t)$ *(can use* $f_2(t)$ *if* $\mathbb{S} = 1$*) and* $\int_{s=0}^{t}\phi(s)ds$, $0 \le t \le 1$. *The input variables are: The intrinsic parameters* γ, μ, α, β, *and m (the number of sample intervals). There are no return values.*

void **initiate** (double γ, double μ, double α, double β, int m)

```
double φ0, φ1;                          // Value at the start and end of each interval
double f0, f1;                          // Derivative at start and end of each interval
set(γ,μ,α,β,m);                         // Store intrinsic parameters and m in object
Allocate memory for b, a;               // Set size = m+1 for b, and m × 4 for a
Δt = 1/m;                               // Set interval size
S = 1;                                  // Temporary, to use φ'(t) = f2(t)
b0 = φ0 = f0 = 0.0;                     // Defined to be zero (basic properties)
for ( int i=1; i < m; i++ )             // For each sampling interval
    double t = i * Δt;                  // The sampling vector marked t_i in definition
    φ1 = φ(t);                          // Algorithm 11
    f1 = φ1 f2(t);                      // Algorithm 13
    bi = integrate(t − Δt, t, φ0, φ1, 1 × 10^−16);  // Algorithm 14 (using max tolerance)
    interpolate(i − 1, φ0, φ1, Δt f0, Δt f1);  // Updating Hermite coefficients {a_{i−1,k}}_{k=0}^3
    φ0 = φ1;                            // Preparing for the next step
    f0 = f1;
bm = integrate(Δt(m − 1), 1, φ0, 0, 1 × 10^−16); // Algorithm 14 (using max tolerance)
interpolate(m − 1, φ0, 0, Δt f0, 0);    // Updating Hermite coefficients {a_{m−1,k}}_{k=0}^3
for ( int i=1; i ≤ m; i≪=1 )            // These three following lines are introduced
    for ( int j=m; j ≥ i; j− = 1 )      // because there in line 13/17 is not used + =
        b_j + = b_{j−i};                // See below for explanation
S = 1/b_m;
```

In line 13 and 17 **b** += *integrate*(...), incremental adding, should have been used. But then there will have been loss of at least one significant bit because of adding a small number to a bigger (and growing) number. To avoid this, the next three last lines are summing up in a binary way. The algorithm is summing neighbors that are nearly equal. The algorithm is, therefore, constructed to optimize the precision. In the algorithm one can see that both integrate() and interpolate() are called twice. The reason for this, is to avoid calling $\phi(t)$ and $f_2(t)$ for $t = 0$ and $t = 1$, because then they are defined to be 0. In addition, let us recall the scaling of the derivatives: they are all, according to the scaling rules, scaled by Δt in the input of the interpolate() function in lines 14 and 18.

The second algorithm, the interpolation, is a very simple algorithm implementing computation of the Hermite interpolation coefficients (see (A.12)).

Algorithm 16. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes the Hermite interpolation coefficients $\mathbf{a}_{i,0}$, $\mathbf{a}_{i,1}$, $\mathbf{a}_{i,2}$, $\mathbf{a}_{i,3}$ *and* $\mathbf{a}_{i,4}$, *and is an implementation of (A.12).*

void **interpolate (** int *i,* double f_0, double f_1, double f'_0, double f'_1 **)**
$\mathbf{a}_{i,0} = f_0$;
$\mathbf{a}_{i,1} = f'_0$;
$\mathbf{a}_{i,2} = 3(f_1 - f_0) - f'_1 - 2f'_0$;
$\mathbf{a}_{i,3} = -2(f_1 - f_0) + f'_1 + f'_0$;
$\mathbf{a}_{i,4} = \frac{f(0)+f(1)}{2} + \frac{f'(0)-f'(1)}{12}$;

The third and last function is the main "evaluation function" $B()$, analogous to Algorithm 14. Note that $B()$ is a B-function not connected directly to a curve or a surface. If there is a curve or surface behind the call to the B-function, then the mapping $B \circ w_{1,i}(t)$ must be done, see (8.4) and (6.11). And after the call the result mast be scaled by $\delta^j_{1,i}$, where j is the order of the derivatives, see (6.13). Remember that the index i in the knot vector is defined by $t_i \leq t < t_{i+1}$.

Algorithm 17. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes $D^jB(t)$, $j = 0,1,2$ *for the set of intrinsic parameters that are initialized in the object. The algorithm assumes that initializing is done. The input variables are:* $t \in [0,1]$, *and* $d \in \{0,1,2\}$ *(the number of derivatives to compute). The return is a* "vector⟨double⟩", *where the first element contains* $B(t)$, *and then ,depending on d follows* $DB(t)$ *and* $D^2B(t)$.

vector⟨double⟩ **B (** double *t,* int *d* **)**
vector⟨double⟩ $R(d+1) = S$; *// Make a vector for return, size d+1, all elements S*
int j = min(int$(t * m), m-1)$; *// j not equal m, due to the "mirroring" around 0.5*
double $dt = \frac{t - j*\Delta t}{\Delta t}$; *// Mapping from total* $[0,1]$ *to* $[0,1]$ *on sample*
switch *(d)*
case *2:* $R_2 * = \frac{\mathbf{a}_{j,1} + dt\left(2\mathbf{a}_{j,2} + dt\ 3\mathbf{a}_{j,3}\right)}{\Delta t}$; *// Using (A.11), scaled by* $\frac{1}{\Delta t}$
case *1:* $R_1 * = \mathbf{a}_{j,0} + dt\left(\mathbf{a}_{j,1} + dt\left(\mathbf{a}_{j,2} + dt\ \mathbf{a}_{j,3}\right)\right)$; *// Using (A.10), no scaling*
if $(dt > 0.5)$ *// Integrating: dt - 1 (A.12)*
$R_0 * = b_{j+1} - \Delta t\left(\mathbf{a}_{j,4} - dt\left(\mathbf{a}_{j,0} + dt\left(\frac{\mathbf{a}_{j,1}}{2} + dt\left(\frac{\mathbf{a}_{j,2}}{3} + dt\frac{\mathbf{a}_{j,3}}{4}\right)\right)\right)\right)$ *// Scaled by* Δt
else *// Integrating: 1 - dt (A.12)*
$R_0 * = b_j + \Delta t\ dt\left(\mathbf{a}_{j,0} + dt\left(\frac{\mathbf{a}_{j,1}}{2} + dt\left(\frac{\mathbf{a}_{j,2}}{3} + dt\frac{\mathbf{a}_{j,3}}{4}\right)\right)\right)$; *// Scaled by* Δt
return *R;*

The next question is the "evaluation" of the "ERB-evaluator" itself. There are two different evaluations that need to be done: evaluation of the efficiency, and of precision.

The efficiency is clearly the reason for making the whole system. Due to the Hermite interpolation and the integration, with regarding to time consumption, the whole system is

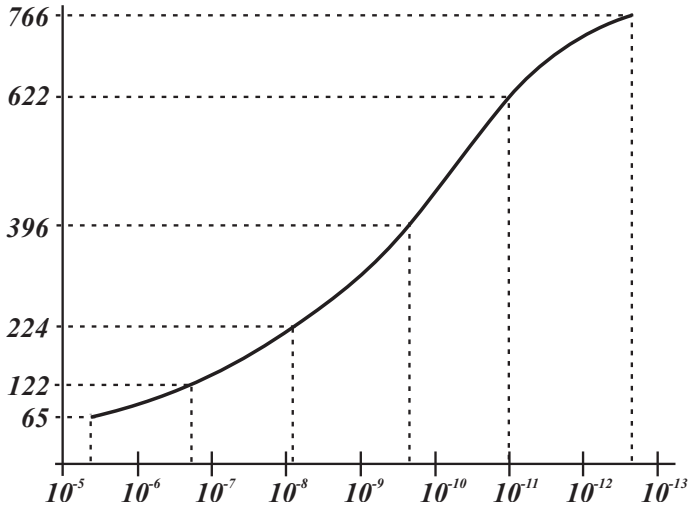


Figure A.2: *The relationship between the precision, ie the tolerance, and the evaluation speed, compared between the use of the $B()$ function in the "ERB-evaluator" and the use of the $B()$ function in Algorithm 12 from section A.2. At a sample rate of 1024, which gives a precision of 3.3e-13, the "ERB-evaluator" is 766 times faster than the evaluator in Algorithm 12, when computing function value and two derivatives.*

identical to evaluating a 4^{th}-degree polynomial function, including the derivatives. This is in some sense a kind of optimal solution. In Figure A.2 there is a graph of the relationship between the use of Algorithm 12 (with two derivatives) and Algorithm 17, the "ERB-evaluator". Algorithm 12 uses the default set of intrinsic parameters, and is, therefore, relatively fast. The speed of the $B()$ in the "ERB-evaluator" is itself independent of the sample rate, but the precision is highly dependent on the sample rate. The evaluator $B()$ in Algorithm 12 is very dependent on the precision, especially the integration part using Romberg integration. This can clearly be seen in Figure A.2. On the horizontal axis you can see the precision (tolerance) of the function value, $B(t)$. On the vertical axis you can see the relation between the speed of the "ERB-evaluator" $B()$ and the "old" $B()$. One can clearly see that the difference in speed is tremendous, i.e. it takes up to 766 *times* longer to use $B()$ in Algorithm 12 than $B()$ in the "ERB-evaluator". The figure also indicates that as we pass the tolerance 10^{-13} we approach the maximum of the possible acceleration of the algorithm by replacing with Algorithm 17.

The precision is a more complex problem to deal with, especially since the precision is getting worse with the increase of the order of derivatives. The reason for this is actually easy to see because $B(t)$ is the integral of $DB(t)$, and also because $DB(t)$ is the integral of $D^2B(t)$. Observe that a simplified computation of a maximal error of an integral of a function in a sample interval, is approximately: $\frac{2}{3} \times$ max error of the function $\times \frac{1}{m}$ (the sample interval). The difference in error between an approximation of a function and an approximation of the derivative of the function is, therefor, close to 10^3 for number of samples m=1024. There are, of course, methods to improve this, but they might decrease the speed or the flexibility (this will be discussed further later).

We get an insight into the precision of the "ERB-evaluator" in Table A.2. The table

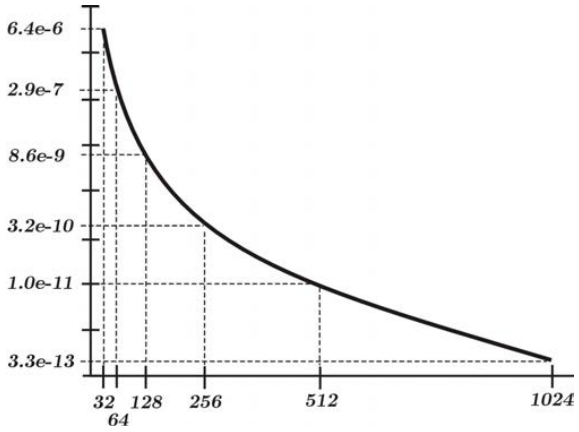


Figure A.3: *The relationship between the number of samples and the precision (max norm) of $B(t)$ (function value) using the "ERB-evaluator". For a sample rate of 32, the precision is 6.4 e-6. For a sample rate of 1024, the precision is 3.3 e-13.*

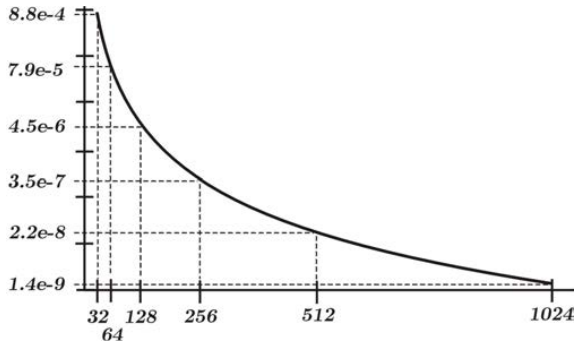


Figure A.4: *The relationship between the number of samples and the precision (max norm) of $DB(t)$ (first derivatives) using the "ERB-evaluator". For a sample rate of 32, the precision is 8.8 e-4. For a sample rate of 1024, the precision is 1.4 e-9.*

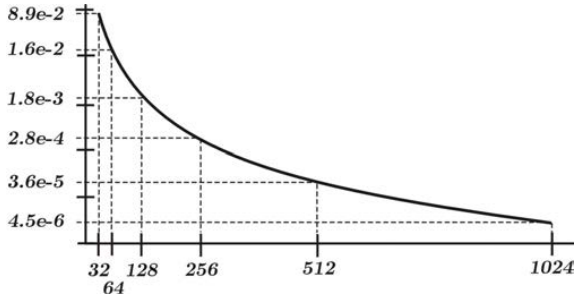


Figure A.5: *The relationship between the number of samples and the precision (max norm) of $D^2B(t)$ (second derivatives) using the "ERB-evaluator". For a sample rate of 32, the precision is 8.9 e-2. For a sample rate of 1024, the precision is 4.5 e-6.*

samples interval		1024	512	256	128	64	32
$B(t)$	$L^1[0,1]$	$7.6e-15$	$2.4e-13$	$7.6e-12$	$7.7e-10$	$8.5e-9$	$2.7e-7$
	$L^2[0,1]$	$2.9e-14$	$9.5e-13$	$3.0e-11$	$9.6e-10$	$3.3e-8$	$9.7e-7$
	$L^\infty[0,1]$	$3.3e-13$	$1.0e-11$	$3.2e-10$	$8.6e-9$	$2.9e-7$	$6.4e-6$
$DB(t)$	$L^1[0,1]$	$4.9e-11$	$7.8e-10$	$1.3e-8$	$2.0e-7$	$3.5e-6$	$5.6e-5$
	$L^2[0,1]$	$1.7e-10$	$2.7e-9$	$4.3e-8$	$6.7e-7$	$1.2e-5$	$1.7e-4$
	$L^\infty[0,1]$	$1.4e-9$	$2.2e-8$	$3.5e-7$	$4.5e-6$	$7.9e-5$	$8.8e-4$
$D^2B(t)$	$L^1[0,1]$	$1.9e-7$	$1.5e-6$	$1.2e-5$	$9.7e-5$	$8.4e-4$	$6.5e-3$
	$L^2[0,1]$	$5.9e-7$	$4.7e-6$	$3.8e-5$	$2.9e-4$	$2.6e-3$	$1.9e-2$
	$L^\infty[0,1]$	$4.5e-6$	$3.6e-5$	$2.8e-4$	$1.8e-3$	$1.6e-2$	$8.9e-2$

Table A.2: *The table shows the connection between the number of sample intervals and the error for the three functions $B(t)$, $DB(t)$ and $D^2B(t)$, using three different norms, $L^1[0,1]$, $L^2[0,1]$ and $L^\infty[0,1]$.*

shows the connection between the number of sample intervals and the error of the ERB-function $B(t)$, and its derivatives $DB(t)$ and $D^2B(t)$. The error is displayed using three different norms. These norms are $L^1[0,1]$ representing an arithmetic mean value, $L^2[0,1]$ representing a geometric mean value, and $L^\infty[0,1]$, a max norm, giving us the guaranteed tolerance. The three figures A.3, A.4 and A.5, show graphs of the error(the max norm) of the "ERB-evaluator" in relation to the number of samples. Figure A.3 shows the error for the function value $B(t)$. It ranges from 6.4×10^{-6} for 32 samples, to 3.3×10^{-13} for 1024 samples. This can be considered as quite a good result. In Figure A.4 the error for the first derivative $DB(t)$ is plotted. It ranges from 8.8×10^{-4} for 32 samples, to 1.4×10^{-9} for 1024 samples. In many cases, this can still be regarded as an acceptable result. In Figure A.5, the errors for the second derivative $D^2B(t)$ are plotted. It ranges from 8.9×10^{-2} for 32 samples, to 4.5×10^{-6} for 1024 samples. This result is significantly closer to the edge of what is acceptable.

The conclusion is thus that for the value, $B(t)$, and the first derivative, $DB(t)$, the errors are acceptable. But the error for the second derivative can be improved, and at the same time we can extend the algorithm to calculate more derivatives. This can be done with general intrinsic parameters, but will mean a much slower algorithm. But for the default set we can extend Algorithm 17 with parts from Algorithm 12 and then get up to 6 derivatives with an acceptable error rate, all derivatives will then get the same error level as $DB(t)$. At the same time we get a relatively fast algorithm.

This leads us to create a specialized ERB-evaluator for an ERB-function with the default set of intrinsic parameters. In C++ -programming, it is natural to make a class inherited from the ERB-evaluator class described on page 291. Much is common, but at least the function $B(t,d)$ is different. Advantageous changes are that $\mathbb{S}$ can be hardcoded and $\phi(t)$ should be made by Algorithm 11, second part.

The main evaluation function *B*(*double* t, *int* d) for an ERB-function with the default set

of intrinsic parameters is:

Algorithm 18. *(For notation, see section "Algorithmic Language", page 6.)*
The algorithm computes $D^jB(t), j = 0,1,2,3,4,5,6$ for the set of default intrinsic parameters. The algorithm assumes that initializing is done. The input variables are: $t \in [0,1]$, and $d \in \{0,1,2,3,4,5,6\}$ (the number of derivatives to compute). The return is a "vector⟨double⟩"*, where the first element contains $B(t)$, and then ,depending on d follows $DB(t)$, $D^2B(t)$, ..., $D^{(j)}B(t)$*

vector⟨double⟩ **B (** double *t,* int *d* **)**

vector⟨double⟩ $R(d+1) = S$; // *Make a vector for return, size d+1, all elements S*

int $j = \min(\text{int}(t*m), m-1)$; // *j not equal m, due to the "mirroring" around 0.5*

double $dt = \frac{t-j*\Delta t}{\Delta t}$; // *Mapping from total* $[0,1]$ *to* $[0,1]$ *on sample*

if $(dt > 0.5)$ // *Integrating: dt - 1 (A.12)*

$R_0 *= b_{j+1} - \Delta t \left(\mathbf{a}_{j,4} - dt \left(\mathbf{a}_{j,0} + dt \left(\frac{\mathbf{a}_{j,1}}{2} + dt \left(\frac{\mathbf{a}_{j,2}}{3} + dt \frac{\mathbf{a}_{j,3}}{4}\right)\right)\right)\right)$; // *Scaled by* Δt

else // *Integrating: 1 - dt (A.12)*

$R_0 *= b_j + \Delta t\ dt \left(\mathbf{a}_{j,0} + dt \left(\frac{\mathbf{a}_{j,1}}{2} + dt \left(\frac{\mathbf{a}_{j,2}}{3} + dt \frac{\mathbf{a}_{j,3}}{4}\right)\right)\right)$; // *Scaled by* Δt

if $(d > 0)$

double $\phi = \mathbf{a}_{j,0} + dt \left(\mathbf{a}_{j,1} + dt \left(\mathbf{a}_{j,2} + dt\ \mathbf{a}_{j,3}\right)\right)$; // *see*[4]

$R_1 *= \phi$; // *Using (A.10), no scaling*

double $\widetilde{t} = (1-2t)^2$;

switch *(d)*

case *6:* $R_6 *= \left(\left(\left(\left(\left(\frac{45}{2}\widetilde{t} + \frac{135}{2}\right)\widetilde{t} - \frac{765}{4}\right)\widetilde{t} + 75\right)\widetilde{t} + 66\right)\widetilde{t} - \frac{85}{2}\right)\widetilde{t} + \frac{15}{4}$;

case *5:* $R_5 *= \left(\left(\left(\left(\frac{15}{2}\widetilde{t} + \frac{45}{4}\right)\widetilde{t} - 33\right)\widetilde{t} + \frac{29}{2}\right)\widetilde{t} + \frac{3}{2}\right)\widetilde{t} - \frac{3}{4}$;

case *4:* $R_4 *= \left(\left(3\widetilde{t} + \frac{3}{2}\right)\widetilde{t} - 5\right)\widetilde{t} + \frac{3}{2}$;

case *3:* $R_3 *= \frac{3}{2}\widetilde{t}^2 - \frac{1}{2}$;

for *(* int *i=2; i* $\le d$*; i++)* $R_i *= \frac{\phi}{(2t(1-t))^{2(i-1)}}$;

for *(* int *i=2; i* $\le d$*; i+=2)* $R_i *= (1-2t)$;

return *R;*

[4]It is also possible to call Algorithm 11, part 2 directly, ie double $\phi = \phi(t)$; The error will be improved to better than 1e-15, but the time cost for just this line is approximately 9:22. That is about 2.5 times as much time. But note that this only applies to this one line.

Appendix B

Programming libraries

Solving linear systems, which are sometimes quite large, is a resource-intensive job for the computer. Therefore, it is wise to use ready-made optimized subroutines for this. There is a defined programming interface for this, BLAS, which is described in the section below. The section also has a list of completed routines that are BLAS compliant.

Another problem associated with resource-intensive programs is being able to use all available resources on the computer. This is called "Heterogeneous computing" and together with parallelization is this dealt with in the section after BLAS.

B.1 Basic Linear Algebra Subprograms - BLAS

Basic Linear Algebra Subprograms - BLAS is a de facto application programming interface standard for publishing libraries to perform basic linear algebra operations such as vector and matrix multiplication. They were first published in 1979, and are used to build larger packages such as LAPACK, and Armadillo[1]. Commercial applications such as MATLAB also use BLAS. BLAS is heavily used in high-performance computing, highly optimized implementations of the BLAS interface have been developed by hardware vendors such as Intel, AMD and NVIDIA for CPU and GPU, as well as by other authors, see below. BLAS homepage can be found at http://www.netlib.org/blas/.

BLAS functionality is categorized into three sets called "levels", which correspond to both the chronological order of definition and publication, as well as the degree of the polynomial in the complexities of algorithms:

Level 1 BLAS operations of $O(n)$. This level consists of all the routines described in the original presentation of BLAS (1979), [111] which defined only vector operations on strided arrays: dot products, vector norms, a generalized vector addition of the form $y \leftarrow \alpha x + y$.

[1]LAPACK, a software library written in Fortran http://performance.netlib.org/lapack/ and LAPACK++ in C++ https://math.nist.gov/lapack++/. Armadillo is another C++ library http://arma.sourceforge.net/. All three are free software

Level 2 BLAS operations of $O(n^2)$. This level contains matrix-vector operations including, among other things, a generalized matrix-vector multiplication $y \leftarrow \alpha Ax + \beta y$ and solving for x, $Tx = y$ with T being triangular. Design of this Level was made in 1984 - 1988, [53].

Level 3 BLAS operations of $O(n^3)$, formally published in 1990, [52]. Contains matrix-matrix operations, including a general matrix multiplication, of the form $C \leftarrow \alpha AB + \beta C$, where A and B can optionally be transposed or hermitian-conjugated inside the routine, and all three matrices may be strided. The ordinary matrix multiplication AB can be performed by setting α to one and C to an all-zeros matrix of the appropriate size. We also find routines for $B \leftarrow \alpha T^{-1}B$ where T is a triangular matrix. There is also other functionalities.

Modern BLAS implementations typically provide all three levels.

Below follows a short (not complete) list of libraries with the BLAS interface:

Accelerate - Apple's framework for macOS and iOS,
https://developer.apple.com/accelerate/

AOCL - is a set of numerical libraries tuned specifically for AMD CPU's
https://developer.amd.com/spack/amd-optimized-cpu-libraries/

ATLAS - Open source implementation - APIs for C and Fortran77,
http://math-atlas.sourceforge.net/

BLIS - for AMD, a BLAS-like dense linear algebra libraries,
https://developer.amd.com/amd-aocl/blas-library/

cuBLAS - Basic Linear Algebra on NVIDIA GPUs,
https://developer.nvidia.com/cublas

NVBLAS - For NVIDIA GPUs, but only Level 3 functions,
https://docs.nvidia.com/cuda/nvblas/index.html

clBLAST - is an OpenCL BLAS library written in C++11,
https://rocmdocs.amd.com/en/latest/ROCm_Tools/clBLA.html

Eigen BLAS - is a C++ template library (free software)
http://eigen.tuxfamily.org/index.php?title=Main_Page

GSL - The GNU Scientific Library for C and C++ (free software),
http://www.gnu.org/software/gsl/

Intel MKL - "oneAPI Math Kernel Library" for Intel CPU's
http://software.intel.com/en-us/intel-mkl

Netlib BLAS - freely available in Fortran, for C see CBLAS,
https://www.netlib.org/blas/

OpenBLAS - is based on GotoBLAS2 1.13 BSD. OpenBLAS is an open source project supported by Lab of Parallel Software and Computational Science,
http://www.openblas.net/

rocBLAS - An implementation on top of AMD's Radeon Open Compute ROCm runtime and toolchains. rocBLAS is implemented in the HIP programming language and optimized for AMD's latest discrete GPUs.
https://rocblas.readthedocs.io/

SurviveGotoBLAS2 - is also based on GotoBLAS2 1.13 BSD, licensed under AGPL-3.
http://prs.ism.ac.jp/~nakama/SurviveGotoBLAS2/

uBLAS - a C++ template class library provides level 1, 2, 3 functionality for dense, packed and sparse matrices.
https://www.boost.org/doc/libs/1_72_0/libs/numeric/ublas

ViennaCL - a free open-source linear algebra library for computations on GPUs and multi-core CPUs. The library is written in C++ and supports CUDA, OpenCL, and OpenMP. It contains core functionality and many other features including BLAS level 1-3 support and also iterative solvers.
http://viennacl.sourceforge.net/

There is a lot of other implementation that can be found if you look on internet, se for example *Basic Linear Algebra Subprograms* at Wikipedia.

B.2 Heterogeneous computing and parallelization

In heterogeneous computing, we use systems that use more than one kind of processor or cores. This can typically be to combine central processing units (CPU) and graphics processor units (GPU). Together with parallelization, ie the use of many similar units/-cores, this is an important way to increase the speed of calculations, especially when large matrices are involved.

CUDA - "Compute Unified Device Architecture", is a parallel computing platform and API model created by Nvidia. We can use a CUDA-enabled GPU for general purpose processing, ie GPGPU. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels (shaders). It can be used together with C, C++ and Fortran. CUDA-powered GPUs also support programming frameworks such as OpenMP, OpenACC and OpenCL, and also HIP by compiling such code to CUDA. https://developer.nvidia.com/about-cuda

HIP - The HPC-ready universal language at the core of AMD's all-open ROCm platform. HIP can run on both AMD and Nvidia GPUs. The HIP API syntax is very similar to the CUDA API, and the abstraction level is the same meaning that porting between the two is easy. https://rocmdocs.amd.com/

OpenMP - The OpenMP API supports multi-platform shared-memory parallel programming in C/C++ and Fortran. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. The core elements of OpenMP are the constructs for thread creation, workload distribution

(work sharing), data-environment management, thread synchronization, user-level runtime routines and environment variables. https://www.openmp.org/

OpenCL - The open standard for parallel programming of heterogeneous systems. It is a framework for writing programs that execute across heterogeneous platforms consisting of central processing units -CPUs, graphics processing units -GPUs, digital signal processors -DSPs, field-programmable gate arrays -FPGAs and other processors or hardware accelerators. OpenCL specifies programming languages (based on C99, C++14 and C++17) for programming these devices and application programming interfaces -APIs to control the platform and execute programs on the compute devices. OpenCL provides a standard interface for parallel computing using task- and data-based parallelism. http://www.khronos.org/opencl/

Appendix C

Miscellaneous proofs

C.1 Newton and Lagrange polynomials, proof Lemma 5.1

Lemma 5.1 states that the Newton and Lagrange polynomials are just two different representations of the same polynomial. Here follows the proof.

Proof. We will show this by induction. First we show it for first degree interpolation,

$$f(t) = \sum_{i=0}^{1} a_i n_i(t) = f(t_0) + (t-t_0)\left(\frac{f(t_0)}{(t_0-t_1)} + \frac{f(t_1)}{(t_1-t_0)}\right) = f(t_0)\frac{(t_0-t_1)+(t-t_0)}{t_0-t_1}$$
$$+ \frac{t-t_0}{t_1-t_0} f(t_1) = \frac{t-t_1}{t_0-t_1} f(t_0) + \frac{t-t_0}{t_1-t_0} f(t_1) = \sum_{i=0}^{1} f(x_i)\, L_{1,i}(t),$$

consequently, the 1^{st}-degree Newton and Lagrange polynomials are the same polynomial. If we have a Lagrange polynomial of degree $d-1$ and then add one more interpolation

point as a Newton polynomial term using (5.6) we get,

$$
\begin{aligned}
f(t) &= \sum_{i=0}^{d-1} f(t_i)\, L_{d-1,i}(t) + f[t_0,\ldots,t_d] \prod_{i=0}^{d-1}(t-t_i) \\
&= \sum_{i=0}^{d-1} \left(f(t_i) \prod_{j=0,j\neq i}^{d-1} \frac{(t-t_j)}{(t_i-t_j)} \right) + \sum_{i=0}^{d} \frac{f(t_i)}{\omega_d'(t_i)} \prod_{j=0}^{d-1}(t-t_j) \\
&= \sum_{i=0}^{d-1} \left(f(t_i) \frac{\prod_{j=0,j\neq i}^{d-1} (t-t_j)(t_i-t_d)}{\prod_{j=0,j\neq i}^{d-1} (t_i-t_j)(t_i-t_d)} \right) + \sum_{i=0}^{d} \left(f(t_i) \frac{\prod_{j=0}^{d-1}(t-t_j)}{\prod_{j=0,j\neq i}^{d}(t_i-t_j)} \right) \\
&= \sum_{i=0}^{d-1} \left(f(t_i) \frac{\prod_{j=0,j\neq i}^{d-1} (t-t_j)(t_i-t_d) + \prod_{j=0,j\neq i}^{d-1} (t-t_j)(t-t_i)}{\prod_{j=0,j\neq i}^{d}(t_i-t_j)} \right) + f(t_d) \prod_{j=0}^{d-1} \frac{(t-t_j)}{(t_d-t_j)} \\
&= \sum_{i=0}^{d-1} \left(f(t_i) \frac{\prod_{j=0,j\neq i}^{d-1} (t-t_j)(t_i-t_d+t-t_i)}{\prod_{j=0,j\neq i}^{d}(t_i-t_j)} \right) + f(t_d) \prod_{j=0}^{d-1} \frac{(t-t_j)}{(t_d-t_j)} = \sum_{i=0}^{d} f(x_i)\, L_{d,i}(t).
\end{aligned}
$$

We see that this gives Lagrange polynomials of degree d, which concludes the proof. □

C.2 Commutativity relations between $T(t)$ and its derivatives

The $T_d(t)$ matrices are matrices containing only linear elements or zeros. It means that T_d', which is the derivative of $T_d(t)$, is a matrix of only constants.

Given that we have a B-spline of degree 3. We will then see that regardless of which of the 3 matrices we derive, the result will be the same, i.e.

$$T_1(t)\, T_2(t)\, T_3' = T_1(t)\, T_2'\, T_3(t) = T_1'\, T_2(t)\, T_3(t),$$

and we shall also see that the same applies regardless of degree.

Lemma C.1. *Derivation of the multiplication of two $T(t)$ matrices is commutative. That is, for all $d > 0$ and for a given knot sequence $t_{j-d}, t_{j-d+1}, \ldots, t_{j+d+1}$, and $t_i \leq t < t_{i+1}$ is*

$$T_d(t)\, T_{d+1}' = T_d'\, T_{d+1}(t). \tag{C.1}$$

Proof. We start by computing the left side of C.1. In each line in all matrices $T_d(t)$, $d = 1, 2, \ldots$, there are only two elements that are different from zero. We multiply these with

the sub-matrix of T'_{d+1} which is the part of the matrix that gives something different from zero.

$$\begin{pmatrix} 1-w_{d,j}(t) & w_{d,j}(t) \end{pmatrix} \begin{pmatrix} -\delta_{d+1,j-1} & \delta_{d+1,j-1} & 0 \\ 0 & -\delta_{d+1,j} & \delta_{d+1,j} \end{pmatrix} = \\ \begin{pmatrix} -(1-w_{d,j}(t))\delta_{d+1,j-1} & (1-w_{d,j}(t))\delta_{d+1,j-1} - w_{d,j}(t)\delta_{d+1,j} & w_{d,j}(t)\delta_{d+1,j} \end{pmatrix}. \tag{C.2}$$

We use the same procedure on the right hand side of C.1,

$$\begin{pmatrix} -\delta_{d,j} & \delta_{d,j} \end{pmatrix} \begin{pmatrix} 1-w_{d+1,j-1}(t) & w_{d+1,j-1}(t) & 0 \\ 0 & 1-w_{d+1,j}(t) & w_{d+1,j}(t) \end{pmatrix} = \\ \begin{pmatrix} -(1-w_{d+1,j-1}(t))\delta_{d,j} & (1-w_{d+1,j}(t))\delta_{d,j} - w_{d+1,j-1}(t)\delta_{d,j} & w_{d+1,j}(t)\delta_{d,j} \end{pmatrix}, \tag{C.3}$$

and we compute the first element of (C.2) and (C.3),

$$(1-w_{d,j}(t))\delta_{d+1,j-1} = \frac{t_{j+d}-t}{(t_{j+d}-t_j)(t_{j+d+1}-t_{j-1})}, \\ (1-w_{d+1,j-1}(t))\delta_{d,j} = \frac{t_{j+d}-t}{(t_{j+d+1}-t_{j-1})(t_{j+d}-t_j)},$$

which shows that they are equal. We now compute the last element of (C.2) and (C.3),

$$w_{d,j}(t)\delta_{d+1,j} = \frac{t-t_j}{(t_{j+d}-t_j)(t_{j+d+1}-t_j)}, \quad w_{d+1,j}(t)\delta_{d,j} = \frac{t-t_j}{(t_{j+d+1}-t_j)(t_{j+d}-t_j)},$$

which shows that they also are equal. The middle element of (C.2) and (C.3) are a combination of the first and last element, and thus they are also equal, which completes the proof. □

The main commutativity theorem now follows.

Theorem C.1. *Derivation of multiplication of a set of $T(t)$ matrices is commutative. That is, for all $d > 0$ and a given knot vector any matrix can be the derivative matrix T'. i.e.*

$$T_d(t)\, T_{d+1}(t)\, \cdots\, T_{d+j-1}(t)\, T'_{d+j} = T'_d\, T_{d+1}(t)\, \cdots\, T_{d+j-1}(t)\, T_{d+j}(t). \tag{C.4}$$

Proof. This follows by an induction of Lemma C.1's statement □

C.3 Beta-functions, proof Lemma 7.1

Lemma 7.1 says the following:

The regularized incomplete beta function, $I_t(a,b)$, has the properties:

I Zero at $t = 0$ $I_0(a,b) = 0$,

II One at $t = 1$ $I_1(a,b) = 1$,

III	Hermite order	$\frac{d^j}{dt^j} I_0(a,b) = 0, \quad j = 1,2,...,a,$	i.e. order a at start
		$\frac{d^j}{dt^j} I_1(a,b) = 0, \quad j = 1,2,...,b,$	i.e. order b at end
IV	Antisymmetric	$I_t(a,b) = 1 - I_{1-t}(b,a),$	i.e. symmetric if a=b
V	monotone	$\frac{d}{dt} I_t(a,b) > 0, \quad 0 < t < 1$ and $\frac{d}{dt} I_t(a,b) = 0, \quad t = \{0,1\}.$	
VI	Recursive	$I_t(a,b) = t\, I_t(a-1,b) + (1-t)\, I_t(a,b-1).$	
		$I_t(a,b) = I_t(a-1,b) - \frac{t^a(1-t)^{b+1}}{a\, \mathscr{B}(a-1,b)} = I_t(a,b-1) + \frac{t^{a+1}(1-t)^b}{b\, \mathscr{B}(a,b-1)},$	

Proof. **I** follows directly from (7.18), **II** follows from (7.19) because $\mathscr{B}(1;a,b) = \mathscr{B}(a,b)$. To prove **III** we start differentiating (7.18), $\frac{d}{dt}\mathscr{B}(t;b,a) = t^a(1-t)^b$. It follows that the subsequent derivatives are a sum where each term contains a factor $t^j, \quad j > 0$ for all derivatives of order up to a, and a factor $(1-t)^j, \quad j > 0$ for all derivatives of order up to b. It follows that if $t = 0$ then all derivatives up to order a is zero, and that if $t = 1$ then all derivatives up to order b is zero, which completes the proof of **III**.

To prove **IV** we start computing

$$\begin{aligned}\mathscr{B}(1-t;b,a) &= \int_0^{1-t} x^b(1-x)^a dx \\ &= \int_t^1 (1-x)^b(1-(1-x))^a dx \\ &= \int_t^1 x^a(1-x)^b dx.\end{aligned}$$

It follows that $\mathscr{B}(t;a,b) + \mathscr{B}(1-t;b,a) = \mathscr{B}(a,b)$, which together with (7.19) completes the proof of **IV**.

The proof of **V** follows from that differentiating (7.18) we get $\frac{d}{dt}\mathscr{B}(t;b,a) = t^a(1-t)^b$, and that the derivative is positive on $0 < t < 1$ and zero at $t = 0$ and $t = 1$.

To prove **VI**, the recursion, we start differentiating the kernel of (7.18),

$$\frac{d}{dx} x^a(1-x)^b = a\, x^{a-1}(1-x)^b - b\, x^a(1-x)^{b-1}.$$

Then we go back by antiderivative (integration) and divide both sides with ab

$$\frac{1}{ab} x^a(1-x)^b = \frac{a}{ab}\int_0^x t^{a-1}(1-t)^b dt - \frac{b}{ab}\int_0^x t^a(1-t)^{b-1} dt. \tag{C.5}$$

We then add $(a+b)x^a(1-x)^b$ on both left and right side and reorganize the right side

$$\begin{aligned}\tfrac{a+b+1}{ab} x^a(1-x)^b = &\tfrac{a}{ab}\left(x^a(1-x)^b + \textstyle\int_0^x t^{a-1}(1-t)^b dt\right) + \\ &\tfrac{b}{ab}\left(x^a(1-x)^b - \textstyle\int_0^x t^a(1-t)^{b-1} dt\right).\end{aligned}$$

Integrating once more,

$$\frac{a+b+1}{ab}\int_0^x t^a(1-t)^b = \frac{1}{b} x \int_0^x t^{a-1}(1-t)^b dt + \frac{1}{a}(1-x)\int_0^x t^a(1-t)^{b-1} dt.$$

then we multiply both sides with $\frac{(a+b)!}{(a-1)!(b-1)!}$,

$$\frac{(a+b+1)!}{a!b!}\int_0^x t^a(1-t)^b = \frac{(a+b)!}{(a-1)!b!}x\int_0^x t^{a-1}(1-t)^b dt + \frac{(a+b)!}{a!(b-1)!}(1-x)\int_0^x t^a(1-t)^{b-1}dt.$$

which together with(7.17) and (7.19) completes the proof of the first part of the recursion. The proof of the two last parts is just a reorganizing of (C.5), which ends the proof. □

C.4 Rational B-functions, proof Theorem 7.4

Theorem 7.4 says that a rational B-function, i.e. an RB-function is a real B-function. The following proves theorem 7.4

Proof. We follow Definitions 7.1. First it is clearly a permutation function (**D1**). If we compute (7.21) we see that $B_{a,b}(0) = 0$ (**D2**) and $B_{a,b}(1) = 1$ (**D3**). The 1^{st}-derivative of (7.21) is

$$B'_{a,b}(t) = (a(1-t)+bt+1)\frac{t^a(1-t)^b}{(t^{a+1}+(1-t)^{b+1})^2},$$

It is clear that $B'_{a,b}(t) \geq 0$, because t and $1-t \geq 0$ for $t \in [0,1]$, and it follows that all terms and factors are greater or equal zero. Thus the RB-function is monotone (**D4**).

Reformulating to $B_{a,b}(t) = \frac{\alpha(t)}{\beta(t)}$. The derivative $B^{(j)}_{a,b}(t)$ is a fraction where the numerator is a sum where each term has factors of $\alpha^{(r)}(t), r \leq j$ and $\beta^{(s)}(t), s \leq j$ where j is the order of the derivative, and the denominator is $\beta(t)^{2j}$. Note that $\beta > 0$ when $t \in [0,1]$.

a) If $j \leq a$ in $B^{(j)}_{a,b}(t)$ then every term in the numerator has a factor t^{a+1-r}, $r \leq j$. It follows that $B^{(j)}_{a,b}(0) = 0$ (left side Hermite order).

b) Because $\lim_{t\to 1}(1-t) = 0$ it follows that if $j \leq b$ then $\lim_{t\to 1}\beta^{(j)}(t) = \alpha^{(j)}(t)$. Thus if $j \leq b$ then $B^{(j)}_{a,b}(1) = 0$ (right side Hermite order).

Finally, the RB-function $B_s(t)$ is symmetric (**D5**) since

$$B_S(t) + B_S(1-t) = \frac{t^{S+1}}{t^{S+1}+(1-t)^{S+1}} + \frac{(1-t)^{S+1}}{(1-t)^{S+1}+t^{S+1}} = 1,$$

which ends the proof. □

C.5 ERB-functions, proof Theorem 7.5

Theorem 7.5 states that $B_d(t)$ defined in (7.31), $B_y(t)$ defined in (7.33), $B_x(t)$ defined in (7.34) and $B_z(t)$ defined in (7.35) are all symmetric Expo-Rational B-functions (ERB-functions) and that they are all complete B functions, cf. Definition 7.3.

The following proves theorem7.5

Proof. We follow Definitions 7.1. First the four functions (7.31), (7.33), (7.34) and (7.35) are permutation functions (**D1**). This follows from that the next three points are fulfilled. If we compute them we see that $B(0) = 0$ (**D2**) and $B(1) = 1$ (**D3**). The first derivative $B'_d(t) > 0,\ t \in (0,1)$ in(7.36) because exponential functions are always positive, and the same argument applies to $B'_x(t)$ and $B'_z(t)$ in (7.37) as well. $B'_y(t)$ in (7.36) is also positive when $t \in [0,1]$ because the the denominator in the fraction is always positive and the numerator in the fraction is also always positive because $(1 - B_y(t)) > 0$ since $B_y(t) < 1,\ 0 < t < 1$. Thus they are all monotone (**D4**).

The symmetry (**D5**) follows from:

Because $\left(t - \frac{1}{2}\right)^2 = \left(1 - t - \frac{1}{2}\right)^2$ the symmetry of $B_d(t)$ follows from

$$B_d(t) + B_d(1-t) = S_d \int_0^t \phi(s)ds + S_d \int_t^1 \phi(s)ds = 1.$$

The two next functions are constructed to be symmetric,

$$B_y(t) + B_y(1-t) = \frac{\Psi(t)}{\Psi(t) + \Psi(1-t)} + \frac{\Psi(1-t)}{\Psi(1-t) + \Psi(t)} = 1,$$

$$B_x(t) + B_x(1-t) = \frac{1}{2}\left(1 - \Psi(1-t) + \Psi(t)\right) + \frac{1}{2}\left(1 - \Psi(t) + \Psi(1-t)\right) = 1.$$

For $B_z(t)$ we see that it will be similar to $B_y(t)$, so we get

$$B_z(t) + B_z(1-t) = \frac{\theta(1-t)}{\theta(1-t) + \theta(t)} + \frac{\theta(t)}{\theta(t)) + \theta(1-t)} = 1,$$

which completes the first part of the proof, $B_d(t)$, $B_y(t)$, $B_x(t)$ and $B_z(t)$ are all symmetric B-functions.

The second part, that $B_d(t)$, $B_y(t)$, $B_x(t)$ and $B_z(t)$ are complete ERB-functions,that is, the Hermite order is infinite, depend on two properties of the exponential function, $(e_x)' = e_x$ and $\lim_{x\to+\infty} x_n e^{-x} = 0$ for every n. For all ERB-functions the exponential function will be a factor on each term of all derivatives. Along with what we observed in Figure 7.16, that is, the mapping from $\mathbb{R}$ to $(0,1)$, thus the override of the exponential function will secure that all derivatives are zero at $t = 0, 1$. This ends the proof. □

C.6 Order symmetry of a B-function, proof Theorem 7.6

Theorem 7.6 states that Beta functions and RB functions are order symmetric. And that the ERB functions $B_d(t;\alpha,\beta)$ and $B_z(t;\alpha,\beta)$ are "order symmetric" in the sense that they are symmetric according to α and β.

The following proves theorem 7.6.

Proof. The proof of Theorem 7.6 is divided in three parts, one for each type of B-functions.

Beta-functions: The Beta-function is the regularized incomplete beta function defined in (7.18). If we first look at the denominator of the fraction, we see that $\mathscr{B}(a,b) = \mathscr{B}(b,a)$ because it follows from (7.17). Therefor, we have a common denominator and we can just sum up the numerators. If we put (7.18) into (7.48) we get

$$\mathscr{B}(t;a,b) + \mathscr{B}(1-t;b,a) = \int_0^t x^a(1-x)^b dx + \int_t^1 (1-x)^b x^a dx = \int_0^1 x^a(1-x)^b dx.$$

It follows that the numerators and the denominator are equal, and it verifies that the sum is 1, and that the Beta-functions are Order-symmetric.

RB-functions: The RB-function is defined in Theorem 7.4, (7.21). If we put (7.21) into (7.48) we get

$$B_{a,b}(t) + B_{b,a}(1-t) = \frac{t^{a+1}}{t^{a+1} + (1-t)^{b+1}} + \frac{(1-t)^{b+1}}{(1-t)^{b+1} + t^{a+1}} = 1,$$

which verifies that the RB-functions are Order-symmetric.

ERB-functions $B_d(t;\alpha,\beta)$ and $B_z(t;\alpha,\beta)$: $B_d(t;\alpha,\beta)$ is defined in(7.31) and (7.44). If we exchange α with β and replace t with $1-t$ in (7.44), it turns out that $\phi(1-t;\beta,\alpha) = \phi(t;\alpha,\beta)$. Thus

$$B_d(t;\alpha,\beta) + B_d(1-t;\beta,\alpha) = S_{\alpha,\beta} \int_0^t \phi(s;\alpha,\beta)ds + S_{\beta,\alpha} \int_t^1 \phi(s;\beta,\alpha)ds = 1.$$

which verifies that the $B_d(t;\alpha,\beta)$ are Order-symmetric in the sense that it is symmetric according to α and β. When it comes to $B_z(t;\alpha,\beta)$ we get,

$$B_z(t;\alpha,\beta) + B_z(1-t;\beta,\alpha) = \frac{e^{\frac{1}{(1-t)^\beta}}}{e^{\frac{1}{(1-t)^\beta}} + e^{\frac{1}{t^\alpha}}} + \frac{e^{\frac{1}{t^\alpha}}}{e^{\frac{1}{t^\alpha}} + e^{\frac{1}{(1-t)^\beta}}} = 1,$$

$B_z(t;\alpha,\beta)$ are Order-symmetric in the sense that it is symmetric according to α and β. □

C.7 Balance symmetry of a B-function, proof Theorem 7.7

Theorem 7.7 states that Rμ functions and ERB functions are balance symmetric.

The following proves Theorem 7.7.

Proof. The proof of Theorem 7.7 is divided in two parts, one for each type of B-functions.

Rμ-functions: The Rμ-function is defined in Corollary 7.1, (7.22). If we put (7.22) into (7.49) we get

$$B(t;\mu) + B(1-t;1-\mu) = \frac{(1-\mu)t^{S+1}}{(1-\mu)t^{S+1} + \mu(1-t)^{S+1}} + \frac{\mu(1-t)^{S+1}}{\mu(1-t)^{S+1} + (1-\mu)t^{S+1}} = 1,$$

which verifies that Rμ-functions are Balance-symmetric.

ERB-functions: $B_d(t;\mu)$, $B_x(t;\mu)$, $B_y(t;\mu)$ and $B_z(t;\mu)$ are defined in subsection 7.7.2. If we replace μ with $1-\mu$ and t with $1-t$ in (7.39) we see that

$$\phi(t;\mu) = e^{-\frac{(t-\mu))^2}{t(1-t)}} \quad \text{and} \quad \phi(1-t;1-\mu) = e^{-\frac{((1-t)-(1-\mu))^2}{(1-t)t}} = e^{-\frac{(\mu-t))^2}{(1-t)t}},$$

i.e. they are equal. It follows that $S_\mu = S_{1-\mu} = \int_0^1 \phi(s;\mu)ds$, and we get,

$$B_d(t;\mu) + B_d(1-t;1-\mu) = S_\mu \int_0^t \phi(s;\mu)ds + S_{1-\mu} \int_t^1 \phi(s;\mu)ds = 1,$$

which verifies that $B_d(t;\mu)$ is Balance-symmetric. We now replace μ with $1-\mu$ and t with $1-t$ in (7.43), we get

$$B_x(t;\mu) + B_x(1-t;1-\mu) = \mu\left(1 - e^{\frac{-2}{1-t}e^{\frac{-1}{t}}}\right) + (1-\mu)e^{\frac{-2}{t}e^{\frac{-1}{1-t}}} + (1-\mu)\left(1 - e^{\frac{-2}{t}e^{\frac{-1}{1-t}}}\right) + \mu e^{\frac{-2}{1-t}e^{\frac{-1}{t}}} = 1.$$

which verifies that $R_x(t;\mu)$ is Balance-symmetric. Next we replace μ with $1-\mu$ and t with $1-t$ in (7.42), we get

$$B_y(t;\mu) + B_y(1-t;1-\mu) = \frac{(1-\mu)\varphi(t)}{(1-\mu)\varphi(t) + \mu\varphi(1-t)} + \frac{\mu\varphi(1-t)}{\mu\varphi(1-t) + (1-\mu)\varphi(t)} = 1,$$

which verifies that $R_y(t;\mu)$ are Balance-symmetric. Finally we replace μ with $1-\mu$ and t with $1-t$ in (7.41) we get

$$B_z(t;\mu) + B_z(1-t;1-\mu) = \frac{e^{\frac{\mu}{(1-t)}}}{e^{\frac{\mu}{(1-t)}} + e^{\frac{1-\mu}{t}}} + \frac{e^{\frac{1-\mu}{t}}}{e^{\frac{1-\mu}{t}} + e^{\frac{\mu}{(1-t)}}} = 1,$$

which verifies that $R_z(t;\mu)$ is Balance-symmetric. This ends the proof. □

C.8 Simultaneous order and balance symmetry, proof Theorem 7.8

Theorem 7.8 says that the following B-functions are order-symmetric and balance-symmetric at the same time: The Rμ functions, the ERB function B_d and the ERB function B_z.

Proof. The proof of Theorem 7.8 is divided in two parts, one for each type of B-functions.

Rμ-functions: The Rμ-function is defined in Corollary 7.1, (7.22). If we exchange a and b, replace μ with $1-\mu$ and t with $1-t$ we get

$$B_{a,b}(t;\mu) + B_{b,a}(1-t;1-\mu) = \frac{(1-\mu)t^{a+1}}{(1-\mu)t^{a+1} + \mu(1-t)^{b+1}} + \frac{\mu(1-t)^{b+1}}{\mu(1-t)^{b+1} + (1-\mu)t^{a+1}},$$

which sums up to 1. This confirms that the Rμ-functions simultaneously are Balance- and Order-symmetric.

The $B_d(t,\mu,\alpha,\beta)$-functions: The kernel of $B_d(t,\mu,\alpha,\beta)$ is defined in (7.44). If we exchange α and β, replace μ with $1-\mu$ and t with $1-t$ in (7.44) we get

$$\phi(t;\mu,\alpha,\beta)=e^{-\frac{|t-\mu|^{\beta+\alpha}}{t^{\alpha}(1-t)^{\beta}}} \quad \text{and} \quad \phi(1-t;1-\mu,\beta,\alpha)=e^{-\frac{|(1-t)-(1-\mu)|^{\beta+\alpha}}{(1-t)^{\beta}t^{\alpha}}}=e^{-\frac{|\mu-t|^{\beta+\alpha}}{t^{\alpha}(1-t)^{\beta}}},$$

which shows that $\phi(t;\mu,\alpha,\beta)=\phi(1-t;1-\mu,\beta,\alpha)$, which in turn means that $S_{1-\mu,\beta,\alpha}=S_{\mu,\alpha,\beta}$ and we get

$$B_d(t;\mu,\alpha,\beta)+B_d(1-t;1-\mu,\beta,\alpha)=$$
$$S_{\mu,\alpha,\beta}\int_0^t\phi(s;\mu,\alpha,\beta)ds+S_{1-\mu,\beta,\alpha}\int_t^1\phi(1-s;1-\mu,\beta,\alpha)ds=S_{\mu,\alpha,\beta}\int_0^1\phi(s;\mu,\alpha,\beta)ds=1,$$

confirming that $B_d(t,\mu,\alpha,\beta)$ simultaneously are Balance- and "Order"-symmetric.

The $B_z(t,\mu,\alpha,\beta)$-functions: The kernel of $B_z(t,\mu,\alpha,\beta)$ is defined in (7.46).

Finally we replace μ with $1-\mu$ and t with $1-t$ in (7.41) we get

$$B_z(t;\mu,\alpha,\beta)+B_z(1-t;1-\mu,\beta,\alpha)=\frac{e^{\frac{\mu}{(1-t)^{\beta}}}}{e^{\frac{\mu}{(1-t)^{\beta}}}+e^{\frac{1-\mu}{t^{\alpha}}}}+\frac{e^{\frac{1-\mu}{t^{\alpha}}}}{e^{\frac{1-\mu}{t^{\alpha}}}+e^{\frac{\mu}{(1-t)^{\beta}}}}=1,$$

confirming that $B_z(t,\mu,\alpha,\beta)$ simultaneously are Balance- and "Order"-symmetric. This ends the proof. □

C.9 Properties of 2-p B-functions $B(u,v)$

In section 11.1, a 2-p B-function $B(u,v)$, $(u,v)\in[0,1]\times[0,1]$ is defined, see (11.11). A B-function $B(u,v)$ of Hermite order d has the following properties:

- that the value is 0 "internally" on two opposite edges (with constant v value),
- that the value is 1 over all the other two edges (with constant u value),
- it is symmetric, i.e. $B(u,v)+B(v,u)=1$,
- it is "internally" C^d-smooth,
- that all derivatives up to order d are 0 on the edges,
- and it is discontinuous in all the corners in the "u direction"

Below are 2 Propositions with proofs that confirm the list of properties,

Proposition C.1. *At two opposite edges is the function value 0 (note that it does not include the corner points), at the other two edges is the function value 1 (including the corner points). i.e.*

$$B(0,v) = B(1,v) = 1, \quad v \in [0,1], \tag{C.6}$$
$$B(u,0) = B(u,1) = 0, \quad u \in (0,1). \tag{C.7}$$

Further, at all corners and all edges has all derivatives up to order d the value zero,

$$\begin{aligned} D_u^{(i)} D_v^{(j)} B(0,v) = D_u^{(i)} D_v^{(j)} B(1,v) = 0, \quad v \in [0,1], \quad 0 \le i,j \le d, \quad i+j > 0, \\ D_u^{(i)} D_v^{(j)} B(u,0) = D_u^{(i)} D_v^{(j)} B(u,1) = 0, \quad u \in (0,1), \quad 0 \le i,j \le d, \quad i+j > 0. \end{aligned} \tag{C.8}$$

Proof. In section 11.1 we have $g(u)$ (11.3), $a(u)$ (11.5), $t(u,v)$ (11.6) and $B(u,v)$ (11.8), and

if $u = 0$ and $v \in [0,1]$	then $a(0) = 0$, $g(0) = 1$ and $t(0,v) = 1$	than $B(0,v) = 1$,
if $u = 1$ and $v \in [0,1]$	then $a(0) = 0$, $g(1) = 1$ and $t(1,v) = 1$	than $B(1,v) = 1$,
if $v = 0$ and $u \in (0,1)$	then $t(u,0) = 0$	than $B(u,0) = 0$,
if $v = 1$ and $u \in (0,1)$	then $t(u,1) = 0$	than $B(u,1) = 0$,

which proves (C.6) and (C.7).

Recall from the table above that $g(0) = g(1) = 1$ and $t(0,v) = t(1,v) = 1$, $t(u,0) = t(u,1) = 0$. Now, from the properties of a B-function (7.1), it follows that (at the boundaries) $B^{(j)}(0) = B^{(j)}(1) = 0$, $j = 1,2,...,d$ where d is the order of the B-function, and we also recognize from the definition (11.4) that $g^{(j)}(0) = g^{(j)}(1) = 0$, $j = 1,2,\ldots,d$. Because of the product derivation rule, all partial derivatives of $B(u,v)$ up to order d will contain a set of terms, where every terms contains a product of either g and some derivatives of B (that is 0 because $B^{(j)} = 0$, $j = 0,1,\ldots,d$), or $g^{(j)}$ and B (that is 0 because $g^{(j)}(0) = g^{(j)}(1) = 0$, $j = 1,2,\ldots,d$). It follows that (C.8) in Proposition C.1 is true, which ends the proof □

The domain of $S(u,v)$ is $U = [0,1] \times [0,1] \subset \mathbb{R}^2$. To prove that $S(u,v)$ is C^d-smooth on the domain except for the corners $p_1 = (0,0)$, $p_2 = (1,0)$, $p_3 = (0,1)$, and $p_4 = (1,1)$, we have the following proposition.

Proposition C.2. *The 2-parameter blending function $B(u,v)$ is $\in C^d(V)$, $V = U \backslash (0,0) \cup (1,0) \cup (0,1) \cup (1,1)$, and where $U = [0,1] \times [0,1] \subset \mathbb{R}^2$.*

- *From $p_1 = (0,0) \in U$ is B discontinuous in the direction $v_1 = (1,z)$, $0 \le z < 2$*
- *From $p_2 = (1,0) \in U$ is B discontinuous in the direction $v_2 = (-1,z)$, $0 \le z < 2$*
- *From $p_3 = (0,1) \in U$ is B discontinuous in the direction $v_3 = (1,-z)$, $0 \le z < 2$*
- *From $p_4 = (1,1) \in U$ is B discontinuous in the direction $v_4 = (-1,-z)$, $0 \le z < 2$*

Proof. The 2-p blending function $B(u,v) = g(u)\ B \circ t(u,v)$ consists of the B-function $B(t)$ described in Section 7.1, the function g defined in (11.3) and the function t defined in (11.6). Each of these factors will be analyzed and then concluded with a result. The properties of $B(t)$ are important and are also transferred to $B(u,v)$. The proof is divided in 4 parts, 1) is about g, 2) is about t, 3) is about $B \circ t$ and 4) gives a conclusion.

1) From (11.3) we see that $g(u)$ is symmetric about $u=\frac{1}{2}$. It follows that the number of derivatives that are zero at $u=\frac{1}{2}$ decide the continuity level of g. It follows that $g^{(j)}(u)|_{u=\frac{1}{2}} = B^{(j)}(u)|_{u=\frac{1}{2}}$. Hence from the property described in Definition 7.2)

$$g(u) \in C^d([0,1]).$$

2) Analyzing $t(u,v)$ in expression (11.6) we see that:

- For a fixed u-value $\bar{u} \in [0,1]$, is $\widetilde{t}(v) = t(\bar{u},v) \in C^0([0,1])$ and $\widetilde{t}(v)$ is piecewise linear.
- For a fixed v-value $\bar{v} \in (0,1)$, is $\widehat{t}(u) = t(u,\bar{v}) \in C^0([0,1])$, $\widehat{t}(u)$ is piecewise smooth.
- For $v=0$ we see that $\widehat{t}(u) = t(u,0) = \frac{0}{a(u)} = 0$ when $0<u<1$.
- For $v=1$ we see that $\widehat{t}(u) = t(u,1) = \frac{1-1}{a(u)} = 0$ when $0<u<1$.

It follows that t is continuous on V (V defined in Proposition C.2). We see that at the four corner points is t continuous in the v-direction, but discontinuous in u-direction.

- A closer examination of the corner points shows the following:
  a - Computing the formula (11.6) we get $t(p_1) = 1$.
  b - From (11.6) it follows that $t(u,v) = 1$ between the two curves in the parameter plane $v = a(u)$ and $v = 1-a(u)$, (11.5), and that $t(u,v) < 1$ else. It follows that in the start of the curve $v = a(u)$ is $(v',a')|_{p_1} = (1,2)$. Investigating the directions $(1,z)$, $0 \le z < 2$ we see that

$$\lim_{u\to 0+} t(u,z\,u) = \frac{z}{2}. \tag{C.9}$$

  Contrary to paragraph a- above is $\lim_{u\to 0+} t(u,z\,u) < 1$. Hence $t(u,v)$ is discontinues from the corner point p_1 in the direction $v_1 = (1,z)$, $0 \le z < 2$. Using symmetry it follows that $t(u,v)$ is discontinues from the corner points
  - $p_2 = (1,0)$ in the direction $v_2 = (-1,z)$, $0 \le z < 2$,
  - $p_3 = (0,1)$ in the direction $v_3 = (1,-z)$, $0 \le z < 2$,
  - $p_4 = (1,1)$ in the direction $v_4 = (-1,-z)$, $0 \le z < 2$.

3) Since $t(u,v)$ is continuous on V (V defined in Proposition C.2), and discontinuous at the four corner points as described above, it follows that $B \circ t(u,v)$ is the same.

We see that $t(u,v)$ is divided in three parts, by two curves in the parameter plane, $v = a(u)$ and $v = 1-a(u)$. Each part is internally smooth, continuous first, second, etc. derivatives. On the two curves we see the following,

$$\begin{aligned} t(u,a(u)) = t(u,1-a(u)) &= 1, \\ B^{(j)}(t(u,a(u))) = B^{(j)}(t(u,1-a(u))) &= 0, \quad j = 1,2,...,d. \end{aligned}$$

It follows that $B \circ t(u,v)$ is not only continuous on V but is $C^d(V)$. This because every place a derivative of some order of t is discontinuous is the value of t equal 1 and all derivatives up to the current order is zero (this is illustrated for B in (11.9)).

4) It follows that $B(u,v)$ will inherit all discontinuities from both g and $B \circ t(u,v)$, hence $B(u,v)$ is in $C^d(V)$, but it is discontinuous from the points p_1 in direction v_1, p_2 in direction v_2, p_3 in direction v_3 and p_4 in direction v_4, which ends the proof. □

C.10 Two-surface blending and continuity

The following theorems will show that it is possible to fill a square hole with a surface so that the result has a desired degree of continuity.

Theorem C.2. *Given two surfaces $S_1(u,v)$ and $S_2(u,v)$ defined over the same domain $U = [0,1] \times [0,1] \subset \mathbb{R}^2$ and a number d that determines the continuity. Then, a surface $S(u,v) = B(u,v)\, S_1(u,v) + (1 - B(u,v))\, S_2(u,v)$ is in $C^d(U)$ if,*
a) *the blending function $B(u,v)$ from Section 11.1 is $\in C^d(V)$, V defined in Proposition C.2,*
b) *and the two surfaces $S_1(u,v)$ and $S_2(u,v)$ both are $\in C^d(U)$,*
c) *and that in the four corners the value and all derivatives up to order d are equal in both the surfaces S_1 and S_2.*

Proof. From Proposition C.2 and the restriction **b)** above on the surfaces S_1 and S_2, it follows that $S \in C^d(V)$, for V defined in Proposition C.2.

Proposition C.2 tells us that B is discontinuous from the four corner points, $p_i,\ i = 1,2,3,4$, in given directions $v_i,\ i = 1,2,3,4$. However, the behavior in the four corners are symmetrical or antisymmetric about the center point of the surface. We therefor only need to look at one of the corner points. Therefore, we use the corner point $p_1 = (0,0)$ for the investigation.

- We first examine the value at the corner p_1, using (11.2), remember from (C.6) that $B(0,0) = 1$,

$$S(0,0) = S_1(0,0) + 1(S_2(0,0) - S_1(0,0)) = S_2(0,0). \tag{C.10}$$

To investigate the limit value when we, on the surface S, move in direction v_1 towards the corner point p_1, we set up

$$S(u,zu) = S_1(u,zu) + B(u,zu)\, \widetilde{S}(u,zu), \quad 0 \le z < 2.$$

It follows from (11.8) that

$$B(u,zu) \le 1, \quad 0 \le z < 2,$$

and together with the restriction **c)** in Theorem C.2 it follows that

$$\lim_{u \to 0} |\widetilde{S}(u,zu)| = 0.$$

Therefor

$$\lim_{u \to 0} S(u,zu) = S_1(u,zu). \tag{C.11}$$

It follows from (C.10) and (C.11) that S is continuous on $U = [0,1] \times [0,1] \subset \mathbb{R}^2$.

- We then examine the first order derivatives at the corner p_1,

$$S_u(0,0) = S_{1u}(0,0) + 1(S_{2u}(0,0) - S_{1u}(0,0)) = S_{2u}(0,0), \tag{C.12}$$
$$S_v(0,0) = S_{1v}(0,0) + 1(S_{2v}(0,0) - S_{1v}(0,0)) = S_{2v}(0,0). \tag{C.13}$$

To investigate the limit value when we, on the surface S, move in direction v_1 towards the corner p_1, we differentiate expression (11.2), i.e.

$$S_u(u,zu) = S_{1u}(u,zu) + B_u(u,zu)\ \widetilde{S}(u,zu) + B(u,zu)\ \widetilde{S}_u(u,zu),$$
$$S_v(u,zu) = S_{1v}(u,zu) + B_v(u,zu)\ \widetilde{S}(u,zu) + B(u,zu)\ \widetilde{S}_v(u,zu),$$

where $0 \leq z < 2$. Further, it follows from (11.8) that

$$B(u,zu) \leq 1, \quad 0 \leq z < 2,$$

and together with the restriction **c)** in Theorem C.2 it follows that both

$$\lim_{u\to 0+} |B(u,zu)\ \widetilde{S}_u(u,zu)| = 0,$$
$$\lim_{u\to 0+} |B(u,zu)\ \widetilde{S}_v(u,zu)| = 0.$$

From (11.3) we see that

$$\lim_{u\to 0+} g(u) = 1,$$
$$\lim_{u\to 0+} g^{(j)}(u) = 0, \quad j = 1,2,...,d,$$

and from expression (C.9) it follows that

$$\lim_{u\to 0+} B\circ t(u,zu) = B\left(\frac{z}{2}\right),$$
$$\lim_{u\to 0+} B^{(j)}\circ t(u,zu) = B^{(j)}\left(\frac{z}{2}\right),$$

and from (11.7),

$$t_u(u,zu) = \frac{zu\ 2(1-2u)}{(2u(1-u))^2} = \frac{1-2u}{(1-u)^2}\ \left(\frac{z}{2}\right)\ \frac{1}{u},$$
$$t_v(u,zu) = \frac{1}{2(1-u)} \qquad = \frac{1}{1-u}\ \left(\frac{1}{2}\right)\ \frac{1}{u}.$$

Therefor it follows that

$$\begin{aligned}
\lim_{u\to 0+} B_u(u,zu)\ \widetilde{S}(u,zu) &= \lim_{u\to 0+} \left((g'B + gB't_u)\ \widetilde{S}(u,zu)\right) \\
&= B'\left(\frac{z}{2}\right)\ \frac{z}{2}\ \lim_{u\to 0+}\left(\left(\frac{1-2u}{(1-u)^2}\right)\frac{\widetilde{S}(u,zu)}{u}\right), \\
&= B'\left(\frac{z}{2}\right)\ \frac{z\sqrt{1+z^2}}{2}\ \lim_{u\to 0+}\frac{\widetilde{S}(u,zu)}{u\sqrt{1+z^2}}, \\
&= k\ d\widetilde{S}_{p_1}(\widehat{v}_1),
\end{aligned}$$

where $k = B'\left(\frac{z}{2}\right)\ \frac{z\sqrt{1+z^2}}{2}$ and $\widehat{v}_1 = \frac{v_1}{|v_1|}$.

$d\widetilde{S}_{p_1}(\widehat{v}_1)$ is the directional derivatives at p_1 in direction $\widehat{v}_1$. From restriction **c)** in Theorem C.2 it follows that, if $d > 0$ then all derivatives of order 1 has the same values in the two surfaces S_1 and S_2. Thus all directional derivatives of order 1, to the surface $\widetilde{S}$ at p_1 must be zero-vectors. Hence

$$\lim_{u\to 0+} |B_u(u,zu)\ S(u,zu)| = 0, \quad d > 0. \tag{C.14}$$

We use the same method to treat the partial derivative B_v.

$$\begin{aligned}\lim_{u\to 0+} B_v(u,zu)\ \widetilde{S}(u,zu) &= \lim_{u\to 0+} \left(gB't_v\ \widetilde{S}(u,zu)\right)\\ &= k\ d\widetilde{S}_{p_1}(\widehat{v}_1),\end{aligned}$$

where $k = B'\left(\frac{z}{2}\right)\ \frac{\sqrt{1+z^2}}{2}$ and $\widehat{v}_1 = \frac{v_1}{|v_1|}$.

$d\widetilde{S}_{p_1}(\widehat{v}_1)$ is the directional derivatives at the point p_1 in direction $\widehat{v}_1$. It also now follows that

$$\lim_{u\to 0+} |B_v(u,zu)\ S(u,zu)| = 0, \quad d > 0. \tag{C.15}$$

It follows from (C.14) and (C.15), that if $d > 0$, the partial derivatives are

$$\lim_{u\to 0+} S_u(u,zu) = S_{1u}(u,zu), \quad 0 \le z < 2, \tag{C.16}$$

$$\lim_{u\to 0+} S_v(u,zu) = S_{1v}(u,zu), \quad 0 \le z < 2. \tag{C.17}$$

As a conclusion of this examination and because of restriction **c)** in Theorem C.2 is $S_{1u}(0,0) = S_{2u}(0,0)$ and $S_{1v}(0,0) = S_{2v}(0,0)$, it follows from (C.12), (C.13), (C.16), (C.17) that at least is $S \in C^1(U),\ U = [0,1]\times[0,1] \subset \mathbb{R}^2$.

- For higher-order derivatives, the argument is analogous to first-order derivatives (though a little more complicated), and applies as long as restriction c) in the Theorem is valid.

This ends the proof. □

Bibliography

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, ninth Dover printing, tenth GPO printing edition, 1964.

[2] H. Akima. A new method of interpolation and smooth curve fitting based on local procedures. *Journal of ACM*, 17:589–602, 1970.

[3] J. P. Allouche and J. Shallit. The ubiquitous prouhet-thue-morse sequence. In H. Niederreiter C. Ding, T. Helleseth, editor, *Sequences and their Applications*, Discrete Mathematics and Theoretical Computer Science. Springer London, 1999.

[4] R. L. Bagula and P. Bourke. Trianguloid trefoil . http://paulbourke.net/geometry/tranguloid/, 2002. [Online; accessed August-2021].

[5] R. L. Bagula and P. Bourke. Bent horns surface . http://paulbourke.net/geometry/benthorns/, 2003. [Online; accessed August-2021].

[6] B. Bang, L. T. Dechevsky, A. Lakså, and P. Zanaty. Blending functions for hermite interpolation by beta-function b-splines on triangulations. In Ivan Lirkov, Svetozar Margenov, and Jerzy Waśniewski, editors, *Large-Scale Scientific Computing*, volume 7116 of *Lecture Notes in Computer Science*, pages 393–401. Springer Berlin Heidelberg, 2012.

[7] R.E. Barnhill, R.F. Riesenfeld, United States. Office of Naval Research, and University of Utah. *Computer aided geometric design: proceedings of a conference held at the University of Utah, Salt Lake City, Utah, March 18-21, 1974*. Academic Press Rapid manuscript Reproduction. Academic Press, 1974.

[8] A. H. Barr. Global and local deformations of solid primitives. In *SIGGRAPH '84: Proceedings of the 11th annual Conference on Computer Graphics*, pages 21–30, 1984.

[9] D. Bechmann. Multidimensional Free-form Deformation Tools. In *Eurographics'98, State of the Art Report*, 1999.

[10] S. Bernstein. Démonstration du théoŕeme de Weierstrass fondée sur le calcul des probabilities. *Comm. Soc. Math.*, 13(1–2), 1912.

[11] A. Beutelspacher and U. Rosenbaum. *Projective geometry: from foundations to applications*. Cambridge University Press, Cambridge, 1998.

[12] P. Bézier. Définition numérique des courbes et surfâces I. *Automatisme*, XI:625–632, 1966.

[13] P. Bézier. Définition numérique des courbes et surfâces II. *Automatisme*, XII:17–21, 1967.

[14] Botsch Steinberg Bischoff, M. Botsch, S. Steinberg, S. Bischoff, L. Kobbelt, and Rwth Aachen. Openmesh – a generic and efficient polygon mesh data structure. In *In OpenSG Symposium*, 2002.

[15] Wolfgang Boehm. Inserting New Knots into B-spline Curves. *Journal of Computer Aided Design*, 12(4):199–201, 1980.

[16] P. Bourke. Mathematical Sea Shell . http://paulbourke.net/geometry/spiral/, 1998. [Online; accessed August-2021].

[17] V. Brun. Gauss' fordelingslov. *Norsk Matematisk Tidsskrift*, 14:81–92, 1932.

[18] P. L. Butzer, M. Schmidt, and E. L. Stark. Observations on the History of Central B-Splines. *Archive for History of Exact Sciences*, 39:137–156, 1988/89.

[19] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological surfaces. *Computer-Aided Design*, 10(6):350–355, 1978.

[20] E. Catmull and R. Rom. A class of local interpolating splines. *Computer Aided Geometric Design*, pages 317–326, 1974.

[21] George Merrill Chaikin. An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3(4):346 – 349, 1974.

[22] Kęstutis Karčiauskas and Jörg Peters. Point-augmented biquadratic C1 subdivision surfaces. *Graphical Models*, 77:18–26, 2015.

[23] E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-splines and subdivision techniques in computer aided geometric design and computer graphics. *Comp. Graphics and Image Process*, 14(2):87–111, 1980.

[24] E. Cohen, T. Lyche, and L. L. Schumaker. Algorithms for degree-raising of splines. *ACM Transactions on Graphics (TOG)*, 4(3):171–181, 1985.

[25] S. D. Conte and C. de Boor. *Elementary Numerical Analyses*. McGraw-Hill, Singapore, 1983.

[26] S. A. Coons. Surfaces for computer aided design. Technical report, MIT, Cambridge, MA, USA, 1964. Available as AD 663 504 from the National Technical Information service, Springfield, VA 22161.

[27] S. Coquillart. Extended Free-form deformation: a sculpturing tool for 3D geometric modeling. In *SIGGRAPH '90: Proceedings of the 17th annual Conference on Computer Graphics*, pages 187–196, 1990.

[28] J. Austin Cottrell, Thomas J. R. Hughes, and Yuri Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley Publishing, 1st edition, 2009.

[29] M. G. Cox. Curve fitting with piecewice polynomials. *J. Inst. Math. Appl.*, 8:36–52, 1972.

[30] H.S.M. Coxeter. *Projective geometry*. University of Toronto Press, Toronto, Ont., second edition, 1974.

[31] B. H. Curry. Review of the paper [137, 138]. *Math, Tables and other Aids to Comp.*, 2:167–169 and 211–213, 1947.

[32] B. H. Curry and I. J. Schoenberg. On Pòlya frequency functions IV: The spline functions and their limits. *Bull. Amer. Math. Soc.*, 53:1114, 1947. Abstract 380t.

[33] B. H. Curry and I. J. Schoenberg. On Pòlya frequency functions IV: The fundamental spline functions and their limits. *J. d'Analyse Math.*, 17:71–107, 1966.

[34] Rune Dalmo. Matrix factorization of multivariate Bernstein polynomials. *International Journal of Pure and Applied Mathematics*, 103:749–780, 01 2015.

[35] P. J. Davis. *Interpolation and Approximation*. Dover Publication Inc. (unabridged republication from a first edition from 1963), New York, N.Y., 1975.

[36] C. de Boor. On calculation with B-splines. *Journal of Approximation Theory*, 6:50–62, 1972.

[37] C. de Boor. *A Practical Guide to Splines*, volume 27 of *Applied Mathematical Sciens*. Springer-Verlag, New York, 1978.

[38] P. de Casteljau. Outillages méthodes calcul. Technical report, A. Citroën, Paris, 1959.

[39] P. de Casteljau. Courbes et surfaces à pôles. Technical report, A. Citroën, Paris, 1963.

[40] P. de Casteljau. Formes à pôles: Courbes et surfaces. *Mathèmatiques et CAO*, Vol 2, 1984.

[41] P. de Casteljau. Shape Mathematics and CAD. *Kogan Page*, 1986.

[42] J. A. de Reyna Martinez. Definition and study of an infinitely differentiable function with compact support. *Rev. Real Acad. Cienc. Exact. Fis. Natur.*, 76(1):21–38, 1982.

[43] L. T. Dechevsky, B. Bang, and A. Lakså. Generalized Expo-Rational B-splines. *International Journal of Pure and Applied Mathematics*, 57(1):833–872, 2009.

[44] L. T. Dechevsky, A. Lakså, and B. Bang. Expo-Rational B-splines. *International Journal of Pure and Applied Mathematics*, 27(3):319–369, 2006.

[45] L. T. Dechevsky, A. Lakså, and B. Bang. NUERBS form of Expo-Rational B-splines. *International Journal of Pure and Applied Mathematics*, 32(1):11–32, 2006.

[46] L. T. Dechevsky and P. Zanaty. Smooth GERBS, orthogonal systems and energy minimization. In *American Institute of Physics Conference Series*, volume 1570 of *American Institute of Physics Conference Series*, pages 135–162, December 2013.

[47] Lubomir Dechevsky. Beta-function b-splines: Definition and basic properties. *International Journal of Pure and Applied Mathematics*, 65, 01 2010.

[48] R. Descartes. *The Geometry of Rene Descartes*. Dover classics of science and mathematics. Dover Publications, 1954.

[49] M. P. do Carmo. *Differential geometry of curves and surfaces*. Prentic Hall, Inc., New Jersey, USA, 1976.

[50] M. P. do Carmo. *Riemannian Geometry*. Birkhäuser, Inc., Bosten, MA, USA, 1992.

[51] Tor Dokken, Tom Lyche, and Kjell Fredrik Pettersen. Polynomial splines over locally refined box-partitions. *Comput. Aided Geom. Des.*, 30(3):331–356, March 2013.

[52] J. J. Dongarra, Jeremy Du Croz, Sven Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, March 1990.

[53] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of fortran basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 14(1):1–17, March 1988.

[54] D. Doo. A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In *Proceedings on Interactive Techniques in Computer Aided Design*, pages 157–165, 1978.

[55] D. Doo and M. Sabin. Behavior of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, 1978.

[56] N. Dyn, D. Levin, and J.A. Gregory. A 4-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design*, 4(4):257–268, 1987. cited By 451.

[57] Nira Dyn and Kai Hormann. Geometric conditions for tangent continuity of interpolatory planar subdivision curves. *Computer Aided Geometric Design*, 29(6):332 – 347, 2012.

[58] Nira Dyn, Frans Kuijt, David Levin, and Ruud van Damme. Convexity preservation of the four-point interpolatory subdivision scheme. *Computer Aided Geometric Design*, 16(8):789 – 792, 1999.

[59] Nira Dyn, David Levine, and John A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. Graph.*, 9(2):160–169, 1990.

[60] Euclide and T. L. Heath. *The Thirteen Books of the Elements*. Number v. 3 in Dover classics of science and mathematics. Dover Publications, 1956.

[61] L. Euler. De Eximio usu Methodi Interpolationum in Serierum Doctrina. *Opuscula Analytica*, 1:157–210, 1783.

[62] J. Fabius. A probabilistic example of a nowhere analytic c^{∞}-function. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 5(2):173–174, 1966.

[63] G. Farin. Triangular Bernstein-Bezier patches. *Computer Aided Geometric Design*, 3(2):83–128, 1986.

[64] G. Farin. *Curves and Surfaces for CAGD*. Morgan Kaufmann, San Francisco, California, fifth edition, 2002.

[65] J. Fauvel, R. Wilson, and R.Flood (Eds.). *Möbius and his Band: Mathematics and Astronomy in Nineteenth-century Germany*. Oxford University Press, Oxford, England, fifth edition, 1993.

[66] T.H. Fay. The Butterfly Curve. *The American Mathematical Monthly*, 96(5):442–443, 1989.

[67] M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.

[68] M. S. Floater, K. Hormann, and G. Koz. A general construction of barycentric coordinates over convex polygons. *Advances in Computational Mathematics*, 24(1-4):311–331, 2006.

[69] M. S. Floater, G. Koz, and M. Reimers. Mean value coordinates in 3D. *Computer Aided Geometric Design*, 22(7):623–631, 2005.

[70] Michael S. Floater. The approximation order of four-point interpolatory curve subdivision. *Journal of Computational and Applied Mathematics*, 236(4):476 – 481, 2011. International Workshop on Multivariate Approximation and Interpolation with Applications (MAIA 2010).

[71] I. P. Gancheva and N. D. Delistoyanova. Euler Beta-function B-spline: definition, basic properties, and practical use in Computer Aided Geometric Design. Master theses, Narvik University College, Narvik, Norway, 2007.

[72] C. F. Gauss. *Theoria Interpolationis Methodo Nova Tractata*, pages 265–327. Göttingen, 1866.

[73] I. Ginkel, J. Peters, and G. Umlauf. Normals of subdivision surfaces and their control polyhedra. *Computer Aided Geometric Design*, 24(2):112–116, 2007.

[74] B. V. Gnedenko. *The theory of probability*. Translated from the fourth Russian edition by B. D. Seckler. Chelsea Publishing Co., New York, 1967.

[75] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[76] Ronald N. Goldman. Blossoming and knot insertion algorithms for b-spline curves. *Computer Aided Geometric Design*, 7(1):69 – 81, 1990.

[77] R. Goldmann. *Pyramid Algorithms: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling*. Morgan Kaufmann Publishers, San Francisco, Calefornia, 2003.

[78] G. H. Golub and F Van Loan. *Matrix Computations, 4th ed*. Johns Hopkins University Press, Boltimore, MD, 2012.

[79] W. J. Gordon. Blending-function method of bivariate and multivariate interpolation and approximation. *SIAM Journal on Numerical Analysis*, 8(1):158–177, 1969.

[80] A. Gray. *Modern Differentioal Geometry of Curves and Surfaces*. CRC Press, Inc., Boca Raton, Florida, first edition, 1993.

[81] T. N. E. Greville. The General Theory of Osculatory Interpolation. *Transactions of the Actuarial Society of America*, 45:202–265, 1944.

[82] Cindy M. Grimm and John F. Hughes. Modeling surfaces of arbitrary topology using manifolds. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 359–368, New York, NY, USA, 1995. ACM.

[83] H. Guggenheimer. Computing frames along a trajectory. *Comput. Aided Geom. Des.*, 6:77–78, February 1989.

[84] S. Guillet and J. C. Leon. Parametrically deformed free form surfaces as a part of variational model. *Computer Aided Design*, 30(1), 1998.

[85] Ayman Habib and Joe Warren. Edge and vertex insertion for a class of c^1 subdivision surfaces. *Computer Aided Geometric Design*, 16(4):223–247, 1999.

[86] E. Hartmann. Parametric G^n blending of curves and surfaces. *The Visual Computer*, 17:1–13, 2001.

[87] M.F Hassan, I.P. Ivrissimitzis, N.A. Dodgson, and M.A. Sabin. An interpolating 4-point c2 ternary stationary subdivision scheme. *Computer Aided Geometric Design*, 19(1):1 – 18, 2002.

[88] J. K. Haugland. Evaluating the fabius function. *ArXiv e-prints*, 1609.07999v1, September 2016.

[89] R. Henderson. A Practical Interpolation Formula. With a Theoretical Introduction. *Transactions of the Actuarial Society of America*, 9(35):211–224, 1906.

[90] Christoph M. Hoffmann. *Geometric and solid modeling: an introduction*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[91] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods*. Dover Publication Inc., New York, N.Y., 1994.

[92] H. Jeffreys and B. S. Jeffreys. L. F. Richarddon‘s methode. *Methods of Mathematical Physics*, 3rd ed.:288, 1988.

[93] W. A. Jenkins. Graduation Based on a Modification of Osculatory Interpolation. *Transactions of the Actuarial Society of America*, 28:198–215, 1927.

[94] S. A. Joffe. Interpolation-Formulae and Central-Difference Notation. *Transactions of the Actuarial Society of America*, 18:72–98, 1917.

[95] Kjetil André Johannessen, Trond Kvamsdal, and Tor Dokken. Isogeometric analysis using lr b-splines. *Computer Methods in Applied Mechanics and Engineering*, 269:471–514, 2014.

[96] J. Karup. Über eine Neue Mechanische Ausgleichungsmethode. In G. King, editor, *Transactions of the Second International Actuarial Congress*, pages 31–77, London, 1899. Charles and Edwin Layton.

[97] F Klok. Two moving coordinate frames for sweeping along a 3D trajectory. *Comput. Aided Geom. Des.*, 3:217–229, November 1986.

[98] L. Kobbelt. A Subdivision Scheme for Smooth Interpolation of Quad-Mesh Data. In *Eurographics*, 1998.

[99] Leif Kobbelt. $\sqrt{3}$-Subdivision. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH'00, pages 103–112, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[100] J. L. Lagrange. Leçons élémentaires sur les Mathématiques Données a l'école Normale. In J. A. Serret, editor, *Euvres de Lagrange*, volume 7, pages 183–287, Paris, 1877. Gauthier-Villars. Lecture notes first published in 1795.

[101] Ming-Jun Lai and Larry L. Schumaker. *Spline Functions on Triangulations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, 2007.

[102] A. Lakså. *Basic properties of Expo-Rational B-splines and practical use in Computer Aided Geometric Design*. Number 606 in unipubavhandlinger. Unipub, Oslo, 2007.

[103] A. Lakså. Non polynomial b-splines. In *41st International Conference "Applications of Mathematics in Engineering and Economics" AMEE '15*, volume 1690 of *American Institute of Physics Conference Series*, page 030001, 2015.

[104] A. Lakså, B. Bang, and L. T. Dechevsky. Exploring expo-rational B-splines for curves and surfaces. In *Mathematical methods for curves and surfaces: Tromsø 2004*, Mod. Methods Math., pages 253–262. Nashboro Press, Brentwood, TN, 2005.

[105] A. Lakså, B. Bang, and L. T. Dechevsky. Geometric modelling with Beta-function B-splines I. *International Journal of Pure and Applied Mathematics*, 65(3):339–360, 2010.

[106] A. Lakså, B. Bang, and L. T. Dechevsky. Geometric modelling with Beta-function B-splines II. *International Journal of Pure and Applied Mathematics*, 65(3):362–380, 2010.

[107] A. Lakså, B. Bang, and A. R. Kristoffersen. GMlib, a C++ library for geometric modeling. Technical report, Narvik University College, Narvik, Norway, 2006.

[108] Arne Lakså. Surfaces from Curves on Triangular Surfaces in barycentric coordinates. In Ivan Lirkov, Svetozar Margenov, and Jerzy Waśniewski, editors, *Large-Scale Scientific Computing*, pages 619–627, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[109] Arne Lakså and Børre Bang. Surface constructions on irregular grids. In Ivan Lirkov, Svetozar D. Margenov, and Jerzy Waśniewski, editors, *Large-Scale Scientific Computing*, pages 385–393, Cham, 2015. Springer International Publishing.

[110] J. M. Lane and R. F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(1):35–46, Jan 1980.

[111] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.*, 5(3):308–323, September 1979.

[112] Xin Li and Jianmin Zheng. Interproximate curve subdivision. *Journal of Computational and Applied Mathematics*, 244:36 – 48, 2013.

[113] C. T. Loop. Smooth Subdivision Surfaces based on Triangles. Master theses, University of Utah, Utha, USA, 1987.

[114] Charles Loop. A G^1 triangular spline surface of arbitrary topological type. *Comput. Aided Geom. Design*, 11(3):303–330, 1994.

[115] T. Lyche. *Discrete polynomial spline approximation methods*, volume 501/1976 of *Lecture Notes in Mathematics*, pages 144–176. Springer, Berlin/Heidenberg., 1976.

[116] T. Lyche and K. Strøm. Knot insertion for Natural Splines. *Annals of Numerical Mathematics*, 3:221–246, 1996.

[117] O. L. Mangasarian and L. L. Schumaker. Discrete Splines via Mathematical Programming. *SIAM Journal on Control*, 9:174–183, 1971.

[118] Martti Mäntylä. *An introduction to solid modeling*, volume 13. Computer Science Press, Incorporated, 1988.

[119] C. Mäurer and B. Jüttler. Rational approximation of rotation minimizing frames using Pythagorean-hodograph cubics. *Journal for Geometry and Graphics*, 3(2):141–159, 1999.

[120] L. Maurer. Über the Mittelwerte der Funktionen einer reellen Variablen. *Math. Ann.*, 47:263–280, 1896.

[121] D. S. Meek and D. J. Walton. Blending two parametric curves. *Computer-Aided Design*, 41:423–431, 2009.

[122] E. Mehlum. Nonlinear splines. *Computer Aided Geometric Design*, pages 173–207, 1974.

[123] E. Mehlum. Appell and apple (nonlinear splines in space). In Larry L. Schumaker Morten Dæhlen, Tom Lyche, editor, *Mathematical Methods for Curves and Surfaces*, pages 365–383. Vanderbilt University Press (Nashville & London), 1995.

[124] E. Meijering. A chronology of interpolation: From ancient astronomy to modern signal and image processing. In *Proceedings of the IEEE*, pages 319–342, 2002.

[125] J.Cotrina Navau and N.Pla Garcia. Modelling surfaces from planar irregular meshes. *Computer Aided Geometric Design*, 17(1):1 – 15, 2000.

[126] H. Olofsen. Blending functions based on trigonometric and polynomial approximations of the fabius function. In *Open Journal Systems*, Norsk Informatikk Konferanse, 2019.

[127] Francesco Patrizi, Carla Manni, Francesca Pelosi, and Hendrik Speleers. Adaptive refinement with locally linearly independent LR B-splines: Theory and applications. *Computer Methods in Applied Mechanics and Engineering*, 369:113230, 09 2020.

[128] Aleksander Pedersen, Jostein Bratlie, and Rune Dalmo. Spline representation of connected surfaces with custom-shaped holes. In Ivan Lirkov, Svetozar D. Margenov, and Jerzy Waśniewski, editors, *Large-Scale Scientific Computing*, pages 394–400, Cham, 2015. Springer International Publishing.

[129] Jörg Peters and Ulrich Reif. The simplest subdivision scheme for smoothing polyhedra. *ACM Trans. Graph.*, 16(4):420–431, October 1997.

[130] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition, 1992.

[131] L. Ramshaw. Blossoming: A Connect-the-Dots Approach to Splines. Report 19, Digital Systems Research Center, Palo Alto, CA., 1987.

[132] L. Ramshaw. Bézier and b-splines as multiaffine maps. In Rae A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, pages 757–776, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[133] L. Ramshaw. Blossoms are polar forms. *Computer Aided Geometric Design*, 6(4):323–359, 1989.

[134] Ulrich Reif. A unified approach to subdivision algorithms near extraordinary vertices. *Computer Aided Geometric Design*, 12(2):153–174, 1995.

[135] R.F. Riesenfeld. On chaikin's algorithm. *Computer Graphics and Image Processing*, 4(3):304 – 310, 1975.

[136] W. Romberg. Vereinfacthe numerische integration. *Det Kongelige Norske Vid. Selsk. Forl.*, 28:30–36, 1955.

[137] I. J. Schoenberg. Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions. Part A – On the Problem of Smoothing or Graduation. A First Class of Analytic Approximation Formulae. *Quarterly of Applied Mathematics*, IV(1):45–99, 1946.

[138] I. J. Schoenberg. Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions. Part B – On the Problem of Osculatory Interpolation. A Second Class of Analytic Approximation Formulae. *Quarterly of Applied Mathematics*, IV(2):112–141, 1946.

[139] I. J. Schoenberg. On spline functions. *Inequalities*, pages 255–291, 1967.

[140] I. J. Schoenberg. Cardinal Spline Interpolation. *CBMS-NSF Series in Applied Mathematics*, 12, SIAM, 1973.

[141] L. L. Schumaker. *Spline Functions: Basic Theory*. A Wiley-interscience publication. John Wiley & Sons Inc., New York, 1981.

[142] T. W. Sederberg, D. L. Cardon, D. G. Finnigan, J. Zheng, and T. Lyche. T-spline Simplification and Local Refinement. *ACM Transactions on Graphics*, 23(2):276–283, 2004.

[143] T. W. Sederberg, J. Zheng, A. Bakenow, and A. Nasri. T-splines and T-nurces. *ACM Transactions on Graphics*, 22(3):477–484, 2003.

[144] T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. In *SIGGRAPH '86: Proceedings of the 13th annual Conference on Computer Graphics*, pages 151–160, 1986.

[145] C. H. Séquin, K. Lee, and J. A. Yen. Fair, G^2- and C^2-continuous circle splines for the interpolation of sparse data points. *Computer-Aided Design*, 37(2):201–211, 2005.

[146] C. H. Séquin and J. A. Yen. Fair and robust curve interpolation on the sphere. Sketches and Application. In *SIGGRAPH '01: Proceedings of the 17th annual Conference on Computer Graphics*, page 182, 2001.

[147] W. F. Sheppard. Central-difference formula. *Proceedings of the London Mathematical Society*, 31:449–488, 1899.

[148] K.L. Shi, J.H. Yong, J.G. Sun, and J.C. Paul. G^n blending multiple surfaces in polar coordinates. *Computer-Aided Design*, 42:479–494, 2010.

[149] A. Sommerfeld. Eine besonders anschauliche ableitung des gaussischen fehlergesetzes. *Festschrift Ludwig Boltzman gewidmet zum 60. Geburtstage, 20. Februar 1904*, pages 848–859, 1904.

[150] M. Spivak. *A Comprehensive Introduction to Differential Geometry I*. Publish or Perish, Inc., Houston, Texas, USA, second edition, 1979.

[151] Jos Stam. Exact evaluation of catmull-clark subdivision surfacessubdivision surfaces at arbitrary parameter values. In *Proceedings of SIGGRAPH*, pages 395–404, 1998.

[152] ANSI/IEEE std. 754-2008. IEEE standard for binary floating-point arithmetic. Copyright standard, The Institute of Electrical and Electronical Engineers, Inc., New York, USA, 2008.

[153] M. Szivasi-Nagy and T.P. Vendel. Generating curves and swept surfaces by blended circles. *Computer Aided Geometric Design*, 17(2):197–206, 2000.

[154] Jieqing Tan, Xinglong Zhuang, and Li Zhang. A new four-point shape-preserving c3 subdivision scheme. *Computer Aided Geometric Design*, 31(1):57 – 62, 2014.

[155] T. N. Thiele. *Interpolationsrechnung*. B. G. Teubner, Leipzig, Germany, 1909. In German.

[156] I. Vardi. The Euler-Maclaurin Formula. *Computational Recreation in Mathematica*, pages 159–163, 1991.

[157] Ping Wang, Jinlan Xu, Jiansong Deng, and Falai Chen. Adaptive isogeometric analysis using rational pht-splines. *Computer-Aided Design*, 43(11):1438 – 1448, 2011. Solid and Physical Modeling 2011.

[158] W. Wang, B. Jüttler, D. Zheng, and Y. Liu. Computation of rotation minimizing frames. *ACM Trans. Graph.*, 27:2:1–2:18, March 2008.

[159] E. Waring. Problems Concerning Interpolations. *Philosophical Transactions of the Royal Society of London*, 69:59–67, 1779.

[160] J. Warren. Blending Algebraic Surfaces. *ACM Trans. Graph.*, 8(4):263–278, 1989.

[161] J. Warren. Barycentric coordinates for convex polytopes. *Advances in Computational Mathematics*, 6(1):97–108, 1996.

[162] E. W. Weisstein. Rose. http://mathworld.wolfram.com/Rose.html, 2006.

[163] H. Wenz. Interpolation of curve data by blended generalized circles. *Computer Aided Geometric Design*, 13(8):673–680, 1996.

[164] E. T. Whittaker. On the Functions which are Represented by the Expansions of Interpolation-Theory. *Proceedings of the Royal Society of Edinburgh*, 35:181–194, 1915.

[165] Wikipedia. Basic linear algebra subprograms — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Basic_Linear_Algebra_Subprograms&oldid=235926608, 2008. [Online; accessed 17-September-2008].

[166] A. Wiltsche. Blending curves. *Journal for Geometry and Graphics*, 9(1):67–75, 2005.

[167] K. C. Wu, T. Fernando, and H. Tawfik. Freesculptor: A computer-aided freeform design environment. In *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings*, pages 188–194, 2003.

[168] Lexing Ying and Denis Zorin. A simple manifold-based construction of surfaces of arbitrary smoothness. *ACM Trans. Graph.*, 23(3):271–275, August 2004.

[169] Denis Zorin, Peter Schroder, and Wim Sweldens. Interpolating subdivision for meshes with arbitrary topology. pages 189–192, 1996. Proceedings of the 1996 Computer Graphics Conference, SIGGRAPH ; Conference date: 04-08-1996 Through 09-08-1996.

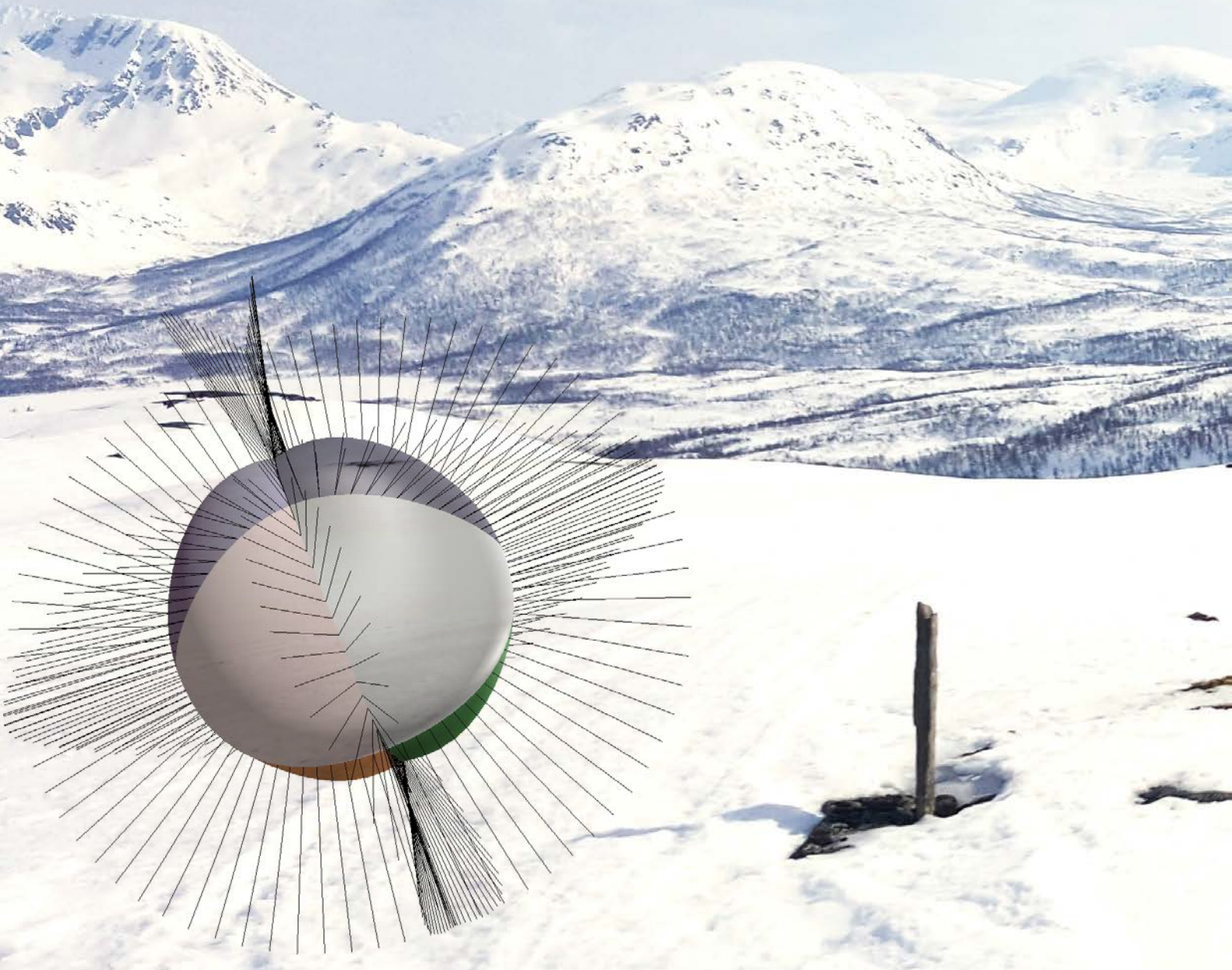
List of Acronyms

2D	–	Describes objects embedded in $\mathbb{R}^2$
3D	–	Describes objects embedded in $\mathbb{R}^3$
API	–	Application Programming Interface
C++	–	Object oriented programming language
CAGD	–	Computer Aided Geometric Design
ERB	–	Expo-Rational B-functions
ERBS	–	Expo-Rational B-splines
eVITA	–	e-Science, includes material from both Computational Science and Software Engineering as well as other topics such as graphics, virtual reality, general computer science
FFD	–	Free Form deformation method
GMlib	–	C++ library for geometric modeling and simulations, developed by the R&D Simulation Group at Narvik University College
GM_Wave	–	C++ wavelet library, developed by the R&D Simulation Group at Narvik University College
GPU	–	Graphic Processor Unit
GPGPU	–	GPU programming for General Purpose
GUI	–	Graphical User Interface
IEEE	–	The Institute of Electrical and Electronics Engineers, Inc.
Kameleon FireEx	–	A simulation program for fire simulation, developed by ComputIT
NaN	–	Not a number
NURBS	–	Non-uniform rational B-splines
NUERBS	–	The ERBS analog to NURBS
ODE	–	Ordinary Differential Equation
OpenGL	–	Software interface to graphics hardware
PDE	–	Partial Differential Equation
SINTEF	–	The Foundation for Scientific and Industrial Research at the Norwegian Institute of Technology
SISL	–	SINTEFs spline library
UIO	–	University of Oslo
UiT	–	The Arctic University of Norway

Index

Geometry – from Ancient Greek, earth measurement – has been an important ingredient of the development of science and later also industry, design, construction and production, and has consequently been important for industry/societal development in general.

Applied/industrial geometry is today a very important factor in, among other things, product development, virtual systems, systems for recognition and orientation and artificial intelligence.



GEOFO
Geometriforlaget
The geometry publishing house

Supported by
THE ARCTIC UNIVERSITY OF NORWAY • UIT •

9 788269 306514
ISBN 978-82-693065-1-4