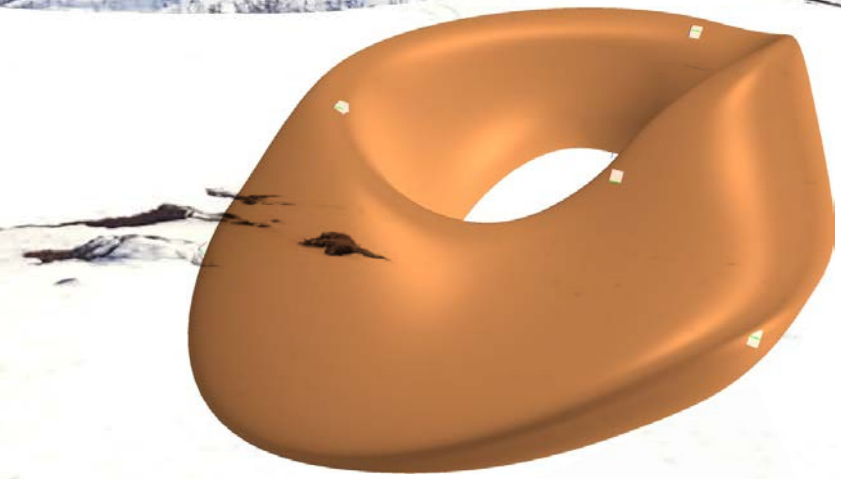
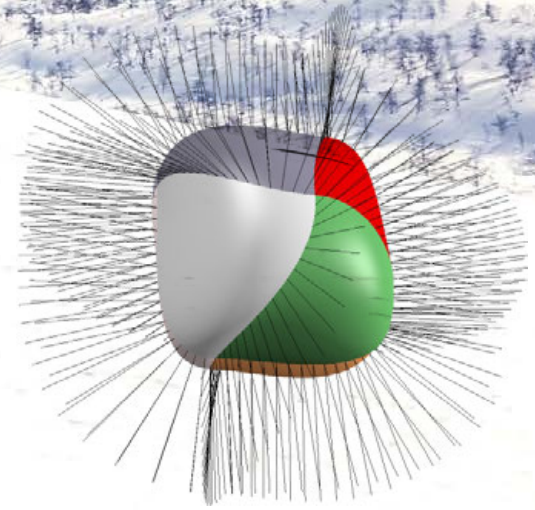
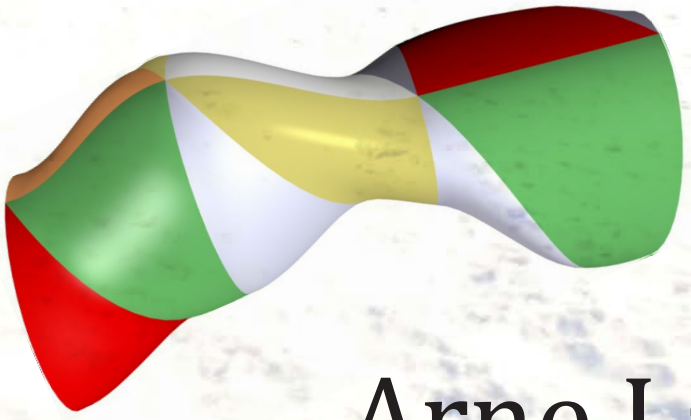
Blendingsteknikker i kurve- og flatekonstruksjoner







Arne Lakså

Arne Lakså

Blendingsteknikker i kurve- og flatekonstruksjoner

ISBN 978-82-693065-0-7

Opphavsrett © 2022 Arne Lakså CCBY

Enhver har rett til fritt å bruke bilder og annet materiale fra boken såfremt det fremkommer hvor det er hentet fra.

GEOFO
Geometriforlaget
The geometry publishing house

Støttet av
NORGES ARKTISKE UNIVERSITET
UIT

Narvik 2022

Arne Lakså
Institutt for datateknologi og beregningsorienterte ingeniørfag,
Fakultet for ingeniørvitenskap og teknologi,
UiT Norges arktiske universitet,
postboks 385, 8505 Narvik.

Geometriforlaget er et ideelt forlag for geometri/matematikk/programmering, og som jobber med å gjøre digitalt bøker gratis tilgjengelig. Papirutgaver kan i noen tilfeller kjøpes til kostpris, hovedsaklig trykkekostnader og frakt.

Geometriforlaget er støttet av UiT Norges arktiske universitet.

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

Forord

Geometri – som betyr jordmåling på gammelgresk – har vært en viktig faktor i utviklingen av vitenskapen og dermed også industri, design, konstruksjon og produksjon og da industri/samfunnsutvikling generelt.

For litt over 50 år så vi konturene av de første fagmiljøene innen geometrisk modellering. Pionerene i fagmiljøet var motivert av introduksjonen av datamaskiner og muligheten for å bruke disse i design, konstruksjon og produksjon. De startet derfor å utvikle metoder og algoritmer for kurve- og flaterepresentasjon på datamaskinen og beregninger på disse og å bruke dette i datagrafikk og simuleringer. Dette blei starten på utviklingen av en ny fagdisiplin som blei kalt geometrisk modellering og som inkluderte datastøttet geometrisk design, 3D modellering, algebraisk geometri, og beregninger på og med geometri. Data-assistert geometrisk design (Computer Aided Geometric Design – CAGD) er den sentrale delen av feltet. Det startet med å finne metoder og algoritmer for DAK/DAP (data-assistert konstruksjon og produksjon). I dag er imidlertid støtte for virtuell og blandet virkelighet, virtuell design, objekt-gjenkjenning, dataspill, simulatorer og animasjoner, virtuelle objekter i film og TV-produksjoner, støtte for kunstig intelligens, støtte for autonome objekter (fly, båter, biler) etc. vel så viktige områder.

I CAGD startet man med klassiske geometriske objekt som linje, sirkelbue, plan, kule, sylinder etc. Dette kom tydelig til uttrykk på formen til biler. Da man startet å bruke dataassistert produksjon blei bilene svært boks-lignende, noe som varte helt fram til slutten av 1980-tallet. I mellomtiden startet utviklingen av friformgeometri, det vil si bézier, hermite, B-splines, rasjonale varianter som rasjonale bézierflater og NURBS, og subdivisjonsflater. Denne utviklingen startet egentlig på 1960-tallet og gikk med full kraft frem til 2000-tallet, og er fortsatt en pågående utvikling som inkluderer T-splines, LR-splines og annen forbedring som Multivariate splines. Nye resultater inkluderer også introduksjon av generaliserte B-splines og dermed er det fortsatt en vei å gå. B-splines og beslektede geometribeskrivelser er i dag den viktigste representasjonen og vil nok være det i lang tid. Det er den defacto industrielle standarden. Det som kan betraktes som en ny generasjon kurve- og flatekonstruksjoner startet etter år 2000, og er følgelig i startfasen. Den inkluderer forskjellige typer kurver og flater konstruert ved hjelp av blendingsteknikker.

For meg akselererte arbeidet med blendingsteknikk sommeren 2003. Jeg var på kontoret til min kollega Børre Bang for å diskutere forbedringer av et C++ -programmeringsbibliotek for geometrisk modellering (se [108]). Lubomir Dechevsky kom da og spurte om vi kunne se på en funksjon han mente var grenseverdien til polynomiske B-splines når

graden (og dermed antall skjøtverdier) går mot uendelig (beskrevet i [44]). Siden vi den gang jobbet med den grafiske delen av programmeringsbiblioteket, implementerte vi denne nye basisfunksjonen og prøvde den på kurver i $\mathbb{R}^3$. Resultatet var en stykkevis lineær kurve akkurat som en 1.-grads B-spline kurver, men det var en stor forskjell; denne nye kurven var C^∞-glatt men samtidig stykkevis lineær, dvs. G^0. Etter en kort diskusjon byttet vi ut koeffisientene til kurven med noe vi kalte “lokale kurver" og da dukket en G^∞-glatt kurve opp. Lubomir foreslo navnet “Expo-Rational B-splines" (ERBS). Et par år senere foreslo Dechevsky å også bruke Beta-funksjoner som basis og Beta function B-splines, BFBS blei utviklet. ERBS blei dermed utvidet til et mer generelt konsept som vi da kalte “Generalized Expo-Rational B-splines" (GERBS).

Noen år senere ble konseptet med B-funksjoner introdusert. Rasjonale og ekspo-rasjonale typer som på et tidligere tidspukt var publisert av andre forfattere ble sammen med nye klasser av B-funksjoner inkludert. Blant disse var Fabius-funksjonen og generelle trigonometriske B-funksjoner. Med dette ble den orginale ekspo-rasjonale B-funksjonen bare en klasse av mange, og det var derfor naturlig å omtale teknikken som blendingsplines. Samtidig er det viktig å vise at den er bygget opp som en B-spline og er en B-spline av 2.-orden (analogt til 1.-grads polynomisk B-splines), med alle egenskapene som følger av å være en B-spline.

Dette manuskriptet var opprinnelig ment å handle utelukkende om blendingsteknikk. Men på grunn av innspill og spørsmål fra studenter, og at dette manuskriptet også er preget av at deler av det har blitt brukt som kompendier i emner i masterprogram i Data/IT ved UiT Norges arktiske universitet, bestemte jeg meg for å utvide det, dvs. å inkludere noe materiale om grunnleggende geometri og splinemetoder generelt. Manuskriptet har allerede en lang historie og har hele tiden vært tilgjengelig for studenter som har fulgt mine kurs. Et første utkast fantes allerede i 2009 (engelsk versjon). Denne norske utgaven er i hovedsak identisk med en tilsvarende engelskspråklig versjon.

Anerkjennelser
Først vil jeg uttrykke min takknemlighet overfor mine kolleger Lubomir T. Dechevsky og Børre Bang. De har vært helt avgjørende for den første utviklingen av ERBS. Deretter alle ph.d.-studenter som har vært involvert i arbeidet, både i deres ph.d.-periode og etterpå; Joakim Gundersen, Arnt Kristoffersen, Peter Zanaty, Rune Dalmo, Jostein Bratlie, Hans Olofsen, Tatiana Kravetc, Tanita Fossli Brustad og Aleksander Pedersen. I tillegg er jeg også takknemlig overfor Dag Nylund for tilbakemeldinger på teksten og spesielt Tom Lyche og Knut Mørken for svært verdifulle diskusjoner på et tidlig stadie, og til Malcolm Sabin som var opponent på mitt doktorgradsforsvar og der stilte noen viktige spørsmål.

Til slutt vil jeg rette en stor takk til min kone Marit for å ha bært over med meg gjennom hele dette arbeidet.

Oktober 2022 Arne Lakså

Innhold

Kapittel 1

Introduksjon

Utvikling av geometri kan spores tilbake til tidlig egyptisk-, babylonsk- og vedisk-India-perioder. I disse tidlige periodene var fokuset på sirkler, trekanter, lengde, areal, volum etc. Dette var i hovedsak knyttet til empirisk forsøk og erfaringer, og var tenkt bruk i praktiske anvendelser innen oppmåling, konstruksjon, astronomi og ulike håndverk. De mer modne periodene begynte imidlertid med grekerne. De startet utviklingen av geometri slik vi kjenner den og som vi kan følges langs en linje som grovt sett deler historien inn i følgende fem perioder:

1. *Grekernes syntetiske geometri* (600 f.Kr. til 200 f.Kr.). Gresk geometri avsluttes i hovedsakelig av Arkimedes. Ved siden av eller etter den greske perioden ble geometri på forskjellige måter utviklet delvis parallelt av indisk, kinesisk og arabisk geometri. Gresk geometri var imidlertid den direkte forgjengeren til den analytiske geometrien omtalt i neste punkt fordi Euclides 13 bøker om geometri, med tittelen “Elementer i geometrien" [60], var den direkte forgjengeren.

2. *Starten av analytisk geometri* (1600 til 1650), der den syntetiske geometrien til Guldin, Desargues, Kepler og Roberval blir smeltet sammen med koordinatgeometrien til Descartes [48] og Fermat.

3. *Beregninger på geometri* (1650 til 1800), inkluderer følgende navn; Newton, Leibniz, Bernoulli’s, Clairaut, Maclaurin, Euler og Lagrange (som alle i stor grad kan sies å være analytikere i stedet for geometrikere).

4. *Starten på utvikling av moderne geometri*, (1800-tallet). Det vil si Lobatsjovsk geometri, projektiv geometri, analytisk geometri, ikke-euklidsk geometri og differensialgeometri. I tillegg er det verdt å nevne noen spesifikke resultat, Riemannsk geometri, Kleins Erlangen-program for klassifisering, Lie-grupper, homogene koordinater, Hilberts aksiomer.

5. *Ny avansert geometri og ny anvendt geometri*. Dette inkluderer moderne algebraisk topologi/geometri og nye utviklinger innen differensialgeometri. For disse finner vi anvendelse i for eksempel moderne fysikk. Andre områder er fraktal geometri, diskret geometri og ny anvendt geometri. Utviklingen av anvendt geometri er i hovedsak knyttet til konkrete anvendelser og er hovedtema videre i denne boken.

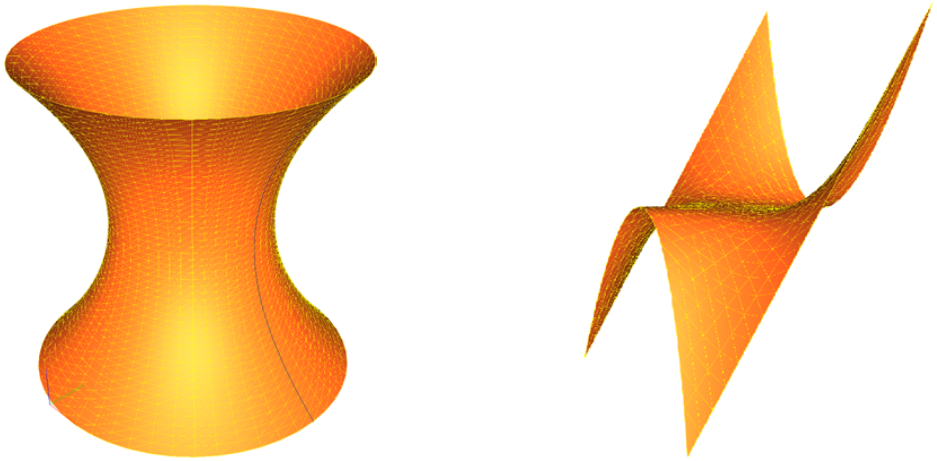
Figur 1.1: *Euklid av Alexandria, "geometriens far". En statue som står inne på "Oxford University Museum of Natural History".*

På 1950-tallet akselererte automatiseringsprosessen i industrien. Samtidig ble datamaskinen introdusert. Til sammen førte dette til et sterkt behov for ny utvikling av anvendt geometri. Dermed begynte et geometrifellesskap å utvikle seg. En ny gren innen geometri dukket opp, industriell geometri. Pionerene i denne perioden var ofte koblet til den mest avanserte industrien; skips-, fly- og bilindustrien, og var motivert av et ønske om å bruke datamaskiner til design, konstruksjon og produksjon. Målet var å utvikle metoder og algoritmer for kurve- og flaterepresentasjoner og beregninger på disse, for så å anvende disse i datagrafikk og simuleringer. Ved å gjøre dette startet de utviklingen av en ny disiplin kalt geometrisk modellering, og som inkludert dataassistert geometrisk design, 3D modellering, beregningsgeometri, digital geometri og piksel/voksel-partisjonering. Det startet med utvikling av metoder og algoritmer for dataassistert konstruksjon og produksjon, DAK/DAP, og ble fulgt av modellering av olje- og gruvefelt, menneskekropper etc. for simuleringsformål. I dag er imidlertid støtte til utvikling av virtuell virkelighet, virtuelt design, virtuell prototyping, virtuell konstruksjon, dataspill, simuleringer og animasjoner i filmer og TV-produksjoner vel så viktige område for geometrisk modellering.

Denne boken forsøker å koble dagens industrielle standard, B-splines, med dens forløpere og etterkommere, blant annet med nye blendingsteknikker i konstruksjon av kurver og flater. Noen ganger vil vi også se på geometriutviklingen i et historiske perspektiv.

I figur 1.1 vises et bilde av en statue av geometriens far, Euclid. De 13 bøkene om geometri "$\Sigma\tau o\iota\chi\varepsilon\iota\alpha$"[1] har vært og er fortsatt viktige for å få innsikt i geometri.

[1] som er gresk og betyr Elementer og som foreligger i en engelsk oversettelse i [60]



Figur 1.2: *Til venstre ser vi en rotasjonsflate som er laget ved å rotere en kurve* 360°. *Til høyre vises en "apesadel" som er laget direkte fra en formel.*

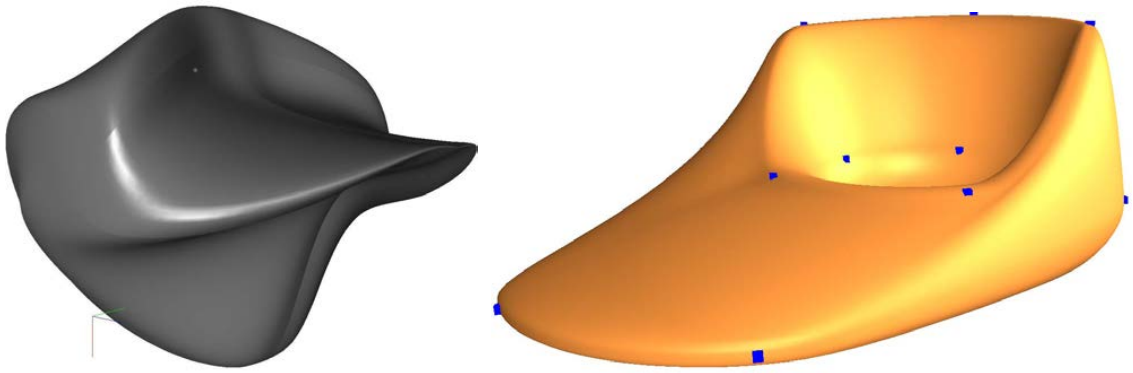
1.1 Industriell geometri

Industriell geometri betyr naturlig nok geometri for industrielle formål. Eksempler er design og konstruksjon, verktøystyring, gjenkjenning, eller det kan være beregninger på geometri. Industriell geometri er i dag viktig for områder som:

- ✓ Datagrafikk
- ✓ Data-assistert design - DAK (produktutvikling)
- ✓ Data-assistert produksjon - DAP (produksjonstyring)
- ✓ Virtuell og sammensatt virkelighet VR og AR.
- ✓ Virtuell kunst (virtuelle skulpturer, matematiske skulpturer, ...)
- ✓ Simuleringer og animasjoner (vitenskapelige simuleringer, filmer, spill, ...)
- ✓ Kartrelaterte applikasjoner, geodesi
- ✓ Selvkjørende biler / busser / lastebiler / båter / fly / droner / romfartøy
- ✓ Beregninger generelt, for eksempel styrkeberegninger, termodynamikk, fluidmekanikk, kinematikk, vibrasjon, resonans etc.
- ✓ Kunstig intelligens, objektgjenkjenning, ansiktsgjenkjenning, ...

Industriell/anvendt geometri er hovedsakelig for bruk på datamaskiner. I figur 1.2 vises et par eksempler. På venstre side av figuren vises en rotasjonsflate, som er laget ved å rotere en kurve 360°, se forøvrig kapittel 9.2. Til høyre vises en "apesadel", en flate som er laget direkte fra en formel. Formelen har merkelappen (9.1) og vi finner den på side 169.

Kurver, flater, volum, ... kan beskrives implisitt eller parametrisk. Parametrisk beskrivelse er mest brukt. En viktig type parametrisk geometri er splines. I dag er splinefamilien stor. Vi skal senere se på klassiske kubiske splines, Hermite splines, B-splines, NURBS, uniform splines og subdivisjon, sirkelsplines og blendingsplines. Vi skal også komme innom "lappeteppe" (patchwork på engelsk), spesielt med trekanter. Dette er en type flater som er satt sammen av mange delflater/trekanter på en mer eller mindre glatt måte.



Figur 1.3: *På venstre side ser vi en flate som i utgangspunktet var en kule, mens flaten til høyre opprinnelig var en torus. Begge flatene er endret ved først å bruke interpolering og deretter interaktiv redigering, dvs. flytting/rotering av interpolasjonspunktene.*

Det er ulike bruksområder for geometri. For eksempel er det behov for:

- Statisk geometri av høy kvalitet.
- Geometri som gjør presisjonsberegninger mulig.
- Geometri som enkelt kan formes, kopieres eller omformes.
- Geometri som dynamisk kan endre form i “sanntid".

Ulike typer geometri har ulike egenskaper. For eksempel er B-splines god for høy kvalitet og beregninger, subdivisjonsflater er best for grafikk, bilder, dataspill, og blendingsplines er bra for vanskelig formgiving og dynamisk endring av form. I figur 1.3 ser vi to flater som er eksempler på blendingsplines. Til venstre er en kule blitt kopiert av et lappeteppe av trekantede flater som deretter er endret til et “andehode", til høyre er en torus blitt kopiert av et lappeteppe av firkantede flater som så er endret til å ligne en “andefot".

1.2 Geometrisk modellering

Begrepet geometrisk modellering kom først i bruk på slutten av 1960-tallet, en tid med rask utvikling innen datagrafikk og datastøttet design og produksjonsteknologi. Disiplinen geometrisk modellering er en sammenhengende, men noe løst integrert samling av matematiske metoder som vi bruker for å beskrive formen til et objekt eller for å uttrykke en fysisk prosess i form av en passende geometrisk metafor. Disse metodene inkluderer;

✓ Datastøttet geometrisk design (internasjonalt forkortet CAGD) bruker formelverket for kurver og flater til modellering. Formlene er vanligvis parametriske uttrykk hentet fra differensialgeometrien, støttet av interpolasjons- og approksimasjonsteori. Det er her vi egentlig finner røttene til moderne geometrisk modellering.
✓ 3D modellering som vanligvis betegnes som konstruktiv solid geometri (CSG), lar oss kombinere enkle former for å lage komplekse solide volum modeller
✓ Algebraisk geometri er den moderne utvidelsen av klassisk analytisk geometri, inkludert differensialgeometri.

✓ Beregningsgeometri er opptatt av design og analyse av algoritmer, og er relatert til numeriske metoder, beregningsteori og kompleksitetsanalyse.

Imidlertid utvides geometrisk modellering stadig og inkluderer nå også fraktal og digital geometri og rompartisjonering.

Det har vært mest vanlig å studere geometriske objekter i planet - $\mathbb{R}^2$ eller i rommet - $\mathbb{R}^3$. Mange av verktøyene og prinsippene kan imidlertid også brukes om objektene er i et rom av hvilken som helst endelig dimensjon. I dag er modellene i datastøttet design og produksjon - DAK/DAP, imidlertid stort sett tre- eller firedimensjonale (3D pluss tid). Det samme gjelder anvendte tekniske områder som konstruksjon- og maskinteknikk, arkitektur, geologi, helse og et stort felt vi kan kalle virtuelle systemer, dataspill, VR-systemer og simulatorer. Geometrisk modellering er grunnlaget for applikasjoner som animasjon og spesialeffekter for avansert modelleringsprogramvare for industriell design og arkitektur og for 3D-printere. Geometriske modeller representerer former og romlige forhold til miljøet som studeres, noe som muliggjør en mye dypere analyse enn det ellers ville vært mulig å gjøre. Hvordan algoritmene er og hvordan disse modellene er kodet er egentlig det vi kaller datastøttet geometrisk design (CAGD).

For **datastøttet geometrisk design** - CAGD var en viktig begivenhet en konferanse holdt ved “University of Utah" i 1974, organisert av Barnhill og Riesenfeld [7]. Før denne konferansen kan vi si at feltet var i sitt embryonale stadium, etter denne konferansen var det klart at CAGD (Computer Aided Geometric Design) var født som en disiplin. I dag er CAGD en moden disiplin, hvor B-spline og splinemetoder er industrielle standarder og implementert i tusenvis av applikasjoner. Feltet er faktisk enda mer spennende enn noen gang, fordi feltets betydning stadig utvides med introduksjonen av virtuelle og utvidede verdener, dataspill, animasjonsfilmer, sosiale verdener, kunstig intelligens, gjenkjennelse og selvstyring. De primære typene av interesse er kurver, flater og volum beskrevet som splines (NURBS), lapper (“patcher"), subdivisjonsflater, samt algoritmer for å generere, analysere og manipulere dem. Denne boken vil gi en oversikt over feltet og se på nye utviklinger i CAGD og dets applikasjoner, inkludert men ikke begrenset til følgende:

- Å samle inn og å spre informasjon om datastøttet design.
- Å gi fellesskapet metoder og algoritmer for å representere og behandle kurver og flater.
- Å vise datastøttet geometrisk design ved å bruke interessante applikasjoner.
- Å kombinere kurve- og flatemetoder med datagrafikk.
- Å forklare vitenskapelige fenomener ved hjelp av datagrafikk.
- Å vise samspillet mellom teori og anvendelse.
- Å avsløre uløste problemer i praksisen.
- Å utvikle nye metoder innen datastøttet geometri.

I denne boken behandler vi også en ny generasjon konstruksjonsmetoder for kurver og flater som er basert på blendingsteknikker. Historisk er dette ikke nye teknikker. Både Coons patch [26] og forskjellige typer “fillet"-konstruksjoner bruker blendingsteknikker, og sirkelsplines [164] har mye til felles med “ekspo-rasjonal B-spline"-konstruksjonen som fra 2005 og fremover har blitt presentert i en rekke artikler, for eksempel [105, 44, 45, 102, 43, 106, 107].

1.3 Algoritmisk språk

Med “algoritmisk språk" mener vi språket som brukes for å beskrive algoritmene som presenteres. All programmering er laget i C++ . Dette gjenspeiles selvsagt også i det algoritmiske språket. En viktig faktor i definisjonen av syntaksen er at den skal være kompakt og samtidig tydelig og lett å forstå. Elementene som beskriver det algoritmiske språket er:

- ✓ Notasjonen i algoritmene er en forenkling, samt en blanding av C++ og en matematisk notasjon.
- ✓ C++ -template notasjon er bruket. Også *standard template library (stl)* brukes - for eksempel vektor<double>, en vektor av tall i dobbel presisjon.
- ✓ For å gjøre notasjonen mer kompakt, er “{"og “}kun markert med innrykk.
- ✓ En rutine kan uttrykkes som **et sett**, som eksempel $vector\langle T\rangle\ C = \{D^j c(t)\}_{j=0}^{n}$. Den vil imidlertid alltid bli etterfulgt av en forklarende kommentar.
- ✓ Vanlig C++ -standard “//" brukes for kommentarer.

Nedenfor er et eksempel, en implementering av en funksjon **bspline** definert i seksjon 6.2.9. C++ -koden vises først, deretter algoritmen ved bruk av det algoritmiske språket.

```
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);	// Retur vektoren, dimensjon $d+1$.
vector⟨double⟩ w(d);	
$b_1 = W_{1,\varsigma}(t;\tau)$;	// se (6.11) på side 82
$b_0 = 1 - b_1$;	// Den generelle Cox/deBoor algoritmen for
for (*int* $i = 2; i \le d; i++$)	// - B-splines, lager settet av alle aktuelle
for (*int* $j = 0; j < i;$ j++)	// - B-spline verdiene av grad d for en gitt t
$w_j = W_{i,\varsigma-j}(t;\tau)$;	// - og hvor $\tau_\varsigma \le t < \tau_{\varsigma+1}$.
$b_i = (1 - w_0)\, b_{i-1}$;	
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;	

Dette eksemplet finner du på side 96, algorithm 3. C++ -koden i dette eksemplet brukes i “evaluatoren" for B-splines. Normalt beregnes også de deriverte. Kode for det finner du i Algoritmen 4 på side 97.

1.4 Oversikt over kapitlene

Boken er delt inn i 4 deler med tilsammen 14 kapitler og 3 vedlegg. Delene er:

- ✓ Introduksjon
- ✓ Kurver
- ✓ Flater
- ✓ Vedlegg

De tre innledende kapitlene er:

Kapittel 1 gir leseren en historisk bakgrunn, først generelt om geometri, så mer spesifikt om industriell geometri og geometrisk modellering. Kapittelet gir også en beskrivelse av algoritmespråket som er brukt i boken, samt denne oppsummeringen.

Kapittel 2 handler om matematiske rom, ulike typer koordinater og notasjoner. Dette kapittelet er viktig for at leseren bedre skal forstå anvendt geometri, formlene, algoritmene og være i stand til å implementere geometri.

Kapittel 3 handler om programmering av geometri, hva affine og projektive rom betyr for programmering, hvorfor indreprodukt er den mest sentrale funksjonen og hvordan lokale koordinatsystem fungerer. I tillegg får man noen verktøy for interaktiv design og simulering.

De fem kapitlene om kurver er:

Kapittel 4 handler om parametrisering av kurver generelt, samt litt dypere om polynombaserte kurver av forskjellige format, og noen grunnleggende algoritmer for å beregne posisjon og deriverte på disse kurvene.

Kapittel 5 handler om klassisk interpolasjonsteori. Den tar for seg dividerte diferanser, om ulike former for interpolasjon samt hermiteinterpolasjon. Til slutt ser vi på polynombaserte stykkevise kurver som er mer praktisk å bruke ved interpolering.

Kapittel 6 handler om B-splines og B-spline kurver av ulike typer, kubisk spline interpolasjon, historien til B-splines, moderne B-splines og uniforme B-splines og subdivisjonskurver.

Kapittel 7 handler om blending, det vil si blendingsfunksjoner og hvordan man bruker disse. B-funksjoner, forkortelse for blendingsfunksjoner, finnes i et stort utvalg, og dette kapittelet viser noen av de viktigste.

Kapittel 8 handler om B-splines med B-funksjoner, og også hvor vi erstatter kontrollpunkt med kontrollkurver. Dermed introduserer dette kapittelet konseptet blendingsplines. Vi får se mange eksempler som viser dynamisk formendring. Til slutt blir det introdusert en kurvekonstruksjon hvor enhver parametrisk kurve, uansett formel,

kan utvides med en skjøtvektor og et sett “koeffisienter", som alle egentlig er matriser, og som gjør at kurven kan endre form.

De seks kapitlene om flater er:

Kapittel 9 introduserer parametriske flater. Både definisjoner og aspekter ved implementering blir diskutert. I tillegg vises forskjellig flatekonstruksjon.

Kapittel 10 handler om subdivisjonsflater, dvs. uniform diskret B-splines hvor vi starter med et sett med punkt organisert i “polygon", og ender opp med et mye tettere sett med punkt og “polygon".

Kapittel 11 handler om en blendingsflatekonstruksjon hvor to flater blendes, og hvor randkurvene, med posisjon og deriverte, er gitt. Dvs. et alternativ til Coons patch.

Kapittel 12 introduserer blendingspline-tensorproduktflater. Både definisjon og aspekter ved implementering diskuteres. I tillegg introduseres komplette “evaluatorer" og bézierflater som lokale flater. Videre diskuteres hermittinterpolasjon samt formendring ved bruk av affin transformasjon av lokale flater. Til slutt blir et nytt konsept introdusert, der enhver parametrisk flate, uansett formel, kan utvides med to skjøtvektorer og tilhørende koeffisienter (matriser) slik at flaten kan endre form.

Kapittel 13 omhandler trekantede flater. Vi får en introduksjon av béziertrekanter med homogene barysentriske koordinater. Deretter introduseres B-funksjoner i homogene barysentriske koordinater, og de grunnleggende egenskapene diskuteres. Deretter introduseres blendingstrekanter, inkludert to forskjellige typer lokale trekanter, béziertrekanter og sub-trekanter i generelle parametriserte flater. Den siste fører til flateapproksimasjon basert på trianguleringer. Til slutt vises flere eksempler.

Kapittel 14 handler om flater som er et satt sammen av trekantede flatestykker. Hver trekant er et resultat av å blende trekantede deler av lokale flater som er koblet til sine respektive interpoleringspunkt. Resultatet er kontinuerlig og glatt i interpoleringspunktene, men ikke nødvendigvis glatt over kantene mellom trianglene. Det finnes imidlertid en dual flate satt sammen av firkantede “lapper" som gir en glatt flate.

Og til slutt har vi tre vedlegg:

Vedlegg A viser implementering av en “evaluator" for en ERB-funksjon av type 1. Vi viser en pålitelig, presis og effektiv måte å beregne ERB-funksjonen. Reliabilitet basert på IEEE-standarden for binær flyttallsaritmetikk diskuteres. Algoritmer for evaluering, dvs. beregninger av funksjonsverdi inkludert deriverte introduseres. Tester blir utført både når det gjelder presisjon og effektivitet. Til slutt introduseres en pålitelig og veldig rask “evaluator", pakket inn i en C++ -klasse.

Vedlegg B omhandler eksterne programmeringsbibliotek som løser lineæralgebra problemer. Vedlegget inneholder også en del om å utnytte dataressurser bedre, dvs. heterogen databehandling og parallellisering.

Vedlegg C Her er en rekke bevis fra forskjellige kapitler samlet.

Kapittel 2

Matematiske rom og notasjoner

Gjennom tidene har “rom" vært en geometrisk abstraksjon av det tredimensjonale rommet vi observerer i det virkelige liv. I matematikken er dette relatert til det to- eller tredimensjonale rommet med euklidsk geometri, så vel som generaliseringene av disse til høyere dimensjoner. Begrepet **euklidsk** skiller disse rommene fra de “krumme" ikke-euklidske rommene, og de er oppkalt etter den greske matematikeren Euklid av Alexandria (300 f.Kr.). Det er vanlig å definere euklidske rom med kartesiske koordinater, dvs. koordinater som typisk beskrives av et senterpunkt kalt origo og n koordinatakser, dvs. enhetsvektorer som er ortogonale til hverandre. Det er ikke uvanlig å betegne rommene for $\mathbb{E}^n$ hvis vi ønsker å understreke dens euklidske natur, men oftest brukes $\mathbb{R}^n$ siden sistnevnte da antas å ha standard euklidsk struktur.

I moderne matematikk er et rom definert som et sett med objekter som har noen tilhørende strukturer. Alternativt beskrives de som ulike typer mangfoldigheter, som er rom som lokalt ligner på euklidske rom, og hvor egenskapene i stor grad er definert rundt lokal sammenheng mellom punkt som ligger på mangfoldigheten. Kurver og flater er mangfoldigheter hvis de er regulære og ikke selvskjærende.

Det er imidlertid mange forskjellige matematiske objekter som kalles rom. Noen vektorrom som for eksempel funksjonsrom kan ha uendelig dimensjon, og en forestilling om avstand som er svært forskjellig fra det vi tenker på som avstand i euklidske rom. Det finner også topologiske rom som erstatter begrepet avstand med en mer abstrakt idé om nærhet.

Matematiske rom danner ofte et hierarki, det vil si at ett rom kan arve alle egenskapene til et overordnet rom. For eksempel er alle indreproduktrom også normerte vektorrom, fordi indreproduktet induserer en norm i indreproduktrommet slik at

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

Ved siden av euklidske rom, vektorrom og endeligdimensjonale funksjonsrom skal vi også se på kompakte rom, affine rom, projektive rom og grassmannien, og vi skal se på noen avbildninger (overganger) mellom rom.

2.1 Euklidske rom, kartesiske koordinater og vektorrom

Notasjonen i denne boken følger den vanligste standarden. Geometri beskrives oftest i planet eller i rommet, og funksjonene gir da typisk vektorer eller punkt (forskjellen på vektor og punkt vil bli beskrevet i seksjon 2.4). Vektorene i $\mathbb{R}^n$ danner sammen med operasjonene et vektorrom. Et vektorrom er en matematisk struktur av n-dimensjonale vektorer som kan legges sammen og multipliseres/skaleres med tall som her kalles skalarer. Skalarer er normalt reelle tall. Resultatet av disse operasjoene er alltid n-dimensjonale vektorer.

En liste som forklarer notasjoner er:

- ✓ Vi benevner euklidske rom som $\mathbb{E}^d$ eller $\mathbb{R}^d$, der d er dimensjonen til rommet, $\mathbb{R}^2$ er planet og $\mathbb{R}^3$ er 3D-rommet.
- ✓ Kartesisk koordinatsystem, er mest vanlig å bruke i euklidske rom. Den spesifiserer hvert punkt unikt med et sett med numeriske koordinater, som er avstandene, med fortegn, fra punktet til et sett med faste linjer som står vinkelrett på hverandre, målt i samme lengdeenhet, dvs. p = (x, y, z).
- ✓ Hver referanselinje i et kartesisk koordinatsystem kalles en koordinatakse eller bare akse, og punktet der de møtes kalles origo, $\mathscr{O} = (0,0,0)$. Koordinatene kan også defineres som posisjonene til de perpendikulære projeksjonene av punktet på aksene, uttrykt som avstander med fortegn fra origo.
- ✓ En vektor eller et punkt er angitt med en bokstav, latinsk eller gresk. En vektor kan vises enten som en radvektor (liggende) eller en kolonnevektor (stående), dvs.

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

- ✓ Et indreprodukt mellom to vektorer r og $s \in \mathbb{R}^d$, $d > 1$ er angitt ved

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

  Merk at hvis $\langle r, s \rangle = 0$, så er r og s ortogonale, dvs. vinkelen mellom dem er 90°. Et indreprodukt i et euklidsk rom er det samme som et skalar- og prikk-produkt.
- ✓ Et vektorprodukt i $\mathbb{R}^3$ (beslektet med et kileprodukt som også kalles som et utvendig produkt) er; gitt vektorene $r = (r_1, r_2, r_3)$ og $s = (s_1, s_2, s_3)$:

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

  Det kan vises at vektorproduktet er ortogonalt til begge sine vektorer, dvs.

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

✓ Et kileprodukt i $\mathbb{R}^2$ er, gitt vektorene $r = (r_1, r_2)$ og $s = (s_1, s_2)$:

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

Et kileprodukt i $\mathbb{R}^2$ er det samme som et "rotert" indreproduktet

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

hvor vektoren s^L er vektor s rotert 90° mot klokken.

2.2 Homeomorfi, diffeomorfi og mangfoldigheter

Vi skal kort (og noe overfladisk sett fra et matematisk ståsted) se på det matematiske grunnlaget for parametriske kurver og flater. Til de lesere som ønsker å studere dette mer grundig anbefaler vi å lese Spivak [151] eller DoCarmo [49, 50].

Vi starter med å forklare homeomorfi. Fra gresk, betyr det noe sånt som "liknende form", og i en geometrisk sammenheng kan det sammenlignes med et ark av "leire" som blir deformert ved strekking, bøying, ... men hvor vi ikke kan klippe og lime fordi naboelementer må forbli naboelementer. En mer formell definisjon er

Definisjon 2.1. *En homeomorfi , dvs. en kontinuerlig transformasjon, er en ekvivalensrelasjon (relasjon mellom to "like objekt") og en-til-en korrespondanse mellom punkt i to geometriske objekter eller topologiske rom, som er kontinuerlig begge veier. Det er en avbildning som bevarer alle de topologiske egenskapene til et gitt rom. To objekt/rom med en homeomorfi mellom seg kalles homeomorfe, og fra et topologisk synspunkt er de like. En transformasjon / avbildning / funksjon / overgang er en homeomorfi hvis den:*

- *er en bijeksjon, dvs. én til én og på (hele),*
- *er kontinuerlig,*
- *og den inverse funksjonen er kontinuerlig.*

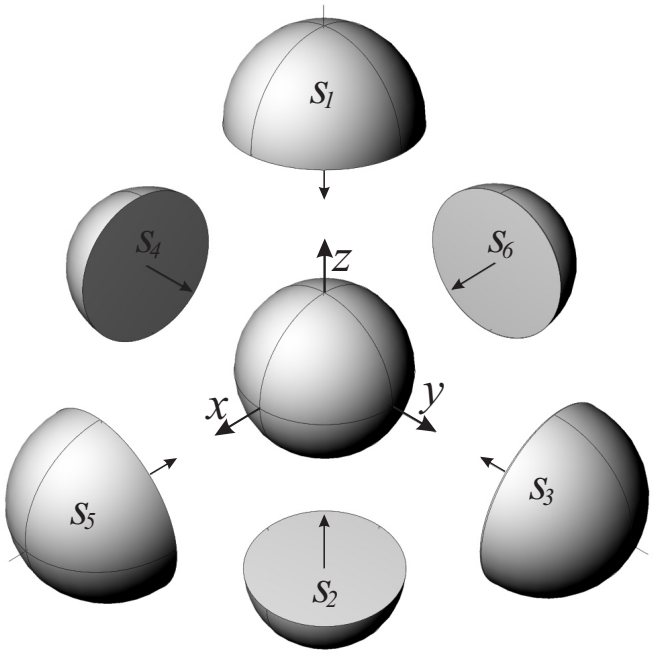
Affine transformasjoner som rotasjon, skalering, translasjon og skjær er blant de mest vanlige typene geometrisk homeomorfier. En homeomorfi som også bevarer avstander kalles en isometri. En isometri er som å bøye et papirark for å lage en sylinder.

Et beslektet uttrykk er diffeomorfi,

Definisjon 2.2. *En diffeomorfi er en én til én kontinuerlig differensierbar avbildning. Det er en inverterbar funksjon som avbilder en differensierbar mangfoldighet til en annen, slik at både funksjonen og dens invers er glatte. En funksjon er en diffeomorfi hvis:*

- *funksjonen er differensierbar,*
- *den inverse funksjonen er differensierbar,*

En mangfoldighet er et objekt/rom som lokalt rundt ethvert punkt ligner et euklidsk rom, dvs. det er en homeomorfi mellom et åpent sett rundt hvert punkt og et euklidsk rom. Endimensjonale mangfoldigheter er typisk kurver. Todimensjonale mangfoldigheter kalles også flater.



Figur 2.1: *Eksempel på en kule / sfære som er dekket av et atlas som har seks kart (parametriseringer). De seks kartene (parametriseringene) er S_1, S_2, S_3, S_4, S_5 og S_6.*

2.2.1 Global og lokal parametrisering, kart og atlas

Vi ser først på kurver og flater og deres parametriske uttrykk.

1. En parametrisk kurve er en kurve definert av en parametrisk formulering som involverer én parameter, oftest s eller t. Typisk vil det være kurver i $\mathbb{R}^2$ eller $\mathbb{R}^3$ (og ofte betegnet $c(t)$). Mer presist, en parametrisert differensierbar kurve er et differensierbart kart (avbildning)

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

dvs. fra et åpent intervall $I = (a,b)$ av den reelle linjen $\mathbb{R}$ på $\mathbb{R}^n, n = 2,3$.

2. En parametrisk flate er en flate definert av en parametrisk formulering som involverer to parametere, som oftest betegnet (u,v) eller (s,t). Vanligvis vil det være flater i $\mathbb{R}^3$ (og ofte betegnet $s(u,v)$). Mer presist, en parametrisert differensierbar lokal flate er et differensierbart kart

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

dvs. fra et åpent sett $U \subset \mathbb{R}^2$ på $\mathbb{R}^3$.

Kurver og flater er imidlertid ofte definert på lukkede eller halvåpne domener, $\overline{I}$ eller $\overline{U}$, men de kan alltid uttrykkes som en delmengde av et åpent domene som er forutsetningen beskrevet i punkt 1 og 2 i definisjonen over.

Enhver kurve kan beskrives med én parametrisering. Dette vil ikke alltid være det mest praktiske og det èr mulig å bruke flere overlappende intervall (med ikke-tomme overlapper) som domener for flere ulike parametriseringer som til sammen dekker hele kurven.

For flater er det ikke alltid mulig å bruke bare én parametrisering. Dette gjelder spesielt parametriserte, begrensede, kompakte og ikke disjunkte flater av forskjellig topologiske genus g (antall hull/håndtak). Et klassisk eksempel er en kuleoverflate / sfære. Det finnes ingen homeomorfi (definisjon 2.1) mellom en sfære og $\mathbb{R}^2$. Men hvis en kuleoverflate / sfære blir punktert, dvs. ett punkt blir tatt bort, da er resten av sfæren og $\mathbb{R}^2$ homeomorfe. Man kan forestille seg å ta tak i hullet å utvide det og brette det ut slik at overlaten på kulen blir en tallerken.

I figur 2.1 vises seks parametriseringer som tilsammen dekker en sfære (med overlapp),

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

hvor domenet for alle seks kartene, $\{s_i\}_{i=1}^{6}$, er den åpne disken $u^2+v^2<1$.

Hvis en kurve eller en flate er regulær og ikke skjærer seg selv, kan den betraktes som en mangfoldighet. Det vil si at den er et topologisk rom som er lokalt homeomorft til et euklidsk rom[1] av en samling (kalt et atlas) av homeomorfier kalt kart. Sammensetningen av ett kart med den inverse av et nabokart er en funksjon som kalles et overgangskart, og definerer en homeomorfi av en åpen delmengde av et euklidsk rom til en annen åpen delmengde av det euklidske rommet.[2]

Merknad 2.1. *Som senere vil bli vist, brukes både lokale og globale kart i konstruksjonen av blendings-splines (i lokal og global geometri). Dette kan sees for både kurver og tensorproduktflater. Flater som er et lappeteppe (patchwork på engelsk) av trekanter er enda mer i samsvar med definisjonen av mangfoldigheter, der er det ingen global parametrisering. Dette åpner for konsistente konstruksjoner av avgrensede, kompakte og ikke disjunkte flater av alle mulige topologiske genus.*

2.3 Kompakte rom

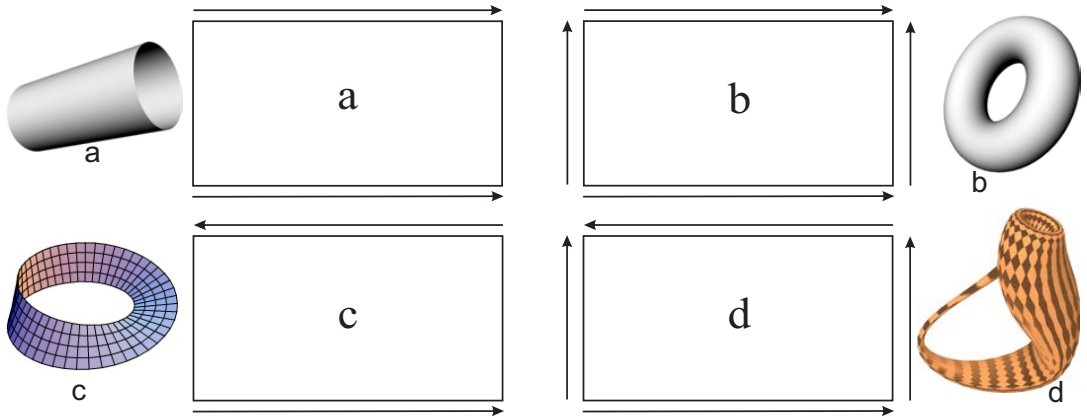
Kurver og flater eller geometriske objekt generelt er som oftest geometrisk begrenset eller “geometrisk endelige", og normalt inkluderer de også deres endepunkt / render.

De vil da normalt være kompakte objekt (mangfoldigheter eller rom). Dette er kurver som inkludert start- og sluttpunkt, rektangler inkludert de fire randkurvene, overflater generelt inkludert deres render, eller en kule, en torus eller lignende.

Formelt kalles et topologisk rom kompakt hvis ethvert sett av åpne overdekninger (et uendelig sett med åpne omegner som sammen dekker hele rommet) har en endelig deldekning

[1] Det vil si at rundt hvert punkt er det en omegn som topologisk sett er det samme som den åpne enhetsballen i $\mathbb{R}^2$ eller respektive $\mathbb{R}^3$, og da faktisk hele $\mathbb{R}^2$ eller $\mathbb{R}^3$ i seg selv.

[2] En nærmere studie av mangfoldigheter finner man i [151] eller [50].



Figur 2.2: *Fire eksempler på 2-dimensjonale kompakte objekter; sylinder, torus, Möbius bånd og Kleins flaske. En sylinder har to render, et Möbius bånd har bare én rand, og en torus og en Kleins flaske har ingen render.*

(kan reduseres til et endelig sett åpne omegner som dekker hele rommet). Ellers kalles det ikke-kompakte.

I figur 2.2 vises fire eksempler på noen spesielle kompakte flater.

a) Rektangelet **a** bøyes slik at to motstående render limes sammen. Resultatet er en sylinder der de to randkurvene er inkludert.

b) Rektangelet **b** bøyes først og to motstående render limes til en sylinder, deretter bøyes sylinderen og de to andre rendene limes sammen. Resultatet er en torus.

c) Rektangelet **c** bøyes, men nå snus de to motstående rendene før de limes sammen. Resultatet er et Möbius-band med bare én randkurve som da er inkludert.

d) Rektangelet **d** bøyes men nå snus de to motstående rendene før de limes sammen, så bøyes det også i den andre retningen og de to andre motstående rendene limes så sammen. Resultatet er Klein flaske.

Det euklidske rommet inkludert det affine rommet som beskrives i neste seksjon er ikke kompakte rom (de inkluderer ikke punkt uendelig langt borte), men det projektive rommet og grassmannrommene er kompakte rom som vi vil se nærmere på i kapittel 2.5.

2.4 Affine rom

Et vektorrom er et sett med elementer vi kaller vektorer. Et vektorrom er lukket under endelig vektoraddisjon og skalar multiplikasjon. Dette betyr at en sum av to vektorer også er en vektor og å skalere en vektor ved skalar multiplikasjon også gir en vektor. Dette gir følgende legale operasjoner,

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

hvor v, v_1 og v_2 er vektorer og k_1 og k_2 er skalarer (reelle tall). Det følger at man kan summere sammen mange vektorer til én hvis antallet vektorer som skal summeres ikke er

uendelig. I forrige avsnitt så vi at vi ikke bare hadde vektorer, men svær ofte har vi punkt. Punkt oppfører seg annerledes enn vektorer men er også koblet til dem i operasjoner.

Vi vil derfor introdusere en ny struktur på euklidske rom.

Affine rom

er et rom av punkt p, med tilhørende vektorer v. Følgende to operasjoner beskriver sammenhengen mellom punkt og vektorer og operasjoner på disse:

$$\begin{aligned} p &= p_1 + k\,v, && \text{sammenhengen mellom punkt og vektorer,} \\ v &= k_1\,v_1 + k_2\,v_2, && \text{operasjoner kun på vektorer,} \end{aligned} \tag{2.1}$$

hvor k-ene er skalarer, dvs. reelle tall.

◇ I tillegg er det enda en lovlig operasjon. Det er en operasjon på kun punkt, og den kalles for en affin kombinasjon og vil bli beskrevet nærmere nedenfor.

Fra den første linjen i (2.1) får vi når vi snur uttrykket

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

Videre, fra en kombinasjon av alle tre uttrykkene ovenfor (2.1) og (2.2) får vi:

Affin kombinasjon

også kalt den barysentriske kombinasjonen. **Dette er den eneste lovlige operasjonen på kun punkt** og er formulert som følger (der p_i er punkt og k_i, $i = 0,...,n$, er skalarer)

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

◇ Navnet indikerer å beregne tyngdepunktet (barysenteret). Det er å summere et sett vektede punkt hvor vektene som brukes summerer opp til 1.

◇ Hvis alle vektene $k_i,\ i = 0,\ldots,n$, er ikke-negative, $k_i \geq 0,\ i = 0,1,...,n$, kaller vi (2.3) for en **konveks** affin kombinasjon.

Den affine kombinasjonen følger fordi (2.3) kan skrives om for å passe til uttrykkene i (2.1),

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

og derfor følger det at

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

Merk fra hermitebasisfunksjonene beskrevet på side 40 at de to basisfunksjonene som blender punktene summerte opp til 1. Legg også merke til at basisfunksjonene for bézierkurver, settet med bernsteinpolynomer som bare blender punkt summerer opp til 1 for alle

grader (lemma 4.2 på side 49). Senere vil vi vise at dette også er tilfellet for B-splines og NURBS.

Grunnen til at dette er så viktig er at å oppfylle en affin kombinasjon er en forutsetning for å kunne bruke affine avbildninger, som er de mest brukte funksjonene i datamaskingrafikk, DAK/DAP etc..

affine avbildninger

er translasjon, skalering, rotasjon, skjær- og parallellprojeksjoner, og er generelt på formen:

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

der p er et punkt og v er en assosiert vektor, begge i et affint rom. Hvis dette er det euklidske $\mathbb{R}^3$, så er A en 3×3 matrise.

Geometrisk sett er affin avbildning (affin transformasjon) i et euklidsk rom en avbildning som bevarer:

1. Kollinearitetsrelasjonen mellom punkt; dvs. tre punkt som ligger på en rett linje fortsetter å være kollineære etter transformasjonen.
2. Forholdet mellom avstander langs en linje; dvs. for tre distinkte kollineære punkt $p_1,\ p_2,\ p_3$, vil forholdet $\frac{|p_2-p_1|}{|p_3-p_2|}$ bli bevart.

En affin avbildning er inverterbar hvis og bare hvis matrisen A er inverterbar. Dette gjelder typisk skalering, rotasjon, skjær og translasjon. Parallell projeksjon er ikke inverterbar.

En skaleringsmatrise er en diagonalmatrise der den inverse er en matrise hvor hvert tall på diagonalen er invertert. Denne matrisen skalerer hver koordinat i en vektor forskjellig.

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

En rotasjonsmatrise er en ortonormal matrise der den transponerte matrisen er den inverse. En rotesjon mot klokken rundt x-aksen med vinkelen α gir

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

En rotesjon mot klokken rundt y-aksen med vinkelen α gir

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

En rotesjon mot klokken rundt z-aksen med vinkelen α gir

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

Hvis vi ønsker å rotere rundt en vilkårlig vektor $\mathbf{r}$, må vi først sørge for at rotasjonsvektoren $\mathbf{r}$ er en enhetsvektor, dvs.

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

Vi lager så matrisene ($\otimes$ er notasjon for ytreprodukt)

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

Hvis rotasjonsvinkelene er α blir den endelige rotasjonsmatrisen

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

og hvor $\mathbf{R}^{-1} = \mathbf{R}^T$

2.5 Projektive rom og grassmannrom

Det projektive rom er rommet av alle 1-dimensjonale underrom til et gitt vektorrom. Det projektive rommet av dimensjon n benevnes $\mathbb{P}^n$. Som oftest er det et rom av reelle tall. En mer presis notasjon er imidlertid $\mathbb{RP}^n$ for reelle projektive rom, og $\mathbb{CP}^n$ for komplekse projektive rom etc. $\mathbb{P}^n$ kan også betraktes som et sett bestående av hele $\mathbb{R}^n$ pluss alle punktene uendelig langt borte (i figur 2.3 tilsvarer dette kantene som er limt sammen).

Figur 2.3 viser en illustrasjon av hva det projektive planet $\mathbb{P}^2$ er. Vi starter med et plan hvor to og to motsatte kanter er limt sammen, den ene kanten limes med kanten på motsatt side men hvor den ene er snudd før de limes sammen. Figuren viser oss avbildningene/homeomorfien fra starten via en tallerken hvor vi ser at de antipodale punktene på randa er det samme punktene, og deretter en halvkule inkludert kantsirkelen som ligger i xy-planet, og videre til settet av alle uendelige rette linjer gjennom origo. Dette følger fordi det er en èn til èn avbildning mellom halvkulen og settet med de rette linjer. Dette gjelder åpenbart for alle punkt på halvkulen bortsett fra de på randa. Men det gjelder også for linjene i xy-planet (på randa) fordi de antipodale punktene er det samme punktet. Følgelig er $\mathbb{P}^2$ settet av alle rette linjer gjennom origo i $\mathbb{R}^3$, dvs. alle 1-dimensjonalt underrom. Fra illustrasjonen i figur 2.3 er det også tydelig at det projektive planet $\mathbb{P}^2$ er kompakt.

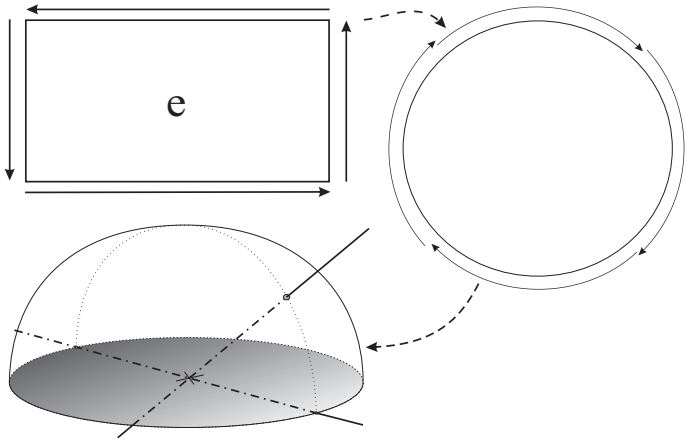
Et punkt i $\mathbb{P}^n$ kan derfor beskrives med $n+1$ koordinater. Men koordinatene kan skaleres med en hvilken som helst skalar forskjellig fra 0 uten at punktet endres. Det følger at $q \in \mathbb{P}^3$ kan uttrykkes ved

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

hvor q er uavhengig av k, dvs. k kan være et hvilket som helst reelt tall som ikke er null.

Det er en entydige overgang mellom det affine rommet $\mathbb{R}^n$ og det projektive rommet $\mathbb{P}^n$. Hele $\mathbb{R}^n$ kan overføres til $\mathbb{P}^n$ med alle strukturene bevart. Overgangen er

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



Figur 2.3: *En illustrasjon av hva et projektivt plan $\mathbb{P}^2$ er. Vi starter med et plan flate med fire kanter. Kantene limes så sammen. Motsatte kanter limes etter at en av dem er snudd. Planet deformeres så til en tallerken hvor de antipodale punktene på randen er det samme punktet. Tallerkenen deformeres videre til en halvkule hvor randa fortsatt er i xy-planet. Vi ser nå at vi har en én til én avbilning (homeomorfi) mellom settet av alle rette linjer gjennom origo i $\mathbb{R}^3$ og det opprinnelige planet med de sammenlimte rendene.*

Affine punkt kan så gjenopprettes fra det projektive rommet med overgangen

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

Generelt kan $\mathbb{P}^n$ sees på som settet av alle linjer gjennom origo i $\mathbb{R}^{n+1}$. I motsetning til de affine/euclidske rommene $\mathbb{R}^n$, er de projektive rommene $\mathbb{P}^n$ kompakte. Det vises tydelig i figur 2.3. Punktene ved “uendelig", som er grenseverdier til $\mathbb{R}^n$, er de “horisontale rette linjene" i $\mathbb{P}^n$, de som skjærer randa. Det viser at det projektive rommet inkluderer punktene ved “uendelig" og at det dermed er kompakt.

Grassmannrom er en generalisering av projektive rom; det er rommet av alle d-dimensjonale underrom til et gitt n-dimensjonalt vektorrom, $0 < d < n$, og benevnes med $Gr(n,d)$. Det følger at det projektive rommet $\mathbb{P}^n = Gr(n,1)$. For eksempel er $Gr(n,2)$ rommet av alle plan gjennom origo i det euklidsk rommet $\mathbb{R}^n$.

Et punkt i et nett, eller en annen organisert struktur, inneholder gjerne flere egensksper, for eksempel en normalvektor, en masse, et lokalt koordinatsystem etc. I resten av boka vil et slikt punkt ofte bli kalt en **verteks**.

2.6 Homogene koordinater

Homogene koordinater, introdusert av August Ferdinand Möbius i 1827 (Der barycentrische Calcül), er et system av koordinater som brukes i projektiv geometri omtrent som kartesiske koordinater brukes i euklidsk geometri. De har den fordelen at koordinatene til alle punktene, inkludert punkt ved uendelig, kan representeres ved bruk av endelige koordinater. Dette fordi det projektive rommet er kompakt og dermed inkluderer punktene ved

uendelig. Formler som involverer homogene koordinater er ofte enklere og mer symmetriske enn deres kartesiske motstykker. Homogene koordinater har en rekke anvendelsesområder, inkludert datagrafikk og 3D geometri generelt, der alle affine transformasjoner og generelt projektive transformasjoner kan representeres av bare èn matrise.

Homogene koordinater har èn koordinat mer enn dimensjonen til rommet. Det følger at hvis den siste koordinaten ikke er 0, så er enhver skalert versjon av elementet det samme elementet, så for å sammenligne valgte vi en felles verdi for den siste koordinaten. Hvis dimensjonen på rommet er 3, så har vi

Punkt $\mathbf{p} = (p_x, p_y, p_z, 1)$, dvs. den siste koordinaten er 1.

Vektor $\mathbf{v} = (v_x, v_y, v_z, 0)$, dvs. den siste koordinaten er 0.

Dette betyr at, i homogene koordinater, er skalering av et punkt umulig fordi hvis et punkt multipliseres med en ikke-null skalar, representerer de resulterende koordinatene det samme punktet. Skalering av punkt er jo også i affine rom en ikke logisk og ikke lovlig operasjon En vektor kan imidlertid skaleres fordi den siste koordinaten fortsatt er 0 etter skaleringen.

Vi ser at å følge dette skjemaet gjør operasjonen i et affint rom komplett og riktige, summering av punkt gir ingen mening, men summering av vektede punkt , der vektene summerer opp til 1 gir et punkt, å legge en vektor til et punkt gir et punkt (siste koordinat er 1), å trekke et punkt fra et annet gir en vektor og så videre.

Spesielt affine avbildninger får en mye enklere formulering fordi vi nå kun får én matrise for alle operasjoner. Husk at en affin avbildning (2.4) er på formen $\mathbf{A}p + \mathbf{v}$. Det betyr at punktet p først bearbeides av matrisen $\mathbf{A}$ som er en kombinasjon av skalering, rotasjon og skjær, for så å flyttes med en vektor $\mathbf{v}$, dvs.

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

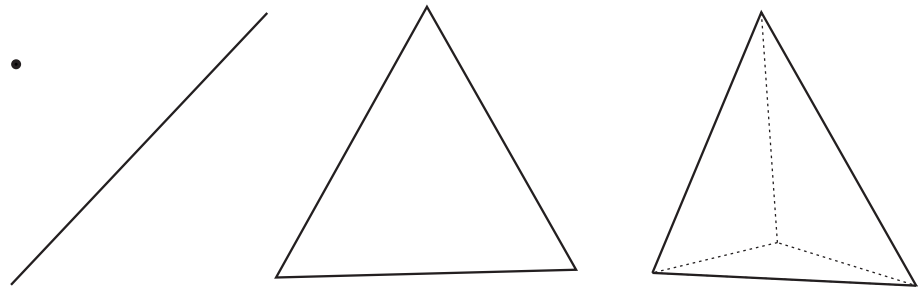
Hvis vi legger dette inn i én “homogen" matrise får vi

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

og hvor den inverse er

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

I denne inverteringen inneholder $\mathbf{A}$ bare rotasjon, som gjør $\mathbf{A}$ ortonormal (ortogonal og det $= 1$) og $\mathbf{A}^{-1} = \mathbf{A}^T$. Hvis også skalering er brukt, og for eksempel kolonnenummer i skaleres med a, da skaleres radnummer i i den inverse matrisen med $\frac{1}{a}$.



Figur 2.4: *Vi ser eksempler på de fire første simpleksene. Fra venstre ser vi et punkt - Δ_1, et linjestykke - Δ_2, en trekant - Δ_3 og en tetraeder - Δ_4.*

2.7 Simplekser

I seksjon 2.2.1 ble kurver definert som 1-dimensjonale objekt, flater som 2-dimensjonale. Dette henger sammen med koblingen/homeomorfien med et euklidsk rom og det manifesterer seg i antall parameter i en parametrisering. Det følger at et punkt er 0-dimensjonalt og et volum 3-dimensjonalt etc.

Et polygon er dermed et 2-domensjonalt objekt, dvs. en 2-dimensjonal polytop. Vi skal se på de enkleste polytopene av gitte dimensjoner, disse kaller simplekser, Δ_n. De 5 første simpleksene er:

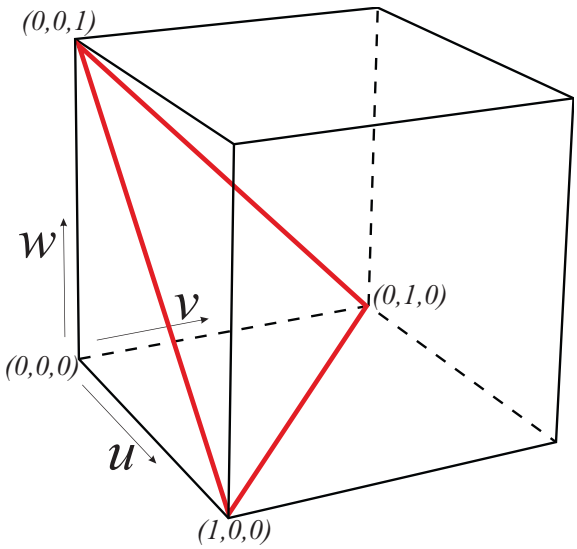
- punkt	er en 1-simpleks - Δ_1 - og har 1 punkt	dimensjonen er 0
- linjestykke	er en 2-simpleks - Δ_2 - og har 2 punkt	dimensjonen er 1
- trekant	er en 3-simpleks - Δ_3 - og har 3 punkt	dimensjonen er 2
- tetraeder	er en 4-simpleks - Δ_4 - og har 4 punkt	dimensjonen er 3
- 5-celle	er en 5-simpleks - Δ_5 - og har 5 punkt	dimensjonen er 4

Legg merke til at randene til en n-simpleks er n stykker (n-1)-simplekser. For eksempel har en trekant 3 kanter/linjestykker. Hvis vi setter inn et punkt inne i en n-simpleks deler vi opp simplekesen i n stykker n-simplekser. En siste observasjon som er lett å se for lave dimensjoner er at for ethvert punkt eksisterer det en direktelinje (kant) mellom punktet og alle de andre punktene. I figur 2.4 er de fire første simpleksene plottet.

2.8 Homogene barysentriske koordinater for simplekser

Barysentriske koordinater ble introdusert av Möbius i 1827, se [65]. Disse kan brukes enten til å uttrykke et punkt inne i en simpleks Δ_n som en konveks kombinasjon av de $n+1$ punktene i simpleksen eller lineært å interpolere data gitt i punktene. Koordinatene tilsvarer masser plassert i verteksene og at punktet selve da er massesenteret (barysenteret). Mye arbeid er gjordt angående oppførsel og anvendelser av barysentriske koordinatene. Blant andre Warren, i [162] og senere publikasjoner.

Først noen fakta om homogene barysentriske koordinater. Legg merke til at i motsetning til barysentriske koordinater generelt er homogene barysentriske koordinater normaliserte



Figur 2.5: *En parametrisk enhetskube beskrevet av koordinatene u, v og w. "Hoveddiagonalen" er markert med røde linjer, og kan sees som en trekant i figuren.*

og dermed unike.[3] Vi tar utgangspunkt i trekanter. Vi starter med en flate $\Omega \subset \mathbb{R}^2$, med et kartesisk koordinatsystem (u,v). Vi lar så dette være et enhetskvadrat. I figur 2.5 har vi utvidet enhetskvadratet til en enhetskube hvor vi har en koordinat w ortogonal til de to andre koordinatene. Hvis vi skjærer denne kuben med et plan som går gjennom de tre hjørnene der en av koordinatene er 1 og de andre to koordinatene er 0, får vi hoveddiagonalen. I figur 2.5 er hoveddiagonalen plottet som en trekant markert med røde linjer. Denne trekanten (Δ_3-simpleksen) er da et domene beskrevet i homogene barysentriske koordinater. Merk at planet som trekanten ligger i har følgende implisitte formel,

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

Samtidig er det naturlig å begrenes domenet til å være i trekanten i figur 2.5, som da betyr følgende begrensning på parameterne,

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

navnet "barysenter" er latinsk for tyngdepunkt. Det refererer til at koordinatene til "massesenteret" som bestemmes via en konveks kombinasjon av punktene til et materialsystem av punkt. Her følger definisjonen av homogene barysentriske koordinater for punkt:

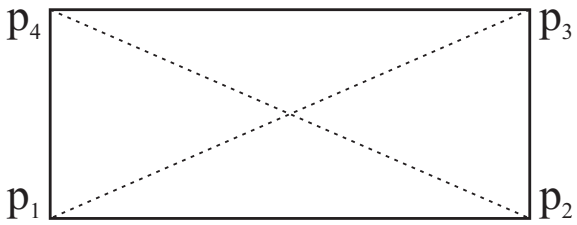
Definisjon 2.3. *Det konvekse settet med punkt som danner hoveddiagonalen til en (n+1)-dimensjonal enhetshyperkube er domenet til en n-dimensjonal simpleks betegnet Δ_n. I homogene barysentriske koordinater er et punkt $\mathbf{p} \in \Delta_n$ definert ved*

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

som oppfylle konveksitetsegenskapen når

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

[3] Merk likhetene mellom homogene barysentriske koordinater vs. homogene koordinater i det projektive rommet. Punkt summerer til 1, vektorer til 0 vs. siste koordinat er 1 eller 0 i det projektive rommet.



Figur 2.6: *Vi ser et rektangel med hjørnene p_1, p_2, p_3 og p_4. De to striplede linjene (diagonalene) er også plottet. Der disse to linjene skjærer hverande forventer man de samme koordinatene uansett hvilken linje vi følger.*

Et punkt $\mathbf{p} \in \Delta_2$ (punkt i trekanten i figur 2.5) er definert som følger;

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

Vi utvider så definisjonen av homogene barysentriske koordinater til å inkludere vektorer. I figur 2.5, setter vi inn et nytt plan, som er parallelt med “hoveddiagonal"-trekanten vi laget tidligere, men som nå går gjennom origo. Dette nye planet kan betraktes som et 2D-vektorrom i barysentriske koordinater. Den implisitte formelen for dette planet er

$$u+v+w = 0$$

Definisjonen av homogene barysentriske koordinater for vektorer er:

Definisjon 2.4. *Vi betegner det n-dimensjonale vektorrommet som går igjennon origo og er paralell med “hoveddiagonalen" til en (n+1)-dimensjonal enhetshyperkube for Υ_n. I homogene barysentriske koordinater er en vektor $\mathbf{d} \in \Upsilon_n$ definert ved*

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

For en trekant (2-dimensjonal simpleks) er en vektor $d \in \Upsilon_2$ definert som følger;

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

Det kan vises at det bare er for simplekser at homogeme baysetriske koordinater er entydige. For dimensjon 2 — flater kan dette vises enkelt. Vi tar utgangspunkt i et firkantet konvekst polygon, for eksempel et rektangel. Vi lager så følgende formel

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

Vi lager så kurvene som knytter de to motsatte hjørnene sammen. Vi lager de med å sette de to andre vektene til 0, dvs. $c_1(t) = s(t,0,1-t,0)$ og $c_2(r) = s(0,r,0,1-r)$. Disse to kurvene vil skjære hverandre ved en bestemt t- og r-verdi. I figur 2.6 er dette illustrert. Dette viser at to forskjellige sett med koordinater gir samme punkt.

Det finnes imidlertid “justeringer" som gir entydige koordinater for konvekse polygon og også 3-dimensjonale polytoper. Michael Floater et al. lanserte middelverdier med barysentriske koordinater i 2D i [67] og [68], og i 3D i [69].

Kapittel 3

Implementering av geometri i C++

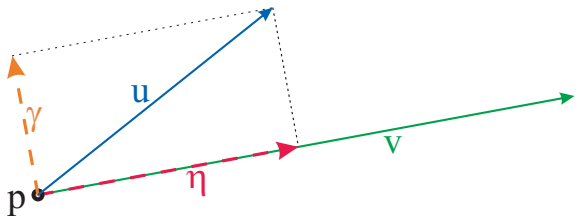
Alle teorier og algoritmer beskrevet i denne boken er implementert og testet, og alle tabeller og figurer, bortsett fra figur 1.1 og figur 6.3, er laget ved hjelp av disse testprogrammene. Implementeringene er gjort i C++ og er i hovedsak basert på et internt men åpent kildekodeprogrammeringsbibliotek kalt GMlib , https://source.coderefinery.org/gmlib.

Noen operasjoner er svært ressurskrevende og kan ta relativt lang tid på en datamaskin. Dette er operasjoner som andre har brukt mye tid på å optimalisere. Dette gjelder for eksempel matrise/vektoroperasjoner med store matriser. Her kan BLAS-kompatible programbibliotek hjelpe. BLAS er en forkortelse for *Basic Linear Algebra Subprograms*. I vedlegg B.1 er det en beskrivelse, samt en liste over BLAS-kompatible programbiblioteker. For å kunne bruke alle tilgjengelige ressurser på datamaskinen er det også en liste over hjelpemidler for “Heterogen databehandling og parallellisering"i vedlegg B.2.

3.1 Matematiske rom for geometrisk programmering

En kombinasjon av affine og projective rom er mest praktisk å bruke i geometrisk programmering. Ved å bruke et euklidsk rom med en affine struktur er vi uavhengige av et koordinatsystem. I et affint rom er origo bare ett punkt blant de andre, og det skiller seg ikke ut fra de andre punktene på noen som helst måte. I hvert punkt kan vi tilordne et vektorrom hvor punktet selv er origo. Dermed kan punkt med tilhørende vektorrom flyttes, roteres og skaleres med en homogen matrise. Dette leder til lokale koordinatsystem. I et geometrisk system i $\mathbb{R}^3$ (og i datagrafikk) har vektorrommet som er knyttet til et punkt tre ortogonale vektorer som til sammen danner et lokalt koordinatsystem lokalisert i punktet.

I det projektive rommet er det naturlig å bruke homogene koordinater. Det betyr å legge til en ekstra koordinat, med verdien 1 for punkt og 0 for vektorer, se side 19. Dette passer også for Affine-rom, og det finnes en enkel mapping mellom disse to rommene, 2.6) og (2.7. I programmering ønsker vi noen ganger å bruke dataene direkte i eksterne rutiner som for eksempel BLAS-kompatible funksjoner. Da kan vi som oftest ikke inkludere den ekstra koordinaten. En løsning er å skille punkt og vektorer med type istedenfor den ekstra koordinaten, noe vi skal se nærmere på.



Figur 3.1: *En projeksjon av en vektor u på en vektor v, begge starter i et felles punkt p. Projeksjonen vises som en rødstriplet vektor η. Restvektoren er da $\gamma = u - \eta$.*

3.2 Homogene koordinater og programmering

Punkt og vektorer er de grunnleggende objektene i affine rom og med det også geometrisk programmering.

For å skille de affine 2D/3D/... vektorene fra std::vektoren som jo er en kontainer, kan vi bruke stor forbokstav på dem. Det er spesielt fordelaktig å bruke template-programmering, fordi man kan bytte mellom enkel og dobbel presisjon, velge dimensjon og også definere typer rekursivt. I template-programmering er det også mulig å lage en egen versjon for spesifikke template-parametere. For eksempel kan Vector⟨T,3⟩ ha en vektorprodukt-operator som produserer en vektor, mens Vector⟨T,2⟩ kan ha den samme operatoren men nå som et kileprodukt som produserer en skalar med fortegn.

Vi kan lage følgende definisjoner basert på: *template⟨typename T, int n⟩*, hvor T eksempelvis kan være double, float eller complex og n er dimensjonen,

Point⟨T,n⟩ - for eksempel Point⟨double,3⟩ - punkt i et 3D-affint rom, dobbel presisjon.

Vector⟨T,n⟩ - for eksempel Vector⟨float,3⟩ - vektor i et 3D-vektorrom, enkel presisjon.

Forholdet mellom Point og Vector må være så konsistent som mulig, og de bør være tett knyttet sammen, men ikke så snevert definert at det skaper problemer. Disse to grunnleggende typene kan eksempelvis så utvides til:

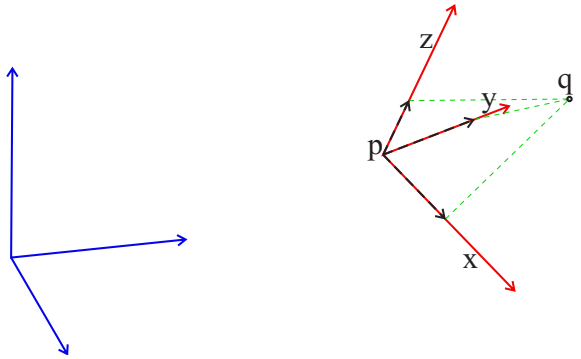
Simplex⟨T,n,m⟩ - Vector⟨Point⟨T,n⟩,m⟩. Legg merke til at Simplex⟨float,3,2⟩ er et linjesegment, Simplex⟨float,3,3⟩ er en triangel og Simplex⟨float,3,4⟩ er et tetraeder.

Matrix⟨T,n,m⟩ - Vector⟨Vector⟨T,n⟩,m⟩. Dette er en $n \times m$ matrise. Hvis $n = m$ da er matrisen en kvadratisk matrise og kan være ikke-singular og i såfall kan en invers matrise lages.

HMatrix⟨T,n⟩ - Matrix⟨T,n+1,n+1⟩ er en homogen matrise, se (2.8)

For den siste matrisen dvs. HMatrix⟨T,n⟩ bør det lages separat multipikasjonsoperatorer for Point⟨T,n⟩ og for Vector⟨T,n⟩ som tar hensyn til den siste koordinaten som henholdvis er 1 eller 0.

En mye brukt operasjon er indreprodukt. Et eksempel er projeksjon av en vektorer på en annen. I figur 3.1 er dette visualisert. Det følger at $\eta = \frac{\langle u,v\rangle}{\langle v,v\rangle} v$. Hvis v er en enhetsvektor (lengde 1), så er formelen $\eta = \langle u,v\rangle v$ og $|\eta| = \langle u,v\rangle$. Dette kan brukes i dekomponering av en vektor. I figur 3.1 er u dekomponert i to ortogonale komponenter med $u = \eta + \gamma$.



Figur 3.2: *Et lokalt koordinatsystem* **p** *med røde koordinatakser. De blå aksene er det globale koordinatsystemet. Punktet* **q** *er projisert ned på de røde vektorene, og de svarte vektorene er dekomponeringen av q med hensyn til* **p** *og dens koordinatakser.*

En homogen matrise kan også representere et lokalt koordinatsystem[1] plassert i et punkt p. Et koordinatsystem i $\mathbb{R}^3$ er bestemt av tre koordinatvektorer, vanligvis betegnet som x-, y- og z-akse, og de er som oftest ortogonale og da er indreproduktet mellom dem alle null. De er alle enhetsvektorer i sitt koordinatsystem, men ikke nødvendigvis i morsystemet.

En homogen matrise vises i (2.8) hvor den er koblet til affine avbildninger, men vi skal her også tolke den som et lokalt koordinatsystem, dvs.

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

hvor $\mathbf{x}$, $\mathbf{y}$ og $\mathbf{z}$ er koordinataksene og $\mathbf{p}$ er punktet som viser plasseringen av det lokale koordinatsystemet. I Figur 3.2 er dette illustrert. Hvis $\mathbf{q}$ er et punkt i det lokale koordinatsystemet, og vi multipliserer med $\mathbf{H}$, da får vi

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

hvor $\widehat{\mathbf{q}}$ er $\mathbf{q}$ i det globale koordinatsystemet (blå aksene i figur 3.2). Den inverse av $\mathbf{H}$, se (2.9), er nå

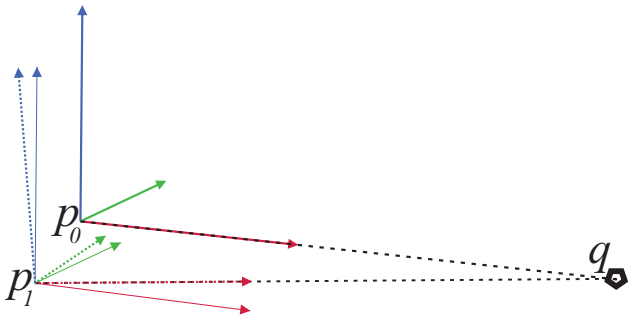
$$\mathbf{H}^{-1} = \begin{bmatrix} x_x & x_y & x_z & -\langle \mathbf{x}, \mathbf{p} \rangle \\ y_x & y_y & y_z & -\langle \mathbf{y}, \mathbf{p} \rangle \\ z_x & z_y & z_z & -\langle \mathbf{z}, \mathbf{p} \rangle \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Hvis vi multipliserer $\widehat{\mathbf{q}}$ med denne matrisen får vi

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

I figur 3.2 vises en illustrasjon av et lokalt koordinatsystemet.

[1] I blending-splines bruker lokale koordinatsystemer for lokale kurver og flater, se kapitlene 8 og 12.



Figur 3.3: *Et objekt med et lokalt koordinatsystem plassert i p_0 og hvor den røde x-aksen peker mot punktet q. Objektet blir så flyttet til p_1 hvor først x-aksen blir endret til å peke mot punktet q og hvor y-aksen justeres til å være normal på x-aksen, se (3.1), og z-aksen justeres til å være normal på de to andre aksene.*

Homogene matriser kan brukes i displayhierarki, dvs. en scenegraf. I et slikt system er scenen referansekoordinatsystemet for alle objekter i scenegrafen og kan brukes til simuleringer som involverer flere objekter. Derfor bør objektene ha to homogene matriser, en lokal og en for referansesystemet.

Det grafiske systemet kan kobles til virtuelle kameraer, som også kan være et objekt i scenegrafen, for eksempel satt inn i en bil i bevegelse. I så fall vil det endelige koordinatsystemet være det som er koblet til kameraet, og som skal brukes i det grafiske systemet. Det betyr at matrisene til hvert objekt er den inverse referansematrisen til kameraets multiplisert med referansematrisene til objektene.

Neste tema er simuleringer, og vi skal se nærmere på to eksempler på bruk av lokale koordinatsystemer/homogene matriser i simuleringer.

Først skal vi se på hvordan vi kan låse retningen til en lokal akse mot et annet objekt. Dette kan for eksempel brukes av et kamera. Siden hvert tids-steg i en simulering er småe, antar vi at endringene også er småe. Ved hvert steg starter vi med å finne avstandsvektoren mellom punktene i de to lokale koordinatsystemene. Så setter vi ønsket akse, for eksempel $\mathbf{x}$ lik denne, og normaliserer den. Deretter korigerer vi $\mathbf{y}$ med

$$\mathbf{y} = \mathbf{y} - \langle \mathbf{y}, \mathbf{x} \rangle \mathbf{x}, \tag{3.1}$$

og normaliser deretter $\mathbf{y}$. Til slutt setter vi $\mathbf{z} = \mathbf{x} \wedge \mathbf{y}$. Nå er retningen låst selv om begge objektene beveger seg. Dette er illustrert i Figur 3.3.

Det andre eksemplet er også nyttig for et kamera, men kan også brukes av mange andre typer objekt. Hvis du skal “rotere et objekt uten å gjøre det", kan du rotere kameraet rundt objektet, eller rundt et punkt i rommet. Dette kan vi gjøre ved å låse kameraets retning mot objektet/punktet, og ved hvert trinn flytte kameraet i planet som er ortogonalt på aksen som peker fremover, dvs. $\mathbf{p} = \mathbf{p} + dy\,\mathbf{y} + dz\,\mathbf{z}$. Her er $\mathbf{x}$ retningen mot objektet og $\mathbf{y}$ og $\mathbf{z}$ spenner ut planet. Husk at $\mathbf{x}$, $\mathbf{y}$ og $\mathbf{z}$ er de tre første kolonnene i matrisen til kameraet. dy og dz er skalarer avhengig av tidstrinnet dt og hastigheten og retningen til musen i brukergrensesnittet. Etter at en låse- og flytteoperasjon er utført, flytter vi posisjonen $\mathbf{p}$

i den homogene matrisen i retning mot låsepunktet slik at avstanden til objektet/punktet opprettholdes.

3.3 Verktøy for interaktiv design

Noen geometriske objekt, og da spesielt kurver og flater, kan endre form. For å gjøre dette interaktivt i grafikkmodus trenger vi noen verktøy. Disse geometriske objektene er enten kontrollpunkt og/eller vektorer eller homogene matriser, som kan flyttes, roteres, skaleres etc.. Vektorer kan for eksempel endres ved å flytte endepunktet.

Et verktøy for å flytte punkt i grafisk modus er en **selector**. Det er et objekt som har en referanse til punktet som skal redigeres og som kan rapportere til det overordnede objektet til punktet om å oppdatere seg selv. En selector kan velges og flyttes ved hjelp av en musepeker. En selector kan typisk vises som en liten ball. Husk at flytting av et objekt kan gjøres basert på det lokale koordinatsystemet til kameraet og deretter endre posisjonen til punktet ved å følge translasjonsvektoren først til referansepunktet (scenen) og deretter ut til objektet.

Et annet verktøy er en **plassholder**. En kurve eller flate eller et annet geometrisk objekt kan vises på skjermen som en plassholder. Dette er da et objekt som både skal kunne flyttes, roteres og skaleres og som typisk vises som en kube på skjermen. Flytting endrer punktet **p** i matrisen til objektet mens rotasjon og skalering vil endre koordinataksene **x**, **y** og **z** i matrisen til objektet. En plassholder brukes til å “redigere" blendingsplines, som vil bli beskrevet i kapitelene 8, 12, 13 og 14.

3.4 Implementering av kurver og flater

I avsnitt 2.3 om kompakte objekt kommer det frem at kurver og flater for datagrafikk bør være kompakte objekt, som enten inkluderer deres endepunkt/kanter eller er en sirkellignende kurve, sfærelignende flate, en toruslignende flate, etc. Dette gjelder da ikke enkeltlapper i et lappeteppe, men komplette kurver og flater. En dataskjerm er en matrise av piksler, og for å plotte/vise en kurve kan vi beregne hver piksel for å se om det er et objekt der som vises som en kurve med en gitt farge. Dette gjelder typisk for “ray-traceing". En annen metode som så langt har vært mer vanlig er å dele en kurve i små linjestykker og deretter vise hvert linjestykke. Vi kaller prosessen med å dele for tessellering. For flater er prosessen å dele opp flaten i små trekanter. Vi kaller hjørnene på trekantene for vertekser. For flater trenger vi i tillegg til posisjonen til verteksene også overflatenormalene i verteksene. Derfor må alle kurver ha funksjoner for å beregne posisjon, og alle flater har funksjoner for å beregne posisjon og flatenormaler i verteksene.

Det er mange måter å implementere en geometristruktur på, enten abstrakt basert på dimensjon, egenskaper og topologi, eller en enkel arvestruktur hvor man også må skille mellom parametrisk og ikke-parametriske typer. Vi skal se på en enkel arvestruktur for parametriske kurver og flater. For kurver må vi sørge for at det finnes funksjoner som definerer parameterdomenet, vi trenger en funksjon for å beregne posisjon og deriverte i en

gitt parameterverdi, og vi må vite om kurven er lukket eller åpen. Alle disse funksjonene kan defineres i en basisklasse hvor disse funksjonene er 0-funksjoner, noe som betyr at alle avledede klasser må ha sine egne versjoner av disse funksjonene. For flater må vi også sørge for at det finnes to funksjoner som definerer parameterdomenet, vi trenger en funksjon for å beregne posisjonen og de partiellderiverte for gitte parameterverdier, og vi må vite om flaten er lukket eller åpen i de to retningene. Som for kurver, må alle disse funksjonene defineres i en basisklasse som en 0-funksjon, noe som betyr at alle avledede klasser må ha sine egne versjoner av disse funksjonene selv. Baseklassene kan imidlertid ta seg av tesselleringen og kontakten med grafikksystemet.

Merknad 3.1. *En funksjon for å beregne posisjon og de (partiell)deriverte for gitte parameterverdier vil bli kalt en* **evaluator** *i resten av boken. Dette gjelder for kurver, flater, volum og andre parametriserte objekt.*

Del I

Kurver

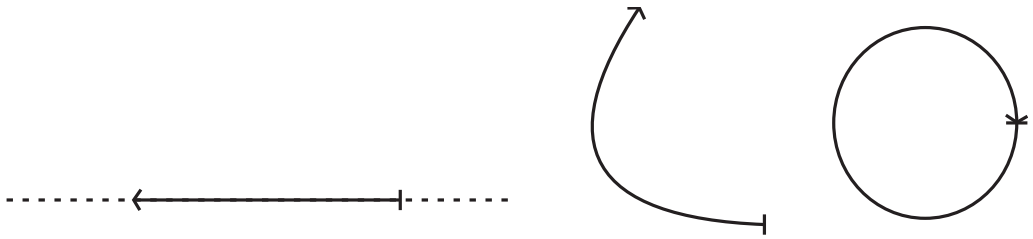
Kapittel 4

Parametriske Kurver

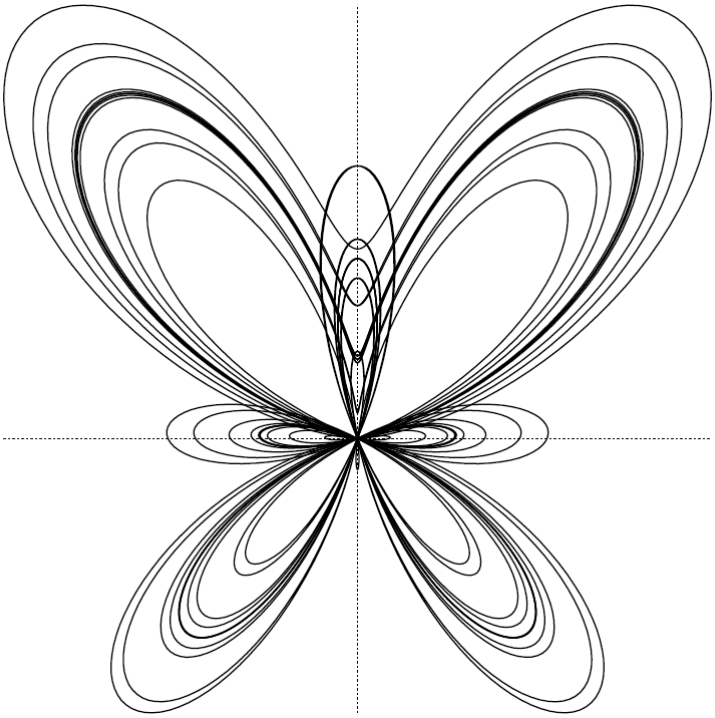
Se for deg rommet av reelle tall $\mathbb{R}$ som en uendelig rett linje. For å lage en kurve trekker vi ut et segment av denne uendelige rette linjen. Vi deformerer segmentet ved å strekke, presse og/eller bøye den på forskjellige måter. Vi putter det så inn i et rom som for eksempel kan være et plan (euklidsk – $\mathbb{R}^2$) eller rommet (euklidsk – $\mathbb{R}^3$). Til slutt plasserer vi resultatet i ønsket posisjon med ønsket orientering og skalering. Resultatet er en parametrisk kurve. Legg merk til at klipping og liming **ikke** er et alternativ her. I figur 4.1 er dette konseptet forsøkt illustrert visuelt. Vi kategoriserer en kurve som et 1-dimensjonalt objekt (én parameter). Hvis vi begrenser kurven slik at den ikke degenererer til et punkt eller skjærer seg selv, kan en kurve kalles en 1-dimensjonal mangfoldighet (se [151, 50]).

Det foregående er et forsøk på å beskrive konseptet - parametrisk kurve. Vi kan også forestille oss en parametrisk kurve som en bane til et objekt i bevegelse og hvor banen også inneholder en tidsangivelse for når objektet er der. For å gjøre dette på en mer matematisk måte generaliserer vi først resultatet til ikke bare å være i $\mathbb{R}^2$ eller $\mathbb{R}^3$, men mer generelt i $\mathbb{R}^n$, $n > 0$. En mer formell definisjon er:

Definisjon 4.1. *En parametrisert differensierbar kurve er en differensierbar avbildning $\alpha : I \longrightarrow \mathbb{R}^n$ av et åpent intervall $I = (a,b)$ av den reelle linjen $\mathbb{R}$ til $\mathbb{R}^n$. (Merk at et halvåpent eller et lukket intervall bare er en avgrensning av et åpent intervall.)*



Figur 4.1: *På venstre side er en del av de reelle tallene illustrert som en stiplet linje, det halvåpne intervallet $I = (0, 2\pi]$ er markert som et heltrukket linjestykke. I midten av figuren er intervallet bøyd og til slutt, til høyre, er den lukket til en sirkel. Hvis sirkelen er en enhetssirkel med radius $r = 1$, som i formel (4.1), så er lengden på intervallet og kurven den samme, der er ingen strekking, mens formel (4.2) strekker til dobbel lengde.*



Figur 4.2: *Et plott av en "sommerfuglkurve" laget etter formel (4.3).*

Det første eksemplet vi skal se på er en sirkel i planet, hvor parameterintervallet $I = (0, 2\pi]$ er halvåpen. I formelen går derfor t fra 0 (ikke inkludert) til 2π (inkludert). En gitt t-verdi resulterer i en vektor med en x og en y komponent, dvs. en vektor i det euklidske rommet $\mathbb{R}^2$. Derfor kaller vi funksjonen under for en vektorfunksjon med to komponenter,

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

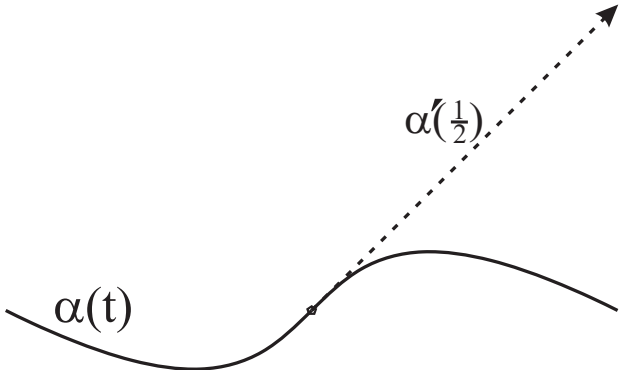
Sentrum av sirkelen definert i (4.1) er i origo og radius er 1. Hvis vi ønsker å endre radius av sirkelen til $r = 2$, og flytte sentrum til en annen posisjon, for eksempel til et punkt med koordinatene $x = 1$, $y = 2$ får vi;

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

Et mer sofistikert kurveeksempel er en "sommerfuglkurve", laget av Temple H. Fay i 1989 og publisert i [66]. Kurven kan sees i figur 4.2 og formelen er som følger:

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

Denne kurven, figur 4.2, er 12 etterfølgende sirkler som er deformert av syklisk endring (og kollaps) av radius som det fremkommer av formelen i (4.3). Parametriske kurver kan lages av alle typer funksjoner, trigonometriske-, logaritmiske-, eksponentialfunksjoner og



Figur 4.3: *Et plott av den polynombaserte kurven fra formel (4.4). Den 1.-deriverte dvs. tangentvektoren $\alpha'\left(\frac{1}{2}\right)$ er også plottet.*

selvfølgelig polynomer som er blant de mest populære valgene. Det neste eksemplet er derfor en 2D-vektoriell parametrisk kurve som er basert på polynomer:

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

Denne kurven (4.4) er plottet i figur 4.3. Av formelen ser vi at kurven ligger i $\mathbb{R}^2$ og starter i origo. Merk at for en gitt t-verdi er $\alpha(t) = (x(t), y(t))$ et punkt på kurven. Variabelen t kalles parameteren til kurven og parameterintervallet $I = [0,1]$ kalles domenet til kurven. Bildet $\alpha(I) \subset \mathbb{R}^2$ kalles sporet til α. Og ordet differensierbar betyr at deriverte av $x(t)$ og $y(t)$ kan beregnes overalt på kurven, dvs. på hele domenet.

Vektor rom (Lineært rom)

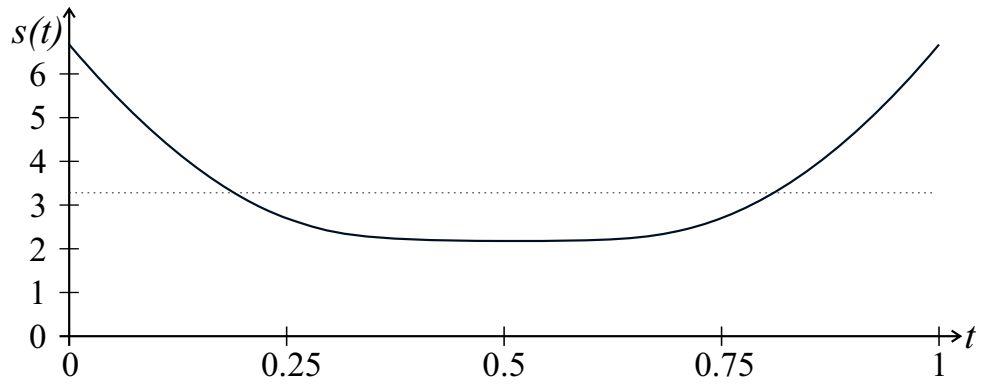
I kapittel 2.1 omtalte vi vektorrom med utgangspunkt i Euklidske rom. Disse er geometrisk inspirert og vi forestiller oss vektorene som piler med utgangspunkt i origo. Vektorrom kan imidlertid generaliseres, og objektene trenger ikke være klassiske vektorer. Kravet er at de oppfører seg på samme måte og at de kan:

- Adderes, hvor «addering» er definert av et sett av aksiomer;
  kommutativitet $u+v = v+u$
  assosiativite $u+(v+w) = (u+v)+w$
  nullelement 0 slik at $u+0 = u$
  invers $(-u)$ slik at $u+(-u) = 0$
- Skalarmultipliseres, hvor «skalarmultiplikasjon» er definert av aksiomene;
  distributivitet $\alpha(u+v) = \alpha u + \alpha v$
  multiplikasjon $(\alpha+\beta)u = \alpha u + \beta v$
  enhetselement 1 slik at $1u = u$

Husk at resultatet av begge operasjonene alltid skal være inneholdt i vektorrommet.

Vektorrom kan utvides til indreproduktrom med å legge til operasjonen indreprodukt, en avbildning $\langle u,v\rangle : V \times V \to \mathbb{R}$ med følgende egenskaper; $\langle u,v\rangle = \overline{\langle v,u\rangle}$, $\langle \alpha u, v\rangle = \alpha\langle u,v\rangle$, $\langle u+w,v\rangle = \langle u,v\rangle + \langle w,v\rangle$ og $\langle u,u\rangle > 0$ for alle $u \neq 0$.

Senere skal vi se at “funksjonsrom" er vektorrom og indreproduktrom.



Figur 4.4: *Figuren viser hastigheten $s(t) = |\alpha'(t)|$ til kurven fra formel (4.4).*

4.1 Derivasjon

En parametrisk kurve $\alpha(t)$ kan betraktes som banen til et objekt som beveger seg med tiden t. Den 1.-deriverte kan derfor naturlig kobles til hastighet.

- Vi angir den 1.-deriverte[1] som, $\frac{d\alpha}{dt}(t)$ eller som $\alpha'(t) = (x'(t), y'(t))$.
- $\alpha'(t)$ kalles tangentvektoren eller hastighetsvektoren til kurven i punktet $\alpha(t)$.
- Lengden på tangent/hastighetsvektoren $|\alpha'(t)|$ kalles hastigheten i punktet $\alpha(t)$.

I figur 4.3 er tangentvektoren i punktet $\alpha(\frac{1}{2})$ plottet. Vi kan tydelig se at den er tangentiell til kurven. Siden parameterdomenet er 1, dvs. fra 0 til 1, vil gjennomsnittslengden av tangenten/hastighetsvektoren være lik kurvelengden, jmf. fotnoten nedenfor. Som eksempel bruker vi $\alpha(t)$ definert i (4.4). Hastigheten til kurven $s(t) = |\alpha'(t)|$, $t \in [0,1]$ er plottet i figur 4.4. Vi kan beregne kurvelengden fra "gjennomsnittshastigheten" ganger tiden som er brukt. I figur 4.4 er dette det samme som arealet under funksjonen $s(t)$ over domenet. Generelt får vi følgende integral for å beregne kurvelengden over intervallet $I = [a,b]$:

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

Hvis vi bruker kurven definert i (4.4) får vi hastighetsfunksjonen

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

Og kurve/buelengden blir da

$$l(\alpha)_{0,1} = \int_0^1 3\sqrt{72t^4 - 144t^3 + 108t^2 - 36t + 5}\, dt \approx 3,28.$$

[1] I avsnitt 9.1.1 blir differensialoperatoren d introdusert. Dette er en generalisering av derivering til også å inkludere multivariable funksjoner. Med en slik generell beskrivelse, kan den 1.-deriverte av en kurve $c(t)$ uttrykkes med $d(c)_t(1)$, som betyr at vi i et punkt $c(t)$ har en vektor som beskriver hvor vi vil være etter 1 "tidsenhet" hvis vi beveger oss uten å endre retning og hastighet.

4.1.1 Regulære kurver - buelengdeparametrisering

En regulær kurve er en kurve, $\alpha(t),\ t \in I$, der tangent/hastighetsvektoren ikke kollapser, dvs. $s(t) = |\alpha'(t)| > 0$ for alle $t \in I$.

For å studere noen viktige egenskaper skal vi se nærmere på en spesiell type kurver, nemlig kurver som er buelengdeparametrisert.

Definisjon 4.2. *En parametrisert kurve $\zeta(t)$ kalles en buelengdeparametrisert kurve hvis*

$$|\zeta'| = 1, \quad \textit{dvs. hastigheten er 1 over hele kurven.}$$

Det følger at,

- ✓ $\langle \zeta'', \zeta' \rangle = 0$, den 2.-deriverte er alltid ortogonal til den 1.-deriverte fordi det bare er retningen til den 1.-deriverte som endres, ikke lengden (hastigheten).
- ✓ $\kappa = |\zeta''|$, Vi benevner krumningen for κ. Krumningen til en kurve er definert til å være lengden av den 2.-deriverte til en buelengdeparametrisert kurve.

For buelengdeparametriserte kurver i $\mathbb{R}^3$ har vi også følgende egenskaper,

- ✓ Frenet-frame (også kalt TNB-frame) er en ramme (matrise), dvs. tre enhetsvektorer, T, N, og B som er ortogonale til hverandre i et høyrehåndssystem. $T = \zeta',\ N = \frac{1}{\kappa}\zeta''$ og $B = T \wedge N$. Eksistensen av en TNB-frame i et punkt krever at $\kappa \neq 0$.
- ✓ τ er torsjonen til en kurve. Den måler rotasjonshastigheten til den binormale vektoren N ved det gitte punktet, og er gitt av $\tau = -\langle N, B' \rangle$.

4.1.2 Reparametrisering

Reparametrisering har ingen effekt på formen til en kurve (kurvebanen). Den endrer hastighet og parameterintervall (domenet) til en kurve, men den påvirker ikke de egenskapene til en kurve som vi kaller iboende egenskaper, dvs. form, kurvelengde, krumning, torsjon osv. Vi skal nå se nærmere på reparametrisering

Gitt en kurve $c(t),\ t \in I \subset \mathbb{R}$. En kurve

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

er en reparametrisering av $c(t)$ hvis

- $\omega(t)$ er differensierbar for $t \in I$,
- og det finnes en invers ω^{-1} som også er differensierbar for $t \in I$.

Det følger at $\omega(t)$ må være en strengt monoton funksjon.

Den 1.-deriverte til kurven $\rho(t)$ er tangenten/hastighetsvektoren til kurven, og lengden til denne vektoren påvirkes direkte av en reparametrisering. Den 1.-deriverte er

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

Det følger da at hastigheten er

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

4.1.3 Krumning

Vi kaller den 2.-deriverte til en kurve $c(t)$ for $c''(t)$. Den 2.-deriverte beskriver den lineære endringen av den 1.-deriverte. Den 2.-deriverte kan dekomponeres til to vektorer som er ortogonale til hverandre, endring av hastigheten og endring av retningen.

Det er åpenbart at en retningsendring påvirkes av hastigheten. Alle som har kjørt en bil har opplevd dette fysisk. Krumning er imidlertid en iboende egenskap som er uavhengig av parametrisering og hastighet. For å finne krumningen må vi derfor reparametrisere en kurve slik at hastigheten alltid er 1, det vi kaller buelengdeparametrisering. Så, gitt en kurve

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

Det følger av (4.6) at $\zeta(t)$ er buelengdeparametrisert hvis $\omega'(t) = \frac{1}{|c'|}$.

For å forenkle lar vi være å bruke parameternotasjonen i uttrykkene i det følgende. Vi bruker så kjerneregelen og produktregelen for derivering og får

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

og

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

For å beregne krumningen og for å unngå å måtte beregne ω'' bruker vi vektorproduktet for å finne $|\zeta''|$. Dette kan vi gjøre fordi vektorproduktet av to parallelle vektorer er 0. Videre vet vi at krumningen, κ, er lik lengden av den 2.-deriverte $|\zeta''|$, og at ζ'' stå normalt på den 1.-deriverte ζ', og at $|\zeta'| = 1$. Denne kunnskapen bruker vi så i beregningen. For enkelhets skyld ser vi først på kurver i $\mathbb{R}^3$. Vi får dermed,

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

som gir

Krumning til kurver i $\mathbb{R}^3$ og $\mathbb{R}^2$

$$\kappa = \frac{|c' \wedge c''|}{|c'|^3}, \quad c \hookrightarrow \mathbb{R}^3. \tag{4.8}$$

der $\wedge$ angir 3D-vektorproduktet (kryssproduktet).
I R^2 kan vi bruke samme formel, men her bruker vi kileproduktet som gir en skalar, $a \wedge b = a_x b_y - a_y b_x$. Dette gir oss også fortegn på krumninger,

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

som gir en positiv verdi på krumning til venstre og negativ verdi på krumning til høyre.
Koblet til krumning er krumningsradiusen som er: $r = \frac{1}{\kappa}$.

4.2 Funksjonsrom og basisfunksjoner

Formel (4.4) til kurven i figur 4.3 kan omskrives på følgende måte

$$\alpha(t) = \begin{pmatrix} 6t - 9t^2 + 6t^3 \\ -3t + 9t^2 - 6t^3 \end{pmatrix} = \begin{pmatrix} 6 \\ -3 \end{pmatrix} t + \begin{pmatrix} -9 \\ 9 \end{pmatrix} t^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix} t^3, \quad 0 \le t \le 1. \tag{4.9}$$

Formelen er nå på den generelle formen

$$\alpha(t) = a_0\, 1 + a_1\, t + a_2\, t^2 + a_3\, t^3, \quad 0 \le t \le 1, \tag{4.10}$$

hvor $a_0 = (0,0),\ a_1 = (6,-3),\ a_2 = (-9,9)$ og $a_3 = (6,-6)$ er koeffisienter, og polynomene på monomial form er basisfunksjoner, dvs. en potensbasis $\{t^i\}_{i=0}^3$.

Parametriske polynombaserte kurver

Generelt vil en parametrisk polynombasert kurve av grad d med potensbasis være på følgende form

$$\alpha(t) = \sum_{i=0}^{d} a_i\, t^i, \quad t \in I \subset \mathbb{R} \tag{4.11}$$

hvor $a_i,\ i = 0,1,...,d$ er elementer i rommet der kurven er lagt inn i.

Potensfunksjonene i (4.11) kan betraktes som basisfunksjoner til et funksjonsrom og er på mange måter lik basisvektorer i et vektorrom, blant annet er de er lineært uavhengige fordi vi ikke kan få en av funksjonene fra en lineær kombinasjon av de andre. Vi kan dermed behandle settet med alle polynombaserte funksjoner med grad opptil d som et vektorrom hvor $1, t, t^2, ..., t^d$ er basisvektorer, som vi ser i (4.11). Dette fører til et mere generelt uttrykk for kurveformler;

Parametriske kurver generelt

$$\alpha(t) = \sum_{i=1}^{k} c_i\, b_i(t), \quad t \in I \subset \mathbb{R} \tag{4.12}$$

hvor $c_i,\ i = 1,...,k$ er vektorer/punkt i rommet der kurven er plasert, og $b_i(t),\ i = 1,...,k$ er et sett med lineært uavhengige funksjoner som spenner ut et gitt endeligdimensjonalt funksjonsrom.

Merk at dette er endeligdimensjonale vektorrom, og kurvene har derfor store begrensninger i formings-mulighetene. For polynomer vil ikke en andregradskurve ha vendepunkt, en tredjegradskurve har bare ett vendepunkt og så videre. Tenk deg så at du tegner en kurve på frihånd. For å finne en formel som **eksakt** gjengir denne kurven vil det kreve et polynom av grad ∞ og dermed et uendelig-dimensjonalt vektorrom. Uendelig-dimensjonale vektorrom kan være nyttige teoretisk, men de er absolutt ikke mulige å implementere for kurver og flater i applikasjoner på en datamaskin. Det vi kan gjøre er å finne en kurve av endelig dimensjon som ligner mest mulig på det vi ønsker, dvs. å approksimere.

Funksjonsrom

er et veldig nyttig konsept for når vi skal konstruere og analysere kurver for geometrisk modellering og design (husk at et funksjonsrom er et vektorrom).

Definisjon 4.3. *Et Funksjonsrom er et sett funksjoner som avbilder en mengde X til en mengde Y. Det er et topologisk vektorrom hvor "vektorene" er funksjoner.*

Et typisk eksempel er settet av alle polynomer av grad 3, som avbilder intervallet $I = [0,1]$ til $\mathbb{R}^n$, $n > 0$ men endelig. Vi kaller dette funksjonsrommet for $\mathscr{P}_3(I)$

I de følgende seksjonene vil vi vise at for polynom er det mange sett med basisfunksjoner som spenner ut et gitt funksjonsrom, og som vi dermed kan velge mellom, alle med spesielle og nyttige egenskaper.

Uendelig-dimensjonale funksjonsrom er imidlertid nyttige teoretisk.

Definisjon 4.4. *Et uendelig-dimensjonale funksjonsrom er $\mathscr{F}(I)$, dvs. settet av alle reelle kontinuerlige funksjoner definert på et gitt intervall I, og hvor $\mathscr{F}^{(n)}(I)$ er settet av alle funksjoner $\in \mathscr{F}(I)$ med n kontinuerlige deriverte.*

Eksempel på et uendelig dimensjonalt funksjonsrom er Hilbert-rommet L^2, settet med alle funksjoner $f : \mathbb{R} \to \mathbb{R}$ slik at integralet til $|f(x)|^2$ over hele den reelle linjen er endelig. I dette tilfellet er indreproduktet

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

4.3 Hermitekurver

Vi skal her bruke eksemplet som først er formulert i (4.4) og dernest omformulert i (4.9).

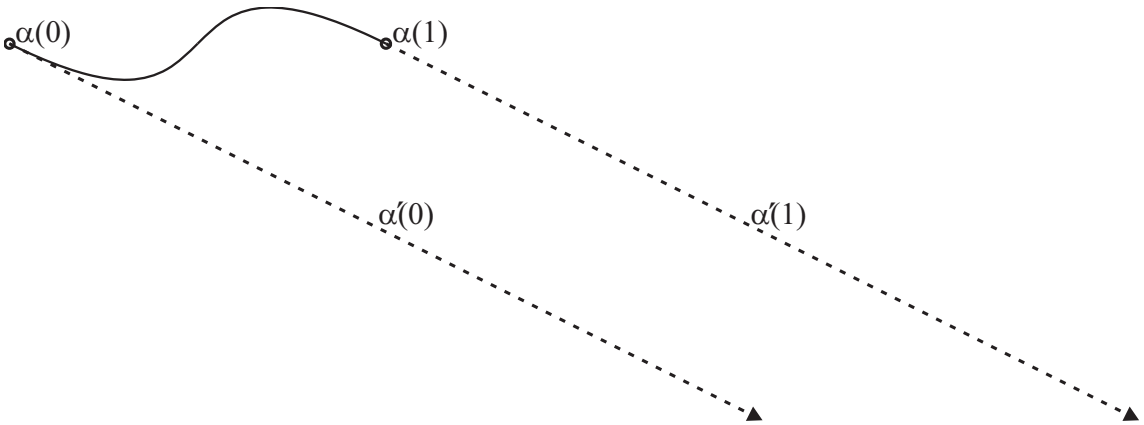
$$\alpha(t) = \begin{pmatrix} 6 \\ -3 \end{pmatrix} t + \begin{pmatrix} -9 \\ 9 \end{pmatrix} t^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix} t^3 \quad 0 \le t \le 1.$$

Den 1.-deriverte er

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

Vi beregner så begge uttrykkene ved startparameterverdien $t = 0$ og sluttparameterverdien $t = 1$,

$$\begin{aligned}
\alpha(0) &= \begin{pmatrix} 6 \\ -3 \end{pmatrix} 0 + \begin{pmatrix} -9 \\ 9 \end{pmatrix} 0^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix} 0^3 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \\
\alpha(1) &= \begin{pmatrix} 6 \\ -3 \end{pmatrix} 1 + \begin{pmatrix} -9 \\ 9 \end{pmatrix} 1^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix} 1^3 = \begin{pmatrix} 3 \\ 0 \end{pmatrix}, \\
\alpha'(0) &= \begin{pmatrix} 6 \\ -3 \end{pmatrix} + 2\begin{pmatrix} -9 \\ 9 \end{pmatrix} 0 + 3\begin{pmatrix} 6 \\ -6 \end{pmatrix} 0^2 = \begin{pmatrix} 6 \\ -3 \end{pmatrix}, \\
\alpha'(1) &= \begin{pmatrix} 6 \\ -3 \end{pmatrix} + 2\begin{pmatrix} -9 \\ 9 \end{pmatrix} 1 + 3\begin{pmatrix} 6 \\ -6 \end{pmatrix} 1^2 = \begin{pmatrix} 6 \\ -3 \end{pmatrix}.
\end{aligned} \tag{4.13}$$



Figur 4.5: *Et plott av kurven fra formel (4.4), men her sammen med de fire hermitekoeffisientene, start og sluttpunktet og deres respektive 1.-deriverte.*

Verdiene fra 4.13 ser vi i figur 4.5, der vi ser et nytt plott av $\alpha(t)$ som tidligere var plottet i figur 4.3. Nå er imidlertid start- og sluttpunktet på kurven, $\alpha(0)$ og $\alpha(1)$ markert med sirkler, og de respektive tangent-/hastighetsvektorene ved starten og slutten av kurven, $\alpha'(0)$ og $\alpha'(1)$ er plottet som stiplede piler.

Vi går nå tilbake og ser på det generelle uttrykket for polynombaserte kurver av grad 3, dvs. formelen i (4.10) i seksjonen 4.2, og dens 1.-deriverte,

$$\begin{aligned} \alpha(t) &= a_0 + a_1\, t + a_2\, t^2 + a_3\, t^3, \\ \alpha'(t) &= a_1 + 2a_2\, t + 3a_3\, t^2. \end{aligned} \tag{4.14}$$

Fra formel (4.14) kan vi beregne startpunktet, $\alpha(0)$, endepunktet, $\alpha(1)$, samt den 1.deriverte (tangens/hastighetsvektoren) ved startpunktet, $\alpha'(0)$, og den 1.-deriverte ved endepunktet, $\alpha'(1)$;

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

Vi omformulerer så uttrykket til matriseform og får

$$\begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}. \tag{4.15}$$

Vi snur så formuleringen i (4.15) og inverterer matrisen og får

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}. \tag{4.16}$$

Vi omformulerer så (4.14), dvs. kurven $\alpha(t)$ til vektornotasjon og indreprodukt,

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

Det neste steget er å erstatte koeffisientene $a_i,\ i = 0,1,2,3$ på høyre side i formelen ovenfor med uttrykket på høyre side i (4.16),

$$\alpha(t) = \begin{pmatrix} 1, & t, & t^2, & t^3 \end{pmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}. \tag{4.17}$$

Til slutt multipliserer vi potensbasisen med matrisen og får,

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

I vektoren på venstre side har vi nå fått et nytt sett med basisfunksjoner for en 3.-grads polynombasert kurve, som er:

Hermitekurver, 3.-grad

er polynom-baserte kurver på følgende form:

$$\alpha(t) = \alpha(0)\, H_1(t) + \alpha(1)\, H_2(t) + \alpha'(0)\, H_3(t) + \alpha'(1)\, H_4(t), \quad t \in [0,1], \tag{4.18}$$

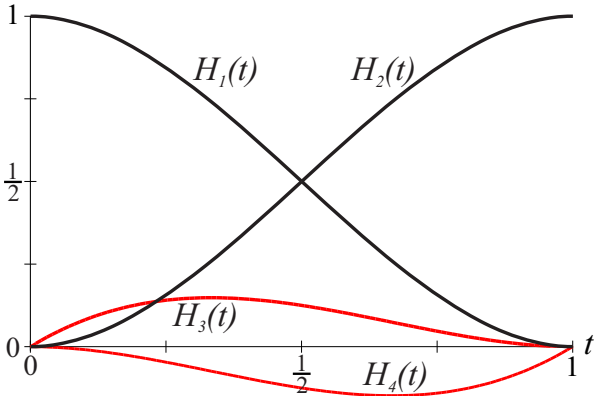
hvor koeffisientene er startverdien $\alpha(0)$ og sluttverdien $\alpha(1)$, og den 1.-deriverte ved start $\alpha'(0)$ og slutt $\alpha'(1)$. De fire 3.-grads hermitebasisfunksjonene er:

$$\begin{aligned} H_1(t) &= 1-3t^2+2t^3 &&= (2t+1)(1-t)^2, \\ H_2(t) &= 3t^2-2t^3 &&= (3-2t)t^2, \\ H_3(t) &= t-2t^2+t^3 &&= t(t-1)^2, \\ H_4(t) &= -t^2+t^3 &&= t^2(t-1). \end{aligned} \tag{4.19}$$

Det faktum at matrisen i (4.15) er invertert i (4.16) og dermed er inverterbar, beviser at settet med hermitebasisfunksjoner er lineært uavhengige og dermed kan være et basissett for 3.-grads polynomkurver. De fire hermitebasisfunksjonene er plottet i figur 4.6.

Det spesielle med hermitekurver er at de fire koeffisientene er startpunktet og dens relaterte hastighetsvektor (første deriverte) og endepunktet med dens hastighetsvektor. Dette er svært praktisk å bruke for å lage en kurve. Dette forklarer også navnet[2]; hermiteinterpolasjon er å interpolere et punkt og deriverte i samme punktet. Dette kalles også oskulatorisk interpolasjon (oskulatorisk betyr kyssing på latin).

[2]Hermitekurver og hermiteinterpolasjon er oppkalt etter Charles Hermite (1822 – 1901), en fransk matematiker som forsket på tallteori, kvadratiske former, invariant teori, ortogonale polynomer, abelske og elliptiske funksjoner, og algebra.



Figur 4.6: *Et plott av de fire hermitebasisfunksjonene formulert i (4.19). Funksjonene knyttet til vektorene er røde. Man kan tydelig se antisymmetrien mellom funksjonene.*

En viktig egenskapen for 3.-grads basisfunksjoner, $\mathbf{H}_3(t) = [H_1(t),\ H_2(t),\ H_3(t),\ H_4(t)]$, er

$$\begin{aligned}\mathbf{H}_3(0) &= [1,\ 0,\ 0,\ 0],\\ \mathbf{H}_3(1) &= [0,\ 1,\ 0,\ 0],\\ \mathbf{H}_3'(0) &= [0,\ 0,\ 1,\ 0],\\ \mathbf{H}_3'(1) &= [0,\ 0,\ 0,\ 1].\end{aligned} \tag{4.20}$$

Legg merke til at $H_1(t) + H_2(t) \equiv 1$. Dette kalles å oppfylle egenskapen **partisjonering av enheten**. Disse to basisfunksjonene, $H_1(t)$ og $H_2(t)$, blender punktene. De to andre basisfunksjonene $H_3(t)$ og $H_4(t)$ blender vektorer, og summerer derfor ikke opp til 1. Grunnen til at dette er viktig ble forklart i seksjonen om affine rom på side 14. Bruken av potensbasiser kalles noen ganger algebraisk form mens bruk av hermitebasiser kalles geometrisk form.

Hermiteinterpolasjon kan også utføres med polynomer av høyere grad. Vi skal derfor se på formelen for polynombaserte kurver av grad 5 som bruker både 1.- og 2.-deriverte,

$$\begin{aligned}\alpha(t) &= a_0 + a_1\, t + a_2\, t^2 + a_3\, t^3 + a_4\, t^4 + a_5\, t^5\\ \alpha'(t) &= a_1 + 2a_2\, t + 3a_3\, t^2 + 4a_4\, t^3 + 5a_5\, t^4.\\ \alpha''(t) &= 2a_2 + 6a_3\, t + 7a_4\, t^2 + 20a_5\, t^3.\end{aligned} \tag{4.21}$$

Vi starter med utgangspunktet i (4.21) å beregne verdien i startpunktet $\alpha(0)$ og endepunktet $\alpha(1)$, samt 1.-deriverte (tangentvektor) i startpunktet $\alpha'(0)$ og i endepunktet $\alpha'(1)$ og 2.-deriverte i startpunktet $\alpha''(0)$ og i endepunktet $\alpha''(1)$:

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

Vi reorganiserer formuleringene på matriseform og får

$$\begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \\ \alpha''(0) \\ \alpha''(1) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6 & 12 & 20 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix}. \tag{4.22}$$

Vi snur uttrykket i (4.22) og inverterer matrisen og får

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ -10 & 10 & -6 & -4 & \frac{-3}{2} & \frac{1}{2} \\ 15 & -15 & 8 & 7 & \frac{3}{2} & -1 \\ -6 & 6 & -3 & -3 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \\ \alpha''(0) \\ \alpha''(1) \end{pmatrix}. \tag{4.23}$$

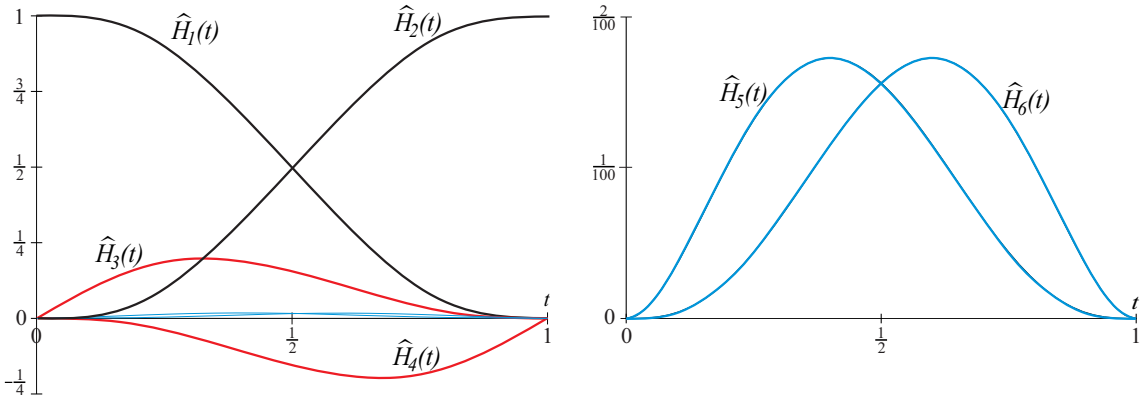
Vi omskriver (4.21), $\alpha(t)$ til vektorer og et indreprodukt, dvs

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

Det neste trinnet er å erstatte koeffisientvektoren $a_i,\ i = 0,1,2,3,4,5$ på høyre side i formuleringen ovenfor med uttrykket på høyre side av (4.23),

$$\alpha(t) = \left(1,\, t,\, t^2,\, t^3,\, t^4,\, t^5\right) \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ -10 & 10 & -6 & -4 & \frac{-3}{2} & \frac{1}{2} \\ 15 & -15 & 8 & 7 & \frac{3}{2} & -1 \\ -6 & 6 & -3 & -3 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \\ \alpha''(0) \\ \alpha''(1) \end{pmatrix}. \tag{4.24}$$

Til slutt multipliserer vi vektoren med potensbasisene på venstre side med matrisen, og vi får det nye settet med hermitebasisfunksjoner til en 5.-grads polynombasert kurve. De seks basisfunksjonene er plottet i figur 4.7. Oppsumert får vi:



Figur 4.7: *Et plott av de seks 5.-grads hermitebasisfunksjonene definert i (4.26). Funksjonene til de 2.-deriverte er plottet separat på høyre side siden verdiene er så små at vi knapt ser dem sammen med de andre basisfunksjonene i plottet på venstre side.*

Hermitekurver, 5.-grad

er polynombaserte kurver på følgende form:

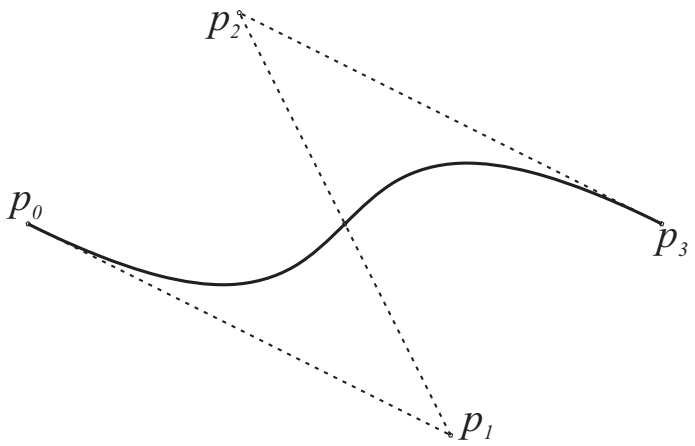
$$\begin{aligned}\alpha(t) = {} & \alpha(0)\,\widehat{H}_1(t) + \alpha(1)\,\widehat{H}_2(t) + \alpha'(0)\,\widehat{H}_3(t) + \\ & \alpha'(1)\,\widehat{H}_4(t) + \alpha''(0)\,\widehat{H}_5(t) + \alpha''(1)\,\widehat{H}_6(t), \qquad t \in [0,1],\end{aligned} \tag{4.25}$$

hvor koeffisientene er verdien ved starten $\alpha(0)$ og slutten $\alpha(1)$, den 1.-deriverte ved start $\alpha'(0)$ og slutt $\alpha'(1)$, og den 2.-deriverte ved start $\alpha''(0)$ og slutt $\alpha''(1)$. De seks 5.-grads hermitebasisfunksjonene er

$$\begin{aligned}
\widehat{H}_1(t) &= 1 - 10t^3 + 15t^4 - 6t^5 &&= \left(6t^2 + 3t + 1\right)(1-t)^3 \\
\widehat{H}_2(t) &= 10t^3 - 15t^4 + 6t^5 &&= \left(6t^2 - 15t + 10\right)t^3 \\
\widehat{H}_3(t) &= t - 6t^3 + 8t^4 - 3t^5 &&= (3t+1)(1-t)^3 t \\
\widehat{H}_4(t) &= -4t^3 + 7t^4 - 3t^5 &&= (3t-4)(1-t)t^3 \\
\widehat{H}_5(t) &= \tfrac{1}{2}t^2 - \tfrac{3}{2}t^3 + \tfrac{3}{2}t^4 - \tfrac{1}{2}t^5 &&= \tfrac{1}{2}(1-t)^3 t^2 \\
\widehat{H}_6(t) &= \tfrac{1}{2}t^3 - t^4 + \tfrac{1}{2}t^5 &&= \tfrac{1}{2}(1-t)^2 t^3.
\end{aligned} \tag{4.26}$$

4.4 Bézierkurver

I figur 4.5 er kurven fra (4.4) plottet sammen som de fire hermitekoeffisientene, to punkt og to vektorer, som alle er merket. Vi skal nå erstatte de to vektorene med to punkt slik at vi kun har punkt som koeffisienter. Merk at dimensjonen til funksjonsrommet i dette eksempelet er 4, og at vi derfor vil ende opp med 4 punkt som koeffisienter. Dette gjør at punktene kan kobles sammen med 3 linjestykker (se figur 4.8). Lengden på den 1.-deriverte vektoren er hastigheten, og siden parameterområdet er 1, er hastigheten (lengden til den 1.-deriverte) og kurvelengden i gjennomsnitt like stor. Vi bruker derfor $\frac{1}{3}$ av lengden til vektorene for å finne punktene. Vi navngir punktene for p_i og får da



Figur 4.8: *Et plott av kurven som først ble formulert (4.4). kontrollpolygonet til bézier-kurven inkludert de fire kontrollpunktene (koeffisientene) er også plottet.*

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

Vi omskriver dette til en matrise/vektor-form,

$$
\begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & \frac{1}{3} & 0 \\ 0 & 1 & 0 & -\frac{1}{3} \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}, \tag{4.27}
$$

og vi snur og inverterer,

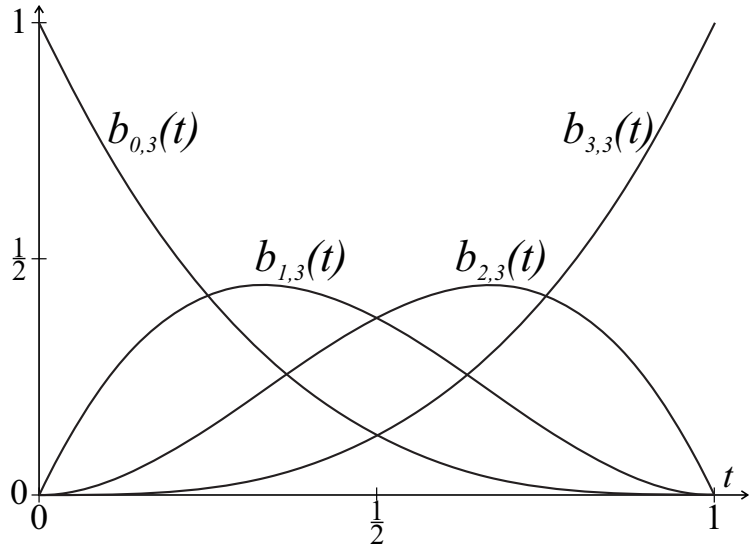
$$
\begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}. \tag{4.28}
$$

Fra formelen i (4.17) har vi

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

Ved å erstatte hermitekoeffisienten med høyre side av (4.28) får vi

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



Figur 4.9: *Et plott av de fire 3. grads bernsteinpolynomer som er gitt i formelen (4.30) og som brukes til å generere 3.-grads bézierkurver som er definert i (4.29).*

Til slutt multipliserer vi potensbasisen med de to matrisene, og får da,

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

Vi har nå et nytt sett med basisfunksjoner for kurven der vi først hadde en vanlig potensbasis (4.10) og deretter hermitebasis, (4.18). Dette nye settet gir:

Bézierkurver av grad 3

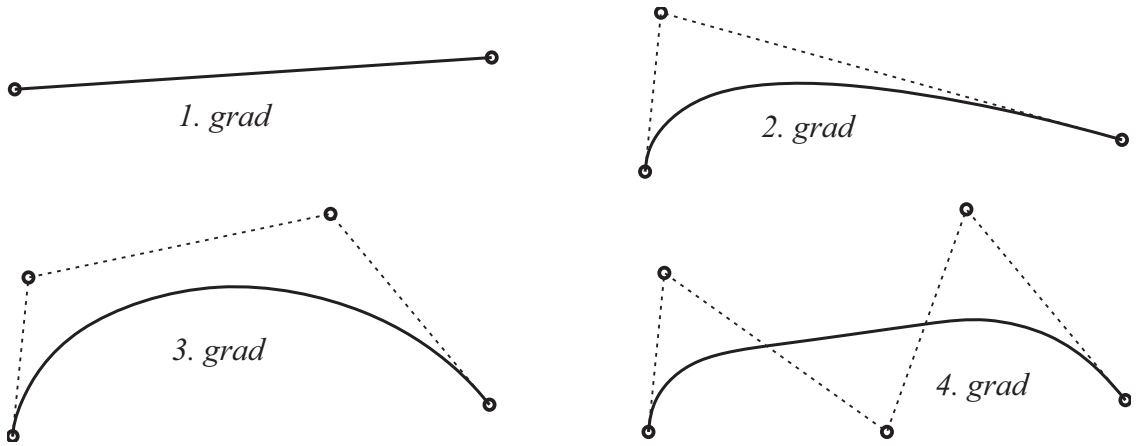
er entydig definert av et sett med fire ordnede punkt som danner et kontrollpolygon (plottet som stiplede linjer i eksemplet i figur 4.8). Parameterdomenet er $[0,1]$ og formelen er:

$$\alpha(t) = p_0\, b_{0,3}(t) + p_1\, b_{1,3}(t) + p_2\, b_{2,3}(t) + p_3\, b_{3,3}(t) = \sum_{i=0}^{3} p_i\, b_{i,3}(t), \quad t \in [0,1], \tag{4.29}$$

der de 4 bézier-basisfunksjonene er settet med 3.-grads bernsteinpolynomer

$$\begin{aligned} b_{0,3}(t) &= (1-t)^3 \\ b_{1,3}(t) &= 3t(1-t)^2 \\ b_{2,3}(t) &= 3t^2(1-t) \\ b_{3,3}(t) &= t^3. \end{aligned} \tag{4.30}$$

Det faktum at matrisen i (4.27) er invertert i (4.28) beviser at settet med 3.-grads bézierbasisfunksjoner (dvs. bernsteinpolynomene) er lineært uavhengige og dermed danner en basis for 3.-grads polynomkurver. Disse fire basisfunksjonene (bernsteinpolynomene) er plottet i figur 4.9, og summen av dem er faktisk 1 over hele domenet $[0,1]$.



Figur 4.10: *Et plott av fire bézierkurver av grad 1, 2, 3 og 4. Også kontrollpunktene (merket som sirkler) og kontrollpolygonene (striplete linjer) er plottet. Førstegradskurven og dens kontrollpolygon sammenfaller.*

Settet med basisfunksjoner for 3.-grads bézierkurver kalles bernsteinpolynomer av grad 3. Det finnes ett sett med bernsteinpolynomer for hver grad. I den neste seksjonen skal vi se nærmere på bernsteinpolynomene, egenskaper og hvordan de kan genereres. Hvert sett oppfyller imidlertid kravet om å være et sett med basisfunksjoner for deres respektive grad av polynomfunksjoner på domenet $[0,1]$. Bézierkurver bruker derfor et sett med bernsteinpolynomer som deres blendingsfunksjoner. Det følger da at det er bézierkurver for alle polynomgrader d og at de er entydig definert av et ordnet sett med $d+1$-punkt.

Bézierkurver av grad d

er entydig definert av $d+1$ kontrollpunkt som beskriver et kontrollpolygon. Det generelle uttrykket for en bézierkurve med parameterdomene $[0,1]$ og grad d er:

$$\alpha(t) = \sum_{i=0}^{d} p_i \, b_{i,d}(t), \quad t \in [0,1], \tag{4.31}$$

hvor koeffisientene $p_i,\ i = 0,1,...,d$ er $d+1$-punkt som definerer kontrollpolygonet til bézierkurven. Settet med de $d+1$ basisfunksjonene $b_{i,d}(t),\ i = 0,..,d$ er Bernsteinpolynomene:

$$b_{i,d}(t) = \begin{pmatrix} d \\ i \end{pmatrix} t^i (1-t)^{d-i},\ i = 0,1,...,d. \tag{4.32}$$

Bézierkurver starter i det første kontrollpunktet og slutter i det siste kontrollpunktet. Tangenten til kurven er lik retningen til kontrollpolygonet i start- og sluttpunktet. Kontrollpolygonet modellerer kurven på en slik måte at variasjonen av kurven er mindre enn variasjonen til kontrollpolygonen (kalt den variasjonsforminskende egenskapen, se punkt 6 på side 85). Eksempler kan sees i figurene 4.8 og 4.10.

Bézierkurver[3] har lenge vært et av de mest populære kurveformatene. De er entydig de-

[3]Bézierkurver er oppkalt etter Pierre Bézier (1910 – 1999). Han jobbet for Renault fra 1933 – 1975,

finert av et ordnet sett med kontrollpunkt som også danner et kontrollpolygon. Som vi kan se i figur 4.10, modellerer kontrollpolygonet kurven. I figuren er det en 1.-grads, en 2.-grads, en 3.-grads og en 4.-grads bézierkurve som alle er plottet sammen med sine respektive kontrollpolygon. 1.-grads-kurven er helt sammenfallende med sitt kontrollpolygon. Senere skal vi se at bézierkurver et spesialtilfeller av B-splinekurver. Flere algoritmer for beregning av bézierkurver vil derfor bli vist senere.

4.4.1 Bernsteinpolynomer

Bernsteinpolynomene er basisfunksjonene for bézierkurver på domenet $[0,1]$. Navnet bernsteinpolynomer ble først brukt på selve funksjonen der disse polynomene var basisfunksjoner.[4] I dag er det imidlertid vanlig å kalle settet med basisfunksjoner for bernsteinpolynomer. Den generelle formelen for et sett med bernsteinpolynom med grad d er gitt i (4.32), og er:

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

og hvor parameterdomet er $[0,1]$. For hver grad d har vi alså et sett med $d+1$ funksjoner. Nedenfor ser vi formelen for de 6 første settene med bernsteinpolynomer, dvs. for $d = 0,1,2,3,4,5$, ett sett på hver rad under:

$$\begin{array}{ccccccc}
 & & & 1 & & & \\
 & & & 1-t & t & & \\
 & & & (1-t)^2 & 2t(1-t) & t^2 & \\
 & & (1-t)^3 & 3t(1-t)^2 & 3t^2(1-t) & t^3 & \\
 & (1-t)^4 & 4t(1-t)^3 & 6t^2(1-t)^2 & 4t^3(1-t) & t^4 & \\
(1-t)^5 & 5t(1-t)^4 & 10t^2(1-t)^3 & 10t^3(1-t)^2 & 5t^4(1-t) & t^5 &
\end{array} \tag{4.33}$$

og hvor notasjonen er som følgende:

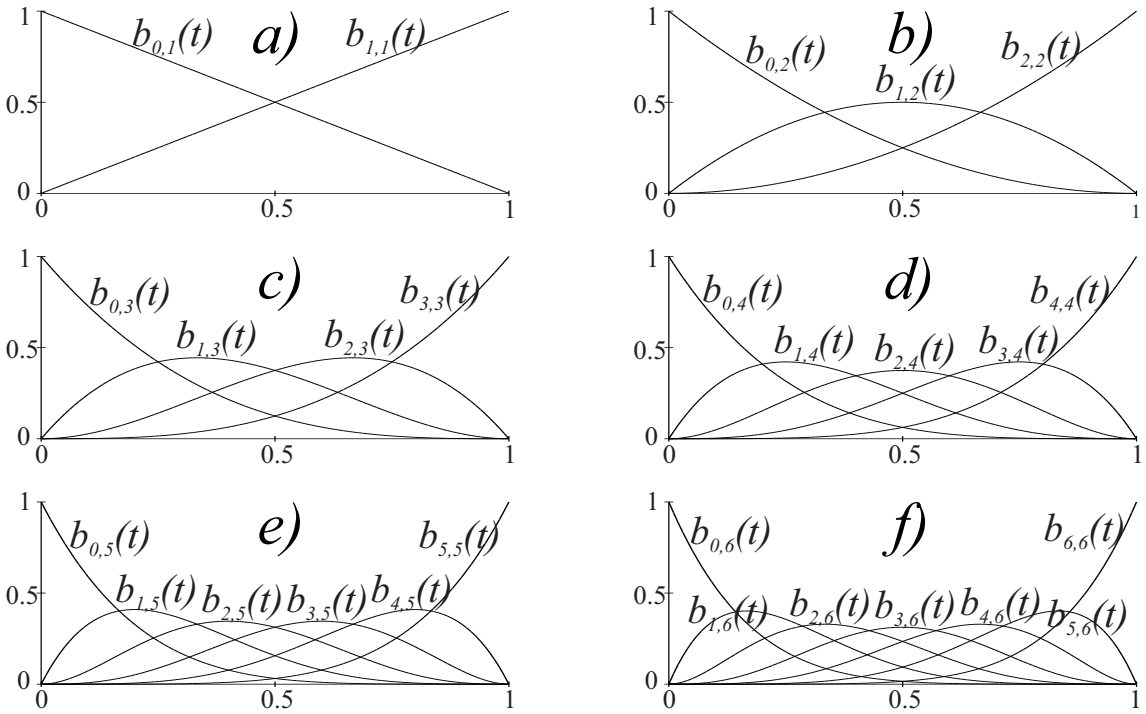
$$\begin{array}{ccccc}
 & & b_{0,0}(t) & & \\
 & b_{0,1}(t) & & b_{1,1}(t) & \\
b_{0,2}(t) & & b_{1,2}(t) & & b_{2,2}(t) \\
b_{0,3}(t) & & b_{1,3}(t) & & b_{2,3}(t) \qquad b_{3,3}(t)
\end{array}$$

———————- og så videre ———————

I figur 4.11 er de seks settene med bernsteinpolynomer av grad 1 til 6 plottet. Dette er funksjonene vi finner i de påfølgende radene i 4.33.

hvor han utviklet sitt UNISURF DAK-DAP-system. Han publiserte og patenterte resultatene i 1962, selv om Paul de Castejau (1930 –) allerede i 1959 hadde laget en algoritme for å beregne bézierkurver, men kun publisert dette i en intern Citroén-rapport.

[4] Polynomet oppstod som et resultat av arbeidet til Sergei Natanovich Bernstein, en ukrainsk matematiker (1880-1968). Bernstein introduserte polynomene i 1911 (publisert i [10]), og brukte dem i et bevis på Stone-Weierstrass aproksimasjonsteorem. For å gjøre en grudigere studie av disse polynomene, se side 183–186 i [91] eller side 108–126 i [35].



Figur 4.11: *Seks sett med bernsteinpolynomer er plottet.* **a)** *er grad 1,* **b)** *er grad 2,* **c)** *er grad 3,* **d)** *er grad 4,* **e)** *er grader 5 og* **f)** *er settet av grad 6. Alle settene er basisfunksjoner for bézierkurver av den respektive graden.*

I det følgende skal vi først se på en enkel algoritme for å lage polynomene. Så skal vi se på et bevis for at de danner en basis, dvs. er libært uavhengige, og til slutt vise at hvert sett alltid summer opp til 1 over alt på domenet $[0,1]$.

Bernsteinpolynomer kan lages rekursivt, da med utgangspunkt i bernsteinpolynomene av en grad lavere. Fra (4.32) følger det at 0.-grads bernsteinpolynomet er 1. Hvis vi setter inn et 0-polynom foran og et 0-polynom bak settet av grad $d-1$ vil rekursjonsfomelen alltid være $b_{i,d}(t) = t\ b_{i-1,d-1}(t) + (1-t)\ b_{i,d-1}(t)$ for $i = 0,1,...,d$. I praksis lager vi imidlertid algoritmen uten 0-polynomene og må derfor lage første og siste polynom separat. Metoden er derfor som følger,

1. Først lager vi den til høyre, $b_{d,d}(t) = t\ b_{d-1,d-1}(t)$.
2. Hvis $d > 1$ lager vi alle interne polynomeene på raden: for $i = d-1, d-2, ..., 1$, er $b_{i,d}(t) = t\ b_{i-1,d-1}(t) + (1-t)\ b_{i,d-1}(t)$.
3. Til slutt lager vi den til venstre, $b_{0,d}(t) = (1-t)\ b_{0,d-1}(t)$,

Algoritmen starter med at $b_{0,0}(t) = 1$, dvs. grad 0, og så med start fra høyre i vektoren lages grad for grad. Algoritmen kalles også en pyramidealgoritme (se [77]).

Algoritme 1. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen lager en vektor $\mathbf{b}_d(t) \in \mathbb{R}^{d+1}$, som inneholder verdiene til de $d+1$ bernsteinpolynomene $\{b_{d,i}(t)\}_{i=0}^{d}$. Innput er: graden d til bernsteinpolynomene og parameterverdien $t \in [0,1]$.*

```
vector⟨double⟩ bernstein ( int d, double t )
    vector⟨double⟩ b(d+1);          // Returvektoren, størrelse d + 1.
    b_0 = 1;                        // En generell Cox/deBoor-lignende algoritme
    for ( int i=1; i ≤ d; i++ )     // - for bernsteinpolynomer, regner ut verdien
        b_i = t b_{i-1};            // - til alle d + 1 elementer.
        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. *Bernsteinpolynomene av grad d danner en basis for polynomfunksjoner av grad opptil d på parameterintervallet* $[0,1]$.

Bevis. Dette er det samme som å bevise at funksjonene i et sett bernsteinpolynomer av grad d er lineært uavhengige av hverandre? Dette kan vises på følgende måte (argumentasjonen er også visuelt illustrert i (4.33)):

- Funksjonen til høyre er den eneste med bare ett ledd av grad d.
- I de neste funksjonen fra høyre og mot venstre øker antall ledd med 1 for hver step mot venstre og gradene går fra d og nedover med ett trinn om gangen, intill den siste som har ledd med alle gradene inkludert en konstant.

Det følger at funksjonen til venstre er lineært uavhengig fra de andre da den er den eneste med et konstantledd. Samme argument kan så brukes for neste funksjon fra venstre som er det eneste av de resterende som har et 1.-gradsledd. Et tilsvarende argument kan så brukes for alle resterende ledd, og dermed er beviset komplett. □

Lemma 4.2. *Bernsteinpolynomene av grad d summerer opp til 1 for alle* $d \in \mathbb{N}$. *Vi kaller denne egenskapen; å danne en partisjon av enhet på domenet* $[0,1]$, *og det uttrykkes ved*

$$\sum_{i=0}^{d} b_{i,d}(t) = 1, \quad \textit{for} \quad d \in \mathbb{N} \quad \textit{og} \quad t \in [0,1].$$

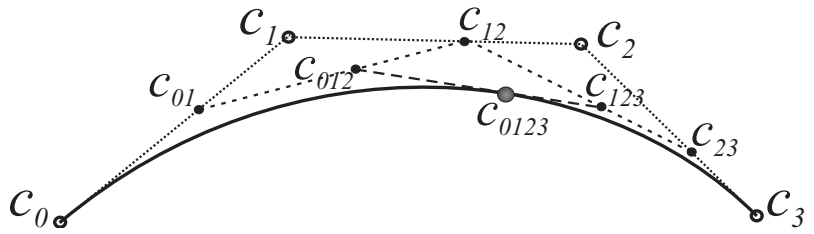
Bevis. Den første raden, $b_{0,0}(t) = 1$ summerer åpenbare opp til 1. Vi kaller så summen av et sett av grad d (en rad) for $s_d(t)$. Med utgangspunkt i den generelle rekursjonsformelen får vi:

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

Hvis vi så ordner dette uttrykket får vi

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

som viser at summen av en rad er lik summen av raden over. Ved induksjon viser dette at et sett med Bernstein-funksjoner summerer opp til 1 for alle polynomgrader. □



Figur 4.12: *En bézierkurve (heltrukken) og punktene (sirkler) og striplete linjene som illustrerer de Casteljaus algoritme for en tredjegrads bézierkurve ved $t = 0,6$.*

4.4.2 Faktorisering og de Casteljau's hjørnekuttings-algoritme

For ytterligere å undersøke algoritmen til bernsteinpolynomer og bézierkurver, starter vi med lineærinterpolasjon mellom to punkt c_0 og c_1 (i planet eller i rommet (2D og 3D):

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

Når $t \in [0,1]$ får vi linjesegmentet mellom c_0 og c_1.

Gitt en sekvens av punkt c_0, c_1, c_2 og c_3. Hvis, vi for en gitt $t \in [0,1]$ interpolerer punktene parvis, dvs. c_0 og c_1, c_1 og c_2, c_2 og c_3, vil vi få tre nye punkt, som vi henholdsvis kaller c_{01}, c_{12} og c_{23}. I matrisevektor-notasjon har vi da gjort

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

Hvis vi så gjentar prosessen med utgangspunkt i disse nye punktene får vi

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

og til slutt får vi

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

Denne prosessen kalles de Casteljaus[5] hjørneskjæringsalgoritme for å evaluere bézierkurver (se [39] og [41]). Prosessen er illustrert i figur 4.12 .

Hvis vi setter sammen disse tre multiplikasjonene, får vi følgende uttrykk for en tredjegrads bézierkurve,

$$c(t) = \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}. \quad (4.34)$$

[5]Paul de Casteljau (1930 -) er en fransk fysiker og matematiker. I 1959, mens han jobbet hos Citroën, utviklet han en algoritme for å beregninger det som senere ble kalt bézierkurver. Han fikk imidlertid ikke anledning til offentlig å publisere sitt tidlige arbeid, bare internt hos Citroën.

Hvis matrisene multipliseres fra venstre følger vi pyramiden i (4.33) og Algoritme 1. For hver matrise vi multipliserer inn får vi en ny rad i pyramiden i (4.33). Dette betyr at når vi har multiplisert sammen alle matrisene har vi de fire bernsteinpolynomene av grad 3, dvs.

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

Vi kaller disse matrisene for bernstein-faktormatriser og benevner dem $T_d(t)$. For eksempel hvis $d=2$ får vi

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

Vi har nå følgende generelle definisjon av bernstein-faktormatriser.

Definisjon 4.5. *$T_d(t)$ er en matrise med dimensjon $d \times (d+1)$ der det på rad nummer j*

a) bare er element nummer j og element nummer $j+1$ som er forskjellig fra 0.
b) og hvor (i bézier-tilfellet) element nummer j er $1-t$ og element nummer $j+1$ er t.

Merk at i det vanlige bézier-tilfellet bruker vi t. Vi kan også skalere og translere domenet ved å bruke $w(t;s,e) = \frac{t-s}{e-s}$ i stedet for t. Senere vil vi se at vi i B-splines bruker en translering- og skaleringsfunksjon $w_{d,i}(t) = \frac{t-t_i}{t_{i+d}-t_i}$, se side 82 og 87.

Fordi vi har valgt denne notasjonen for bernstein-faktormatrisene, vil vi bruke følgende notasjon for vektoren av bernsteinpolynomene av grad d:

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

Det følger at formelen i (4.34) da blir

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

der $\mathbf{c} = (c_0, c_1, c_2, c_3)^T$ er en vektor av punkt. Tradisjonelt er imidlertid formelen

$$c(t) = \sum_{i=0}^{3} b_{i,d}(t)\ c_i. \quad \text{se (4.29)}$$

Legg merke til at den deriverte av matrisen $T_d(t)$ er en matrise av konstanter. Vi benevner den derfor T_d'. For $d=1$ får vi

$$T_1' = (-1, \quad 1).$$

Matrisenotasjonen til en bézierkurve slik den er formulert i (4.34) kan nå skrives kompakt, og hvis vi som eksempel bruker en 3.-grads bézierkurve og dens tre deriverte, får vi følgende notasjon på matriseform.

$$\begin{array}{rlll} c(t) = & \mathbf{T}^3(t)\ C & = & T_1(t)T_2(t)T_3(t)\ C, \\ c'(t) = & 3\ \mathbf{T}^2(t)\mathbf{T}'\ C & = & 3\ T_1(t)T_2(t)\ T_3'\ C, \\ c''(t) = & 6\ \mathbf{T}^1(t)\mathbf{T}'^2\ C & = & 6\ T_1(t)\ T_2'T_3'\ C, \\ c'''(t) = & 6\ \mathbf{T}'^3\ C & = & 6\ T_1'T_2'T_3'\ C. \end{array} \tag{4.35}$$

Indeksene i faktormatrisen angir antall rader i matrisen. Den deriverte av matrisen $T(t)$, betegnet T' er uavhengig av t. Når vi ekspanderer (4.35), får vi,

$$
\begin{aligned}
c(t) &= \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\
c'(t) &= 3 \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}, \\
c''(t) &= 6 \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}, \\
c'''(t) &= 6 \begin{pmatrix} -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}.
\end{aligned}
\tag{4.36}
$$

Formlene er også avhengig av at $T_d(t)$-matrisene og deres deriverte er kommutative. Det betyr at hvilken av matrisene som erstattes av en derivertematrise kan endres uten at resultatet endres. For eksempel er $T_1'T_2(t)T_3(t) = T_1(t)T_2'T_3(t) = T_1(t)T_2(t)T_3'(t)$, og det følger dermed at

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

En forklaring og et bevis på kommutativitetsrelasjonen er gitt i vedlegg C.2.

Når vi oppsummerer faktoriseringen, kan vi tydelig se at:

i) Hvis vi regner fra høyre (og hopper over nullene), får vi de Casteljau's hjørnekuttingsalgoritme.
ii) Hvis vi regner fra venstre, får vi en algoritme av typen Cox/de Boor (se Definisjon 6.3 på side 82).
iii) Hvis vi multipliserer sammen alle matrisene uten koeffisientvektoren på høyre side, får vi bernsteinpolynomene og deres deriverte. [6]

[6] I seksjon 6.2.3, er matrisenotasjonen også brukt på B-splines. Dette gir en enkelt Cox/de Boor-algoritmen for B-splines og også en geometrisk de Casteljau-beskrivelse når vi går fra høyre mot venstre. Hvis vi multipliserer matrisene får vi også en utvidet versjon av en "evaluator". Matrisenotasjon for B-splines er også behandlet i [103] og [117].

4.4.3 Bernstein/hermitematrisen

Vi tar utganspunkt i (4.36), multipliserer de tre matrisene i hver rad sammen og setter sammen resultatet til èn matrise, dvs.

$$\begin{pmatrix} c(t) \\ c'(t) \\ c''(t) \\ c'''(t) \end{pmatrix} = \begin{pmatrix} (1-t)^3 & 3t(1-t)^2 & 3t^2(1-t) & t^3 \\ -3(1-t)^2 & 9t^2-12t+3 & -3t(3t-2) & 3t^2 \\ 6-6t & 18t-12 & 6-18t & 6t \\ -6 & 18 & -18 & 6 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}. \tag{4.37}$$

Matrisen i (4.37) er Bernstein/hermitematrisen av orden 4. Den er egentlig settet av bernsteinpolynomer av grad 3 og alle deres deriverte. Legg merke til at hvis vi snur formuleringen i (4.37) og inverterer matrisen får vi hermiteinterpolasjonsformelen

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 & -t & \frac{1}{2}t^2 & -\frac{1}{6}t^3 \\ 1 & \frac{1}{3}-t & \frac{1}{2}t^2-\frac{1}{3}t & \frac{1}{6}t^2(1-t) \\ 1 & \frac{2}{3}-t & \frac{1}{2}t^2-\frac{2}{3}t+\frac{1}{6} & -\frac{1}{6}t(1-t)^2 \\ 1 & 1-t & \frac{1}{2}t^2-t+\frac{1}{2} & \frac{1}{6}(1-t)^3 \end{pmatrix} \begin{pmatrix} c(t) \\ c'(t) \\ c''(t) \\ c'''(t) \end{pmatrix}. \tag{4.38}$$

Husk at settet med bernsteinpolynomer av en gitt grad summerer opp til 1, dermed følger det at den første raden i bernstein/hermitematrisen (4.37) summerer opp til 1 og alle de andre radene summerer opp til 0. Det følger også, som vi kan se i (4.37) at alle elementene i den første kolonnen i den inverterte matrisen er 1, jfr. taylorekspansjonen i seksjon 5.5.

Fra seksjonene 2.4, 2.5 og 2.6 vet vi at et punkt i homogene koordinater har en siste koordinat som er 1,. Likeledes er den siste koordinaten til en vektor 0. Egenskapene til bernstein/hermitematrisen gjenspeiler dette: c_0, c_1, c_2, c_3 og $c(t)$ er punkt, mens $c'(t)$, $c''(t)$, og $c'''(t)$ er vektorer. Både (4.37) og (4.38) er konsistente for punkt og vektorer.

Bernstein/hermitematrisen av orden k

der $k = d+1$ og d er graden til bernsteinpolynomene den representerer. Matrisen er en $k \times k$ inverterbar matrise definert som følgende

$$\mathbf{B}_d(t,\delta) = \begin{pmatrix} b_{0,d}(t) & b_{1,d}(t) & \cdots & b_{d,d}(t) \\ \delta\, Db_{0,d}(t) & \delta\, Db_{1,d}(t) & \cdots & \delta\, Db_{d,d}(t) \\ \vdots & \vdots & \ddots & \vdots \\ \delta^d D^d b_{0,d}(t) & \delta^d D^d b_{1,d}(t) & \cdots & \delta^d D^d b_{d,d}(t) \end{pmatrix}. \tag{4.39}$$

hvor $d > 0$, $t \in [0,1]$ og $\delta > 0$.
Vanligvis er $\delta = 1$, men hvis domenet er $[a,b]$ og ikke $[0,1]$, og inngangsparameteren er $s \in [a,b]$, må vi skalere og translateret, og dermed bruke $t = \frac{s-a}{b-a}$ og $\delta = \frac{1}{b-a}$.

Det spesielle med denne definisjonen (4.39) av matrisen er skaleringen δ^j, hvor eksponenten j, er radnummeret (den første raden er nummerert 0). Grunnen til denne skaleringen er egentlig skaleringen av parameterdomenet, dvs. reparametrisering.

Som vi forstår fra (4.37) og (4.38) er det to måter vi kan bruke matrisen på:

1. Regne ut bernsteinpolynomene og deres deriverte og dermed posisjonen og de deriverte til en bézierkurve, jmf. (4.37), dvs. lage en evaluator, jmf. merknad 3.1.

2. Lage en bézierkurve som for en gitt parameterverdi t interpolerer en gitt posisjon og d etterfølgende deriverte (hermiteinterpolasjon), jmf. uttrykk (4.38).

Følgende algoritme lager matrisen $\mathbf{B}_d(t,\delta)$ slik den er beskrevet i (4.39).

Algoritme 2. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen lager den kvadratiske matrisen $\mathbf{B}_d(t,\delta) \in \mathbb{R}^{d+1\times d+1}$, som i den første raden inneholder verdiene til de $d+1$ bernsteinpolynomene $\{b_{d,i}(t)\}_{i=0}^{d}$, og i de følgende radene verdiene til de deriverte, $\{D^j b_{d,i}(t)\}_{i=0}^{d}$, $\quad j=1,2,\ldots,d$ dvs. i hver av de følgende j radene. I tillegg, skal alle elementene i alle rader der $j>0$ (radene er nummerert fra 0 til d), multipliseres med δ^j. Hvis matrisen skal brukes i en generell evaluator, vil normalt $\delta=1$, og hvis matrisen skal brukes i hermiteinterpolasjon må vi bruke δ som er parameterintervallet vi ønsker å etterligne. Innput er: graden d til bernsteinpolynomene, parameterverdien $t\in[0,1]$, og skaleringsfaktoren δ.*

matrix⟨double⟩ **BernsteinHermiteMat (** int d, double t, double δ **)**
  matrix⟨double⟩ B(d+1,d+1); // Returmatrisen, med dimensjon $(d+1)\times(d+1)$.
  $B_{d-1,0} = 1-t$;
  $B_{d-1,1} = t$; // En tilpasset Cox/deBoor-algoritmen for
  for (int i=d-2; $i\geq 0$; $i--$) // - bernsteinpolynomer, dvs. beregning av
    $B_{i,0} = (1-t)\,B_{i+1,0}$; // - en trekant med verdier av $b_{i,j}(t)$
    for (int j=1; $j<d-i$; j++) // - av henholdsvis grad 1 til d, hver i sin rad.
      $B_{i,j} = t\,B_{i+1,j-1} + (1-t)\,B_{i+1,j}$;
    $B_{i,d-i} = t\,B_{i+1,d-i-1}$;
  $B_{d,0} = -\delta$;
  $B_{d,1} = \delta$; // Multipliser alle rader unntatt den øverste
  for (int k=2; $k\leq d$; k++) // - med de deriverte-matrisene i
    double $s = k\,\delta$; // - uttrykk (4.36), og skaleringene,
    for (int $i=d$; $i>d-k$; $i--$) // - så hver rad utvider antallet
      $B_{i,k} = s\,B_{i,k-1}$; // - av ikke-null elementer til 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;

Ordenen til denne algoritmen er helt klart mellom $\mathcal{O}(n^2)$ og $\mathcal{O}(n^3)$, og følgende tabell viser antall multiplikasjoner avhengig av d, for d opptil 10.

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

Hastigheten til algoritmen er i egentlig ikke avgjørende fordi algoritmen normalt skal brukes når kurvene lages, eller når samplingen endres. Men for små d-verdier ($d<4$) er algoritmen egentlig ganske rask.

4.4.4 Gradsheving av bézierkurver

Husk at polynombaserte kurver i utganspunktet har en potensform,

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

hvor koeffisientene a_i er elementer av det rommet kurven er i, typisk $\mathbb{R}^2$ eller $\mathbb{R}^3$.

Enhver polynombasert kurve av grad $g < d$ kan uttrykkes på denne formen, men hvor koeffisientene $|a_i| = 0,\ i > g$. Gradsheving av bézierkurver handler i utgangpunktet om å gjøre dette, men hvor vi bruker bernsteinpolynomer som basisfunksjoner i stedet for potensbasis. Dermed kan gradheving gjøres ved å endre basis fra Bernstein til potensbasis, øke graden og konvertere tilbake til bernsteinpolynomer. En enklere måte å finne formelen på er imidlertid:

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

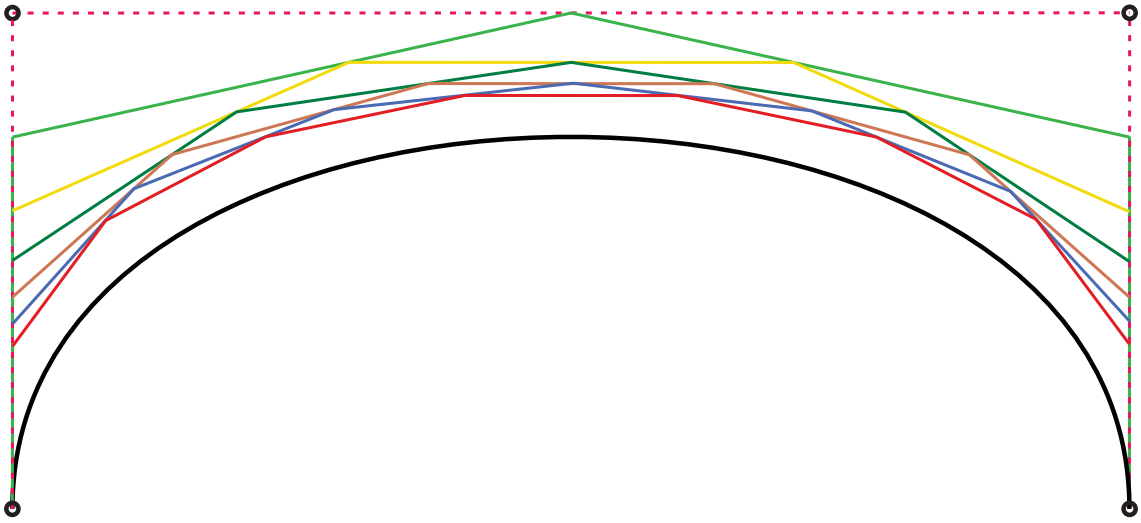
som på matriseform blir:

Béziergradshevings-matrise

Gitt en bézierkurve av grad d, $c(t) = \sum_{i=0}^{d} b_{i,d}(t) c_i$. For å øke graden til $d+1$ må vi generere et nytt sett med kontrollpunkt der vi beholder det første punktet, oppretter d nye punkt i midten og så beholder det siste. På vektor/matriseform får vi, $\tilde{\mathbf{C}} = \mathbb{D}_d\, \mathbf{C}$, hvor $\tilde{\mathbf{C}} = (\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_d)$, $\mathbf{C} = (c_0, c_1, \ldots, c_d)$, og $\mathbb{D}_d$ er bézier gradshevings-matrisen for dimensjon $d \times (d+1)$, dvs.

$$\begin{pmatrix} \tilde{c}_1 \\ \tilde{c}_2 \\ \vdots \\ \tilde{c}_d \end{pmatrix} = \begin{pmatrix} 1 - \frac{d}{d+1} & \frac{d}{d+1} & 0 & \cdots & 0 \\ 0 & 1 - \frac{d-1}{d+1} & \frac{d-1}{d+1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 1 - \frac{1}{d+1} & \frac{1}{d+1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{d-1} \\ c_d \end{pmatrix} \tag{4.40}$$

For bézierkurver hvor graden er hevet til $d+1$ blir da settet av $d+2$ kontrollpunkt $\{c_0, \tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_{d-1}, \tilde{c}_d, c_d\}$.



Figur 4.13: *En 3.-grads bézierkurve (svart) og de fire kontrollpunktene (sirkler) og kontrollpolygon (striplet rød) samt kontrollpolygonene til et 4.-grads (grønnt), 5.-grads (gult), 6.-grads (mørkegrønnt), 7.-grads (brunt), 8.-grads (blått) og 9.-grads (rødt) bézierkurve generert fra den orginale 3.-grads bézierkurven med gradshevings-algoritmen.*

Fra matrisen i (4.40) kan vi tydelig se at gradsheving er hjørneskutting, noe som er tydelig illustrert i Figur 4.13. I figuren ser vi tydelig at alle de nye kontrollpunktene vi får ved å øke graden med én ligger på det forrige kontrollpolygonet. Under følger gradshevings-matrisen fra grad 2 til 3, matrisen fra grad 3 til 4, matrisen fra grad 4 til 5 og matrisen fra grad 5 til 6:

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

Det følger dermed at for en enkel gradsheving får vi følgende koeffisienter uttrykt med koeffisientene fra en polynomgrad lavere:

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

Et eksempel på kontinuerlig gradheving er gitt i figur 4.13. Det er i utgangspunktet en 3.-graders bézierkurve der de fire kontrollpunktene er markert som svarte sirkler. Kurvens grad heves en grad om gangen opptil grad 9, og som vi kan se på det siste røde kontroll-polygonet vil vi til slutt ha 10 kontrollpunkt. Eksemplet viser også at kontrollpolygonet konvergerer svært sakte mot kurven.

Tabell 4.1: *Matriser for basisskifte*

Type basisfunksjon	fra potensbasis	til potensbasis
Hermite 3.-grads	$\mathrm{H} = \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$	$\mathrm{H}^{-1} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 3 \end{bmatrix}$
Bernstein 3.-grads	$\mathrm{B} = \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\mathrm{B}^{-1} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & \frac{1}{3} & \frac{2}{3} & 1 \\ 0 & 0 & \frac{1}{3} & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

4.5 Konvertering mellom hermite- og bézier-format

Hermitekurver og bézierkurver er mye brukt i dataprogram. Derfor kan det være svært nyttig å kunne konvertere fra ett format til et annet. Faktisk kan alle polynombaserte kurve-format av samme grad og over samme domene konverteres til hverandre med en basisskifte-matrise.

I tabell 4.1 finner vi basisskifte-matrisene for 3.-grads hermite- og bernsteinbasiser. Vi kaller settene med 3.-grads basiser som dekker $\mathscr{P}_3[0,1]$ for

$$\begin{aligned}
\mathbf{M}_3(t) &= \begin{pmatrix} 1, & t, & t^2, & t^3 \end{pmatrix}^T && \text{potensbasis,} \\
\mathbf{H}_3(t) &= \begin{pmatrix} 1-3t^2+2t^3, & 3t^2-2t^3, & t-2t^2+t^3, & -t^2+t^3 \end{pmatrix}^T && \text{hermitebasis,} \\
\mathbf{T}^3(t) &= \begin{pmatrix} (1-t)^3, & 3t(1-t)^2, & 3t^2(1-t), & t^3 \end{pmatrix}^T && \text{bézierbasis,}
\end{aligned}$$

og får

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

dvs.

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

Gitt en kurve $c(t) = \mathbf{p}\mathbf{H}_3(t)$, hvor $\mathbf{p} = (c(0),\ c(1)\ c'(0),\ c'(1),)$. Det følger at vi kan finne kontrollpunktene $\mathbf{c} = (c_0,\ c_1,\ c_2,\ c_3)$ av den tilsvarende bézierkurven, $c(t) = \mathbf{c}\mathbf{T}^3(t)$, med

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

Som gir

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

4.6 Implementering og tessellering

Rendering (plotting) av parametriske kurver krever normalt tessellering. Dette kan gjøres av enten grafikkenheten, GPU'en eller av en "ordinær" prosessor CPU. Hermitekurver og bézierkurver har normalt ingen interne diskontinuiteter for noen av de deriverte, da med unntak av geometrisk diskontinuitet fra singulariteter basert på "cusp", dvs. hvor hastigheten, lengden på den 1.-deriverte, er null. Derfor er det ikke naturlig å dele parameterdomenet inn i partisjoner. Det vi normalt gjør er å tessellere kurven, dvs. dele kurven i små biter som plottes som linjestykker. Vi skal se nærmere på tre mye brukte tesselleringsmetoder for kurver,

- Uniform tessellering
- Tessellering basert på hastighet
- Tessellering basert på krumning

Uniform tessellering er basert på $n+1$ punkt, som betyr å dele domenet i n like store deler. For hermite- og bézierkurver er domenet 1, dvs. $dt = \frac{1}{n}$ slik at de $n+1$-punktene som bestemmer tesselleringen blir $\{c(i*dt)\}_{i=0}^{n}$. Dette er den enkleste og dermed den mest brukte metoden.

Tessellering basert på hastighet kan utføres mer eller mindre komplekst. En enkel versjon er først å beregne kurvelengden $l(c)$ (se (4.5)) ved hjelp av rombergintegrasjon, jmf. modifisert algoritme 14. Så setter vi $dl = l(c)/n$. Vi starter så med å lage punktene. Det første punktet er $c(0)$, dvs for $t = 0$. For de påfølgende t-verdiene må alle øke forige verdi med en unik dt. For hver ny t-verdi må vi dermed først beregne tidssteget dt. Den er basert på at $|c'(t)|\ dt = dl$ og følgelig er $dt = \frac{dl}{|c'(t)|}$. Dermed får vi neste sampelpunkt $c(\hat{t}+dt)$ hvor $\hat{t}$ er t-verdien ved forige beregning. Det siste punktet er $c(1)$. En mer kompleks metode er å korrigere for at hastigheten kan endre seg over et step. For å beregne tidstrinnet får vi da $\left(|c'(t)| + \frac{\langle c'(t), c''(t)\rangle}{2|c'(t))|} dt\right) dt = dl$. Dette betyr å løse en andregradsligning i stedet for en lineær ligning.

Tessellering basert på krumning starter ikke med antall punkt, men gir bare et minimum antall. Hoved-innputten er imidlertid δ, dvs. den maksimalt tillatte avstanden mellom et kurvesegment og en rett linje. Beregning av et trinn dt er basert på buelengden $b \approx dt\ |c'|$, krumningen κ, (4.7) (der $r = \frac{1}{\kappa}$) og $|c'|$. Vi har $\cos\alpha = \frac{r-\delta}{r}$ og $\alpha = \frac{b}{r}$, så

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

Dette uttrykket forutsetter at krumningen (egentlig produktet av krumningen og hastigheten) ikke er veldig liten. Derfor, hvis vi over et område har noe i nærheten av en rett linje, må vi basere algoritmen på et lite antall punkt og uniform sampling over området.

Kapittel 5

Klassisk interpolasjonsteori

I sin bok om interpolasjon karakteriserte Thiele interpolasjon som *kunsten å lese mellom linjer i en numerisk tabell* (se [156]). Frem til introduksjonen av datamaskiner og spesielt datagrafikk var dette faktisk en veldig presis formulering.

Historien om interpolasjon inkluderer astronomi i Mesopotamia og Babylon, matematikere i antikkens Hellas, kinesiske matematikere og indisk astronomi.

I Europa, tilsynelatende uvitende om resultater oppnådd tidligere i andre deler av verden, begynte interpolasjonsteorien å utvikle seg i kjølvannet av den store "vitenskapelige revolusjonen" ca. 1500 e.Kr. Spesielt utviklingen innen astronomi og fysikk, initiert av Copernicus og videreført av Kepler og Galileo, og som kulminerte med teoriene til Newton, ga et kraftig puff til videre utvikling av matematikken, inkludert det som vi i dag kaller "klassisk interpolasjonsteori". Vi starter derfor med Newton og dividerte differanser.

5.1 Dividerte differanser

Gitt et punktsett $\{p_i\}_{i=0}^n = \{(x_0,y_0),(x_1,y_1),...,(x_n,y_n)\}$. Definisjonen av førsteordens dividerte differansere er,

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

Høyere-ordens dividerte differanser er definert rekursivt, dvs.

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

Dividerte differanser av order 0 benevnes ved $[y_0] = y_0$. Den rekursive prosessen kan beskrives i tabellform (pyramidisk), dvs.

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

Som et eksempel, gitt et punktsett $\{(x_i,y_i)\}_{i=0}^3 = \{(0,0),(1,1),(2,1),(3,0)\}$. Dette punktsettet gir følgende rekursjon og tabell,

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

Hvis datapunktene er gitt som en funksjon, dvs. som t_i og $f(t_i)$, da er datapunktet $(t_i, f(t_i))$. For en hvilken som helst funksjon $f : \mathbb{R} \to \mathbb{R}^m$, $m \in \mathbb{Z}$, og for ethvert sett av to verdier $t_0 \neq t_1$, er førsteordens dividerte differanser nå

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

Høyere-ordens dividerte differanser er også her definert rekursivt, dvs.

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

Noen viktige egenskaper til dividerte differanser,

◇ **Linearitet**

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

◇ **Leibniz regel**

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

◇ **Dividerte differanser er symetriske**, Hvis σ er en permutation av $\{0,\ldots,n\}$ da er

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

◇ **Dividerte differanser og middelverditeoremet**

$f[t_0,\ldots,t_n] = \frac{f^{(n)}(\xi)}{n!}$ hvor ξ er i det åpne intervallet bestemt av den minste og den største av t_m-ene

Formelene for dividerte differanser kan ekspanderes, som vi skal se nedenfor,

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

Vi får da en direkte formel (ikke rekursiv) for dividerte differanser,

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

som kan formuleres mer kompakt,

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

hvor $\omega_n(t) = (t-t_0)\cdots(t-t_n)$, dvs.

$$\omega_n(t) = \prod_{k=0}^{n}(t-t_k), \quad \text{og det følger at} \quad \omega_n'(t_i) = \prod_{k=0,k\neq i}^{n}(t_i - t_k). \tag{5.2}$$

For å utvide dividerte differanser til også å inkludere like verdier, må vi inkludere deriverte. Vi vet at

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

Dermed kan vi definere

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

og hvis, i uttrykket for dividerte differanser, t_i gjentas n ganger får vi en sammenheng mellom dividerte differanser og de deriverte av en funksjon generelt, legg merke til den inverse skaleringen $(n-1)!$,

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

Et viktig moment er at dividerte differanser kan være basert på vektorer eller punkt. Det vil si at hvis $f(t_i) \in \mathbb{R}^d,\ d > 0$ så er alle tilknyttede dividerte differanser $f[t_i,\ldots,t_{i+k}] \in \mathbb{R}^d$.

5.2 Newtonpolynomene

Vi starter nå med førsteordens dividerte differanser. Gitt et sett med verdier $t_i \in I \subset \mathbb{R}$. Så for $t \in I$ men $t \neq t_0$ får vi

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

Det følger at verdien av f **for enhver** t kan skrives som

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

Vi fortsetter så med andre ordens dividerte differanser

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

Figur 5.1: *Kurven fra eksemplet med (5.7). Interpolasjonspunktene er markert.*

Snur vi dette får vi

$$f[t,t_0] = f[t_0,t_1] + (t-t_1)f[t,t_0,t_1]. \tag{5.6}$$

vi kan nå erstatte $f[t,t_0]$ i (5.4) med uttrykk (5.6). Så kan vi gjenta denne prosessen med dividerte differanser av orden høyere og høyere. Resultatet blir

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

En kompakt notasjon er

$$f(t) = \sum_{j=0}^{k} a_j n_j(t), \tag{5.7}$$

$$\text{hvor} \quad a_j = f[t_0,\ldots,t_j], \quad \text{og} \quad n_0(t) \equiv 1 \quad \text{og} \quad n_j(t) = \prod_{i=0}^{j-1}(t-t_i) \ \text{ når } \ j > 0.$$

Hvis vi ikke bruker en funksjonsnotasjon, men et sett med punkt knyttet til hver sin parameterverdi, dvs. $\{(t_i,p_i)\}$ får vi $a_j = [p_0,\ldots,p_j], \quad p_i \in \mathbb{R}^d, \quad i = 0,1,\ldots,j, \quad d > 1.$

Newtonpolynomer er laget for interpolasjon. For et gitt sett med distinkte parameterverdier $\{t_j\}_{j=0}^d$ og tilhørende punkt $\{p_j\}_{j=0}^d$ får vi et polynom som i hver parameterverdi t_j gir punktet p_j, dvs. at polynomet av lavest mulig grad interpolere alle punktene.

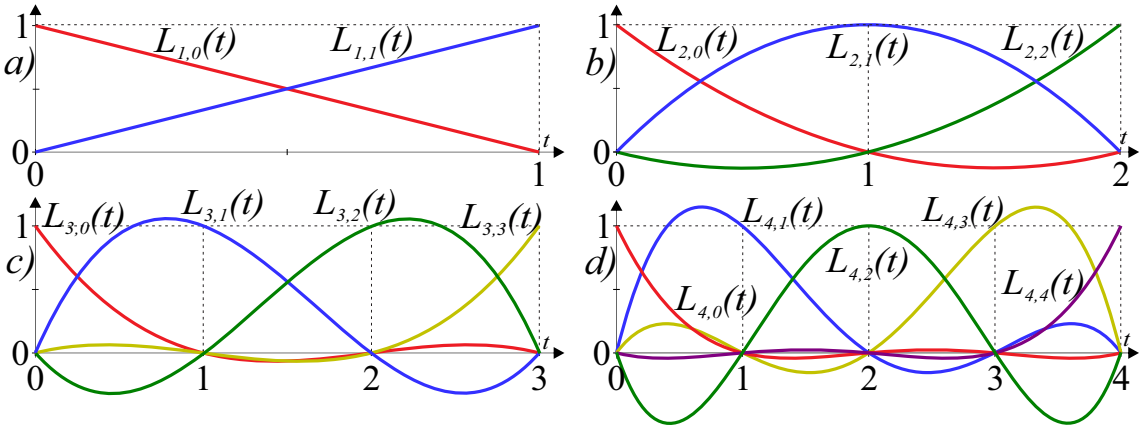
Som et eksempel kan vi bruke kurven i uttrykket (4.4) og som er plottet i figur 4.3. Parameterdomenet er $[0,1]$. Vi deler den i 3 og får, $t_0 = 0$, $t_1 = \frac{1}{3}$, $t_2 = \frac{2}{3}$ og $t_3 = 1$. Når vi bruker disse fire parameterverdiene i (4.4) får vi punktene $p_0 = (0,0)$, $p_1 = (\frac{11}{9}, -\frac{2}{9})$, $p_2 = (\frac{16}{9}, \frac{2}{9})$ og $p_3 = (3,0)$. Hvis vi beregner de dividerte differansene, får vi,

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

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

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

Punktene over markert med rødt er koeffisientene a_1, a_2 og a_3, og sammen med $a_0 = p_0$ og $t_0 = 0$, $t_1 = \frac{1}{3}$, $t_2 = \frac{2}{3}$ og $t_3 = 1$ definerer de en kurve med å bruke (5.7). Kurven er den samme som i figur 4.3 og er i figur 5.1 plottet sammen med interpolasjonspunktene.



Figur 5.2: *Et plott av fire sett av lagrangepolynomer (5.9).* **a)** *er polynomene av grad 1, som danner 1.-gradskurver.* **b)** *er polynomene av grad 2, som danner 2.-gradskurver.* **c)** *er polynomene av grad 3, som danner 3.-gradskurver.* **d)** *er polynomene av grad 4, som danner 4.-gradskurver. Alle lagrangepolynomer benevnes $L_{d,i}(t)$, der d er polynomgraden og i er indeksen til interpolasjonspunktet polynomet er koblet til.*

5.3 Lagrangepolynomene

En svært elegant alternativ representasjon av Newtons generelle formel, dvs. (5.7), som ikke krever bruk av dividerte differanser, ble publisert i 1779 av Waring [160]. Denne formelen er

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

Legg merke til mønsteret til indeksen i i t_i. En kompakt notasjon av formelen er,

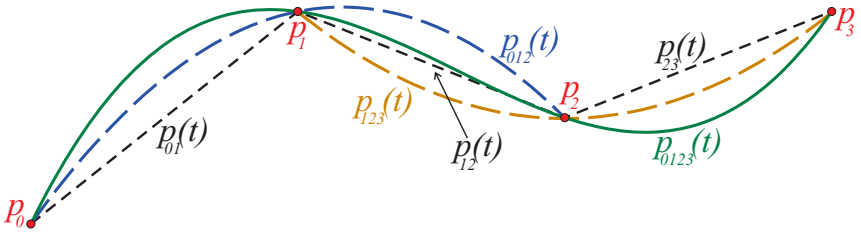
$$f(t) = \sum_{i=0}^{d} f(t_i)\, L_{d,i}(t), \tag{5.8}$$

der d er graden, og der basisfunksjonene, dvs. lagrangepolynomene er

$$L_{d,i}(t) = \prod_{j=0, j\neq i}^{d} \frac{(t-t_j)}{(t_i-t_j)}. \tag{5.9}$$

I figur 5.2 er alle lagrangepolynomer $L_{d,i}(t)$ opp til grad fire plottet. Merk at $L_{d,i}(t_i) = 1$ og $L_{d,i}(t_j) = 0, \quad j \neq i$. lagrangepolynomer[1] brukes til interpolering. Dvs. For et gitt sett

[1] I dag er det vanlig å kreditere disse polynomene til Lagrange, som imidlertid publiserte dem 16 år etter Waring [100]. I følge Euler [61] kan formelen (5.8) være knyttet til en relatert representasjon av Newtons formelen, og ifølge Joffe [94], var det Gauss som først la merke til sammenhengen mellom formlene til Newton, Euler og Waring-Lagrange, som det fremgår av hans posthume verk [72], skjønt Gauss refererte ikke til sine forgjengere.



Figur 5.3: *Fire punkt p_0, p_1, p_2 og p_3 er plottet i rødt. Tre 1.-gradskurver $p_{01}(t), p_{12}(t)$, og $p_{23}(t)$ interpolerer punktene og er plottet i stiplet svart. To 2.-gradskurver $p_{012}(t)$ og $p_{123}(t)$ er plottet i henholdsvis stiplet blått og stiplet oransje, og en 3.-gradskurve $p_{0123}(t)$ i helgrønt interpolerer alle fire punktene.*

med distinkte parameterverdier $\{t_j\}_{j=0}^d$ og et sett med korresponderende punkt $\{p_j\}_{j=0}^d$ kan vi generere lagrangepolynomer hvor vi med å bruke (5.8) lage en kurve som ved hver verdi t_j går igjennom det korresponderende punktet p_j.

Lemma 5.1. *Newton- og lagrangepolynomene er to forskjellige representasjoner av det samme polynomet.*

Beviset for lemma 5.1 finner vi i vedlegg C.1.

Selv om newton- og lagrangepolynomer egentlig er samme polynom har de ulike fordeler. Lagrangepolynomer er enklest og raskest å bruke fordi koeffisientene er interpolasjonspunktene direkte. Newtonpolynomene har imidlertid den fordelen at man kan legge til nye interpolasjonspunkt ved bare å legge til et nytt ledd.

Newton- og lagrangepolynomene viser oss også et mer grunnleggende resultat, nemlig at det bare er ett polynom av grad mindre eller lik n som interpolerer $n+1$ punkt ved $n+1$ gitte parameterverdier. Dette gir oss følgende teorem.

Teorem 5.1. *Hvis $x_0, x_1, \ldots, x_n$ er distinkte reelle tall, så vil det for et sett vilkårlige tall $y_0, y_1, \ldots, y_n$ være ett unikt polynom p_n av grad høyest n slik at $p_n(x_i) = y_i, \quad 0 \leq i \leq n$.*

Bevis. Anta at det finnes to slike polynom, p_n og q_n. Da ville polynomet $p_n - q_n$ ha egenskapen $(p_n - q_n)(x_i) = 0$ for $0 \leq i \leq n$. Siden graden av $p_n - q_n$ maksimalt kan være n, må dette polynomet ha maksimalt n nullpunkt hvis det ikke er 0-polynomet. Siden x_i er distinkte, bør $p_n - q_n$ ha $n+1$ nullpunkt; Dermed må det være 0. Følgelig er $p_n = q_n$. □

5.3.1 Neville's algoritme

Algoritmen er oppkalt etter den engelske matematikeren E.H. Neville (1889-1961). Den er en geometrisk tolkning av Newton/Lagrange polynomene. Vi tar (5.7), utvider det, omorganiserer det og introduserer Neville's notasjon av et interpolasjonspolynom $p_{\ldots}(t)$,

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

Denne prosedyren og er i hovedsak den samme som er brukt i beviset til Lemma 5.1 og kan utvides til høyere grad på samme måte. Dette betyr at vi nå har en prosedyre for interpolering av et punktsett $p_0, p_1, p_2, \ldots$ som faktisk gir newton- alternativt lagrangepolynomer. Det første trinnet er å finne,

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

og videre $p_{23},\ p_{34},\ldots$ som alle gir rette linjer fra p_0 til p_1, fra p_1 til p_2 og så videre. De neste nivået er

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

Hvor vi øverst har 2.-graderskurver som interpolerer p_0, p_1, p_2 og p_1, p_2, p_3, og nederst har en 3.-graderskurve som interpolerer alle 4 punktene. Videre kan vi lage flere nivå og dermed flere interpolasjonspunkt med den samme prosedyren.

I figur 5.3 ser vi et eksempel på interpolering av 4 punkt med Neville's algoritme. I boken *Pyramid Algorithms* [77] bruker forøvrig Ron Goldman Neville's algoritme som en av de grunnleggende pyramidealgoritmene.

5.4 Hermiteinterpolasjon

Newtons klassiske interpolasjonsformler (5.7) er også utgaangspunkt for den neste interpolasjonsmetoden. Hermiteinterpolasjon er å interpolere punkt med et gitt antall tilhørende deriverte. Resultatet av interpolasjonen er et polynom med en grad lik antall punkt + antall deriverte brukt i de samme punktene - 1 (kan være mindre hvis resultatet er et degenerert polynom). I hermiteinterpolasjon starter vi med å behandle deriverte som ekstrapunkt. I dividerte differanse-tabellen gjentas punktene tilsvarende antall deriverte. For å unngå divisjon med null, erstatter vi så uttrykket med deriverte i henhold til (5.3). For eksempel, hvis en verdi t_i gjentas 4 ganger, får vi

$$
f[t_i,t_i,t_i,t_i] = \frac{f^{(4-1)}(t_i)}{(4-1)!} = \frac{1}{6}f'''(t_i).
$$

NB, som nevn tidligere trenger ikke dividerte differanser å være en skalar men kan også være vektorer eller punkt.

I hermiteinterpolasjon starter vi med to punkt, $f(t_i)$ og $f(t_{i+1})$, som for eksempel er i rommet - $\mathbb{R}^3$, og 1.-deriverte i hvert av punktene $f'(t_i)$ og $f'(t_{i+1})$. Vi lager så et 3.-gradspolynom ved å interpolere de to punktene sammen med deres respektive 1.-deriverte ved å bruke (5.7)[2],

$$\begin{aligned} c_i(t) = f(t_i) &+ (t-t_i)\, f[t_i,t_i] \\ &+ (t-t_i)(t-t_i)\, f[t_i,t_i,t_{i+1}] \\ &+ (t-t_i)(t-t_i)(t-t_{i+1})\, f[t_i,t_i,t_{i+1},t_{i+1}]. \end{aligned} \tag{5.10}$$

Hvis vi nøster ut alle dividerte differanser, får vi

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

Siden $(t-t_{i+1}) = (t-t_i) + (t_i - t_{i+1})$ og vi bruker det som en erstatning for nevneren i brøkene, får vi

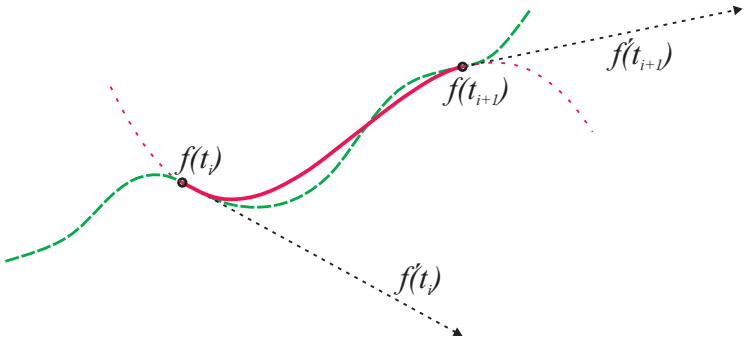
$$\begin{aligned} c_i(t) = f(t_i) &+ (t-t_i)\, f'(t_i) \\ &+ (t-t_i)^2\, \frac{2f'(t_i) + f'(t_{i+1}) - 3f[t_i,t_{i+1}]}{t_i - t_{i+1}} \\ &+ (t-t_i)^3\, \frac{f'(t_i) + f'(t_{i+1}) - 2f[t_i,t_{i+1}]}{(t_i - t_{i+1})^2}. \end{aligned} \tag{5.11}$$

Merk at ved å bruke newtonpolynomer kan vi, med formel (5.10),legge til deriverte til punktene og også mellom punktene (med noen begrensninger). I figur 5.4 er det et plott i stiplet grønt av en kurve $f(t)$ som er basert på en kombinasjon av trigonometriske og polynomiske uttrykk. I rødt er en 3.-grads kurve plottet. Den er laget med å interpolere to punkt på originalkurven og de respektive 1. deriverte i punktene. Vi ser at punktene burde vært nærmere hverandre for å gi en bedre approksimasjon.

Imidlertid kan (5.11) omformes til et mer praktisk og kjent format. La oss starte med å ekspandere formel (5.11),

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

[2]Grunnen til å bruke en indeks i ved kurven $c_i(t)$ i (5.10) vil bli forklart på side 69.



Figur 5.4: *En kurve $f(t)$ i stiplet grønnt og en 3.-grads polynomkurve (heltrukken rødt) som er en hermiteinterpolasjon av f ved t_i og t_{i+1} og med vektorene $f'(t_i)$ og $f'(t_{i+1})$.*

Så ordner vi uttrykket slik at det fremstår i en annen form

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

Dermed er de kubiske polynomene nå omformulert til det vi kaller en *geometriske form*:

$$c_i(t) = f(t_i)H_{i,3,0}(t) + f(t_{i+1})H_{i,3,1}(t) + f'(t_i)H_{i,3,2}(t) + f'(t_{i+1})H_{i,3,3}(t) \tag{5.12}$$

hvor

$$\begin{aligned} H_{i,3,0}(t) &= 2w_i(t)^3 - 3w_i(t)^2 + 1, \\ H_{i,3,1}(t) &= -2w_i(t)^3 + 3w_i(t)^2, \\ H_{i,3,2}(t) &= \Delta t_i\left(w_i(t)^3 - 2w_i(t)^2 + w_i(t)\right), \\ H_{i,3,3}(t) &= \Delta t_i\left(w_i(t)^3 - w_i(t)^2\right), \end{aligned} \tag{5.13}$$

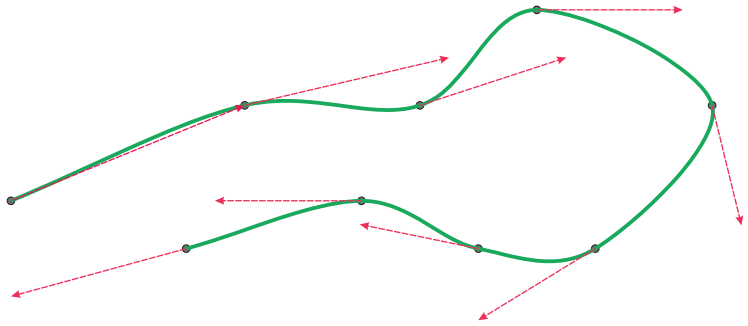
og hvor

$$w_i(t) = \frac{t-t_i}{t_{i+1}-t_i}, \quad \text{og} \quad \Delta t_i = t_{i+1}-t_i. \tag{5.14}$$

I (5.12) er nå start- og sluttpunktene og deres respektive 1.-deriverte koeffisientene. Basisfunksjonene, (5.13), er definert ved hjelp av en reparametrisering med en translering og skaleringsfunksjonen $w_i(t)$ (5.14), som vi senere vil finne med en ekstra indeks i definisjonen av B-splines, (6.11) i kapittel 6.3.

Sammenligner vi basisfunksjonene i (5.13) med hermitebasisfunksjonene i (4.19) på side 40 ser vi at de er di samme, med unntak reparametriseringa, dvs. at t er byttet med $w_i(t)$ og at de to siste basisfunksjonene er skalert med parameterintervallet.

Merknad 5.1. *Legg merke til at i et system med homogene koordinater, som vi finner i grafiske systemer som f.eks. OpenGL, vil både translasjon og rotasjon fungere godt for alle fire koeffisientene i (5.12) hvis de to første koeffisientene gjenkjennes som punkt og de to siste som vektorer.*



Figur 5.5: *En Catmull-Rom spline plottet som en grønn kurve. Den interpolerer 9 punkt som alle er markert som små sirkler. Tangentvektorene vises som røde piler.*

5.5 Taylor-ekspansjon

Dette er en slags hermiteinterpolasjon, men ble først introdusert som en rekke av Brook Taylor i 1715. Hvis vi bruker et 3.-grads polynom som eksempel og starter med Newton-polynomene (5.7), og deretter endrer formelen fra å interpolere fire forskjellige punkt til å interpolere det samme punktet fire ganger, dvs.

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

og hvis vi så erstatter dividerte differanser i henhold til (5.3) får vi

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

Uttrykket for Taylor-ekspansjonen på generelle “analytisk" funksjoner er

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

5.6 Hermitesplines

Formelen i (5.12) tar utgangspunkt i to punkt med tilhørende deriverte. Det er imidlertid mulig å utvidet den med flere punkt og deriverte over et større område og forhåpentlig genererer en god approksimerer av en originalkurve. Ulempen er imidlertid at resultatet er en polynombasert kurve av høy polynomgrad.

En annen måte å gjøre det på er å løsne på kravet om kontinuitet i interpolasjonspunktene og bare kreve en glatt kurve, dvs. C^1. Vi lager da separate 3.-grads kurver i hvert intervall mellom interpolasjonspunktene, men som da er limt sammen med at de har felles 1.-deriverte i skjøten, interpolasjonspunktet. Det er også mulig å bruke 5.-grads kurver og felles 1.- og 2.-deriverte i skjøten, jmf. (4.25) i kapittel 4.3. Dette ble tidligere kalt “oskulatorisk interpolasjon"[3] men i dag kalles det vanligvis hermitesplines.

[3] Gjentatt interpolasjon ved et punkt ble kalt “oskulatorisk interpolasjon" siden det gir høyere enn første ordens kontakt mellom funksjonen og dens interpolant, “osculari" betyr å kysse på latinsk, side 7 i [37].

En hermitesplines er en sekvens av kurver av polynomer med grader opp til d (oddetall) som er $C^{\frac{d-1}{2}}$-glatt over hele domenet. Kubiske hermitesplines som er C^1-glatt er meste vanlig. Det er n delkurver, $\{c_i(t)\}_{i=0}^{n-1}$, hvor hver delkurve er definert av (5.12). Det følger dermed at en spline er unikt definert av $n+1$ skjøtverdier $\{t_i\}_{i=0}^{n}$, samt $n+1$ punkt $\{f(t_i)\}_{i=0}^{n}$ og $n+1$ vektorer $\{f'(t_i)\}_{i=0}^{n}$, hvor normalt $n > 2$.

Noen ganger er ikke 1.-deriverte (tangenter/hastighetsvektorer) gitt eller tilgjengelig, derfor er det utviklet flere strategier for å lage glatte kurver uten gitte deriverte:

♦ Kardinal-splines[4] er en kubisk hermitesplines hvor tangentene/1.-deriverte i skjøtene er definert av vektoren mellom punktene før og etter det gjeldende punktet og skalert med en strammningsverdi. Dette gir en kurve mellom to punkt som tar hensyn til punktene før og etter, og som da gir en kurve som oppfører seg glatt og naturlig. Dvs. gitt n+1 punkt $p_0, ..., p_n$, som skal interpoleres med n kubiske kurvesegmenter, hvor hver kurve starter i p_i og slutter i p_{i+1} og med med starttangent v_i og slutttangent v_{i+1}, og hvor tangentene er:

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

Den første og den siste tangenten v_0 og v_n må enten være gitt, eller vi kan bruke $(t_1 - t_0)\, c\, (p_1 - p_0)$ og $(t_n - t_{n-1})\, c\, (p_n - p_{n-1})$. Videre er c en konstant som endrer lengden på tangentene (stramming). Det følger at $c > 0$.

♦ En Catmull-Rom splines (se [20]) er en kardinal-splines med strammingsparameter $c = 1$. Figur 5.5 viser et eksempel på en Catmull-Rom splinekurve laget av 9 punkt.

♦ En annen splines relatert til kubisk hermitesplines er “kubisk Bessel-spline" (se [37]). Her velger man vektorer/tangenter som er den deriverte i t_i av et 2.-grads-polynom som interpolerer i $t_{i-1},\ t_i,\ t_{i+1}$ med bruk av et newtonpolynom, dvs. andre ordens dividerte differanser (5.5). Dette gir tangentene

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

som viser at Bessel-interpolering også er en lokal metode.

♦ Enda en lokal metode er Akimas interpolasjonsmetode fra 1970 (se [2]). Akima valgte å beregne tangentene ved,

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

hvor

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

5.7 Kubisk splineinterpolasjon

Det er imidlertid en annen klassisk"interpoleringsmetode, som bruker stykkevise polynomer og som har en høyere grad av kontinuitet, nemlig kubisk splineinterpolasjon (se [37]).

[4] Schoenberg (se [141]) definerer Kardinal-splines som klassen av polynomiske splines på $\mathbb{R}$ med en uendelig skjøtvektor av etterfølgende heltallsverdier og hvor alle skjøtintervaller følgelig er 1.

Denne metoden er ikke lokal, så endringer har ikke bare en lokal innvirkning. Beregningskostnadene er imidlertid bare litt høyere enn for de lokale metodene fordi matrisen som skal løses er en båndsmatrise med kun 3 diagonale elementer.

Så hvis vi ønsker en C^2-glatt funksjon, kan vi fortsatt ta utgangspunkt i kubisk hermiteinterpolasjon, men vi avstår nå fra å spesifisere de 1.-deriverte i de indre skjøtene. I stedenfor gitte vektorer sier vi at:

$$c''_{i-1}(t_i) = c''_i(t_i) \quad \text{for} \quad i = 1,2,...,n-1, \tag{5.15}$$

og bruker så disse til å løse et system med lineære ligninger, dvs. invertere en $(n-1 \times n-1)$ matrise, for å finne de deriverte i de interne skjøtene, som er

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

For å finne ligningene for kubisk splineinterpolasjon tar vi (5.11) og deriverer den to ganger, merk at i (5.11)er alt unntatt $(t-t_i)^j$ konstanter. På to nabo-kurvesegment bruker vi så betingelsen (5.15) i skjøten mellom dem. Vi får da følgende matrise/vektor-ligning:

$$\mathbf{Ax} = \mathbf{b}, \tag{5.16}$$

dvs.

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

Her er

$$\lambda_i = \frac{\Delta t_i}{\Delta t_{i-1} + \Delta t_i}, \quad \text{og} \quad b_i = 3\beta_i - \delta_i, \tag{5.17}$$

hvor

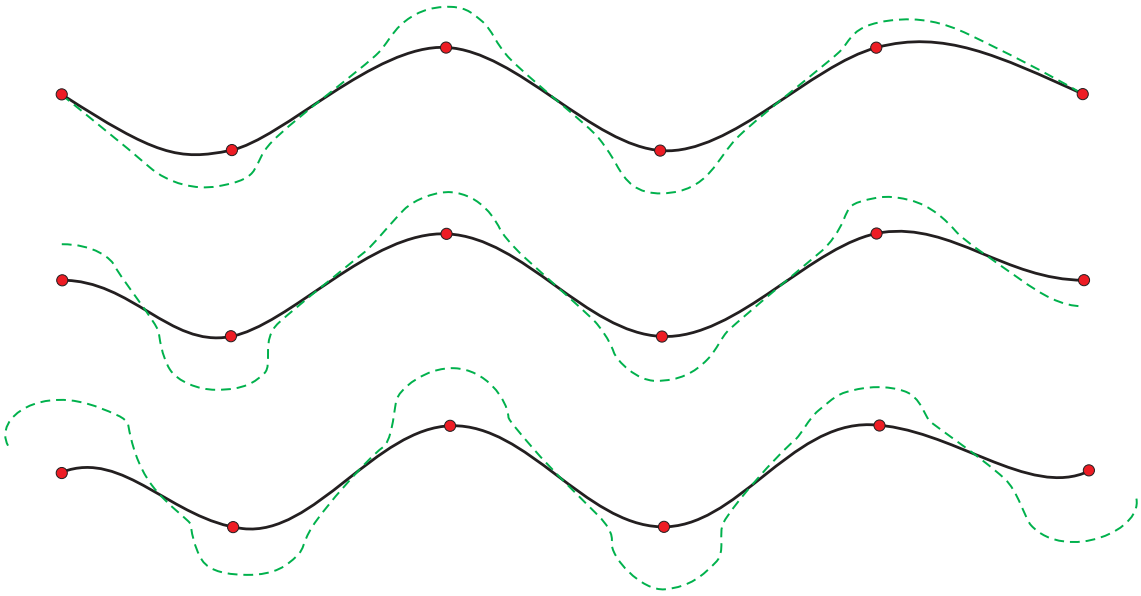
$$\beta_i = \lambda_i \, f[t_{i-1}, t_i] + (1-\lambda_i) \, f[t_i, t_{i+1}], \tag{5.18}$$

og

$$\delta_i = \begin{cases} \lambda_1 \, c'_0(t_0), & i = 1, \\ 0, & 1 < i < n-1, \\ (1-\lambda_{n-1}) \, c'_{n-1}(t_n), & i = n-1. \end{cases} \tag{5.19}$$

I (5.19) kan vi se at vi trenger å vite (eller estimere) tangentene ved starten og slutten av splinefunksjonen. Denne friheten fører til flere typer grensebetingelser. I [37] er flere mulige grensebetingelser nevnt.

En annen måte å organisere likningene på og som fører til en annen type grensebetingelse er å angi den 2.-deriverte i endene. Ligningen er den samme som den i (5.16), men dimensjonen til matrisen vil nå være $n+1 \times n+1$. Start- og slutt-tangentene, og 2. deriverte, er



Figur 5.6: *Fri-ende betingelsen er brukt i det øverste eksempelet, det vil si at den 2.-deriverte er null ved start og slutt, dette kan tydelig sees på figuren. Horisontale tangentvektorer er brukt som endebetingelser er gitt i det midterste eksemplet. Tangentvektorer som begge er* 45° *opp til høyre er brukt som endebetingelser i det nedre eksemplet.*

nå inkludert i en nye første og en ny siste linje i matrisen (beregnet fra en andrederivert av (5.11). Vi får nå

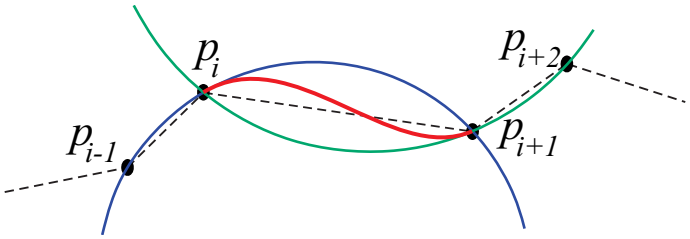
$$\begin{pmatrix} 2 & 1 & 0 & \cdots & \cdots & 0 \\ \lambda_1 & 2 & 1-\lambda_1 & \ddots & \ddots & \vdots \\ 0 & \lambda_2 & 2 & 1-\lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \lambda_{n-1} & 2 & 1-\lambda_{n-1} \\ 0 & \cdots & \cdots &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

Her er λ definert i (5.17) og β er definert i (5.18), men nå er

$$b_i = \begin{cases} 3f[t_0,t_1] - \frac{\Delta t_0}{2} c_0''(t_0), & i = 0, \\ 3\beta_i, & 0 < i < n, \\ 3f[t_{n-1},t_n] - \frac{\Delta t_{n-1}}{2} c_{n-1}''(t_n), & i = n. \end{cases} \tag{5.20}$$

Hvis, i (5.20), $c_0''(t_0) = c_{n-1}''(t_n) = 0$ (eller en null-vektor) har vi en såkalte *fri-ende betingelse*, og splinekurven er en "naturlig splines" med en oppførsel som er ganske lik en ekte fleksibel stav.[5] I figur 5.6 er det et eksempel på kubisk spline-interpolering med frie ender (den øverste kurven). Som vi ser i figuren er krumningen null i endene. I de andre to

[5]For å tegne kurver, spesielt i skipsbygging, brukte tegneren ofte lange, tynne, fleksible strimler av tre,



Figur 5.7: *Den røde kurven vi kan se mellom punktet p_i og p_{i+1} er et segment av en sirkelsplinekurve. Mellom punktet p_i og p_{i+1} er deler av to sirkelbuer, en grønn og en blå, blendet til det røde kurvesegmentet. Som vi ser er det røde kurvesegmentet mellom punktene p_i og p_{i+1} entydig definert av de fire punktene: p_{i-1}, p_i, p_{i+1} og p_{i+2}..*

eksemplene er endebetingelsene gitte tangenter ved start og slutt. Det faktum at energien (eller tilnærmingen og forenklingen ved å bruke $c''(t)$ i stedet for krumning) minimeres av en tredjegrads splinefunksjon (i Sobolev rommet $H^2(a,b)$), dvs. min $\int_a^b |c''(t)|^2 dt$, er på en måte illustrert i figuren. Vi kan faktisk se fra plottene i figur 5.6 at fri-ende betingelsen gir “den minste krumningen".

I følge Schumaker [142] la Euler og Bernoulli-brødrene allerede på midten av 1700-tallet merke til at formen til en fleksibel stav blei svært godt beskrevet av 3.-grads-polynomer.

5.8 Sirkelsplines

I 1996 publiserte Hans-Jörg Wenz [164] en interpolasjonsmetode basert på å blende sirkelbuer. Gitt en sekvens av punkt $\{p_i\}_{i=0}^n$. I hvert punkt p_i, $i = 1,2,\ldots,n-1$ (ikke endepunktene) finnes det en sirkelbue $\vartheta_i(u)$, $u \in [0,1]_{i-1} + [0,1]_i$, som starter i p_{i-1}, interpolerer p_i og slutter i p_{i+1}, men som da er parametrisert fra 0 til 1 separat i hver av segmentene $[p_{i-1}, p_i]$ og $[p_i, p_{i+1}]$. For å lage et glatt kurvesegment mellom to punkt p_i og p_{i+1}, $i = 0,1,\ldots,n-1$, brukte Wench et svært enkelt blendingsskjema,

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

Merk at til tross for at han brukte en lineær blendingsfunksjon, er kurven G^1-glatt fordi hver bue interpolerer tre punkt og den utvidede hermiteinterpolasjonsegenskapen, som vil bli forklart i teorem 7.2 (på side 119) forteller oss at dette øker kontinuiteten med én.

Wenz ble fulgt av Márta Szilvási-Nagy og Teréz P. Vendel, [154]. De forbedret blendingsskjemaet ved å erstatte den lineære blendingsfunksjonen med en trigonometrisk blendingsfunksjon $b(t) = \sin^2\left(\frac{\pi t}{2}\right)$ og som gir en G^2-glatt kurve

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

plast, eller metall kalt en rei i Nord-Norge (spline på engelsk). Den fleksible staven ble holdt på plass med blyvekter. Elastisiteten til materialet kombinert med begrensningen som kontrollpunktene (skjøtene) medførte, får stanga til å ta en form som minimerer energien som kreves for å bøye den mellom de låste punktene, og dermed anta “en glattest mulig" form (ifølge elastisitetsteori).

Figur 5.7 viser et eksempel på å konstruere en sirkelsplinekurve ved å blande to sirkelbuer. I eksempelet er det den trigonometrisk blendingsfunksjonen som er brukt.

For å unngå løkker og cusps introduserte Carlo Séquin, Kiha Lee og Jane Yen en annen blendingsteknikk i [147, 146]. De bruker i utgangspunktet en trigonometrisk blendingsfunksjon. De bruker den imidlertid ikke direkte slik som Nagy og Vendel, men til å beregnet retningsvinkelen $\tau(t)$ fra τ_i og τ_{i+1},

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

hvor vinkelen τ_i er vinkelen mellom linjestykket $[p_i, p_{i+1}]$ og tangenten $\vartheta_i'(0)$, dvs. tangenten i starten av den første sirkelbuen. De tenker seg så et plan som går igjennom linjestykket $[p_i, p_{i+1}]$. Dette planet roterer fra også å gå gjennom $[p_{i-1}, p_i]$ til også å gå igjennom $[p_{i+1}, p_{i+2}]$. Rotasjonen er da gjort med hjelp av den trigonometrisk blendingsfunksjon. De beregner deretter posisjon på kurven $c_i(u)$ ved å beregne vinkelen ϕ mellom linjestykket $[p_i, p_{i+1}]$ og linjestykket $[p_i, c_i(t)]$ i det roterte planet,

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

og distansen $|c_i(t) - p_i|$ ved

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

Kapittel 6

B-splinekurver

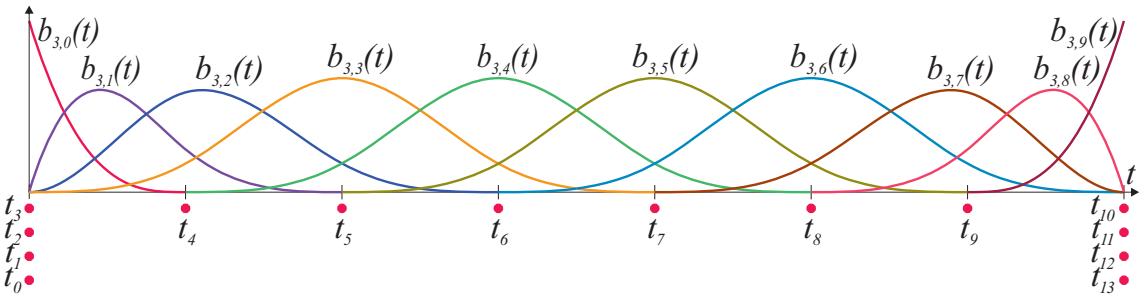
I dag er splinekurver synonymt med B-splinekurver. B-splines er de facto industristandarden for modellering i datastøttet design. B-splines er en måte å representere polynomiske splines på. Den kobler polynomiske splines til hjørnekuttingsteknikker, som igjen induserer mange nyttige egenskaper. I dette kapittelet skal vi ta et dypdykk inn i polynomiske B-splines, men vi starter med en oversikt for å gi en idé om hva B-splines er.

B-splines er i utgangspunktet et sett med “lokale" basisfunksjoner som sammen summerer til 1 over hele domenet. Hver basisfunksjon er koblet til ett kontrollpunkt. Formelen er,

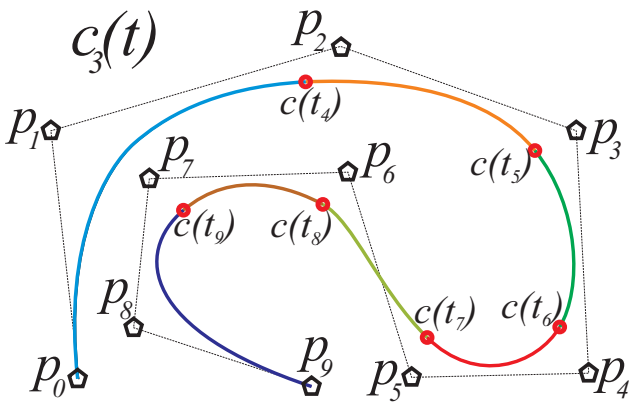
$$c(t) = \sum_{i=0}^{n-1} c_i \, b_{d,i}(t), \quad \text{hvor} \quad \sum_{i=0}^{n-1} b_{d,i}(t) = 1, \quad \text{når} \quad t_d \leq t \leq t_n,$$

og der c_i er n punkt som sammen danner et kontrollpolygon, og der $b_{d,i}(t)$ er basisfunksjonene, d er polynomgraden og $\tau = \{t_i\}_{i=0}^{n+d}$ er en vektor av skjøtverdier. Basisfunksjonene danner sammen et splinerom, og er definert av en sekvens av skjøtverdier og en polynomgrad. Domenet er begrenset til $[t_d, t_n]$ fordi det er der basisfunksjonene summerer opp til 1. En viktig egenskap er at B-splines er C^{d-1}-kontinuerlig over ikke-multiple skjøtverdier.

Vi skal se på et eksempel. Gitt en skjøtvektor $\tau = \{0,0,0,0,1,2,3,4,5,6,7,7,7,7\}$. Sammen med graden $d = 3$ genererer den 10 basisfunksjoner som er vist i figur 6.1. I figuren ser vi at det er 7 intervall mellom skjøtene. Inne i hvert intervall er det 4 aktive basisfunksjoner. Vi introduserer så et sett med 10 kontrollpunkt $p_i,\ i = 0,1,\ldots,9$.



Figur 6.1: *Et sett av 10 3.-grads B-spline (basis) funksjoner $b_{3,i}(t),\ i = 0,1,\ldots,9$, definert av en vektor med 14 skjøtverdier, alle merket som røde kuler.*



Figur 6.2: *Vi ser en 3.-grads B-spline kurve der de 7 delene har forskjellige farger. Kontrollpunkt og polygon er markert. De røde kulene markerer posisjonen til skjøtverdier.*

Resultatet vises i figur 6.2. Vi ser de 10 punktene p_i markert som svarte femkanter, vi ser det stiplede svarte kontrollpolygonet og den C^2-glatte splinekurven $c_3(t)$. Hver del av kurven er plottet i forskjellige farger. De røde kulene er der parameterverdien er lik en av skjøtverdiene t_i, $i = 4,5,\ldots,9$. Hver av de 7 delene er en polynomkurve og vi kan finne formelen på bézier-formatet. På grunn av det lokale parameterdomenet til basisfunksjonene $b_{3,2}(t)$ ser vi fra figur 6.1 at hvis vi flytte p_2 vil bare den blå, den oransje og den grønne kurven i figur 6.2 bli påvirket.

6.1 Historien om B-splines

Vi går nå mer enn et århundre bak i tid. Behovet for bedre og glattere interpolanter i noen applikasjoner førte på slutten av 1800-tallet til utviklingen av interpolasjonsformler basert på stykkevise polynomer, såkalte *osculatoriske interpolasjonsteknikker* , hvorav de fleste dukket opp i aktuarlitteraturen, Greville [81]). Et eksempel på dette er formelen foreslått i 1899 av Karup [96]. Formelen resulterer i en stykkevis 3.-grads polynominterpolant som er kontinuerlig, og kontinuerlig differensierbar, overalt. Ved å bruke denne formelen er det mulig å reprodusere polynomer opp til 2. grad. Et annet eksempel er formelen foreslått i 1906 av Henderson [89] og som også gir en kontinuerlig differensierbar, stykkevis 3.-grads polynominterpolant og som er i stand til å reprodusere polynomer opp til 3. grad. Et tredje eksempel er formelen publisert i 1927 av Jenkins [93]. Den resulterende kompositttkurven er stykkevis 3.-gradspolynomer som er to ganger kontinuerlig differensierbar. Denne kurven er imidlertid ikke en interpolant. (Alle disse formlene finnes i [81] og [125].)

Behovet for praktisk anvendelige metoder for interpolasjon eller glatting av empiriske data var drivkraften til Schoenbergs første studie av emnet. I artikklene hans fra 1946 [138, 139] bemerket han at for hver *oskulatorisk interpolasjonsformel* brukt på ekvidistante data, hvor han antok at avstanden var 1, eksisterer det en symetrisk (om y-aksen) funksjon $\Phi : \mathbb{R} \to \mathbb{R}$, som gjør at en formel kan skrives som

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

hvor Φ, som han kalte *basisfunksjon*, bestemmer fullstendig egenskapene til den resulterende interpolanten og avslører seg selv når den første formelen brukes på impulssekvensen definert av $a_0 = 1$ og $a_k = 0,\ \forall k \neq 0$. I analogi med Whittakers kardinalserie [165], viste Schoenberg til det generelle uttrykket (6.1) som en formel av kardinaltypen, men bemerket at den grunnleggende funksjonen til kardinalserien, $\Phi(x) = sin(\pi x)/(\pi x)$, er utilstrekkelig for numeriske formål på grunn av sin lav dempningsrate. Basisfunksjonene involvert i Waring-Lagrange interpolasjon, derimot, har begrensende egenskap ved å være høyst kontinuerlig, men ikke kontinuerlig differensierbar. Han pekte deretter på de glatte kurvene som ble oppnådd ved bruk av en mekanisk spline,[1] og hevdet at disse er stykkevise kubikk-kurver med kontinuerlige 1.- og 2.-deriverte, og fortsatte deretter med å introdusere forestillingen om en matematisk spline:

Definisjon 6.1. *En reell funksjon f definert for alle reelle tall x kalles en splinefunksjon (kurve) av orden k og betegnet med S_k hvis den har følgende egenskaper:*

i) den er satt sammen av polynom-kurvestykker av grad høyest $d = k-1$,
ii) den er C^{k-2}-glatt, dvs. at f har $k-2$ kontinuerlige deriverte,
iii) de eneste mulige skjøtverdier til de forskjellige polynom-kurvestykkene er $x = k$ hvis k er partall, ellers hvis k er oddetall $x = k + \frac{1}{2}$.

Merk at disse kravene tilfredsstilles av kurvene fra den tidligere nevnte formelen foreslått av Jenkins, og også studert av Schoenberg [138], og som utgjør et av de tidligste eksemplene på en splinegenererende formel. Merk også at hermitesplines og dens avledede splines ikke oppfyller alle kravene.

Etter å ha definert splinekurven, fortsatte Schoenberg å bevise at enhver splinekurve S_k kan representeres unikt på formen

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

for passende verdier av koeffisientene a_j. Det finnes ingen problemer med konvergens siden, som vi vil se nedenfor, $M_k(x)$ er 0 for $|x| > k/2$. Dermed er (6.2) den mest generelle representasjon av S_k for alle mulige sett av $\{a_j\}$.

Her betegner $M_k : \mathbb{R} \to \mathbb{R}$ en såkalt sentral *B-spline* av grad $d = k-1$, som Schoenberg definerer som det inverse Fourier-integralet

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

og, når den blir gjort eksplisitt blir[2]

[1] Bruken av ordet “spline" for en fleksibel stang kan spores tilbake til skipsbygging i det 18. århundre. opprinnelig var det et øst-anglisk dialektord. Fleksible stenger (spline) har blitt brukt av skipsbyggere overalt i svært lang tid. I Norge er det faktisk 3 varianter av navnet, avhengig av hvor verftet ligger. I nord (hvor denne forfatteren har sin erfaring) kalles det “rei", uttalt og relatert til engelsk ray, som i “ray-tracing".

[2] Scoenberg bemerket at (6.3) hadde blitt vurdert eksplisitt for lave verdier av k av forskjellige forfattere, blant annet S. Bochner i forelesninger om Fourier-analyse i 1936. I en artikkel av Butzer, Schmidt og Stark [18] er flere tidligere arbeid om B-splines nevnt som sannsynligvis ikke er kjent av Scoenberg. Blant disse er Maurer i 1896 [121] (diskuterer en funksjon relatert til den sentrale B-splinefunksjonen til Schoenberg), Sommerfeld i 1904 og 1929 [150] (gjør en geometrisk tolkning, en Box-spline-lignende metode for å konstruere B-splines) og Brun i 1932 [17] utvikler en rekursiv hjørnekuttingsmetode som de Casteljau.

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

der δ^p er p.-ordens sentrale differensoperator [3] og x_+^n angir den en-sidige potensfunksjonen definert som

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

Rett etter at disse første artikklene [138, 139] ble skrevet, ble mulighetene for en utvidelse til ikke uniforme skjøtverdier antydet av Curry i hans revju, [31], av Schoenbergs artikkel. Og i et abstrakt, [32], ble B-splines[4] for vilkårlige skjøter introdusert. Hele artikkelen ble skrevet i 1946-47, men ble først publisert 20 år senere [33]. Curry og Schoenbergs definisjon av B-splines er som følger,

Definisjon 6.2. *En B-spline benevnt som $M_k(x)$ er definert av $k+1$ økende reelle tall $x_0,\ldots,x_k$, der $x_0 < x_k$, dets eksplisitte uttrykk er*

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

hvor

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

Curry og Schoenberg slo fast følgende egenskaper for B-splines:

1) Det viser seg at de er klokkeformede.
2) De er også projeksjonene på x-aksen av volumene til passende n-dimensjonale simplekser.
3) Ved hjelp av Brunn´s teorem ser vi at en geometriske tolkningen gir konklusjonen - at B-splinefunksjonen er logisk konkav.
4) Vi ser at de danner en basis for alle splinefunksjoner av grad $n-1$ med gitte skjøtverdier.
5) De kan defineres med multiple skjøter.
6) B-splines $M_k(x)$ (slik den er definerti (6.4)) er frekvensfunksjoner, som betyr at de er ikke-negative og deres integral over $\mathbb{R}$ er 1.[5]

I figur 6.3 er det en kopi av en skisse fra Curry og Schoenberg, [33], av seks kvadratiske B-splines, dvs. når $(k=3)$, som er definert av 9 skjøtverdier, og hvor noen av dem er like. De seks B-splinefunksjonene er:

[3] Ofte kallt Sheppard's sentrale differensoperator δ (se [148]), definert av

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

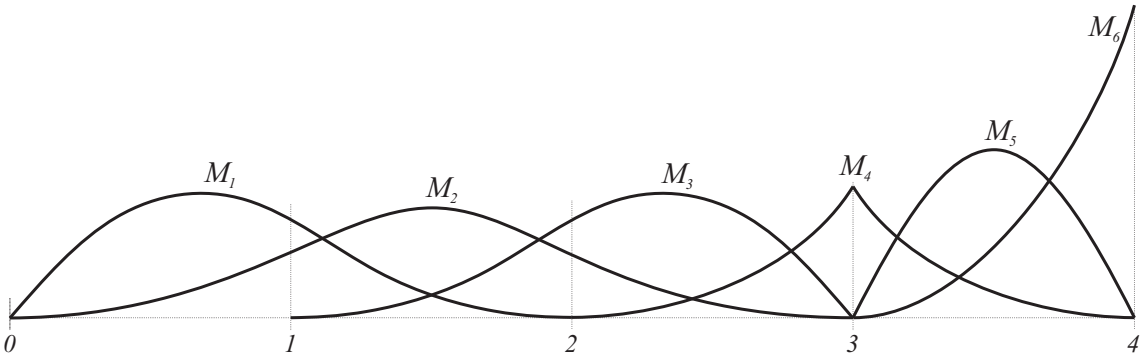
og

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

for enhver funksjon $\phi : \mathbb{R}\to\mathbb{R}$ til enhver ξ. Må ikke sammenblandes med høyresidig differensoperator.

[4] Curry og Schoenberg kalte dem faktisk først de fundamentale splinefunksjonene. Navnet B-spline refererer til basis-splines fordi de danner en basis for spline-funksjoner, noe som ble bevist av Curry og Schoenberg i [32]. I [140], publisert i 1967, brukte Schoenberg navnet B-spline.

[5] NB! Moderne B-splines skaleres på en annen måte fordi i CAGD er partisjon av enheten (at de summerer opp til 1) en mye viktigere egenskap. Dette fordi affine avbildninger da kan brukes, se side 16



Figur 6.3: *En kopi av en skisse fra Curry og Schoenberg [33] av seks kvadratiske B-splines definert av 9 skjøverdier;* $x_1 = x_2 = 0,\ x_3 = 1,\ x_4 = 2,\ x_5 = x_6 = 3$ *og* $x_7 = x_8 = x_9 = 4$.

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

Den klassiske utvidede definisjonen av dividerte differanser, (5.1), sier følgende

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

hvor $\omega(x)$ er definert i (5.2), og er den samme som i (6.5).

Uttrykk (6.6) er på en måte lik (6.4), bortsett fra at i (6.4) erstatter funksjonen $(x_\nu - x)_+^{k-1}$ skalarene i telleren i (6.6). Vi må derfor klargjøre notasjonen. En dividert differanse-notasjon av en B-spline, $M_k(x)$, er derfor

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

Notasjonen kalles en “plassholder"-notasjon og en mer detaljert beskrivelse av denne notasjonen finnes på side 108 i [37].

Dividerte differanser er definert rekursive, så det er derfor naturlig å prøve å utvikle en enkel rekursiv algoritme for å regne ut B-splines. Dette ble gjort i 1972 av minst tre forskjellige forfattere omtrent samtidig. Cox [29] laget det for enkle skjøter. Resultatet for generelle skjøtverdier krediteres deBoor [36]. I sin artikkel nevnte han at Louis Mansfield også hadde oppdaget rekursjonen. I samme artikkel introduserte deBoor også derivasjonsformelen for B-splines. Noen andre viktige algoritmer i splinehistorien er; skjøtinnsetting introdusert samtidig i 1980 av Boehm [15] for enkeltskjøter og av Cohen, Lyche og Riesenfield [23] for generell skjøtinnsetting, gradheving som kom i 1985 av Cohen, Lyche og Schomaker [24]. Det bør selvfølgelig nevnes at Paul de Casteljau i 1959 utviklet en algoritme for beregning av bézierkurver (men den ble kun publisert i en intern Citroë-rapport [38]) og at Pierre Étienne Bézier i 1966 publiserte konstruksjonen av bézierkurver/flater [12, 13]. Bézierkurver er B-splinekurvene uten indre skjøter.

Som en siste historisk bemerkning i denne seksjonen kan den nevnes at ifølge Farin [64] undersøkte N. Lobachevsky så tidlig som i det nittende århundre B-splines som konvolusjoner av visse sannsynlighetsfordelinger (over en svært spesiell skjøtsekvens). Også

Peano´s kjerne for 3.-differanse gir en 2.-grads B-spline, som det er et plot av på side 73 i Davis-bok fra 1963, [35]. Navnet spline refererer til en mekanisk fleksibel stav som minimerer energien når den bøyes. Dette kan formuleres som et uttrykk som minimere kvadratet av krumningen, dvs.

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

En kubisk splinefunksjon tilfredsstiller nesten formelen, men hvor den andre deriverte erstatter krumningen. Dette er selvfølgelig ikke det samme som (6.8), men hvis hastigheten er nær 1 over hele kurven, er dette en ganske god tilnærming. Det finnes arbeider for å finne en bedre tilnærminger, f.eks. Mehlum i [123, 124]. Hvordan man bruker begrepet spline er derfor ikke åpenbart. Det burde muligens på en eller annen måte være en tilnærming til en fleksibel stav (spline-enhet), enten det er en optimal eller ikke optimal løsning i henhold til en funksjonal. I dag er det mest vanlig å tenke på splines som stykker/deler som skjøtes sammen til én kurve eller flate med kontrollert glatthet.

Dette fører oss til moderne normalisert B-splines. I neste seksjon skal vi se nærmere på definisjoner, notasjonen og algoritmene som er viktige for B-splines slik den brukes idag.

6.2 Moderne B-splines

Moderne B-splines har flere uttrykk. Det er for eksempel blomstring som ble introdusert av Ramshaw [134], og i en annen form av de Casteljau [41]. Han kalte det polar form. I denne seksjonen skal vi først se på B-splines på det vi nå kan kalle den “klassiske måten", og vi skal se på noen eksempler og til slutt skal vi se på B-splines-algoritmer. Men for ikke å gå oss vill i indekser, skal vi noen ganger bruke matrise/vektor-formuleringer.

Det er flere forskjellige B-spline-notasjoner i bruk. De vanligste er,

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

der d er polynomgraden, $k = d+1$ er ordenen, dimensjonen til funksjonsrommet. B-splines er her synonymt med normaliserte B-splines, som betyr at alle B-splines definert på en uendelig skjøtsekvens summerer opp til 1 (danner partisjonen av enhet). I praksis betyr det.

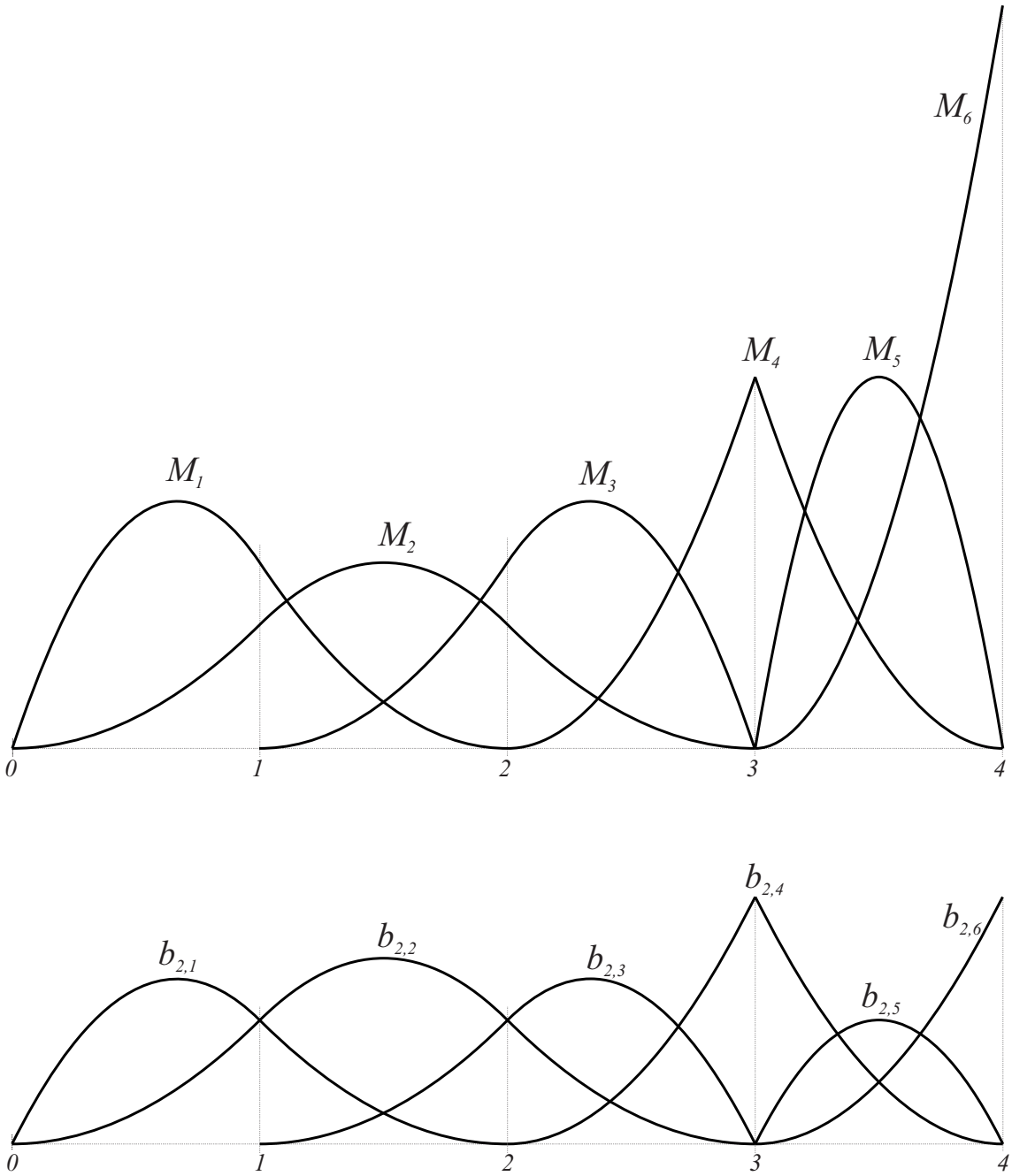
$$\sum_{i=j-k}^{j} b(t;t_i,\ldots,t_{i+k}) = 1, \quad t_j \le t < t_{j+1}. \tag{6.9}$$

Sammenlignet med de originale Curry-Schoenberg B-splines $M(t;t_i,\ldots,t_{i+k})$ har vi dermed følgende forhold (en visuelt sammenligne kan sees i figur 6.4)

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

Merk at for ikke multiple heltallsskjøter, dvs. (1,2,...) er $b(t;t_i,\ldots,t_{i+k}) = M(t;t_i,\ldots,t_{i+k})$.

Dette fører til følgende definisjon av B-splines, som vanligvis kalles Cox-de‘Boor rekursjon-formelen, men er noen ganger også kalt Mansfield-Cox-de‘Boor:



Figur 6.4: *Vi ser to plott av seks 2.-grads B-splines som begge er definert av de 9 skjøtverdiene som er brukt i figur 6.3;* $t_1 = t_2 = 0$, $t_3 = 1$, $t_4 = 2$, $t_5 = t_6 = 3$ *og* $t_7 = t_8 = t_9 = 4$. *Den øverste figuren er de originale B-splines fra Curry og Schoenberg [33], skissert i figur 6.3 og hvor integralet av hver basisfunksjon (arealet) er 1, dvs. de er frekvensfunksjoner. De nederste er moderne B-splines som danner partisjonen av enhet, det vil si at de summerer opp til 1 der det er et komplett sett med basisfunksjoner, dvs. at k basisfunksjoner er forskjellige fra null, der k er ordenen (dimensjonen til funksjonsrommet). Skalaen er lik for begge plottene. Merk at* $M_2 = b_{2,2}$ *fordi skjøtene er* $\{0,1,2,3\}$, *dvs. ikke-multiple heltallsverdier.*

Definisjon 6.3. *For en gitt grad d, orden $k = d+1$, og $k+1$ reelle og økende tall $\{t_i, t_{i+1}, \ldots, t_{i+k}\}$ der $t_{i+k} > t_i$ (de trenger ellers ikke være strengt økende), er en B-spline av grad d definert av rekursjonsformelen:*

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

hvor termineringen av rekursjonen er

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

og hvor den lineære translering og skaleringsfunksjonen er

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

Under følger en liste over de grunnleggende egenskapene til $b(t;t_i,\ldots,t_{i+k})$.

P1. Hver basisfunksjon $b(t;t_i,\ldots,t_{i+k})$ er positiv på (t_i,t_{i+k}) og null ellers.

P2. Settet med basisfunksjonene $b(t;t_i,\ldots,t_{i+k})$ for $i = j-d,\ldots,j$ danner en partisjon av enhet på $[t_j,t_{j+1})$, dvs.

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

P3. En B-spline $b(t;t_i,\ldots,t_{i+k})$ er $C^r(\mathbb{R})$ hvor $r = d-s$ og s er maksimal multiplisitet i skjøtene $t_i,\ldots,t_{i+k}$, dvs. maksimalt antall av like skjøtverdier.

P4. En B-spline $b(t;t_i,\ldots,t_{i+k})$ er, ved enkle skjøtverdier "klokkeformet", dvs. den v^{te}-deriverte $b^{(v)}(t;t_i,\ldots,t_{i+k})$ (for $v = 1$ og opp til $v = d-1$) har nøyaktig v nullpunkt på intervallet (t_i,t_{i+k}). Ved enkle skjøter er alle disse nullpunktene forskjellige.

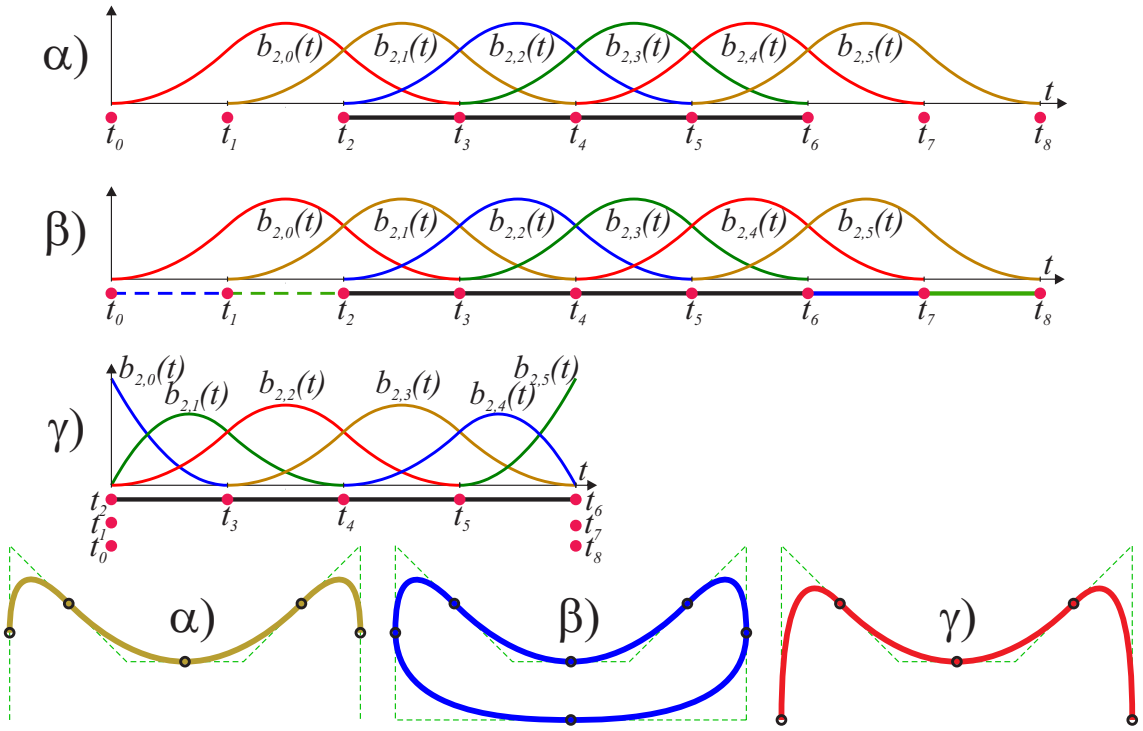
6.2.1 Skjøtvektorer

En skjøtvektor er et sett med reelle tall med økende verdier. Intervallet mellom to skjøteverdiene beskriver domenet for hver polynomfunksjon. Disse funksjonene skjøtes sammen i skjøtverdiene. Dette er vist i figur 6.2 og 6.7, der de røde sirklene markerer $c(t_i)$, altså hvor på kurven skjøtene er, og dermed hvor to kurvedeler er limt sammen. Antallet like skjøtverdier (multiplisiteten) forteller graden av kontinuitet over skjøten.

En skjøtvektor

er et sett med økende skjøtverdier som sammen med en polynomgrad d definerer et sett av B-splines/basisfunksjoner, dvs. et splinerom. I [6.9] er kravet om at summen av basisfunksjonene skal være 1 gitt (partisjon av enheten). Det følger dermed at det må være $d+1$ aktive basisfunksjoner i alle skjøtintervall. Det betyr at; for en gitt skjøtvektor $\tau = \{t_0,t_1,\ldots,t_{n+d}\}$ er derfor domenet til enhver B-splinefunksjon/kurve av grad d, som er basert på denne skjøtvektoren, avgrenset til $[t_d,t_n]$.

En B-spline funksjon/kurve kan ha tre tilstander koblet til skjøtvektoren, åpen, åpen og klemt eller lukket og syklisk. Disse vil bli beskrevet mer detaljert i neste seksjon.



Figur 6.5: *Tre 2.-grads B-splinekurver er plottet, α) er åpen, β) er lukket og syklisk og γ) er åpen og klemt. Også B-splines (basisfunksjonene) og skjøtvektorene (røde prikker) er plottet, og domenene er markert som heltrukne svarte linjer gjennom de røde kulene.*

6.2.2 B-splinekurver - Åpen, klemt eller lukket

B-splinekurver er en generalisering av bézierkurver. Vi skal i det kommende kalle en B-splinekurve for $c(t)$. Den er definert av en polynomgrad d, orden $k = d+1$, og av en skjøtvektor $\tau = \{t_0, t_1, ..., t_{n+d}\}$ og n kontrollpunkt. Den generelle formelen er

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

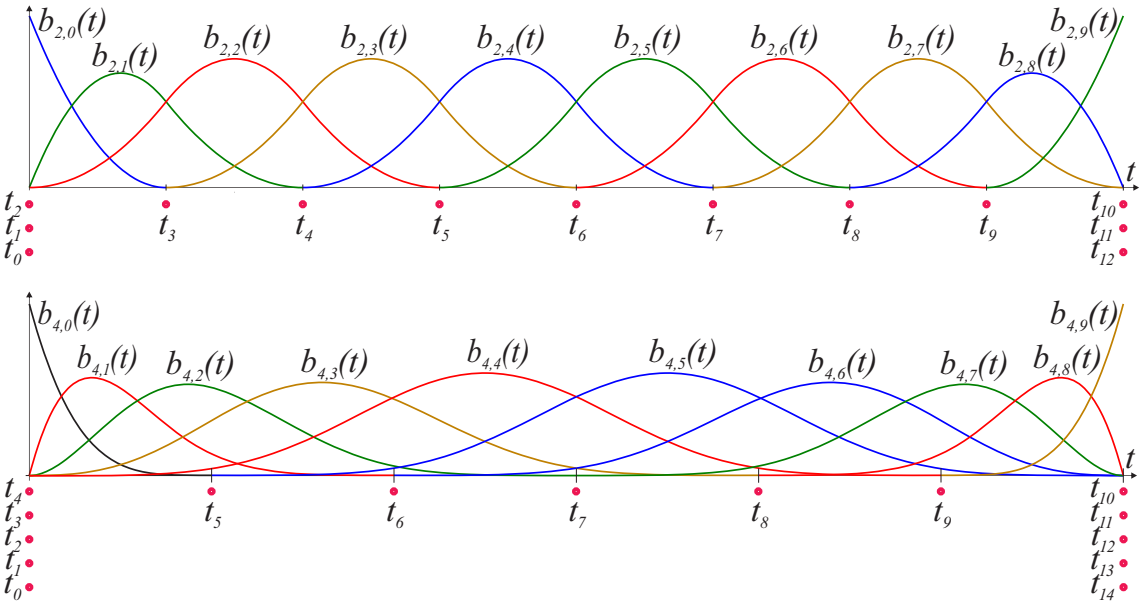
hvor $\{c_i\}_{i=0}^{n-1}$, er en vektor av koeffisienter/kontrollpunkt. Disse punktene danner kontrollpolygonet til kurven. $b_{k,i}(t)$ er settet med B-splines (basisfunksjoner) definert av skjøtvektoren τ, som har $n+k$ skjøtverdier.

Som nevnt kan en B-spline kurve ha tre tilstander, åpen, åpen/klemt eller lukket/syklisk. Hvis en kurve har n basisfunksjoner og da n kontrollpunkt, da er:

Åpen - en kurve som er basert på en ordinært skjøtvektor $\tau = \{t_0, t_1, ..., t_{n+d}\}$ og hvor domenet til kurven er begrenset til $[t_d, t_n]$.

Åpen og klemt - en åpen kurve med en skjøtvektor $\tau = \{t_0, t_1, ..., t_{n+d}\}$, hvor de første k og de siste k-skjøtverdier er like, dvs. $t_0 = t_2 = ... = t_d$ og $t_n = t_{n+1} = ... = t_{n+d}$,

Lukket og syklisk - er en kurve der skjøtvektor "biter seg selv i halen", dvs. er også C^{d-1}-glatt over endene. Den har en skjøtvektor $\tau = \{t_0, t_1, ..., t_{n+d}\}$, men domenet er utvidet til $[t_d, t_{n+d})$. I tillegg må $\quad t_{n+i+1} - t_{n+i} = t_{i+1} - t_i, \quad i = 0, 1, \ldots, d-1$.



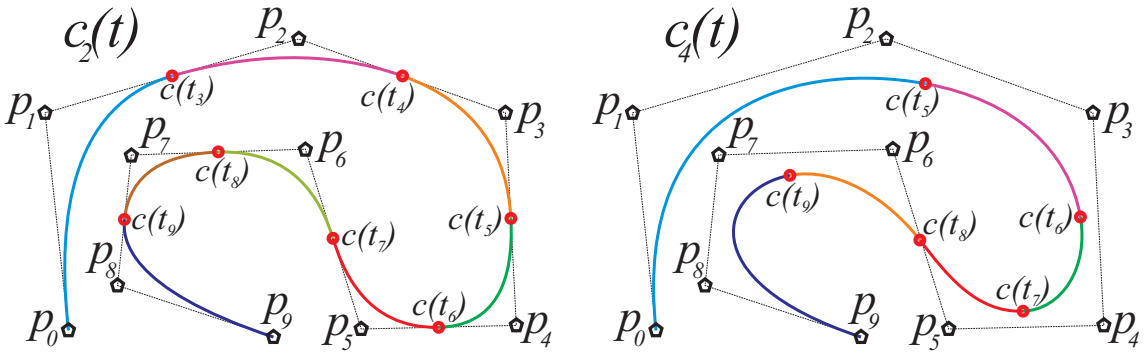
Figur 6.6: *Øverst er det et sett med 10 2.-grads B-spline funksjoner $b_{2,i}(t)$, definert av en skjøtvektor med 13 tall, alle tallene merket med røde kuler. Nederst er det et sett med 10 4.-grads B-spline funksjoner $b_{4,i}(t)$, definert av en skjøtvektor med 15 tall.*

I figur 6.5 ser vi tre 2.-grads B-splinekurver samt egne plott av de respektive basisfunksjonene. Alle tre kurvene er basert på de samme 6 kontrollpunktene. Kurven α) er åpen og "ikke klemt", β) er lukket og syklisk og γ) er åpen og klemt.

En åpen kurve kan være klemt eller ikke klemt. En åpen ikke-klemt B-splinekurve er vist som kurve α) i figur 6.5. På plottet til basisfunksjonene er domenet merket og er $[t_2, t_6]$, dvs. over 4 skjøtintervall. I plottet av kurven kan vi se at den starter og slutter i en skjøtverdi mellom to kontrollpunkt.

En åpen og klemt B-splinekurve starter i den første koeffisienten, og går deretter ut i samme retning som kontrollpolygonet. Den ender i den siste koeffisienten og kommer også inn i samme retning som kontrollpolygonet. I figur 6.5 er kurven γ) et eksempel på en klemt B-splinekurve. Kurven starter og slutter ved første og siste kontrollpunkt. 1.-deriverte ved endepunktene følger også kontrollpolygonet og det er $k = d + 1$ like skjøtverdier ved start og slutt. **Denne typen B-spline kurver er den vanligste, og generelt hvis ingenting er sagt, har vi denne typen kurve.**

En lukket og syklisk B-spline kurve er vist som kurve β) i figur 6.5. For de åpne kurvene α) og γ) så vi at domenet til begge disse var over 4 skjøtintervall. For den lukkede og sykliske kurven er domenet derimot over 6 intervall. Dette ser vi tydeligst i plottet av α) og β) nederst i figur 6.5. Vi ser der at den åpne ikke klemte kurven er lik den lukkede sykliske kurven over 4 skjøtintervall. De to ekstra skjøtintervallene i kurven β) er parameterdomenet som lukker kurven. Det følger da at domenet til β) er det halvåpne intervallet $[t_2, t_8)$. Begrensningen på skjøtvektoren er imidlertid at de to siste skjøtintervallene er lik de to første, dvs. $t_1 - t_0 = t_7 - t_6$ og $t_2 - t_1 = t_8 - t_7$, dette fordi β) er av grad $d = 2$. Generelt er det d like intervall i starten og slutten.



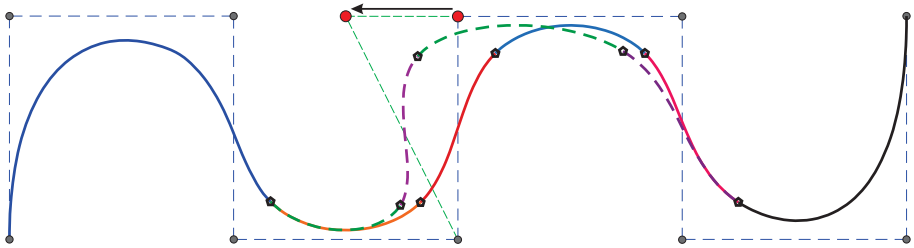
Figur 6.7: *Fra venstre, en 2.-grads B-splinekurve $c_2(t)$ og en 4.-grads B-splinekurve $c_4(t)$, begge laget av de samme kontrollpunktene. Røde kuler markerer interne skjøter.*

Figur 6.1 viser et sett med 3.-grads B-spline funksjoner basert på en skjøtvektor $\tau = \{0,0,0,0,1,2,3,4,5,6,7,7,7,7\}$. Den korresponderende figuren 6.2 viser en kurve, som er laget fra disse basisfunksjonene samt et sett med 10 kontrollpunkt p_i, $i = 0,1,\ldots,9$.

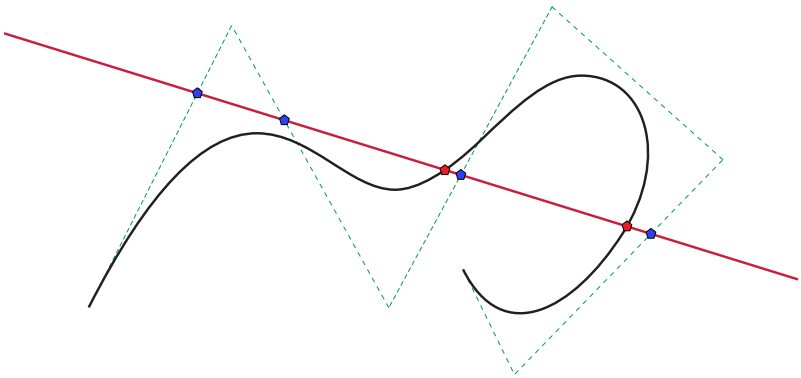
Fra samme sett med kontrollpunkt lager vi så en kurve med polynomgrad 2 basert på skjøtvektoren $\{0,0,0,1,2,3,4,5,6,7,8,8,8\}$, og en annen kurve med polynom grad 4 og en skjøtvektor $\{0,0,0,0,0,1,2,3,4,5,6,6,6,6,6\}$. I figur 6.6 ser vi plott av B-splines basert på disse to skjøtvektorene. I figur 6.7 er B-splinekurver basert på disse to settene med basisfunksjoner plottet. Vi ser at 2.-gradskurven berører kontrollpolygonet på grunn av at i skjøtene er det bare to basisfunksjoner $\neq 0$, illustrert øverst i figur 6.6. Følgelig er berøringspunktene også der hvor kurvesegmentene limes sammen. I 2.-gradskurven har vi 8 segment, vises til venstre i figur 6.7, mens i 3.-gradskurven har vi 7 segment, vises i figur 6.2, og i 4.-gradskurven 6 segment, vises til høyre i figur 6.7.

En B-spline kurve har mange nyttige egenskaper, noen av dem er listet opp nedenfor.

1. En B-splinekurve $c(t)$ er en stykkevis kurve der hvert segment er en kurve med grad opptil d. Kurven er en union av kurvesegment definert på hvert sitt skjøtintervall.
2. En klemt B-splinekurve starter i det første kontrollpunktet og slutter i det siste kontrollpunktet. Videre forlater kurven det første punktet tangentielt til linjen mellom det første og andre kontrollpunkt og kommer inn til det siste kontrollpunktet tangentielt til linjen mellom det nest siste og siste kontrollpunktet.
3. B-splinekurver har en sterk “konveks omhyldningsegenskap". Det betyr at en B-splinekurve er i sin helhet i den konvekse omhyldningen til sine kontrollpunkt.
4. En B-spline kurve har et lokalt modifikasjonsskjema. Det vil si at endring av posisjonen til kontrollpunktet c_i kun påvirker kurven $c(t)$ i intervallene $[t_i, t_{i+k}]$.
5. En B-splinekurve $c(t)$ er C^{d-j} kontinuerlig i en skjøt med multiplisitet j. Det vil si at hvis $t_i = t_{i+1} = \ldots = t_{i+j-1}$, så er kurven C^{d-j} kontinuerlig ved $c(t_i)$.
6. En B-spline kurve har en *variasjonsminkende egenskap*. Det vil si at i $\mathbb{R}^2$, skjærer ingen rett linje en B-splinekurve flere ganger enn den skjærer kontrollpolygonet.
7. Bézierkurver er spesialtilfeller av B-splinekurver. De er klemte B-splinekurver på



Figur 6.8: *En 3.-grads B-splinekurve og dens kontrollpolygon er plottet. En av kontrollpunktene flyttes til venstre (rød kule). For å illustrere den lokale kontrollen er den delen av kurven som endrer seg stiplet.*



Figur 6.9: *En 3.-grads B-splinekurve og dens kontrollpolygon. En linje skjærer kurven 2 ganger og kontrollpolygonet 4 ganger*

domenet $[0, 1]$, uten interne skjøter, kun skjøtverdier ved start og slutt.

8. Den affine invariansegenskapen gjelder for B-splinekurver. Dersom en B-spline kurve skal translateres, roteres, skaleres, ..., dvs. påføres en affin transformasjon, se (2.4), kan dette gjøres ved å bruke en affine avbildning kun på kontrollpunktene. Dette gjelder også for bézierkurver, seksjon 4.4, og dersom en homogene matrise brukes, (2.8), gjelder det også for hermitekurver, se seksjon 4.3 .

9. Kontrollpolygonen til en B-splinekurve konvergerer mot B-splinekurven når antall skjøter går mot uendelig. Dette vises tydelig i skjøtinnsettingsalgoritmen vist i seksjon 6.2.6. Dette fordi B-splinealgoritmen i utgangspunktet er hjørnekutting, se også avsnitt 6.7 om subdivisjonskurver, spesielt Lane-Riesenfeld subdivisjonsalgoritme i seksjon 6.7.3.

Figur 6.8 illustrerer lokal modifikasjon. Vi ser en 3.-grads B-splinekurve hvor hvert intervall har forskjellig farge. Skjøtvektoren som bestemmer B-splines er den samme som i figur 6.1. 9 av de 10 kontrollpunktene er merket med blått, mens ett er rødt. Det røde punktet flyttes så til venstre og vi ser da hvilken del av kurven som er påvirket. Det er 4 intervall fordi graden er 3.

Figur 6.9 illustrerer den variasjonsminkende egenskapen. En rett linje skjærer en 3.-grads B-splinekurve 2 ganger, mens den skjærer kontrollpolygonet 4 ganger.

6.2.3 B-splines faktormatriser T(t)

I seksjon 4.4.2 definerte vi faktormatriser for bézierkurve $T_d(t)$, og i (4.34) er en 3.-grads bézierkurve formulert med bruk av matrisenotasjon. Algoritmen for å beregne bernstein-polynomene er beskrevet i seksjon 4.4.1. En sammenligning av Bernstein-rekursjonen med B-spline-rekursjonen, definition 6.3, viser at de er svært like, men hvor t i Bernstein-formelen er erstattet av funksjonen $w_{d,i}(t)$, beskrevet i (6.11) i B-spline definisjonen. Husk at $w_{d,i}(t)$ overfører et tall i intervallet $[t_i, t_{i+d}]$ til et tall i $[0,1]$.

En B-splinekurve er et sett med kurvesegment som er limt sammen i skjøtene. For en 2.-grads B-splinekurve $c(t)$ får vi et kurvesegment for hvert parameterintervall $[t_i, t_{i+1})$, $i = d, d+1, \ldots, n$ med følgende formel (forutsatt at $t_{i+1} > t_i$),

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

Matrisene i formelen over kalles for *faktormatriser* fordi de faktoriserer B-splines over ett skjøtsintervall (introdusert i [103]). En generell definisjon av en faktormatrise følger.

Definisjon 6.4. *En B-spline faktormatrise $T_d(t)$ er en $d \times (d+1)$ båndmatrise med to ikke-null-elementer (basert på $w_{d,i}(t)$ (6.11)) på hver rad. Matrisen er som følger*

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

Merk at denne matrisen $T_d(t)$, er for skjøtintervallet $[t_i, t_{i+1})$.

Vi ser i matrisen $T_d(t)$ at den andre indeksen til w reduseres med 1 fra en rad til neste rad. I siste raden er den i og i raden over $i-1$ og så videre.

For grad 3 har vi følgende formel for en B-splinekurve på intervallet $[t_i, t_{i+1})$,

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

hvor indeksen i er bestemt av $t_i \leq t < t_{i+1}$. Legg for øvrig merk til at indeksene til $w(t)$ på siste rad i alle matrisene $T_d(t)$ er (d,i). Som for bézierkurver (4.35), er uttrykket for en 3.-grads B-splinekurve:

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

der $T^3(t)$ er en vektor av 4 B-splines av grad 3 på intervallet $[t_i, t_{i+1})$.

Teorem 6.1. *Matrisene $T_q(t)$, $q = 1,2,\ldots,d$ i definisjon 6.4 er tilsammen faktoriseringen av et sett med B-splines på et skjøtintervall, og de er identisk med den rekursive beskrevelsen i Definisjon 6.3, Cox-de'Boor´s rekursjonsformelen.*

Bevis. Vi vet at $T_1(t) = (1 - w_{1,i}(t) \quad w_{1,i}(t)) = (b(t;t_{i-1},t_i,t_{i+1}) \quad b(t;t_i,t_{i+1},t_{i+2}))$. Vi ser også at $T^1(t) = T_1(t)$. Vi antar nå at $T^{d-1}(t)$ er en vektor av d B-spline funksjoner av grad $d-1$ på intervallet $[t_i,t_{i+1}]$. Beregning av denne vektoren med hver av kolonnene i $T_d(t)$ gir hver B-splines av grad d og er nøyaktig det samme som uttrykket (6.10) beskriver i Definisjon 6.3. □

For å undersøke den deriverte av $T(t)$ må vi først se på den deriverte av den lineære translasjons- og skaleringsfunksjonen $w_{d,i}(t)$, definert i (6.11), som vi kaller

$$\delta_{d,i} = \frac{1}{t_{i+d} - t_i}, \quad \text{hvor } d \text{ er graden og } i \text{ er bestemt av} \quad t_i \le t < t_{i+d} \tag{6.13}$$

$\delta_{d,i}$ er en konstant, som bare reflekterer skaleringen, uavhengig av translering, og bare avhengig av t for å finne indeksen i. I béziertilfellet er $\delta_{d,i} = 1$ for alle relevante i og d.

Den deriverte av matrisen $T(t)$ er da som følger.

Definisjon 6.5. *En B-spline derivertematrise T'_d er en $d \times (d+1)$ båndmatrise med to konstante elementer som ikke er null på hver rad (uavhengig av t). Matrisen er:*

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

Selv om T'_d i seg selv er uavhengig av t, er indeksen i i definisjonen avhengig av t og følger regelen $t_i \le t < t_{i+1}$ som er den samme som for $T_d(t)$ fra definisjonen 6.4.

6.2.4 B-splines på matriseform

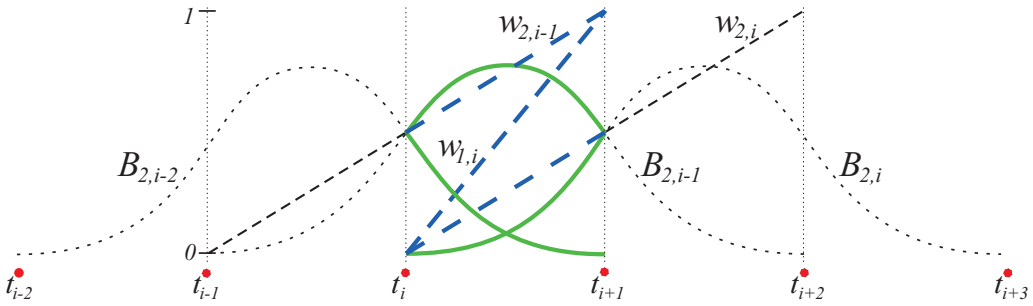
Først et konkret eksempel, gitt en tredje grads B-spline kurve

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

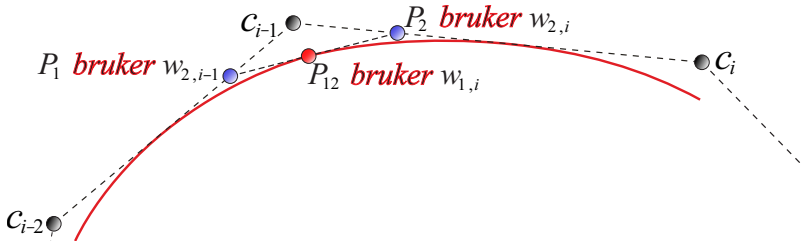
Vi bruker så derivasjon av multiplikasjon og kommutativitetsteoremet C.1 fra vedlegg C.2 og får da de deriverte

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

Et generelt uttrykk for en B-spline funksjoner/kurver og dens deriverte på matriseform følger samme logikk og vi får da følgende definisjon;



Figur 6.10: *Vi ser tre 2.-grads B-splines på skjøtintervallet $[t_i, t_{i+1}]$ (heltrukket grønt) og 3 lineære translasjons- og skaleringsfunksjoner (stiplet blå) som er involvert i beregningene av B-splines. Både B-splines og w er stiplet utenfor intervallet $[t_i, t_{i+1}]$.*



Figur 6.11: *De Casteljaus hjørnekuttingsalgoritme er brukt på en 2.-grads B-splinekurve med skjøtene $0,1,2,3$. I eksemplet er posisjonen p_{12} i $c\left(\frac{3}{2}\right)$.*

Definisjon 6.6. *En B-spline funksjon/kurve av grad d på matriseform er som følger*

$$c(t) = T_1(t)\ T_2(t)\ \cdots\ T_d(t)\ \mathbf{c} = T^d(t)\ \mathbf{c}$$

hvor $T_1(t)\ T_2(t)\ \cdots\ T_d(t)$ er faktoriseringen (definisjon 6.4) av settet med $d+1$ B-splines av grad d på intervallet $[t_i, t_{i+1}]$, og $\mathbf{c} = (c_{i-d},\ c_{i-d+1},\ \ldots,\ c_i)^T$ er koeffisientvektoren med $d+1$ punkt, og hvor indeksen i er bestemt av $t_i \le t < t_{i+1}$.

Den j^{te}-deriverte, $j = 1,2,\ldots,d$, av en B-spline funksjon/kurve av grad d er en funksjon av grad $d-j$ og er som følger

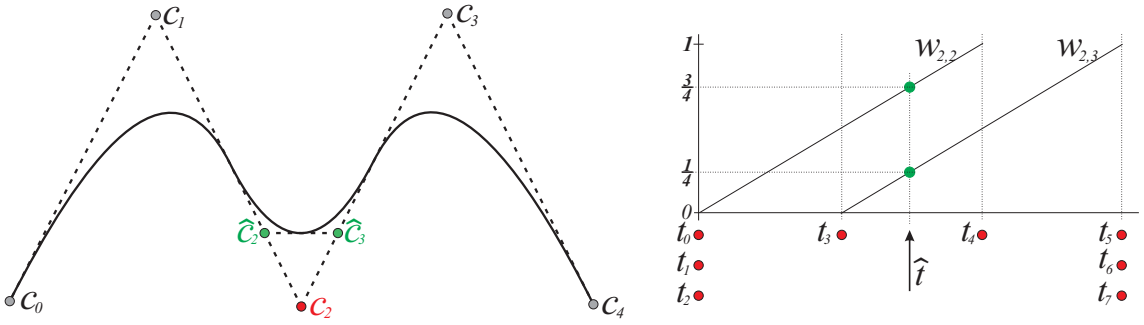
$$c^{(j)}(t) = \frac{d!}{(d-j)!}\ T_1(t)T_2(t)\cdots T_{d-j}(t)\ T'_{d-j+1}\ T'_{d-j+2}\ \cdots\ T'_d\ \mathbf{c} = \frac{d!}{(d-j)!}\ T^{d-j}(t)\ T'^{j}\ \mathbf{c}.$$

6.2.5 Et eksempel på B-splines og de Casteljaus algoritme

Vi skal gjøre det mest mulig enkelt, dvs. bruke en 2.-grads B-splinecurve

$$\begin{aligned} c(t) &= T^2(t)\ \mathbf{c} = T_1(t)\ T_2(t)\ \mathbf{c} = (B_{2,i-2}(t),\ \ B_{2,i-1}(t),\ \ B_{2,i}(t))\,\mathbf{c} \\ &= \begin{pmatrix} 1-w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1-w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1-w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}. \end{aligned}$$

Figur 6.10 viser tre 2.-grads B-splines, $B_{2,i-2}(t), B_{2,i-1}(t)$ og $B_{2,i}(t)$, sammen med de tre lineære translasjons- og skaleringsfunksjonene de er konstruert fra (husk at dette kun er på intervallet $[t_i, t_{i+1})$). De 3 funksjonene er $w_{1,i}$ som er i matrise $T_1(t)$, og $w_{2,i-1}$ og $w_{2,i}$ som er i den andre matrisen $T_2(t)$.



Figur 6.12: *Til venstre vises en 2.-grads B-splinekurve og dens kontrollpolygon. Til høyre er skjøtvektoren illustrert. En ny skjøt, $\widehat{t}$ blir så satt inn. Resultatet er, som vi kan se til venstre, to nye kontrollpunkt $\widehat{c}_2$ og $\widehat{c}_3$ i grønt, som erstatter det gamle kontrollpunktet c_2 som vises i rødt.*

Figur 6.11 viser de Casteljaus hjørnekuttingsalgoritme brukt på en 2.-grads B-splinekurve med skjøtene $\{t_{i-1},\ldots,t_{i+2}\} = \{0,1,2,3\}$. $\widehat{t} = 3/2$ er brukt som parameter i eksempelet. Dette gir $w_{1,1}(\widehat{t}) = 1/2$ i den første matrisen $T_1(t)$, og $w_{2,i-1}(\widehat{t}) = 3/4$ og $w_{2,i}(\widehat{t}) = 1/4$ i den andre matrisen $T_2(t)$. De to første nye punktene p_1 og p_2 beregnes, og til slutt blir punktet p_{12} (det på kurven) beregnet,

$$\begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} 1-\frac{3}{4} & \frac{3}{4} & 0 \\ 0 & 1-\frac{1}{4} & \frac{1}{4} \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} = \begin{pmatrix} \frac{1}{4}c_{i-2} + \frac{3}{4}c_{i-1} \\ \frac{3}{4}c_{i-1} + \frac{1}{4}c_i \end{pmatrix} .$$

$$p_{12} = \begin{pmatrix} 1-\frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\,p_1 + \frac{1}{2}\,p_2 \end{pmatrix}$$

6.2.6 B-splines og skjøtinsetting

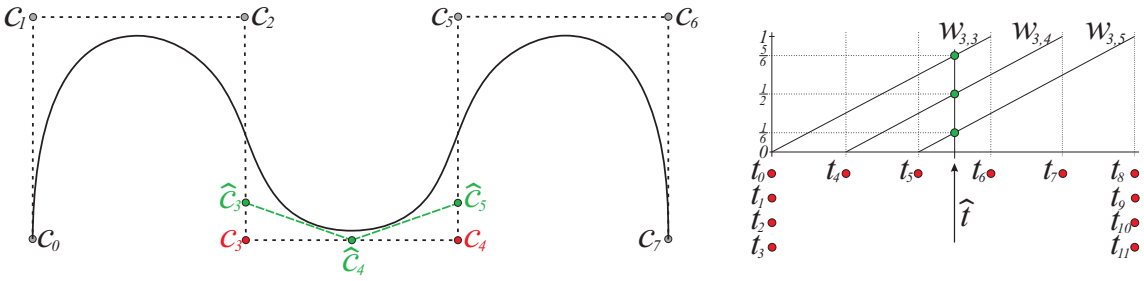
Skjøtinnsetting er å legge til en ny skjøt inn i en eksisterende skjøtvektoren uten å endre formen på kurven. Denne nye skjøten kan også være lik en eksisterende skjøtverdi, og da økes multiplisiteten til skjøten med én. Å legge til en ny skjøt betyr at antallet B-splines øker med én og følgelig må antall kontrollpunkt økes med én. Skjøtinnsetting ble introdusert samtidig i 1980 av Boehm [15] for enkeltskjøter, og av Cohen, Lyche og Riesenfield [23] for generell skjøtinnsetting, kalt for Oslo-algoritmen.

Skjøtinnsetting er i utgangspunktet hjørnekutting og kan derfor også uttrykkes på matriseform. Når en enkelt skjøt settes inn får vi følgende vektor- matriseuttrykk

$$\widehat{\mathbf{c}} = T_d(\widehat{t})\ \mathbf{c} \tag{6.14}$$

der $\widehat{t}$ er den nye skjøten og $\widehat{\mathbf{c}}$ er en vektor av d nye kontrollpunkt som erstatte de $d-1$ kontrollpunktene som er i det indre av vektor $\mathbf{c}$ (bortsett fra det første og det siste kontrollpunktet). Her bestemmes som vanlig indeksen i av $t_i \le \widehat{t} < t_{i+1}$. Vi skal nå se på noen eksempler, først en 2.-grads B-splinekurve

$$\begin{pmatrix} \widehat{c}_{i-1} \\ \widehat{c}_i \end{pmatrix} = \begin{pmatrix} 1-w_{2,i-1}(\widehat{t}) & w_{2,i-1}(\widehat{t}) & 0 \\ 0 & 1-w_{2,i}(\widehat{t}) & w_{2,i}(\widehat{t}) \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} . \tag{6.15}$$



Figur 6.13: *Til venstre er en 3.-grads B-splinekurve og dens kontrollpolygon plottet. Til høyre er skjøtvektoren illustrert. En ny skjøt, $\widehat{t}$ er satt inn. Resultatet er, som vi kan se til venstre, tre nye kontrollpunkt $\widehat{c}_3$, $\widehat{c}_4$ og $\widehat{c}_5$ i grønt, som erstatter de to gamle kontrollpunktene c_3 og c_4 som vises i rødt.*

Til venstre i figur 6.12 er en 2.-grads B-splinekurve samt koeffisientene og dermed kontrollpolygon $\{c_0, c_1, c_2, c_3, c_4\}$ plottet. Til høyre er det en illustrasjon av skjøtvektoren og de respektive w-funksjonene. Midt i domenet skal en ny skjøt, $\widehat{t}$, settes inn. Det følger av verdien til $\widehat{t}$ at $i = 3$. De to lineære translerings- og skaleringsfunksjonene $w_{2,2}$ og $w_{2,3}$ som er involvert i matrisen $T_2(\widehat{t})$ i (6.15), vises til høyre i figuren. Resultatet av skjøtinnsettingen er at den interne koeffisienten i vektoren til høyre i (6.15), kalt c_2 i figur 6.12, erstattes av to nye koeffisientene i vektoren til venstre i (6.15), kalt $\widehat{c}_2$ og $\widehat{c}_3$ i figur 6.12.

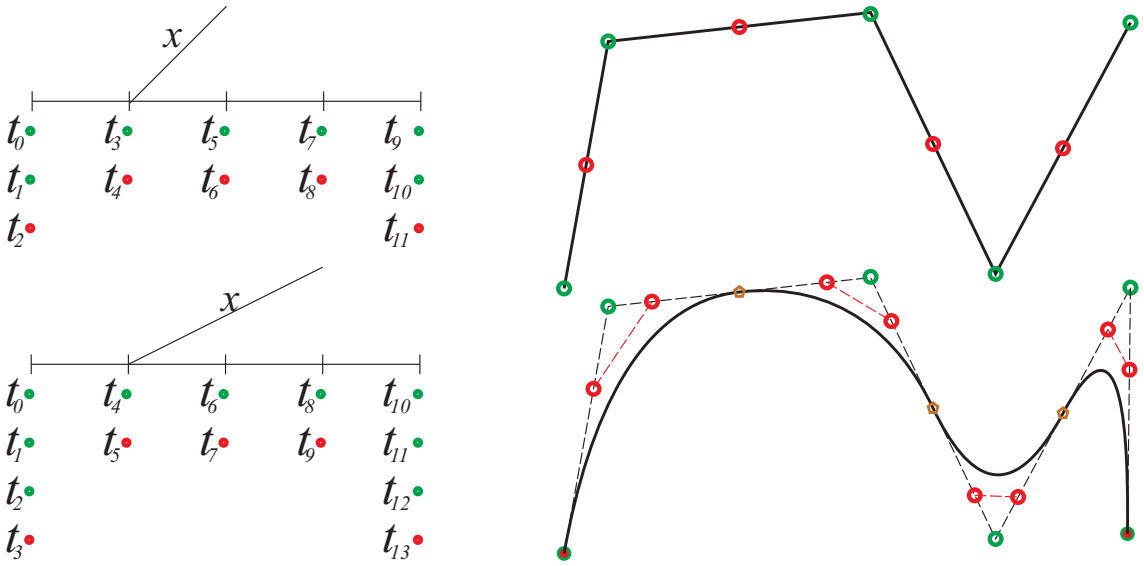
Vi skal vise et eksempåel til, nå med en 3.-grads B-splinekurve

$$\begin{pmatrix} \widehat{c}_{i-2} \\ \widehat{c}_{i-1} \\ \widehat{c}_i \end{pmatrix} = \begin{pmatrix} 1 - w_{3,i-2}(\widehat{t}) & w_{3,i-2}(\widehat{t}) & 0 & 0 \\ 0 & 1 - w_{3,i-1}(\widehat{t}) & w_{3,i-1}(\widehat{t}) & 0 \\ 0 & 0 & 1 - w_{3,i}(\widehat{t}) & w_{3,i}(\widehat{t}) \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} \quad (6.16)$$

Til venstre i figur 6.13 er en 3.-grads B-splinekurve sammen med koeffisientene og med det kontrollpolygonet $(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7)$ plottet. På høyre side er skjøtvektoren illustrert (på $\mathbb{R}$ - horisontalt). Midt på domenet skal en ny skjøt, $\widehat{t}$, settes inn. Det følger av verdien til $\widehat{t}$ at $i = 5$. De tre lineære translerings- og skaleringsfunksjonene som er involvert i matrisen $T_3(\widehat{t})$ i (6.16), $w_{3,3}$, $w_{3,4}$ og $w_{3,5}$ er vist til høyre i figur 6.13. Resultatet av skjøtinnsettingen er at de interne koeffisientene på høyre side av (6.16), kalt c_3 og c_4 i figur 6.13 erstattes av de tre nye koeffisientene på venstre side av (6.16), kalt $\widehat{c}_3$, $\widehat{c}_4$ og $\widehat{c}_5$ i figur 6.13. Hele hjørnekuttingen er tydelig illustrert til venstre i figur 6.13.

Fra seksjon 4.4.2 og definisjon 6.4 er det klart at matrisen $T_d(t)$ er en hjørnekuttingsmatrise. Det er derfor åpenbart at skjøtinnsetting også er hjørnekutting. I de to eksemplene kan vi også observere at de nye kontrollpunktene vi får når vi setter inn én ny skjøt alle ligger på det gamle kontrollpolygonet. Dette er også koblet til diskrete B-splines.[6] Å sette inn mer enn én knute, dvs. Oslo-algoritmen), leder oss også til blomstring (Blossoming), se for øvrig [76].

[6]Diskrete splines på et uniformt grid ble introdusert av Mangasarian og Schumaker i [118] som løsninger på visse variasjonsproblemer. De ble så diskutert i detalj av Tom Lyche i sin doktorgradsavhandling, “Discrete polynomial spline approximation methods", som det finnes en oppsummering av i [116].



Figur 6.14: *Oppe til venstre er skjøtvektoren til en 1.-grads B-splinekurve illustrert som grønne kuler. Til høyre ser vi selve kurven og dens 5 (grønne) kontrollpunkt. Etter gradshevingen til grad 2 er 5 (røde) skjøtverdier og 4 (røde) kontrollpunkt lagt tiI. Nederst til venstre er skjøtvektoren til en 2.-grads B-splinekurve illustrert som grønne kuler. Til høyre er selve kurven og dens 6 (grønne) kontrollpunkt plottet. Etter gradshevingen til grad 3 er 5 (røde) skjøtverdier lagt til, og 8 (røde) kontrollpunkt har erstattet 4 (grønne) kontrollpunkt.*

6.2.7 Gradsheving av B-splines

I seksjon 4.4.4 er gradsheving av bézierkurver vist. Ved hvert skjøtintervall hjar vi et polynombasert kurvesegment av en gitte grad. Hvis vi i hvert skjøtintervall transformerer formatet til potensbasis, og så legger til ett ledd t^{d+1}, dvs. $c(t) = \sum_{i=0}^{d+1} a_i t^i$, hvor $a_{d+1} = 0$, da har vi hevet graden med 1. Utfordringen er å gjøre dette direkte på B-splineformatet.

På grunn av kontinuitetsegenskapen over skjøter til B-splines, følger det at for å opprettholde kontinuiteten når vi øker graden med 1, må vi øke multiplisiteten til hver klynge av skjøter med 1. Dermed må antallet kontrollpunkt økes med det totale antallet skjøtintervall.

Som eksempel skal vi først se på en 1.-grads, stykkevis kontinuerlig lineær kurve. Her er det ganske åpenbart at etter gradshevingen vil det være ett nytt kontrollpunkt midt mellom hver av de gamle, dvs. i hvert intervall. Det betyr selvsagt at de nye kontrollpunktene vil ligge på den opprinnelige kontrollpolygonen.

For å finne formelen / algoritmen for å heve en 1.-gradskurve til en 2.-gradskurven. Skal vi videre bruke $\sim$ i notasjonen til den opprinnelige 1.gradskurven. Gitt en skjøtvektor til en 1.-grads B-spline $\tilde{\tau} = \{\tilde{t}_i\}_{i=0}^{6} = \{0,0,1,2,3,4,4\}$. Etter gradshevingen får vi $\tau = \{t_i\}_{i=0}^{11} = \{0,0,0,1,1,2,2,3,3,4,4,4\}$. Husk (6.11), at $\widetilde{w}_{1,i}(t) = \frac{t-\tilde{t}_i}{\tilde{t}_{i+1}-\tilde{t}_i}$. For å forenkle setter vi $x = \widetilde{w}_{1,i}(t)$ og $\bar{x} = 1-x$. Hvis $t_{j-1} = t_j$ og $t_{j+1} = t_{j+2}$ etter gradhevingen, så er $x = \widetilde{w}_{1,i}(t) = w_{2,2i-1}(t) = w_{2,2i}(t)$. Vi får dermed følgende uttrykk i hvert skjøtintervall,

$$c(t) = \begin{pmatrix} \bar{x} & x \end{pmatrix} \begin{pmatrix} \widetilde{c}_{i-1} \\ \widetilde{c}_i \end{pmatrix} = \begin{pmatrix} \bar{x} & x \end{pmatrix} \begin{pmatrix} \bar{x} & x & 0 \\ 0 & \bar{x} & x \end{pmatrix} \begin{pmatrix} c_{2i-2} \\ c_{2i-1} \\ c_{2i} \end{pmatrix}$$

Fra $x = 0$ (starten av intervallet) følger det at $c_{2i-2} = \widetilde{c}_{i-1}$ og fra $\bar{x} = 0$ (slutten av intervall) at $c_{2i} = \widetilde{c}_i$. Ved å løse $\bar{x}^2\widetilde{c}_{i-1} + 2\bar{x}x\, c_{2i-1} + x^2\widetilde{c}_i = \bar{x}\widetilde{c}_{i-1} + x\widetilde{c}_i$ vi får det nye punktet $c_{2i-1} = \frac{1}{2}\widetilde{c}_{i-1} + \frac{1}{2}\widetilde{c}_i$. Øverst i figur 6.14 er denne gradhevingen illustrert. På venstre side ser vi den originale skjøtvektoren som grønne kuler, og de røde kulene er de nye skjøtverdiene. Også $x = \widetilde{w}_{1,2}(t) = w_{2,3}(t) = w_{2,4}(t)$ er plottet. Til høyre er den stykkevis lineære 1.-grads B-splinekurven og dens kontrollpunkt (grønn) plottet. 2.-gradskurven er den samme kurven, men hvor de røde punktene er de nye kontrollpunktene Sammen med de originale grønne punktene danner de den nye kontrollpolygonen.

For en 2.-graders B-spline kurve, vil antall kontrollpunkt også øke med antall skjøtintervall. Dvs. hvis $\tilde{t}_i < \tilde{t}_{i+1}$, må $\tilde{c}_{i-1}$ erstattes av to nye kontrollpunkt. Dette er fordi vi setter inn en ny skjøt i hver klynge med skjøter. For å illustrere prosessen starter vi med en skjøtvektor $\widetilde{\tau} = \{\tilde{t}_i\}_{i=0}^{8} = \{0,0,0,1,2,3,4,4,4\}$. Dette vil generere seks 2.-grads B-splines, og B-splinekurven $c(t) = \sum_{i=0}^{5} b_{2,i}(t)\ \widetilde{c}_i$. Nederst i figur 6.14 er det et eksempel på en slik kurve, illustrert med 9 grønne skjøter og 6 grønne kontrollpunkt. Den nye skjøtvektoren $\tau = \{t_i\}_{i=0}^{13} = \{0,0,0,0,1,1,2,2,3,3,4,4,4,4\}$ er illustrert nederst til venstre i figuren med ekstra røde kuler. Til høyre ser vi kurven med både gamle og nye kontrollpunkt samt en liten brun femkant ved hver klynge av innvendige skjøter. For å finne det nye settet med kontrollpunkt, må vi gjøre følgende. For hvert skjøtintervall, dvs. for $i = 0,1,2,\ldots,\tilde{n}-2$, (i figur 6.14 er $\tilde{n} = 6$) er:

$q = (1 - w_{2,i+1}(\tilde{t}_{i+2}))\tilde{c}_i + w_{2,i+1}(\tilde{t}_{i+2})\tilde{c}_{i+1}$, dvs. posisjonen til en skjøt på kurven.
Vi lager to nye kontrollpunkt i dette skjøtintervallet, jmf. algoritme(4.40),
$c_{2i} = \frac{2}{3}\tilde{c}_i + \frac{1}{3}q$ og $c_{2i+1} = \frac{1}{3}q + \frac{2}{3}\tilde{c}_{i+1}$,

En oppsummering av dette gir da en gradshevingsalgoritme fra grad 2 til 3:

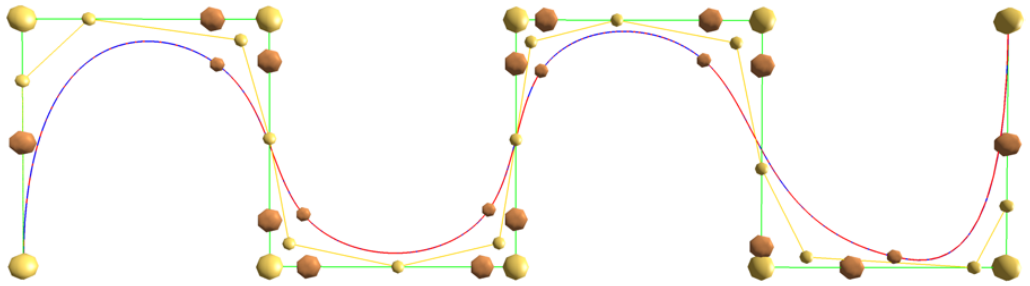
Gradsheving fra 2 til 3

For $i = 0,1,2,\ldots,\tilde{n}-2$

$x = w_{2,i+1}(\tilde{t}_{i+2})$	// Relativ fordeling i dette skjøtintervallet
$c_{2i} = \left(1 - \frac{x}{3}\right)\tilde{c}_i + \frac{x}{3}\tilde{c}_{i+1}$	// To kontrollpunkt i dette intervallet for
$c_{2i+1} = \frac{1}{3}(1-x)\,\tilde{c}_i + \frac{1}{3}(2+x)\,\tilde{c}_{i+1}$	// den resulterende 3.-grads B-splinekurven.

Hvis det er multiple interne skjøter i den originale 2.-grads B-splinekurven vil vi her få ett kontrollpunkt to ganger. For eksempel, hvis $\tilde{t}_4 = \tilde{t}_5$, vil c_5 bli lik c_6, og den resulterende skjøtvektor vil ha 4 like skjøtverdier $t_6 = t_7 = t_8 = t_9$. Men vi kan redusere dette til bare 3 like skjøtverdier, men vi må da hoppe over c_6 og redusere indeksen for de neste kontrollpunktene med 1 i algoritmen.

Som et siste eksempel skal vi se på en 3.-gradskurve. I dette eksemplet er skjøtvektoren $\widetilde{\tau} = \{0,0,0,0,1.08,2,3.2,4,5.1,6.5,7,7,7,7\}$, dvs. ikke uniform med med kun enkle indre skjøter. Dermed vil skjøtvektoren til den gradshevede 4.-grads B-splinekurven være $\tau = \{0,0,0,0,0,1.08,1.08,2,2,3.2,3.2,4,4,5.1,5.1,6.5,6.5,7,7,7,7,7\}$. Eksemp-



Figur 6.15: *Vi ser en 3.-grads B-splinekurve i blått og dens 10 kontrollpunkt som vises som store messingfargede kuler. Kurven er vist sammen med den gradshevede 4.-grads B-splinekurven i rødt og dens 17-kontroll punkt, små messingfargede kuler. De store kobberfargede kulene er punktene i det første trinnet, q_0 og q_1. De små kobberfargede kulene er posisjonen til de indre skjøtene.*

let er vist i figur 6.15. Merk at de nye basisfunksjonene b_0 og b_1 dekker 1 skjøtintervall, b_2 og b_3 dekker 2 intervall, b_4 dekker 3 intervall, b_5 dekker 2 intervall, og så videre med 2 og 3 skjøtintervall annenhver gang, til slutten speiles starten. Vi har i utgangspunktet 10 kontrollpunkt og vi får 17 etter gradhevingen. De to første og de to siste kontrollpunktene følger bézier-algoritmen fordi de relaterte basisfunksjonene kun dekker 1 skjøtintervall, men vi trenger likevel ikke å behandle dem separat. Videre må vi skille mellom kontrollpunktene knyttet til basisfunksjoner som dekker 2 og de som dekker 3 skjøtintervall. Ved start og slutt er dekningen redusert på grunn av skjøtens multiplisitet, men dette vil ikke påvirke hvordan vi må behandle dem. Vi trenger bare å skille algoritmen i; ett for de nye kontrollpunktene med et partall som indeks, og ett for de nye kontrollpunktene med et oddetall som indeks. I vårt eksempel; for punktene med en partall-indeks får vi, for $i = 0, 1, \ldots, 8$:

$$c_{2i} = (1-x)\,\tilde{c}_i + x\,\tilde{c}_{i+1}, \qquad \text{hvor} \qquad x = \frac{w_{3,i+1}(\tilde{t}_{i+2}) + w_{3,i+1}(\tilde{t}_{i+3})}{2}.$$

I figur 6.15 kan disse punktene sees som de små messingfargede kulene, men bare de som ligger på den opprinnelige kontrollpolygonet.

Det neste steget er å beregne punktene med en oddetallsindeks, dvs. for $i = 0, 1, \ldots, 7$:

$$\begin{aligned} q_0 &= x\tilde{c}_i + (1-x)\tilde{c}_{i+1}, && \text{hvor} && x = \frac{1 - w_{3,i+1}(\tilde{t}_{i+3})}{2}, \\ q_1 &= (1-y)\tilde{c}_{i+1} + y\tilde{c}_{i+2}, && \text{hvor} && y = \frac{w_{3,i+2}(\tilde{t}_{i+3})}{2}, \\ c_{2i+1} &= \frac{q_0+q_1}{2}, \end{aligned}$$

Merk at $w_{3,i}(t)$ bruker den opprinnelige skjøtvektoren $\widetilde{\tau}$. I figur 6.15 er q_0 og q_1 merket som store kobberfargede kuler og kan sees på hver side av ett originalt intern kontrollpunkt (som er store messingfargede kuler), bortsett fra ett som er dekket av det andre originale kontrollpunktet og ett som er dekket av det nest siste originale kontrollpunktet. De nye kontrollpunktene med odde indekser (vises som små messingfargede kuler) ligger i midten mellom deres respektive q_0 og q_1. Vi gjør så algoritmen generell, forkorter den og får gradhevingsalgoritmen fra grad 3 til 4:

Gradsheving fra 3 to 4

For $i = 0,1,2,\ldots,\tilde{n}-2$
$x = \frac{w_{3,i+1}(\tilde{t}_{i+2})+w_{3,i+1}(\tilde{t}_{i+3})}{2}$ // Relativ fordeling i dette skjøtintervallet
$c_{2i} = (1-x)\,\tilde{c}_i + x\,\tilde{c}_{i+1}$ // Kontrollpunkt med et "like" indeksnummer

For $i = 0,1,2,\ldots,\tilde{n}-3$
$x = w_{3,i+1}(\tilde{t}_{i+3})$ // Første translering- og skaleringsavbildning
$y = w_{3,i+2}(\tilde{t}_{i+3})$ // Andre translering- og skaleringsavbildning
$c_{2i+1} = \frac{1-x}{4}\tilde{c}_i + \frac{3+x-y}{4}\tilde{c}_{i+1} + \frac{y}{4}\tilde{c}_{i+2}$ // Kontrollpunkt med et "odde" indeksnummer

Hvis det er multiple interne skjøter i den originale 3. grads B-splinekurven vil vi få et kontrollpunkt to ganger. For eksempel, hvis $\tilde{t}_5 = \tilde{t}_6$, vil c_6 ligge på linjen mellom c_5 og c_7, og den resulterende skjøtvektoren vil ha 4 like skjøtverdier $t_7 = t_8 = t_9 = t_{10}$. Men vi kan redusere dette til bare 3 like skjøtverdier. Derfor må vi hoppe over c_6 ved ganske enkelt å redusere indeksene for de neste "odde" kontrollpunktene med 1, og øke indeksene for de neste "like" kontrollpunktene med 1, men i det "like" tilfelle må vi også øke indeksene i beregningen til $w_{3,i+2}(\tilde{t}_{i+3})$, $w_{3,i+2}(\tilde{t}_{i+4})$, $\tilde{c}_{i+1}$ og $\tilde{c}_{i+2}$.

6.2.8 Blomstring - polar form blomstring

I 1987 introduserte L. Ramshaw blomstring, som gjør B-splines og polynomer generelt til multiaffine funksjoner, [132], [133] og [134]. Dette var også relatert til arbeidet til P. de Casteljau fra 1984, [40]. Flere andre har senere brukt blomstring, blant andre Goldman i [76] og [77].

Blomstring eller polare form til et polynom med en enkelt variabel av grad d, $P_d(t)$, er det symmetriske multiaffine polynomet $b_p(u_1,\ldots,u_d)$ hvor diagonalen $b_p(t,\ldots,t) = P_d(t)$, dvs. er polynomet av grad d. Blossoming betyr å erstatte et grad d polynom i én variabel med et ekvivalent symmetrisk polynom i d variabler hvor hver ny variabel kun har grad 1. Fordelen med blomstringen $b_p(t_1,\ldots,t_d)$ er at den har tre viktige egenskaper:

Diagonal: $b_p(t,t,t) = p_3(t)$, hvor $d = 3$ her, men kan være et hvilket som helst tall > 0.

Symmetri: $b_p(u_1,\ldots,u_i,\ldots,u_j,\ldots,u_d) = b_p(u_1,\ldots,u_j,\ldots,u_i,\ldots,u_d)$

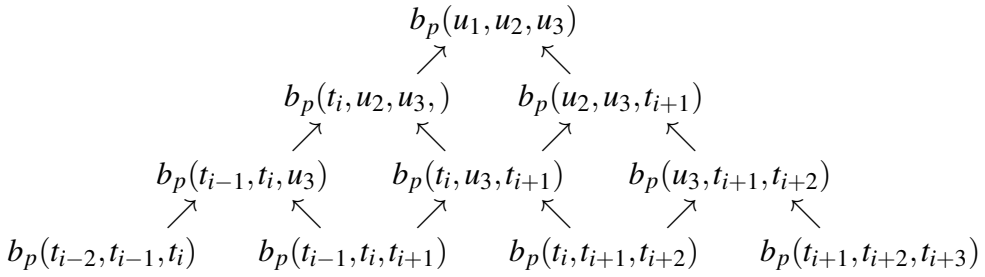
Multiaffin: $b_p(\ldots,a\,u_i + (1-a)u_i,\ldots) = a\,b_p(\ldots,u_i,\ldots) + (1-a)b_p(\ldots,u_i,\ldots)$

Det følger at blomstringen er en slags faktorisering ned til d affine funksjoner, på en måte det vi ser i matrisenotasjon:

$$b_p(u_1,u_2,u_3) = \begin{pmatrix} 1-w_{1,i}(u_1) & w_{1,i}(u_1) \end{pmatrix} \begin{pmatrix} 1-w_{2,i-1}(u_2) & w_{2,i-1}(u_2) & 0 \\ 0 & 1-w_{2,i}(u_2) & w_{2,i}(u_2) \end{pmatrix}$$
$$\begin{pmatrix} 1-w_{3,i-2}(u_3) & w_{3,i-2}(u_3) & 0 & 0 \\ 0 & 1-w_{3,i-1}(u_3) & w_{3,i-1}(u_3) & 0 \\ 0 & 0 & 1-w_{3,i}(u_3) & w_{3,i}(u_3) \end{pmatrix} \begin{pmatrix} b_p(t_{i-2},t_{i-1},t_i) \\ b_p(t_{i-1},t_i,t_{i+1}) \\ b_p(t_i,t_{i+1},t_{i+2}) \\ b_p(t_{i+1},t_{i+2},t_{i+3}) \end{pmatrix},$$

der $[t_i,t_{i+1}]$ er det aktive domenet, og u_j, $j = 1,2,3$ må være i domenet. Hvis vi organiserer

delresultatene som en pyramide får vi,



6.2.9 Algoritmer for B-splines

Rekursjonsformelen for B-splines ble gitt i Definition 6.3 og tydelig illustrert av matrisenotasjonen i seksjon 6.2.3. En algoritme vil derfor være å multipisere faktormatrisene fra venstre mot høyre, men hoppe over elementene som er null. I den følgende algoritmen vil vi fylle ut en vektor med $d+1$ reelle tall, en for hver av de aktive B-splines (basisfunksjoner) ved gjeldende intervall $[t_i, t_{i+1})$ når $t_i \leq t < t_{i+1}$.

Algoritme 3. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.)*
Algoritmen regner ut en vektor $\mathbf{b}_d(t) \in \mathbb{R}^{d+1}$*, som inneholder verdiene til* $d+1$ *B-splines* $\{b_{d,j}(t)\}_{j=\varsigma-d}^{\varsigma}$*, der* ς *er bestemt av* $t_\varsigma \leq t < t_{\varsigma+1}$*. Innputt er: skjøtvektoren* τ*, polynomgraden til B-splines d, indeksen* ς*, og parameterverdien* $t \in [t_\varsigma, t_{\varsigma+1}]$*.*

vector⟨double⟩ **bspline(** vector⟨double⟩ τ, int d, int ς , double t **)**
vector⟨double⟩ *b(d+1);* // *Returvektoren med* $d+1$ *elementer.*
vector⟨double⟩ *w(d);*
$b_1 = W_{1,\varsigma}(t;\tau)$; // *se (6.11)*
$b_0 = 1 - b_1$; // *Den generelle Cox/deBoor algoritmen*
for *(* int $i = 2$*;* $i \leq d$*;* $i{+}{+}$*)* // *- for B-splines, som regner ut settet*
for *(* int $j = 0$*;* $j < i$*;* $j{+}{+}$ *)* // *- av alle B-splines av grad d for t*
$w_j = W_{i,\varsigma-j}(t;\tau)$; // *- hvor* $\tau_\varsigma \leq t < \tau_{\varsigma+1}$*.*
$b_i = (1 - w_0)\, b_{i-1}$;
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;*

Algoritme 3 er en klassisk optimal Cox-deBoor-algoritme for B-splines.

Ofte vil vi trenge deriverte av 1.-orden, 2.-orden osv. Derfor trengs det en algoritme som ikke bare beregner verdiene til B-splinefunksjonene, men også deriverte av flere ordener. For en grad d polynomfunksjon, eksisterer det d påfølgende deriverte med verider som kan være forskjellige fra null. Derfor trenger vi en $d+1 \times d+1$ matrise $B_{d,\tau}(t)$ av reelle tall for å lagre alle disse verdiene. Vi får $\mathbf{c}(t) = B_{d,\tau}(t)\ \mathbf{c}$, der $\mathbf{c}(t)$ er en vektor med én verdi og d påfølgende deriverte. Legg merke til at $\mathbf{c} = B_{d,\tau}(t)^{-1}\mathbf{g}$, dvs. taylorekspansjon i t, der $\mathbf{g}$ er posisjonen og d påfølgende derivater. For å beregne posisjonen og d-derivertene til en 3.-grads B-splinekurve, har vi følgende formler,

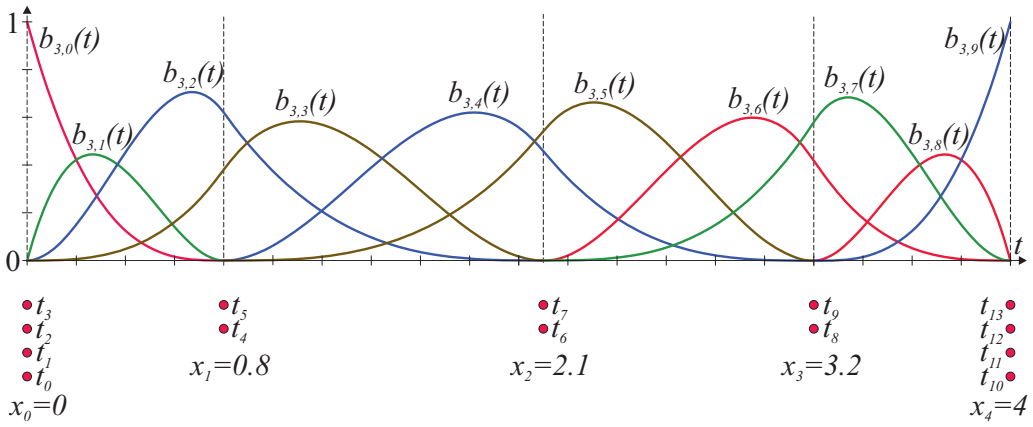
$$\begin{aligned} c(t) &= \mathbf{T}^3(t)\,\mathbf{c} &&= T_1(t)T_2(t)T_3(t)\,\mathbf{c}, \\ c'(t) &= 3\,\mathbf{T}^2(t)\mathbf{T}'\,\mathbf{c} &&= 3\,T_1(t)T_2(t)\,T_3'\,\mathbf{c}, \\ c''(t) &= 6\,\mathbf{T}^1(t)\mathbf{T}'^2\,\mathbf{c} &&= 6\,T_1(t)\,T_2'T_3'\,\mathbf{c}, \\ c'''(t) &= 6\,\mathbf{T}'^3\,\mathbf{c} &&= 6\,T_1'T_2'T_3'\,\mathbf{c}. \end{aligned} \tag{6.17}$$

I den første linja i (6.17) beregner vi den øvre venstre trekanten i matrisen med verdiene til B-splinefungsjonene fra grad 0 til d fra bunnen og oppover. I neste trinn beregner vi T_1' på bunnraden til matrisen, så beregner vi T_2' på først bunnraden og så neste rad, T_3' beregnes de 3 nederste radene og så videre til alle deriverte er beregnet i henhold til (6.17).

Følgende algoritme lager matrisen $\mathbf{B}_{d,\tau}(t,\varsigma)$, der d er polynomgraden, $\tau = \{t_0, t_1, \ldots, t_{n+d}\}$ er skjøtvektoren, $t \in [t_d, t_n]$ er parameterverdien og indeksen ς er hentet fra $t_\varsigma < t \le t_{\varsigma+1}$. B-spline/hermitematrise er definert på samme måte som bernstein/hermitematrisen, beskrevet i seksjon 4.4.3, men hvor Bernstein-faktormatrisene, definert i seksjon 4.4.2, erstattes av matrisene i definisjon 6.4 og 6.5. Matrisen passer i en evaluator, jmf. merknad 3.1.

Algoritme 4. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen regner ut den utvidede kvadratiske matrisen $\mathbf{B}_{d,\tau}(t,\varsigma) \in \mathbb{R}^{d+1\times d+1}$, som inneholder i første raden, verdiene til $d+1$ B-splines $\{b_{d,i}(t)\}_{i=\varsigma}^{\varsigma+d}$, og i de følgende radene verdier for hver av d-deriverte, $\{D^j b_{d,i}(t)\}_{i=\varsigma}^{\varsigma+d}$, $j = 1,2,\ldots,d$. Innput er: skjøtvektoren τ, polynomgraden til B-splinesfunksjonene d, indeksen ς, som er hentet fra $t_\varsigma < t \le t_{\varsigma+1}$, og parameterverdien tß$[t_d, t_n]$.*

matrix⟨double⟩ **BSplineHermiteMat (** vector⟨double⟩ τ, int d, int ς , double t **)**
- matrix⟨double⟩ *B(d+1,d+1);* // *Returmatrisen med $(d+1)\times(d+1)$ elementer.*
- vector⟨double⟩ *w(d);*
- $B_{d-1,1} = W_{1,\varsigma}(t;\tau)$; // *se (6.11)*
- $B_{d-1,0} = 1 - B_{d-1,1}$; // *Den generelle Cox/deBoor algoritmen*
- for *(* int *i=d-2, k=2; $i \ge 0$; $i--$, $k++$)* // *- for B-splines, som regner ut triangelen*
  - for *(* int *j=1; $j < k$; j++)* // *- av alle B-splines av grad*
    - $w_j = W_{k,\varsigma-k+j+1}(t;\tau)$; // *- 1 til d, henholdsvis i hver rad.*
  - $B_{i,0} = (1-w_0)\,B_{i+1,0}$;
  - for *(* int *j=1; $j < d-i$; j++)*
    - $B_{i,j} = w_{j-1}\,B_{i+1,j-1} + (1-w_j)\,B_{i+1,j}$;
  - $B_{i,d-i} = w_{k-1}\,B_{i+1,d-i-1}$;
- $B_{d,1} = \delta_{1,\varsigma}(\tau)$; // *se (6.13)*
- $B_{d,0} = -B_{d,1}$; // *Multipliser alle rader unntatt den øverste*
- for *(* int *k=2; $k \le d$; k++)* // *- med derivertematrisene i*
  - for *(* int *j=0; $j < k$; j++)* // *- definisjon 6.5, slik at hver rad*
    - $w_j = k\,\delta_{k,\varsigma-k+j+1}(\tau)$; // *- til slutt får d+1*
  - for *(* int *$i = d$; $i > d-k$; $i--$)* // *- elementer som er forskjellig fra 0.*
    - $B_{i,k} = w_{k-1}\,B_{i,k-1}$;
    - 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;*



Figur 6.16: *Et plott av B-spline-basisfunksjonene for kubisk hermitesplines. Vi ser 5 interpolasjonspunkt, 14 skjøtverdier og 10 B-splines. Skjøtverdiene er markert som røde kuler. Som vi ser er multiplisiteten til de indre skjøtene 2.*

6.3 Hermitesplineinterpolasjon på B-spline form

I seksjon 5.6, side 68 er kubiske hermitesplines beskrevet på både algebraisk og geometrisk form. Vi skal nå se på kubisk hermitesplines på B-spline form.

Gitt m strengt voksende reelle tall $\{x_i\}_{i=1}^m$, m punkt $\{p_i\}_{i=1}^m$ og m vektorer $\{v_i\}_{i=1}^m$. Vi kan nå vise hvordan en lager en kubisk hermitesplines på B-spline form, dvs.

$$c(t) = T^3(t)\,\mathbf{c}.$$

Kurven skal interpolere punktene og vektorene med de reelle tallene som parameterverdier, dvs. $c(x_i) = p_i$ og $c'(x_i) = v_i$. Vi lager først en skjøtvektor, $\{t_i\}_{i=0}^{n+3}$, hvor $n = 2m$. Vi setter så

$$\begin{aligned}
&\text{i starten:} && t_0 = t_1 = t_2 = t_3 = x_1 \\
&\text{på enden:} && t_n = t_{n+1} = t_{n+2} = t_{n+3} = x_m \\
&\text{og ellers:} && t_i = t_{i+1} = x_j, \quad \text{for} \quad i = 4,6,8,\ldots,2(m-1) \quad \text{og} \quad j = \tfrac{i}{2}.
\end{aligned} \tag{6.18}$$

Vi beregner deretter kontrollpunktene, som vi lager på følgende måte

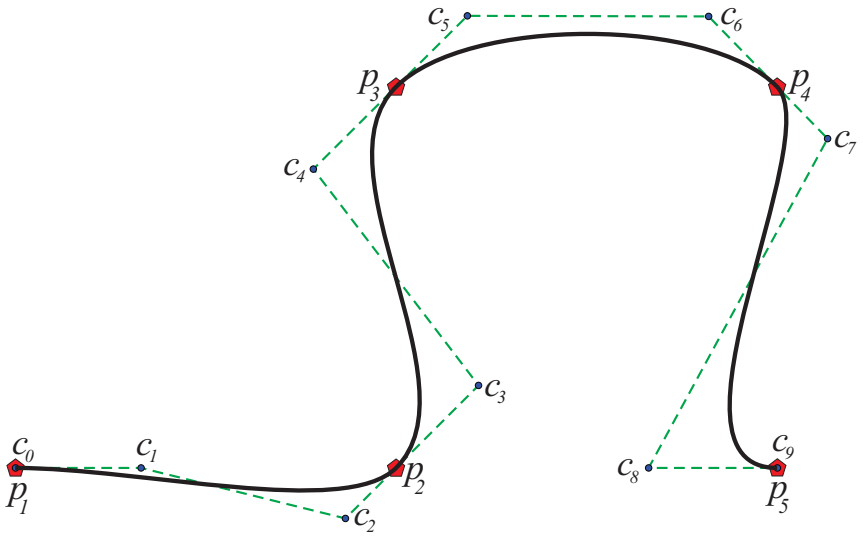
$$c_0 = p_1 \quad \text{og} \quad c_{n-1} = p_m,$$

og deretter

$$\begin{aligned}
c_i &= p_j + \tfrac{\Delta x_j}{6}\, v_j, && \text{for} \quad i = 1,3,\ldots,n-3 \quad \textit{hvor} \quad j = \tfrac{i+1}{2}, \\
c_i &= p_{j+1} - \tfrac{\Delta x_j}{6} v_{j+1}, && \text{for} \quad i = 2,4,\ldots,n-2 \quad \textit{hvor} \quad j = \tfrac{i}{2},
\end{aligned} \tag{6.19}$$

hvor $\Delta x_j = x_{j+1} - x_j$.

Vi kan selvfølgelig generere vektorene $\{v_i\}_{i=1}^m$ ved å bruke enten Cardinal eller Catmull-Rom spline, eller Bessels interpolasjon, eller Akimas interpolasjonsmetode, som alle er beskrevet i seksjon 5.6. I figur 6.16 vises et eksempel på et sett med B-splines (basisfunksjoner) som er konstruert for å hermiteinterpolere $m = 5$ punkt. Vi ser i figuren og kjenne



Figur 6.17: *Kurven, en Catmull-Rom splines, er en hermiteinterpolasjon på B-spline form av 5 punkt $\{p_i\}_{i=1}^{5}$ med tilhørende vektorer $\{v_i\}_{i=1}^{5}$. Kontrollpolygonet er plottet som stiplet grønne linjer. Kontrollpunktene $\{c_i\}_{i=0}^{9}$ er blå, og interpolasjonspunktene $\{p_i\}_{i=0}^{4}$ er plottet i rødt. Den resulterende kurven er i heltrukken svart.*

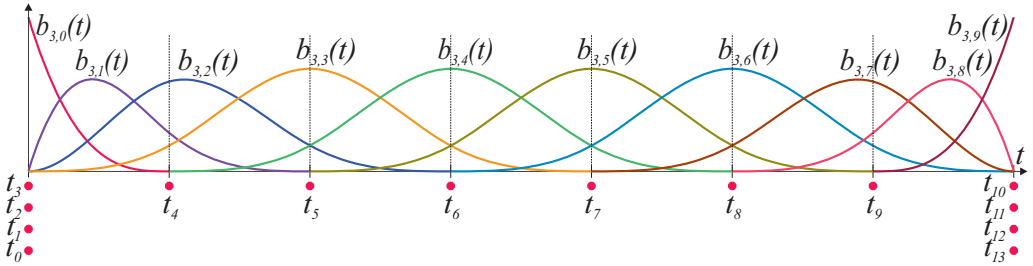
også igjen fra uttrykkene i (6.18) at alle indre skjøter har multiplisitet 2. Dette fordi en hermitesplines er C^1-glatt. En 3.-grads B-spline med enkle interne skjøter er $C^2[t_d,t_n]$, så for å være bare $C^1[t_d,t_n]$ må multiplisiteten være 2. Dette sikrer også hermiteinterpolsjonen. I figur 6.16 ser vi at det internt er 5 klynger av skjøtverdier, alle med multiplisitet 2 og i endene er multiplisiteten 4. Verdien i hver av disse klyngene er parameterverdien til interpolasjonspunkt, dvs. $c(x_i) = p_i$. Vi kan se i figur 6.16 at over de indre skjøtene er det bare to B-splines som er forskjellig fra null. Dette sikrer at interpolasjonspunktene ligger på kontrollpolygonet, den rette linjen mellom to kontrollpunkt, og det er hvordan vi faktisk konstruerer kontrollpunkt i (6.19).

Figur 6.17 viser en Catmull-Rom spline hvor vektorene $v_i = \frac{1}{2}(p_{i+1} - p_{i-1})$ og x-verdiene er $\{0, 0.8, 2.1, 3.2, 4\}$ som vi ser i figur 6.16. I figur 6.17 kan vi se en heltrukken svart B-splinekurve sammen med kontrollpolygonet plottet i stiplet grønt. De 10 kontrollpunktene $\{c_i\}_{i=0}^{9}$ er merket som blå kuler og de 5 interpolasjonspunktene $\{p_i\}_{i=1}^{5}$ er merket med røde stjerner. Tangentvektorene er $v_0 = (p_2 - p_1)$, $v_1 = \frac{1}{2}(p_3 - p_1)$, $v_2 = \frac{1}{2}(p_4 - p_2)$, $v_3 = \frac{1}{2}(p_5 - p_3)$ og $v_4 = v_0$. Hvis vi introduserer spenningsparametre, er eksemplet også potensielt en kardinalspline, også kalt en kanonisk spline.

Hvis vi introduserer spenningsparametere $\{\rho_i\}_{i=1}^{n}$, må vi endre (6.19) til

$$
\begin{aligned}
c_i &= p_j + \tfrac{\Delta x_j\, \rho_j}{6}\, v_j, && \text{for} \quad i = 1,3,\ldots,n-3 \quad \textit{hvor} \quad j = \tfrac{i+1}{2},\\
c_i &= p_{j+1} - \tfrac{\Delta x_j\, \rho_{j+1}}{6} v_{j+1}, && \text{for} \quad i = 2,4,\ldots,n-2 \quad \textit{hvor} \quad j = \tfrac{i}{2}.
\end{aligned}
$$

Også Bessels spline eller Akimas metode beskrevet i seksjon 5.6 kan brukes til å generere tangentvektorene.



Figur 6.18: *Figuren viser de 10 3.-grads B-splines som genereres av en skjøtvektor med 14 elementer, $\{t_i\}_{i=0}^{13}$, som brukes til kubisk splineinterpolering av 8 punkt. Som vi kan se er bare 3 B-splines forskjellig fra null i de interne skjøtene. Derfor er matrisen A i ligningen i (6.20) 3-diagonal.*

6.4 Kubisk splineinterpolasjon på B-spline form

I seksjon 5.7 beskrives en kubisk splineinterpolasjon. Vi skal her se på kubisk splineinterpolasjon på B-spline form.

I seksjonen 5.7 brukte vi hermiteinterpolasjon som utgangspunkt selv om kurven var C^2-glatt. Dette var grunnen til at de ukjente var de 1.-deriverte i interpolasjonspunktene. Når vi nå har B-splines, er det mer naturlig at de ukjente er koeffisientene, dvs. kontrollpunktene.

Gitt m strengt økende reelle tall $\{x_i\}_{i=1}^m$ og m punkt $\{p_i\}_{i=1}^m$. Vi skal nå konstruere en 3.-grads B-splinekurve,

$$c(t) = T^3(t)\,\mathbf{c}.$$

Kravet er så at $c(t)$ interpolerer de gitte punktene ved de gitte tallene, dvs. $c(x_i) = p_i$, $i = 1,2,\ldots,m$. Fordi vi skal bruke 3.-grads B-splines, følger det at antall skjøtverdier må være $m+6$, dette fordi vi ønsker enkle skjøter internt siden kurven skal være C^2-glatt, og at vi må ha 4 like skjøtverdier ved start og ved slutt. Siden antall skjøter for en 3.-grads B-splinekurve er lik antall kontrollpunkt pluss ordenen $k = 4$, følger det at antall kontrollpunkt må være $n = m+2$. Skjøtvektoren blir da:

$$\begin{aligned}
&\text{i starten:} && t_0 = t_1 = t_2 = t_3 = x_1 \\
&\text{til slutt:} && t_n = t_{n+1} = t_{n+2} = t_{n+3} = x_m \\
&\text{og ellers:} && t_i = x_{i-2}, \quad \text{for} \quad i = 4,5,\ldots,n-2,n-1.
\end{aligned}$$

Figur 6.18 viser skjøtvektoren som blir generert av en gitt x-vektor og et plott av de B-splines den genererer. Når graden er 3 må det være 4 like skjøtverdier ved start og slutt. Det gjør at kontrollpunktene ved start og ved slutt er lik interpolasjonspunkt, dvs.

$$c_0 = p_1 \quad \text{og} \quad c_{n-1} = p_m.$$

For å beregne de resterende kontrollpunktene må vi løse et lineære ligningssystem som ligner på ligning (5.16) i seksjonen 5.7, dvs.

$$\mathbf{A}\,\mathbf{c} = \mathbf{b} \tag{6.20}$$

hvor

$$\mathbf{A} = \begin{pmatrix} B''_{3,1}(t_3) & B''_{3,2}(t_3) & 0 & \cdots & \cdots & 0 \\ B_{3,1}(t_4) & B_{3,2}(t_4) & B_{3,3}(t_4) & 0 & \ddots & \vdots \\ 0 & B_{3,2}(t_5) & B_{3,3}(t_5) & B_{3,4}(t_5) & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & B_{3,n-4}(t_{n-1}) & B_{3,n-3}(t_{n-1}) & B_{3,n-2}(t_{n-1}) \\ 0 & \cdots & \cdots & 0 & B''_{3,n-3}(t_n) & B''_{3,n-2}(t_n) \end{pmatrix},$$

og

$$\mathbf{c} = \begin{pmatrix} c_1 \\ \vdots \\ \vdots \\ \vdots \\ c_{n-2} \end{pmatrix} \quad \text{og} \quad \mathbf{b} = \begin{pmatrix} -B''_{3,0}(t_3)\, c_0 \\ p_2 \\ \vdots \\ p_{m-1} \\ -B''_{3,n-1}(t_n)\, c_{n-1} \end{pmatrix}.$$

I denne ligninga er det brukt fri-ende-betingelse, som betyr at den 2.-deriverte er null i starten og i slutten (se (5.20) i seksjon 5.7). Alle rader i $\mathbf{A}$, unntatt den første og siste, kan lages med å bruke Algoritmen 3. Den første og siste raden kan lages med bruk av Algoritmen 4, for så å bruke de 2 første og 2 siste elementene fra tredje rad i resultatmatrisen.

Figur 6.18 viser et eksempel på B-splines i kubisk splineinterpolasjon, her er $m = 8$ (det er 8 klynger med skjøtverdier). Det følger at det er $n = 10$ B-splines, og 14 skjøtverdier (markert med røde kuler). Til venstre i figur 6.18 er det bare 4 B-splines $\neq 0$ i første segment, $B_{3,0}$, $B_{3,1}$, $B_{3,2}$ og $B_{3,3}$. Men siden $B''_{3,3}(t_3) = 0$, får vi et uttrykk med 3 ledd som beskriver 2.-deriverte ved start. På slutten av kurven ser vin det samme, bare speilvendt. Dermed får vi følgende betingelser:

$$\begin{aligned} c''(t_3) &= B''_{3,0}(t_3)\, c_0 + B''_{3,1}(t_3)\, c_1 + B''_{3,2}(t_3)\, c_2 = 0 \\ c''(t_{10}) &= B''_{3,7}(t_{10})\, c_7 + B''_{3,8}(t_{10})\, c_8 + B''_{3,9}(t_{10})\, c_9 = 0, \end{aligned}$$

og hvis vi reorganiserer disse to uttrykkene får vi,

$$\begin{aligned} B''_{3,1}(t_3)\, c_1 + B''_{3,2}(t_3)\, c_2 &= -B''_{3,0}(t_3)\, c_0 \\ B''_{3,7}(t_{10})\, c_7 + B''_{3,8}(t_{10})\, c_8 &= -B''_{3,9}(t_{10})\, c_9 \end{aligned}$$

Disse to restriksjonene blir da den første og den siste linjen i matrise A i (6.20). Som vi ser i figur 6.18 er bare tre B-splines forskjellige fra null i alle interne skjøter. For så å interpolere i disse interne skjøtene, følger det at (som vi kan se fra figuren):

$$c(t_i) = B_{3,i-3}(t_i)\, c_{i-3} + B_{3,i-2}(t_i)\, c_{i-2} + B_{3,i-1}(t_i)\, c_{i-1} = p_{i-2},$$

som er det alle de andre linjene (untatt første og siste) i matrisen A i ligning (6.20) viser.

6.5 B-splineapproksimasjon og minste kvadrater

Som vi så i kubisk splineinterpolasjon, er antall interpolasjonspunkt lik antall kontrollpunkt minus punktene ved start og slutt. Det vil si at systemet er bestemt og kan løses. Hvis antallet interpolasjonspunkt derimot er større enn frihetsgradene, kan vi ikke interpolere og vi får ingen kvadratisk matrise. Så hva kan gjøres? En mulighet er å bruke **minste kvadraters metode**.

Så gitt m strengt økende reelle tall $\{x_i\}_{i=1}^m$ og m punkt $\{p_i\}_{i=1}^m$. Vi kan nå lage en B-splinekurve av grad d,

$$c(t) = T^d(t)\,\mathbf{c},$$

hvor antall kontrollpunkt er $n < m$. I motsetning til kubisk splineinterpolasjon, har vi nå frihet til å lage en skjøtvektor delvis uavhengig av x_i-verdiene, men skjøtvektoren må lages slik at det finnes minst en x_i verdi internt i domenet til hver eneste B-spline (basis). Vi har så følgende optimale ønske for hvert av de m oppgitte punktene,

$$c(x_i) = \sum_{j=0}^{n-1} c_j b_{d,j}(x_i) = p_i, \quad i = 1,2,\ldots,m.$$

Hvis vi formulerer dette i en matrise/vektor-ligning, får vi

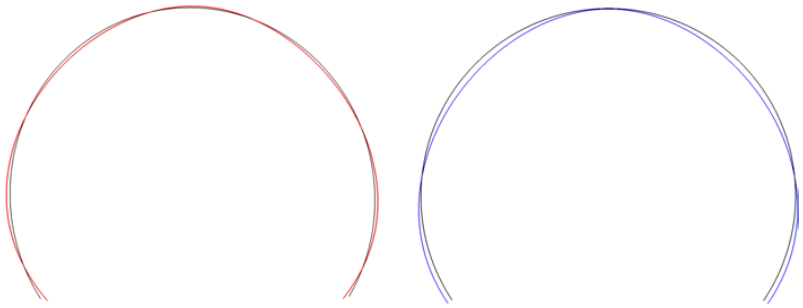
$$\mathbf{A}\,\mathbf{c} = \mathbf{p}. \tag{6.21}$$

Her er $\mathbf{A}$ en $m \times n$-matrise der hver rad har maksimalt $d+1$ elementer som ikke er null. Vi skal se på et eksempel, en 2.-grads B-splinekurve hvor vi har laget en skjøtvektor der $t_0 = t_1 = t_2 = x_1$, $t_n = t_{n+1} = t_{n+2} = x_m$, $t_3 > x_2$ og $t_{n-1} < x_{m-1}$. Elementene i ligningen blir,

$$\mathbf{A} = \begin{pmatrix} B_{2,0}(x_1) & 0 & \cdots & \cdots & \cdots & 0 \\ B_{2,0}(x_1) & B_{2,1}(x_1) & B_{2,2}(x_1) & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & B_{2,1}(x_2) & B_{2,2}(x_2) & B_{2,3}(x_2) & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & B_{2,n-4}(x_{m-1}) & B_{2,n-3}(x_{m-1}) & B_{2,n-2}(x_{m-1}) & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \cdots & 0 & B_{2,n-3}(x_m) & B_{2,n-2}(x_m) & B_{2,n-1}(x_m) \\ 0 & \cdots & \cdots & \cdots & 0 & B_{2,n-1}(x_m) \end{pmatrix}, \tag{6.22}$$

og hvor

$$\mathbf{c} = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{pmatrix} \quad \text{og} \quad \mathbf{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_{m-1} \\ p_m \end{pmatrix}.$$



Figur 6.19: *Til venstre er en del av en sirkel plottet i svart, og en 3.-grads B-splinekurve i rødt. Den er laget av 60 punkt på sirkelen og bruk av minste kvadraters metode. Til høyre er sirkeldelen i svart og en 3.-grads B-spline kurve i blått. Denne er laget med minste kvadraters metode inkludert glatting med* $\alpha = 1$.

Alle rader i $\mathbf{A}$ kan lages ved hjelp av Algoritmen 3. En av parameterne i algoritme 3 er ς. I rad nummer i i $\mathbf{A}$ bruker vi x_i, $i = 1,\ldots,m$ som parameter til basisfunksjonene. Det følger at ς kommer fra $t_\varsigma \leq x_i < t_{\varsigma+1}$. Hvis $i = m$ er $\varsigma = n-1$.

Siden matrisen $\mathbf{A}$ ikke er kvadratisk og dermed ikke inverterbar, kan vi ikke løse den som den er. Så hva gjør vi? Vi tar utgangspunkt i min $|\mathbf{A}\,\mathbf{c}-\mathbf{p}|^2$, deriverer med hensyn på $\mathbf{c}$, og finner når den er 0. Det vil si $\frac{d}{d\mathbf{c}}(|\mathbf{Ac}|^2 - 2\mathbf{pAc} + |\mathbf{p}|^2) = 0$, og som gir $2\mathbf{A}^T\mathbf{Ac} - 2\mathbf{A}^T\mathbf{p} = 0$. Dermed, for å finne minste kvadraters uttrykket, må vi løse

$$\mathbf{Bc} = \mathbf{y}, \quad \text{hvor} \quad \mathbf{B} = \mathbf{A}^T\mathbf{A} \quad \text{og} \quad \mathbf{y} = \mathbf{A}^T\mathbf{p}. \tag{6.23}$$

Hvis $m = n$, kan (6.21) løses direkte og vi har en interpolasjon, men som er forskjellig fra klassisk kubisk splineinterpolasjon. Hvis $m > n$ og typisk mye større, vil (6.23) lage en ligning med en $n \times n$ matrise $\mathbf{B} = \mathbf{A}^T\mathbf{A}$ og en vektor med n punkt $\mathbf{y} = \mathbf{A}^T\mathbf{b}$, dvs. en ligning som er lett å løse. Dette fordi matrisen er symmetrisk rundt hoveddiagonalen og også diagonaldominant fordi den mest innflytelsesrike basisen er på diagonalen. Matrisen kalles positiv-definite (definert av $\mathbf{x}^T\mathbf{Ax} > 0$ for enhver vektor $\mathbf{x}$ som ikke er null), og kan løses bedre og raskere ved å bruke Cholesky- eller QR-faktorisering i stedet for LU-faktorisering. Se avsnitt om BLAS - B.1, eller numeriske beregninger i [78].

I figur 6.19 er et eksempel plottet. Det er et stykke av en sirkel samplet i 60 punkt. Disse punktene sammen med en vektor av parameterverdier brukt i samplingen er så brukt til å lage en 3.-grads B-splinekurve. Denne kurven er klemt og har ellers en uniform skjøtvektor. Til venstre i figuren er B-splinekurven laget ved å bruke en rein minste kvadratmetoden og med 5 kontrollpunkt. I figuren er sirkelbuen svart mens B-splinekurven er rød. En nærmere studie av disse to kurvene viser at B-splinekurven oscillerer rundt sirkelbuen. Dette er typisk for interpolasjon og også approksimasjon ved bruk av minste kvadraters metode. Det er et resultat av å balansere feilen.

Graden til en B-splinekurve begrense formmuligheten til kurven. For å illustrere problemet kan vi bare tegne en frihåndskurve på et ark, og så stille spørsmålet; er det mulig å finne formelen for denne kurven? Svaret er ja, men det er en hake med det, dimensjonen til funksjonsrommet blir uendelig. Hvis vi reduserer denne dimensjonen til 4, dvs. en 3.-grads B-splinekurve, får vi en approksimasjon. Hvis vi tvinger kurven til å gå gjennom

et sett med punkt eller optimalt så nært som mulig vil kurven nødvendigvis oscillere i forhold til en utgangskurve.

Ofte vil vi ha en glatt kurve som er tilstrekkelig nær. Vi kan da legge til et ekstra ledd i ligningen, dvs. minimere kvadratet av krumningen, eller det som er enklere, kvadratet av 2*re*-deriverte. Så i tillegg får vi $c''(t) = \sum_{i=0}^{n-1} b''_{d,i}(t)c_i$, som gir min $|\mathbf{Dc}|^2$, der matrisen $\mathbf{D}$ er lik matrisen $\mathbf{A}$ bortsett fra at basisfunksjonene erstattes av 2.-deriverte av basisfunksjonene. Vi får dermed $\mathbf{Bc} = \mathbf{b}$, hvor $\mathbf{B} = \mathbf{A}^T\mathbf{A} + \alpha\ \mathbf{D}^T\mathbf{D}$ og $\mathbf{b} = \mathbf{A}^T\mathbf{p}$. Her er α en skalar som bestemmer vekten av glattingen. På høyre side i figur 6.19 kan vi se den samme approksimasjonen i blått som den til venstre, men her er det lagt på en glatting med $\alpha = 1$.

Minste kvadrater og B-splines

Gitt m punkt $\mathbf{p}$, og m parameterverdier $\mathbf{x}$. Vi lager så en B-spline kurve av grad d, med $n < m$ kontrollpunkt og en skjøtvektor τ, der alle x-verdiene er i domenet, og hvor domenet til hver B-spline inkluderer minst én x-verdi. Vi får da

$$\mathbf{Bc} = \mathbf{y}, \quad \text{hvor} \quad \mathbf{B} = \mathbf{A}^T\mathbf{A} + \alpha\ \mathbf{D}^T\mathbf{D} \quad \text{og} \quad \mathbf{y} = \mathbf{A}^T\mathbf{p}.$$

$m \times n$ matrisen $\mathbf{A}$ er i henhold til (6.22), og som også gjelder for matrisen $\mathbf{D}$, men hvor B-splines er erstattet med 2.-deriverte av B-splines. α er en skalar som bestemmer glattingen. Hvis $\alpha = 0$ har vi en vanlig minste-kvadraters metode og kan bruke algoritme 3 for å generere $\mathbf{A}$. Ellers er det en minste kvadratisk glattingsmetode og vi må bruke algoritme 4 - 1. rad for å generere $\mathbf{A}$ og 3. rad for å generere $\mathbf{D}$.

6.6 NURBS

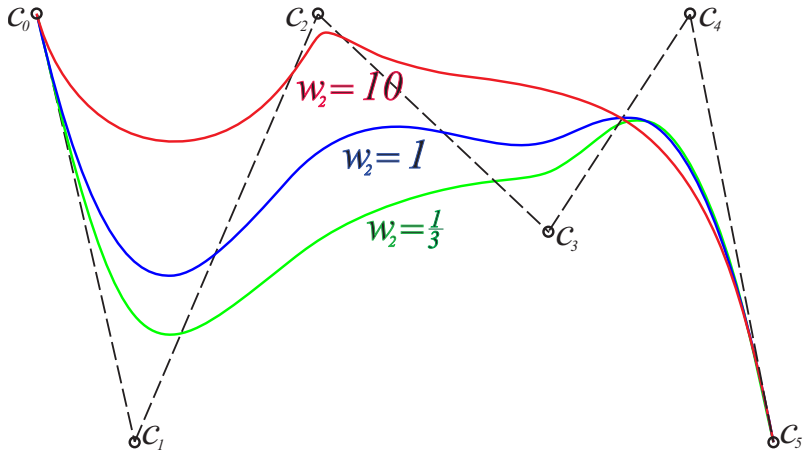
NURBS er en forkortelse for “non uniform rational B-splines". Ikke uniforme B-splines er de mest vanlige B-splines, og er definert av skjøter som kan være vilkårlig fordelt. Dette i motsetning til uniform B-splines som ble definert (på en implisitt heltallsskjøtvektor) av Schoenberg i [138, 139]. For å beskrive rasjonale B-splines er det nødvendig å kjenne til homogene koordinater, ofte brukt i grafiske systemer som OpenGL. Et homogent koordinatsystem er knyttet til det projektive rommet $\mathbb{P}^n$ [7]. En konkret beskrivelse er at $\mathbb{P}^n$ kan være definert som rommet til alle uendelige rette linjer i $\mathbb{R}^{n+1}$ som går gjennom origo. Forskjellen i forhold til kartesiske koordinater er at vi får én ekstra koordinat sammenlignet med en tilsvarende euklidske/affine-rom[8]. Vi har $q = (x, y, z, w)$, når q er et element i et 3D-rom. Bruker denne beskrivelse det følger at $q \in \mathbb{P}^3$ kan uttrykkes ved

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

hvor q er uavhengig av k, dvs. k kan være en hvilken som helst reell som ikke er null.

[7] Projektive rom er beskrevet i seksjon 2.5 og homogene koordinater i seksjon 2.6. For en grundigere studie, se f.eks. [11] eller [30].

[8] Affine rom er omtalt i seksjon 2.4. Affine rom er rom av punkt med tilhørende vektorer. Punktene i affine rom er uavhengig av origo, og origo er bare ett av punktene på lik linje med alle andre. Se også http://en.wikipedia.org/wiki/Affine_space eller https://encyclopediaofmath.org/wiki/Affine_space.



Figur 6.20: *Tre versjoner av en rasjonal B-splimekurve (NURBS) av grad 3. Det er bare vekten, w_2, knyttet til kontrollpunktet c_2, som har ulike verdier i de tre eksemplene. Kontrollpolygonet og skjøtvektoren er det samme for alle tre kurvene.*

Det er en kanonisk injeksjon av $\mathbb{R}^n$ inn i $\mathbb{P}^n$. Dette betyr at et affint rom $\mathbb{R}^n$ kan være inkludert i $\mathbb{P}^n$ etter standardovergangen

$$(x_1,\ldots,x_n) \mapsto (x_1,\ldots,x_n,1).$$

Affine punkt kan gjenopprettes fra projektive punkt med overgangen

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

Definisjon 6.7. *Ikke-uniforme rasjonale B-splines (NURBS) er vanlige ikke uniforme B-splines i det projektive rommet, men som så overføres til et affint rom. Det følger da at en B-spline i $\mathbb{P}^n$ er*

$$c(t) = T^d(t)\,\mathbf{c},$$

hvor hvert element c_j, $j = i-d, i-d+1,\ldots,i$ av $\mathbf{c}$ er gitt i homogene koordinater

$$c_j = w_j(x_{j,1},\ldots,x_{j,n},1).$$

Til slutt, selve overgangen fra det projektive rommet $\mathbb{P}^n$ til et affint rom $\mathbb{R}^n$ blir da

$$c(t) = \frac{T^d(t)\,\mathbf{c}}{T^d(t)\,\mathbf{w}}, \tag{6.24}$$

hvor $\mathbf{w} = (w_{i-d}, w_{i-d+1},\ldots,w_i)^T$. Formelen 6.24 er da NURBS-definisjonen.

Det følger av uttrykk (6.24) at hvis $w_i = 1$ for alle i, så er NURBS en vanlig B-spline funksjon i et affint rom. For å skissere kontrollpolygonet, dvs. koeffisientene c_i, $i = 0,\ldots,m-1$, der m er antall koeffisienter, må punktene overføres fra $\mathbb{P}^n$ til $\mathbb{R}^n$, dvs.

$$w_i(x_{i,1},\ldots,x_{i,n},1) \mapsto (x_{i,1},\ldots,x_{i,n}).$$

Figur 6.20 viser en rasjonal 3.-grads B-spline kurve plottet 3 ganger hvor vekten av den tredje koeffisienten er $w_2 = \frac{1}{3}$, $w_2 = 1$ og $w_2 = 10$. Effekten er tydelig demonstrert - som vi ser vil en liten vekt skyve kurven fra kontrollpunktet, mens en stor vekt trekker den mot kontrollpunktet.

6.7 Uniforme B-splines og subdivisjon

Uniforme B-splines med heltalls-skjøtvektor , beskrevet i (6.2) og (6.3), var den første typen B-splines utviklet av Schoenberg. Spesielt symmetrien og det faktum at alle basisfunksjoner er like, bare translatert i forhold til hverandre, medfører at skjøtinnsetting i midten mellom skjøtene kan forenkles svært mye slik at vi får et enkelt skjema. Skjøtvektoren brukes egentlig bare for å lage skjemaet. Etter en runde med innsettinger kan vi tenke oss en reparametrisering til heltallsskjøtverdier igjen, slik at vi med neste subdivisjon kan bruke det samme skjemaet. Dette fører til subdivisjonskurver.

Subdivisjon er egentlig vanlig hjørnekutting, hvor et punktsett erstattes med et nytt større punktsett der de nye punktene ligger på linjene mellom punktene i det forrige settet. Men subdivisjon kan også være interpolasjon, hvor de nye punktene legges til det første punktsettet, og hvor de nye punktene ikke nødvendigvis er inne i den konvekse omhyldningen til det originale punktsettet. I de følgende seksjonene skal vi se på begge typer subdivisjon, og vi starter med interpolasjon.

6.7.1 Catmull-Rom subdivisjonssplines

Catmull-Rom splines er omtalt i avsnitt 5.6, side 69, og senere vist i figur 6.17, side 99. Catmull-Rom splines interpolerer et sett med punkt på en slik måte at de 1.-deriverte i hvert punkt er lik vektoren som går fra punktet før til punktet etter det gitte punktet, men da skalert med $\frac{1}{2}$. Hvis vi bruker en uniform heltallsbasert skjøtvektor, da kan Catmull-Rom splines utvikles som en interpolerende C^1-glatt subdibisjonskurve, [20].

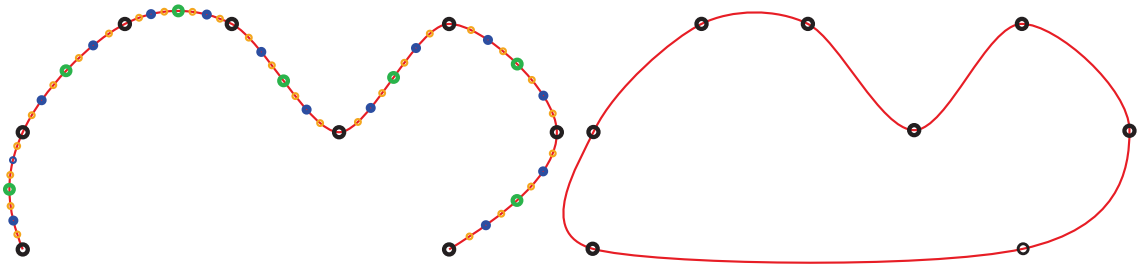
Gitt skjøtverdiene $\{-1,0,1,2\}$ og punktene p_{i-1}, p_i, p_{i+1} og p_{i+2}, på lagrangepolynomform får vi da en kurve som interpolerer punktene på formen,

$$\begin{aligned} c(t) &= L_{3,0}(t)p_{i-1} + L_{3,1}(t)p_i + L_{3,2}(t)p_{i+1} + L_{3,3}(t)p_{i+2} \\ &= \frac{t(t-1)(t-2)}{-1(-2)(-2)}p_{i-1} + \frac{(t+1)(t-1)(t-2)}{1(-1)(-2)}p_i \\ &\quad + \frac{(t+1)t(t-2)}{2(1)(-1)}p_{i+1} + \frac{(t+1)t(t-1)}{3(2)(1)}p_{i+2} \\ &= -\frac{1}{6}\left(t^3 - 3t^2 + 2t\right)p_{i-1} + \frac{1}{2}\left(t^3 - 2t^2 - t + 2\right)p_i \\ &\quad - \frac{1}{2}\left(t^3 - t^2 - 2t\right)p_{i+1} + \frac{1}{6}\left(t^3 - t\right)p_{i+2}. \end{aligned} \tag{6.25}$$

Hvis vi så regner ut formelen for Catmull-Rom splines ved $t = \frac{1}{2}$, midtveis mellom punktene p_i og p_{i+1}, får vi et 4-punkts subdivisjonsskjema for Catmull-Rom splines, også kalt Dubuc-Deslaurier subdivisjonsskjema;

$$\widehat{p_i} = -\frac{1}{16}p_{i-1} + \frac{9}{16}p_i + \frac{9}{16}p_{i+1} - \frac{1}{16}p_{i+2}, \tag{6.26}$$

der punktet $\widehat{p_i}$ er et nytt punkt som ligger mellom p_i og p_{i+1}. Fremgangsmåten er ganske klar. Gitt et punktsett. Dette settet kan vi så utvide med å lage nye punkt mellom alle de



Figur 6.21: *Til venstre ser vi et plott av en åpen Catmull-Rom subdivisjonskurve. De svarte sirklene er startpunktene, de grønne punktene er 1.-nivået, de blå er 2.-nivået og de oransje er 3.-nivået med nye punkt. Til høyre vises et plott av en lukket Catmull-Rom subdivisjonskurve med de samme startpunktene som kurven til venstre.*

gamle punktene ved å bruke Dubuc-Deslaurier-subdivisjonsskjemaet. Denne prosessen kan vi så gjenta inntil vi mange nok punkt. Merk at vi kun bruker det gamle punktsettet til å generere de nye punktene, som da skal være mellom de gamle punktene, samt at vi beholder alle de gamle punktene.

Hvis kurven er lukket, dvs. topologisk lik en sirkel, implementerer vi “at hodet biter halen", se algoritme 5. Men hvis kurven er åpen, dvs. inkluderer et start- og et sluttpunkt, da fungerer ikke skjemaet ved start og slutt. Vi kan enten trekke oss vekk fra endene etter hvert som vi går inn i rekursjonen, eller vi kan endre skjemaet i endene. Vi gjør det siste og i utgangspunktet bruker vi da samme formel, (6.25). Indeksen for de første fire punktene er $0,1,2,3$. Fordi vi nå ønsker et nytt punkt mellom p_0 og p_1 endrer vi t-verdien, dvs. vi bruker formel (6.25) med $t = -\frac{1}{2}$, dvs. halvveis mellom punktene p_0 og p_1, og får følgende skjema,

$$\widehat{p}_0 = \frac{5}{16}p_0 + \frac{15}{16}p_1 - \frac{5}{16}p_2 + \frac{1}{16}p_3, \tag{6.27}$$

hvor punktet $\widehat{p}_0$ er et nytt punkt som ligger mellom p_0 og p_1. På slutten kan vi bruke det samme skjemaet, men speilet. Hvis det siste punktet har indeks n, dvs. p_n, så får vi

$$\widehat{p}_{n-1} = \frac{5}{16}p_n + \frac{15}{16}p_{n-1} - \frac{5}{16}p_{n-2} + \frac{1}{16}p_{n-3}, \tag{6.28}$$

hvor punktet $\widehat{p}_{n-1}$ er et nytt punkt som ligger mellom p_{n-1} og p_n.

En spenningsparameter ble lansert i [56]. Dyn et al. introduserer $\omega = \frac{1}{16}$, som gir:

$$\widehat{p}_i = \left(\frac{1}{2} + \omega\right)(p_i + p_{i+1}) - \omega\,(p_{i-1} + p_{i+2}), \tag{6.29}$$

som er en omformulering av (6.26). Men ω kan endres, og Dyn viste i [56] at kurven er C^1-glatt hvis $0 < \omega < \frac{\sqrt{5}-1}{8}$.

En beskrivelse av en algoritme uten spenningsparameter følger nå. Det vil være noen forskjeller mellom en algoritme for å lage en åpen kurve og en algoritme for en lukket kurve. I algoritmebeskrivelsen nedenfor er forskjellene beskrevet etter selve algoritmen, som i utgangspunktet er for en åpen kurve.

Algoritme 5. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen lager en vektor av samplingspunkt til en kurve basert på Dubuc-Deslaurier-subdivisjonsskjema. Det er en algoritme for åpne kurver men kan endres for lukkede kurver. Innput er: startpunktene* $\{P_i\}_{i=0}^{n}$ *og forfiningsnivået* d.

Vector⟨Point⟩ **CatmulRom(** vector⟨Point⟩ *P,* int d **)**

```
int  n = P.size − 1;                          // Antallet intervall
int  m = 2^d n;                               // Det endelige antallet punkt
vector⟨Point⟩  Φ(m+1);                        // Returvektoren - m+1 punkt.
for ( int  i=0; i < P.size; i++ )
    Φ_{2^d i} = P_i;                          // Setter inn de innkommende punktene
for ( int  j=1; j ≤ d; j++ )                  // For hvert nivå av forfining
    int  h = 2^{d−j};
    int  k = 2h;
    Φ_h = 5/16 p_0 + 15/16 p_k − 5/16 p_{2k} + 1/16 p_{3k};        // a) - fra (6.27)
    for ( int  i=1; i < n − 1; i++ )
        Φ_{ki+h} = −1/16 p_{k(i−1)} + 9/16 p_{ki} + 9/16 p_{k(i+1)} − 1/16 p_{k(i+2)};   // b) - fra (6.26)
    Φ_{kn−h} = 5/16 p_{kn} + 15/16 p_{k(n−1)} − 5/16 p_{k(n−2)} + 1/16 p_{k(n−3)}       // c) - fra (6.28)
return Φ;
```

Hvis kurven er lukket, endrer vi antallet intervall til, int $n = P.size;$
etter å ha satt inn de opprinnelige punktene legger vi til $\Phi_{2^d n} = P_0;$
vi erstatter raden merket a) med $\Phi_h = -\frac{1}{16}p_{k(n-2)} + \frac{9}{16}p_k + \frac{9}{16}p_{2k} - \frac{1}{16}p_{3k};$
og vi erstatter raden merket c) med $\Phi_{kn-h} = -\frac{1}{16}p_{2k} + \frac{9}{16}p_{kn} + \frac{9}{16}p_{k(n-1)} - \frac{1}{16}p_{k(n-2)};$

Husk at i programmeringsspråk av C-familien, kan 2^d implementeres med $1 << d$.

På venstre side i figur 6.21 vises en åpen Catmull-Rom subdivisjonskurve laget ved å bruke algoritmen som er beskrevet ovenfor. Vi starter med $n+1$ "svarte punkt" hvor $n = 7$. Dette er de opprinnelige punktene. Vi bruker så 3 forfiningsnivå, dvs. $d = 3$. Vi følger beskrivelsen ovenfor og finner at $m = 2^3 7 = 56$, så vi vil ha 57 punkt når hele prosessen er fullført. Vi kopierer de 8 originalpunktene til $q_0, q_8, q_{16}, q_{24}, q_{32}, q_{40}, q_{48}$ og q_{56}. På 1.-nivå er dermed $\widehat{k} = 4$ og $k = 8$. I samsvar med algoritmen legges det nå til 7 nye "grønne punkt" med indeksene $4, 12, 20, 28, 36, 44$ og 52. På 2.-nivå legger vi til 14 nye "blå punkt" med indeksene $2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50$ og 54. Til slutt på 3.-nivå setter vi inn 28 nye "oransje punkt" med indeksene $1, 3, 5, ..., 51, 53$ og 55. Som vi kan se til venstre i figur 6.21 er det totale antall poeng 57. Hvis vi hadde brukt 4 forfiningsnivå, ville det totale antallet vært 113.

Til høyre i figur 6.21 er et plott av en lukket Catmull-Rom subdivisjonskurve laget med å bruke den samme sett med punkt som kurven på venstre side i figuren.

6.7.2 Chaikin's algoritme, 2.-grads subdivisjonssplines

Vi skal nå se på den enkleste prosedyren for hjørnekutting. George Chaikin holdt en forelesning ved University of Utah i 1974, hvor han kom med en ny prosedyre for å generere kurver med utgangspunkt i et begrenset sett med startpunkt [21]. Denne algoritmen

var blant de første forfiningsalgoritmene basert på hjørnekutting. Algoritmen forfiner et punktsett på en slik måte at det konvergerer mot en glatt kurve. Algoritmen er avledet fra en 2.-grads uniform B-splines [136]. Vi tar utgangspunkt i matrisen fra skjøtinnsetting, uttrykk (6.15), og legg inn en ny skjøt midt mellom t_i og t_{i+1},

$$\begin{pmatrix} q_{2i-1} \\ q_{2i} \end{pmatrix} = \begin{pmatrix} 1-w_{2,i-1}(\frac{t_i+t_{i+1}}{2}) & w_{2,i-1}(\frac{t_i+t_{i+1}}{2}) & 0 \\ 0 & 1-w_{2,i}(\frac{t_i+t_{i+1}}{2}) & w_{2,i}(\frac{t_i+t_{i+1}}{2}) \end{pmatrix} \begin{pmatrix} p_{i-1} \\ p_i \\ p_{i+1} \end{pmatrix},$$

hvor de to nye punktene ligger på hver sin sider av p_i. Hvis vi antar en uniform skjøtvektor av heltall, da er

$$w_{2,i-1}\left(\frac{t_i+t_{i+1}}{2}\right) = \frac{\frac{1}{2}-(-1)}{2} = \frac{3}{4} \qquad \text{og} \qquad w_{2,i}\left(\frac{t_i+t_{i+1}}{2}\right) = \frac{\frac{1}{2}-0}{2} = \frac{1}{4}$$

og hele punktsettet p_i kan erstattes av følgende sett med to nye punkt i hvert av intervallene p_i, p_{i+1},

$$\begin{aligned} q_{2i} &= \frac{3}{4}p_i + \frac{1}{4}p_{i+1} \qquad \text{og} \\ q_{2i+1} &= \frac{1}{4}p_i + \frac{3}{4}p_{i+1}. \end{aligned} \tag{6.30}$$

Hvis antall intervall er n så er antallet punkt for en åpen kurve $n+1$ og for en lukket kurve n. Antall punkt vi får etter en runde med hjørnekutting er altså 2 punkt i hvert intervall, dvs. $2n$. Dette betyr at for åpne kurver er nå antall intervall $2n-1$, men for lukkede kurver er det $2n$. Imidlertid må vi for lukkede kurver legge til en kopi av det første punktet på slutten. Etter d runder med forfining får vi at det endelige antallet med punkt er:

$$m = 2^d(n-1)+2, \quad \text{for åpne kurver, og} \qquad m = 2^d n + 1, \quad \text{for lukkede kurver.} \tag{6.31}$$

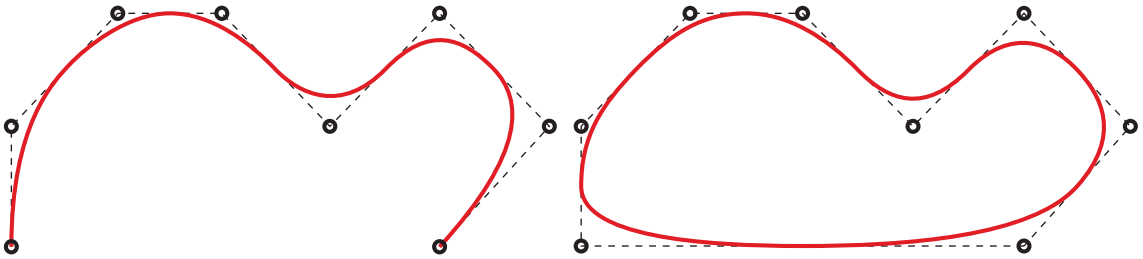
På grunn av (6.30) ser vi at for en åpen kurve vil startpunktet flyttes fra p_0 mot p_1, på samme måte vil endepunktet flyttes fra p_n mot p_{n-1}. Vi kan imidlertid beregne hvor mye start- og sluttpunktene vil bevege seg. Fra (6.30) ser vi at etter den første hjørnekuttingen er startpunktet flytte $x = \frac{1}{4}(p_1 - p_0)$. I neste trinn vil den flyttes $x = \frac{1}{4}\frac{1}{2}(p_1 - p_0)$, og slik fortsetter det slik at for d forfiningsnivå får vi,

$$x = \sum_{i=1}^{d} \frac{1}{2^{i+1}}, \quad \text{hvor } x \text{ er en faktor som forteller hvor endepunktene ender.} \tag{6.32}$$

En løsning på dette er å flytte start- og sluttpunktene slik at første og siste punkt i punktsettet vil være ved start og slutt på kurven etter forfiningen. Merk at dette kun er mulig for en 2.-grads subdivisjonskurver. Dette fordi en 2.-grads B-splinekurve berører kontrollpolygonet i skjøtverdier. Dermed får vi,

$$p_0 = p_0 + \frac{x}{1-x}(p_0 - p_1) \quad \text{og} \quad p_n = p_n + \frac{x}{1-x}(p_n - p_{n-1}) \tag{6.33}$$

Dette subdivisjonskjemaet er det samme som vi finner i Doo-Sabin flatekonstruksjon, [54, 55], og det er derfor naturlig å kalle kurven som en Doo-Sabin kurve. Figur 6.22 viser et eksempel på en Doo-Sabin-kurver, en åpen kurve til venstre og en lukket kurve til høyre. Her følger nå algoritmene.



Figur 6.22: *Hjørnekutting i subdivisjonskurver ved bruk av Chaikin's algoritme, laget av samme sett med punkt som kurven i figur 6.21. På venstre side er en åpen kurve plottet og på høyre side er en lukket kurve plottet. De svarte sirklene er de initiale punktene.*

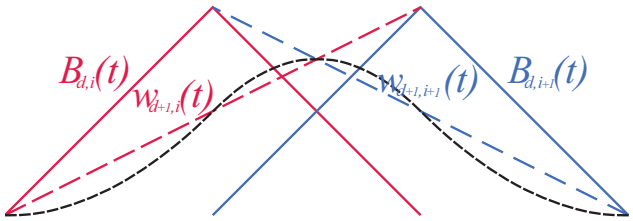
Algoritme 6. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen lager en vektor av sampelpunkt til en kurve basert på Chaikin's algoritme. Det er en algoritme for åpne kurver og en for lukkede kurver. Innput er: startpunktsettet* $\{P_i\}_{i=0}^{n}$ *og forfiningsnivået* d.

vector⟨Point⟩ **ChaikinOpen(** vector⟨Point⟩ P, int d **)**

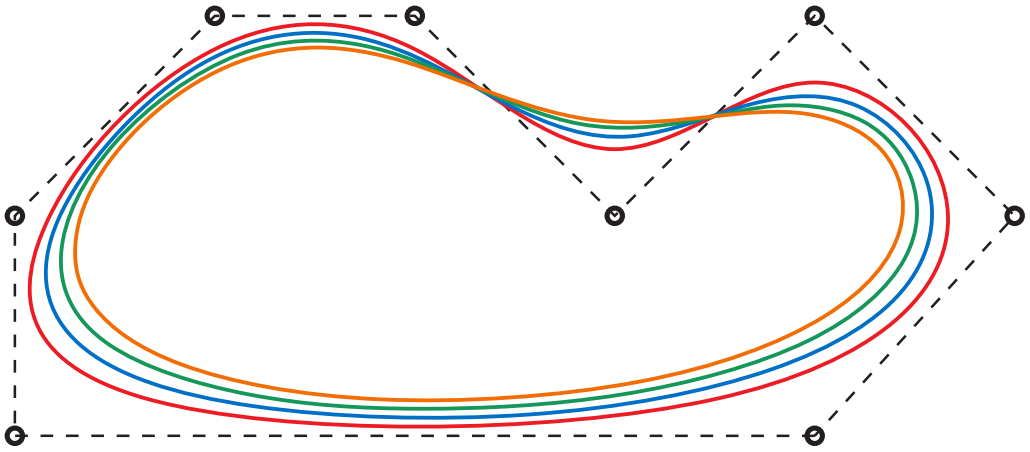
```
int  n = P.size − 1;                          // Antallet intervall
int  m = 2^d (n − 1) + 2;                     // Det endelige antall punkt, fra (6.31)
vector⟨Point⟩  Φ(m);                          // Returvektoren med m punkt.
double  x = Σ_{i=1}^{d} 1/2^{i+1};            // faktoren for å flytte endepunktene, se (6.32)
Φ_0 = P_0 + x/(1−x) (P_0 − P_1);              // Flytter det første punktet iht (6.33)
for ( int  i=1; i < n; i++ )
    Φ_i = P_i;                                // Setter inn de orginale punktene
Φ_n = P_n + x/(1−x) (P_n − P_{n−1});          // Flytter siste punkt iht (6.33)
for ( int  j=1; j ≤ d; j++ )                  // For hvert nivå av forfining
    for ( int  i=n-1; i ≥ 0; i−− )
        P_{2i} = 3/4 P_i + 1/4 P_{i+1};       // Lage nye punkt iht (6.30)
        P_{2i+1} = 1/4 P_i + 3/4 P_{i+1};
    n = 2n − 1;                               // Antall intervaller på neste nivå
return Φ;
```

vector⟨Point⟩ **ChaikinClosed(** vector⟨Point⟩ P, int d **)**

```
int  n = P.size;                              // Antallet intervall
int  m = 2^d n + 1;                           // Det endelige antall punkt, fra (6.31)
vector⟨Point⟩  Φ(m);                          // Returvektoren med m punkt.
for ( int  i=0; i < n; i++ )
    Φ_i = P_i;                                // Setter inn de orginale punktene
Φ_n = P_0;                                    // Kopier startpunktet til slutten
for ( int  j=1; j ≤ d; j++ )                  // For hvert nivå av forfining
    for ( int  i=n-1; i ≥ 0; i−− )
        P_{2i} = 3/4 P_i + 1/4 P_{i+1};       // Lage nye punkt iht (6.30)
        P_{2i+1} = 1/4 P_i + 3/4 P_{i+1};
    Φ_n = Φ_0;                                // Kopier startpunktet til slutten
    n = 2n;                                   // Antall intervaller på neste nivå
return Φ;
```



Figur 6.23: *En B-spline $B_{d+1,i}(t)$, i striplet svart. Den er laget av en sum av $w_{d+1,i}(t)B_{d,i}(t)$ (rødt) og det speilede produktet av den neste B-spline av grad d (blått).*



Figur 6.24: *Plott av 4 lukkede subdivisjonskurver som alle er generert fra samme sett med punkt som kurvene i figurene 6.21 og 6.22. I rødt ser vi en 3.-gradskurve, i blått en 4.-gradskurve, i grønt en 5.-gradskurve og i oransje en 6.-gradskurve. De svarte sirklene er startpunktene.*

6.7.3 Lane-Riesenfeld's subdivisjonsalgoritme

Det finnes imidlertid en enklere og også mer generell måte å utvikle subdivisjonsskjemaer for uniforme B-splines. I 1980 ble Lane-Riesenfeld's subdivisjonsskjema lansert, [111]. Denne algoritmen genererer enten lukkede kurver eller åpne kurver som ikke er klemt. Den er en kompakt, veldig enkel og dermed også en elegant algoritme, og følger egentlig av symmetrien rundt midtverdiene mellom to skjøter og rekursjonen $B_{d+1,i}(t) = w_{d+1,i}(t)B_{d,i}(t) + (1 - w_{d+1,i+1}(t))B_{d,i+1}(t)$ som er illustrert i figur 6.23. Algoritmen er delt inn i to deler. Først dobles punktsettet, noe som kan gjøres ved enten å lage to av hvert punkt eller å sette inn nye punkt midt mellom alle punktene. Neste trinn er glatting/gradsheving. Her erstatter vi alle punktene med nye punkt midt mellom alle de gamle punktene, dvs. $p_i = \frac{1}{2}(\tilde{p}_i + \tilde{p}_{i+1})$ og resultatet er en gradsøkning. Dette trinnet gjentas til vi har oppnådd ønsket polynomisk grad. Totalt har vi nå doblet antall punkt. Hele prosedyren med dobling og glatting gjentas til vi har det ønskede antall med punkt.

I figur 6.24 vises fire kurver, alle laget ved hjelp av Lane-Riesenfeld's subdivisjonsalgoritme. Kurvene er lukkede B-splines av polynomgrad 3, 4, 5 og 6, alle laget fra det samme settet med 8 punkt som også er brukt i figurene 6.21 og 6.22. Lane-Riesenfeld's subdivisjonsalgoritmen følger nå.

Algoritme 7. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen lager en vektor av sampelpunkt til en kurve basert på Lane-Riesenfeld's algoritme. Det er en algoritme for lukkede kurver og en for åpne, men ikke for klemte kurver. Det er 3 hjelpefunksjoner som brukes av hovedalgoritmene. Innput er: startpunktsettet* $\{P_i\}_{i=0}^{n-1}$ *og forfiningsnivå k og grad d.*

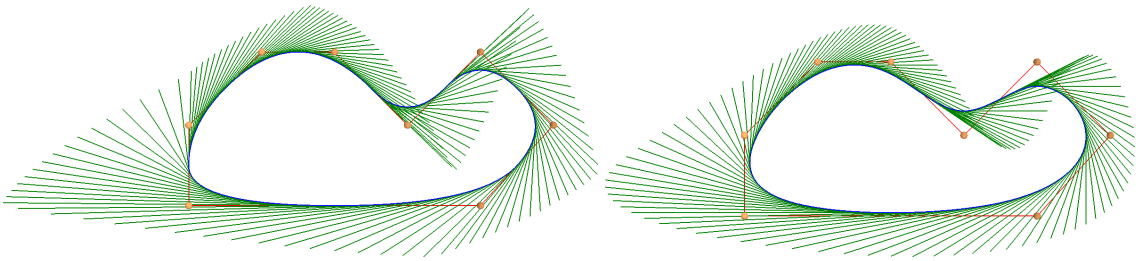
```
vector⟨Point⟩ LaneRiesenfeldOpen( vector⟨Point⟩ P, int k, int d )
    int n = P.size;                          // Antallet intervall
    int m = 2^k(n−d)+d;                      // Det endelige antall punkt
    vector⟨Point⟩ Φ(m+d−1);                  // Returvektoren med m punkt.
    for ( int i=0; i < n; i++ )
        Φ_i = P_i;                           // Setter inn de orginale punktene
    for ( int i=0; i < k; i++ )              // For hvert nivå av forfining
        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;                          // Antallet intervall
    int m = 2^k n+1;                         // Det endelige antall punkt
    vector⟨Point⟩ Φ(m);                      // Returvektoren med m punkt.
    for ( int i=0; i < n; i++ )
        Φ_i = P_i;                           // Setter inn de orginale punktene
    Φ_n = P_0;                               // Lukking av kurven
    for ( int i=0; i < k; i++ )              // For hvert nivå av forfining
        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 = ½(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 = ½(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 = ½(P_i + P_i+1);
        P_n−1 = P_0;
```



Figur 6.25: *Til venstre vises en Doo-Sabin-kurve sammen med dens 1.-deriverte plottet i grønt. Til høyre er en Catmull-Clak-kurve og dens 1.-deriverte også plottet i grønt. De er begge laget fra samme punktsett, plottet som kobberfargede kuler.*

Som leseren kanskje ser, er det mulig å erstatte de nestede løkkene i glattingsdelen med et skjema som er laget med Pascals trekant, dvs. hvis graden er 3 kan vi bruke $P_i = \frac{1}{4}(p_i + 2p_{i+1} + p_{i+2})$, eller hvis graden er 4, $P_i = \frac{1}{8}(p_i + 3p_{i+1} + 3p_{i+2} + p_{i+3}))$. Det er også mulig å optimalisere ved å slå sammen de to delene og utvikle spesifikke opplegg for gitte grader. Ser vi på en grad 3-kurve ser vi at vi har to nivå. På grunn av doblingen av punkt har vi 3 og 3 punkt som ligger på en rett linje. Derfor, på grunn av 2 nivåer, vil hvert andre punkt være lik midtpunktene som ble generert. De andre punktene vil da være $Q_i + 2Q_{i+1} + Q_{i+2}$ der Q_i og Q_{i+2} vil være midtpunkt. Vi får da $P_j = \frac{1}{4}\left(\frac{1}{2}(P_{i-1} + P_i) + 2P_i + \frac{1}{2}(P_i + P_{i+1})\right)$. Dette fører til det vi kan kalle Catmull-Clark-kurver.

Catmull-Clark-kurver, navngitt etter Catmull-Clark-flater, [19], - er 3.-grads uniforme B-splines. Algoritmen som følger er en optimalisert Lane-Riesenfeld-algoritme der både doublePart()- og smoothPartOpen()-funksjonene erstattes av,

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 $2n - d$;

eller hvis den er lukket der både doublePart()- og smoothPartClosed()-funksjonene erstattes av,

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 $2n$;

Til venstre i figur 6.25 er det et plott av en Doo-Sabin-kurve og til høyre et plott av en Catmull-Clark-kurve. Den 1.-deriverte for begge kurvene er også plottet. Dette er gjort som vektorer i hvert samplepunkt og er vist som grønne linjestykker. De derivert er generert ved bruk av dividerte-differanser. Vi kan tydelig se at Doo-Sabin-kurven til venstre bare er $C^1 - glatt$, mens Catmull-Clark-kurven til høyre er minst $C^2 - glatt$. Dette er da også i samsvar med kontinuitetsegenskapene til B-splines.

Subdivisjon er svært mye brukt, spesielt innen datagrafikk. Her er bare et lite utdrag av artikler på feltet som går utover det vi har gått igjennom her; [56, 58, 87, 70, 57, 113, 155].

Kapittel 7

Blending

Bézier og B-splines er basert på blending av punkt, hermitekurver er basert på blending av punkt og vektorer, Coons Patch [26] er egentlig blending av flater, og Gordon Surfaces [79] er også laget ved bruk av blending av kurver og flater. Det er gjort mye arbeid med blending, eksempler er [161], [86], [149]. Det er derfor av interesse å undersøke blending mer detaljert, og spesielt se på blendingsteknikker for å blende funksjoner generelt. Vi starter med å definere B-funksjoner som først ble omtalt i [104].

7.1 B-funksjonen

B-funksjon er en forkortelse for blendingsfunksjon. Den er konstruert for å blende funksjoner (enten de er skalar-, vektor-, punkt-baserte), punkt, kurver, tensorproduktflater eller trekantedeflater osv. Vi avgrenser en B-funksjon til å være monotone og vi skal se spesielt på symmetriske B-funksjoner og hva det betyr. Definisjonen er:

Definisjon 7.1. *En B-funksjon er:*

D1 *en homeomorfi ("permuteringsfunksjon")* $B : I \to I \quad (I = [0,1] \subset \mathbb{R})$,

– D2 *dermed er* $B(0) = 0$,

– D3 *og* $B(1) = 1$,

– D4 *og den er monoton, dvs.* $B'(t) \geq 0, \quad t \in I$.

D5 *En B-function kalles symmetrisk hvis,* $B(t) + B(1-t) = 1, \quad t \in I$.

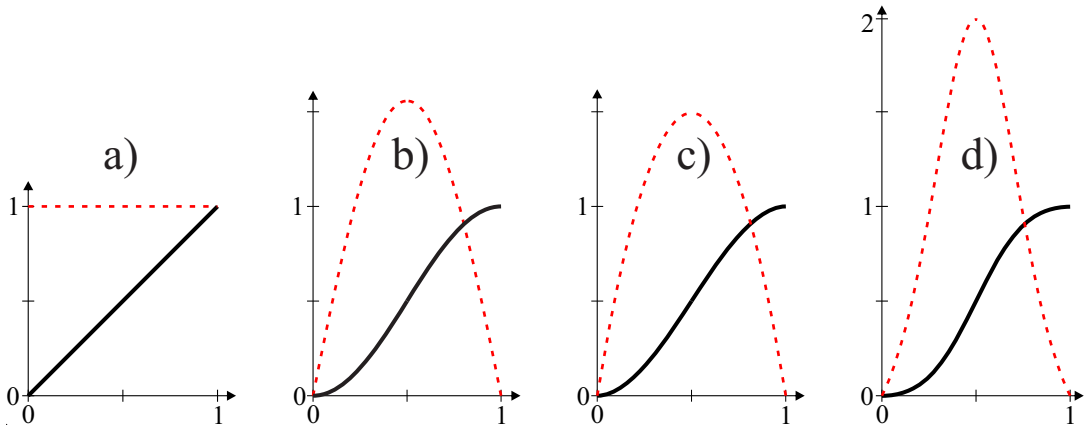
Denne symmetrien er en punktsymmetri, rundt punktet $(0.5\ 0.5)$. Andre typer symmetri vil bli introdusert i seksjoner 7.8. En mer generell definisjon av en B-funksjon er gitt i Definisjon 13.1. For å gi en idè om hva en B-funksjon er, skal vi se på fire enkle eksempler på symmetriske B-funksjoner:

a) lineær funksjon $B(t) = t$

b) trigonometrisk funksjon $B(t) = \sin^2 \frac{\pi t}{2}$

c) polynomfunksjon av orden 1 $B(t) = 3t^2 - 2t^3$

d) rasjonal funksjon av orden 1 $B(t) = \frac{t^2}{t^2 + (1-t)^2}$



Figur 7.1: *Fire B-funksjoner (heltrukken svart) og deres deriverte (rød stiplet). Fra venstre ser vi a) $B(t) = t$, b) $B(t) = \sin^2 \frac{\pi t}{2}$, c) $B(t) = 3t^2 - 2t^3$ og d) $B(t) = \frac{t^2}{(1-t)^2+t^2}$.*

Figur 7.1 viser de fire eksemplene på B-funksjoner plottet sammen med deres 1.-deriverte. Vi kan tydelig se fra figuren at disse fire funksjonene alle starter på 0 og slutter på 1 og at de alle er monotone. At de er symmetriske følger av at;

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-funksjoner kan organiseres i grupper. Alle eksemplene ovenfor er med i slike grupper av B-funksjoner. Senere skal vi foreta en grundig gjennomgang av noen av gruppene og deres egenskaper. Men la oss først se på definisjonen av en viktig egenskap.

Definisjon 7.2. *En egenskap som spiller en sentral rolle er antallet etterfølgende deriverte som er null ved start og slutt. Vi kaller dette for*

B-funksjonens orden

Ordenen til en B-funksjon, forkortelse for hermiteorden til en B-funksjon og betegnet S, er for en symmetrisk B-funksjon bestemt av:

$$B^{(j)}(0) = B^{(j)}(1) = 0, \quad j = 1, 2, ..., S. \tag{7.1}$$

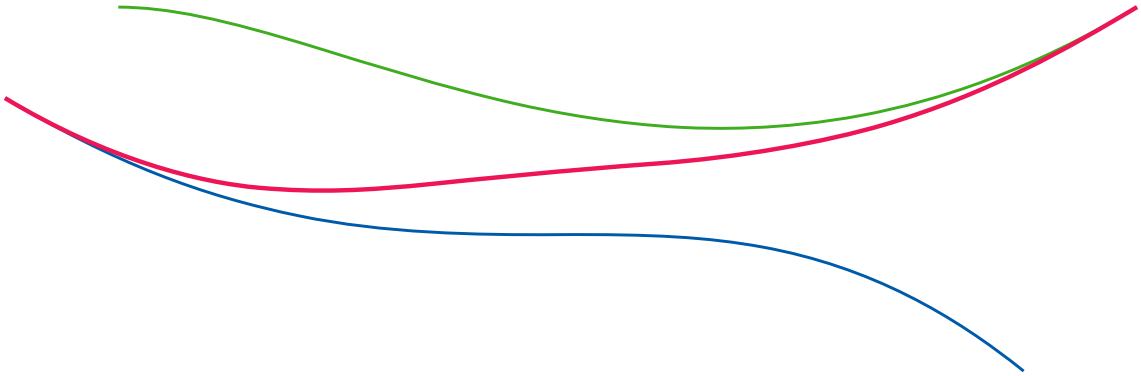
For en ikke-symmetrisk B-funksjon må vi skille mellom start og slutt,

$$\begin{aligned} B^{(j)}(0) &= 0, \quad j = 1, 2, ..., S_0. \\ B^{(j)}(1) &= 0, \quad j = 1, 2, ..., S_1. \end{aligned} \tag{7.2}$$

Dette kalles hermiteegenskapen til en B-funksjon og er forklart videre i teorem7.1.

For eksemplene b), c) og d) er $B'(0) = B'(1) = 0$, dvs. den 1.-deriverte er null ved start og slutt, men den 2.-deriverte er ikke det. Det følger at dette er 1.-ordens B-funksjoner.

Senere i dette kapittelet skal vi se på høyere-ordens B-funksjoner, og også komplette B-funksjoner der alle deriverte er null både ved start og slutt.



Figur 7.2: *I figuren er to bézierkurver blendet sammen til en kurve ved hjelp av en B-funksjon. De originale bézierkurvene er plottet i henholdvis blått og i grønt. Resultatet av blendingen er kurven som er plottet i rødt.*

7.2 Blending av to funksjoner

Den enkleste blendingen er blending av to funksjoner/kurver, $g_1(t)$ og $g_2(t)$, organisert i en sekvens der $g_1(t)$ er den første kurven og $g_2(t)$ er den andre kurven, Begge kurvene må ha et felles domene som vi velger å være $[0,1]$. Et eksempel er vist i figur 7.2. To bézierkurver er plottet, $g_1(t)$ i blått og $g_2(t)$ i grønt. Resultatet av blendingen er vist som en rød kurve. B-funksjonen som brukes her er $B(t) = 3t^2 - 2t^3$. Resultatet av blendingen er avhengig av rekkefølgen av de to kurvene samt hermiteordenen til B-funksjonen som er brukt. Dette vil bli drøftet nærmere senere. Først skal vi se på formlene.

Formlene for blending av to funksjoner

Formlene for en to-funksjons-blending er

$$\begin{aligned} f(t) &= (1-B(t))\, g_1(t) + B(t)\, g_2(t) \\ &= g_1(t) + B(t)\, (g_2(t) - g_1(t)). \end{aligned} \tag{7.3}$$

hvor $B(t)$ er en B-funksjon.
Hvis vi betegner differansefunksjonen for $h(t) = g_2(t) - g_1(t)$, får vi formelen

$$f(t) = g_1(t) + B(t)\, h(t)). \tag{7.4}$$

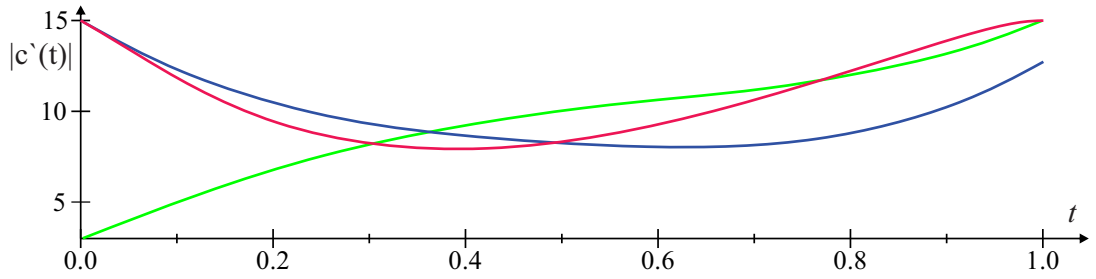
Den 1.-deriverte er

$$f'(t) = g_1'(t) + B(t)\, h'(t) + B'(t)\, h(t), \tag{7.5}$$

og formelen for høyere ordens deriverte er

$$f^{(j)}(t) = g_1^{(j)}(t) + \sum_{i=0}^{j} \binom{j}{i} B^{(i)}(t)\, h^{(j-i)}(t). \tag{7.6}$$

I figur 7.3 er hastigheten til kurvene fra figur 7.2 plottet. Der kan vi se at hastigheten til den resulterende kurven er den samme som hastigheten til den første kurven ved start, dvs. $|f'(0)| = |g_1'(0)|$, og hastigheten til den resulterende kurven er den samme som has-



Figur 7.3: *Funksjonene viser hastighetene, $|c'(t)|$, til de to bézierkurvene og den resulterende blendede kurven fra figur 7.2. Plottene av hastighetsfunksjonene har også de samme fargene som de tilsvarende kurvene i figur 7.2.*

tigheten til den andre kurven ved slutten, dvs. $|f'(1)| = |g_2'(1)|$.

Relasjonene mellom de to originale funksjonene kalt lokale funksjoner og den resulterende funksjonen kalt den globale funksjonen, og da spesielt det som skjer i starten og på slutten av funksjonene er av spesiell interesse. Følgende teorem gir en beskrivelse av dette.

Teorem 7.1. *En to-funksjons-blending med B-funksjon har følgende egenskap*

Hermiteinterpolasjonsegenskapen

I en to-funksjons-blending med en B-funksjon, $f(t) = g_1(t) + B(t)\ (g_2(t) - g_1(t))$, følger det at den globale funksjonen interpolerer den første lokale funksjonen i startpunktet med posisjon og deriverte opp til ordre S_0, dvs.

$$f^{(j)}(0) = g_1^{(j)}(0), \quad j = 0, 1, ..., S_0, \tag{7.7}$$

og likeledes vil den globale funksjonen interpolerer den andre lokale funksjonen i endepunktet med posisjon og deriverte opp til ordre S_1, dvs.

$$f^{(j)}(1) = g_2^{(j)}(1), \quad j = 0, 1, ..., S_1, \tag{7.8}$$

hvor S_0 og S_1 er hermiteordenene ved start og slutt til den gjeldende B-funksjonen.

Bevis. Vi ser først på (7.7). dvs. i starten av funksjonene, $t = 0$. Fra **D2** i Definisjon 7.1 og fra (7.2) ser vi at $B^j(0) = 0$, for $j = 0, 1, ..., S$. Dermed følger det av (7.4) og (7.6) at $f^{(j)}(0) = g_1^{(j)}(0)$, for $j = 1, ..., S_0$.

Å bevise (7.8) er basert på et lignende resonnement som ovenfor på grunn av symmetrien. I slutten er $t = 1$. Husk fra **D3** i definisjon 7.1 at $B(1) = 1$. Fra (7.3) følger det at $f(1) = g_2(1)$. Hvis vi skriver om (7.6) ved å skille ut første ledd av summen får vi

$$f^{(j)}(t) = g_1^{(j)}(t) + B(t)\ (g_2^{(j)}(t) - g_1^{(j)}(t)) + \sum_{i=0}^{j} \binom{j}{i} B^{(i)}(t)\ h^{(j-i)}(t),$$

og siden $B(1) = 1$ og $B^{(j)}(1) = 0$ så er $f^{(j)}(1) = g_2^{(j)}(1)$, for $j = 1, ..., S_1$. □

Hermiteinterpolasjonsegenskapen påvirkes også av de lokale funksjonene. Dette kan oppsummeres i følgende teorem;

Teorem 7.2. *Hermiteinterpolasjonsegenskapen til en to-funksjons-blending påvirkes av*

Den utvidede hermiteinterpolasjonsegenskapen

I en to-funksjons-blending, hvis de to lokale funksjonene er like ved starten, dvs. $g_1(0) = g_2(0)$*, øker ordenen til hermiteinterpolasjonen med 1 ved* $t = 0$*, dvs.*

$$f^{(j)}(0) = g_1^{(j)}(0), \quad j = 0, 1, ..., S_0 + 1, \tag{7.9}$$

og hvis de to lokale funksjonene er like ved slutten, dvs. $g_1(1) = g_2(1)$*, så øker ordenen til hermiteinterpolasjonen med 1 ved slutten, dvs.*

$$f^{(j)}(1) = g_2^{(j)}(1), \quad j = 0, 1, ..., S_1 + 1, \tag{7.10}$$

hvor S_0 *og* S_1 *er (hermite)ordenene ved start og slutt av gjeldende B-funksjon.*

Generelt, hvis de to lokale funksjonene har lik posisjon og også deriverte ved start, dvs. $g_1^{(j)}(0) = g_2^{(j)}(0),\ j = 0, 1, ...d_0$*, så øker ordenen til hermiteinterpolasjonen med* $d_0 + 1$ *ved start, dvs.*

$$f^{(j)}(0) = g_1^{(j)}(0), \quad j = 0, 1, ..., S_0 + d_0 + 1, \tag{7.11}$$

og hvis de to lokale funksjonene har lik posisjon og også deriverte like i slutten, dvs. $g_1^{(j)}(1) = g_2^{(j)}(1),\ j = 0, 1, ...d_1$*, så øker ordenen til hermiteinterpolasjonen med* $d_1 + 1$ *i slutten, dvs.*

$$f^{(j)}(1) = g_2^{(j)}(1), \quad j = 0, 1, ..., S_1 + d_1 + 1, \tag{7.12}$$

hvor S_0 *og* S_1 *er (hermite)orderen ved start og slutt av gjeldende B-funksjon.*

Bevis. Hvis vi skriver om (7.6) ved å trekke ut siste ledd av summen i uttrykket får vi

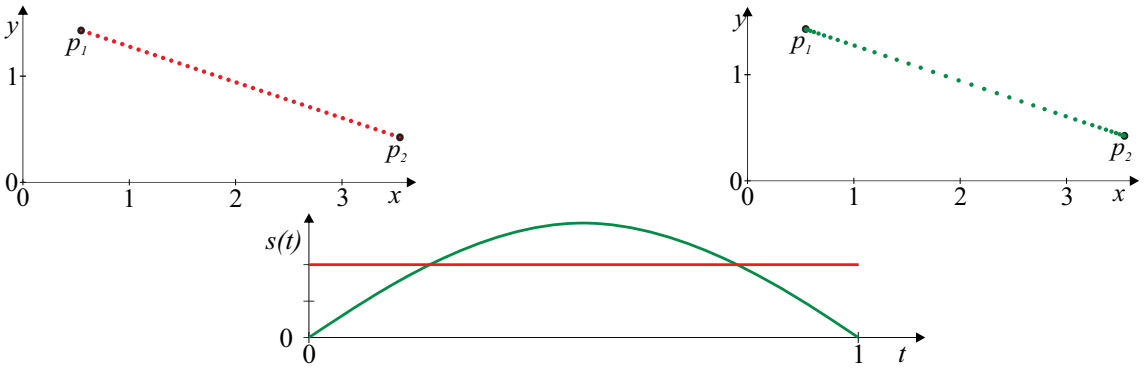
$$f^{(j)}(t) = g_1^{(j)}(t) + \sum_{i=0}^{j-1} \binom{j}{i} B^{(i)}(t)\ h^{(j-i)}(t) + B^{(j)}(t)\ (g_2(t) - g_1(t)).$$

Siden $g_2(0) - g_1(0) = 0$ følger det da at summen samt det siste leddet også er null for $j = 0, 1, \ldots, S_0 + 1$ fordi $B^{(i)}(0) = 0$ for $j = 0, 1, \ldots, S_0$.

Den samme typen argument er også gyldig i slutten av kurven. Dette fulfører beviset for den første delen av teoremet.

Hvis vi skriver om (7.6) ved å ta ut de d siste leddene av summen i uttrykket får vi

$$f^{(j)}(t) = g_1^{(j)}(t) + \sum_{i=0}^{j-d-1} \binom{j}{i} B^{(i)}(t)\ h^{(j-i)}(t) + \sum_{i=j-d}^{j} \binom{j}{i} B^{(i)}(t)\ (g_2^{(j-i)}(t) - g_1^{(j-i)}(t)).$$



Figur 7.4: *Figuren øverst til venstre viser en kurve med en lineær blending av de to punktene p_1 og p_2. Kurven er plottet med røde prikker med like avstand (i parameterverdi) som da viser konstant hastighet. Til høyre er den trigonometriske B-funksjonen brukt i blendingen (grønne prikker) og vi kan tydelig se at hastigheten ikke er konstant. Nederst er hastigheten plottet. Den røde viser konstant hastighet, mens den grønne starter og slutter på null. Integralet av de to funksjonene er imidlertid den samme fordi kurvelengden er den samme.*

Fordi $B^{(i)}(0) = 0$, er den nest siste summen også null for $i = 0, 1, ..., S_0$. Og siden $j - i = d, d-1, \ldots, 0$ når $i = j-d, j-d+1, \ldots, j$ er også $g_1^{(j-i)}(0) - g_2^{(j-i)}(0) = 0$ når $j - i = 0, 1, ...d$. Dermed følger det at ordenen til de deriverte som er null går opp til $S_0 + d + 1$.

Det samme argumentet er gyldig også for $t = 1$, dvs. i slutten av kurven. Dermed er beviset for den andre delen av teoremet fullført. □

7.2.1 Eksempler, blending av orden 0 og 1

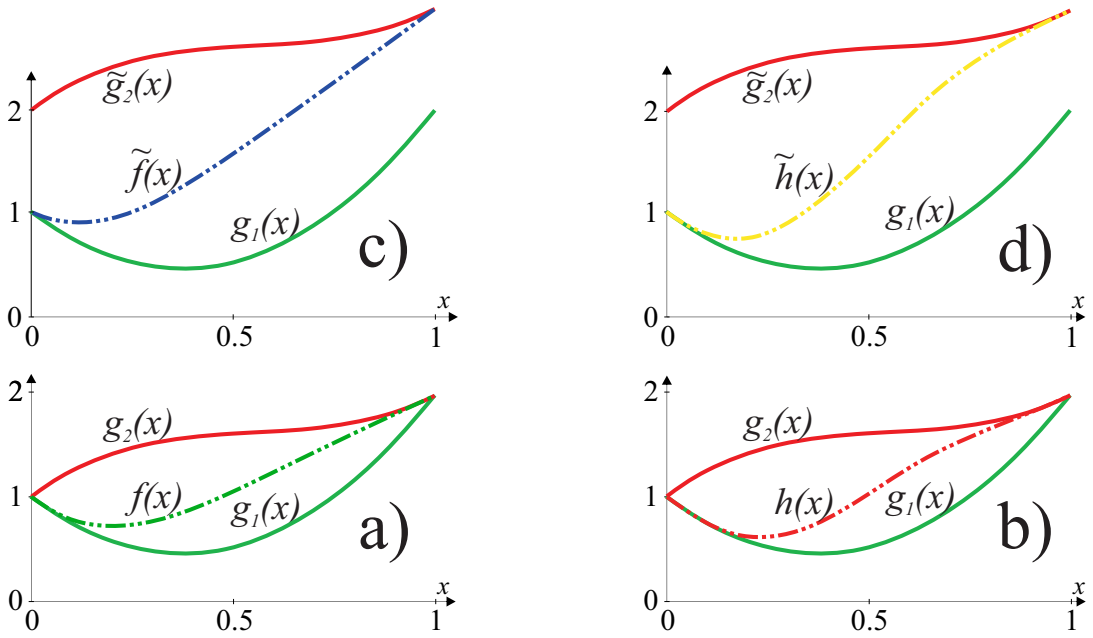
Eksempel på blending av to punkt (figur 7.4):
Den enkleste B-funksjonen er den lineære blendingsfunksjonen, $B(t) = t$, mest kjent fra lineærinterpolasjon. Helt til venstre i figur 7.1 er denne orden 0 B-funksjon plottet sammen med den 1^{st}-deriverte, og til venstre i figur 7.4 kan vi se en kurve som er en lineær blending av to punkt, p_1 og p_2. Fra formel (7.4) får vi

$$\begin{aligned} c(t) &= p_1 + t\,(p_2 - p_1), \\ c'(t) &= p_2 - p_1. \end{aligned}$$

Som vi kan se er her den deriverte $c'(t)$ en konstant, og hastigheten til kurven er dermed den samme overalt. Dette kan tydelig sees i figur 7.4 hvor den røde kurven er plottet med en jevnt fordelt punktsekvens øverst til venstre, og den røde linjen i det nederste plottet som viser hastigheten til kurven.

Øverst til høyre i figur 7.4 er et plott av en lineær kurve laget av formel (7.4) og hvor B-funksjonen er den trigonometriske $B(t) = \sin^2 \frac{\pi}{2} t$ av ordre 1. Dette gir formelen

$$\begin{aligned} c(t) &= p_1 + \left(\sin^2 \tfrac{\pi}{2} t\right)(p_2 - p_1) \\ c'(t) &= \left(\pi \cos \tfrac{\pi}{2} t \sin \tfrac{\pi}{2} t\right)(p_2 - p_1) \end{aligned}$$



Figur 7.5: *Alle kurver i grønt i plottet er $g_1(x)$, uttrykk (7.13). Kurvene i de to nederste plottene plottet i rødt er $g_2(x)$, uttrykk (7.14). Kurvene plottet i rødt i de to øverste plottene er $\widetilde{g}_2(x)$, uttrykk 7.15. Blendingene på venstre side bruker en lineær B-funksjon av orden 0, mens blendingene på høyre side bruker en trigonometrisk B-funksjon av orden 1.*

Fra formelen er det klart at hastigheten ikke er konstant. Den er null ved start og slutt fordi $\sin 0 = 0$ og $\cos \frac{\pi}{2} = 0$. I figur 7.4 øverst til høyre er prikkene i den grønne kurven ikke jevnt fordelt, tettheten er størst i begge ender. Dette bekreftes i plottet nederst i figur 7.4. Der ser vi en (grønn) funksjon som beskriver kurvens hastigheten over domenet $[0,1]$.

Eksempel på blending av to funkjoner (figur 7.5):
I det neste eksemplet blender vi to funksjoner. Den første funksjonen er

$$g_1(x) = 4x^2 - 3x + 1 \tag{7.13}$$

som er plottet som en grønn kurve i figur 7.5. Den andre funksjonen er

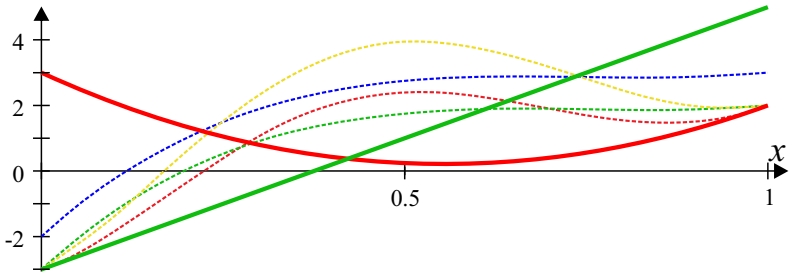
$$g_2(x) = 3x^3 - 5x^2 + 3x + 1, \tag{7.14}$$

som vises som en rød kurve i plott a) og b) i figur 7.5. I plott c) og d) er g_2 flyttet 1 oppover (i y-retning), som endrer formelen til

$$\widetilde{g}_2(x) = 3x^3 - 5x^2 + 3x + 2. \tag{7.15}$$

I eksemplene brukes to forskjellige B-funksjoner, den 0.-ordens lineære B-funksjon og den 1.-ordens trigonometriske B-funksjonen. Den lineære B-funksjon $B(t) = t$ gir

$$\begin{aligned} f(x) &= g_1(x) + x\,(g_2(x) - g_1(x)), \\ f'(x) &= g_1'(x) + x\,(g_2'(x) - g_1'(x)) + g_2(x) - g_1(x), \\ f''(x) &= g_1''(x) + x\,(g_2''(x) - g_1''(x)) + 2(g_2'(x) - g_1'(x)). \end{aligned}$$



Figur 7.6: *Plot av de 1.-deriverte; dvs. $g_1'(x)$-solid grønn, $g_2'(x)$-solid rød og blendingskurvene $f'(x)$-stiplet grønn, $h'(x)$-stiplet rød , $\widetilde{f}'(x)$-stiplet blå og $\widetilde{h}'(x)$-stiplet gul.*

Bruke vi den trigonometriske B-funksjonen $B(t) = \sin^2 \frac{\pi t}{2}$ får vi

$$\begin{aligned}
h(x) &= g_1(x) + \sin^2 \tfrac{\pi}{2} x (g_2(x) - g_1(x)) \\
&= \tfrac{1}{2}(g_1 + g_2 + (g_1 - g_2)\cos \pi x), \\
h'(x) &= \tfrac{1}{2}\left((g_1' + g_2') + (g_1' - g_2')\cos \pi x - \pi\,(g_1 - g_2)\sin \pi x\right), \\
h''(x) &= \tfrac{1}{2}\left((g_1'' + g_2'') + \left((g_1'' - g_2'') - \pi^2\,(g_1 - g_2)\right)\cos \pi x - 2\pi\,(g_1' - g_2')\sin \pi x\right).
\end{aligned}$$

Vi har nå to blendingskurvene $f(x)$ og $h(x)$, og om vi bytter ut $g_2(x)$, (7.14), med $\widetilde{g_2}$, (7.15), får vi to nye blendingskurvene $\widetilde{f}(x)$, og $\widetilde{h}(x)$. Disse fire kurvene er alle plottet i figur 7.5 med stipler. Legg merke til at kurven i plott c) kun interpolerer posisjonen til g_1 i starten og g_2 på slutten (orden 0). Kurvene i plott a) og d) interpolerer også den første deriverte (orden 1), og kurven i plott b) interpolerer også den andre deriverte (orden 2).

For å verifisere disse observasjonen er alle 1.- og 2.-deriverte regnet ut i tabellen nedenfor. Selvsagt er alle deriverte for $\widetilde{g_2}$ og g_2 like (translasjon endrer ikke deriverte). I tabellen er den første og andre deriverte i starten og slutten av g_1, g_2, og i starten og slutten av blendingskurvene f, h, $\widetilde{f}$ og $\widetilde{h}$ regnet ut;

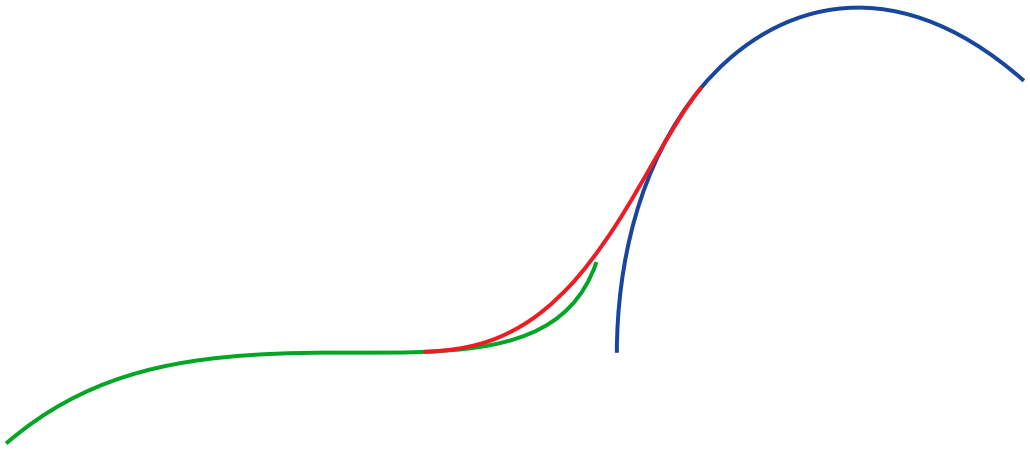
g_1-grønn, g_2-rød	$g_1'(0) = -3$	$g_2'(1) = 2$	$g_1''(0) = 8$	$g_2''(1) = 8$	orden
a) $f(x)$-grønn	$f'(0) = -3$	$f'(1) = 2$	$f''(0) = 20$	$f''(1) = 2$	1
b) $h(x)$-rød	$h'(0) = -3$	$h'(1) = 2$	$h''(0) = 8$	$h''(1) = 8$	2
c) $\widetilde{f}(x)$-blå	$\widetilde{f}'(0) = -2$	$\widetilde{f}'(1) = 3$	$\widetilde{f}''(0) = 20$	$\widetilde{f}''(1) = 2$	0
d) $\widetilde{h}(x)$-gul	$\widetilde{h}'(0) = -3$	$\widetilde{h}'(1) = 2$	$\widetilde{h}''(0) = 8 + \frac{\pi^2}{2}$	$\widetilde{h}''(1) = 8 - \frac{\pi^2}{2}$	1

Bokstavene og fargene i første kolonne i tabellen refererer til bokstavene og fargene i figur 7.5 og også til fargen til de 1.-deriverte i figur 7.6. De røde tallene i tabellen indikerer samsvar med g_1 ved $x = 0$ og samsvar med g_2 ved $x = 1$.

I figur 7.6 er de 1.-deriverte til g_1 og g_2 plottet sammen med 1.-deriverte til blendingskurvene. De illustrer godt effekten av hermiteordenen og de samsvarer med tabellen.

7.2.2 Eksempel, koble sammen to kurver ved å bruke en B-funksjon

Vi skal se på et eksempel hvor to separate kurver kobles sammen til én glatt kurve ved hjelp av en B-funksjon. Det finns mange artikler om både dette og forrige emne, eksempelvis [166], [122] og [86]. Vi skal i dette eksempelet bruke et splinekonsept og dermed



Figur 7.7: *Vi ser to kurver koblet sammen ved hjelp av en B-funksjon. Den grønne kurven $c_1(t)$ er koblet til den blå kurven $c_3(t)$ med den røde kurven $c_2(t)$ som kobler de to opprinnelige kurvene sammen på en glatt måte, dvs. med C^1-kontinuitet.*

en skjøtvektor. Vi tar utgangspunkt i to 3.-grads bézierkurver i $\mathbb{R}^2$,

$$c_1(t) = (1-t)^3 \begin{pmatrix} 1 \\ 1 \end{pmatrix} + 3t(1-t)^2 \begin{pmatrix} 2 \\ 2 \end{pmatrix} + 3t^2(1-t) \begin{pmatrix} 3.6 \\ 1 \end{pmatrix} + t^3 \begin{pmatrix} 3.9 \\ 2 \end{pmatrix}, \quad t \in [0,1],$$

og

$$c_3(t) = (1-t)^3 \begin{pmatrix} 4 \\ 1.5 \end{pmatrix} + 3t(1-t)^2 \begin{pmatrix} 4 \\ 3 \end{pmatrix} + 3t^2(1-t) \begin{pmatrix} 5 \\ 4 \end{pmatrix} + t^3 \begin{pmatrix} 6 \\ 3 \end{pmatrix}, \quad t \in [0,1].$$

Disse to kurvene er plottet i figur 7.7, $c_1(t)$ i grønt på venstre side, og $c_3(t)$ i blått til høyre. Vi bruker 40% av hver av de opprinnelige kurvene til å lage "sammenføyningskurven". $t = [0, 0.6)$ gir første del, $t = [0.6, 1)$ gir midterste del og med translering av domenet gir $t = [1, 1.6]$ siste del. Vi får da skjøtvektoren $\{t_i\}_{i=0}^3 = \{0,\ 0.6,\ 1,\ 1.6\}$, og vi får følgende formel for midtdelen, sammenføyningskurven,

$$c_2(t) = c_1(t) + B \circ w_{1,1}(t)(c_3(t-t_1) - c_1(t)), \quad t_1 \le t < t_2.$$

der $B(t)$ er en B-funksjon, og $w_{1,i}(t) = \frac{t-t_i}{t_{i+1}-t_i}$, se (6.11). I dette eksemplet er $B(t) = 3t^2 - 2t^3$. Dette nye kurvesegmentet $c_2(t)$ er vist som en røde kurven i figur 7.7.

Formelen for den totale sammenkoblede kurven er

$$f(t) = \begin{cases} c_1(t), & \text{for } \ t_0 \le t < t_1, \\ c_2(t), & \text{for } \ t_1 \le t < t_2, \\ c_3(t-t_1), & \text{for } \ t_2 \le t \le t_3. \end{cases} \tag{7.16}$$

Domenet til $f(t)$, definert i (7.16), er $[t_0, t_3]$. Kurven er C^1-glatt fordi vi har brukt en B-funksjon av orden 1, jmf. teorem 7.1. Den resulterende kurven kan sees i figur 7.7 som en sammenhengende og glatt grønn, rød og blå kurve.

7.3 Beta-funksjoner - polynomiske B-funksjoner

Euler-betafunksjonen, også kalt Euler-integralet av den første typen, er en spesialfunksjon definert ved

$$\mathscr{B}(a,b) = \int_0^1 x^a (1-x)^b dx$$

for $a, b \geq 0$, se [1]. Euler-betafunksjonen er symmetrisk, dvs. $\mathscr{B}(a,b) = \mathscr{B}(b,a)$. Hvis a og b er positive heltall, kan uttrykket forenkles til:

$$\mathscr{B}(a,b) = \frac{a!\, b!}{(a+b+1)!}. \tag{7.17}$$

Euler-betafunksjonen ble først studert av Euler og Legendre, men fikk visnok navnet sitt av Jacques Binet.

Den ufullstendige betafunksjonen er en generalisering av Euler-betafunksjonen som erstatter det bestemte integralet til Euler-betafunksjonen med et ubestemt integral, dvs.

$$\mathscr{B}(t;a,b) = \int_0^t x^a (1-x)^b dx. \tag{7.18}$$

Den regulariserte ufullstendige betafunksjonen er den ufullstendige betafunksjonen skalert med den komplette betafunksjonen, dvs.

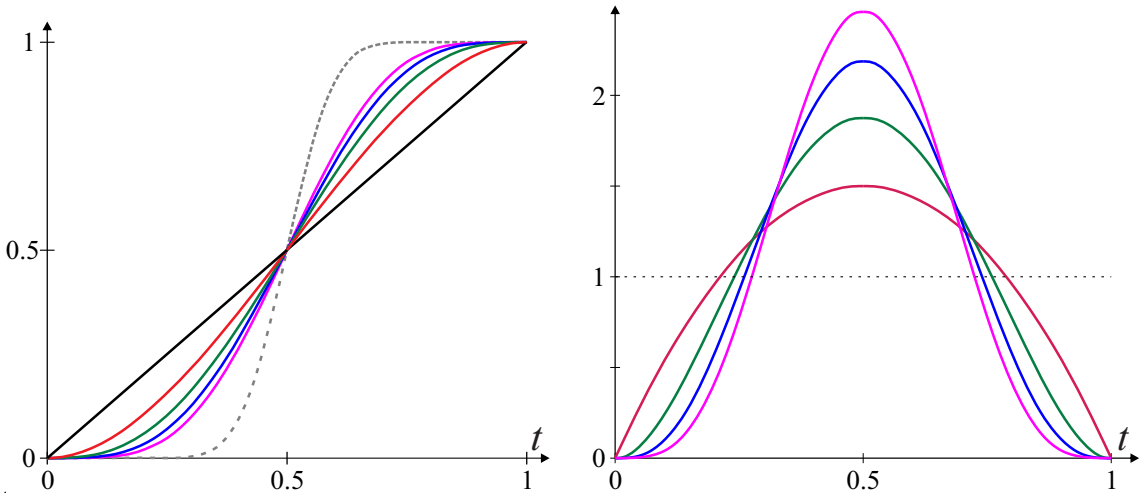
$$I_t(a,b) = \frac{\mathscr{B}(t;a,b)}{\mathscr{B}(a,b)}, \quad t \in [0,1]. \tag{7.19}$$

Hvis vi regner ut integralet for heltallsverdier av a og b, får vi (regnet ut i [131])

$$I_t(a,b) = \sum_{j=a+1}^{a+b+1} \frac{(a+b+1)!}{j!\,(a+b+1-j)!}\, t^j (1-t)^{a+b+1-j}. \tag{7.20}$$

Lemma 7.1. *Den regulariserte ufullstendige betafunksjonen $I_t(a,b)$ har følgende egenskaper:*

I	*Null ved $t=0$*	$I_0(a,b) = 0$,	
II	*En ved $t=1$*	$I_1(a,b) = 1$,	
III	*Hermiteorden*	$\frac{d^j}{dt^j} I_0(a,b) = 0, \quad j = 1,2,...,a,$	*i.e. orden a ved start*
		$\frac{d^j}{dt^j} I_1(a,b) = 0, \quad j = 1,2,...,b,$	*i.e. orden b i enden*
IV	*Antisymmetrisk*	$I_t(a,b) = 1 - I_{1-t}(b,a),$	*i.e. symmetrisk hvis a=b*
V	*monoton*	$\frac{d}{dt} I_t(a,b) > 0, \quad 0 < t < 1 \quad og \quad \frac{d}{dt} I_t(a,b) = 0, \quad t = \{0,1\}.$	
VI	*Rekursiv*	$I_t(a,b) = t\, I_t(a-1,b) + (1-t)\, I_t(a,b-1).$	
		$I_t(a,b) = I_t(a-1,b) - \frac{t^a (1-t)^{b+1}}{a\ \mathscr{B}(a-1,b)} = I_t(a,b-1) + \frac{t^{a+1}(1-t)^b}{b\ \mathscr{B}(a,b-1)},$	



Figur 7.8: *På venstre side ser vi seks symmetriske Beta-funksjoner, sort: 0.-orden, rød: 1.-orden, grønn: 2.-orden, blå: 3.-orden, lilla: 4.-orden og stiplet grå: 20.-orden. Til høyre er de 1.-ordens deriverte av Beta-funksjonene (unntatt orden 20) i samme farge.*

Beviset til lemma 7.1 finner vi i vedlegg C.3.

Teorem 7.3. *Den regulariserte ufullstendige betafunksjonen er en B-funksjon når verdiene a,b er heltall. Vi kaller den Beta-funksjon (se [43]) og uttrykkes ved;*

$$B_{a,b}(t) = I_t(a,b), \quad for \;\; t \in [0,1],$$

og for $a,b \geq 0$, $a,b \in \mathbb{Z}$, og hvor $a = S_0$ og $b = S_1$ er (hermite)ordenen til Beta-funksjonen. Hvis $a = b$ bruker vi notasjonen $B_S(t) = I_t(S,S)$

Bevis. Teoremet følger av definisjon 7.1 og definition 7.2 og lemma 7.1. □

Merk at Beta-funksjonen av lavest orden er lineær og gruppen konvergerer mot en trinnfunksjon. Beta-funksjonen ble introdusert i 2006 av Dechevsky sammen med Gancheva og Delistoyanova, se [71] og [47]. Vi skal se på den symmetriske serien av Beta-funksjoner.

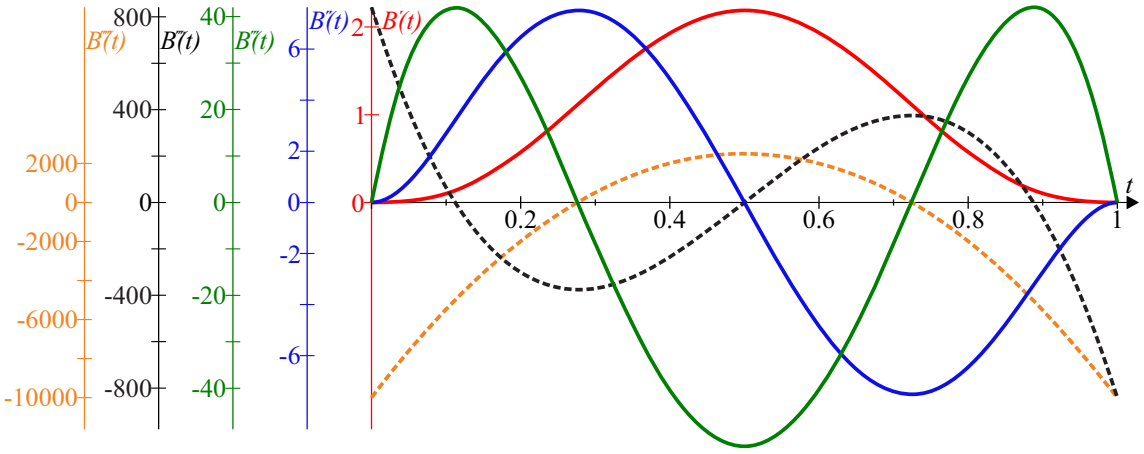
Symmetriske Beta-funksjoner

En symmetrisk Beta-funksjon $B_s(t)$ er symmetrisk på grunn av egenskapen **IV** i lemma 7.1. Ordenen til Beta-funksjonen er S. De første fem Beta-funksjonene av orden 0 til 4 er:

$$B_0(t) = t \qquad \text{orden 0}$$
$$B_1(t) = -2t^3 + 3t^2 \qquad \text{orden 1}$$
$$B_2(t) = 6t^5 - 15t^4 + 10t^3 \qquad \text{orden 2}$$
$$B_3(t) = -20t^7 + 70t^6 - 84t^5 + 35t^4 \qquad \text{orden 3}$$
$$B_4(t) = 70t^9 - 315t^8 + 540t^7 - 420t^6 + 126t^5 \qquad \text{orden 4}$$

Alle beregnet fra formel (7.20)

I figur 7.8 er de fem første symmetriske Beta-funksjonene plottet sammen med Beta-funksjonen av orden 20. Merk at serien ser ut til å konvergere mot en trinnfunksjon ved



Figur 7.9: *Figuren viser fem deriverte til en 3.-ordens symmetrisk Beta-funksjon; 1.-deriverte i rød, 2.-deriverte i blått, 3.-deriverte i grønt, 4.-deriverte i stiplet svart og 5.-deriverte i stiplet oransje. Funksjonene er koblet til sin respektive akse på venstre side med sin farge. 4.- og 5.-deriverte er stiplet for å markere at de ikke er null ved $t = 0,1$.*

$t = 0,5$. De symmetriske Beta-funksjonene er for øvrig de samme funksjonen vi finner som basisfunksjon for hermitekurver av grad $3,5,...$, se basisfunksjon 2 i (4.19) og (4.26).

7.3.1 Beta-funksjoner, differensiering

De driverte til en generell Beta-funksjon er ganske komplekse, spesielt høyereordens-deriverte. De tre første deriverte for generelle Beta-funksjonen er:

$$
\begin{aligned}
B'_{a,b}(t) &= \tfrac{(a+b+1)!}{a!b!}\, t^a(1-t)^b, \\
B''_{a,b}(t) &= \tfrac{(a+b+1)!}{a!b!}\, (a(1-t)-bt)\, t^{a-1}(1-t)^{b-1}, \\
B'''_{a,b}(t) &= \tfrac{(a+b+1)!}{a!b!}\, (a(a-1)-(a+b-1)(2a-(a+b)t)t)\, t^{a-2}(1-t)^{b-2}.
\end{aligned}
$$

For konkrete Beta-funksjoner er de deriverte mye enklere å finne. Det er egentig bare å derivere polynomer.

Vi skal nå lage en visuell illustrasjon på ordenen til en Beta-funksjon. Som eksempel skal vi bruke en 3.-ordens beta-funksjon. Formlene for de første fem deriverte til en symmetrisk 3.-ordens beta-funksjon er:

$$
\begin{aligned}
B_3(t) &= -20t^7 + 70t^6 - 84t^5 + 35t^4, \\
B'_3(t) &= 140t^3(1-t)^3, \\
B''_3(t) &= 420t^2(1-2t)(1-t)^2, \\
B'''_3(t) &= 840t(1-t)(5t^2-5t+1), \\
B^{(4)}_3(t) &= 840(1-2t)(10t^2-10t+1), \\
B^{(5)}_3(t) &= 10080(-5t^2+5t-1).
\end{aligned}
$$

I figur 7.9 viser disse første fem deriverte til den symmetriske 3.-ordrens Beta-funksjonen. For å kunne vise dem i samme figur har vi valgt å bruke en egen akse for hver av dem. Fargen på aksen er da den samme som fargen på den respektive funksjonen. Den røde kurven er den 1.-deriverte, den blå er den 2.-deriverte, den grønne er den 3.-deriverte. Merk at verdiene til alle disse tre deriverte er null ved både start, $t = 0$, og slutt, $t = 1$. Dette er fordi ordenen til Beta-funksjonen er tre. Den 4.-deriverte er plottet i stiplet svart og den 5.-deriverte er plottet i stiplet oransje, og ingen av disse to er null ved $t = 0$ og ved $t = 1$. Vi ser også at den 1.-deriverte er glatt av orden 2, den 2.-deriverte er glatt av orden 1 og at for den 2.-deriverte ser vi at funksjonen “stuper" ned mot t-aksen.

7.4 Gruppen av rasjonale B-funksjoner

I 2001 ble parametrisk G^n-blending[1] diskutert av E. Hartmann i [86]. Han introduserte en rasjonal blendingsfunksjon, forkortet til RB-funksjon.

$$b_n(t;\mu) = \frac{(1-\mu)t^{n+1}}{(1-\mu)t^{n+1} + \mu(1-t)^{n+1}},\ t \in [0,1],\ 0 < \mu < 1,\ n \geq 0.$$

Hvis $\mu = 0,5$ er kurvene punktsymmetriske om punktet (0,5, 0,5). For andre μ er kurvene asymmetriske. Derfor kalte han μ balansen til blendingsfunksjonen $b_n(t;\mu)$. Tallet n bestemmer antallet påfølgende deriverte som er null ved start og slutt.

Å bruke en balanseparameter μ kan være nyttig, og vi skal se nærmere på det senere. Men hvis vi ønsker å begrense antall formparametere til kun ordens-parameter, kan vi endre formelen litt for å få en enklere B-funksjon.

Teorem 7.4. *Den rasjonale funksjonen er en B-funksjon. Forenklet kaller vi den RB-funksjon. Den generelle formelen er*

$$B_{a,b}(t) = \frac{t^{a+1}}{t^{a+1} + (1-t)^{b+1}}, \quad \textit{for} \quad t \in [0,1], \tag{7.21}$$

og $a,b \geq 0$, $a,b \in \mathbb{Z}$, hvor $a = S_0$ og $b = S_1$ er høyre og venstre sides (hermite)orden. Hvis $a = b$ så er RB-funksjonen symmetrisk i henhold til **D5** *i definisjon 7.1.*

Beviset til teorem 7.4 finner vi i vedlegg C.4.

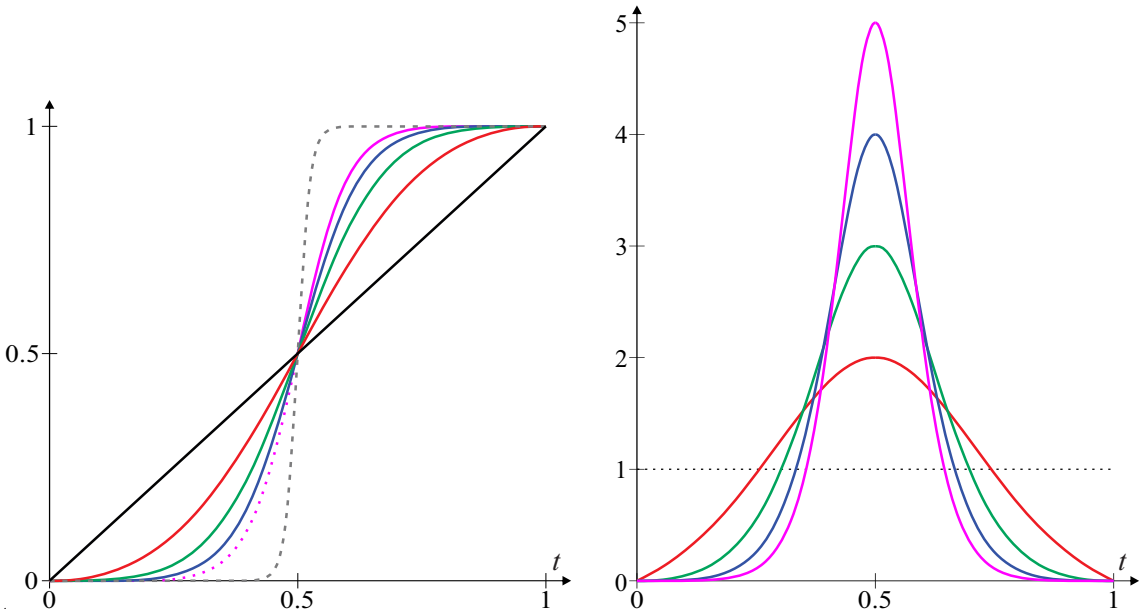
Merk at RB-funksjonen av laveste orden er lineær, dvs.

$$B_0(t) = \frac{t}{t + (1-t)} = t,$$

og at gruppen ser ut til å konvergere mot en trinnfunksjon ved $t = 0,5$. Bortsett fra den lineære funksjonen har de alle en slags S-form.

Vi skal nå se på rekken av symmetriske RB-funksjoner.

[1] G^n betyr at n deriverte er geometrisk kontinuerlig, i motsetning til C^n som betyr matematisk kontinuerlig, dvs. at lengden av deriverte-vektorer også er kontinuerlig, ikke bare retningen.



Figur 7.10: *Til venstre ser vi seks symmetriske RB-funksjoner; sort: 0.-orden, rød: 1.-orden, grønn: 2.-orden, blå: 3.-orden, lilla: 4.-orden og stiplet grå: 20.-orden. Til høyre er den 1.-ordens deriverte for de fem første RB-funksjonene plottet i tilsvarende farger.*

Symmetriske RB-funksjoner

I teorem 7.4 er det vist at en RB-funksjon $B_s(t)$ er symmetrisk, og at ordenen er S. Den generelle formelen for symmetriske RB-funksjoner er:

$$B_s(t) = \frac{t^{s+1}}{t^{s+1} + (1-t)^{s+1}}$$

De fem første symmetriske RB-funksjonene, av orden 0,1,2,3,4 er:

$$B_0(t) = \frac{t^1}{t^1+(1-t)^1} = t$$

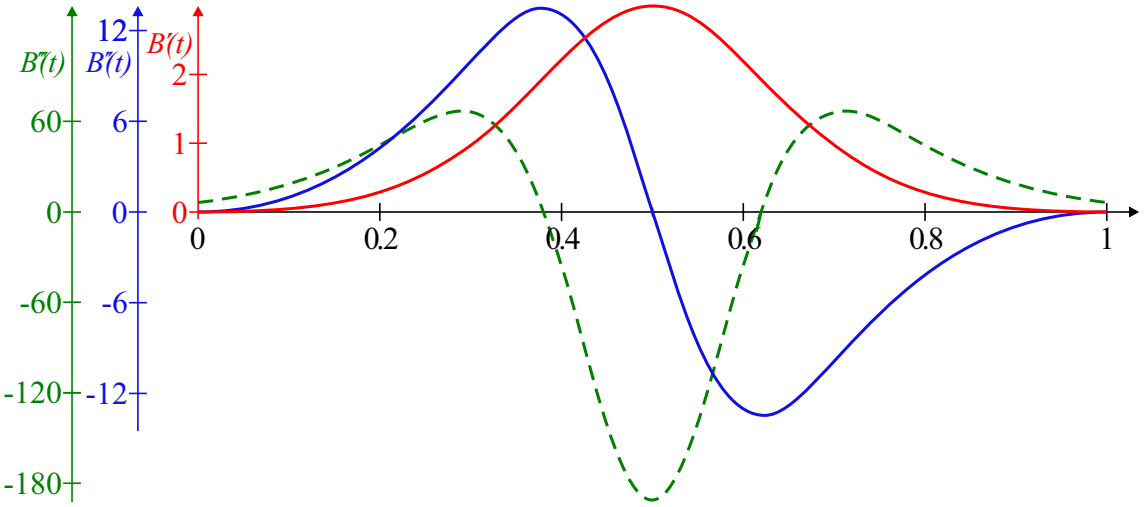
$$B_1(t) = \frac{t^2}{t^2+(1-t)^2} = \frac{t^2}{2t^2-2t+1}$$

$$B_2(t) = \frac{t^3}{t^3+(1-t)^3} = \frac{t^3}{2t^3-3t^2+3t-1}$$

$$B_3(t) = \frac{t^4}{t^4+(1-t)^4} = \frac{t^4}{2t^4-4t^3+6t^2-4t+1}$$

$$B_4(t) = \frac{t^5}{t^5+(1-t)^5} = \frac{t^5}{2t^5-5t^4+10t^3-10t^2+5t-1}$$

I figur 7.10 er de fem første symmetriske RB-funksjonene plottet sammen med den symmetriske RB-funksjonen av orden 20. Her ser vi at serien ser ut til å konvergere mot en trinnfunksjon ved $t = \frac{1}{2}$. Sammenlignet med de symmetriske Beta-funksjonene fra figur 7.8, ser vi fra plottene av førstederiverte i figur 7.10 at RB-funksjonene er brattere.



Figur 7.11: *I figuren viser tre deriverte av en 2.-ordens symmetriske RB-funksjoner; 1.-deriverte - heltrukken rød, 2.-deriverte - heltrukken blå og 3.-deriverte- stiplet grønt. Hver vertikal akse er koblet til den derivertefunksjonen med samme farge.*

7.4.1 RB-funksjoner, differensiering

De driverte til en generell RB-funksjon er ganske komplekse, spesielt høyereordens-deriverte. De to første deriverte for generelle Beta-funksjonen er:

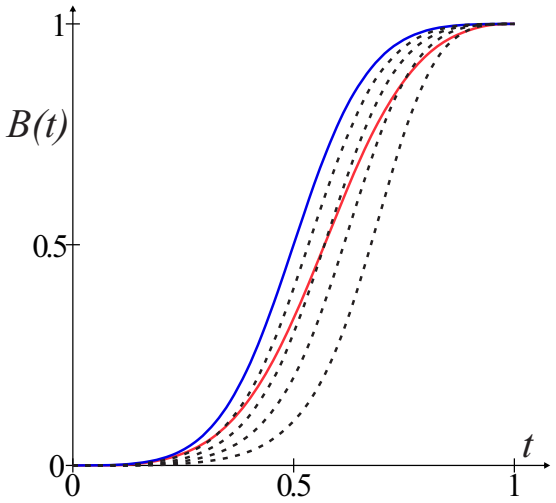
$$B'_{a,b}(t) = (a(1-t)+bt+1)\frac{t^a(1-t)^b}{(t^{a+1}+(1-t)^{b+1})^2},$$

$$B''_{a,b}(t) = \Big[\left[t^2\left(a\left(a+1\right)-b\left(b+1\right)\right)+a\left(a+1\right)\left(1-2t\right)\right]\left(t^{a+1}+\left(1-t\right)^{b+1}\right)-$$
$$2\left(a(1-t)+bt+1\right)t\left(1-t\right)\left(\left(a+1\right)t^a-\left(b+1\right)\left(1-t\right)^b\right)\Big]\frac{t^{a-1}(1-t)^{b-1}}{\left(\left(1-t\right)^{b+1}+t^{a+1}\right)^3}.$$

For konkrete RB-funksjoner er de deriverte mye enklere. For eksempel, formlene for de første tre deriverte av en symmetrisk 2.-ordens RB-funksjon er:

$$\begin{aligned}
B_2(t) &= \frac{t^3}{t^3+(1-t)^3},\\
B'_2(t) &= 3\frac{t^2(1-t)^2}{(t^3+(1-t)^3)^2},\\
B''_2(t) &= -6\frac{(2t-1)t(1-t)}{(t^3+(1-t)^3)^3},\\
B'''_2(t) &= -6\frac{18t^2(1-t)^2-1}{(t^3+(1-t)^3)^4}.
\end{aligned}$$

I figur 7.11 er tre deriverte til en symmetriske 2.-ordens RB-funksjon plottet. For å kunne vise dem i samme figur har vi brukt en egen akse for hver av dem. Fargen på aksen er den samme som fargen til den respektive derivertefunksjonen. Den heltrukken-røde er den 1.-deriverte, den heltrukken-blå er den 2.-deriverte og den stiplede grønne er den 3.-deriverte. Merk at verdien av den 1.-deriverte og 2.-deriverte er null ved start og slutt. Det følger at ordenen til denne RB-funksjonen er 2. Som vi kan se i figur 7.11 er den 3.-deriverte ikke null ved start og slutt.



Figur 7.12: *Eksempler på Rμ-funksjoner. Den blå kurven er den symmetriske B-funksjonen $B_{2,2}(t;0,5) = B_2(t)$. De fire svarte stiplede kurvene er Rμ-funksjoner hvor μ er $0,6, 0,7, 0,8$ og $0,9$. Den røde kurven er en asymmetrisk RB-funksjon, $B_{2,1}(t)$*

7.4.2 RB-funksjoner med en balanseparameter

Hvis vi inkluderer en balanseparameter til den rasjonale B-funksjonen får vi:

Korrolar 7.1. *En Rμ-funksjon er på formen*

$$B_{a,b}(t;\mu) = \frac{(1-\mu)t^{a+1}}{(1-\mu)t^{a+1} + \mu(1-t)^{b+1}}, \quad \text{for } t \in [0,1], \tag{7.22}$$

og hvor $a,b \in \mathbb{N}$, $a,b > 0$ og $0 < \mu < 1$, $\mu \in \mathbb{R}$.

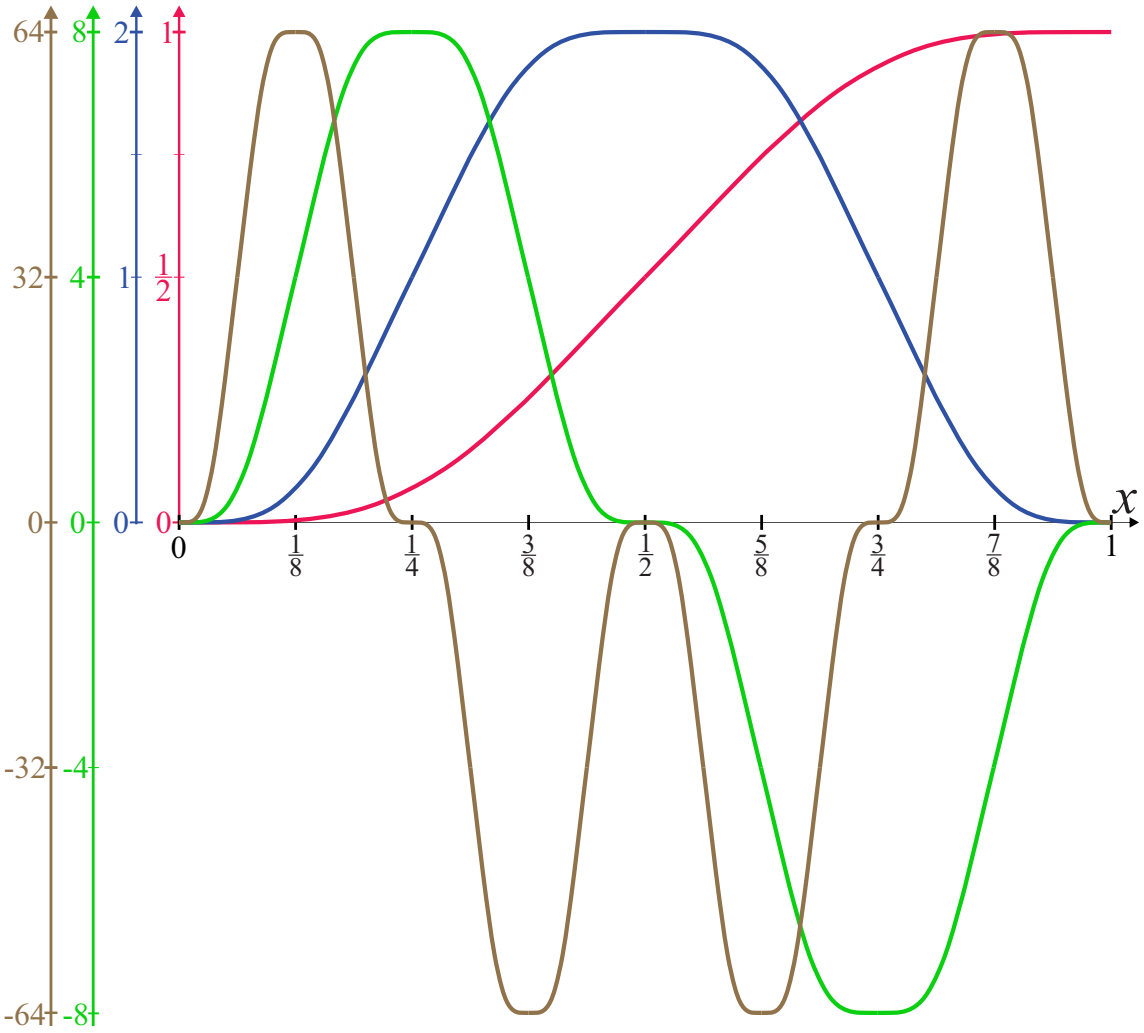
Det følger at en Rμ-funksjon er en B-funksjon på grunn av teorem 7.4.

Som for RB-functions, ordenen på venstre side $a = S_0$, og ordenen på høyre side $b = S_1$. Hvis $a = b$ og $\mu = 0.5$ da er Rμ-funksjonen symmetrisk i henhold til **D5** *i definisjon 7.1.*

For å sammenligne Rμ-funksjoner med RB-funksjoner skal vi først se på en 2.-ordens symmetrisk RB-funksjon $B_2(t)$. Vi gjør demme RB-funksjonen om til en Rμ-funksjon der $a = b = 2$ og $\mu = 0.5$, dvs. $B_{2,2}(t;0.5)$. Så endrer vi μ og ser hva som skjer.

I figur 7.12 er funksjonen, $B_{2,2}(t;0.5) = B_2(t)$ plottet i blått. Vi ser også fire funksjoner plottet i stiplet svart. Dette er $B_{2,2}(t;0,6)$, $B_{2,2}(t;0,7)$, $B_{2,2}(t;0,8)$ og $B_{2,2}(t;0,9)$. Som vi ser, skyves de mot høyre i forhold til verdien av μ. Husk at alle fem funksjonene er av hermiteorden 2 på begge sider, men bare den blå er symmetrisk.

For å se effekten av balanseparameteren μ viser vi også en asymmetrisk RB-funksjon $B_{2,1}(t) = \frac{t^3}{t^3+(1-t)^2}$ plottet i rødt i figur 7.12. Balanseparameteren μ påvirker ikke brattheten til funksjonen, mens hermiteordenen påvirker brattheten. I avsnitt 7.8, skal vi se nærmere på det vi kaller “balansesymmetri" av B-funksjoner.



Figur 7.13: *Et plott av Fabius-funksjonen (rød) og dens 1.-deriverte (blå), 2.-deriverte (grønn) og 3.-deriverte (brun). De vertikale aksene til venstre har samme farger som plottene de er koblet til.*

7.5 Fabius-funksjonen, den komplette B-funksjonen

H. Olofsen introduserte Fabius-funksjonen som B-funksjon i [127]. Fabius-funksjonen, introdusert av Jaap Fabius [62], er en uendeligdifferensierbar, men ingensteds analytisk funksjon som er selv-differensial. Fabius-funksjonen tilfredsstiller differensialligningen:

$$\mathfrak{F}'(x) = 2\mathfrak{F}(2x), \quad x \in \mathbb{R}+ \quad (\text{som er } [0,+\infty)). \tag{7.23}$$

Det følger at $\mathfrak{F}(x+2^n) = -\mathfrak{F}(x)$ for $0 \leq x \leq 2^n$ der n er et positivt heltall. Sekvensen av intervall der funksjonen er positiv eller negativ, følger samme mønster som Thue-Morse-sekvensen [3]. Figur 7.13 viser Fabius-funksjonen og dens 1.-, 2.- og 3.-deriverte på $0 \leq t \leq 1$. Vi ser at de deriverte er smalere og skalerte (pluss eller minus) kopier av selve funksjonen. Skaleringen av de deriverte gjenspeiles ved at $\max|\mathfrak{F}^{(j)}(x)| = 2^{\Sigma_{i=1}^{j} i}$. dvs. $2, 8, 64, 1024, \ldots$, på grunn av (7.23), og som vi kan observeres i figur 7.13.

Hvis vi begrenser domenet til $[0,1]$, ser vi at Fabius-funksjonen tilfredsstiller startbetingelsen $\mathfrak{F}(0)=0$, symmetribetingelsen $\mathfrak{F}(x)+\mathfrak{F}(1-x)=1$, dvs. den er punktsymmetrisk om $\left(\frac{1}{2},\frac{1}{2}\right)$. Det følger at $\mathfrak{F}(x)$ er monotont økende, med $\mathfrak{F}(\frac{1}{2})=\frac{1}{2}$ og $\mathfrak{F}(1)=1$. Et uttrykk for Fabius-funksjonen for $x\in[0,1]$, er

$$\mathfrak{F}(x)=\begin{cases}\displaystyle\int_0^{2x}\mathfrak{F}(s)ds, & \text{for } x\in\left[0,\tfrac{1}{2}\right],\\[2ex] \displaystyle\int_{2x-1}^{1}\mathfrak{F}(s)ds+2x-1, & \text{for } x\in\left(\tfrac{1}{2},1\right].\end{cases}\tag{7.24}$$

Fabius-funksjonen oppfyller helt klart alle kravene for å være en symmetrisk B-funksjon, Alle de 5 punktene fra definition 7.1 er oppfylt. Det er imidlertid en ekstra egenskap som gjør Fabius-funksjonen spesiell. Fordi alle deriverte er en halv ganger smalere og skalerte versjoner av selv (skalert med 2^n langs x-aksen) som følger av (7.23) og (7.24) og som vi også ser i figur 7.13 er alle deriverte null ved $t=0$ og $t=1$. Det følger at hermiteordenen er ∞, og det fører til følgende definisjon:

Den komplette B-funksjonen

Definisjon 7.3. *En komplett B-funksjon er en funksjon som oppfyller kravene i Definisjon 7.1, og hvor alle deriverte ved $t=0$ og $t=1$ er null. Dermed er ordenen til en komplett B-funksjon ∞.*

- En komplett B-funksjon er ikke analytisk ved $t=0$ og $t=1$. Det følger at Taylor-ekspansjonen ikke fungerer i disse to punktene.
- Fabius-funksjonen er en komplett B-funksjon, men er i tillegg til å være en komplett B-funksjon ikke-analytisk overalt.

I tillegg til å være ikke-analytisk overalt er det enda et praktisk problem med å bruke Fabius-funksjonen som B-funksjon, vi kan ikke beregne verdien direkte hvor som helst. Det er kun mulig å beregne med dyadiske rasjonale tall på formen $\frac{m}{2^n}$, der n er et positivt heltall $1,2,3,\ldots$ og m er et positivt oddetall $1,3,...,2^n-1$, som vist av Jan K. Haugland i 2016 i [88]. Beregninger av (7.24) for n opptil 5 gir oss følgende tall,

$$\begin{aligned}
&\mathfrak{F}\left(\tfrac{1}{2}\right)=\tfrac{1}{2}\\
&\mathfrak{F}\left\{\tfrac{1}{4},\tfrac{3}{4}\right\}=\left\{\tfrac{5}{72},\tfrac{67}{72}\right\}\\
&\mathfrak{F}\left\{\tfrac{1}{8},\tfrac{3}{8},\tfrac{5}{8},\tfrac{7}{8}\right\}=\left\{\tfrac{1}{288},\tfrac{73}{288},\tfrac{215}{288},\tfrac{287}{288}\right\}\\
&\mathfrak{F}\left\{\tfrac{1}{16},\tfrac{3}{16},\ldots,\tfrac{15}{16}\right\}=\left\{\tfrac{143}{2073600},\tfrac{46657}{2073600},\tfrac{305857}{2073600},\tfrac{777743}{2073600},\tfrac{1295857}{2073600},\tfrac{1767743}{2073600},\tfrac{2026943}{2073600},\tfrac{2073457}{2073600}\right\}\\
&\mathfrak{F}\left\{\tfrac{1}{32},\tfrac{3}{32},\ldots,\tfrac{31}{32}\right\}=\left\{\tfrac{19}{33177600},\tfrac{25219}{33177600},\tfrac{334781}{33177600},\tfrac{1396781}{33177600},\tfrac{3470381}{33177600},\tfrac{6555581}{33177600},\tfrac{10393219}{33177600},\right.\\
&\qquad\left.\tfrac{14515219}{33177600},\tfrac{18662381}{33177600},\tfrac{22784381}{33177600},\tfrac{26622019}{33177600},\tfrac{29707219}{33177600},\tfrac{31780819}{33177600},\tfrac{32842819}{33177600},\tfrac{33152381}{33177600},\tfrac{33177581}{33177600}\right\}.
\end{aligned}$$

For å beregne Fabius-funksjonen generelt kan man bruke interpolasjon mellom disse dyadiske rasjonelle tallene eller/og bruke polynomiske approksimasjoner av Fabius-funksjonen, se vedlegg A og/eller [127].

7.6 Gruppen av trigonometriske B-funksjoner

TB-funksjon er forkortelse for trigonometriske B-funksjoner. I 2019 introduserte Hans Olofsen blendingsfunksjoner basert på trigonometriske og polynomiske approksimasjoner av Fabius-funksjonen [127]. Han viste oss hvordan vi kan konstruere trigonometriske B-funksjoner av stigende orden ved å approksimere Fabius-funksjonen. En sum av vektede cosinus kan approksimere Fabius–funksjonen på domenet $[0,2]$, se [42, eq.(30)]. Dermed får vi

$$F_M(t) = \frac{1}{2} - \sum_{m=1}^{M} c_m \cos(m\pi t), \quad t \in [0,1]. \tag{7.25}$$

For å kunne utvikle trigonometriske B-funksjoner som er punktsymmetriske, må vi følge disse tre punktene;

- Hvis m er oddetall, er $\cos(m\pi t)$ punktsymmetrisk om punktet $\left(\frac{1}{2},\frac{1}{2}\right)$. Hvis m er partall, er $\cos(m\pi t)$ symmetrisk om den vertikale aksen $x = \frac{1}{2}$. Derfor, for å sikre punktsymmetri, må m kun være oddetall dvs. $\{1,3,5,\ldots\}$, og følgelig må M også være oddetall.
- For å sikre at $F(0) = 0$ og $F(1) = 1$ må $\sum c_m = \frac{1}{2}$ for $m = 1,3,...,M$.
- Ordenen til B-funksjonen er koblet til at alle 1.-, 2.-,..., M.-deriverte er null ved $t = 0,1$. Dette er oppfylt hvis $\sum m^j c_m = 0$ for $m = 1,3,...,M$ og $j = 2,4,...,M-1$.

Vi skal se på et par eksempler på hvordan man bygger opp symmetriske B-funksjoner. I (7.25), for å finne c_m når $m = 1,3$ og c_m når $m = 1,3,5$ lager vi en $\frac{M+1}{2} \times \frac{M+1}{2}$ matrise og beregne deretter c_m for $M = 3$ og så for $M = 5$ på følgende måte,

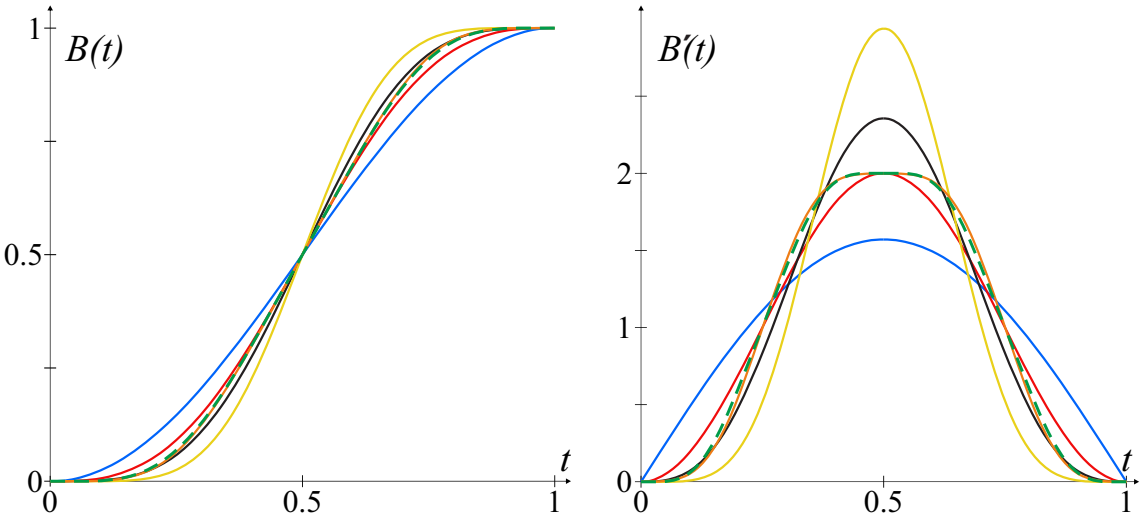
$$M=3 \;\Rightarrow\; \begin{pmatrix} 1 & 1 \\ 1^2 & 3^2 \end{pmatrix}\begin{pmatrix} c_1 \\ c_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix} \;\Rightarrow\; \begin{pmatrix} c_1 \\ c_3 \end{pmatrix} = \begin{pmatrix} \frac{9}{16} \\ -\frac{1}{16} \end{pmatrix},$$

$$M=5 \;\Rightarrow\; \begin{pmatrix} 1 & 1 & 1 \\ 1^2 & 3^2 & 5^2 \\ 1^4 & 3^4 & 5^4 \end{pmatrix}\begin{pmatrix} c_1 \\ c_3 \\ c_5 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \end{pmatrix} \;\Rightarrow\; \begin{pmatrix} c_1 \\ c_3 \\ c_5 \end{pmatrix} = \begin{pmatrix} \frac{75}{128} \\ -\frac{25}{256} \\ \frac{3}{256} \end{pmatrix}.$$

Resultatet er trigonometriske B-funksjoner av oddetall, Vi får orden 1 med koordinatene $c_1 = \frac{1}{2}$, orden 3 med koordinatene $c_1 = \frac{9}{16}$ og $c_3 = -\frac{1}{16}$, og orden 5 med koordinatene $c_1 = \frac{75}{128}$, $c_3 = -\frac{25}{256}$ og $c_5 = \frac{3}{256}$.

For å få trigonometriske B-funksjoner av partallssorden, må vi bruke B-funksjoner av oddetallsorden $M = 1,3,5,\ldots$ som den første deriverte. Dette kan gjøres fordi $\cos m\pi t$ er symmetrisk om den vertikale aksen $x = 1$ for $m = 1,3,5,\ldots$. Det følger at de deriverte ved $t = 2$ også er null for M etterfølgende deriverte. Dermed er det mulig å få en B-funksjon av orden $m+1$, $B_{m+1}(t)$ fra en B-funksjon av orden m, dvs. $B_m(t)$ når $m = 1,3,5,\ldots$.

Metoden er å bruke den antiderivert til $B_m(t)$, $m = 1,3,...$ skalert med 2, fordi (7.23) angir at maksimumsverdien til den første deriverte av Fabius-funksjonen er 2, og vi bruker den på domenet $[0,2]$ fordi parameteren er skalert med 2 i (7.23). Resultatet er $B_n(t)$, $n = 2,4,...$ på domenet $[0,1]$.



Figur 7.14: *Til venstre er Fabius-funksjonen plottet i stiplet grønt, samt fem trigonometriske B-funksjoner, orden 1 i blått, 2 i rødt, 3 i svart, 4 i oransje og 5 i gult. På høyre side er plottene til de 1.-deriverte i samme farge som deres respektive funksjoner.*

Vi skal se på et eksempel; vi starter med $B_1(t) = \frac{1}{2} - \frac{1}{2}\cos \pi t$. Vi erstatter deretter t med $2t$, og vi multipliserer funksjonen med 2. Resultatet er

$$B_2'(t) = 1 - \cos 2\pi t.$$

Vi bruker deretter den antideriverte (integral), og får,

$$B_2(t) = t - \frac{1}{2\pi}\sin 2\pi t, \quad t \in [0,1].$$

I figur 7.14 på venstre side er det plott av 5 trigonometriske B-funksjoner (av orden 1,2,3,4,5) sammen med Fabius-funksjonen. Til høyre er deres 1.-deriverte plottet.

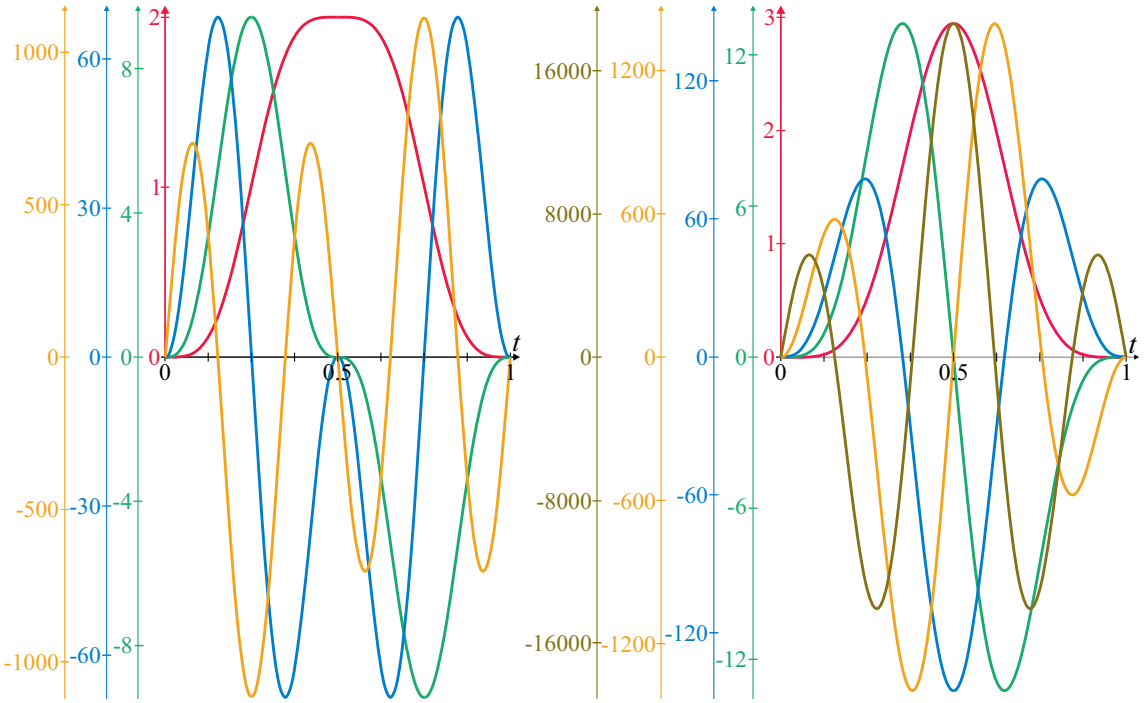
Symmetric TB-functions

En serie symmetriske TB-funksjoner som er oppnådd ved å approksimere Fabius-funksjonen. Formlene for de første fem symmetriske TB-funksjonene er:

$$\begin{aligned}
B_1(t) &= \tfrac{1}{2} - \tfrac{1}{2}\cos \pi t && \text{orden 1} \\
B_2(t) &= t - \tfrac{1}{2\pi}\sin 2\pi t && \text{orden 2} \\
B_3(t) &= \tfrac{1}{2} - \tfrac{1}{16}(9\cos \pi t - \cos 3\pi t) && \text{orden 3} \\
B_4(t) &= t - \tfrac{1}{48\pi}(27\sin 2\pi t - \sin 6\pi t) && \text{orden 4} \\
B_5(t) &= \tfrac{1}{2} - \tfrac{1}{256}(150\cos \pi t - 25\cos 3\pi t + 3\cos 5\pi t) && \text{orden 5}
\end{aligned} \tag{7.26}$$

Merk at $B_1(t)$ er den samme funksjonen som er plottet i b) i figur 7.1 og brukt i sirkelsplines (seksjon 5.8), dvs.

$$\sin^2 \frac{\pi t}{2} = \frac{1}{2} - \frac{1}{2}\cos \pi t.$$



Figur 7.15: *På venstre side ser vi plottene av fire deriverte av $B_4(t)$, $B_4'(t)$-rød, $B_4''(t)$-grønn, $B_4'''(t)$- blå og $B_4^{(4)}(t)$-oransje. Til høyre er plottene av fem deriverte av $B_5(t)$, $B_5'(t)$-rød, $B_5''(t)$-grønn, $B_5'''(t)$-blå, $B_5^{(4)}(t)$-oransje og $B_5^{(5)}(t)$-brun.*

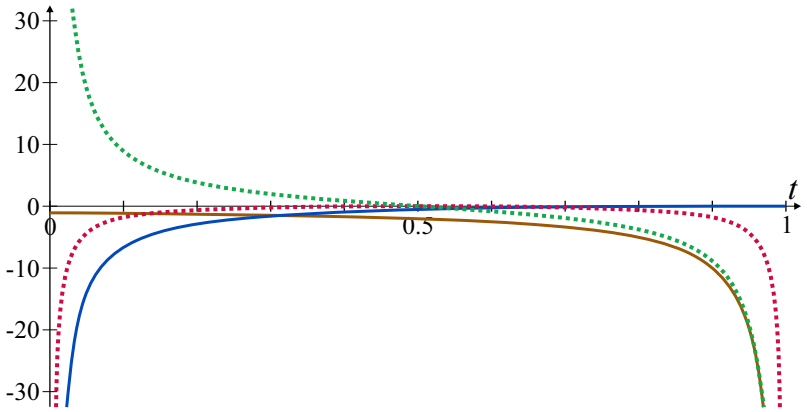
Derivasjon av de trigonometriske B-funksjonene i (7.26) er en enkel og lite resurskrevende operasjon å utføre både på en CPU og på en GPU. Til venstre i figur 7.15 er de fire første etterfølgende deriverte av $B_4(t)$ plottet, og til høyre er de fem første deriverte, dvs. fra 1 til 5 av $B_5(t)$ plottet. I begge settene i figur 7.15 har hver derivertefunksjon en gitt farge. De forskjellige vertikale aksen til venstre er da koblet til de respektive deriverte som har den samme fargen som de vertikale aksene.

Plottet i figur 7.15 bekrefter helt klart at hermiteordenen til de to funksjonene er fire og fem fordi alle deriverte i plottene er null ved $t = 0$ og for $t = 1$. En annen observasjon er at $B_4(t)$ er funksjonen som ser ut til å være den beste approksimasjonen av Fabius-funksjonen, noe vi ser fra en kombinasjon av figurene 7.13, 7.14 og 7.15. Vi ser også at den maksimale absoluttverdien av deriverte for $B_4(t)$ øker bare litt mer enn tilsvarende for Fabius-funksjonen når ordenen på de deriverte øker. For $B_5(t)$ økte absoluttverdien mye mer.

Ett åpent spørsmål reist av Olofsen; er Fabius-funksjonen den komplette B-funksjonen hvor den maksimale absoluttverdien av de deriverte øker minst når ordenen til de deriverte øker? Det vil si, er

$$\max|\mathfrak{F}^{(j)}(t)| \leq \max|B_c^{(j)}(t)|, \quad 0 \leq t \leq 1,$$

når $j \to \infty$, og for alle komplette B-funksjoner $B_c(t)$?



Figur 7.16: *Plott av eksponentene til de ekspo-rasjonale funksjonene, (7.27) stiplet rødt, (7.28) brunt og blått, og (7.30) prikket grønt.*

7.7 Gruppen av ekspo-rasjonale B-funksjoner

Egenskapene til eksponentialfunksjonen er nyttige når du skal bygge en B-funksjon. Den er alltid positiv og den er sin egen deriverte, og den går alltid raskere mot $+\infty$ enn enhver potens av x når $x \to +\infty$, dermed overstyrer den også enhver rasjonal polynomfunksjon når den går mot null[2]. Legger vi til en rasjonal eksponent eller, en hierarkisk modell sammen med en rasjonal eksponent, kan vi få flere forskjellige ekspo-Rasjonale B-funksjoner. Av spesiell interesse er de ekspo-rasjonale funksjonene og deres 1.-deriverte nedenfor,

$$\phi(t) = e^{-\frac{\left(t-\frac{1}{2}\right)^2}{t(1-t)}}, \qquad \phi'(t) = \frac{1-2t}{4(t(1-t))^2}\phi(t), \tag{7.27}$$

$$\psi(t) = e^{\frac{-2}{t}e^{\frac{-1}{1-t}}}, \qquad \psi'(t) = 2\frac{t^2-t+1}{t^2(1-t)^2}e^{\frac{-1}{1-t}}\psi(t), \tag{7.28}$$

$$\theta(t) = e^{\frac{1}{t}}, \qquad \theta'(t) = -\frac{1}{t^2}\theta(t), \tag{7.29}$$

$$\varphi(t) = \frac{\theta(t)}{\theta(1-t)} = e^{\frac{1}{t}-\frac{1}{1-t}}, \qquad \varphi'(t) = -\left(\frac{1}{t^2}+\frac{1}{(1-t)^2}\right)\varphi(t). \tag{7.30}$$

En eksponentialfunksjon avbilder normalt $\mathbb{R}$ til $\mathbb{R}+$. Funksjonene i (7.27), (7.28) og (7.30) begrenser imidlertid domenene til segmentet $(0,1)$. I figur 7.16 er de rasjonale eksponentene fra de nevnte tre funksjonene plottet. Den stiplede røde kurven er eksponenten i (7.27), som tilordner $(0,1)$ til $\mathbb{R}$-, den er symmetrisk om aksen $t = \frac{1}{2}$. Resultatet er at $\phi(t)$ blir en hatt-funksjon med maksverdi 1, og at eksponenten når $t = 0,\ 1$ er $-\infty$, som jo ikke er $\in \mathbb{R}$ men er i det utvidede reelle tallsystemet. Følgelig er $\phi(t)$ ikke analytisk ved $t = 0,\ 1$. Det samme gjelder for $\psi(t)$ (7.28). I figur 7.16 er det brune og det blå plottet den øvre og den totale eksponenten til $\psi(t)$. Den blå kommer fra $-\infty$ og den brune går til $-\infty$ og begge er negative over alt. Følgelig er også $\psi(t)$ ikke analytisk ved $t = 0,\ 1$. Det stiplede grønne kurven er eksponenten til $\varphi(t)$ (7.30) som avbilder $(0,1)$ til $\mathbb{R}$, og er punktsymmetrisk rundt $((0,5),0)$. Følgelig er heller ikke $\varphi(t)$ analytisk ved $t = 0,\ 1$.

[2] $\lim_{x\to+\infty} x^n e^{-x} = 0$ for hver n og $e^0 = 1$, $e^{a+b} = e^a e^b$, $e^a = \frac{1}{e^{-a}}$ og $\mathbb{R}+$ er $[0,+\infty)$, $\mathbb{R}$- er $(-\infty,0]$.

I 2004 ble ekspo-rasjonale B-splines presentert på konferansen for Mathematical metoder for kurver og flater av Dechevsky, Lakså og Bang, og senere publisert i [105],[44],[45] og [102]. Det blei opprinnelig brukte en ekspo-rasjonal funksjon med flere iboende parametere, men standardfunksjon som blei brukt var $\phi(t)$ fra (7.27). Denne funksjonen er relatert til den gaussiske klokkefunksjonen og den kumulative fordelingsfunksjonen, se [74]. I 2015 ble ekspo-rasjonale B-splines satt inn i rammen av B-funksjoner [104]. Uttrykket for det vi kan kalle type 1 ERB-funksjon, er:

$$B_d(t) = \mathbb{S} \int_0^t \phi(s)ds, \quad t \in [0,1], \tag{7.31}$$

hvor den ekspo-rasjonale funksjonen $\phi(t)$ er gitt i (7.27) og skaleringen er

$$\mathbb{S} = \left[\int_0^1 \phi(t)dt \right]^{-1} \approx 1.6571376797382103. \tag{7.32}$$

Fordi det ikke finnes noen analytisk løsning på integralet i (7.31), er dette en beregningskrevende formel. Men i 2007 ble en approksimativ beregningsalgoritme med toleranser bedre enn $3,4e-13$ ($L^\infty[0,1]$ og dobbel presisjon) publisert, hvor antall multiplikasjoner er 6 for funksjonsverdien, 4 for 1.-deriverte og 5 for 2.-deriverte. Metoden og algoritmen er beskrevet i vedlegg A.4 og mer detaljert i [102].

I 2004 publiserte Ying og Zorin et arbeid som handlet om å lage “fullstendig" glatte flater [168]. Artikkelen var basert på Grimm og Hughes arbeid fra 1995 [82], og Navau og Garcia fra 2000 [126] om samme emne. Navau og Garcia brukte en basisfunksjon (B-funksjon) som ikke var punktsymmetrisk i henhold til Definisjon 7.1. Denne funksjonen var (7.28). Ying og Zorin laget en symmetrisk B-funksjon ved å kombinere denne ikke-symmetriske funksjonen med den rasjonale konstruksjonen til Hartmann gitt i 7.21, dvs.

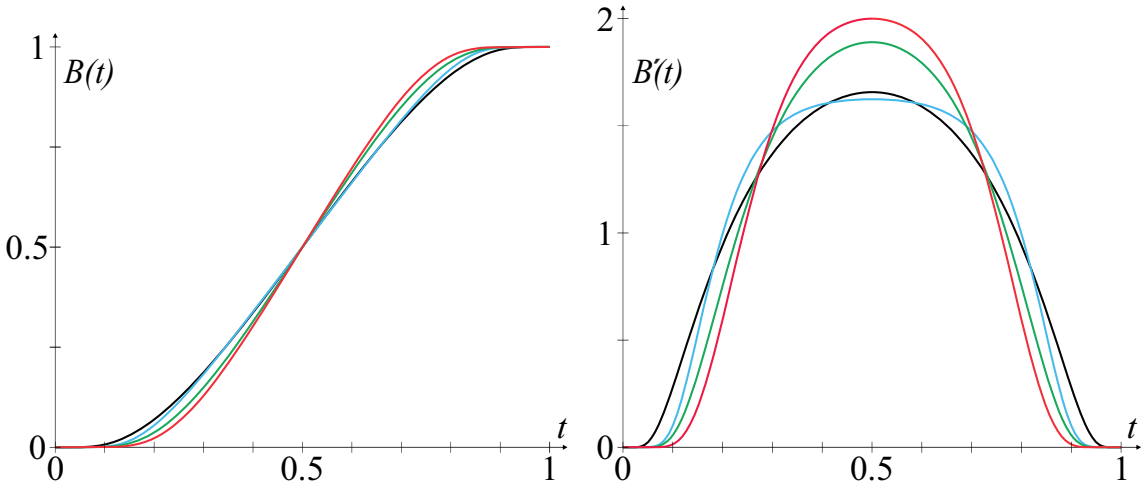
$$B_y(t) = \frac{\psi(t)}{\psi(t) + \psi(1-t)}. \tag{7.33}$$

Det finnes imidlertid en standard måte å lage en symmetrisk B-funksjon fra en ikke-symmetrisk B-funksjon. Det er å summere funksjonen med sin antisymmetriske tvilling og dele resultatet på to. Hermiteordenen til den resulterende B-funksjonen vil være lik den laveste ordenen av den orginale ikke-symmetriske B-funksjonen. Hvis vi bruker denne metoden på (7.30), får vi følgende symmetriske ERB-funksjon,

$$B_x(t) = \frac{1}{2}\left(1 - \psi(1-t) + \psi(t)\right). \tag{7.34}$$

I 2013 introduserte Zanaty og Dechevsky en logistisk ERB-funksjon [46], som tok utgangspunkt i enten funksjonen $\theta(t)$, (7.29) eller $\varphi(t)$, (7.30). Følgende brøker gir LERB-funksjonen,

$$B_z(t) = \frac{\theta(1-t)}{\theta(1-t) + \theta(t)} = \frac{1}{\varphi(t)+1}, \tag{7.35}$$



Figur 7.17: *Til venstre i figuren ser vi fire ekspo-rasjonale B-funksjoner, $B_d(t)$ i svart (7.31), $B_y(t)$ i blått (7.33), $B_x(t)$ i grønt (7.34) og den logistiske ERB-funksjonen $B_z(t)$ i rødt (7.35). På høyre side er det et plott av de 1.-deriverte til disse fire ERB-funksjonene i samme farge som deres respektive funksjoner på høyre side.*

som faktisk er en standard logistisk funksjon med en sammentrekning av domenet fra $\mathbb{R}$ til det åpne intervallet $(0,1)$. Når vi følger de definerte ERB-funksjonene får vi de 1.-deriverte av $B_d(t)$ (7.31), $B_y(t)$ (7.33), $B_x(t)$ (7.34) og $B_z(t)$ (7.35):

$$B_d'(t) = \mathbb{S}\,\phi(t), \qquad B_y'(t) = \frac{\psi'(t)B_y(1-t) + \psi'(1-t)B_y(t)}{\psi(t)+\psi(1-t)}, \tag{7.36}$$

$$B_x'(t) = \frac{1}{2}\left(\psi'(1-t) + \psi'(t)\right), \qquad B_z'(t) = -\varphi'(t)B_z(t)^2. \tag{7.37}$$

Til venstre i figur 7.17 er det et plott av de fire funksjonene $B_d(t)$ i svart, $B_y(t)$ i blått, $B_x(t)$ i grønt og $B_z(t)$ i rødt sammen med deres 1.-deriverte på høyre side.

Ekspo-rasjonale B-funksjoner

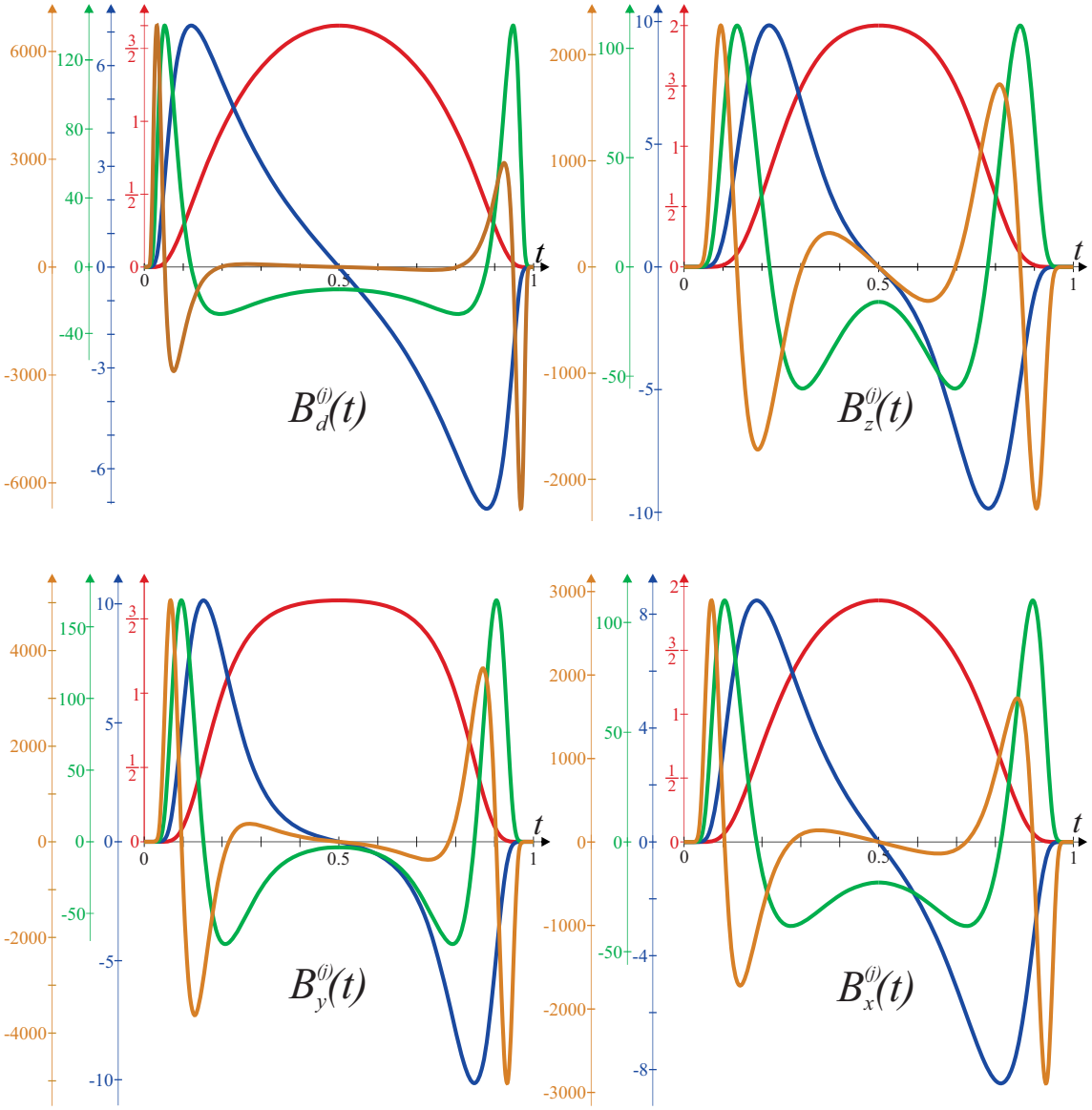
Definisjon 7.4. *En ekspo-rasjonal B-funksjon (ERB-funksjon) er:*

- *en B-funksjon i henhold til definisjon 7.1,*
- *det er en komplett B-funksjon i henhold til definisjon 7.3 (orden ∞),*
- *er ikke analytisk ved $t = 0$ og $t = 1$,*
- *og er konstruert ved hjelp av eksponentialfunksjoner med rasjonal eksponent.*

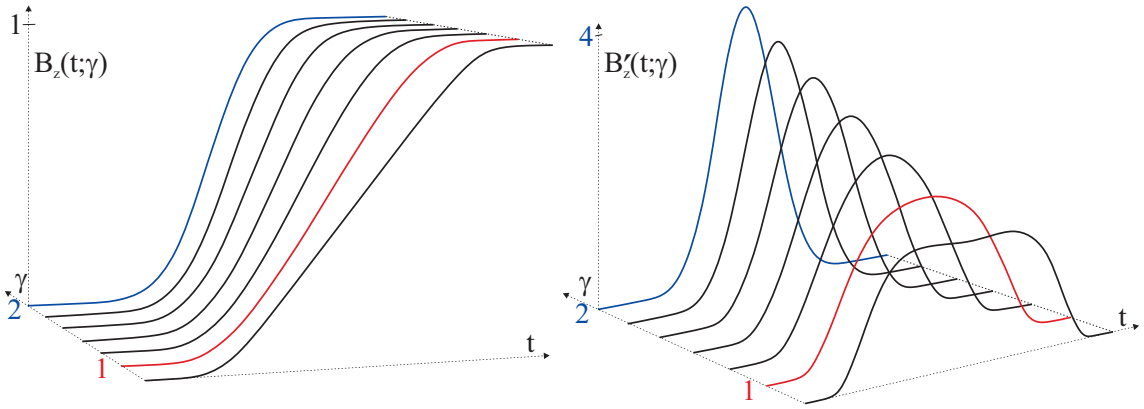
Teorem 7.5. *$B_d(t)$ definert i (7.31), $B_y(t)$ definert i (7.33), $B_x(t)$ definert i (7.34) og $B_z(t)$ definert i (7.35) er alle symmetriske ekspo-rasjonale B-funksjoner (ERB-funksjoner).*
Det betyr at de er komplette B-funksjoner, med orden ∞, og de er ikke analytiske ved $t = 0$ og $t = 1$.

Beviset til teorem 7.5 finner vi i vedlegg C.5.

I figur 7.18 er deriverte av de fire ERB-funksjonene plottet. Øverst til venstre ser vi de fire første deriverte av $B_d(t)$, (7.31), på øvre høyre side har vi de fire første deriverte av $B_z(t)$,



Figur 7.18: *I figuren er fire deriverte av fire ERB-funksjoner plottet. Øverst på venstre side ser vi de fire første deriverte av $B_d(t)$, (7.31), på øvre høyre side ser vi de fire første derivert av $B_z(t)$, (7.35), på nedre venstre side ser vi de fire første deriverte av $B_y(t)$, (7.33) og på nedre høyre side ser vi de fire første deriverte av $B_x(t)$, (7.34). 1.-deriverte er plottet i rødt, 2.-deriverte er plottet i blått, 3.-deriviverte er plottet i grønt og 4.-deriverte er plottet i oransje. Aksene på venstre side av hvert plott har samme farge som deres respektive funksjoner.*



Figur 7.19: *Til venstre ser vi et plott av $B_z(t;\gamma)$ for $\gamma = \{0.8, 1, 1.2, 1.4, 1.6, 1.8, 2\}$. $B_z(t;\gamma = 1) = B_z(t)$ er plottet i rødt og $B_z(t;2)$ er plottet i blått og alle de andre i svart. På høyre side er de respektive 1.-deriverte plottet i samme farger.*

(7.35), på nedre venstre side ser vi de fire første deriverte av $B_y(t)$, (7.33) og på nedre høyre side ser vi de fire første deriverte av $B_x(t)$, (7.34). 1.- deriverte er plottet i rødt, 2.-deriverte i blått, 3.- deriverte i grønt og 4.- deriverte er plottet i oransje. Generelt harde de deriverte samme oppførsel for alle de fire ERB-funksjonene. Men det er noen små forskjeller. For $B_d(t)$ skjer endringene av de deriverte nærmere $t = 0$ og $t = 1$ enn for de andre, og også den maksimale verdien av de deriverte vokser raskere med økende orden på de deriverte. På motsatt side er $B_z(t)$ som ser ut til å ligge nærmest Fabius-funksjonen.

I de følgende seksjoner skal vi introdusere fire iboende parameter til ERB-funksjonene.

7.7.1 Stigningsparameteren γ

Den første iboende parameteren vi skal se på er stigningsparameteren γ. Vi kan legge til en stigningsparameteren til (7.27), (7.28) og (7.30) og får da,

$$\phi(t;\gamma) = e^{-\gamma\frac{\left(t-\frac{1}{2}\right)^2}{t(1-t)}}, \quad \psi(t;\gamma) = e^{\frac{-2\gamma}{t}e^{\frac{-\gamma}{1-t}}}, \quad \varphi(t;\gamma) = e^{\gamma\left(\frac{1}{t}-\frac{1}{1-t}\right)}, \quad \gamma \in \mathbb{R},\ \gamma > 0. \tag{7.38}$$

Effekten er at vi kan justere stigningen på ERB-funksjonene. Merk at ERB-funksjonene fortsatt er (punkt)symmetriske. Figur 7.19 illustrerer effekten av stigningsparameteren. Funksjon $B_z(t;\gamma)$ er brukt som eksempel. Funksjonen og dens 1.-deriverte er plottet for et sett med stigningsparametere. Funksjonene vises til venstre i figuren og de 1.-deriverte vises på høyre side. Den slakkeste ERB-funksjonen har stigningsparameter $\gamma = 0,8$ er plottet i svart. Plottet i rødt er standardfunksjonen med $\gamma = 1$. Så følger funksjoner med $\gamma = 1,2$, $\gamma = 1,4$, $\gamma = 1,6$ og $\gamma = 1,8$, alle plottet i svart. Til slutt ser vi en funksjon med $\gamma = 2$ plottet i blått. Når vi ser på 1.-deriverte ser vi tydelig at stigningen blir brattere ettersom γ vokser. For $\gamma = 1$ er maksimumsverdien 2, mens for $\gamma = 2$ er maksimumsverdien 4. For alle fire ERB-funksjonene er oppførselen ganske lik. Merk at for $B_d(t,\gamma)$ vil konstanten $\mathbb{S}$ også bli påvirket av γ, dvs. $\mathbb{S}_\gamma = \left[\int_0^1 \phi(t;\gamma)dt\right]^{-1}$.

For $\psi(t;\gamma)$ (7.38), er det mulig å bruke to stigningsparametere, dvs.

$$\psi(t;\gamma_1,\gamma_2) = e^{\frac{-\gamma_1}{t}e^{\frac{-\gamma_2}{1-t}}}.$$

Standardsammenhengen mellom dem er da $\gamma_1 = 2\gamma_2$. Det er også mulig å koble dem, dvs. $\gamma_1 = 1+\gamma$ og $\gamma_2 = 1-\gamma$.

Stigningsparameteren har samme effekt på formen til B-funksjonen som ordenen til de symmetriske Beta-funksjonene og de symmetriske RB-funksjonene.

7.7.2 Balanseparameteren μ

Den neste iboende parameteren er balanseparameteren μ. Denne er analog med balanseparameteren i Rμ-funksjonen, se Corollary 7.1 i seksjon 7.4.2. Vi legger så til en balanseparameter til (7.27), (7.28) og (7.30) og får,

$$\phi(t;\gamma,\mu) = e^{-\gamma\frac{(t-\mu)^2}{t(1-t)}}, \qquad \gamma,\mu\in\mathbb{R}, \quad \gamma>0 \quad \text{og} \quad 0<\mu<1, \tag{7.39}$$

$$\psi(t;\gamma,\mu) = (1-\mu)\, e^{\frac{-2\gamma}{t}e^{\frac{-\gamma}{1-t}}}, \qquad \gamma,\mu\in\mathbb{R}, \quad \gamma>0 \quad \text{og} \quad 0<\mu<1, \tag{7.40}$$

$$\varphi(t;\gamma,\mu) = e^{2\gamma\left(\frac{1-\mu}{t}-\frac{\mu}{1-t}\right)}, \qquad \gamma,\mu\in\mathbb{R}, \quad \gamma>0 \quad \text{og} \quad 0<\mu<1. \tag{7.41}$$

Merk at balanseparameteren gir følgende formler for (7.33) og (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 $B_d(t;\mu)$ er formelen rett frem, bortsett fra at vi må beregne $\mathbb{S}_\mu$. For $B_z(t;\mu)$ er formelen også rett frem.

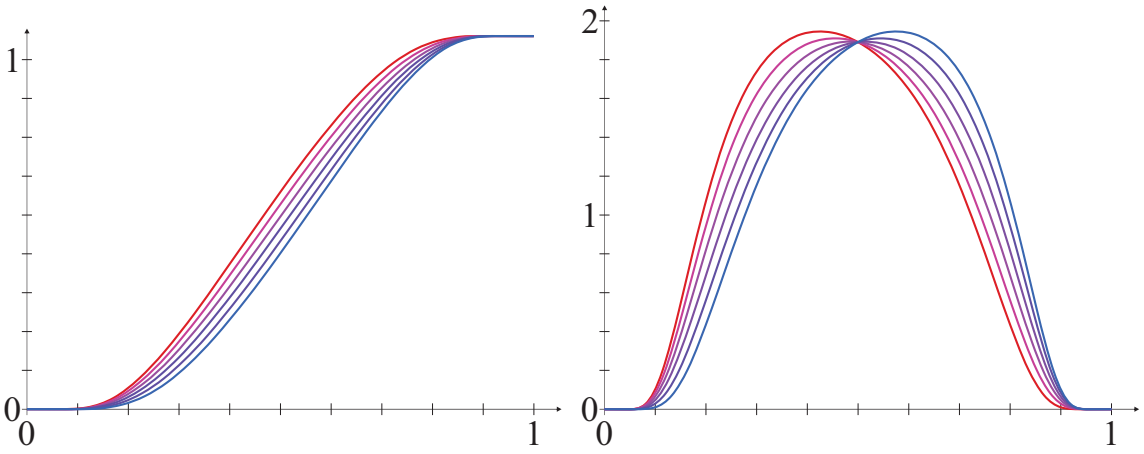
Effekten forventes å være lik effekten av Rμ-funksjonen. ERB-funksjonene (7.39), (7.40) og (7.40) med en balanseparameter $\mu = 0.5$ er punktsymmetriske som standarddefinisjon i (7.27),(7.28) og (7.30), mens alle andre verdier av μ gir asymmetriske funksjoner.

I figur 7.20 er $B_x(t;\mu)$ brukt som eksempel. Dette fordi resultatet for $B_x(t;\mu)$ er ganske likt $B_z(t;\mu)$ og $B_y(t;\mu)$. For $B_d(t;\mu)$ er oppførselen mer lik RB-funksjonen med balanseparameter (7.22), mer som en slags parallellforskyvning som vi kan observere i figur 7.12.

I seksjon 7.8 skal vi forøvrig undersøke balansesymmetri for B-funksjoner.

7.7.3 De asymmetriske stramningsparameterne α og β

De neste iboende parameterne er sannsynligvis mest av teoretisk interesse. Vi kaller dem de asymmetriske stramningsparameterne α og β. De er formmessig analoge med hermiteordenen a og b i Beta- og RB-funksjonen, se avsnitt 7.3, teorem 7.3 og seksjon 7.3 refsek-RB-funksjon, teorem 7.4. Husk at, i motsetning til Beta- og RB-funksjoner, påvirker ikke disse parameterne ordenen til ERB-funksjonen, fordi ordenen er ∞. Formen på funksjonen er imidlertid selvsagt påvirket.



Figur 7.20: *Til venstre ser vi $B_x(t;\mu)$ for $\mu = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. $B_x(t;\mu = 0)$ er plottet i rødt og $B_x(t;\mu = 1)$ er plottet i blått. De fire andre er plottet i farger som gradvis går fra rødt til blått. På høyre side er de respektive 1.-deriverte plottet i samme farge som deres respektive funksjoner.*

Vi legger så disse asymmetriske stramningsparameterne til (7.39), (7.40) og (7.41). Inkludert alle iboende parametere får vi nå følgende definisjon,

$$\phi(t;\gamma,\mu,\alpha,\beta) = e^{-\gamma\frac{|t-\mu|^{\alpha+\beta}}{t^{\alpha}(1-t)^{\beta}}}, \qquad \gamma,\mu,\alpha,\beta \in \mathbb{R}, \quad \gamma,\alpha,\beta > 0 \quad \text{og} \quad 0 < \mu < 1, \tag{7.44}$$

$$\psi(t;\gamma,\mu,\alpha,\beta) = \mu e^{\frac{-2\gamma}{t^{\alpha}}e^{\frac{-\gamma}{(1-t)^{\beta}}}}, \qquad \gamma,\mu,\alpha,\beta \in \mathbb{R}, \quad \gamma,\alpha,\beta > 0 \quad \text{og} \quad 0 < \mu < 1, \tag{7.45}$$

$$\varphi(t;\gamma,\mu,\alpha,\beta) = e^{2\gamma\left(\frac{1-\mu}{t^{\alpha}} - \frac{\mu}{(1-t)^{\beta}}\right)}, \quad \gamma,\mu,\alpha,\beta \in \mathbb{R}, \quad \gamma,\alpha,\beta > 0 \quad \text{og} \quad 0 < \mu < 1. \tag{7.46}$$

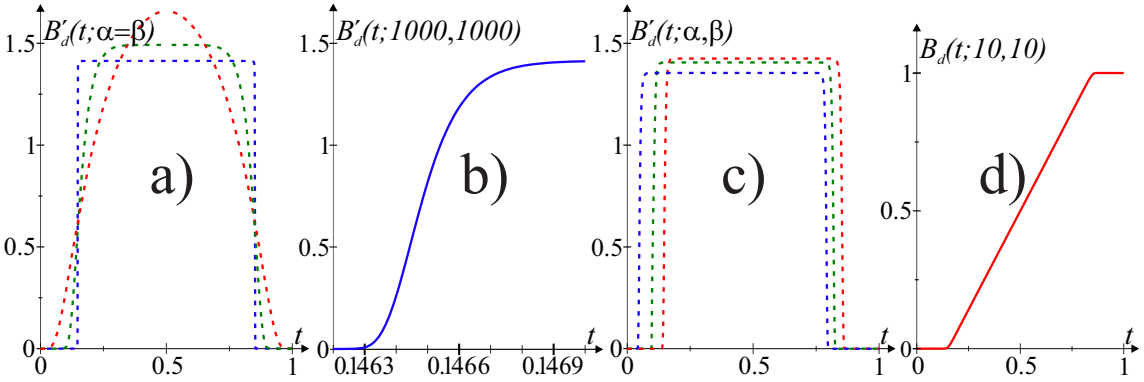
I [102, Setning 2.1, side 18–24] er det gitt et fullstendig bevis på at $B_d(t;\gamma,\mu,\alpha,\beta)$ er en B-funksjon . Det samme beviset vil også være gyldig for de andre ERB-funksjonene.

De forskjellige ERB-funksjonene vil oppføre seg forskjellig med disse iboende parameterne. For $B_d(t,\gamma,\mu,\alpha,\beta)$ med økende α og β, tenderer plottene mot en lineær funksjon på en del av domenet, se figur 7.21 - **d)**. Dette bekreftes av figur 7.21 - **a)**, der vi ser at den 1.-deriverte går mot noe som ligner på en døråpning. Imidlertid er ERB-funksjonen fortsatt C^{∞}-glatt, som kan sees i figur 7.21 -**b)**.

Hvis α og β er forskjellige, vil kurven forskyves parallelt mot siden med lavest tall og samtidig bli litt mindre bratt. Dette kan observeres i figur 7.21 - **c)**. Husk at integralet til $B_d'(t)$ alltid er 1, så i figuren, hvis “boksen" blir lavere, blir den også bredere.

I figur 7.22 - **a)** er $B_z(t;\alpha,\beta)$ plottet for $(\alpha+\beta = 8)$ hvor α og β er positive heltall. Funksjonsplottene er “ nesten vertikale" og i henhold til forskjellen mellom α og β er de forskjøvet horisontalt i forhold til hverandre.

I $B_d(t;\alpha,\beta)$ og $B_z(t;\alpha,\beta)$ er α og β antisymmetriske, der α er assosiert med $t = 0$ og β



Figur 7.21: *I plottet* **a)** *er* $B'_d(t;\alpha=\beta=1)$ *i striplet rødt,* $B'_d(t;\alpha=\beta=3)$ *i striplet grønt og* $B'_d(t;\alpha=\beta=1000)$ *i striplet blått. I plott* **b)** *er t-aksen til plott* **a)** *skalert kraftig og vi kan se en del av* $B'_d(t;\alpha=\beta=1000)$*. I plott* **c)** *er* $B'_d(t;\alpha=\beta=20)$ *i striplet rødt,* $B'_d(t;\alpha=15,\beta=25)$ *i striplet grønt og* $B'_d(t;\alpha=10,\beta=30)$ *i striplet blått. I plott* **d)** *er* $B_d(t;\alpha=\beta=10)$ *plottet i rødt.*

er knyttet til $t=1$. Dette i motsetning til $B_x(t;\alpha,\beta)$ og $B_y(t;\alpha,\beta)$ hvor α og β virker på to nivåer. I figur 7.22 - **b)** er et plott av $B_x(t;\alpha,\beta)$ der $\alpha=1$ og $\beta=\{1,2,3,4,5,1000\}$. Det ser ut som at $\lim_{\beta\to\infty} B_x(t,1,\beta)=\frac{1}{2}$. Et plott av $B_y(t;\alpha,\beta)$ med samme verdier for α og β gir et lignende resultat.

I figur 7.22 - **c)** er det et plott av $B_x(t;\alpha,\beta)$ der $\beta=1$ og $\alpha=\{1,2,3,4,5,1000\}$. Fra $\beta=1$ og oppover begynner funksjonen å bli brattere, men fra $\beta=2$ snur den og ser ut til å konvergere mot $\frac{1}{2}$, som i **b)** hvor β var varierer. I figur 7.22 - **d)** er et plott av $B_y(t;\alpha,\beta)$ der $\beta=1$ og $\alpha=\{1,2,3,4,5,10\}$. Den oppfører seg annerledes og ser ut til å konvergere mot en trinnfunksjon ved $t=\frac{1}{2}$.

7.7.4 ERB-funksjoner, derivasjon

Som et eksempel, la oss se på deriverte for $B_d(t)$. For de andre ERB-funksjonene vil det være tilsvarende prosedyrer.

La oss kalle første faktoren i (7.27) for $g(t)$, som betyr at $\phi'(t)=g(t)\ \phi(t)$, I (7.36) er 1.-deriverte gitt, $B'_d t=\mathbb{S}\ \phi(t)$. Høyere ordens deriverte kan da formuleres som,

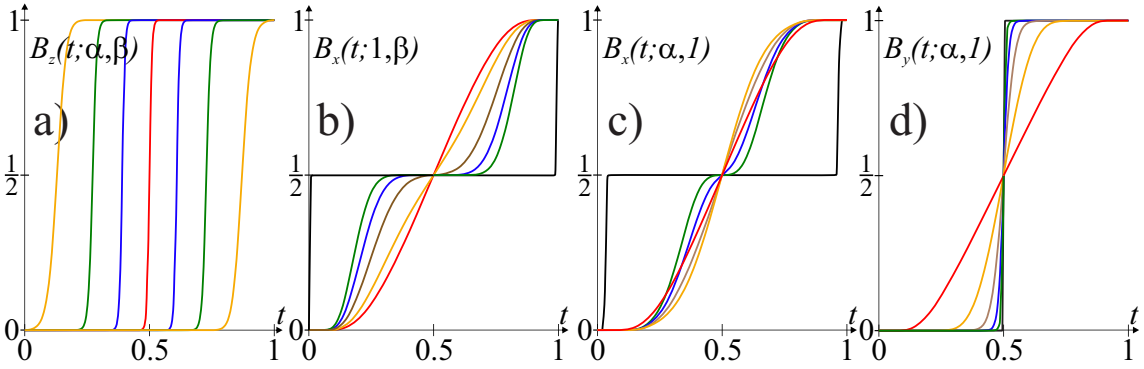
$$B_d^{(j)}(t)=f_j(t)\ B'_d(t), \tag{7.47}$$

hvor $f_1(t)=1$. $f_j(t)$ kan uttrykkes med $g(t)$. For standardversjonen av $B_d(t)$ er

$$g(t)=\frac{h(t)}{s(t)},\quad \text{hvor}\quad h(t)=1-2t\quad \text{og}\quad s(t)=(2t(1-t))^2. \tag{7.48}$$

Generelt ved derivering av et produkt med g og ϕ, får vi følgende uttrykk for $f_j(t)$,

$$\begin{aligned} f_2(t)&=g(t),\\ f_3(t)&=g'(t)+g(t)^2,\\ f_4(t)&=g''(t)+3g(t)\ g'(t)+g(t)^3. \end{aligned} \tag{7.49}$$



Figur 7.22: *I plottet* **a)** *ser vi fra venstre* $B_z(t;\alpha=1,\beta=7)$ *i oransje,* $B_z(t;2,6)$ *i grønt,* $B_z(t;3,5)$ *i blått,* $B_z(t;4,4)$ *i rødt,* $B_z(t;5,3)$ *i blått,* $B_z(t;6,2)$ *i grønt og* $B_z(t;7,1)$ *i oransje. I plottet* **b)** *ser vi* $B_x(t;\alpha=\beta=1)$ *i rødt,* $B_x(t;1,2)$ *i oransje,* $B_x(t;1,3)$ *i brun,* $B_x(t;1,4)$ *i blått,* $B_x(t;1,5)$ *i grønt og* $B_x(t;1,1000)$ *i svart. I plottet* **c)** *ser vi* $B_x(t;\alpha=\beta=1)$ *i rødt,* $B_x(t;2,1)$ *i oransje,* $B_x(t;3,1)$ *i brun,* $B_x(t;4,1)$ *i blått,* $B_x(t;5,1)$ *i grønt og* $B_x(t;1000,1)$ *i svart. I plott* **d)** *ser vi* $B_y(t;1,1)$ *i rødt,* $B_y(t;2,1)$ *i oransje,* $B_y(t;3,1)$ *i brunt,* $B_y(t;4,1)$ *i blått,* $B_y(t;5,1)$ *i grønt og* $B_y(t;10,1)$ *i svart.*

For standardversjonen av $B_d(t)$ får vi følgende uttrykk for $f_j(t)$ for j opptil 4,

$$\begin{aligned} f_2(t) &= \tfrac{h(t)}{s(t)}, \\ f_3(t) &= \tfrac{\frac{3}{2}h(t)^4-\frac{1}{2}}{s(t)^2}, \\ f_4(t) &= \tfrac{\left(3h(t)^6+\frac{3}{2}h(t)^4-5h(t)^2+\frac{3}{2}\right)h(t)}{s(t)^3}. \end{aligned} \tag{7.50}$$

der $h(t)$ og $s(t)$ er gitt i 7.48. En mye dypere analyse av de deriverte finnes på side 14-16 og 30-34 i [102], http://urn.nb.no/URN:NBN:no-15022.

7.8 Punkt-, orden- og balansesymmetri av B-funksjoner

Symmetriegenskapene er viktige for B-funksjonene. Den forteller oss hvordan de kan brukes, spesielt i en sekvens av påfølgende blendinger. i avsnitt 7.1 ble punktsymmetrien til en B-funksjon introdusert. Det er at B-funksjonen er symmetrisk om punktet $(0.5,\ 0.5)$.

Punktsymmetrien til en B-funksjon

En B-funksjon er symmetrisk, dvs. punktsymmetrisk, hvis

$$B(t)+B(1-t)=1, \quad t\in[0,1]\subset\mathbb{R}.$$

Dette er den viktigste typen symmetri og forteller oss at påvirkningen på resultatet til begge funksjoner i blendingen er lik. Vi har sett at for alle grupper og serier av B-funksjoner finnes det punktsymmetriske funksjoner.

Den neste typen symmetri er ordenssymmetri. Denne typen symmetri åpner for forskjellig orden, koblet til skjøter i en spline-funksjon basert på blending.

Ordenssymmetri til en B-funksjon.

En B-funksjon kalles ordenssymmetrisk hvis (for $a,b \geq 0$, $a,b \in \mathbb{Z}$)

$$B_{a,b}(t) + B_{b,a}(1-t) = 1, \quad t \in [0,1] \subset \mathbb{R}, \tag{7.51}$$

hvor a og b er (hermite)ordenene som skifter side i de to funksjonene.

Teorem 7.6. *Følgende B-funksjoner er ordenssymmetriske:*

- *Beta-funksjoner er ordenssymmetriske*
- *RB-funksjoner er ordenssymmetriske*
- *ERB-funksjonene $B_d(t;\alpha,\beta)$ og $B_z(t;\alpha,\beta)$ er "ordens"symmetrisk i den forstand at de er symmetrisk iht α og β.*

Beviset til teorem 7.6 finner vi i vedlegg C.6.

Den siste typen symmetri er balansesymmetri. Denne typen symmetri åpner for forskjellig balanse og dermed vekt, koblet til skjøter i en spline-funksjon basert på blending.

Balansesymmetri til en B-funksjon

En B-funksjon kalles balansesymmetrisk hvis

$$B(t;\mu) + B(1-t;1-\mu) = 1. \tag{7.52}$$

Teorem 7.7. *Følgende B-funksjoner er balansesymmetriske:*

- *Rμ-funksjoner er balansesymmetriske.*
- *ERB-funksjoner er balansesymmetriske.*

Beviset til teorem 7.7 finner vi i vedlegg C.7.

Som en slags konklusjon til seksjonen vil vi tilslutt vise at noen B-funksjoner kan være ordenssymmetriske og balansesymmetriske samtidig.

Samtidig orden og balansesymmetri av B-funksjoner

Teorem 7.8. *Følgende B-funksjoner er ordenssymmetriske og balansesymmetriske samtidig:*

Rμ-funksjonene oppfyller $B_{a,b}(t;\mu) + B_{b,a}(1-t;1-\mu) = 1,$

ERB-funksjonen B_d oppfyller $B_d(t;\mu,\alpha,\beta) + B_d(1-t;1-\mu,\beta,\alpha) = 1.$

ERB-funksjonen B_z oppfyller $B_z(t;\mu,\alpha,\beta) + B_z(1-t;1-\mu,\beta,\alpha) = 1.$

Beviset til teorem 7.8 finner vi i vedlegg C.8.

7.9 Implementering av B-funksjoner

For de fleste B-funksjoner og deres deriverte kan man enkelt lage algoritmer direkte fra formlene deres. En algoritme for å beregne B-funksjoner, inkludert deres deriverte kalles en *evaluator*. Det er enkelt å implementere effektive evaluatorer for Beta-funksjoner, RB-funksjoner og trigonometriske B-funksjoner, alle også med iboende parametere. Det er imidlertid mer komplisert å lage en generell evaluator for B-funksjoner, og en evaluator

for ERB-funksjoner av type 1 er ikke mulig å lage direkte, fordi den krever numeriske integrasjoner og approksimasjon.

I vedlegg A er en numerisk evaluator beskrevet. Evaluatoren ble opprinnelig laget for ERB-funksjoner av type 1, men er senere modifisert til andre B-funksjoner.[3] Evaluatoren er basert på approksimasjon av B-funksjonen ved stykkevis 3.-grads Hermitt-polynomer, hvor koeffisientene for alle Hermitt-funksjonene i alle samlingspunkt er forhåndsberegnet og lagret. Standard partisjonering er 1024 og denne partisjoneringen krever $1024 \times 6 \times 8 = 48k$-byte minne.

Avvikene er viktige og for ERB-funksjoner av type 1 er det største avvik: 10^{-13} for funksjonsverdien, 10^{-9} for 1.-deriverte og 10^{-6} for 2.-deriverte.

Kostnaden til evaluatoren er 6 multiplikasjoner for funksjonsverdien, 4 multiplikasjoner for 1.-deriverte, 5 multiplikasjoner for 2.-deriverte ...

C++ -kode for evaluatoren finnes på https://source.coderefinery.org/gmlib/gmlib1/gmlib/-/tree/master/modules/parametrics/evaluators.

[3] Evaluatoren ble introdusert i [102], laget for ekspo-rasjonale B-splines. Den ble laget for det skalerbare delsettet, som er praktisk talt identisk med ERB-funksjonen beskrevet her. Senere utvidet Gancheva og Delistoyanova evaluatoren i [71], til også å inkludere Beta-funksjoner.

Kapittel 8

Blendingsplines

Vi starter med å se tilbake på formelen for en vanlig B-splinekurve beskrevet i (6.12),

$$c(t) = \sum_{j=0}^{n-1} c_j \, b_{d,j}(t).$$

hvor $\{c_j\}_{j=0}^{n-1}$, er kontrollpunktene, dvs. punktene som definerer kontrollpolygonet til kurven. Settet med B-splines (basisfunksjonene) $\{b_{d,j}(t)\}_{j=0}^{n-1}$ er definert av skjøtvektoren $\bar{t} = \{t_0, t_2, ..., t_{n+d}\}$ og polynomgraden d. Ordenen til en B-spline er $k = d+1$ og er dimensjonen til funksjonsrommet.

Fordi $b_{d,j}(t)$ har lokal støtte, dvs. er forskjellig fra null bare på intervallet $t \in [t_j,\ t_{j+k})$, kan vi omformulere formelen for en B-Splinekurve til følgende,

$$c(t) = \sum_{j=i-k}^{i} c_j \, b_{d,j}(t),$$

hvor indeksen i følgelig er bestemt av $t_i \leq t < t_{i+1}$. Dette er da en spesifikk formel for skjøtintervallet $[t_i, t_{i+1})$.

I seksjon 6.2.3 ble matriseformulering av B-splines introdusert. Ved å bruke den vil en 3.-grads B-splinekurve ha følgende formel for skjøtintervallet $[t_i, t_{i+1})$,

$$c(t) = \begin{pmatrix} 1-w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1-w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1-w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \begin{pmatrix} 1-w_{3,i-2}(t) & w_{3,i-2}(t) & 0 & 0 \\ 0 & 1-w_{3,i-1}(t) & w_{3,i-1}(t) & 0 \\ 0 & 0 & 1-w_{3,i}(t) & w_{3,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}, \tag{8.1}$$

hvor

$$w_{d,i}(t) = \frac{t - t_i}{t_{i+d} - t_i}, \tag{8.2}$$

og hvor da indeksen i bestemmes av $t_i \leq t < t_{i+1}$.

Matriseformuleringen viser at formelen for B-splinekurver er basert på hjørnekutting, hver rad i matrisene er en lineærinterpolasjon mellom nabopunkt. Noen viktige egenskaper til B-spline kurver kommer nettopp fra hjørnekuttingen vist av matriseformen i (8.1). Som beskrevet i seksjon 6.2.2 er egenskapene som er knyttet til hjørnekutting:

- Den sterke konvekse omhyldningsegenskapen.
- Den variasjonsminkende egenskapen.

I tillegg kan følgende egenskaper lett sees fra matriseformelen;

- Den lokale modifikasjonen.
- Den affine invariansen.

Lokal modifikasjon følger av at formelen kun inneholder k punkt (4 for 3.-grad - jmf. (8.1)), og den affine invariansen følger av at hver rad summerer opp til 1 i alle matrisene.

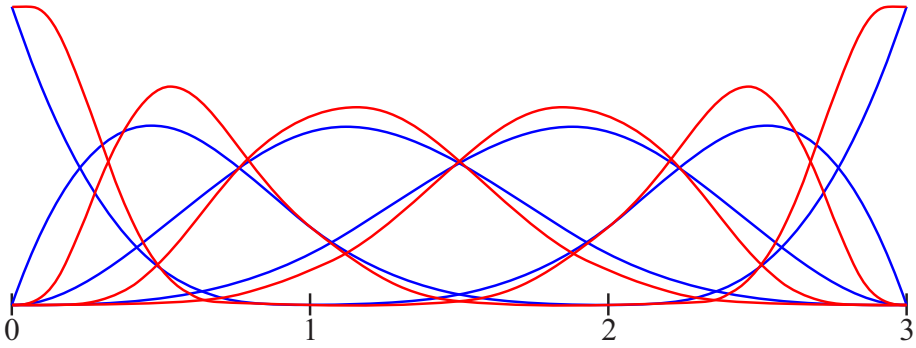
Kontinuitetsegenskapen kan vi imidlertid avledes direkte fra den lineære translerings- og skaleringsfunksjonen $w_{d,i}(t) : [t_i, t_{i+d}] \rightarrow [0,1]$ beskrevet i (8.2). Husk hermiteordenen til B-funksjonene i definisjon 7.2. Den lineære B-funksjonen $B(t) = t$ har hermiteorden 0, videre har t^2 venstre side hermiteorden 1, t^3 har venstre side hermiteorden 2 osv. Det følger dermed at hermiteordenen til B-splines er $d-1$ i endene av basisene fordi den starter med $\tilde{t}^d$, hvor $\tilde{t}$ er en translert og skalert t, og slutter med $(1-\tilde{t}))^d$. Ved enhver intern enkelskjøt slutter og starter kun èn lineær funksjon, som da reduserer kontinuiteten fra d til $d-1$. Dette forklarer B-spline-kontinuiteten ved en enkel skjøtverdi. Det følger at kontinuiteten er polynomgraden minus en. Med de samme argumentene følger det at kontinuiteten ytterlige reduseres med 1 hvis to skjøtverdier er like, og at kontinuiteten generelt er graden minus multiplisiteten av skjøtene.

8.1 B-splines med B-funksjoner

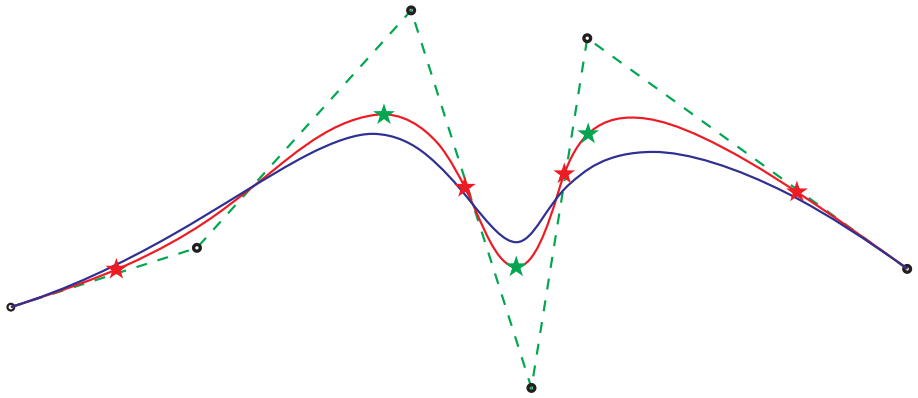
Vi tar for oss matrisene i (8.1) på nytt. De fleste elementene i matrisene er null. I hver rad er det bare to elementer som ikke er null, disse er $w_{d,j}(t)$ og $1 - w_{d,j}(t)$. Siden $w_{d,j}(t)$ er en lineær funksjon som fører t fra $[t_j, t_{j+d}]$ over til $[0,1]$, er hver rad egentlig en lineærinterpolasjon mellom to punkt. Merk at det fortsatt er en lineærinterpolasjon om vi legger til en høyere ordens symmetrisk B-funksjon (definisjon 7.1) til den lineær funksjon i B-spline-matrisene. Vi får da følgende utvidede uttrykk for en B-spline kurve:

$$
\begin{aligned}
c(t) = &\begin{pmatrix} 1 - B \circ w_{1,i}(t) & B \circ w_{1,i}(t) \end{pmatrix}
\begin{pmatrix} 1 - B \circ w_{2,i-1}(t) & B \circ w_{2,i-1}(t) & 0 \\ 0 & 1 - B \circ w_{2,i}(t) & B \circ w_{2,i}(t) \end{pmatrix} \\
&\begin{pmatrix} 1 - B \circ w_{3,i-2}(t) & B \circ w_{3,i-2}(t) & 0 & 0 \\ 0 & 1 - B \circ w_{3,i-1}(t) & B \circ w_{3,i-1}(t) & 0 \\ 0 & 0 & 1 - B \circ w_{3,i}(t) & B \circ w_{3,i}(t) \end{pmatrix}
\begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix},
\end{aligned}
\tag{8.3}
$$

hvor $B \circ w(t) = B(w(t))$ og hvor skjøtintervallet og dermed indeksen i bestemmes av $t_i \leq t < t_{i+1}$.



Figur 8.1: *De blå basisfunksjonene er polynomiske 3.-grads B-splines på skjøtvektoren* $\vec{t} = \{0,0,0,0,1,2,3,3,3,3\}$. *De røde basisfunksjonene er B-splines inkludert en 2.-ordens B-funksjon på den samme skjøtvektoren.*



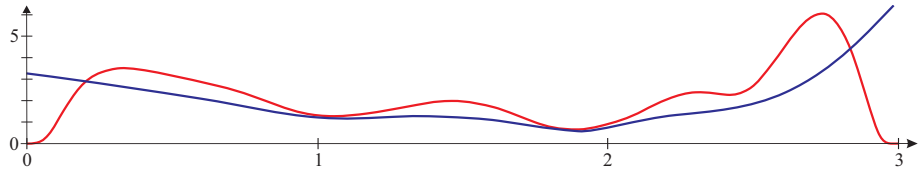
Figur 8.2: *Et plott av en 3.-grads polynomisk B-splinekurve i blått. De stiplede grønne linjene er kontrollpolygonet. Den røde kurven er en B-spline med B-funksjoner på den samme skjøtvektoren og kontrollpunkt som den polynomiske B-splinekurven. De røde og grønne stjernene markerer ekstremverdier for hastigheten til den røde kurven.*

Utvidelsen endrer ikke de viktigste egenskapene til B-splinekurven. Det er fortsatt hjørnekuttingsalgoritmen som induserer egenskapene; konvekst omhyldning, lokalt modifikasjonsskjema, variasjonsforminskning ..., og lineærinterpolasjonen den affine invariansen.

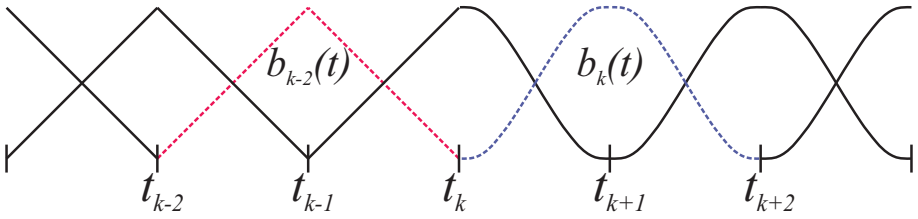
Hovedforskjellen sammenlignet med polynomiske B-splines er kontinuitetsegenskapene. Kontinuiteten ved skjøtene vil øke med ordenen til B-funksjonen. Hvis B-funksjonen har orden ∞, vil kurven bli C^∞-glatt. dette ble først beskrevet i [104].

I figur 8.1 er to sett B-splines plottet, polynomiske B-splines i blått, og B-splines med en B-funksjon i rødt. Hovedforskjellen vi kan observere i figur 8.1 er at de deriverte av alle basisfunksjoner med en B-funksjon er null i starten og i slutten av supportområdet. Dette kan tydeligst sees for de to første basisfunksjonene, ved $t = 0$, og for de to siste basisfunksjonene, ved $t = 3$. På de indre skjøtene, ved $t = 1$ og $t = 2$, kan vi observere at basisfunksjonene er nærmere null over et lengere område.

I figur 8.2 er en 3.-grads polynomisk B-splinekurve plottet i blått. Skjøtvektoren (splinerom) er den samme som i figur 8.1. De 6 kontrollpunktene er markert som kuler. Den



Figur 8.3: *Et plott av hastighetene til kurvene i figur 8.2. Den blå er hastigheten til den polynomiske B-splinekurven, og den røde er hastigheten til kurven med B-funksjoner.*



Figur 8.4: *Et plott av B-splines (basisfunksjoner). Til venstre ser vi polynomiske 1.-grads B-splines $b_{k-2}(t) = w_{1,k-2}(t)$ og til høyre B-splines med B-funksjoner, $b_k(t) = B \circ w_{1,k}(t)$.*

røde kurven er B-splines med B-funksjoner basert på samme skjøtevektor og kontrollpunkt, men hvor basisfunksjonene plottet i rødt i figur 8.1 er brukt.

Eksempelet i figur 8.2 viser, sammen med plottet i figur 8.3 av de respektive hastighetene, én spesiell egenskap til en B-splinekurve med B-funksjoner. De røde stjernene i figur 8.2 markerer punktene der hastigheten har et lokalt maksimum, og de grønne stjernene markerer punktene der hastigheten har lokalt minimum. Vi kan tydelig se at stjernene korrelerer med krumningen, der krumningen er minst er hastigheten størst og der krumningen er størst er hastigheten minst. Dette kan minne om å kjøre bil over en gitt distanse, man starter med lle deriverte lik null (hastighet, akselerasjon), og man regulerer farten med hensyn til kurvene og avslutter med å stoppe, hvor da alle deriverte igjen er null.

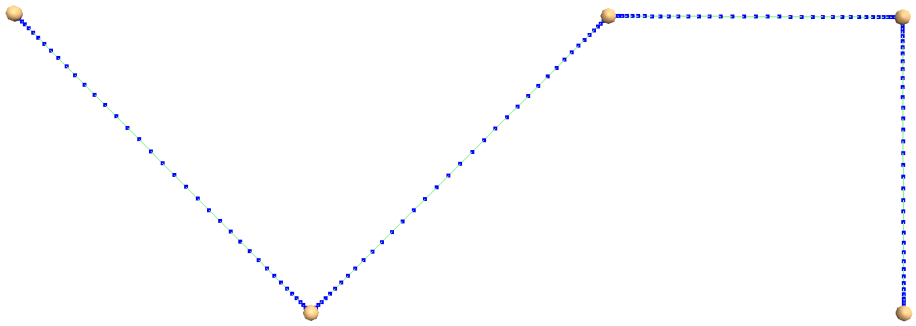
8.1.1 2.-ordens B-splines med B-funksjoner

Formelen for en 2.-ordens B-splinekurve er ganske enkel. Hvis vi inkluderer B-funksjoner er formelen over et skjøtintervall $[t_i, t_{i+1})$

$$c(t) = \begin{pmatrix} 1 - B \circ w_{1,i}(t) & B \circ w_{1,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-1} \\ c_i \end{pmatrix}, \tag{8.4}$$

som åpenbart er en kurve av rette linjer mellom kontrollpunktene på grunn av lineærinterpolasjonen i formelen. Men hvis B-funksjonen er en ERB- eller Fabius-funksjon, er kurven faktisk C^∞-glatt (funksjonen og alle deriverte er kontinuerlig), noe som er ganske spesielt siden kurven er stykkevis lineær. Forklaringen på dette er at alle deriverte er null i alle kontrollpunkt, noe vi skal se nærmere på senere.

I figur 8.4 ser vi et plott av basisfunksjoner, til venstre polynomiske 2.-ordens (1.-grader) B-splines, til høyre 2.-ordens B-splines med B-funksjoner. Formlene følger av (6.10) og (8.4) og er



Figur 8.5: *En 2.-ordens ekspo-rasjonal B-splinekurve er plottet med blå prikker. Punktene er plottet med samme avstand mellom parameterverdiene. Dermed vil tettheten av punkt illustrere hastigheten. Merk at kurven er C^∞-glatt, men bare G^0. Dette fordi (hermite)ordenen til B-funksjonen er ∞.*

$$b_{1,i}(t) = \begin{cases} B \circ w_{1,i}(t), & t_i < t \leq t_{i+1}, \\ 1 - B \circ w_{1,i+1}(t), & t_{i+1} < t < t_{i+2}, \\ 0, & \text{ellers.} \end{cases}$$

For en polynomisk B-splines er $B(t) = t$, ellers kan $B(t)$ være en hvilken som helst symmetrisk B-funksjon som beskrevet i kapittel 7.

I figur 8.5 er en 2.-ordens ekspo-rasjonal B-splinekurve plottet. Punktene er plottet med samme avstand mellom parameterverdiene. Dermed vil tettheten av punkt illustrere hastigheten. På figuren ser vi at punktene blir tettere og tettere jo nærmere kontrollpunktene vi kommer.

For å verifisere påstanden tidligere i seksjonen og observasjonen i figur 8.5 om at alle deriverte er null i kontrollpunktene, deriverer vi (8.4) og får

$$\begin{aligned} c'(t) &= \begin{pmatrix} -\delta_{d,i} B' \circ w_{1,i}(t) & \delta_{d,i} B' \circ w_{1,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-1} \\ c_i \end{pmatrix}, \\ &= \delta_{d,i} B' \circ w_{1,i}(t) \ (c_i - c_{i-1}) \end{aligned}$$

hvor $\delta_{d,i}$, definert i (6.13), er den deriverte av $w_{1,i}(t)$, definert i (6.11). $\delta_{d,i}(c_i - c_{i-1})$ er vektoren fra kontrollpunktet c_{i-1} til c_i, skalert med parameterintervallet. Det følger dermed at

$$c^{(j)}(t) = \delta_{d,i}^j B^{(j)} \circ w_{1,i}(t) \ (c_i - c_{i-1}), \quad \text{for} \quad j = 1, 2, \ldots. \tag{8.5}$$

Som vi ser i 8.5 er de deriverte avhengige av de deriverte til den valgte B-funksjonen. Hvis B-funksjonen er en ERB- eller Fabius-funksjon, vil alle deriverte være null i alle kontrollpunkt. Dette samsvarer også med teorem 7.1 selv om vi bare har punkt her.

Også her kan bilkjøringsanalogien være fordelaktig å bruke, vi starter med null hastighet, akselererer, ..., så øker vi hastigheten til vi er midt mellom to punkt, da begynner vi å bremse og stopper i punktet. Der svinger vi før vi kjører videre. På denne måten kjører vi i en rett linje mellom alle punktene på en matematisk C^∞-glatt måte, mens det geometrisk sett (formmessig) kun er G^0-glatt, dvs. geometrisk kun kontinuerlig og med skarpe knekker.

8.2 2.-ordens B-splines som blendingsplines

2.-ordens B-splines med B-funksjoner har noen helt spesielle egenskaper, som

- basisfunksjonene har minimal support, dvs. over to skjøtintervall,
- kurven interpolerer kontrollpunktene, dette følger av at ordenen er 2
- kurven er C^k-glatt der k er ordenen til B-funksjonen,

Sammen med at konstruksjonen er veldig enkel, er disse egenskapene en god grunn til å bruke 2.-ordens B-splines med B-funksjoner som grunnlag for blendingsplines.

Formelen til en 2.-ordens B-splinekurve, der kontrollpunktene er erstattet av kontrollkurver (også kalt lokale kurver), og hvor basisfunksjonene både genereres fra en skjøtvektor $\tau = \{t_i\}_{i=0}^{n+1}$ og er koblet med en B-funksjon, er

$$c(t) = \sum_{i=0}^{n-1} c_i(t)\, b_{1,i}(t), \quad \text{for} \quad t \in [t_1, t_n], \tag{8.6}$$

hvor $c_i(t),\ i = 0,1,...,n-1$ er lokale kurver, hver definert over skjøtintervallene (t_i, t_{i+2}), og hvor $b_{1,i}(t)$ er 2.-ordens B-splines med en symmetrisk B-funksjon i henhold til definisjon 7.1, vist i (8.4). Vi kaller denne konstruksjonen for blendingsplines. Du finner dem også under navnet ERBS eller GERBS i flere artikler[1]. På grunn av at ordenen er 2 og dermed medfører minimal support, kan formelen (8.6) på skjøtintervallet $t \in [t_i, t_{i+1}]$ forenkles til

$$\begin{aligned} c(t) &= \begin{pmatrix} 1 - B \circ w_{1,i}(t) & B \circ w_{1,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-1}(t) \\ c_i(t) \end{pmatrix} \\ &= c_{i-1}(t) + B \circ w_{1,i}(t)\, \Delta c_i(t), \end{aligned} \tag{8.7}$$

hvor $\Delta c_i(t) = c_i(t) - c_{i-1}(t)$. Dette er i samsvar med to-funksjons-blending beskrevet i seksjon 7.2 og er det samme som uttrykk (7.3).

Definisjon 8.1. *En 2.-ordens B-splines med B-funksjoner, og med kontrollkurver, også kalt lokale kurver kalles*

Blendingsplines

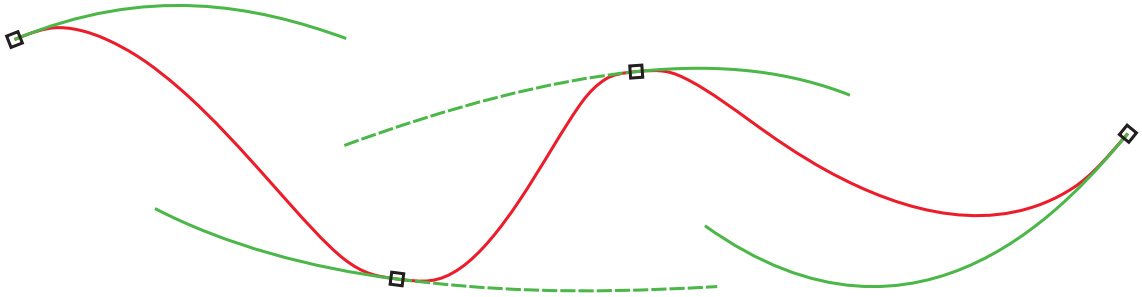
Det følger av (7.6) og (7.1) at for $j = 0,1,2,\ldots,S$ (S er ordenen til B-funksjonen) er

$$c^{(j)}(t_i) = c_{i-1}^{(j)}(t_i), \qquad \text{for} \quad i = 1,2,\ldots,n, \tag{8.8}$$

$$c^{(j)}(t) = c_{i-1}^{(j)}(t) + \sum_{k=0}^{j} \binom{j}{k} \delta_{1,i}^{k} B^{(k)} \circ w_{1,i}(t)\, \Delta c_i^{(j-k)}(t) \quad \text{når} \quad t_i < t < t_{i+1}, \tag{8.9}$$

hvor $\delta_{1,i}$ er definert i (6.13). (8.8) er i henhold til hermiteinterpolasjonsegenskapen beskrevet i teorem 7.1. Uttrykk (8.9) ser komplisert ut, men har samme struktur som bernsteinpolynomene der tallene fra binomialformelen følger Pascals trekant.

[1] ERBS og senere GERBS ble først presentert på "Mathematical Methods for Curves and Surfaces"-konferansen i Tromsø i 2004 og senere publisert flere steder, [105], [44], [45], [102], [47], [6],...



Figur 8.6: *En blendingsplinekurve plottet i rødt, og med fire lokale kurver plottet i grønt. Den resulterende kurven er en blending av dens lokale kurver, hvor de er blendet med en B-funksjonen. Den resulterende kurven interpolerer verdien og alle deriverte til hver av de tilstøtende lokale kurver i "midt"skjøten.*

2.-ordens B-splines med B-funksjoner

For å forenkle notasjonen av basisfunksjonene, inkludert deres deriverte, bruker vi

$$B_i^{(k)}(t) = \delta_{1,i}^k B^{(k)} \circ w_{1,i}(t), \quad \textit{for} \quad k = 0,1,\ldots \quad \textit{og} \quad t \in [t_i, t_{i+1}]. \tag{8.10}$$

Deretter kaller vi det andre leddet av (8.9) for: $g(t) = \sum_{k=0}^{j} \binom{j}{k} \delta_{1,i}^k B^{(k)} \circ w_{1,i}(t)\ \Delta c_i^{(jk)}(t)$. Funksjonsverdien og 1.-, 2.- og 3.-deriverte til $g(t)$, andre termen av (8.9), er

$$\begin{aligned}
g_i(t) &= B_i(t)\ \Delta c_i(t), \\
g_i'(t) &= B_i'(t)\ \Delta c_i(t) + B_i(t)\ \Delta c_i'(t), \\
g_i''(t) &= B_i''(t)\ \Delta c_i(t) + 2B_i'(t)\ \Delta c_i'(t) + B_i(t)\ \Delta c_i''(t), \\
g_i'''(t) &= B_i'''(t)\ \Delta c_i(t) + 3B_i''(t)\ \Delta c_i'(t) + 3B_i'(t)\ \Delta c_i''(t) + B_i(t)\ \Delta c_i'''(t).
\end{aligned} \tag{8.11}$$

Vi oppsummerer nå en av hovedegenskapene til blendingsplines.

Hermiteinterpolasjonsegenskapen til blendingsplines

En blendingsplines definert i definisjon 8.1 interpolerer hver lokale kurve c_i i skjøtverdien t_{i+1}, ikke bare verdien, men med alle deriverte opp til ordenen til B-funksjonen som brukes, (samsvarer med uttrykket 8.8). Dette kalles hermiteinterpolasjonsegenskapen til blendingsplines.

Er det noen restriksjoner på de lokale kurvene? Svaret er ja, men for all praktisk bruk er det nesten umulig å finne eksempler på kurver som ikke kan brukes som lokale kurver. Av teoretisk interesse er disse begrensningene beskrevet i [102], avsnitt 2.6. De vanligste typene av lokale kurver er bézierkurver, sub-kurver, sirkulære buer, blendingsplines selv, B-splines og kurver laget med Taylor-ekspansjon. Selvfølgelig avhenger kontinuiteten til den resulterende kurven av kontinuiteten til de lokale kurvene som brukes.

Figur 8.6 viser et eksempel på en blendingsplinekurve og dens lokale kurver. Vi ser 4 lokale kurver (grønn). Skjøtvektoren er $\tau = \{t_i\}_{i=0}^5$, der $t_0 = t_1$ og $t_4 = t_5$ er de multiple start- og ende-skjøtverdiene, og t_2 og t_3 er enkle interne skjøter. Hver basisfunksjon og dermed også lokal kurve spenner over to skjøtintervall, og hver lokal kurve interpolerer

den globale kurven ved dens midtre skjøt. Fordi de to første skjøtene er like og t_1 er den midterste skjøte vil den første lokale kurven, som spenner over $[t_0, t_2]$, interpolere starten til den globale kurven. Av samme grunn interpolerer den siste lokale kurven slutten av den globale kurven. Kurven i figur 8.6 er delt i tre deler. “Delingspunktene" er der de lokale kurvene berører den globale kurven, dvs. i skjøtene t_2 og t_3. Den første tredjedelen av den globale kurven er en blending av hele den første lokale kurven og første halvdel, til skjøt t_2, av den andre lokale kurven. Den andre tredjedelen er en blending av den andre delen, fra skjøt t_2, av den andre lokale kurve og den første halvdelen, til skjøt t_3, av tredje lokale kurve (begge deler stiplet grønt). Den tredje tredjedelen er en blanding av den andre halvdelen (fra skjøt t_3) av den tredje lokale kurven og hele den fjerde lokal kurve. Dette viser den ekstremt lokale supporten, index lokal support dvs. hvis vi endrer den første lokale kurven vil bare den første tredjedel av den globale kurven bli endret.

En generell algoritme for en “evaluator" for blendingsplines er som følger:

Algoritme 8. *(Forklaring av notasjonen, se avsnitt “Algoritmisk språk", side 6.) Algoritmen beregner $\{c^{(j)}(t)\}_{j=0}^{d}$ for en blendingsplinekurve. Algoritmen forutsetter at “evaluatorer" for de lokale kurvene og for B-funksjonen er tilgjengelig. Skjøtvektoren $\{t_i\}_{i=0}^{n+1}$ antas også å være tilgjengelig. Innput er: $t \in [t_1, t_n]$, og d - som er antallet deriverte som skal regnes ut. Retur er en* vector⟨Vector⟩*, der det første elementet inneholder $c(t)$, og deretter $c'(t), ..., c^{(d)}(t)$.*

vector⟨Vector⟩ **eval** (double t, int d)

```
int  i = i :\; t_i ≤ t < t_{i+1};          // Indeks for gjeldende skjøtintervall.
vector⟨Vector⟩  c_0 = {c_{i-1}^{(j)}(t)}_{j=0}^{d};   // "Evaluering" av lokal kurve c_{i-1}(t).
if (t == t_i)  return c_0;                 // Returner kun lokal kurve c_{i-1}(t), se (8.8).
vector⟨Vector⟩  c_1 = {c_i^{(j)}(t)}_{j=0}^{d};      // "Evaluering" av lokal kurve c_i(t).
vector⟨double⟩  a(d+1);                    // Vektor for å lagre "Pascals trekant nr".
vector⟨double⟩  B = {B_i^{(j)}(t)}_{j=0}^{d};        // Beregning av B-funksjon, se (8.10).
c_1 -= c_0;                                // c_1 blir nå Δc_i, se (8.9).
for ( int j=0; j ≤ d; j++ )
    a_j = 1;
    for ( int k=j-1; k > 0; k-- )
        a_k += a_{k-1};                    // Regne ut "Pascals trekant nr".
    for ( int k=0; k ≤ i; k++ )
        c_{0,j} += (a_k B_k) c_{1,j-k};    // "vektor += skalar*vektor", fra (8.9).
return c_0;
```

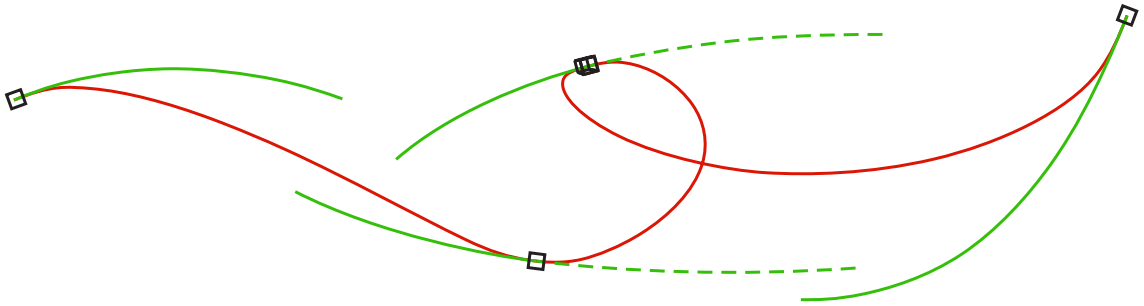
I algoritmen er “evalueringen" av de lokale kurvene markert med rødt.

8.2.1 Affine transformasjoner av lokale kurver

Affine avbildninger er beskrevet i (2.4) på side 16. Dette er typisk skalering, rotasjon, skjær og translasjon, og er generelt på den formen

$$Ap + v,$$

hvor p er et punkt og v er en assosiert vektor i et affint rom. Dette affine rommet kan typisk være $\mathbb{R}^3$, som da betyr at A er en 3×3 matrise



Figur 8.7: *Vi ser blendingsplinekurven fra figur 8.6, men hvor tre av de lokale (grønne) kurvene er flyttet og to er rotert. Kurven er i* $\mathbb{R}^3$.

Men hvis vi bruker homogene koordinater, se seksjon 2.6 og 3.2, vil alt samles i en 4×4 matrise (i $\mathbb{R}^3$) for alle affine avbildninger. Matrisen ser da slik ut;

$$A_i = \begin{bmatrix} x_{i,x} & y_{i,x} & z_{i,x} & p_{i,x} \\ x_{i,y} & y_{i,y} & z_{i,y} & p_{i,y} \\ x_{i,z} & y_{i,z} & z_{i,z} & p_{i,z} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{8.12}$$

De tre første kolonnene i matrisen er vektorer der den homogene koordinaten er 0, den siste kolonnen er et punkt der den homogene koordinaten er 1. Matrisen representerer posisjonen og orienteringen til en lokal kurve, og beskriver et lokalt koordinatsystem der punktet p_i er origo og de tre kolonnevektorene er koordinataksene x_i (rød), y_i (grønn) og z_i (blå). Hvis vi begrenser avbildningene til rotasjon, skalering og translasjon, beholder vi et ortogonalt koordinatsystem. I algoritme 8 ser vi to steder som er markert med rødt, det er "evaluatoren" til de lokale kurvene. Hvis vi ønsker å implementere affine transformasjoner av lokale kurver må vi legge til en homogen matrise til "evaluatoren". Hvis vi kaller de lokale kurvene i seg selv for $\alpha_i(t)$ og lar resultatene multipliseres med kurvenes homogene matriser får vi

$$c_i^{(j)}(t) = A_i \; \alpha_i^{(j)}(t), \quad j = 1, 2, ...,$$

der A_i er en homogen matrise lik (8.12) og som er assosiert med dens lokale kurve $\alpha_i(t)$. Merk at for en gitt t er $\alpha_i(t)$ og dermed $c_i(t)$ er punkt der den homogene koordinaten er 1, og for $j > 0$, er $\alpha_i^{(j)}(t)$ og dermed også $c_i^{(j)}(t)$ deriverte og vektorer der den homogene koordinaten er 0.

I figur 8.7 er kurven i figur 8.6 endret ved å flytte og rotere noen av de lokale kurvene. Den andre lokale kurven er bare flyttet, den tredje lokale kurven er flyttet og rotert om en vertikal akse og den fjerde kurven er flyttet oppover og rotert litt rundt en akse som kommer rett ut av arket.

Som vi ser i figur 8.7, er rotasjonene av de lokale kurvene gjordt rundt interpolasjonspunktene til kurvene $c_i(t_{i+1})$. En annen mulighet er at rotasjonsaksen kan kobles til tangenten eller cotangenten til kurven i interpolasjonspunktene. For at det skal kunne gjøres stilles det krav til hvordan de lokale koordinatsystemene bør se ut.

✓ Det er å foretrekke at det lokale origo til en lokal kurve er i punktet der blendingskurven interpolerer den lokale kurven. Måten å gjøre dette på er avhengig av typen lo-

kale kurver. Vi starter med å finne posisjonen til interpolasjonspunktet, $p_i = c_i(t_{i+1})$, dernest puttes p_i inn som siste kolonne i matrisen A_i. Vi korrigerer så dataene i den lokale kurven med $-p_i$ på en måte som er mulig i henhold til formelen.

✓ Det kan være nyttig å bruke **Frenet-ramme**, matrisen $F(t_{i+1})$, også kalt en TNB-rammen, se seksjon 4.1.2. Vi lager den som en kolonnematrise. Dvs. en matrise der den første kolonnen er $T = c_i'(t_{i+1})$ normalisert, den andre kolonnen er $N = c_i''(t_{i+1}) - \langle c_i''(t_{i+1}), T\rangle T$ normalisert, og den tredje kolonnen er $B = T \wedge N$. Siden matrisen er ortonormal er $F^{-1} = F^T$. Dermed korrigerer vi dataene/formlene i $c_i(t)$ med $F^{-1}(t_{i+1})$ og setter $F(t_{i+1})$ som øvre venstre submatrise i A_i.

8.2.2 Bézierkurver som lokale kurver

Bézierkurver er praktisk å bruk som lokale kurver. Bézierkurver er omtalt i seksjon 4.4, uttrykk (4.31), og formelen er:

$$\alpha(t) = \sum_{j=0}^{d} c_j \, b_{d,j}(t), \quad \text{for } \; t \in [0,1],$$

der d er polynomgraden $b_{d,j}(t)$, er basisfunksjonene, dvs. bernsteinpolynomer som er beskrevet i seksjon 4.4.1, $c_j \in \mathbb{R}^s$, $j = 0,1,...d$ er koeffisienter som danner kontrollpolygonet, og hvor s vanligvis er 2 eller 3 (dvs. i planet eller "rommet").

Fra Definisjon 8.1 ser vi at domenet til hver lokale kurve $c_i(t)$ er $[t_i, t_{i+2}]$. Dette betyr at hver bézierkurve $c_i(t)$ må reparametriseres fra $[0,1] \to [t_i, t_{i+2}]$. Dette kan gjøres med å bruke de lineære translerings- og skaleringsfunksjonene (6.11) med $d = 2$, dvs.

$$w_{2,i}(t) = \frac{t - t_i}{t_{i+2} - t_i} \quad \text{og} \quad \delta_{2,i} = w_{2,i}' = \frac{1}{t_{i+2} - t_i}. \tag{8.13}$$

Så for lokale bézierkurver får vi følgende "evaluator", som gir posisjon og d-deriverte og som dermed kan brukes i Algoritme 8 der det er markert med rødt.

$$c_i^{(j)}(t) = \delta_{2,i}^j \alpha_i^{(j)} \circ w_{2,i}(t), \quad j = 0,1,\ldots,d \quad \text{og} \quad t \in [t_i, t_{i+2}]. \tag{8.14}$$

I seksjon 4.4.3 er bernstein/hermitematrisen $\mathbf{B}_d(t,\delta)$ beskrevet i (4.39). Dette er matrisen vi bruker for å beregne posisjonen og alle deriverte for en gitt t-verdi, dvs.

$$\begin{pmatrix} c_i(t) \\ c_i'(t) \\ \vdots \\ c_i^{(d)}(t) \end{pmatrix} = \begin{pmatrix} b_{0,d}(w_{2,i}(t)) & b_{1,d}(w_{2,i}(t)) & \cdots & b_{d,d}(w_{2,i}(t)) \\ \delta_{2,i}\, Db_{0,d}(w_{2,i}(t)) & \delta_{2,i}\, Db_{1,d}(w_{2,i}(t)) & \cdots & \delta_{2,i}\, Db_{d,d}(w_{2,i}(t)) \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{2,i}^d D^d b_{0,d}(w_{2,i}(t)) & \delta_{2,i}^d D^d b_{1,d}(w_{2,i}(t)) & \cdots & \delta_{2,i}^d D^d b_{d,d}(w_{2,i}(t)) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_d \end{pmatrix}$$

og i vektor/matrise-notation

$$\{c_i^{(j)}(t)\}_{j=0}^d = \mathbf{B}_d(w_{2,i}(t), \delta_{2,i})\ \mathbf{c}. \tag{8.15}$$

Når vi skal regne ut (8.15) og dermed (8.14) kan vi bruke Algoritmen 2, som lager matrisen $\mathbf{B}_d(w_{2,i}(t), \delta_{2,i})$, beskrevet i avsnitt 4.4.3. Matrisen kan så multipliseres med vektoren

av kontrollpunktene $\mathbf{c}$. Hvis vi ønsker å beregne færre enn d-deriverte, kan algoritme 2 modifiseres til å redusere antall rader i matrisen. I både figur 8.6 og figur 8.7 er 2.-grads bézierkurver brukt som lokale kurver for blendingsplines.

For at alle lokale bézierkurver skal ha sitt lokale koordinatsystem, hvor det lokale origo er der blendingskurven interpolerer den lokale kurven, må vi korrigere kontrollpunktene. Vi gjør dette ved å finne det lokale origo, som er $p_i = \alpha_i \circ w_{2,i}(t_{i+1})$, jamfør (8.8), og deretter korrigere alle kontrollpunktene til $c_{i,j} = c_{i,j} - p_i$, for $j = 0,1,\ldots,d$. Samtidig må siste kolonne i den homogene matrisen A_i settes til p_i.

I figur 8.7 er noen lokale bézierkurver rotert rundt sitt lokale origo.

8.2.3 En blendingspline-approksimasjon av en kurve

Gitt en kurve $\varphi(t)$, med et domene $[a,b]$ om kurve er åpen, eller $[a,b)$ om kurve er lukket/syklisk. For å lage en approksimativ "kopi" av den gitte kurven trenger vi

- antallet n og polynomgraden d til de lokale bézierkurvene som skal lages,
- å lage en skjøtvektor, $\{t_i\}_{i=0}^{n+1}$ som spenner over domenet til $\varphi(t)$,
- å lage kontrollpunktene $\{\mathbf{c}_{ij}\}_{j=0}^{d}$ til bézierkurvene $c_i(t),\ i = 0,1,\ldots,n-1$.

Den enkleste måten å lage en skjøtvektor på er å lage en uniform skjøtvektor der $[t_1,t_n]$ er domenet til $\varphi(t)$. De to endeskjøtverdiene settes så i samsvar med at $\varphi(t)$ er åpen eller lukket (se avsnitt 6.2.2). Vi tar så utgangspunkt i (8.15), og da bruke like mange deriverte som polynomgraden vi har valgt, dermed er matrisen $\mathbf{B}_d(w_{2,i}(t),\delta_{2,i})$ en $d+1 \times d+1$ matrise (laget av Algoritmen 2). Fra den opprinnelige kurven $\varphi(t)$ trenger vi derfor posisjonen og d deriverte i hver skjøtverdi $t_1,t_2,\ldots,t_n$. Vi snur så uttrykk (8.15) og får følgende prosedyre for å lage en blendingsplinekopi av en kurve.

Kopiering med hermiteinterpolasjon

Denne metoden for å "kopiere" kurver er egentlig en **hermiteinterpolasjon**. Vi får

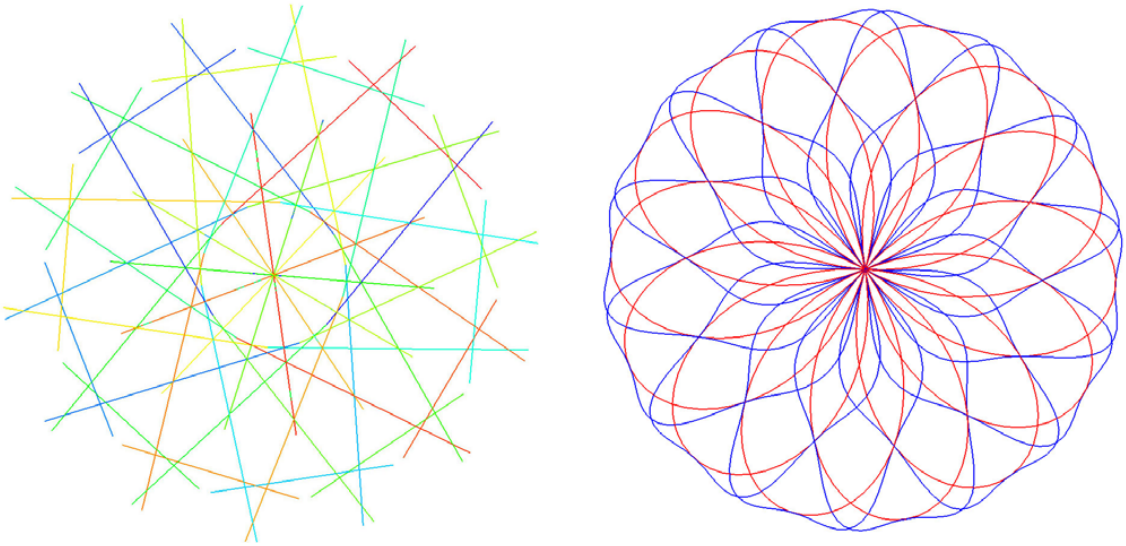
$$\mathbf{c}_{i-1} = \mathbf{B}_d(w_{2,i}(t),\delta_{2,i})^{-1}\ \overline{\varphi}(t_i), \quad \text{for} \quad i = 1,2,\ldots,n,$$

der $\mathbf{c}_{i-1}$ er kontrollpunktene til den lokale bézierkurven c_{i-1}, og $\overline{\varphi}(t_i)$ er posisjonen og d deriverte til den opprinnelige kurven i parameterverdien t_i.
Dermed, i de interne skjøtene, vil posisjonen og d-deriverte til blandingsplinekurven $c(t)$ være lik posisjonen og d-derivertene til den opprinnelige kurven $\varphi(t)$, dvs.

$$c^{(j)}(t_i) = \varphi^{(j)}(t_i), \quad j = 0,1,\ldots,d \quad \text{og} \quad i = 1,2,\ldots,n.$$

8.2.4 Eksempler

Vi skal se eksempler på hermiteinterpolasjon av tre forskjellige originalkurver. I eksemplene skal kurvene approksimeres med blendingsplines med lokale bézierkurver av forskjellige grader. Hensikten er å vise effekten av å bruke lokale bézierkurver. De fleste



Figur 8.8: *Til høyre, en blendingsplinekurve (blå) interpolerer en "rosekurve" (rød) i 56 interpolasjonspunkt. Posisjonen og 1.-deriverte er brukt i hvert interpolasjonspunkt. Denne approksimasjonen ser ut til å gjøre kurven for lang, slik at kurven bukler seg mellom interpolasjonspunktene. Til venstre er de 56 lokale bézierkurvene (linjene) til blendingsplinekurven plottet i forskjellige farger. Fordi "Rosekurven" har 14 kronblader og 14*4=56 interpolasjonspunkt, er de lokale kurvene (linjene) "symmetriske". rundt "rosesenteret". I tillegg skjærer par av påfølgende linjer hverandre.*

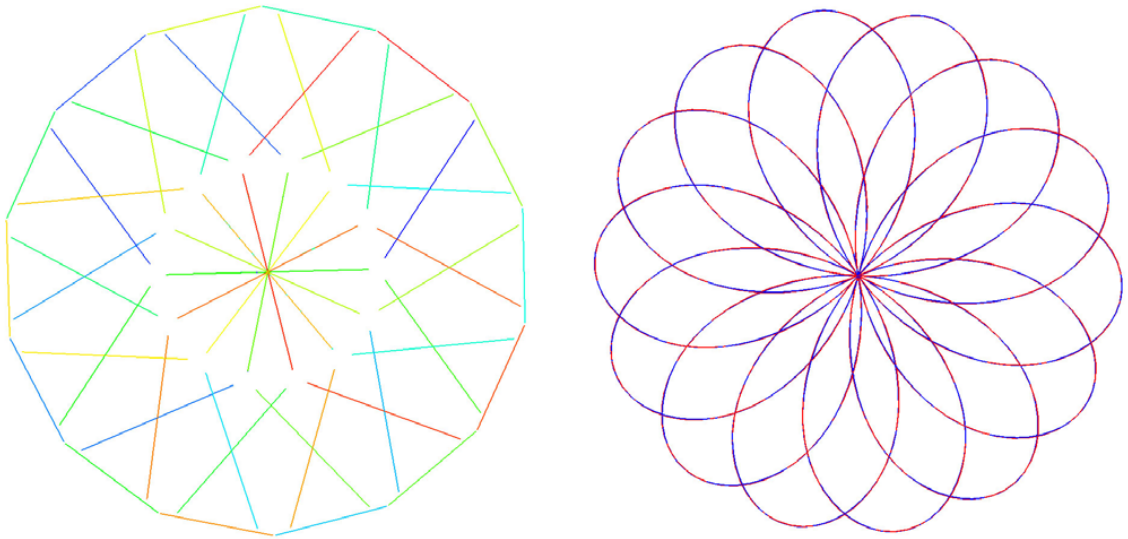
eksemplene er lukkede/sykliske kurver, men det er også ett eksempel på en åpen kurve. Eksemplene viser at hermiteinterpolasjon ikke nødvendigvis er en "optimal" approksimasjon og at det er mulig å forbedre løsningen ved å skalere de lokale kurvene, eller det lokale domenet i innputt til interpolasjonsprosessen.

Det første eksemplet er en såkalt "rosekurve" beskrevet i [163], og definert av formelen,

$$g(t) = \begin{pmatrix} \cos t \ \cos\left(\frac{7}{4}t\right) \\ \sin t \ \cos\left(\frac{7}{4}t\right) \\ 0 \end{pmatrix} \quad \text{for} \ \ t \in [0, 8\pi). \tag{8.16}$$

Den vil se ut som en rose med 14 kronblad. Antallet kronblad er 2 ganger telleren i brøken i cosinus i formelen i (8.16). Hastigheten er oscillerende mellom 1 og 1,75, tregest i midten av rosen og raskest i spissen av hvert kronblad.

I figur 8.8 er "rosekurven" approksimert av en blendingsplines med 4 interpolasjonspunkt på hvert kronblad, dvs. totalt 4*14=56 interpolasjonspunkt jevnt fordelt på parameterdomenet. I hvert interpolasjonspunkt er posisjon og 1.-deriverte brukt, og vi kan se effekten av å bare bruke 1.-deriverte. Krumningen blir dermed null i alle interpolasjonspunkt. Resultatet er at kurven blir for lang. Dette kan tydelig sees til høyre i figur 8.8. Til venstre i figuren vises de lokale kurvene. Fordi alle lokale kurver er av grad 1, er de rette linjer. Lengden på linjene, og det faktum at de skjærer hverandre, indikerer også at den resulterende kurven blir for lang og dermed vil bukle seg.



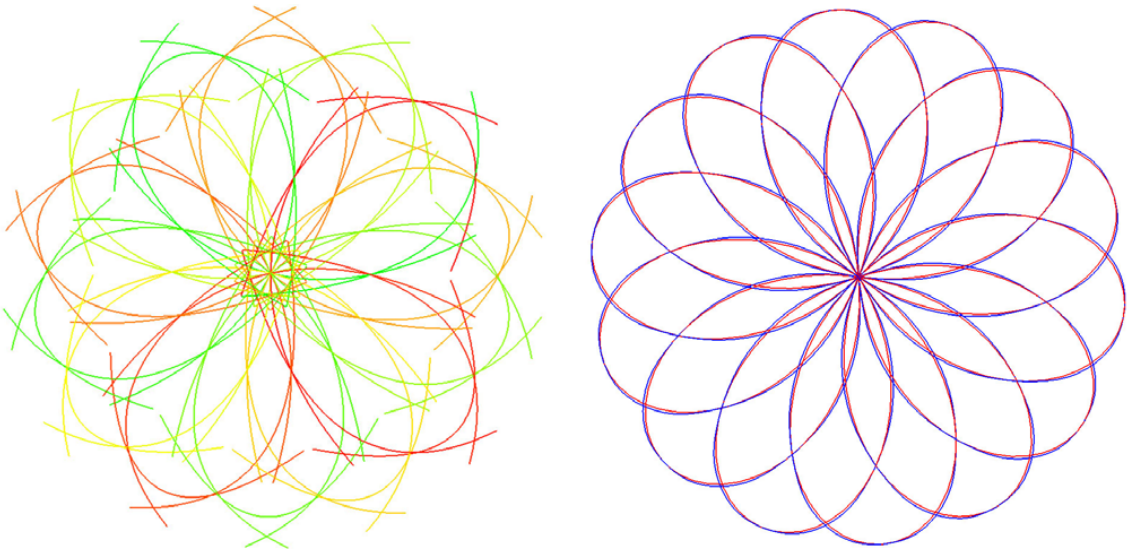
Figur 8.9: *Til høyre, en blendingsplinekurve (blå) interpolerer en "rosekurve" (rød) i 56 interpolasjonspunkt. Også her er posisjon og 1.-deriverte brukt i hvert punkt. Men nå er de lokale kurvene skalert med 0,5 etter interpolasjonen. Resultatet er geometrisk veldig nær den opprinnelige kurven. Til venstre ser vi de 56 lokale bézierkurvene (linjene) til blendingsplinekurve, alle er plottet i forskjellige farger.*

Kurven i figur 8.9 er laget med samme interpolasjon som i det forrige eksempelet. Imidlertid er de lokale kurvene nå skalert med 0,5 etter at de er laget. Dette ser vi til venstre i figur 8.9. Ved skalering er det viktig at interpolasjonspunktet er origo til det lokale koordinatsystemet for hver lokale kurve, slik at interpolasjonspunktene ikke flytter seg. Den resulterende kurven er geometrisk veldig lik den opprinnelige "rosekurven", men hastigheten vil svinge sterkere. Det er mulig å få samme resultat ved å skalere de (innputt) deriverte i stedet for å skalere den resulterende kurven (linjen). I dette eksempelet er resultatet geometrisk svært bra. Et enda større potensial for å forbedre resultatet er imidlertid dersom skaleringsfaktoren er forskjellig for de forskjellige lokal kurvene.

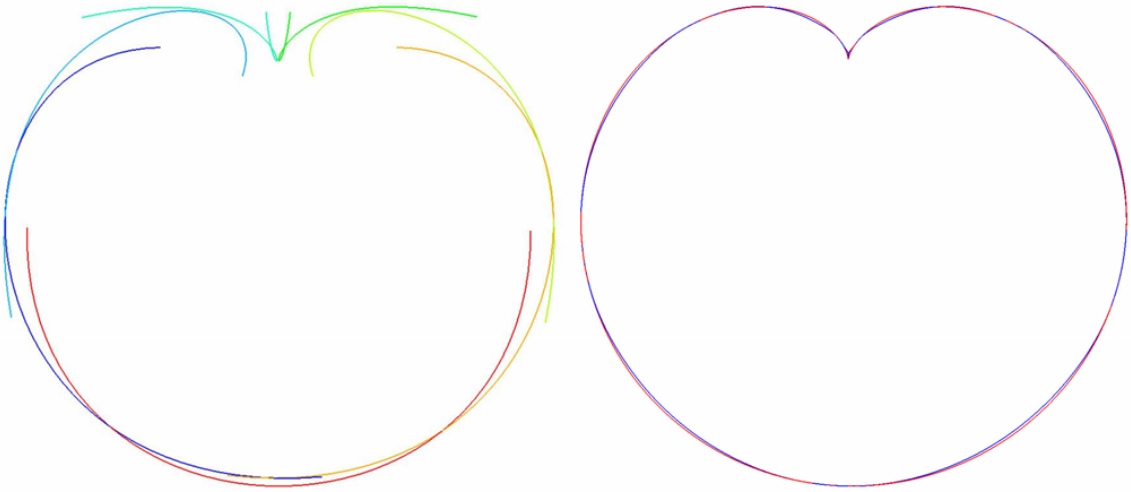
I figur 8.10 er "rosekurven" approksimert som i de to foregående eksempler, med 56 interpolasjonspunkt men nå med posisjon, 1.- og 2.-deriverte i hvert interpoleringspunkt. Resultatet er ganske bra, men ikke like godt geometrisk som i forrige eksempel. Hastigheten derimot er svært lik den opprinnelige kurven, og er dermed mye bedre enn i forrige eksempel. Til venstre i figuren er de lokale kurvene plottet. De er alle av grad 2, og "symmetriske" i betydningen at på hvert kronblad er det et sett med lokale kurver som har samme form på hvert kronblad. Man kan også forbedre resultatet ved å bruke en lignende metode som i forrige eksempel, dvs. individuell skalering.

Det neste kurveeksemplet er en såkalt "kardioidekurve" beskrevet i [80], og definert av formelen,

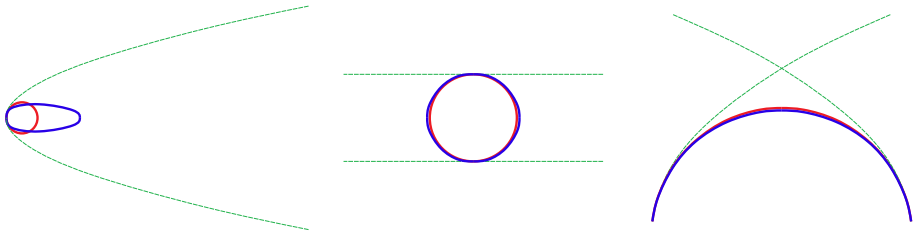
$$g(t) = \begin{pmatrix} 2\cos t\,(1+\cos t) \\ 2\sin t\,(1+\cos t) \\ 0 \end{pmatrix}, \quad \text{for} \quad t \in [0, 2\pi). \tag{8.17}$$



Figur 8.10: *Til høyre to kurver som delvis dekker hverandre. Den røde kurven er en original "rose-curve", den blå kurven er en blendingsplinekurve. Approksimasjonen er laget av 56 interpolasjonspunkt, der posisjon, 1.- og 2.-deriverte i hvert interpoleringspunkt er brukt. Til venstre ser vi de 56 lokale bézierkurvene til blendingsplinekurven plottet gradvis fra grønt til rødt. De er alle 2.-grads bézierkurver.*



Figur 8.11: *Til høyre ser vi to kurver som delvis dekker hverandre. Den røde kurven er en "kardioidekurve". Den blå kurven en blendingsplinekurve. Approksimasjonen er laget med 7 interpolasjonspunkt, der posisjonen, 1.-, 2.- og 3.-deriverte i hvert interpoleringspunkt er brukt. Til venstre er de syv lokale bézierkurvene plottet i forskjellige farger. Alle de syv kurvene er 3.-grads bézierkurver med 4 kontrollpunkt.*



Figur 8.12: *Tre eksempler på approksimasjon av sirkler/sirkelbuer med blendingsplines som har lokale bézierkurver vises. De originale kurvene er røde, blandingsplines-kurvene er blå, og de lokale bézierkurvene er grønne. Til venstre er bare Ã©n 2.-grads bézierkurve brukt som lokal kurve for å approksimere en sirkel. I midten er to 1.-grads bézierkurver (linjer) brukt som lokale kurver for å approksimere en sirkel. På høyre side er en del av en sirkel approksimert av en blendingsplinekurve med to 2.-grads bézierkurver som lokale kurver.*

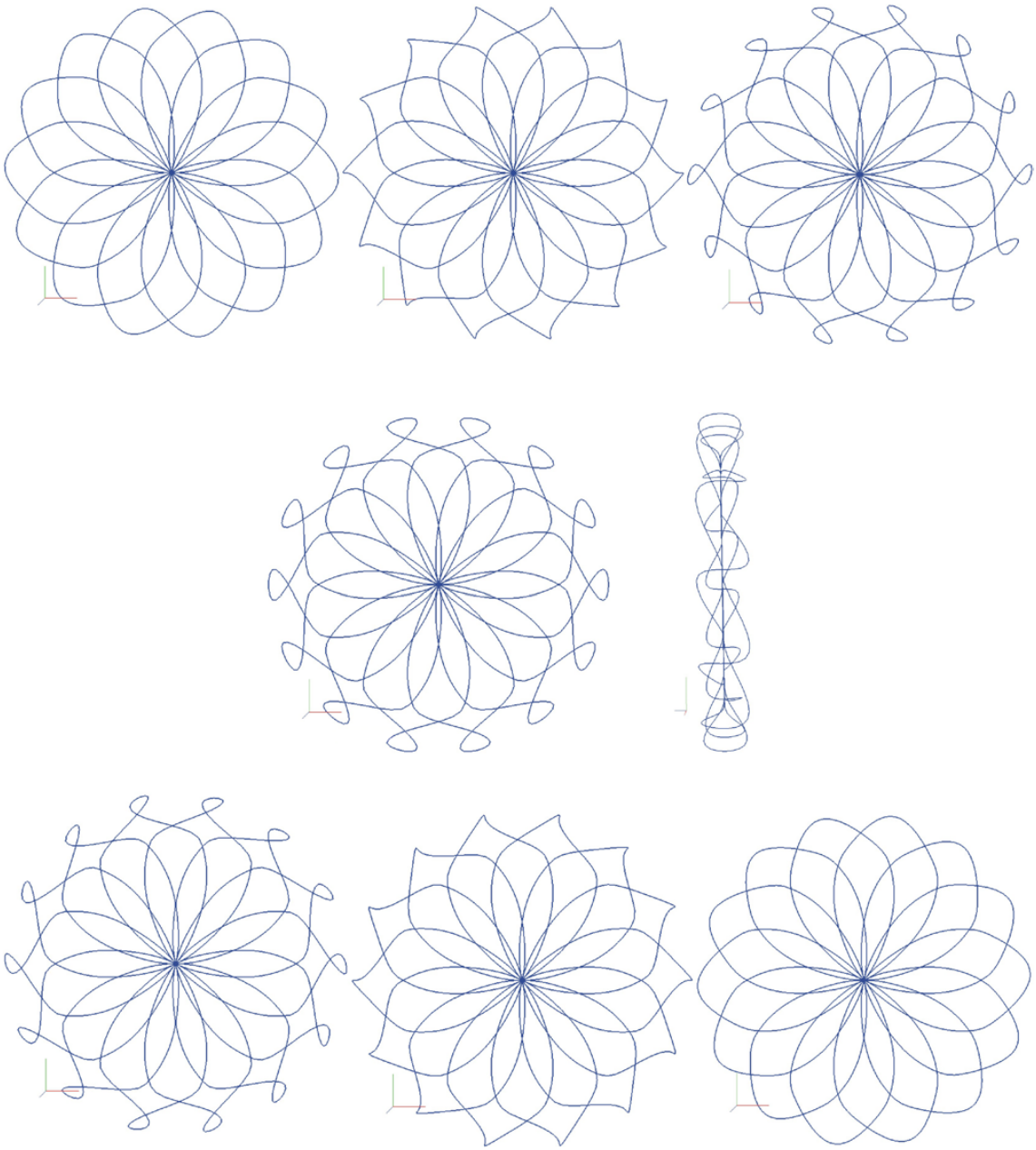
Et plott av (8.17) vil se ut som et eple, med en cusp øverst. Til høyre i figuren 8.11 er en “kardioidekurve" approksimert av en blendingsplines ved bruk av 7 interpolasjonspunkt jevnt fordelt over parameterintervallet. I hver av interpolasjonspunktene er posisjonen, 1.-, 2.- og 3.-deriverte brukt i hermiteinterpolasjonen. Eksempelet viser hvordan interpolasjonen takler singulariteter som i dette eksempelet er en cusp. Til venstre i figur 8.11 er alle syv lokale kurver plottet i forskjellige farger. De er alle 3.-grads bézierkurver og modellerer formen til den opprinnelige kurven ganske bra, men de ser ut til å være litt for lange som vi kan se i overlappingen med neste kurve (de overlapper også litt av kurve etter). Grunnen til dette er den samme som i forrige eksempel, dvs. 4.-deriverte er null.

De siste eksemplene er noen kuriositeter. Det er to approksimasjoner av sirkler og en av en sirkelbue. Til venstre i figur 8.12 er det en approksimasjon av en sirkel med kun ett interpoleringspunkt, posisjonen, 1.- og 2.-deriverte, “halen" blendes med “hode". Den lokale kurven er derfor bare en 2.-grads bézierkurve. I figuren er den opprinnelige sirkelen rød, blendingsplinekurven blå, og den lokale bézierkurven stiplet grønt.

I midten i figur 8.12 er en sirkel approksimert med to interpolasjonspunkt, posisjonen og 1.-deriverte i hvert av de to punktene. De lokale linjene skaleres så med 0,92 etter interpolasjonen. Resultatet er visuelt ganske bra.

Til høyre i figur 8.12 vises en sirkelbue som er approksimert av en blendingsplinekurve. To lokale kurver er laget i to interpolasjonspunkt med posisjon, 1.- og 2.-deriverte. I dette tilfellet vises en annen type korreksjon, domenet er skalert med 0,635. Resultatet er ganske godt. Man kan nesten bare se den blå blendingsplinekurven som dekker den røde sirkelbuen.

I figur 8.13 vises en animasjon ved å rotere noen av de lokale kurvene. På en “kopiert" “rosekurve" roteres alle lokale kurvene som er på spissen av hvert kronblad 6 ganger rundt en vektor fra rosekurvens senter til interpolasjonspunktet. Du finner mer om alle eksemplene vist her og andre eksempler, og om hvordan forbedre resultatet, i [102] som kan lastes ned fra http://urn.nb.no/URN:NBN:no-15022



Figur 8.13: *Figuren viser "rosekurve" approksimasjonen med blendingsplines med 56 interpolasjonspunkt og lokale 2.-grads bézierkurver. De lokale kurvene på tuppen av kronbladene roteres rundt deres lokale y-akse (som sammenfaller med retningen til radiusen til rosekurven). På figuren er 7 kurver med "roterte" lokale kurver plottet, hver kurve rotert* 30° *fra den forrige. Den fjerde kurven, hvor de lokale kurvene på spissen av kronbladene er rotert totalt* 90°*, er også plottet fra siden.*

8.3 En delkurvekonstruksjon

Om å utvide en hvilken som helst parametrisk kurve til en blendingsplinekurve

Vi kan utvide en hvilken som helst parametrisk kurve til en splinekurve ved å legge til en skjøtvektor. Resultatet er at vi får et sett med overlappende delkurver, som hver er knyttet til en av de interne skjøtverdiene og som har et domene som dekker to skjøtintervall, en på hver side av den aktuelle skjøtverdien. En delkurve er bare en begrensning av domenet til en kurve. Å bruke delkurver som lokale kurver betyr at en blendingsplineutvidelse av en kurve i utgangspunktet er identisk med selve kurven. Dette fordi blending av en kurve med seg selv gir selve kurven. Når man legger til affine transformasjoner (se seksjon 8.2.1), vil en delkurvealgoritme i utgangspunktet være lik algoritmen for lokale bézierkurver. Når vi endrer (8.7), får vi, for $i = 1,2,\ldots,n$

$$c(t) = \begin{pmatrix} 1-B_i(t) & B_i(t) \end{pmatrix} \begin{pmatrix} A_{i-1}\ \varphi(t) \\ A_i\ \varphi(t) \end{pmatrix}, \quad t \in [t_i, t_{i+1}),$$

hvor $B_i(t) = B \circ w_{1,i}(t)$ (se (8.10)), $\varphi(t)$ er orginalkurven og A_i er en homogen matrise som beskrevet i seksjon 8.2.1, uttrykk (8.12). Formelen kan ytterligere forenkles til,

$$c(t) = A_{i-1}\ \varphi(t) + B_i(t)(A_i - A_{i-1})\varphi(t)$$

som gir oss følgende metode.

Å utvide en parametrisk kurve til blendingsplines

Gitt en kurve $\varphi(t)$. For å konvertere $\varphi(t)$ til en blendingsplinekurve må vi
- bestemme antallet n av redigeringskuber vi vil bruke,
- lage en skjøtvektor for en 2.-ordens B-spline, dvs. $\{t_i\}_{i=0}^{n+1}$,
- til hver indre skjøt tildeler vi en homogen (identitets)matrise.

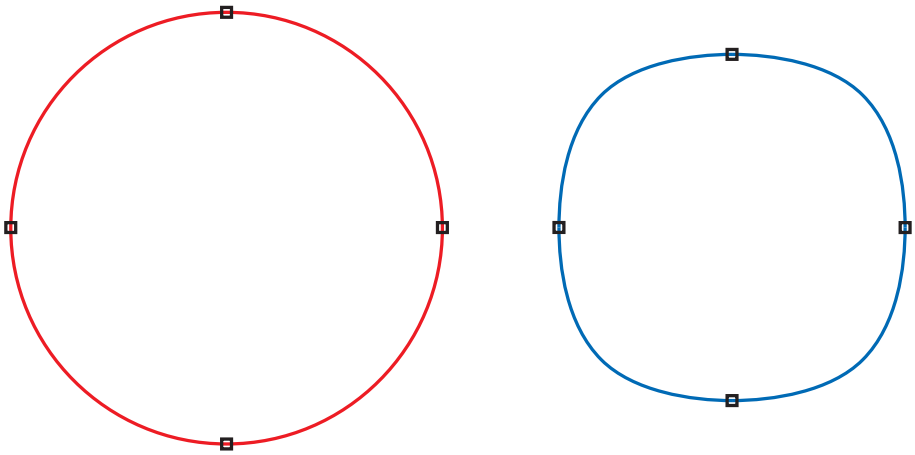
Da har vi følgende formel for skjøtintervallet $[t_i, t_{i+1})$:

$$c(t) = (A_{i-1} + B_i(t)\ \Delta A_i)\ \varphi(t) \tag{8.18}$$

hvor A_i er matrisen i skjøten t_{i+1}, og $\Delta A_i = A_i - A_{i-1}$, $B_i(t) = B \circ w_{1,i}(t)$ er splinebasisen med B-funksjon og $w_{1,i}(t)$ definert i (6.11). Merk at $\varphi(t)$ er et punkt, med den homogene koordinaten 1. For de deriverte har vi samme struktur som i (8.9) og dermed (8.11), dvs. vi bruker $B_i^{(k)}(t) = \delta_{1,i}^k B^{(k)} \circ w_{1,i}(t)$, definert i (8.10). De tre første deriverte er dermed:

$$\begin{aligned}
c'(t) &= A_{i-1}\ \varphi'(t) + \Delta A_i \left(\underline{B_i'(t)\ \varphi(t) + B_i(t)\ \varphi'(t)} \right), \\
c''(t) &= A_{i-1}\ \varphi''(t) + \Delta A_i \left(\underline{B_i''(t)\ \varphi(t) + 2B_i'(t)\ \varphi'(t) + B_i(t)\ \varphi''(t)} \right), \\
c'''(t) &= A_{i-1}\ \varphi'''(t) + \Delta A_i \left(\underline{B_i'''(t)\ \varphi(t) + 3B_i''(t)\ \varphi'(t) + 3B_i'(t)\ \varphi''(t) + B_i(t)\ \varphi'''(t)} \right).
\end{aligned}$$

Legg merke til at i den første delen multipliseres A_{i-1} med en vektor hvor den homogene koordinaten er 0, og i den andre delen multipliseres ΔA_i med et punkt (understreket) der den homogen koordinat er 1.



Figur 8.14: *En blendingsplinekurve laget av en sirkel, (4.1). Til venstre ser vi den innledningsvis, en perfekt sirkel hvor de fire redigeringspunktene er markert. Til høyre er kurven endret ved å flytte de fire redigeringspunktene nærmere midten.*

I seksjon 8.2.1 blei homogene matriser A_i introdusert. Matrisene representerer lokale koordinatsystem i $\mathbb{R}^3$, plassert i et punkt p_i og med koordinataksene x_i, y_i og z_i. Matrisen A_i, punktet og aksene kan visualiseres som kuber. Dette ser vi i figurene 8.6, 8.7 og 8.14.

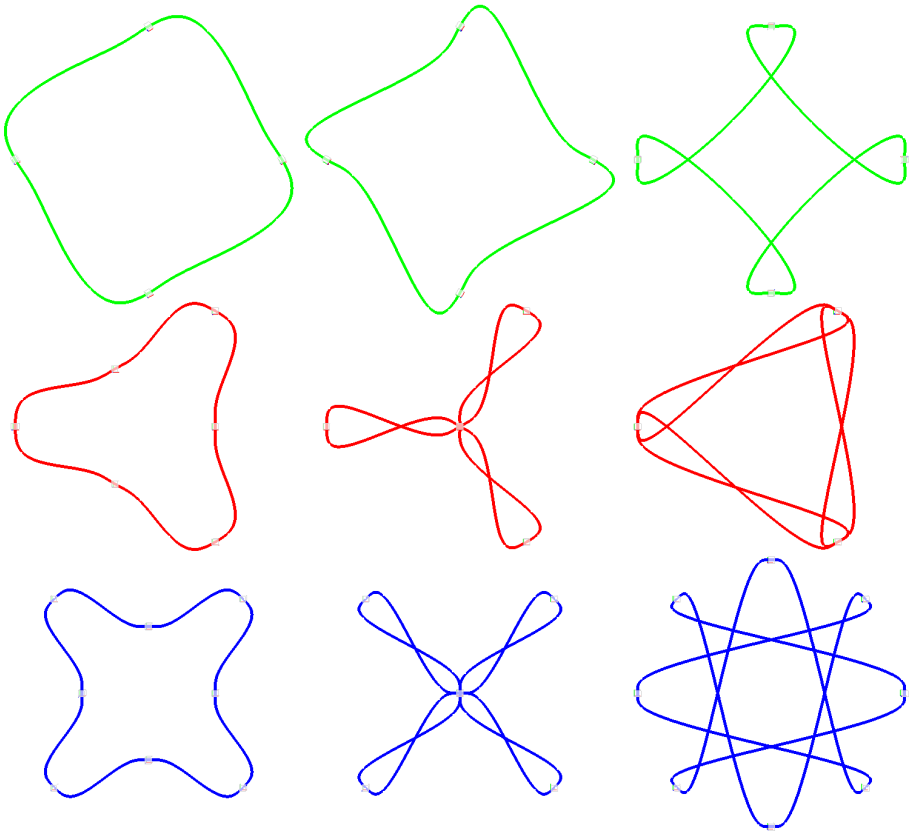
Vi skal nå se på et eksempel, en sirkel gitt av formelen (4.1). Domenet er $[0, 2\pi)$ og det eneste vi kan gjøre her er å endre radius fordi kurven er låst til formelen. Vi bestemme oss for å bruke 4 redigeringspunkt som vi fordeler uniformt, samt hvilken B-funksjon vi vil bruke. Kurven er lukket, og fra seksjon 6.2.2 ser vi at domenet er $[t_1, t_5)$. Skjøtvektoren blir da $\tau = \{-\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi\}$. De fire matrisene $\{A_i\}_{i=0}^{3}$ er i utganspunktet identitetsmatriser. I eksempelet er (7.31) brukt som blendingsfunksjon.

Figur 8.14 viser dette eksemplet. Kurven er i utgangspunktet en perfekt sirkel, som vi ser til venstre i figuren. Til høyre i figuren ser vi kurven etter at alle fire redigeringspunktene er flyttet nærmere midten av sirkelen. Redigeringspunktene kan nå flyttes i forskjellige retninger og de kan roteres og skaleres. Kurven vil totalt endre form. men den vil alltid være lukket og alltid topologisk lik en sirkel.

Figur 8.14 viser dette eksemplet. Kurven er i utgangspunktet en perfekt sirkel, som vi ser til venstre i figuren. Til høyre i figuren ser vi kurven etter at alle fire redigeringspunktene er flyttet nærmere midten av sirkelen. Redigeringspunktene kan nå flyttes i forskjellige retninger og de kan roteres og skaleres. Kurven vil totalt endre form. men den vil alltid være lukket og alltid topologisk lik en sirkel.

Et problem i dette eksemplet, og også i oppskriften på forrige side, er at de lokale koordinatsystemene for redigeringspunktene er det globale koordinatsystemet. For å gjøre det enklere å endre/redigere kurven bør vi følge rådene fra seksjon 8.2.1 på side 155, vi bør justere de lokale kurvene, og sette inn tilsvarende rotasjon og translasjon i matrisene $A_i,\ i = 0, 1, 2, 3$.

Derfor legger vi til en ekstra matrise ved hvert redigeringspunkt, $\mathfrak{T}_i$. Dvs. at vi for hvert redigeringspunkt må:



Figur 8.15: *En blendingsplinekurve laget av en sirkel, formel (4.1). De grønne kurvene har 4 skjøtintervall. Ved hver skjøt roteres alle delkurvene, henholdsvis* $30°$, $60°$ *og* $180°$ *for kurvene sett fra venstre. I de røde kurvene med 6 skjøtintervall og de blå med 8 skjøtintervall flyttes annenhver delkurve nærmere midten av sirkelen, henholdsvis* $\frac{1}{2}$, *1, 2 ganger avstand til sentrum.*

- sette $A_i = F(t_{i+1})$, dvs. lage Frenet-ramme i hvert punkt $\varphi(t_{i+1}),\ i = 1,2,\ldots,n$. Frenet-ramme er omtalt på side 155 og i seksjon 4.1.1,
- Deretter setter vi $\mathfrak{T}_i = A_i^{-1},\ i = 0,1,\ldots,n-1$.

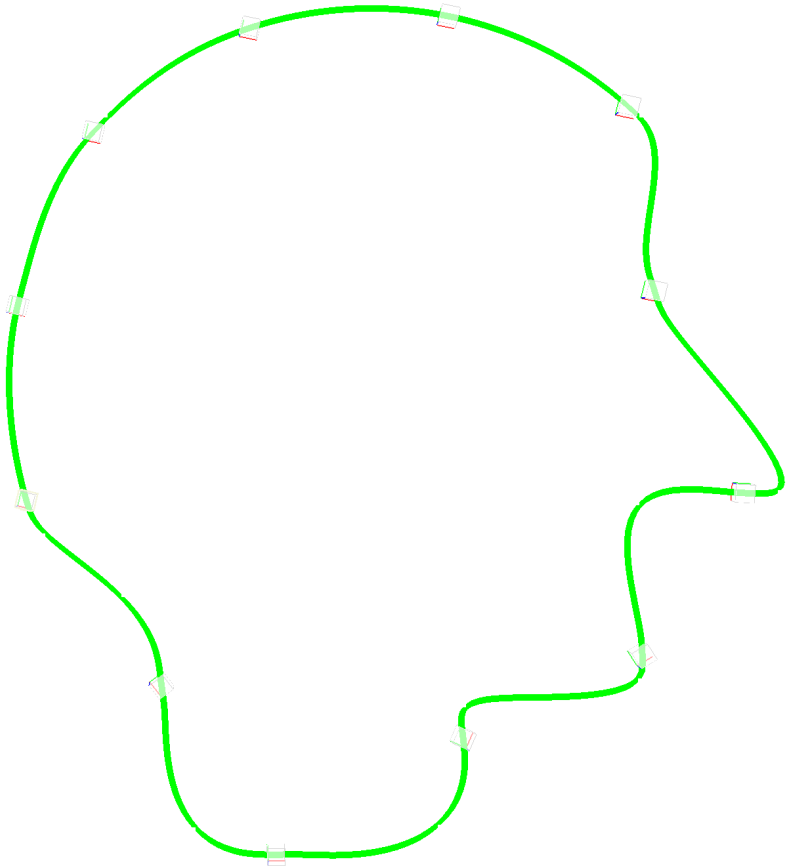
Konsekvensen er at “evalueringen" av den opprinnelige kurven nå avhenger av skjøtintervallet, dvs. uttrykk (8.18) blir nå

$$c(t) = (A_{i-1} + B_i(t)\ \Delta A_i)\,(\mathfrak{T}_i\ \varphi(t))$$

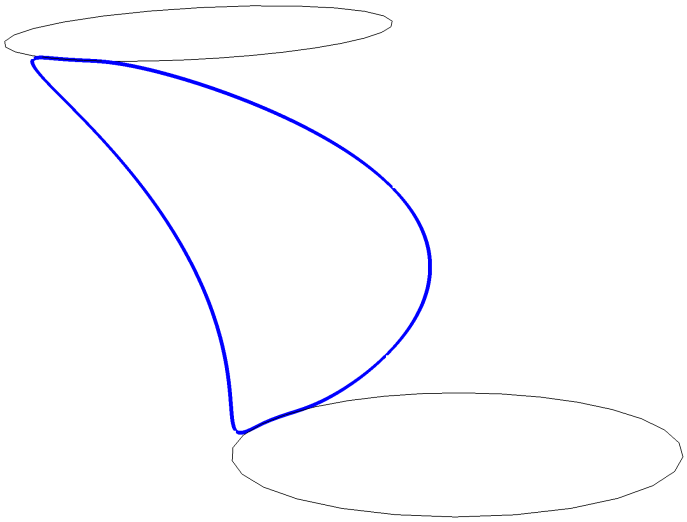
og en tilsvarende endring må da også gjøres for de deriverte. Når redigeringen er fullført, kan A_i oppdateres ved å multiplisere A_i med $\mathfrak{T}_i$, for deretter å fjerne $\mathfrak{T}_i$.

Figur 8.15 viser flere eksempler på delkurvekonstruksjonen. De grønne kurvene viser bruken av rotasjon og de røde og blå kurvene viser bruken av translasjon. Merk at kun matrisene A_i endres, rotasjonen vises i den 3×3 undermatrisen øverst til venstre, se side 16. For flytting endres kun kolonnevektoren helt til høyre, som beskrevet i (8.12).

Figur 8.16 og 8.17 viser hvordan vi interaktivt kan forme figurer ved hjelp av rotasjon og translasjon. Det kan sammenlignes med å tegne med blyant, og er gjort på sekunder.



Figur 8.16: *En silhuett av et ansikt modellert fra en sirkel med 12 skjøter.*



Figur 8.17: *Kun 2 skjøter er brukt, begge er rotert og en er flyttet. To sirkler er plottet for å illustrere blendingen.*

Del II

Flater

Kapittel 9

Parametriske flater

Se for deg $\mathbb{R}^2$ som et stort plan. For å lage en flate velger vi en del av dette planet som vi så kaller et parameterplan/domene. Så deformerer vi det ved enten å strekke, trykke, bøye det på forskjellige måter. Til slutt putter vi det i ønsket posisjon og orientering inn i et rom, vanligvis det euklidske rommet $\mathbb{R}^3$. Merk at å "klippe og lime" **ikke** er et alternativ her. Figur 9.2 viser oss et eksempel på konseptet som er beskrevet over. Dette betyr at en flate er et 2-dimensjonalt objekt. Hvis en flate verken er degenerert eller selvskjærende kan vi også kalle den for en 2-dimensjonal mangfoldighet. Beskrivelsen over er et forsøk på å illustrere konseptet med parametriske flater. Som oftest er flaten plasert inn i $\mathbb{R}^3$, men den kan egentlig være plasert inn i et annet (euklidsk) rom eller i en mangfoldighet av hvilken som helst dimensjon > 0. En mer formell definisjon er:

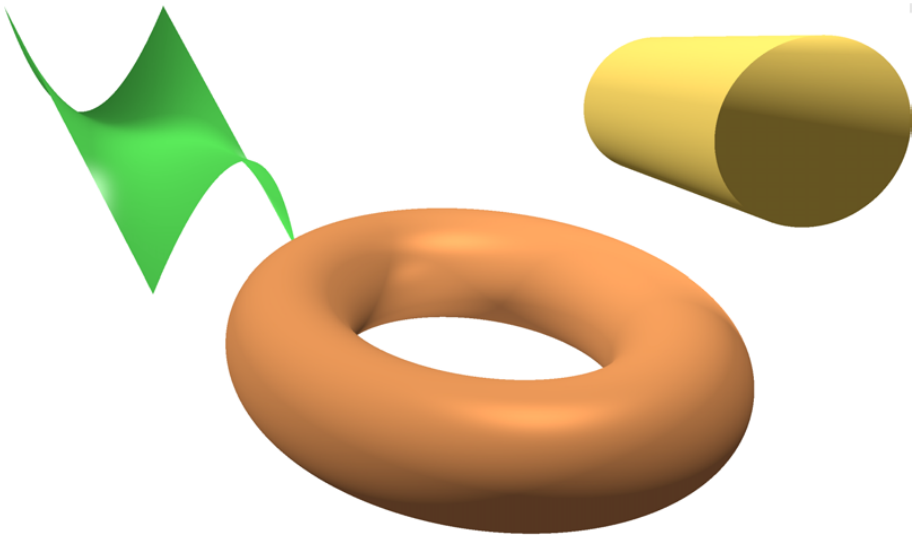
Definisjon 9.1. *En parametriserte differensierbar flate er et differensierbart kart/avbildning $S : U \subset \mathbb{R}^2 \longrightarrow \mathfrak{M}^n,\ n > 0$ av et åpent sett $U \subset \mathbb{R}^2$ inn i et euklidsk rom eller en mangfoldighet. Merk at et halvåpene og lukkede sett bare er avgrensninger av åpne sett.*

Ofte tenker man seg at en flate er overflata til et tredimensjonalt objekt i rommet. Vanligvis er da flatene lukket og topologisk lik en kule eller torus, eller en toruslignende gjenstand med flere hull. Det er ikke alltid like lett å parametrisere slike flater og det er kun en torus som enkelt kan parametriseres. Alle andre typer må dekkes av flere parametriseringer der overgangene mellom dem må være konsistente, dvs.homeomorfier.

I resten av kapittelet vil vi anta at flatene er i $\mathbb{R}^3$. Et kart/avbildning er vanligvis markert med en bokstav, f.eks. $S : U \subset \mathbb{R}^2 \to \mathbb{R}^3$ der U kalles domenet/parameterplanet, hvor koordinatene ofte er angitt med bokstavene (u, v). En flate kan ha en ytre rand som i "apesadel"-eksemplet (9.1) eller en delvis ytre rand som i en sylinder (9.2), eller ingen ytre render som i en torus (9.3). Det første eksemplet vi skal se på er en "apesadel" definert ved,

$$S(u,v) = \begin{pmatrix} u \\ v \\ uv^2 \end{pmatrix}, \quad u \in [-1,\ 1], \quad v \in [-1,\ 1]. \tag{9.1}$$

Her er domenet $U = [-1,\ 1] \times [-1,\ 1]$ og parameterplanet samsvarer med xy-planet i $\mathbb{R}^3$, $x = u$ og $y = v$. Flaten strekkes i avbildningen og arealet blir følgelig større enn parameterplanets areal som er 4. Den grønne flaten i figur 9.1 er en "apesadel".



Figur 9.1: *Tre eksempler på flater, en såkalt apesadel fra (9.1) i grønt, en sylinder fra (9.2) i gult og en torus fra (9.3) i "kobberfarge".*

Det neste eksempelet er en sylinder, med formelen

$$S(u,v) = \begin{pmatrix} \cos v \\ \sin v \\ u \end{pmatrix}, \quad u \in [0,\ 6], \quad v \in [0,\ 2\pi). \tag{9.2}$$

Her er domenet $U = [0,\ 6] \times [0,\ 2\pi)$. Denne avbildningen er som å ta et stykke papir og rulle det. Overflaten strekkes ikke i prosessen og arealet er det samme som for parameterplanet, dvs. 12π. Den gule flaten vi ser i figur 9.1 er en sylinder.

Eksampel tre er en torus,

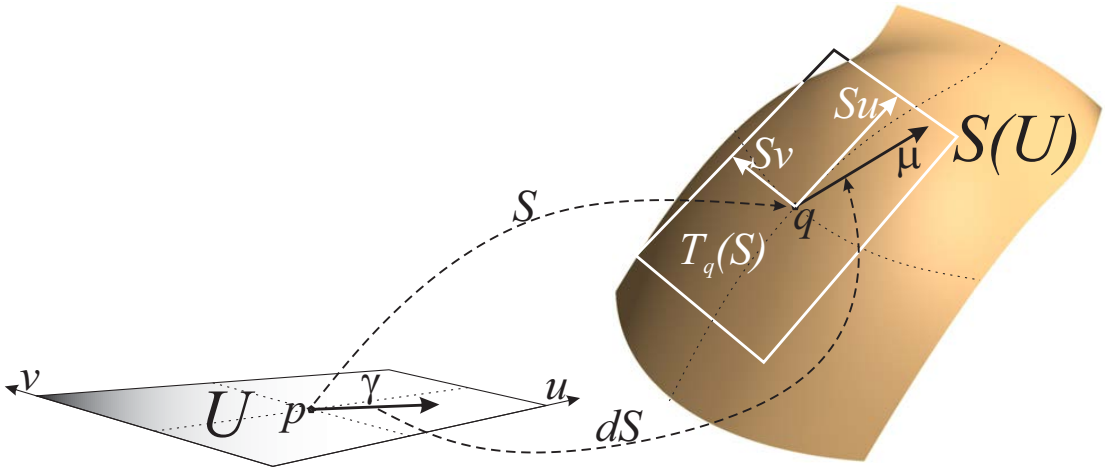
$$S(u,v) = \begin{pmatrix} (\cos u + 3)\cos v \\ (\cos u + 3)\sin v \\ \sin u \end{pmatrix}, \quad u \in [0,\ 2\pi), \quad v \in [0,\ 2\pi). \tag{9.3}$$

Her er domenet $U = [0,\ 2\pi) \times [0,\ 2\pi)$. Et plott av flaten er vist nederst i figur 9.1, i en slags "kobberfarge".

Figur 9.2 er en visuell illustrasjon av hva en parametrisk flate er. I figuren ser vi til venstre parameterplanet U med koordinatene er u og v. Avbildningen er kalt S og vises som en stiplet buet pil. Til høyre i figuren ser vi avbildningen av parameterplanet i rommet $\mathbb{R}^3$, dvs. flaten $S(U)$. I figuren ser vi også at S overfører et punkt p i parameterplanet til et punkt q på flaten.

En Oppsummering av figur 9.2 er at:
- U er parameterplanet/domenet, $S(U)$ er hele flaten i R^3.
- $p = (u,v)$ er et punkt i parameterplanet $U \subset \mathbb{R}^2$, der $(u,\ v)$ er koordinatene.
- $q = S(p) = S(u,v)$ er et punkt på flaten og er dermed også i $\mathbb{R}^3$ (med 3 koordinater).
- S avbilder punkt i parameterplanet til punkt på flaten og følgelig i $\mathbb{R}^3$.

Vi skal i seksjon 9.1.1 forklare hva dS i figur 9.2 er og gjør.



Figur 9.2: *Nede til venstre ser vi parameterplanet $U \subset \mathbb{R}^2$. Øverst til høyre ser vi flaten $S(U)$ i rommet $\mathbb{R}^3$. Et punkt p i parameterplanet avbildes av S til et punkt $q = S(p)$ på flaten i rommet. En vektor γ fra punktet p i parameterplanet er avbildet ved differensialavbildningen dS til en vektor $\mu = dS_p(\gamma)$ som ligger i tangentplanet $T_q(S)$ i punktet q på flaten.*

9.1 Differensiering

Det neste trinnet, som også er vist i figur 9.2, er differensiering. Vi vet at S overfører et punkt p fra parameterplanet til et punkt q på flaten. Vi skal nå finne retningen vi må bevege oss i hvis vi er i q og ønsker å bevege oss i en retningen hvor bare en av de to parameterverdiene endres. Vi kaller vektorene som beskriver disse retningene for partiellderiverte. En overflate har to parameterverdier og dermed to partiellderiverte. Det er flere forskjellige notasjoner som brukes for partiellderiverte. I det følgende vil vi se 3 forskjellige notasjoner som er vanlig å bruke.

- Den partiellderiverte i u-retningen, angitt på tre forskjellige måter: $D_uS = S_u = \frac{\partial S}{\partial u}$, er en vektor (er 3-dimensjonal hvis $S(U) \subset \mathbb{R}^3$), og er en tangentvektor til flaten i den gitte posisjonen $q = S(p)$.
- Den partiellderiverte i v-retningen, angitt på tre forskjellige måter $D_vS = S_v = \frac{\partial S}{\partial v}$, er en vektor (er 3-dimensjonal hvis $S(U) \subset \mathbb{R}^3$), og er en tangentvektor til flaten i den gitte posisjonen $q = S(p)$.

De to tangentvektorene S_u og S_v spenner ut et tangentplan i punktet $q = S(p)$. Tangentplanet, betegnet $T_q(S)$, er også vist i figur 9.2 som en hvit firkant. Husk at dimensjonen til tangentvektorene, de partielle deriverte, er lik dimensjonen til rommet som flaten er i. Som et eksempel på partiellderiverte kan vi se på "apesadlen" i uttrykk (9.1). Vi får da følgende partiellderiverte (i alle tre notasjonene),

$$D_uS = S_u = \frac{\partial S}{\partial u} = \begin{pmatrix} 1 \\ 0 \\ v^2 \end{pmatrix}, \qquad D_vS = S_v = \frac{\partial S}{\partial v} = \begin{pmatrix} 0 \\ 1 \\ 2uv \end{pmatrix}, \tag{9.4}$$

9.1.1 Differensialen dS_p

Det neste avbildningen er differensialen dS, en matrise $\mathbb{M}$ der de to partiellderiverte er kolonnene,

$$dS = [S_u \; S_v], \quad \in \mathbb{M}(g,2),$$

der g er dimensjonen til rommet flaten er i. Dette betyr at dS har to kolonner og antall rader er lik dimensjonen til rommet flaten er i (dvs. 3 i $\mathbb{R}^3$). Den overfører en vektor i punktet p i parameterplanet, til en vektor i tangentplanet $T_q(S)$ i punktet $q = S(p)$.

I figur 9.2 er det et eksempel hvor en vektor $\gamma \in \mathbb{R}^2$ er avbildet til en vektor μ i rommet av dS_p, dvs. $dS_p(\gamma) = \mu$. Vi kaller μ den retningsderiverte i retning γ. Hvis flaten er i $\mathbb{R}^3$ så er dS_p en 2×3 matrise, dvs.

$$dS_p : \mathbb{R}^2 \to \mathbb{R}^3, \quad p \in U \subset \mathbb{R}^2. \tag{9.5}$$

Det følger også at d er en differensialoperator som overfører en avbildning til en differensialavbildning, vi får dermed

$$\begin{aligned} d\,(S_u) &= [(S_u)_u \; , \; (S_u)_v] = [S_{uu} \; , \; S_{uv}]\,, \\ d\,(S_v) &= [(S_v)_u \; , \; (S_v)_v] = [S_{vu} \; , \; S_{vv}]\,. \end{aligned}$$

Differensialopperatoren kan anvendes på enhver avbildning og lager da en differensialavbildning. I de neste seksjonene vil vi se flere eksempler på dette.

9.1.2 Kurver på flater

Vi skal se på konstruksjonen av hvordan kurver som er definert i parameterplanet til en flate er avbildet i $\mathbb{R}^3$.

Gitt en kurve $h : I \subset \mathbb{R} \to \mathbb{R}^2$,

$$h(t) = \sum_{i=1}^{n} c_i b_i(t).$$

der $c_i, \;\; i = 1, ..., n$ er punkt eller vektorer i $\mathbb{R}^2$ og $b_i(t)$ er basisfunksjoner som spenner ut et endelig dimensjonalt funksjonsrom.

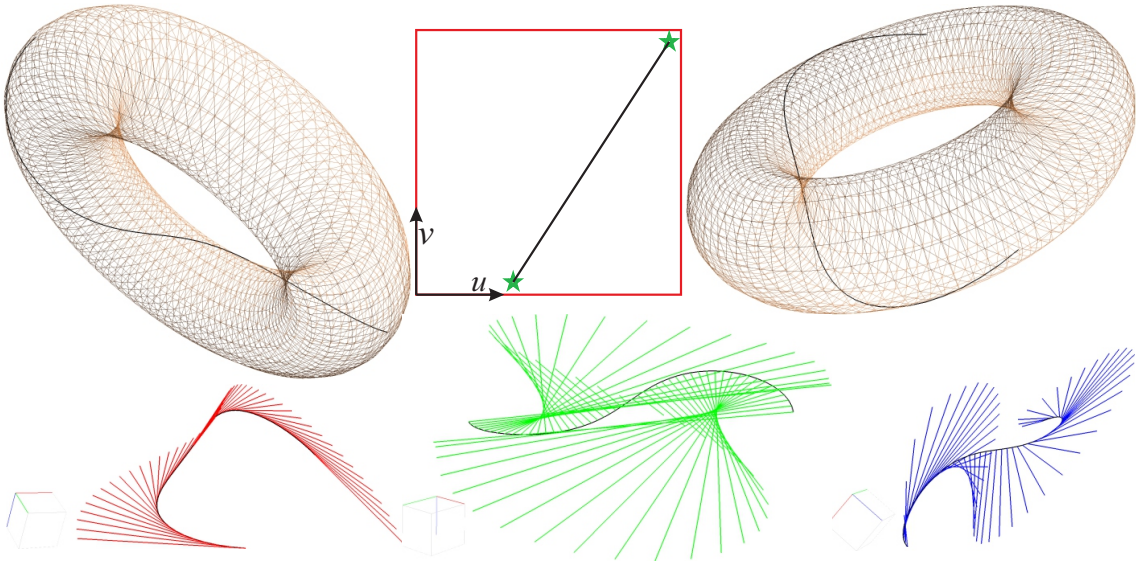
Spørsmålet er hvilken kurve vi får når kurven i parameterrommet blir overført til $\mathbb{R}^3$. Gitt en flate $S : U \subset \mathbb{R}^2 \to \mathbb{R}^3$. Kurvedefinisjonen er da,

$$c(t) = S\,(h(t)) = S \circ h(t), \quad t \in I \subset \mathbb{R}. \tag{9.6}$$

Ved å bruke kjerneregelen på (9.6) får vi 1.-, 2.- og 3.-deriverte,

$$\begin{aligned} c'(t) &= dS_h(h'), \qquad \text{(en forenkling, skal egentlig stå } h(t) \text{ og } h'(t)) \\ c''(t) &= d(dS_h(h'))_h(h') + dS_h(h''), \\ c'''(t) &= d(d(dS_h(h'))_h(h'))_h(h') + d(dS_h(h'))_h(h'') + d(dS_h(h''))_h(h') + dS_h(h'''), \end{aligned} \tag{9.7}$$

der differensialmatrisene som brukes i de deriverte i (9.7) er (for å forenkle hopper vi over posisjonsindeksen h i det følgende,



Figur 9.3: *Øverst ser vi parameterplanet til en torus sammen med torusen vist fra to forskjellige posisjoner. I parameterplanet er det en kurve (rett linje mellom to punkt). På torusen blir dette en kurve i rommet. Under vises den samme kurven fra tre nye posisjoner. Til venstre er den plottet sammen med 1.-deriverte som røde vektore, i midten med 2.-deriverte i grønt og til høyre med 3.-deriverte i blått.*

$$\begin{aligned} d(dS(h')) &= d\left([S_u\ ,\ S_v]\ h'\right) \\ &= \left[\left([S_u\ ,\ S_v]\ h'\right)_u\ ,\ \left([S_u\ ,\ S_v]\ h'\right)_v\right] \\ &= \left[[S_{uu}\ ,\ S_{vu}]\ h'\ ,\ [S_{uv}\ ,\ S_{vv}]\ h'\right], \end{aligned}$$

og

$$\begin{aligned} d(d(dS(h'))(h')) &= d\left(\left[[S_{uu}\ ,\ S_{vu}]h'\ ,\ [S_{uv}\ ,\ S_{vv}]h'\right]\ h'\right) \\ &= \left[\left(\left[[S_{uu}\ ,\ S_{vu}]h'\ ,\ [S_{uv}\ ,\ S_{vv}]h'\right]\ h'\right)_u\ ,\ \left(\left[[S_{uu}\ ,\ S_{vu}]h'\ ,\ [S_{uv}\ ,\ S_{vv}]h'\right]\ h'\right)_v\right] \\ &= \left[\left[[S_{uuu}\ ,\ S_{vuu}]h'\ ,\ [S_{uvu}\ ,\ S_{vvu}]h'\right]\ h'\ ,\ \left[[S_{uuv}\ ,\ S_{vuv}]h'\ ,\ [S_{uvv}\ S_{vvv}]h'\right]\ h'\right]. \end{aligned}$$

Et eksempel er vist i figur 9.3. En 1.-grads bézierkurve, dvs en rett linje i parameterplanet er gitt. Parameterplanet tilhører en torus. I øvre del av figur 9.3 vises også torusen med kurven sett fra to forskjellige posisjoner. Under er det tre plott av bare kurven, men nå med dens 1.- deriverte i rødt (til venstre), dens 2.-deriverte i blått (i midten) og dens 3.-deriverte i grønt, alle laget fra formlene i (9.7).

Et annet eksempel på “kurver" på en flate er en vektor-funksjon som beskriver en retningsderivert langs en kurve, ikke i kurveretningen men tilnærmet “vinkelrett" på kurveretningen. I dette tilfellet trenger vi både en punkt-funksjon, som beskriver posisjonen i parameterplanet, og en vektor-funksjon, som beskriver en vektoren i parameterplanet, $\mathbb{R}^2$, i det relaterte punktet. Disse to koblede funksjonene kan da formuleres som:

$$h(t) = \sum_{i=1}^{n_1} c_i b_i(t), \quad \text{og} \quad r(t) = \sum_{i=1}^{n_2} d_i b_i(t), \tag{9.8}$$

begge for $t \in I \subset \mathbb{R}$. Overføringen fra parameterplanet til $\mathbb{R}^3$ kaller vi g og blir

$$g(t) = dS_{h(t)}\left(r(t)\right), \tag{9.9}$$

og den deriverte er,

$$\begin{aligned} g'(t) &= d(dS(r))(h') + dS(r'), \\ g''(t) &= d(d(dS(r))(h'))(h') + d(dS(h'))(r') + d(dS(r'))(h') + dS(r''), \end{aligned} \tag{9.10}$$

der differensialmatrisene er,

$$\begin{aligned} d(dS(r)) &= d\left([S_u\ ,\ S_v]\ r\right) \\ &= [([S_u\ ,\ S_v]\ r)_u\ ,\ ([S_u\ ,\ S_v]\ r)_v] \\ &= [[S_{uu}\ ,\ S_{vu}]\ r\ ,\ [S_{uv}\ ,\ S_{vv}]\ r], \end{aligned}$$

og

$$\begin{aligned} d(d(dS(r))(h')) &= d\left([[S_{uu}\ ,\ S_{vu}]\, r\ ,\ [S_{uv}\ ,\ S_{vv}]\, r]\ h'\right) \\ &= \left[\left([[S_{uu}\ ,\ S_{vu}]\, r\ ,\ [S_{uv}\ ,\ S_{vv}]\, r]\ h'\right)_u\ ,\ \left([[S_{uu}\ ,\ S_{vu}]\, r\ ,\ [S_{uv}\ ,\ S_{vv}]\, r]\ h'\right)_v\right] \\ &= \left[\left[[S_{uuu}\ ,\ S_{vuu}]\, r'\ ,\ [S_{uvu}\ ,\ S_{vvu}]\, r\right]\ h'\ ,\ [[S_{uuv}\ ,\ S_{vuv}]\, r\ ,\ [S_{uvv}\ S_{vvv}]\, r]\ h'\right]. \end{aligned}$$

Eksempeler på bruk av denne typen av funksjoner, (9.8), (9.9) og (9.10) er gitt i seksjonene 9.6.4 og 14.1.

9.1.3 Tangentplanet $T_q(S)$

Vi tar utganspunkt i alle kurver c i en flate som går igjennom et punkt $q = S(p)$ i flata. Den 1.-deriverte til disse kurvene i punktet q er, i henhold til (9.7),

$$c' = dS_p(h') = [S_u\ ,\ S_v] \begin{pmatrix} h'_u \\ h'_v \end{pmatrix} = h'_u\, S_u + h'_v\, S_v.$$

Vi ser at den 1.-deriverte / tangentvektoren til alle kurvene i flata som går igjennom q kan beskrives som en lineærkombinasjon av de to vektorene S_u og S_v. Dette viser at alle ligger i planet som spennes ut av S_u og S_v. Dette planet er følgelig et tangentplan til flaten $S(U)$ i punktet q, og vi kaller det for

$$T_q(S) = dS_q(\mathbb{R}^2).$$

Det er opplagt at tangentplanet er det samme selv om vi forandrer parametriseringen. Det er følgelig en iboende egenskap som er uavhengig av parametriseringen.

I $\mathbb{R}^3$ kaller vi vektoren som står ortogonalt på tangentplanet for normalen til flata. Notasjonene og orienteringen er som følger. Først definerer vi normalen med en liten bokstav,

$$n_q = S_u \wedge S_v(p), \tag{9.11}$$

hvor $\wedge$ er vektorproduktet i $\mathbb{R}^3$. Så definerer vi enhetsnormalen som

$$N_q = \frac{n_q}{|n_q|}. \tag{9.12}$$

Enhetsnormalen er dermed også uavhengig av parametriseringen og er følgelig også en iboende egenskap.

9.1.4 Første fundamentale form

Vi skal her se nærmere på en geometrisk struktur som er koblet til tangentplanet til en flate. *Første fundamentale form* er et indreprodukt i tangentplanet til en flate som er i et 3D-euklidsk rom, dvs. det er et indreprodukt av tangentvektorer. Den gir oss muligheten til å beregne krumning og metriske egenskaper til en flate som lengde og areal i samsvar med rommet flata er i. *Første fundamentale form* er benevnt med romertallet I og er:

$$I_q(a,b) = \langle a,b\rangle_q, \quad a,b \in T_q(S) \quad \text{og hvor } q \text{ er et punkt på flaten } S.$$

Gitt en flate $S : U \in \mathbb{R}^2 \to \mathbb{R}^3$ med en gitt parametrisering, og punktet $p = (u,v) \in U$. Da er $S(p)$ et punkt på flaten. Hvis vi så har to vektorer i $\mathbb{R}^2$, $\hat{r} = (a,b)$ og $\hat{s} = (c,d)$ da vil indreproduktet av disse to tangentvektorene $r = dS_p(\hat{r})$ og $s = dS_p(\hat{s})$ være:

$$\begin{aligned} I_q(r,s) &= \langle dS_p(\hat{r}), dS_p(\hat{s})\rangle_{S(p)} \\ &= \langle aS_u + bS_v, cS_u + dS_v\rangle \\ &= ac\langle S_u, S_u\rangle + (ad+bc)\langle S_u, S_v\rangle + bd\langle S_v, S_v\rangle \\ &= ac\,E + (ad+bc)\,F + bd\,G, \end{aligned}$$

hvor E, F og G er koeffisientene til første fundamentale form, og vi har da -

Første fundamentale form

Koeffisientene er

$$\begin{aligned} E &= \langle S_u, S_u\rangle \\ F &= \langle S_u, S_v\rangle \\ G &= \langle S_v, S_v\rangle \end{aligned} \tag{9.13}$$

Første fundamentale form, representert som en symmetrisk matrise.

$$I_q(r,s) = \hat{r}^T \begin{pmatrix} E & F \\ F & G \end{pmatrix} \hat{s}.$$

hvor $\hat{r},\ \hat{s} \in \mathbb{R}^2$ og koblet til punktet p i parameterplanet, og $r,\ s \in T_q(S)$, dvs. tangentplanet til S i punktet $q = S(p)$.
Hvis vektorene r og s er samme vektor μ vil første fundamentale form bli

$$I_q(\mu) = \hat{\mu}^T \begin{pmatrix} E & F \\ F & G \end{pmatrix} \hat{\mu}.$$

hvor $\mu = dS_p(\hat{\mu})$ og $\hat{\mu} \in \mathbb{R}^2$.

Første fundamentale form er en fullstendig beskriver de metriske egenskapene til en flate. Dermed kan vi beregne lengdene til kurver på flaten og arealene til områder på flaten. I det følgende skal vi se nærmere på et linje-element ds og et areal-element dA, for så å gi et eksempel på bruken av hver av dem.

Linje-elementet ds kan uttrykkes med koeffisientene til første fundamentale form, som

$$ds^2 = I_q\,(d\hat{s})$$

hvor $d\hat{s} = \begin{pmatrix} du \\ dv \end{pmatrix}$, er en vektor i $\mathbb{R}^2$ (jmf. μ i rammen over). Vi får da

$$ds^2 = (du\ ,\ dv)\begin{pmatrix} E & F \\ F & G \end{pmatrix}\begin{pmatrix} du \\ dv \end{pmatrix} = E\,du^2 + 2F\,dudv + G\,dv^2. \tag{9.14}$$

Det klassiske areal-elementet $dA = |S_u \wedge S_v|\,dudv$ kan uttrykkes med første fundamentale form ved hjelp av Lagranges identitet dvs. $|a \wedge b|^2 = \langle a,a\rangle\langle b,b\rangle - \langle a,b\rangle^2$, som gir

$$dA = |S_u \wedge S_v|\,dudv = \sqrt{EG - F^2}\,dudv. \tag{9.15}$$

✓ Første eksempel er å beregne lengden l til en kurve c på en flate, med å bruke (9.14),

$$l(c) = \int_a^b \frac{ds}{dt}\,dt = \int_a^b \sqrt{E\,u'^2 + 2F\,u'v' + G\,v'^2}\,dt$$

Ved å bruke kurven på en torus, gir formelen i 9.3), og som er vist i figur 9.3,

$$S_u = \begin{pmatrix} -\sin u\cos v \\ -\sin u\sin v \\ \cos u \end{pmatrix} \quad \text{og} \quad S_v = \begin{pmatrix} -(\cos u + 3)\sin v \\ (\cos u + 3)\cos v \\ 0 \end{pmatrix},$$

og da er,

$$E = 1, \quad F = 0, \quad G = (\cos u + 3)^2$$

Kurven i figur 9.3 definert i parameterplanet til torusen er

$$h(t) = \begin{pmatrix} 2.3 \\ 0.3 \end{pmatrix} + \begin{pmatrix} 3.7 \\ 5.7 \end{pmatrix} t, \quad \text{og den deriverte} \quad h'(t) = \begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} 3.7 \\ 5.7 \end{pmatrix},$$

som gir

$$l(c) = \int_0^1 \sqrt{13.69 + 32.49(\cos(2.3 + 3.7t) + 3)^2}\,dt \approx 16,$$

hvor den tilnærmede løsningen er beregnet med å bruke rombergintegrasjon, dvs en modifisert algoritme 14 som vi finner på side 290.

✓ Det andre eksemplet er å beregne arealet til torusen definert i (9.3) ved å bruke (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 Andre fundamentale form

Andre fundamentale form (eller form-tensoren) er en såkalt “kvadratisk form" i tangentplanet til en glatt flate i $\mathbb{R}^3$, vanligvis betegnet med II (les “to"). Sammen med første fundamentale form gjør den det mulig for oss på en enkel måte å definere ytre invarianter til flaten, dens prinsipale krumninger.

Enhetsnormalen (i punktet q) N_q er definert i (9.12). Det er en nær sammenheng mellom en enhetssfære og en enhetsnormal. Den kan betraktes som en avbildning, dvs.

$$N : S \to S^2,$$

der S^2 er den 2-dimensjonale enhetenssfæren (en kule). Dette kalles Gauss-avbildningen til S. Det følger at differensialen til Gauss-avbildningen N er

$$dN_q : T_q(S) \to T_{N_q}(S^2).$$

Men siden $T_q(S)$ og $T_{N_q}(S^2)$ er parallelle plan, følger det at at vi kan si at

$$dN_q : T_q(S) \to T_q(S).$$

Den lineære differensialen dN_q er helt klart relatert til endring av den retningsderiverte fordi normalen er ortogonale til alle tangentene i punktet. Det kan vises at $|dN_q(r)|$ er krumningen i retningen til en *enhetsvektoren r* som ligger i tangentplanet.

Andre fundamentale form er en kvadratisk form i tangentplanet og et indreprodukt. Gitt en vektor $\mu = dS_p(\hat{\mu}) \in T_p(S)$, hvor $\widehat{\mu} = (a,b) \in \mathbb{R}^2$. Andre fundamentale form er:

$$\begin{aligned} II_p(\mu) &= -\langle dN_q(\mu), \mu \rangle \\ &= -\langle N_u a + N_v b, S_u a + S_v b \rangle \\ &= -a^2 \langle N_u, S_u \rangle - ab\left(\langle N_u, S_v \rangle - \langle N_v, S_u \rangle\right) - b^2 \langle N_v, S_v \rangle \\ &= a^2\ e + 2ab\ f + b^2\ g, \end{aligned}$$

hvor e, f og g er koeffisientene til andre fundamentale form. Fra definisjonen av normalen følger det at $\langle N, S_u \rangle = \langle N, S_v \rangle = 0$. De deriverte både med hensyn til u og v er derfor også null. Da følger det at $\langle N, S_u \rangle_u = \langle N, S_{uu} \rangle + \langle N_u, S_u \rangle = 0$, og så videre. Dette gir oss -

Andre fundamentale form

Koeffisientene til andre fundamentale form er

$$\begin{aligned} e &= \langle N, S_{uu} \rangle = -\langle N_u, S_u \rangle, \\ f &= \langle N, S_{uv} \rangle = -\langle N_u, S_v \rangle = -\langle N_v, S_u \rangle, \\ g &= \langle N, S_{vv} \rangle = -\langle N_v, S_v \rangle. \end{aligned} \tag{9.16}$$

Andre fundamentale form,representert som en symmetrisk matrise er

$$II_p(\mu) = \mu^T \begin{pmatrix} e & f \\ f & g \end{pmatrix} \mu.$$

hvor $\mu \in T_q(S)$, dvs. er i tangentplanet til S i punktet $q = S(p)$.

Følgende definisjoner er relatert til krumning:

- Gauss-krumningen $K = det(dN_q)$ av S, dvs. $K = \frac{det(II)}{det(I)}$.
- Maksimum normalkrumning k_1 og minimum normalkrumning k_2 kalles hovedkrumningene til S ved punktet q. Det følger at $dN_q(e_1) = k_1 e_1$ og $dN_q(e_2) = k_2 e_2$. e_1 og e_2 er ortogonale til hverandre, de kalles hovedretninger og er egenvektorer til dN_q. Dermed er k_1 og k_2 egenverdiene til dN_q.
- Middelkrumningen til S i punktet q er $H = \frac{k_1+k_2}{2}$.
- Det følger at gauss-krumningen $K = k_1 k_2$.

Ved utregninger (se [49], side 154-156) følger følgende formler:

$$
\begin{aligned}
K &= \frac{eg - f^2}{EG - F^2} \\
H &= \frac{1}{2}\frac{eG - 2fF + gE}{EG - F^2} \\
k &= H \pm \sqrt{H^2 - K} \\
dN &= \frac{-1}{EG-F^2}\begin{pmatrix} e & f \\ f & g \end{pmatrix}\begin{pmatrix} G & -F \\ -F & E \end{pmatrix} = \frac{-1}{EG-F^2}\begin{pmatrix} fF - eG & eF - fE \\ gF - fG & fF - gE \end{pmatrix}.
\end{aligned}
$$

9.2 Rotasjonsflater

På engelsk kalles de “surface of revolution" - dreide overflater. Flatetypen inkluderer de fleste av de klassiske flatene som kjegle, sylinder, kule og torus. Grunnkonstruksjonen er:

- Vi starter med en kurve $c(u)$ i $\mathbb{R}^2$,

$$
c(u) = \begin{pmatrix} c_x(u) \\ c_y(u) \end{pmatrix} \quad \text{med domenet} \quad \begin{cases} u \in [s,e], & \text{hvis åpen,} \\ u \in [s,e), & \text{hvis lukket.} \end{cases} \tag{9.17}
$$

- Så må vi bestemme aksen som kurven skal rotere rundt. Velger vi z-aksen får vi

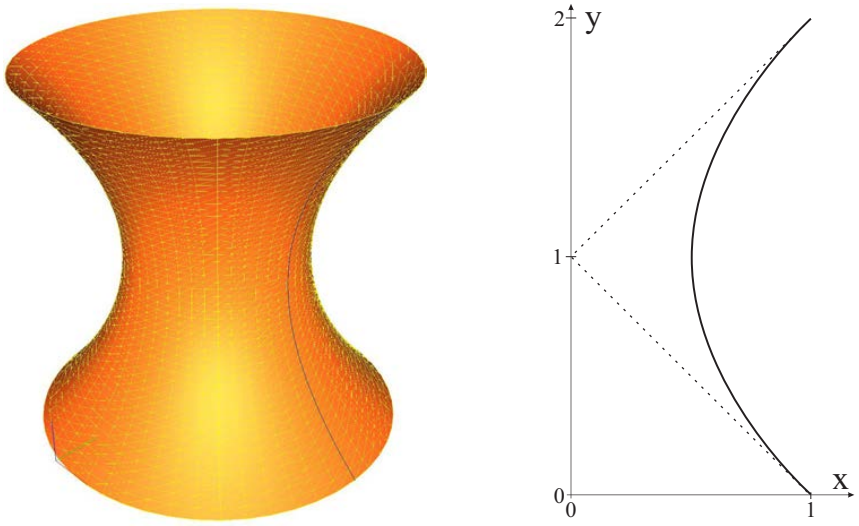
$$
S(u,v) = \begin{pmatrix} c_x(u)\cos v \\ c_x(u)\sin v \\ c_y(u) \end{pmatrix} \tag{9.18}
$$

- Til slutt velger vi domene. I u-retning må vi bruke domenet til kurven $c(u)$, men i v-retning må det være $[a,b]$ hvis $0 < b-a < 2\pi$ (ikke fullstendig rotasjon) eller $[a,b)$ når $b-a = 2\pi$ (fullstendig rotasjon).

Vi har allerede sett to rotasjonsflater, en sylinder, formel (9.2), og en torus, formel (9.3. Kurvene (9.17) for sylinderen $\hat{c}(u)$ og torusen $\bar{c}(u)$ er

$$
\hat{c}(u) = \begin{pmatrix} 1 \\ u \end{pmatrix} \quad u \in [0,5] \quad \text{og} \quad \bar{c}(u) = \begin{pmatrix} \cos u \\ \sin u \end{pmatrix} + \begin{pmatrix} 3 \\ 0 \end{pmatrix} \quad u \in [0, 2\pi),
$$

og formlene til begge, (9.2) og (9.3), stemmer når $\hat{c}(u)$ og $\bar{c}(u)$ erstatter (c_x, c_y) i (9.18).



Figur 9.4: *Et eksempel på en retasjonsflate, kurven til høyre roteres* 360° *rundt sin y-akse for å få flaten vist på venstre side.*

Et annet eksempel er vist i figur 9.4. I figuren er kurven $c(u)$ vist på høyre side. Kurven er en andregrads bézierkurve og vi kan se kontrollpolygonet og dermed koeffisientene (kontrollpunktene) i figuren. Det følger at kurven er

$$c(u) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}(1-u)^2 + \begin{pmatrix} 0 \\ 1 \end{pmatrix} 2u(1-u) + \begin{pmatrix} 1 \\ 2 \end{pmatrix} u^2 = \begin{pmatrix} 1-2u+2u^2 \\ 2u \end{pmatrix}, \quad u \in [0,1],$$

og formelen til flaten blir

$$S(u,v) = \begin{pmatrix} (1-2u+2u^2)\cos v \\ (1-2u+2u^2)\sin v \\ 2u \end{pmatrix} \quad u \in [0,1], \quad v \in [0,2\pi).$$

9.3 Kurvesveipte flater

På engelsk “surface by sweeping". Tenk deg to kurver. Den ene kurven, $g(u)$ - kalt tverrsnitt eller profilkurve, sveiper langs den andre kurven, $h(v)$ - kalt “ryggraden", dvs.

$$s(u,v) = h(v) + A(v)\left(g(u) - h(v_0)\right), \tag{9.19}$$

hvor v_0 startparameterverdien i v-retningen og $A(v)$ er en orthonormal 3×3 matrise,

$$A(v) = \begin{bmatrix} t(v) & f_2(v) & f_3(v) \end{bmatrix}, \tag{9.20}$$

hvor $t(v)$ er enhetstangensvektoren til “ryggradskurven" h, dvs.

$$t(v) = \frac{h'(v)}{|h'(v)|}, \tag{9.21}$$

og hvor $f_2(v)$ og $f_3(v)$ er enhetsvektorer normale til hverandre og til $t(v)$.

Matrisen $A(v)$ er en rotasjonsmatrise som beskriver hvordan man roterer profilkurven når vi beveger oss langs ryggraden. Vi ønsker imidlertid at profilkurven skal holde orienteringen i forhold til ryggradskurven. Derfor bør den første kolonnevektoren være tangentvektoren til ryggradskurven. Det er imidlertid flere valg for rotasjon rundt tangentvektoren, og vi skal se nærmere på to forskjellige valg.

Frenet-ramme, også kalt TNB-rammen og er omtalt i seksjon 4.1.1, er ett valg. Her navngir vi kolonnen i matrisen med

$$A(v) = [T \quad N \quad B]. \tag{9.22}$$

hvor

$$T = t(v) = \frac{h'}{|h'|} \quad \text{og} \quad B = T \wedge N = \frac{h' \wedge h''}{|h' \wedge h''|} \quad \text{og} \quad N = \frac{T'}{|T'|} = B \wedge T.$$

De partiellderiverte til en kurvesveipt flat er

$$S_u = A(v)g'(u) \quad \text{og} \quad S_v = h'(v) + A'(v)(g(u) - h(v_0)). \tag{9.23}$$

Når vi bruker Frenet-rammen må vi bruke Frenet-Serret-formler for å finne $A'(v)$, dvs.

$$\frac{d}{dv}[T \quad N \quad B] = |h'| \; [T \quad N \quad B] \begin{bmatrix} 0 & -\kappa & 0 \\ \kappa & 0 & -\tau \\ 0 & \tau & 0 \end{bmatrix} = |h'| \; [\kappa N \quad \tau B - \kappa T \quad -\tau N].$$

der κ er krumningen til ryggraden h og τ er torsjonen, og der formlene for å beregne krumningen og torsjonen er

$$\kappa = \frac{|h' \wedge h''|}{|h'|^3}, \quad \tau = \frac{\langle (h' \wedge h''), h''' \rangle}{|h' \wedge h''|^2}, \quad \text{se (4.8).}$$

Forøvrig kan bevis for formlene knyttet til Frenet-rammer finnes fra side 130 i [80].

Det er et problem knyttet til å bruke Frenet-rammen. Hvis h'' er null eller er parallell med h' i et punkt på ryggradskurven h, så får vi ikke en orhonormal matrise i punktet, og flata kan brått flippe i punktet. Flaten kan også oppføre seg merkelig hvis h'' er nær null eller er nær parallell med h'.

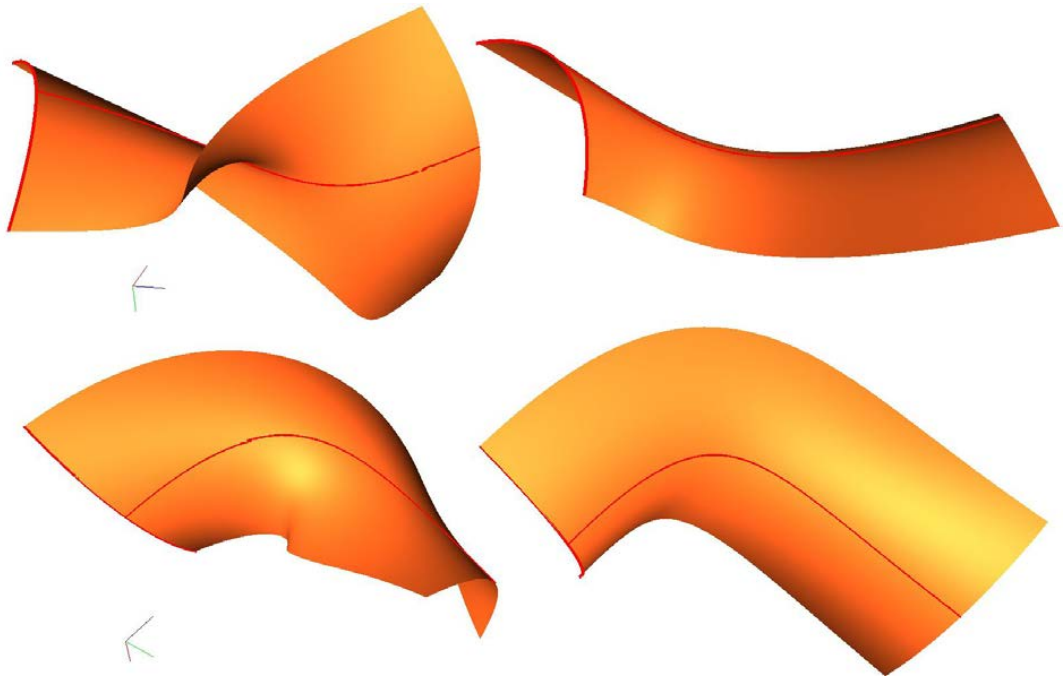
Det er ikke alltid mulig å unngå singulariteter, men det er mulig å gjøre rotasjonen glattere. Et annet valg er derfor å bruke en rotasjonsminimerende ramme - RMF, utviklet og diskutert i [97], [83], [120] og [159].

Sammenlignet med Frenet-rammen (9.22), kan RMF-rammen (9.20) beskrives som

$$\widetilde{A}(v) = A(v) \, R(v). \tag{9.24}$$

der $A(v)$ er definert i (9.22) og den orthonormal matrisen

$$R(v) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\omega & -\sin\omega \\ 0 & \sin\omega & \cos\omega \end{bmatrix}. \tag{9.25}$$



Figur 9.5: *På venstre side er det en* kurvesveipt flate *med TNB-ramme. Flaten vises fra to forskjellige posisjoner. På høyre side er det en* kurvesveipt flate *med RMF-ramme. Flaten vises fra to forskjellige posisjoner. Samme profil- og ryggradskurve er brukt i begge flatene.*

med en vinkel $\omega = \omega(v)$ som bestemmer rotasjonen rundt T-aksen - differansen mellom TNB og RMF. Vinkelen ω kan beregnes ved integralformelen

$$\omega(v) = \omega_0 - \int_{v_0}^{v} \tau(t)|h'(t)|dt, \tag{9.26}$$

med en integrasjonskonstant ω_0, som gir en startvinkel.

For å beregne den deriverte, $\widetilde{A}'(v)$, får vi

$$\widetilde{A}'(v) = A'(v)R(v) + A(v)R'(v). \tag{9.27}$$

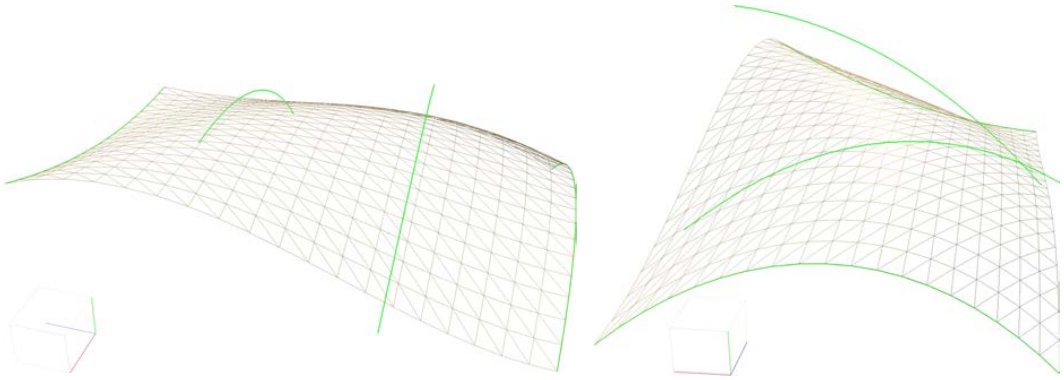
hvor

$$R'(v) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\omega'\sin\omega & -\omega'\cos\omega \\ 0 & \omega'\cos\omega & -\omega'\sin\omega \end{bmatrix}. \tag{9.28}$$

og

$$\omega'(v) = \tau(v)|h'(v)|. \tag{9.29}$$

I figur 9.5 er en profilkurve (kurve langs den ene kanten av flatene) og en ryggradskurve (kurven fra en kant til motsatte kant i midten av flatene, plottet i rødt) gitt. To forskjellige flater er laget av samme profil og ryggradskurve, en på venstre side og en på høyre side. Flaten på venstre side er laget med en TNB-ramme (Frenet-ramme), og flaten på høyre side er laget med en RMF-ramme (rotasjonsminimerende ramme). I dette tilfellet er $\omega_0 = 0$, se (9.26). Begge flatene er vist fra to forskjellige posisjoner for å gi et bedre inntrykk av dem. Flaten til høyre er åpenbart mye glattere enn den andre.



Figur 9.6: *En flate som er vist fra to forskjellige posisjoner. Den er laget ved å blende 4 2.-grads bézierkurver (plottet i grønt) og hvor blendingsfunksjonene er 3.-grads bernsteinpolynomer.*

9.4 Flate fra blending av kurver

Vi har sett på kurvekonstruksjonene, hermite, bézier, B-splines osv. Alle disse konstruksjonene er basert på blending av punkt eller punkt og vektorer, ved bruk av forskjellige typer blendingsfunksjoner. Det er også mulig å lage en flate med samme metode, da med å erstatte punktene/vektorene med kurver/funksjoner.

Følgende generelle konstruksjon kan brukes til flater, lik den generelle kurvekonstruksjonen i(4.12),

$$S(u,v) = \sum_{i=1}^{n} b_i(u)\, c_i(v). \tag{9.30}$$

Her må alle kurver ha et felles domene,

$$c_i(v) : I \subset \mathbb{R}^2 \to \mathbb{R}^3, \quad i = 1,2,...,n,$$

og settet med basisfunksjonene $b_i(u),\ \ i = 1,2,...,n$, må være lineært uavhengige og spenne ut et n-dimensjonalt funksjonsrom. Et eksempel på en flate laget av blending av kurver er gitt i figur 9.6. Merk at det er få grenser for hvilke blendingsfunksjoner som kan brukes, eksempelvis Lagrange, Bernstein, Hermite, B-splines. Valget vil avhenge av hvilke interpolasjons-/approksimasjonsegenskaper du ønsker å bruke i modelleringen.

Differensieringen er rett fram, og de partiellderiverte er

$$S_u = \sum_{i=1}^{n} b_i'(u)\, c_i(v), \quad S_v = \sum_{i=1}^{n} b_i(u)\, c_i'(v), \quad S_{uv} = \sum_{i=1}^{n} b_i'(u)\, c_i'(v).$$

Alternativt kan vi bytte parametrene,

$$\bar{S}(u,v) = \sum_{i=1}^{n} b_i(v) c_i(u). \tag{9.31}$$

I resten av kapitlet vil en strek over flatebetegnelsen til en *flate fra blending av kurver* bety et bytte av parameter mellom kurvene og basisfunksjonene.

9.5 Tensorproduktflate

Navnet tensorprodukt refererer til et produkt av to vektorrom, i dette tilfellet endeligedimensjonale funksjonsrom.

Vi vet fra seksjon 4.2 hvordan et funksjonsrom bestemmes av et sett med basisfunksjoner. I seksjon 4.2 var det første eksemplet $\mathscr{P}_d(I)$, dvs. rommet til polynomfunksjoner av grad d på domenet $I \subset \mathbb{R}$. I det følgende er tensorproduktet til to polynombaserte funksjonsrom temaet. Fra avsnitt 4.2 vet vi at $\mathscr{P}_d(I)$ for en gitt d og I kan uttrykkes ved forskjellige sett med basisfunksjoner, dvs.

- Monomiale basisfunksjoner $\{t^i\}_{i=0}^d$, der d, polynomgraden er endelig
- Hermitebasisfunksjoner $\{H_i\}_{i=0}^d$, spesielt for gradene $d = 3$ eller $d = 5$, se seksjon 4.3
- Lagrange-interpolasjonsfunksjoner $\{L_{d,i}\}_{i=0}^d$, hvor graden d er endelig, se seksjon 5.3
- Bernsteinpolynomer $\{b_{i,d}\}_{i=0}^d$, hvor graden d er endelig, se seksjon 4.4.1
- B-splines $\{b(t,t_i,\ldots,t_{i+d+1}) = b_{d,i}\}_{i=0}^n$, der d er graden, n antallet, se seksjon 6.2

For hermitebasisfunksjonene og settet av bernsteinpolynomer, er domenet $I = [0,1]$.

Et tensorprodukt er egentlig et ytreprodukt mellom to vektorer. For eksempel, gitt to vektorer av monomiale basisfunksjoner, en 2.-grads, $\mathbf{m}(u) = (1,u,u^2)$, og en 3.-grads, $\mathbf{m}(v) = (1,v,v^2,v^3)$. Tensorproduktet vil da være en matrise av basisfunksjonene,

$$M(u,v) = \mathbf{m}(u)^T\ (\mathbf{m}(v) = \begin{pmatrix} 1 \\ u \\ u^2 \end{pmatrix} \begin{pmatrix} 1 & v & v^2 & v^3 \end{pmatrix} = \begin{bmatrix} 1 & v & v^2 & v^3 \\ u & uv & uv^2 & uv^3 \\ u^2 & u^2v & u^2v^2 & u^2v^3 \end{bmatrix}.$$

Til hver av disse 12 basisfunksjonene tildeles en koeffisient $a_{i,j}$ og resultatet (funksjonen) blir som følgende,

$$S(u,v) = \begin{pmatrix} 1 & u & u^2 \end{pmatrix} \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} \begin{pmatrix} 1 \\ v \\ v^2 \\ v^3 \end{pmatrix}$$

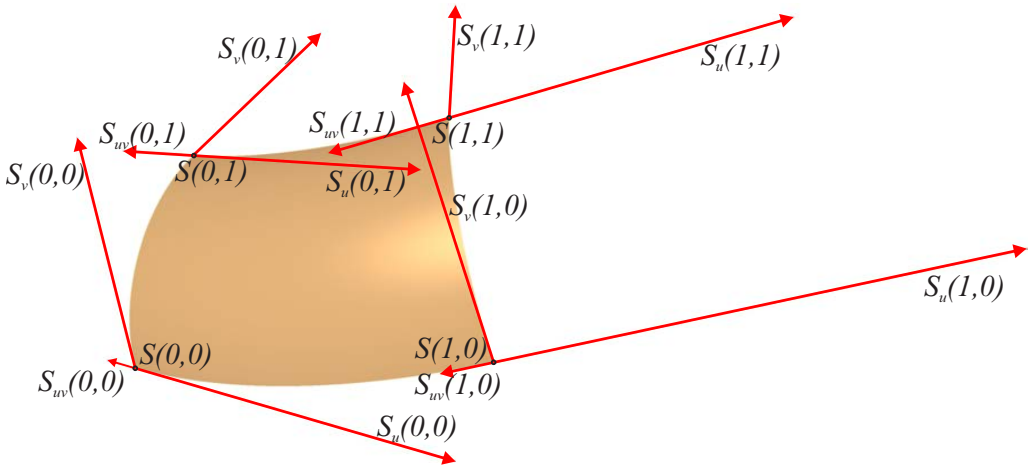
En flate er som oftest i $\mathbb{R}^3$, som betyr at koeffisientene er elementer i $\mathbb{R}^3$, for eksempel punkt eller vektorer. En generell formel for en tensorproduktflate basert på polynomer er

$$S(u,v) = \sum_{i=0}^{d_u} \sum_{j=0}^{d_v} a_{i,j}\, b_j(v)\, b_i(u), \quad u \in I_u,\ v \in I_v, \tag{9.32}$$

der d_u er polynomgraden og I_u er parameterdomenet i u-retning, d_v er polynomgraden og I_v er parameterdomenet i v-retning, og der $b_i(u)$ og $b_j(v)$ er to polynombaserte basissett. Ved implementering kan uttrykket deles inn i to kurvelignende uttrykk, jamfør (9.30),

$$S(u,v) = \sum_{i=0}^{d_u} c_i(v) b_i(u), \quad \text{der} \quad c_i(v) = \sum_{j=0}^{d_v} a_{i,j} b_j(v), \quad \text{for} \quad i = 0,1,...,d_v. \tag{9.33}$$

Dette viser at en tensorproduktflate bare er en “flate fra blending av kurver" (seksjon 9.4), og at alle algoritmer for kurver dermed også kan brukes for tensorproduktflater.



Figur 9.7: *Vi ser en 3.-grads hermiteflate $S(u,v)$. I hjørnene er posisjonen S og de partiellderiverte S_u og S_v og den kryssderiverte S_{uv} plottet. Disse 4 punktene og 12 vektorene er i den 4×4 matrisen M i formel (9.34).*

9.5.1 Hermite-tensorproduktflater

I seksjon 4.3 er basisfunksjonene for 3.-grads hermiteinterpolasjon vist i (4.19), ie

$$\mathbf{H}(t) = \begin{pmatrix} H_1(t) \\ H_2(t) \\ H_3(t) \\ H_4(t) \end{pmatrix} = \begin{pmatrix} 1-3t^2+2t^3 \\ 3t^2-2t^3 \\ t-2t^2+t^3 \\ -t^2+t^3 \end{pmatrix}.$$

Det følger at i henhold til (9.32), vil en hermite-tensorproduktflate være som følger:

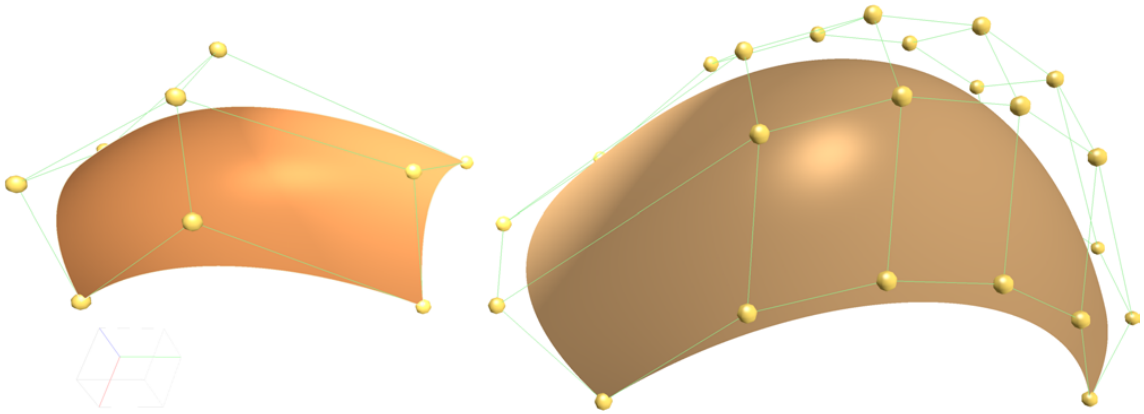
$$s(u,v) = \mathbf{H}(u)^T M\, \mathbf{H}(v) = \sum_{i=1}^{4}\sum_{j=1}^{4} M_{i,j}\, H_j(v)\, H_i(u), \quad u\in[0,1],\ v\in[0,1], \tag{9.34}$$

som ekspandert er

$$s(u,v) = (H_1(u)\ \ H_2(u)\ \ H_3(u)\ \ H_4(u)) \begin{bmatrix} s(0,0) & s(0,1) & s_v(0,0) & s_v(0,1) \\ s(1,0) & s(1,1) & s_v(1,0) & s_v(1,1) \\ s_u(0,0) & s_u(0,1) & s_{uv}(0,0) & s_{uv}(0,1) \\ s_u(1,0) & s_u(1,1) & s_{uv}(1,0) & s_{uv}(1,1) \end{bmatrix} \begin{pmatrix} H_1(v) \\ H_2(v) \\ H_3(v) \\ H_4(v) \end{pmatrix}.$$

Den 4×4 matrisen M er dataene som beskriver formen på flaten. Matrisen kan deles inn i fire 2×2 undermatriser. Øvre venstre undermatrise består av 4 punkt som gir de 4 hjørnene. Nedre venstre undermatrise består av partielle deriverte i u-retningen i de 4 hjørnene. Øvre høyre undermatrise består av de partielle deriverte i v-retningen i de 4 hjørnene og den nedre venstre undermatrisen består av kryssderiverte i de 4 hjørnene.

Figur 9.7 viser et eksempel på en hermiteflate. Den er egentlig en kopi av en del av en torus, (9.3). Parameterintervallet på torusen er $[[0.4,0.9]\times[0.4,1.6]]$ og de 4 partiellderiverte s_u fra torusen skaleres med 0.5, de 4 partiellderiverte s_v skalert med 1.2 og de 4 kryssderiverte s_{uv} skalert med $0,5*1,2=0,6$. Alle deriverte (skalerte) vises som røde piler i figur 9.7.



Figur 9.8: *Vi ser to bézierflater inkludert deres kontrollpunkt og kontrollpolygon. Flaten til venstre er av grad 2 i begge retninger, flaten til høyre er av grad 5 i den ene retningen og grad 3 i den andre retningen.*

9.5.2 Bézier-tensorproduktflater

Vi starter med bézierkurver fra seksjon 4.4. Gitt en grad d og et kontrollpolygon med $d+1$ kontrollpunkt. Hvis vi nå erstatter kontrollpunktene med kurver, dvs. bézierkurver, får vi en bézier-tensorproduktflate,

$$s(u,v) = \sum_{i=0}^{d_u} \sum_{j=0}^{d_v} p_{i,j}\, b_{j,d_v}(v)\, b_{i,d_u}(u), \quad u,\, v \in [0,1], \tag{9.35}$$

Polynomfunksjonsrommet i bézier-tilfellet er av grad d_u i u-parameterretningen og grad d_v i v-parameterretningen. De to funksjonsrommene spennes ut av bernsteinpolynomene $b_{i,d_u}(u),\ i = 0,1,\ldots,d_u$ og av $b_{j,d_v}(v),\ j = 0,1,\ldots,d_v$. Kontrollpunktene $\{p_{i,j}\}$ definerer et $(d_u+1)\times(d_v+1)$ kontollnett, som skisserer flaten.

Vi ser først på et 2.-grads eksempel, som vist på venstre side i figur 9.8 og ser på matriseformuleringen fra (9.35) og får

$$s(u,v) = \begin{pmatrix} b_{0,2}(u) & b_{1,2}(u) & b_{2,2}(u) \end{pmatrix} \begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} \\ C_{1,0} & C_{1,1} & C_{1,2} \\ C_{2,0} & C_{2,1} & C_{2,2} \end{bmatrix} \begin{pmatrix} b_{0,2}(v) \\ b_{1,2}(v) \\ b_{2,2}(v) \end{pmatrix}.$$

Vi henter fram bernstein/hermitematrisen fra seksjonen 4.4.3, samt algoritme 2 for å beregne den. Ved å bruke denne matrisen kan 2.-grads eksemplet fra figur 9.8, og dermed uttrykket ovenfor, utvides til

$$\begin{bmatrix} s & s_u & s_{uu} \\ s_v & s_{uv} & s_{uuv} \\ s_{vv} & s_{uvv} & s_{uuvv} \end{bmatrix} = \begin{bmatrix} (1-u)^2 & 2u(1-u) & u^2 \\ 2u-2 & 2-4u & 2u \\ 2 & -4 & 2 \end{bmatrix} \begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} \\ C_{1,0} & C_{1,1} & C_{1,2} \\ C_{2,0} & C_{2,1} & C_{2,2} \end{bmatrix} \begin{bmatrix} (1-v)^2 & 2v-2 & 2 \\ 2v(1-v) & 2-4v & -4 \\ v^2 & 2v & 2 \end{bmatrix}$$

I figur 9.8 vises to eksempler på bézierflater. Til venstre vises en 2.-grads bézierflate inkludert de 9 kontrollpunktene. Til høyre vises en $5.\times 3.$-grads bézierflate inkludert dens 24 kontrollpunkt.

Figur 9.9: *Vi ser 4 B-splineflater sammen med deres kontrollpunkt og kontrollpolygon. Graden og kontrollpunktene er de samme for alle de fire flatene. Forskjellen er at øvre venstre flate er åpen/klemt i begge retninger, øvre høyre er lukket i en retning, nedre venstre er lukket i andre retning og nedre høyre er lukket i begge retninger.*

9.5.3 B-spline-tensorproduktflater

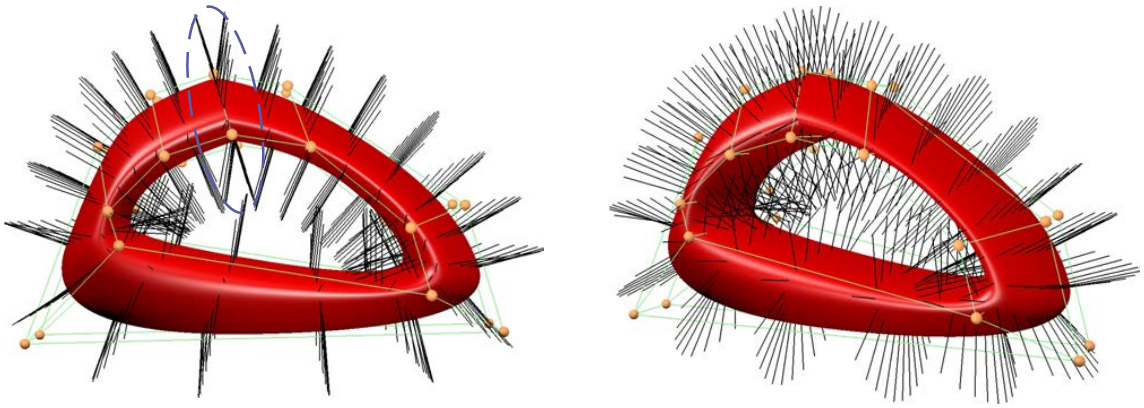
Vi tar utgangspunkt i B-splinekurver fra kapittel 6, Seksjon 6.2. Definisjonen inkluderer en skjøtvektor $\tau = \{t_0, t_1, \ldots, t_{n+d}\}$, en grad d og n kontrollpunkt $c_0, c_1, \ldots, c_{n-1}$. Den generelle formelen, (6.12), ligner formelen for bézierkurver.

Hvis vi, som i bézier-tilfellet, erstatter kontrollpunktene, men nå med B-splinekurver definert over samme spline-rom (dvs. felles skjøtvektor og grad), så får vi en B-spline-tensorproduktflate,

$$s(u,v) = \sum_{i=0}^{n_u-1} \left(\sum_{j=0}^{n_v-1} c_{i,j}\, b_{d_v,j}(v) \right) b_{d_u,i}(u), \tag{9.36}$$

Parametdomenet avhenger av følgende (illustrert i figur 9.9):

åpen i u	$u \in [u_{d_u}, u_{n_u}]$	åpen i v	$v \in [v_{d_v}, v_{n_v}]$	figur 9.9 - oppe til venstre
åpen i u	$u \in [u_{d_u}, u_{n_u}]$	lukket i v	$v \in [v_{d_v}, v_{n_v+d_v})$	figur 9.9 - oppe til høyre
lukket i u	$u \in [u_{d_u}, u_{n_u+d_u})$	åpen i v	$v \in [v_{d_v}, v_{n_v}]$	figur 9.9 - nede til venstre
lukket i u	$u \in [u_{d_u}, u_{n_u+d_u})$	lukket i v	$v \in [v_{d_v}, v_{n_v+d_v})$	figur 9.9 - nede til høyre



Figur 9.10: *Vi ser flaten fra figur 9.9 som er lukket i begge retninger fra to forskjellige posisjoner. Her vises også normalene $N_{u,v}$. På venstre side i figuren ser vi en stiplet blå ellipse, som markerer at det er to normaler i samme punkt for alle punktene langs en gitt konstant v-parameterverdi $v = 2$, dvs. $s(u,2)$.*

Flatene i figur 9.9 har grad 2 i begge retninger. For den åpne flaten er skjøtvektorene $\mathbf{u} = \{0,0,0,1,2,2,2\}$, $\mathbf{v} = \{0,0,0,1,2,2,3,4,4,4\}$ og domenet $[0,2] \times [0,4]$. For den lukkede flater er $\mathbf{u} = \{-2,-1,0,1,2,3,4\}$, $\mathbf{v} = \{-2,-1,0,1,2,2,3,4,5,6\}$ og domenet $[0,4) \times [0,6)$. Kontrollpunktene $c_{i,j},\ i = 0,1,2,3,\ j = 0,1,\ldots,6$ definerer et kontrollnett som skisserer overflaten bedre enn en bézierflaten med samme kontrollnett. I figur 9.9 er det $4 \times 7 = 28$ kontrollpunkt, kobberfargede kuler, forbundet med grønne linjestykker.

For å implementere lukkede B-splineflater er det best å bruke en modifisert versjon av Algoritmen 4 som vi finner i Seksjon 6.2.9, hvor vi reduserer antall deriverte etter behov. Da vil det av praktiske grunner være en fordel å utvide skjøtvektorene ved å legger til d nye skjøter på slutten, hvor avstanden mellom dem reflekterer avstanden mellom skjøtene fra knute t_d til knute t_{d+d}. I figur 9.9 og 9.10 betyr dette $\mathbf{u} = \{-2,-1,0,1,2,3,4,5,6\}$ og $\mathbf{v} = \{-2,-1,0,1,2,2,3,4,5,6,7,8\}$.

Merk at i skjøtvektoren $\mathbf{v}$ fra eksempelet i figur 9.9 er det to like interne skjøtverdier, $v_4 = v_5 = 2$. Siden polynomgraden er 2 i v-retningen, reduseres dermed kontinuiteten med 1 over parameterverdien $v = 2$, jfr. definisjon 6.3 og egenskap **P3** og punkt 5 i listen over egenskaper i seksjon 6.2.2. En 2.-grads B-spline er C^1-glatt over en enkelt skjøtverdi. Det betyr at over to like skjøtverdier vil flaten ha en knekk. Dette er hva figur 9.10 viser. Den nedre høyre flaten i figur 9.9, som er lukket i begge retninger, er også vist i figur 9.10. Men her er også flatenormalene i hvert sampelpunkt plottet. I figur 9.10 ser vi en stiplet blå ellipse som markerer et område der vi kan se to ulike normaler i samme sampelpunkt langs den konstante parameterverdien $v = 2$. Ved nærmere undersøkelser av figuren kan vi registrere at det er en knekk i flaten langs kurven, $s(u,2)$.

På matrisenotasjon er beregningene av B-splineflater som følger,

$$\overline{s}(u,v) = \mathbf{B}_{d,\mathbf{u}}(u,i)\ \mathbf{C}\ \mathbf{B}_{d,\mathbf{v}}(v,j)^T, \quad \text{når} \quad u_i \le u < u_{i+1} \quad \text{og} \quad v_i \le v < v_{i+1},$$

som er B-spline-versjonen av siste uttrykket i seksjon 9.5.2. B-spline/hermitematrisene $\mathbf{B}$ i både u- og v-retning kan beregnes med algoritme 4 på side 97.

9.6 Boolsk sum-flater

Boolsk sum-flater er å lage en flate basert på en "boolsk sum"-metode. Hvis $A(F)$ og $B(F)$ er to operatorer som virker på F. Da er den boolske summen $(A \oplus B)(F)$ definert som unionen av de to settene, som inneholder skjæringssettet, $A(B(F)) = B(A(F)) = AB(F)$, bare én gang. Derfor

$$(A \oplus B)(F) = A(F) + B(F) - AB(F).$$

Tenk deg at randa til en flate, representert ved 4 sammenkoblede kurver, er kjent. For å lage en flate med denne randa kan Coons patch - bilineære blendinger brukes.

Ved modellering av objekt ønsker vi ofte glatte overflater. Hvis flere flater settes sammen, ønsker vi at ikke bare flatene skal henge sammen, men at de deriverte over en kant er like på to tilstøtende flater. "Coons Patch" - bikubiske blending, er da en løsning.

En annen mulighet er at et nett av kurver som beskriver en flate er kjent. Kurvene i den ene retningen skjærer alle kurvene i den andre retningen. En overflate laget av disse kurvene kalles en Gordon-flate (se [79]).

Konstruksjonen er den samme for disse tre blendingstypene og kan tilpasses alle andre typer boolske sum-flater. Den grunnleggende konstruksjonen er:

- Vi lages først en flate $S_1(u,v)$ fra blending kurver. Kurvene er i v-retning, $c(v)$. Settet med basisfunksjoner er $\{b_i\}$.
- Vi lages så en til flate $\bar{S}_2(u,v)$ fra blending kurver. Kurvene er i u-retning, $g(u)$. Settet med basisfunksjoner er $\{\bar{b}_i\}$.
- Til slutt må vi lage en tensorproduktflate, der $\{\bar{b}_i\}$ og $\{b_i\}$ er basisfunksjonene. Koeffisientene avhenger av typen basisfunksjoner. Det kan være posisjon og deriverte i hjørnene, eller et "punktnet". Denne flaten er $S_3(u,v)$.
- En boolsk sum-flate er da definert som,

$$S(u,v) = S_1(u,v) + \bar{S}_2(u,v) - S_3(u,v). \tag{9.37}$$

9.6.1 *Coons patch*, bilineær blending

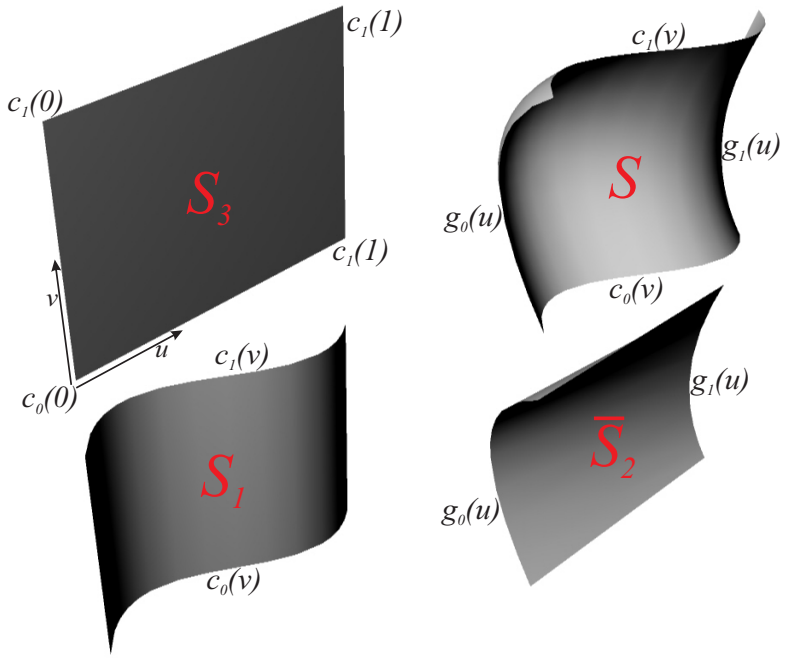
Steven Anson Coons [1] introduserte "Coons patch" i 1964, [26]. "Coons patch" er basert på forutsetningen at en "flatelapp" kan beskrives i form av fire distinkte randkurver. En enkel tilnærming er "Coons patch" bilineær blending. Den lages fra fire randkurver, dvs. to sett. De to settene er (se også figur 9.11)

$$\text{u-settet: } g_0(u) \text{ og } g_1(u),\ u \in [0,1], \quad \text{v-settet: } c_0(v) \text{ og } c_1(v),\ v \in [0,1].$$

Kurvene må henge sammen, og det er derfor en forutsetning at

$$c_0(0) = g_0(0),\ \ c_0(1) = g_1(0),\ \ c_1(0) = g_0(1),\ \ c_1(1) = g_1(1).$$

[1] Steven Anson Coons (1900 – 1979) var professor ved *Massachusetts Institute of Technology* i maskinteknikk. Under andre verdenskrig jobbet han med flater for fly, og utviklet matematikken for generaliserte "flatelapper". Senere (1960) publiserte han arbeider om det som i dag er kjent som "Coons patch".



Figur 9.11: *Et eksempel på "Coons patch". Flaten S er en boolsk sum av de tre flatene* S_1, $\bar{S}_2$ *og* S_3, *dvs.* $S = S_1 + \bar{S}_2 - S_3$.

Vi lager først tre flater ved lineærinterpolasjon i u retning, S_1, i v retning, $\bar{S}_2$ og til slutt en tensorproduktflate S_3, en bilineær interpolasjon som interpolerer hjørnene, dvs.

$$
\begin{aligned}
S_1(u,v) &= \begin{pmatrix} c_0(v) & c_1(v) \end{pmatrix} \begin{pmatrix} 1-u \\ u \end{pmatrix} && \text{blending av kurver,} \\
\bar{S}_2(u,v) &= \begin{pmatrix} 1-v & v \end{pmatrix} \begin{pmatrix} g_0(u) \\ g_1(u) \end{pmatrix} && \text{rotert blending av kurver,} \\
S_3(u,v) &= \begin{pmatrix} 1-v & v \end{pmatrix} \begin{pmatrix} c_0(0) & c_1(0) \\ c_0(1) & c_1(1) \end{pmatrix} \begin{pmatrix} 1-u \\ u \end{pmatrix} && \text{tensorprodukt.}
\end{aligned}
\tag{9.38}
$$

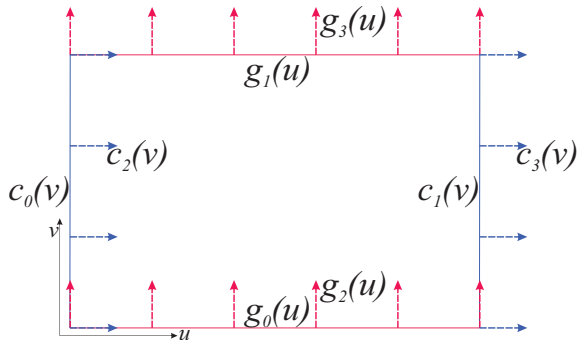
For å lage den endelige flaten S bruker vi den boolske summen av de tre flatene, dvs.

$$S(u,v) = S_1(u,v) + \bar{S}_2(u,v) - S_3(u,v),$$

der S_1, $\bar{S}_2$ og S_3 er definert i (9.38). Prosessen er illustrert i figur 9.11. Som en kontroll ser vi på randa til den resulterende flaten S,

$$
\begin{aligned}
S(u,0) &= S_1(u,0) + \bar{S}_2(u,0) + S_3(u,0) \\
&= (1-u)\, c_0(0) + u\, c_1(0) + g_0(u) - (1-u)\, c_0(0) - u\, c_1(0) = g_0(u), \\
S(u,1) &= S_1(u,1) + \bar{S}_2(u,1) + S_3(u,1) \\
&= (1-u)\, c_0(1) + u\, c_1(1) + g_1(u) - (1-u)\, c_0(1) - u\, c_1(1) = g_1(u), \\
S(0,v) &= S_1(0,v) + \bar{S}_2(0,v) + S_3(0,v) \\
&= c_0(v) + (1-v)\, g_0(0) + v\, g_1(0) - (1-v)\, c_0(0) - v\, c_0(1) = c_0(v), \\
S(1,v) &= S_1(1,v) + \bar{S}_2(1,v) + S_3(1,v) \\
&= c_1(v) + (1-v)\, g_0(1) + v\, g_1(1) - (1-v)\, c_1(0) - v\, c_1(1) = c_1(v),
\end{aligned}
$$

som viser at randa er lik de opprinnelig kurvene.



Figur 9.12: *Vi ser fire randkurver $c_0(v)$, $c_1(v)$, $g_0(u)$ og $g_1(u)$, og fire vektorfunksjoner $c_2(v)$, $c_3(v)$, $g_2(u)$ og $g_3(u)$, som beskriver deriverte "ortogonale" til sine respektive randkurver.*

9.6.2 *Coons patch*, bikubisk blending

Det er også mulig å konstruere "Coons patch" med en høyere grad av kontroll av randa, dvs. hvor vi også kan bestemme den deriverte i v-retning på randa der bare u varierer, og den deriverte i u-retning på randa der bare v varierer. Dette kalles en bikubisk blendet "Coons patch", og den er konstruert på samme måte som den bilineært blendede "Coons patch". Forskjellen er at vi må bruke hermiteinterpolasjon i stedet for lineærinterpolasjon.

Vi starter med 4 randkurver, to sett av to kurver,

$$\begin{aligned} c_0(v) \quad &\text{og} \quad c_1(v), \quad v \in [0,1], \\ g_0(u) \quad &\text{og} \quad g_1(u), \quad u \in [0,1]. \end{aligned}$$

Så må vi ha to vektorfunksjoner som gir de deriverte i u retning langs rendene i v retning, og de to som gir de deriverte i v-retning langs rendene i u-retning,

$$\begin{aligned} c_2(v) \quad &\text{og} \quad c_3(v), \quad v \in [0,1], \\ g_2(u) \quad &\text{og} \quad g_3(u), \quad u \in [0,1]. \end{aligned}$$

Kurvene må samsvare med hverandre, derfor er det et krav at

$$\begin{aligned} c_0(0) &= g_0(0), & c_0(1) &= g_1(0), \\ c_1(0) &= g_0(1), & c_1(1) &= g_1(1). \\ c_0'(0) &= g_2(0), & c_0'(1) &= g_3(0), \\ c_1'(0) &= g_2(1), & c_1'(1) &= g_3(1), \\ g_0'(0) &= c_2(0), & g_0'(1) &= c_3(0), \\ g_1'(0) &= c_2(1), & g_1'(1) &= c_3(1). \end{aligned}$$

Sammenhengen mellom kurvene og vektorfunksjonene er også vist i figur 9.12. Før vi starter å bygge "Coons patch" bikubisk blending, definerer vi vektorene

$$\begin{aligned} \mathbf{H}(t) &= [\ H_1(t), \quad H_2(t), \quad H_3(t), \quad H_4(t)\ ], \\ \mathbf{c}(t) &= [\ c_1(t), \quad c_2(t), \quad c_3(t), \quad c_4(t)\ ], \\ \mathbf{g}(t) &= [\ g_1(t), \quad g_2(t), \quad g_3(t), \quad g_4(t)\ ]. \end{aligned}$$

Vi må så lage en flate med å bruke hermiteinterpolering av to kurver og to vektorfunksjoner i u retning - S_1, og en tilsvarende flate i v-retning - $\bar{S}_2$, pluss en hermite-tensorproduktflate der posisjonen, de partiellderiverte med hensyn på u og v og de kryssetderiverte i alle fire hjørner må brukes - S_3. Dvs.

$$\begin{aligned} S_1(u,v) &= \langle \mathbf{c}(v),\, \mathbf{H}(u) \rangle && \text{blending av kurve,} \\ \bar{S}_2(u,v) &= \langle \mathbf{H}(v),\, \mathbf{g}(u) \rangle && \text{rotert blending av kurve,} \\ S_3(u,v) &= \mathbf{H}(v)\, \mathbf{M}\, \mathbf{H}(u)^T && \text{tensorprodukt,} \end{aligned} \tag{9.39}$$

hvor matrisen $\mathbf{M}$ er

$$\mathbf{M} = \begin{pmatrix} c_0(0) & c_1(0) & c_2(0) & c_3(0) \\ c_0(1) & c_1(1) & c_2(1) & c_3(1) \\ c_0'(0) & c_1'(0) & a_{11} & a_{12} \\ c_0'(1) & c_1'(1) & a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} g_0(0) & g_0(1) & g_0'(0) & g_0'(1) \\ g_1(0) & g_1(1) & g_1'(0) & g_1'(1) \\ g_2(0) & g_2(1) & a_{11} & a_{12} \\ g_3(0) & g_3(1) & a_{21} & a_{22} \end{pmatrix}.$$

De fire tallene nederst til høyre er de kryssderiverte i de fire hjørnene. Vi skal senere vise hva disse fire verdiene må være dersom kravene i kantene skal oppfylles, men mønsteret i radene (og kolonnene) indikerer også hva disse verdiene må være. For å lage den endelige flaten S bruker vi en boolsk sum av (9.39), dvs.

$$S(u,v) = S_1(u,v) + \bar{S}_2(u,v) - S_3(u,v),$$

Vi gjentar de spesielle hermitebasisegenskapen beskrevet i (4.20),

$$\mathbf{H}(0) = [1,\, 0,\, 0,\, 0], \quad \mathbf{H}(1) = [0,\, 1,\, 0,\, 0], \quad \mathbf{H}'(0) = [0,\, 0,\, 1,\, 0], \quad \mathbf{H}'(1) = [0,\, 0,\, 0,\, 1].$$

Vi bruker så dette for å sjekke rendene til flaten,

$$\begin{aligned} S(u,0) &= S_1(u,0) + \bar{S}_2(u,0) - S_3(u,0) = \langle \mathbf{c}(0),\, \mathbf{H}(u) \rangle + g_0(u) - \langle \mathbf{c}(0),\, \mathbf{H}(u) \rangle = g_0(u), \\ S(u,1) &= S_1(u,1) + \bar{S}_2(u,1) - S_3(u,1) = \langle \mathbf{c}(1),\, \mathbf{H}(u) \rangle + g_1(u) - \langle \mathbf{c}(1),\, \mathbf{H}(u) \rangle = g_1(u), \\ S(0,v) &= S_1(0,v) + \bar{S}_2(0,v) - S_3(0,v) = c_0(v) + \langle \mathbf{H}(v),\, \mathbf{g}(0) \rangle - \langle \mathbf{H}(v),\, \mathbf{g}(0) \rangle = c_0(v), \\ S(1,v) &= S_1(1,v) + \bar{S}_2(1,v) - S_3(1,v) = c_1(v) + \langle \mathbf{H}(v),\, \mathbf{g}(1) \rangle - \langle \mathbf{H}(v),\, \mathbf{g}(1) \rangle = c_1(v), \end{aligned}$$

som viser at rendene er som forventet. Den deriverte i den “ortogonale" retning på de fire kantene må være lik de fire vektorfunksjonene, dermed må

$$S_v(u,0) = \langle \mathbf{c}'(0),\, \mathbf{H}(u) \rangle + g_2(u) - \langle (g_2(0),\, g_2(1),\, a_{11},\, a_{12}),\, \mathbf{H}(u) \rangle = g_2(u),$$

som krever at $a_{11} = c_2'(0)$ og $a_{12} = c_3'(0)$. Videre er

$$S_v(u,1) = \langle \mathbf{c}'(1),\, \mathbf{H}(u) \rangle + g_3(u) - \langle (g_3(0),\, g_3(1),\, a_{21},\, a_{22}),\, \mathbf{H}(u) \rangle = g_3(u),$$

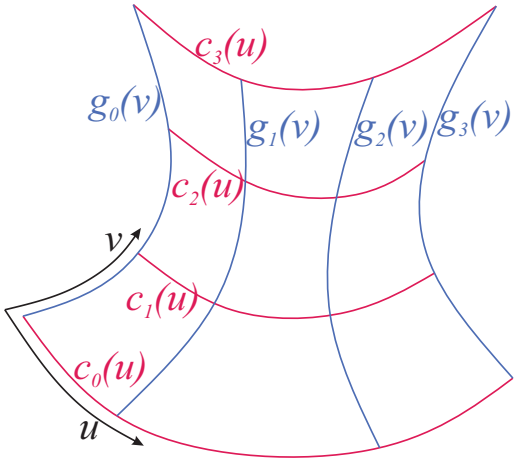
som krever at $a_{21} = c_2'(1)$ og $a_{22} = c_3'(1)$. Videre er

$$S_u(0,v) = c_2(v) + \langle \mathbf{H}(v),\, \mathbf{g}'(0) \rangle - \langle \mathbf{H}(v),\, (c_2(0),\, c_2(1),\, a_{11},\, a_{21}) \rangle = c_2(v),$$

som krever at $a_{11} = g_2'(0)$ og $a_{21} = g_3'(0)$. Videre er

$$S_u(1,v) = c_3(v) + \langle \mathbf{H}(v),\, \mathbf{g}'(1) \rangle - \langle \mathbf{H}(v),\, (c_3(0),\, c_3(1),\, a_{12},\, a_{22}) \rangle = c_3(v),$$

som krever at $a_{12} = g_2'(1)$ og $a_{22} = g_3'(1)$.



Figur 9.13: *Vi ser et nett av kurver, de røde i u-retningen og de blå i v-retningen. Hver rød kurve skjærer alle blå kurver og omvendt. En Gordon-flate interpolerer alle disse kurvene totalt (de ligger i flaten).*

For at den endelige flaten skal oppfylle kravene til at de fire randkurvene og at de "ortogonale" deriverte til de fire rendene skal være like de gitte randkurvene og vektorfunksjonene, må matrisen $\mathbf{M}$ være,

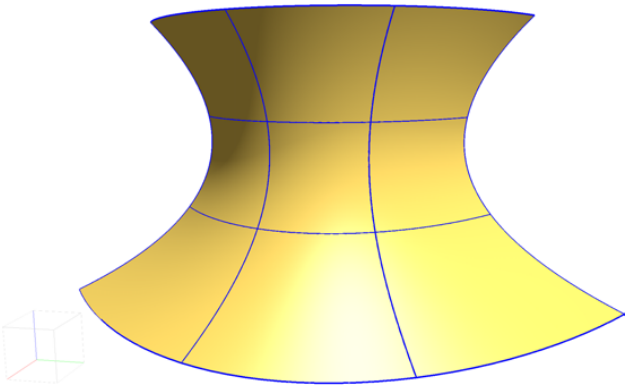
$$\mathbf{M} = \begin{pmatrix} c_0(0) & c_1(0) & c_2(0) & c_3(0) \\ c_0(1) & c_1(1) & c_2(1) & c_3(1) \\ c_0'(0) & c_1'(0) & c_2'(0) & c_3'(0) \\ c_0'(1) & c_1'(1) & c_2'(1) & c_3'(1) \end{pmatrix} = \begin{pmatrix} g_0(0) & g_0(1) & g_0'(0) & g_0'(1) \\ g_1(0) & g_1(1) & g_1'(0) & g_1'(1) \\ g_2(0) & g_2(1) & g_2'(0) & g_2'(1) \\ g_3(0) & g_3(1) & g_3'(0) & g_3'(1) \end{pmatrix}. \quad (9.40)$$

9.6.3 Gordon-flate

Gordon-flater ble utviklet på slutten av 1960-tallet av W. Gordon, [79], som den gang jobbet for *General Motors* forskningslaboratorier. Han kalte metoden for en "transfinitt interpolasjon".

Metoden er i utgangspunktet lik metoden for "Coons patch" da det er en boolsk sum-operasjon. Forskjellen er at hermiteinterpolasjon erstattes med en ordinær interpolasjon, dvs. at basisfunksjonene her er lagrangepolynomer. Inndataene til en Gordon-flate kan i utgangspunktet være et regulært nett, dvs. en matrise av punkt. Vi lager så kurver som interpolerer alle punktene i sine respektive rader og kolonner. Dette kan gjøres med Lagrang-polynomer, seksjon 5.3. Vi har da et nett av kurver, ett sett med kurver $c_i(u)$ i u-parameterretningen, og et annet sett med kurver $g_j(v)$ i v-parameterretningen som vi kan se illustrert i figur 9.13. Hver kurve i det første settet $c_i(u)$ skjærer da alle kurvene i det andre settet $g_j(v)$ og omvendt.

Vi skal her bruke interpolasjon med Langrange-polynomene, formel (5.9) i seksjon 5.3, dvs. $L_{d,i}(t) = \prod_{j=0, j\neq i}^{d} \frac{(t-t_j)}{(t_i-t_j)}$, for $i = 0, 1, \ldots, d$, der d er polynomgraden. Hele settet med lagrangepolynomer er da $\mathbf{L}_d(t) = \begin{pmatrix} L_{d,0}(t) & L_{d,1}(t) & \ldots & L_{d,d}(t) \end{pmatrix}$.



Figur 9.14: *Vi ser en Gordon-flate som interpolerer et nett av* 4×4 *kurver (i blått).*

Inndata til en algoritme er et sett med $d_v + 1$ kurver $\{g_i(u)\}_{i=0}^{d_v}$ i u-retningen. De må ha et felles domene I_u, og det må være et sett med $d_u + 1$ parameterverdier $u_i \in I_u$ der dette settet med kurver skjærer det andre settet. På samme måte må det være et sett med $d_u + 1$-kurver $\{c_i(v)\}_{i=0}^{d_u}$ i v-retningen med et felles domene I_v, og også et sett med $d_v + 1$ parameterverdier $v_i \in I_v$ der dette andre settet med kurver skjærer det første settet. Begge sett med kurver kan/bør lages med interpolasjon av skjæringspunktene med bruk av lagrangepolynomene. Oppsummert gir dette:

- ✓ Første sett med kurver $\mathbf{g}(u) = \{g_i(u)\}_{i=0}^{d_v}$,
- ✓ andre sett med kurver $\mathbf{c}(v) = \{c_j(v)\}_{j=0}^{d_u}$,
- ✓ der begrensningene er $g_i(u_j) = c_j(v_i)$, for $i = 0, 1, \ldots, d_v$ og $j = 0, 1, \ldots, d_u$,
- ✓ og ved åpne kurver er $[u_0, u_{d_u}] = I_u$, og $[v_0, v_{d_v}] = I_v$.

Delformlene for Gordon-flater er

$$\begin{aligned} S_1(u,v) &= \langle \mathbf{c}(v),\, \mathbf{L}_{d_u}(u) \rangle && \text{blending av kurver,} \\ \bar{S}_2(u,v) &= \langle \mathbf{L}_{d_v}(v),\, \mathbf{g}(u) \rangle && \text{blending av kurver med parameterbytte,} \\ S_3(u,v) &= \mathbf{L}_{d_v}(v)\, \mathbf{M}\, \mathbf{L}_{d_u}(u)^T && \text{tensorproduktflate,} \end{aligned} \tag{9.41}$$

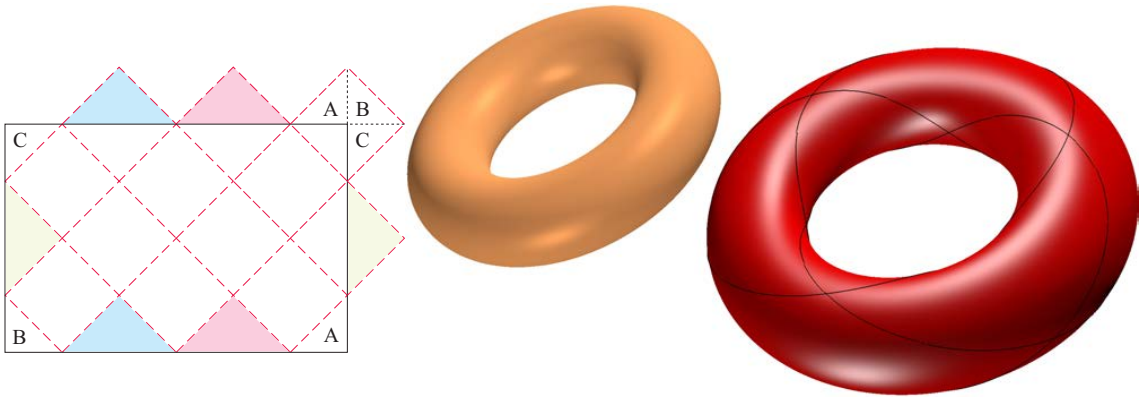
hvor matrisen $\mathbf{M}$ er

$$\mathbf{M} = \begin{bmatrix} c_0(u_0) & c_0(u_1) & \cdots & c_0(u_{d_u}) \\ c_1(u_0) & \ddots & \ddots & c_1(u_{d_u}) \\ \vdots & \ddots & \ddots & \vdots \\ c_{d_v}(u_0) & c_{d_v}(u_1) & \cdots & c_{d_v}(u_{d_u}) \end{bmatrix} = \begin{bmatrix} g_0(v_0) & g_1(v_0) & \cdots & g_{d_u}(v_0) \\ g_0(v_1) & \ddots & \ddots & g_{d_u}(v_1) \\ \vdots & \ddots & \ddots & \vdots \\ g_0(v_{d_v}) & g_1(v_{d_v}) & \cdots & g_{d_u}(v_{d_v}) \end{bmatrix}.$$

Til slutt er den totale formelen for Gordon-flater

$$S(u,v) = S_1(u,v) + \bar{S}_2(u,v) - S_3(u,v).$$

I figur 9.14 ser vi en Gordon-flate laget av 4 kurver i hver parameterretning. Algoritmen startet med 16 punkt organisert i et nett. 4 og 4 kurver ble så laget ved interpolering med lagrangepolynomer, som så ble brukt i flatekonstruksjonen. Parameterdomenet er



Figur 9.15: *Vi ser en "bronsefarget" torus og en approksimasjon av torusen i rødt. Approksimasjonen er laget som 12 "Coons patch" bikubiske blendingsflater. Kantene er tegnet i svart. Til venstre ser vi parameterplanet, som vise hvordan domenet til de 12 flatene er plassert.*

$[0,3] \times [0,3]$. Polynomgraden er 3 i begge retninger. Flaten kan konverteres til en bézier-tensorproduktflate, men da må domenet endres til $[0,1] \times [0,1]$ med å bruke reparametrisering, som er omtalt i seksjon 4.1.2.

9.6.4 Et eksempel med *Coons patch*

"Coons Patch" bikubisk blending er en boolsk sum av tre flater, to flater laget ved å blende kurver med å bruke hermiteinterpolasjon av kurver og vektorfunksjoner, og en hermite-tensorproduktflate.

Vi skal her se på et eksempel på å approksimasjon av en torus med 12 "Coons patch" bikubiske blendingsflater ved å bruke kurver på flater (torusen). Til venstre i figur 9.15 vises parameterdomenet til en torus, $\Omega = [0, 2\pi) \times [0, 2\pi)$. Domenet er delt inn i 12 deler som illustrert i figuren. Hver del er en rotert firkant. For å vise at alle deler er firkanter, er deler av noen av dem tilsynelatende utenfor domenet, de er merket enten rødt, blått, grønt eller med bokstavene A, B eller C. Alle disse delene er også merket inne i domenet med tilsvarende farge eller bokstav.

I parameterdomenet $\Omega \subset \mathbb{R}^2$ har vi 17 punkt, $p_i, i = 0, 1, \ldots, 16$ som alle er i hjørnene av rutene. Punktene er organisert regelmessig slik at avstanden mellom dem er $\frac{2}{3}\pi$ i u-retningen og π i v-retningen. Det er totalt 24 kanter, linjer mellom to punkt, $\sigma_i(t) = (1-t)p_r + tp_s$ og hvor $\sigma_i'(t) = p_s - p_r$, dvs. $\left(\frac{\pi}{3}\ \frac{\pi}{2}\right)$ for linjene som går opp til høyre, og $\left(-\frac{\pi}{3}\ \frac{\pi}{2}\right)$ for linjene som går opp til venstre. Vi navngir torusen for $\theta(p), p = (u,v)$. Kurvene (kantene) i $\mathbb{R}^3$ kaller vi

$$c_i(t) = \theta \circ \sigma_i(t),$$

og vi betegner derivertefunksjonene "ortogonale" til kantene for

$$g_i(t) = [\theta_u\ \theta_v]_{\sigma_i(t)}\, \sigma_i'(t) = \frac{\pi}{2}\theta_v \circ \sigma_i(t) \pm \frac{\pi}{3}\theta_u \circ \sigma_i(t).$$

vi får da følgende flater ved å blending av kurvene og vektorfunksjonene

$$S_1(u,v) = H_1(u)\ c_i(v) + H_2(u)\ c_j(v) + H_3(u)\ g_i(v) + H_4(u)\ g_j(v),$$
$$\overline{S}_2(u,v) = H_1(v)\ c_r(u) + H_2(v)\ c_s(u) + H_3(v)\ g_r(u) + H_4(u)\ g_s(v).$$

For tensorproduktflaten trenger vi, i hvert hjørne, posisjonen, de to partiellderiverte og kryssderiverte. Vi lager så vektorer i parameterplanet som samsvarer med kantene i de små roterte kvadratene vi ser til venstre i figur 9.15. La oss kalle disse retningsvektorene i parameterplanet for $a = \left(\frac{\pi}{3}\ \ \frac{\pi}{2}\right)$ og $b = \left(-\frac{\pi}{3}\ \ \frac{\pi}{2}\right)$. Dermed får vi i hvert punkt p_i - posisjonen, de to partielle deriverte og kryssderiverte som følger:

$$\begin{aligned}
&\theta(p_i)\\
&\theta_a(p_i) = [\theta_u\ \theta_v]_{p_i} a = \frac{\pi}{2}\theta_v(p_i) + \frac{\pi}{3}\theta_u(p_i),\\
&\theta_b(p_i) = [\theta_u\ \theta_v]_{p_i} b = \frac{\pi}{2}\theta_v(p_i) - \frac{\pi}{3}\theta_u(p_i),\\
&\theta_{ab}(p_i) = \left[[\theta_{uu}\ \theta_{uv}]_{p_i} a\ \ [\theta_{vu}\ \theta_{vv}]_{p_i} a\right] b = \frac{\pi^2}{4}\theta_{vv}(p_i) + \frac{\pi^2}{9}\theta_{uu}(p_i).
\end{aligned}$$

Nå, for de fire hjørnene av hver rute, legger vi disse verdiene inn i matrisen **M** i (9.40).

I figur 9.15 ser vi, i midten, en “bronsefarget" torus laget fra formel (9.3) og til høyre et sett med 12 røde “Coons Patch" bikubiske blendingsflater som approksimerer torusen. kurvene mellom flatene er også plottet, i svart. Ved hjelp av testing finner vi at det største avviket fra torus er omtrent 0,0003, som er veldig lite siden rør-radiusen er 1.

Kapittel 10

Subdivisjonsflater

Subdivisjonsflater regnes egentlig ikke som parametriske flater selv om de er basert på B-splines, jamfør subdivisjonskurver beskrevet i avsnitt 6.7. Parametriske flater har formler/algoritmer for beregning av posisjon og flatenormaler for gitte parameterverdier. For å plottes må flaten “tesselleres". Den må deles opp i et sett av koblede små polygoner/trekanter. Hjørnene (verteks) i disse små flatene er sampelpunkt, vanligvis laget ved å “evaluere", dvs. beregne posisjon og normaler med utgangspunkt i parameterverdier i et regulært grid i parameterplanet. Men B-splineflater kan også tesseleres med skjøtinsetting.

Subdivisjonsflater kan beskrives som uniforme heltallsskjøtverdier, skjøtinsetting og reparametrisering til uniforme implesitte heltallsskjøtverdier igjen. De ble først beskrevet i 1978 av Edwin Catmull og Jim Clark, Catmull-Clark subdivisjonsflater [19], og samtidig av Daniel Doo og Malcom Sabin, Doo-Sabin subdivisjonsflate [55]. Den første er basert på bi-kubiske uniforme B-splines og den andre på bi-kvadratiske uniforme B-splines. På grunn av fleksibiliteten, og dermed muligheten til å bruke trekanter, har Box-splines senere blitt mye brukt som utgangspunkt for utvikling av nye skjemaer/algoritmer.

I det følgende skal vi ikke utvikle/bevise subdivisjonsskjemaer, kun beskrive dem. Det samme gjelder analyser av konvergens og kontinuitet. For å være et gyldig skjema må det imidlertid konvergere mot en kontinuerlig flate av type B-splines, alternativt Box-splines, og typisk flater som er C^1 eller C^2-glatte. Vi skal i dette kapittelet bare se på lukkede flater, dvs. flater som er topologisk lik kule, torus eller “toruser" med flere hull. Skjemaer for **vilkårlig** topologi er imidlertid modifikasjoner av splinebaserte skjemaer, og derfor vil det være hjørner/vertekser som kalles ekstraordinære[1]. Disse verteksene endrer regelmessigheten til et nett. Samtidig er det disse punktene som gjør subdivisjonsflater så anvendelige. Ulrich Reif beskrev oppførselen nær ekstraordinære vertekser i 1995 i [135]. I disse punktene er kontinuiteten typisk en orden lavere enn i resten av flaten.

Figur 10.1 viser 6 forskjellige subdivisjonsskjemaer med sjablonger. I figuren er de blå kulene de orginale verteksene, og de røde kulene er vertekser etter én subdivisjon. De røde

[1] Ekstraordinære vertekser kan sammenlignes med “T-kryss/Stjernekryss", beskrevet i seksjon 12.6.2. Noen ganger kalles de uregelmessige/singulære vertekser. Fra grafteori er graden til en verteks antall kanter koblet til verteksen. En ekstraordinær verteks er en som har en grad som er forskjellig fra naboene.

kulene er enten nye eller modifiserte gamle vertekser. Grønne kuler er midlertidige punkt som bare brukes i beregningene. Legg merke til skjemaet for Doo-Sabin og Catmull-Clark. Doo-Sabin er laget av 2.-grads B-splineflater. Figur 6.12 viser skjøtinnsettingen for 2.-grads B-splines. Her er ett kontrollpunkt erstattet med to. Vi ser det samme i Doo-Sabin-skjemaet der ett punkt erstattes med fire (dvs. 2×2). Catmull-Clark er laget av 3.-grads B-splineflater. Figur 6.13 viser skjøtinnsettingen for 3.-grads B-splines. Her er to kontrollpunkt erstattet med tre. Vi ser det samme i Catmull-Clark-skjemaet. Siden subdivisjon er basert på uniforme B-splines kunne Jos Stam gi en metode for eksakt evaluering for Catmull-Clark subdivisjonsflater for vilkårlige parameterverdier i [152].

Skjemaene kan klassifiseres på flere måter. For eksempel kan de klassifiseres som primær eller dual, basert på flate-deling eller verteks-splitting. Klassifiseringene basert på geometriske egenskaper er hovedsakelig:

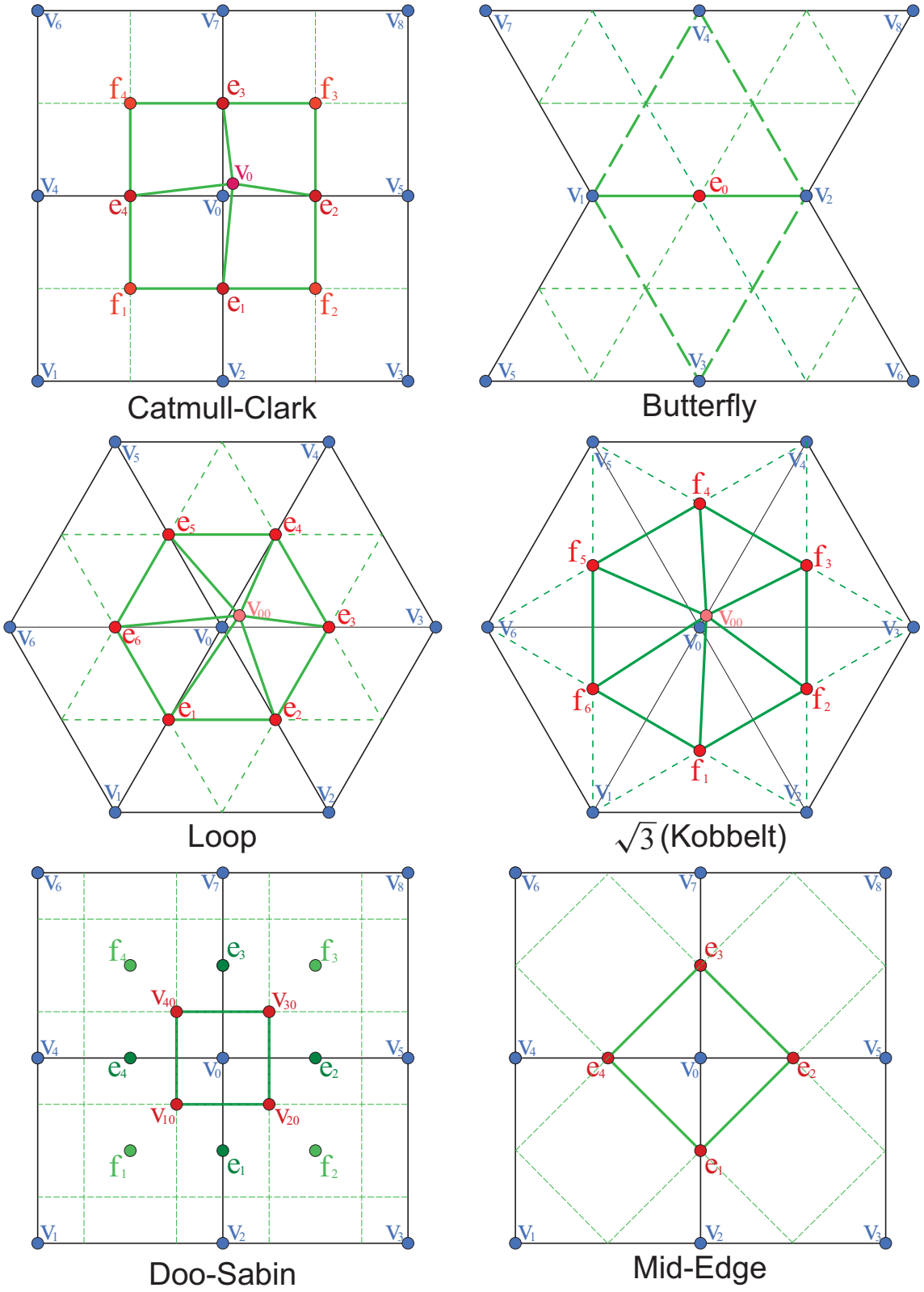
- ✓ om de er hjørnekutting (dvs. approksimative) eller interpoleringsskjemaer,
- ✓ om de er C^1- eller C^2-glatte,
- ✓ om de i hovedsak er basert på trekanter eller på firkanter.

Legg merke til verteksene med grad forskjellig fra de fleste andre, ekstraordinære vertekser som konvergerer mot et punkt der glattheten er 1 mindre. I motsetning til de andre skjemaene finner vi dem ikke i for eksempel Doo-Sabin og Mid-Edge. Etter første subdivisjon i Catmull-Clark er alle polygonene firkanter, og i Loop er alle polygonene trekanter. Etter første subdivisjon i Doo-Sabin og Mid-Edge er graden til verteksene alltid 4, og de fleste polygon er firkanter, men de som ikke er det, vil forbli det de er etter alle senere subdivisjoner, og vil konvergere mot et punkt som har 1 størrelsesorden lavere kontinuitet, akkurat som ekstraordinære vertekser.

Utgangspunktet for en subdivisjonsalgoritme er et sett med punkt som er koblet sammen med et nett som dekker og definerer en underliggende flate. Dataene skal ordnes slik at innsiden og utsiden kan bestemmes lokalt i hvert polygon. Det finnes ferdige datastrukturer for programmering av subdivisjon, eksempelvis OpenMesh [14] (lastes ned fra https://www.graphics.rwth-aachen.de/software/openmesh/), men det er også mulig å bruke en enkel selvlaget datastruktur. For eksempel tabell av tre typer objekt, en verteks (punkt og flatenormal), en flate (“engelsk face" er et polygon, et ordnet sett med verteksindekser organisert mot klokken sett fra utsiden) og en kant som kan være to verteksindekser pluss to flate-indekser. NB! Ved implementering er det viktig, og også mulig, å lage alle algoritme slik at de er av $\mathcal{O}(n)$. Hvis noen rutiner er av $\mathcal{O}(n \log n)$ eller $\mathcal{O}(n^2)$ vil subdivisjonen gå svært sakte når antall vertekser vokser.

10.1 En samling av subdivisjonsskjema

Figur 10.2 viser 6 subdivisjonsflater, alle laget fra samme boks i $\mathbb{R}^3$ og vist som blå linjer i figuren. Ved firkantelementer er det 8 vertekser, 6 flater og 12 kanter, og for trekanter 8 vertekser, 12 flater og 18 kanter. Sjablonger av subdivisjonsskjemaer for disse flatene er vist i figur 10.1. I det følgende skal vi se nærmere på disse skjemaene. Husk at skjemaer alltid er affine kombinasjoner av punkt, dvs. bruk av vekter som summerer opp til 1.



Figur 10.1: *Sjablonger med 6 forskjellige subdivisjonsskjemaer for flater. Øverst til venstre Catmull-Clark, basert på kubiske B-splines. Til høyre Butterfly, et interpolasjonsskjema basert på trekanter. I midten Loop og $\sqrt{3}$-Kobbelt, begge basert på trekanter og box-splines. Nederst Mid-Edge og Doo-Sabin, begge basert på 2.-grads B-splines.*

10.1.1 Catmull-Clark

Dette er et firkant-basert skjema basert på skjøtinnsetting i bi-kubiske uniforme B-spline-flater, først beskrevet i [19]. For vilkårlige startnett genererer dette skjemaet flater som konvergerer mot C^2-kontinuitet overalt bortsett fra i ekstraordinære vertekser hvor de er C^1. En flate vises øverst i figur 10.2. Øverst til venstre i figur 10.1 ser vi en sjablong av skjemaet. De blå kulene er de gamle verteksene og de røde er de nye som er generert rundt en gammel verteks v_i. Det er 3 typer "nye" vertekser, kant-vertekser, flate-vertekser og korrigering av gamle vertekser. Skjemaet er som følger; for hvert subdivisjonstrinn og for alle flater, kanter og vertekser, lager vi følgende nye **vertekser** (punkt):

Flate-punkt er et punkt i midten av en flate: $f_i = \frac{1}{m}\sum_j v_j$, der m er antall vertekser som definerer flaten. Etter den første subdivisjonen er alltid $m = 4$.

Kant-punkt er et punkt midt på en kant: $e_i = \frac{1}{4}(v_a + v_b + f_h + f_g)$, der v_a og v_b er de gamle verteksene som definerer gjeldende kant, og f_g og f_h er de nye flate-punktene i de to naboflatene som deler gjeldende kant.

Verteks-punkt er oppdatering av en gammel verteks: $v_i = \frac{1}{n}(F + 2E + (n-3)v_i)$, der n er graden til verteksen, $F = \frac{1}{n}\sum_j f_j$ - er gjennomsnittet av alle tilkoblede flate-punkt og $E = \frac{1}{n}\sum_j e_j$ - er gjennomsnittet av alle tilkoblede kant-punkt.

Når alle nye vertekser er laget, angir vi at indeksen i verteksen til det første flate-punktet er m_f og det første kant-punktet til m_e. Da er det en enkel overgang mellom indeksen til en flate og et flate-punkt, og mellom indeksen til en kant og et kant-punkt. Nå kan vi lage en ny tabell av **flater**.

✓ For hver gammel flate F_i (som har n vertekser første gang og 4 senere),
hvor $F_i = \{v_1, v_2, \ldots, v_n\}$, flate-punktene er: $f_i = m_f + i$,
og kant-punktene er $\{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n\} = \{e_1 + m_e,\ e_2 + m_e,\ \ldots,\ e_n + m_e\}$,
vi lager så n nye flater som erstatter F_i, dvs.
$F_{i1} = \{f_i, \varepsilon_n, v_1, \varepsilon_1\}$, $F_{i2} = \{f_i, \varepsilon_1, v_2, \varepsilon_2\}$, $\ldots$, $F_{in} = \{f_i, \varepsilon_{n-1}, v_n, \varepsilon_n\}$.

I algoritmen ovenfor bruker vi indeksene til kantene til en flate, $\{e_1,\ e_2,\ \ldots,\ e_n\}$. De kan lagres i enten en egen tabell eller i hver flate, dvs. en flate er et sett med indekser for vertekser og et tilsvarende sett med indekser for kanter. Hvis det fortsatt er subdivisjonstrinn som skal utføres, må en tabell av **kanter** opprettes, og indeksene til kantene i flatene må settes. Som et hjelpeverktøy lager vi en matrise Ψ med størrelse lik den nye Verteks-tabellen, og hvor hvert element er en liste med indekspar til en verteks og en kant.

✓ For hver ny flate $F_i = \{v_1, v_2, v_3, v_4\}$ (som alltid har 4 vertekser),
vi lager 4 kanter: $\mathbb{E}_1 = \{v_4, v_1, i\}$, $\mathbb{E}_2 = \{v_1, v_2, i\}$, $\mathbb{E}_3 = \{v_2, v_3, i\}$, $\mathbb{E}_4 = \{v_3, v_4, i\}$.
Så for hver $\mathbb{E}_j$, $j = 1,\ 2,\ 3,\ 4$
Hvis $\Psi_{\mathbb{E}_j[0]}$ ikke inneholder $\mathbb{E}_j[1]$, kanten er ikke allerede laget, da må:
$\Psi_{\mathbb{E}_j[1]}$.insert $\{\mathbb{E}_j[0], s\}$, hvor s er størrelsen på kant-vektoren E, og
E.push_back($\mathbb{E}_j$) (vi legger den nye kanten i slutten av kant-vektoren).
Hvis kanten allerede er laget, får vi kantindeksen fra $\Psi_{\mathbb{E}_j[0]}$ og oppdaterer deretter F_i, og vi oppdaterer kanten med indeks i til den nye flatens.

Legg merke til at hvis inngangsflatene har riktig orientering, så er orienteringen riktig også etter en subdivisjon.

Etter å ha gjort ferdig subdivisjonen trenger vi flatenormalene i hver verteks. Siden orienteringen er lagret i flatene, bruker vi den til å lage flatenormalene. Verteksene/kantene til en flate er organisert mot klokken sett utenfra. Det finnes flere artikler om å beregne flatenormaler, for eksempel [73]. Men her skal vi bruke en enkel teknikk basert på summen av normalene til de tilstøtende trekantede delene av flatene til en verteks. Disse normalene konvergerer mot de korrekte flatenormalene. Vi bruker samme teknikk som for kantene, dvs. vi bruker et hjelpeverktøy, vi lager en tabell Ψ med størrelse lik den nye Verteks-tabellen, og hvor hvert element er en liste over indekser til flater. For å lage denne matrisen Ψ går vi gjennom alle flater, og for hver flate setter vi inn flateindeksen i Ψ_i, der i er indeksene til alle verteksene til flaten. Neste trinn er å gå gjennom hvert element i Ψ, dvs

✓ For hvert element av Ψ, som er Ψ_i.
  For hver flate av Ψ_i, i listen over indekser til verteksene,
    finn indeksen (f)ør og (e)tter indeks i,
      beregn $n = n + (v_e - v_i) \wedge (v_f - v_i)$, (husk at $\wedge$ er vektorproduktet)
    oppdater v_i med dens enhetsnormal $\frac{n}{|n|}$.

Flaten **a)** i figur 10.2 er en Catmull-Clark-subdivisjonsflate laget fra en kube (plottet i blått). 5 subdivisjonstrinn er brukt, og vi fikk følgende antall elementer,

Subdivisjonstrinn	0	1	2	3	4	5	formel
Antallet vertekser	8	26	98	386	1538	6146	$v+k+f$
Antallet kanter	12	48	192	768	3072	12288	$2v+f$
Antallet flater	6	24	96	384	1536	6144	$4f$

10.1.2 Doo-Sabin og Mid-Edge

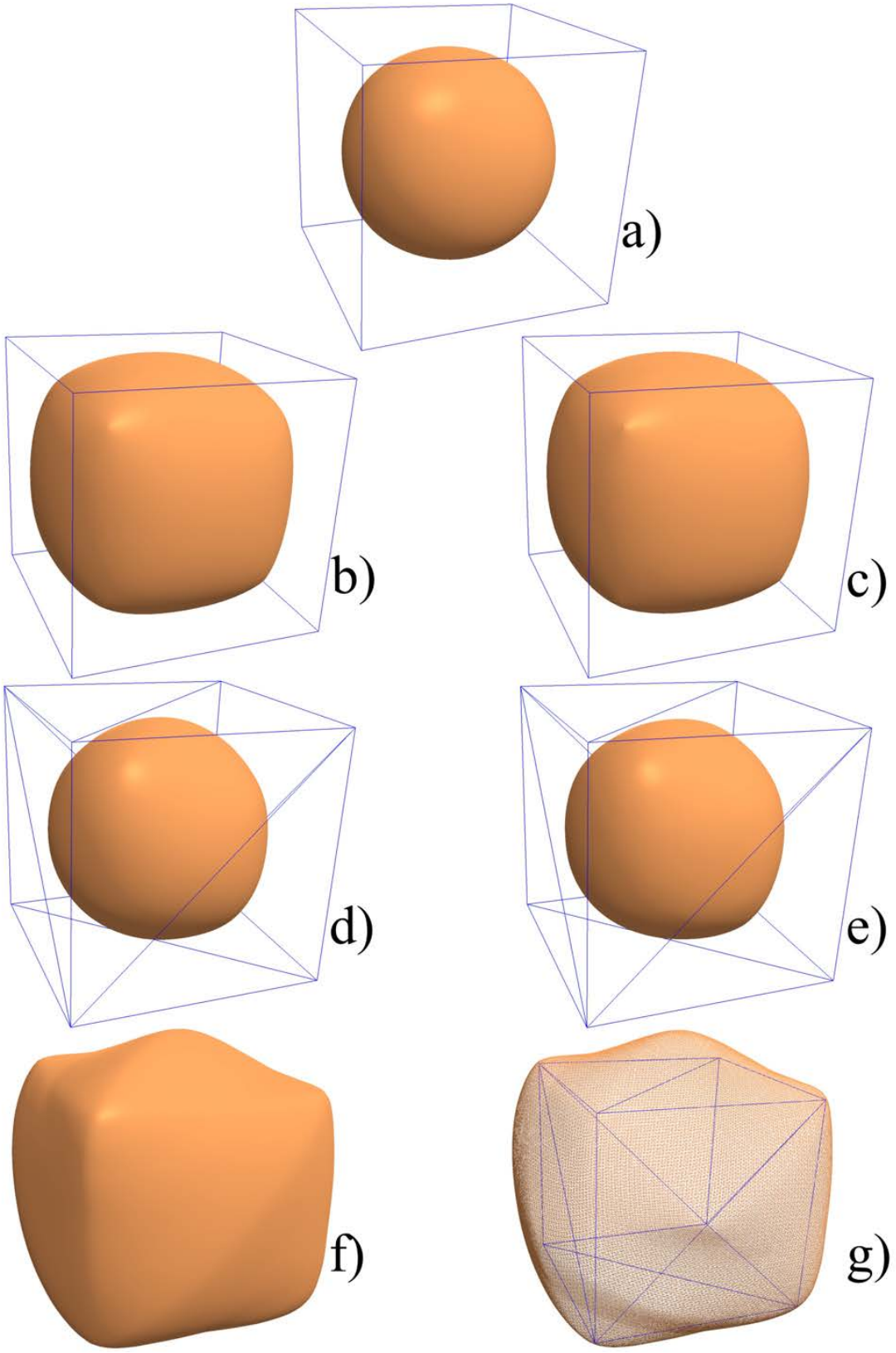
Doo-Sabin-subdivisjonsskjema er også basert på firkanter. Skjemaet ble lansert i [55], og er egentlig Chaikins hjørnekuttingsmetode videreutviklet fra kurver til flater, se seksjon 6.7.2. Den er basert på bi-kvadratiske uniforme B-splineflater og konvergerer mot en C^1-glatte flate med et start-nett med vilkårlig topologi [2]. Etter èn subdivisjon har alle vertekser grad 4.

En sjablong av Doo-Sabin-skjemaet vises nederst til venstre i figur 10.1. Der ser vi at flate- og kant-punktene som skal lages kun er hjelpepunkt og ikke nye vertekser. De nye verteksene er de 4 punktene som erstatter hver av de gamle verteksene. Dernest, for hvert subdivisjonstrinn og for alle flater og kanter, lager vi følgende to tabeller med hjelpepunkt:

flate-punkt er et punkt i midten av en flate: $f_i = \frac{1}{m}\sum_j v_j$, der m er antall vertekser som definerer flaten. For de fleste flatene er $m = 4$.

kant-punkt er et punkt i midten av en kant: $e_i = \frac{1}{2}(v_a + v_b)$, der v_a og v_b er de gamle verteksene som definerer gjeldende kant.

[2] Et hjelpepunkt kan forbedre formen til Doo-Sabin-subdivisjon, se [22].



Figur 10.2: *Catmull-Clark er vist i a), Doo-Sabin i b), Mid-Edge i c), Loop i d), $\sqrt{3}$-Kobbelt i e) og interpolasjonsskjemaet Butterfly er vist i f) og g). Vi ser at Doo-Sabin og Mid-Edge konvergerer mot samme flate, og det gjør også Loop og $\sqrt{3}$-Kobbelt.*

Videre kan vi enten bruke hjelpetabeller eller en utvidet datastruktur for å koble flater og kanter til hver verteks. Neste trinn er å lage en ny tabell med vertekser som erstatter de gamle.

✓ For hver gammel verteks v_i, (som har n flater og kanter første gang, 4 senere)
For hver flate F_j koblet til verteks v_i lager vi nye vertekser v_{ji},
hvor f_j er flate-punktene, e_a og e_b er de tilkoblede kant-punktene,
$v_{ji} = \frac{1}{4}\left(v_i + f_j + e_a + e_b\right)$.
Til slutt lager vi en flate basert på de n-verteksene vi nå har laget.

Etter dette lager vi en flate inni hver gammel flate og vi lager en flate rundt hver gammel kant. Kantene, koblingen til flatene og flatenormalene kan lages på samme måte som i Catmull-Clark-skjemaet.

Flaten **b)** i figur 10.2 er en Doo-Sabin-subdivisjonsflate laget fra en kube som er plottet i blått. Der er brukt 5 step, og vi fikk følgende antall elementer,

Subdivisjonstrinn	0	1	2	3	4	5	formel
Antallet vertekser	8	24	96	384	1536	6144	$4v$
Antallet kanter	12	48	192	768	3072	12288	$8v$
Antallet flater	6	26	98	386	1538	6146	$v+k+f$

Merk at formelen ikke er riktig for det første trinnet på grunn av verteksgraden.

Mid-Edge subdivisjonsskjema ble foreslått uavhengig av Jörg Peters og Ulrich Reif [130] og Ayman Habib og Joe Warren [85]. Førstnevnte brukte midtpunktet på hver kant for å bygge hvert nytt nett. Sistnevnte brukte en fireretnings-bokspline for å bygge skjemaet. Dette skjemaet genererer samme overflate som Doo-Sabin, men konvergerer merkbart langsommere, faktisk 2 til 1 som vi kan se i tabellene.

Skjemaet er ganske enkelt, vi erstatter de gamle verteksene med nye. De nye er midtpunktet på hver gamle kant. Etter å ha laget de nye verteksene lager vi flatene inne i hver gammel flate og rundt hver gammel verteks. Kanter og flatenormaler lages på samme måte som for Doo-Sabin skjema.

Flaten **c)** i figur 10.2 er en Mid-Edge-subdivisjonsflate laget fra en kube. 8 trinn er brukt, og vi fikk følgende antall elementer,

Subdivisjonstrinn	0	1	2	3	4	5	6	7	8	formel
Antallet vertekser	8	12	24	48	96	192	384	768	1536	k
Antallet kanter	12	24	48	96	192	384	768	1536	3072	$2k$
Antallet flater	6	14	26	50	98	194	386	770	1538	$v+f$

10.1.3 Loop og $\sqrt{3}$

Loop-subdivisjonsskjema, basert på trekanter, ble introdusert i 1987 av Charles Loop, [114] og [115]. Han foreslo et subdivisjonsskjema basert på en fjerdegrads boks-spline med seks retninger som gir et skjema som konvergerer mot en C^2-kontinuerlig flater overalt bortsett fra i ekstraordinære vertekser som er C^1. En sjablong av skjemaet er vist til venstre midt i figur 10.1, og en flate generert fra en "boks" er vist i figur 10.2 merket

med **d)**. “Boksen" er vist som et sett trekanter tegnet med blå linjer i figuren. Mangelen på symmetri som vi ser i plottet skyldes trianguleringen av kuben. Normalt skal graden være 6 for de fleste verteksene, men i eksempelet i figur 10.1 har halvparten av de innledende verteksene grad 4 og resten har grad 5, og de vil alle beholde graden etter alle subdivisjonene. Imidlertid vil alle nye vertekser ha grad 6.

Loop-skjemaet er enkelt. Det er å dele hver kant i to og dermed alle trekanter i 4. I hvert forfiningstrinn må vi sørge for at kantene og flatene er koblet, så må vi for alle kanter og vertekser oppdatere de gamle verteksene og lage nye **vertekser** på følgende måte:

Verteks-punkt er en oppdatering av posisjonen til en gammel verteks: $v_i = (1-\alpha_n)\,v_i + \frac{\alpha_n}{n}\sum_{j\in\beta} v_j$, der n er graden til verteksen som skal oppdateres, β er settet med indekser til naboverteksene, og α_n er en grads-verdi beskrevet i (10.1). Eksempelet fra figur 10.1 er: $v_{00} = (1-\alpha_6)\;v_0 + \frac{\alpha_6}{6}\sum_{j=1}^{6} v_j \approx 0{,}635 v_0 + 0{,}0625\sum_{j=1}^{6} v_j$.

Kant-punkt er en verteks i midten av en kant: $e_i = \frac{1}{8}\left(3v_a + 3v_b + v_f + v_g\right)$, der v_a og v_b er gamle vertekser som definerer gjeldende kant, og v_f og v_g er de manglende verteksene fra de to sammenkoblede flatene/trekantene. Et eksempel fra figur 10.1 er $e_3 = \frac{1}{8}\left(3v_0 + 3v_3 + v_2 + v_4\right)$.

Alle verteksene er nå oppdatert eller laget. Vi kaller indeksen i verteksene til det første Edge-punktet for m_e. Nå er det en enkelt sammenheng mellom indeksen til en kant og et kant-punkt. Vi kan nå oppdatere og lage nye flater/trekanter:

✓ For hver gammel flate/trekant F_i (som har 3 vertekser),
hvor $F_i = \{v_1, v_2, v_3\}$ og
kant-punktene er $\{\varepsilon_1, \varepsilon_2, \varepsilon_3\} = \{e_1 + m_e,\ e_2 + m_e,\ e_3 + m_e\}$,
oppdaterer vi de gamle flatene F_i, og lager 3 nye flater F_{i2}, F_{i3}, F_{i4}, dvs.
$F_i = \{v_1, \varepsilon_1, \varepsilon_3\}$, $F_{i2} = \{\varepsilon_1, v_2, \varepsilon_2\}$, $F_{i3} = \{\varepsilon_1,\ \varepsilon_2,\ \varepsilon_3\}$ og $F_{i4} = \{\varepsilon_2, v_3, \varepsilon_3\}$.

Kanter og flatenormaler kan nå lages på samme måte som i Catmull-Clark-skjemaet.

Vektingsverdiene α_n for Loop-subdivisjon som brukes til å oppdatere gamle vertekser er

$$\alpha_n = \frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2. \tag{10.1}$$

Beregnede verdier for gradstallene 4, 5, 6, 7 og 8 kan sees i Tabell 10.1.

Objekt **d)** in Figur 10.2 er en Loop-subdivisjonsflate laget av en triangulert kube som er plottet i blått. 5 subdivisjonstrinn er brukt, noe som ga følgende antall elementer,

Subdivisjonstrinn	0	1	2	3	4	5	formel
Antallet vertekser	8	26	98	386	1538	6146	$v+k$
Antallet kanter	18	72	288	1152	4608	18432	$2k+3f$
Antallet flater	12	48	192	768	3072	12288	$4f$

Hvis vi sammenligner tabellen med Catmull-Clark-tabellen på side 201, ser vi at antall vertekser er det samme, antallet flater er dobbelt så stort (en firkant er to trekanter), og antall kanter for Loop er lik antall kanter pluss antall flater i Catmull-Clark.

Grad:	$n=$	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

Tabell 10.1: *Vektingsverdiene α_4, α_5, α_6, α_7, og α_8.*

$\sqrt{3}$-Kobbelt skjema er også basert på trekanter - Det ble utviklet av Leif Kobbelt [99], og det håndterer vilkårlige trekantbaserte net, det er C^2 kontinuerlig overalt bortsett fra ved ekstraordinære vertekser der det er C^1 kontinuerlig, og det tilbyr en naturlig tilpasset forfining når det er nødvendig. Samtidig er det et "dualt" skjema for et trekant-net som konvergerer tregere enn Loop, men det konvergerer mot samme flate.

Et $\sqrt{3}$-Kobbelt-skjema er å lage nye vertekser i midten av hver flate/trekant, oppdatere gamle vertekser og dele de gamle trekantene i 3 og rotere alle gamle kanter. Se sjablongen til høyre i midten av figur 10.1. Dermed får vi

Verteks-punkt er en oppdatering av posisjonen til en gammel verteks: $v_i = (1-\alpha_n)\, v_i + \frac{\alpha_n}{n}\sum_{j\in\beta} v_j$, hvor n er graden til verteksen som skal oppdateres, β er settet med indekser til naboverteksene, og α_n er en gradsverdi i (10.2). Exampelet fra figur 10.1 er: $v_{00} = (1-\alpha_6)\; v_0 + \frac{\alpha_6}{6}\sum_{j=1}^{6} v_j$.

Flate-punkt er et punkt midt i en flate: $f_i = \frac{1}{3}(v_a + v_b + v_c)$, hvor v_a, v_ab og v_c er de gamle verteksene som definerer gjeldende flate/trekant. Et eksempel fra figur 10.1 er $f_3 = \frac{1}{3}(v_0 + v_3 + v_4)$.

Alle verteksene er nå oppdatert eller laget. Vi kaller indeksen i vertekstabellen til det første flate-punktet for m_f. Det et nå en enkelt sammenheng mellom indeksen til en flate og et flate-punkt. Nå kan vi oppdatere og lager nye flater/trekanter.

✓ For hver gammel kant E_i,
  delt av to flater/trekanter med indeksene f_a og f_b
  og der indeksene til flate-punktene er $\phi_a = f_a + m_f$ og $\phi_b = f_b + m_f$
lager vi to nye flater $F_{2i} = \{E_i[0],\ \phi_a,\ \phi_b\}$ og $F_{2i+1} = \{E_i[1],\ \phi_b,\ \phi_a\}$

Merk at det er viktig at de to flatene til en kant er organisert slik at f_a er på høyre side av kanten sett fra utsiden av overflaten. Forøvrig kan kanter og flatenormaler lages på samme måte som i Catmull-Clark.

Vektingsverdiene α_n for $\sqrt{3}$-Kobbelt-subdivisjon som brukes til å oppdatere gamle vertekser er

$$\alpha_n = \frac{4 - 2\cos\frac{2\pi}{n}}{9}, \tag{10.2}$$

og beregnede verdier for noen gradstall kan sees i Tabell 10.1.

Objekt **e)** i figur 10.2 er en $\sqrt{3}$-Kobbelt subdivisjonsflate laget av en triangulert kube som er plottet i blått. 6 trinn er brukt, og vi fikk følgende antall elementer,

Subdivisjonstrinn	0	1	2	3	4	5	6	formel
Antallet vertekser	8	20	56	164	488	1460	4376	$v+f$
Antallet kanter	18	54	162	486	1458	4374	13122	$e+3f$
Antallet flater	12	36	108	324	972	2916	8748	$2e$

10.1.4 Butterfly

Catmull-Rom subdivisjonsspline, diskutert i seksjon 6.7.1, og generalisert til “Et 4-punkts interpolatorisk subdivisjonsskjema" i [56], interpolerer alle punkt, både startpunktene og punkt generert av subdivisjonene.

I 1990 lanserte Nira Dyn o.a. et subdivisjonsskjema for flater basert på trekanter [59]. Skjemaet er en utvidelse av (6.29) med at en spenningsparameter ω ble introdusert. I en vanlig Catmull-Rom subdivisjonsspline er $\omega = \frac{1}{16}$. Hvis vi bruker indeksene til vertekssene fra sjablongen øverst til høyre i figur 10.1, startet Dyn o.a. med $e_0 = u(v_1+v_2) + v(v_3+v_4) - \omega(v_5+v_6+v_7+v_8)$ og viste at for å være C^0 så må $2u+2v-4\omega = 1$ og hvis $u = \frac{1}{2}$ så har vi C^1-continuitet, og det følger også at $v = 2\omega$.

Skjemaet kalles Butterfly (sommerfugl) på grunn av formen. Den er basert på trekanter og graden er derfor vanligvis 6, med den fungerer også med grad 5, 7 og 8. I eksemplet vi skal se på er graden innledningsvis 4 i noen av verteksene. Vi har da satt v_6 og v_8 til å være samme punkt. Skjemaet er da som følger. For hvert forfiningstrinn og for alle flater, kanter og vertekser, lager vi følgende nye vertekser (punkt),

Verteks-punkt er settet av de gamle verteksene og de vil forbli uendret.

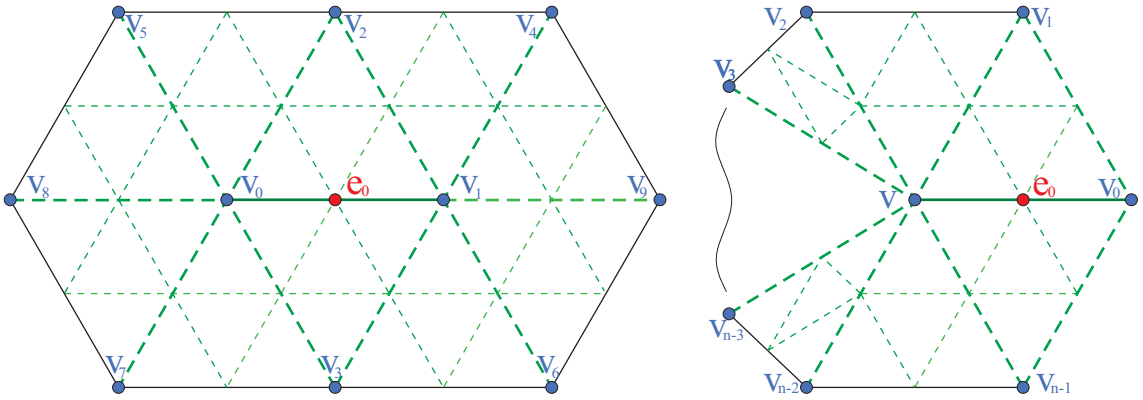
Kant-punkt er et punkt i midten av en kant. Vi bruker indeksene fra sjablongen øverst til høyre i figur 10.1: $e_0 = \frac{1}{2}(v_1+v_2) + 2\omega(v_3+v_4) - \omega(v_5+v_6+v_7+v_8)$.

Spenningsparameteren ω bør ikke være langt fra $\frac{1}{16}$, og det er også mulig å ha forskjellig verdi på hver enkelt verteks. Alle vertekser er nå oppdatert eller klare for gjenbruk. Vi angir at indeksen i toppunktet til det første Kant-punkt er m_e. Da er det et enkelt overgang mellom indeksen til en kant og et kant-punkt. For å oppdatere og lage nye flater/trekanter, kanter og til slutt normaler bruker vi samme prosedyre som for Loop.

Objekt **f)** i figur 10.2 er laget ved hjelp av Butterfly-skjemaet. Den er laget av den triangulerte kuben som er sees i blått i **g)** i figur 10.2. 6 trinn er brukt, og vi fikk følgende antall elementer,

Subdivisjonstrinn	0	1	2	3	4	5	6	formel
Antallet vertekser	8	26	98	386	1538	6146	24578	$v+k$
Antallet kanter	18	72	288	1152	4608	18432	73728	$2k+3f$
Antallet flater	12	48	192	768	3072	12288	49152	$4f$

Butterfly-skjemaet er et 8-punkts interpolasjonsskjema, som vi kan observere i Butterfly-sjablongen i figur 10.1. Det finnes en utvidelse, et 10-punktsskjema også kalt “modifisert Butterfly", beskrevet i [169]. Her har vertekser med grad forskjellig fra 6 et modifisert skjema som bare bruker den ekstraordinære verteksen. Dette skjemaet er illustrert på høyre side i figur 10.3. Til venstre i figuren ser vi det ordinær 10 punktskjemaet som krever



Figur 10.3: *Modifisert Butterfly-skjema. Til venstre ser vi en sjablong av et ordinært 10-punktsskjema, der begge verteksene til en kant har grad 6. Til høyre er en sjablong av et skjema der en verteks v har grad forskjellig fra 6.*

at begge verteksene til kanten som behandles har grad 6.

Modifiserte Butterfly endrer litt måten å lage ny kant på, noe som gir,

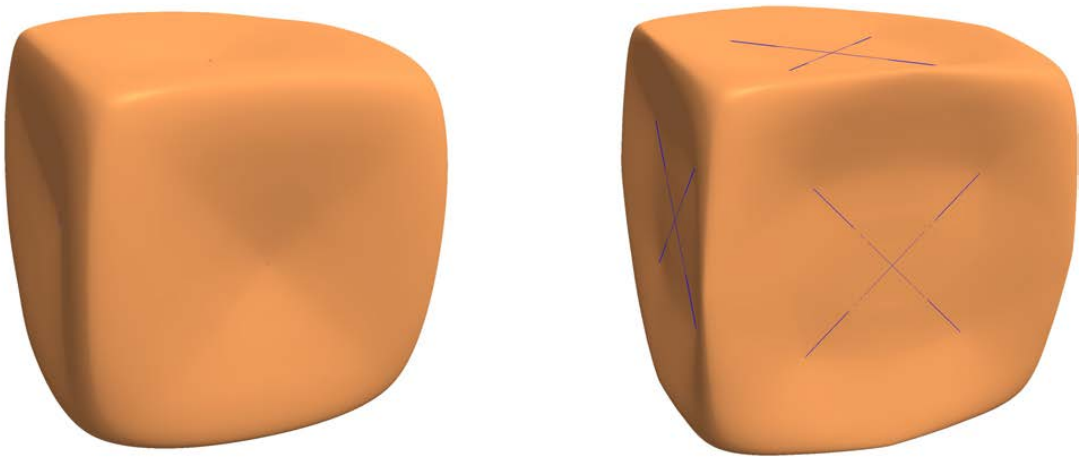
Kant-punkt , hvor begge verteksene har grad 6:
Indeksnavn er hentet fra sjablongen til venstre i figuren 10.3:
$e_0 = a(v_0+v_1)+b(v_2+v_3)+c(v_4+v_5+v_6+v_7)+d(v_9+v_9)$
hvor $a=\frac{1}{2}-w$, $b=\frac{1}{8}+2w$, $c=-\frac{1}{16}-w$ og $d=w$.

Kant-punkt , hvor en verteks har grad $\neq 6$:
Indeksnavn er hentet fra sjablongen til høyre i figuren 10.3:
Hvis graden er $n=3$, $e_0 = \frac{3}{4}v+\frac{5}{12}v_0-\frac{1}{12}v_1-\frac{1}{12}v_2$.
Hvis graden er $n=4$, $e_0 = \frac{3}{4}v+\frac{3}{8}v_0+\cancel{0v_1}-\frac{1}{8}v_2+\cancel{0v_3}$.
Hvis graden er $n>4$, $e_0 = \frac{3}{4}v+\sum_{j=0}^{n-1}\frac{1}{n}\left(\frac{1}{4}+\cos\frac{2\pi j}{n}\frac{1}{2}+\cos\frac{4\pi j}{n}\right)v_j$.

Kant-punkt , hvor begge verteksene har grad $\neq 6$:
Bruk formelen ovenfor for både v og v_0, og bruk snittet, dvs $e_0 = \frac{e_0(v)+e_0(v_0)}{2}$.

Merk at w brukt for grad 6 ikke er det samme som den forrige ω brukt i vanlig Butterfly. Hvis $w=0$ får vi faktisk et vanlig 8-punkts Butterfly-skjema med $\omega=\frac{1}{16}$. Merk også at siden $\sum_{j=0}^{n-1}\frac{1}{n}\left(\frac{1}{4}+\cos\frac{2\pi j}{n}\frac{1}{2}+\cos\frac{4\pi j}{n}\right)=\frac{1}{4}$ vil vektene i alle deler av et Modifisert Butterfly-skjema summere opp til 1.

I figur 10.4 er det til venstre en flate laget av et vanlig Butterfly-skjema og til høyre en flate laget av modifisert Butterfly. De startet begge fra en kube med 8 hjørnepunkt pluss ett punkt i midten av hver av de 6 sidene til kuben, totalt 14 punkt som definerer 24 trekanter. I figuren ser vi litt av de innledende trekantene i blått. Som vi kan se har verteksene i midten av sidene grad 4. Som vi kan observere til venstre håndterer ikke det vanlige Butterfly-skjemaet nødvendigvis grad 4 på en glatt måte.



Figur 10.4: *Til venstre ser vi en flate fra vanlig Butterfly og til høyre ser vi en fra modifisert Butterfly. Utgangspunktet for begge er 8 punkt som danner en terning pluss ett punkt i midten av hver av de 6 sidene i terningen, det vil si totalt 14 punkt*

10.1.5 Kvadratisk interpolerende - Kobbelt

Som vi har sett for kurver, interpolerer “4-punkts interpolatorisk subdivisjonsskjema", (6.29), alle punkt, både startpunkt og punkt generert av alle subdivisjonstrinn. Hvis vi utvider dette til to retninger, som i B-spline-tensorprodukt, får vi et kvadratisk interpolasjonsskjema. Dette ble gjort og testet av Leif Kobbelt i [98], og han introduserte også en mulig løsning for ekstraordinære vertekser.

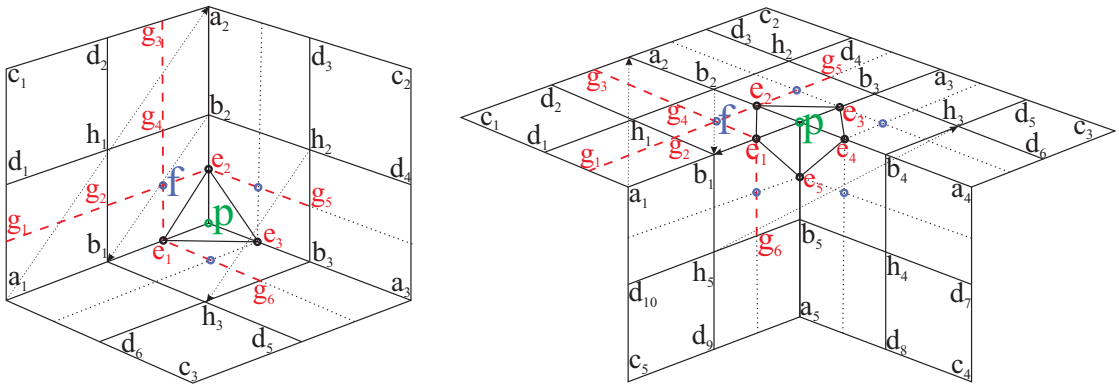
I seksjon 9.5.3 viste vi at en Tensor-produktflate bare er en flate fra blending av kurver. Det følger at et skjema for interpolering da ganske enkelt er et kurveskjema i to retninger. Hvis vi har vanlige vertekser, lager vi kun kant-punkt for alle kanter ved å bruke kurveskjemaet (6.29). For flate-punkt er det også enkelt, vi kan lage dem fra de 4×4 verteksene som omgir flaten ved å bruke et tensorprodukt (ytre produkt) av kurveskjemaet. Men siden vi allerede har laget kantpunktene, og på grunn av symmetrien til tensorproduktet, kan vi i stedet kun bruke kurveskjemaet på disse nye kantpunktene, og resultatet blir det samme uansett hvilken retning vi velger.

Derfor er skjemaet å lage en ny verteks i “midten" av hver kant, og en ny verteks i “midten" av hver flate. For vanlige vertekser, med grad 4, får vi

Kant-punkt er et punkt i “midten" av en kant. Hvis begge verteksene til en kant er regelmessige, bruker vi skjemaet (10.3), der p_i og p_{i+1} er verteksene som definerer kanten og der p_{i-1} og p_{i+2} er de neste verteksene i forlengelsen av kanten.

Flate-punkt er et punkt i “midten" av en flate. Hvis alle verteksene i en flate er regelmessige, bruker vi skjemaet (10.3), der p_i og p_{i+1} er nye kant-punkt på to motsatte kanter av flaten, og p_{i-1} og p_{i+2} er de nye kant-punktene på motsatte kanter av de motsatte flatene av de to første kantene.

For å forenkle implementeringen av kant- og flate-punkt nær ekstraordinære vertekser, er det nyttig å beholde symmetrien til flate-punktene. Derfor, for å beregne et kant-punkt når



Figur 10.5: *En illustrasjon av hvordan du genererer et flate-punkt f når graden til verteks p er 3 (figuren til venstre) og 5 (figuren til høyre). g_3, g_4, e_1, g_6 og g_1, g_2, e_2, g_5 er kant-punkt, e_1, e_2, e_3 og e_4, e_5 er kant-punktene nærmest p.*

en eller begge verteksene på kanten er ekstraordinære, kan **virtuelle vertekser** introduseres. Skjemaet i (6.29) kan nå omformuleres til

$$\widehat{p}_i = \mu(p_i + p_{i+1}) - \omega(p_{i-1} + p_{i+2}), \quad \text{hvor} \quad \mu = \frac{1}{2} + \omega \tag{10.3}$$

Først ser vi på grad 3, illustrert på venstre side i figur 10.5. Vi beregner flate-punktet f fra g_3, g_4, e_1, g_6 og setter dette lik f beregnet fra g_1, g_2, e_2, g_5, dvs.

$$\begin{Bmatrix} \omega^2 c_1 & -\omega\mu\, d_2 & -\omega\mu a_2 & +\omega^2 d_3 \\ -\omega\mu\, d_1 & +\mu^2 h_1 & +\mu^2 b_2 & -\omega\mu\, h_2 \\ & +\mu\, e_1 & & \\ +\omega^2 d_6 & -\omega\mu\, h_3 & -\omega\mu b_3 & +\omega^2 h_2 \end{Bmatrix} = \begin{Bmatrix} \omega^2 c_1 & -\omega\mu\, d_1 & -\omega\mu\, a_1 & +\omega^2 d_6 \\ -\omega\mu\, d_2 & +\mu^2 h_1 & +\mu^2 b_1 & -\omega\mu\, h_3 \\ & +\mu\, e_2 & & \\ +\omega^2 d_3 & -\omega\mu\, h_2 & -\omega\mu\, b_3 & +\omega^2 h_3 \end{Bmatrix}.$$

En oppsummering og omorganisering av dette gir

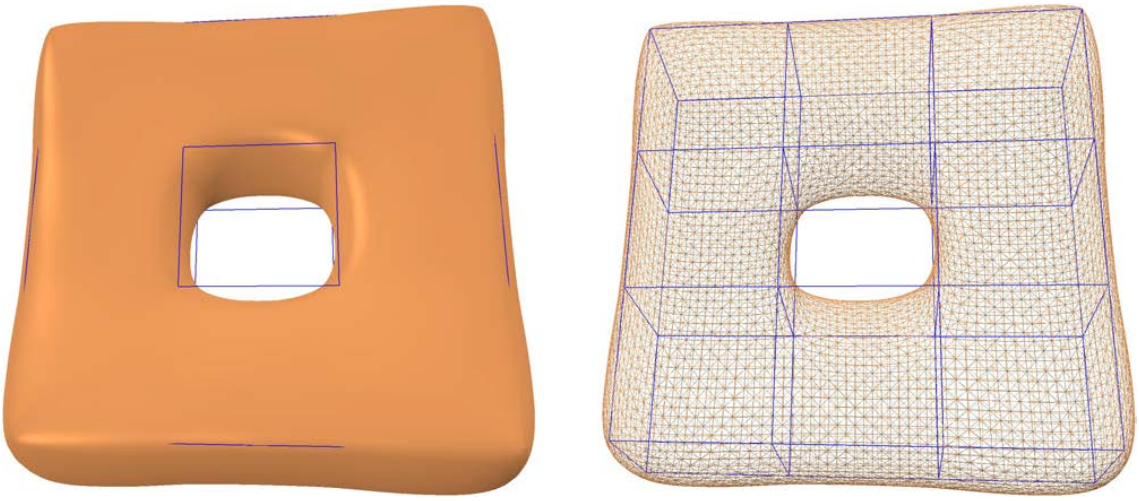
$$e_1 - e_2 = \omega(a_2 - a_1) + \mu(b_1 - b_2) + \frac{\omega^2}{\mu}(h_3 - h_2). \tag{10.4}$$

Vi gjør nå det samme for grad 5, dvs. høyre side av figur 10.5, og får

$$\begin{Bmatrix} \omega^2 c_1 & -\omega\mu\, d_2 & -\omega\mu a_2 & +\omega^2 d_3 \\ -\omega\mu\, d_1 & +\mu^2 h_1 & +\mu^2 b_2 & -\omega\mu\, h_2 \\ & +\mu\, e_1 & & \\ +\omega^2 d_{10} & -\omega\mu\, h_5 & -\omega\mu b_5 & +\omega^2 h_4 \end{Bmatrix} = \begin{Bmatrix} \omega^2 c_1 & -\omega\mu\, d_1 & -\omega\mu\, a_1 & +\omega^2 d_{10} \\ -\omega\mu\, d_2 & +\mu^2 h_1 & +\mu^2 b_1 & -\omega\mu\, h_5 \\ & +\mu\, e_2 & & \\ +\omega^2 d_3 & -\omega\mu\, h_2 & -\omega\mu\, b_3 & +\omega^2 h_3 \end{Bmatrix},$$

$$e_1 - e_2 = \omega(a_2 - a_1) + \mu(b_1 - b_2) + \omega(b_5 - b_3) + \frac{\omega^2}{\mu}(h_3 - h_4). \tag{10.5}$$

Merk at ett kant-punkt brukes til å lage flate-punkt i to tilstøtende flater. Det følger at i alle flater knyttet til et ekstraordinært vertekser, er det en låst vektor mellom de to kant-punktene på de tilstøtende kantene. Dette betyr at for alle ekstraordinære vertekser med grad n, ligger de nærmeste kant-punktene i hjørnene av en n-kantet polygon. For $n = 3$ er dette en trekant og for $n = 5$ er det en femkant, som begge kan se i figur 10.5.



Figur 10.6: *En interpolerende Kvadratisk-subdivisjonsflate. Til høyre vises orginalpunktene og firkantene i blått. Det er 8 utvendige hjørner med grad 3 og 8 innvendige hjørner med grad 5. De resterende 16 punktene er vanlige vertekser med grad 4.*

Formen og orienteringen til polygonet er gitt. Plasseringen av polygonet er imidlertid en frihet vi kan bruke, men under begrensningen at tyngdepunktet til kant-punktene må konvergere glatt mot den ekstraordinære verteksen. En måte å gjøre dette på er å sammenligne med vanlige vertekser. Tyngdepunktet X for en vanlige verteks $n = 4$ er

$$X = \frac{1}{n}\sum_{i=1}^{n} e_i = \mu\ p + \frac{1}{n}\left(\frac{1}{2}\sum_{i=1}^{n} b_i - \omega \sum_{i=1}^{n} a_i\right) \tag{10.6}$$

Fordi vi, for ekstraordinære vertekser, kan beregne avstandsvektoren mellom hvert par av kant-punkt, for eksempel (10.4) og (10.5), kan vi beregne tyngdepunktet bare avhengig av ett av kant-punktene,

$$X = \frac{1}{n}\sum_{i=1}^{n} e_i = e_1 + \frac{1}{n}\sum_{i=1}^{n-1}(n-i)(e_{i+1} - e_i) \tag{10.7}$$

Vi setter så (10.6) lik (10.7) og omorganiserer det. Resultatet er en generell formel for å beregne kantpunkt koblet til en ekstraordinær verteks:

$$e_1 = \mu\ p + \frac{1}{n}\left(\frac{1}{2}\sum_{i=1}^{n} b_i - \omega \sum_{i=1}^{n} a_i + \sum_{i=1}^{n-1}(n-i)(e_i - e_{i+1})\right). \tag{10.8}$$

Til slutt omformulerer vi dette til

$$e_1 = -\omega\, a_1 + \mu\, b_1 + \mu\ p - \omega\, v, \tag{10.9}$$

der v er en virtuell verteks som kan uttrykkes ved å kombinere (10.8) med enten (10.4) hvis $n = 3$ eller (10.5) hvis $n = 5$, eller et relatert uttrykk hvis $n > 5$. Merk at hvis b_1 også er en ekstraordinær verteks, erstatter vi a_1 med en annen virtuell verteks.

Hvis $n = 3$ erstatter vi vektorene $e_1 - e_2$ i (10.8) med (10.4), og $e_2 - e_3$ med en rotert (10.4), og får den virtuelle verteksen

$$v = \frac{1}{3}\sum_{i=1}^{3} b_i + \frac{\omega}{\mu}\left(h_2 - \frac{1}{3}\sum_{i=1}^{3} h_i\right). \tag{10.10}$$

Hvis $n = 5$ erstatter vi vektorene $e_1 - e_2$ med (10.5), og $e_2 - e_3$, $e_3 - e_4$ og $e_4 - e_5$ med en 1 gang, 2 ganger og 3 ganger rotert (10.5), og får den virtuelle verteksen

$$v = b_3 + b_4 - \left[\frac{1}{5}\sum_{i=1}^{5} b_i + \frac{\omega}{\mu}\left(h_3 - \frac{1}{5}\sum_{i=1}^{5} h_i\right)\right]. \tag{10.11}$$

I figur 10.6 vises et eksempel på en interpolatorisk subdivisjonsflate basert på kvadrater. Algoritmen bruker formel (10.3), (10.9), (10.10) og (10.11). Venstre side av figur 10.6 viser startpunktene og firkantene i blått. Det er 8 ytre hjørner med grad 3 og 8 indre hjørner med grad 5. De resterende 16 punktene er vanlige vertekser med grad 4. Fire forfiningstrinn ble kjørt. Fra de første 32 punktene, 64 kantene og 32 flatene genererte dette 8192 vertekser, og programmet brukte 95 millisekunder på beregningen. Totalt, inkludert generering av normaler og trekanter for grafikken, brukte programmet 161 millisekunder. Programmet ble kjørt på Intel Core i9-9900 CPU, 3,6 GHz og er laget av enkle, men $\mathcal{O}(n)$-algoritmer og uten optimalisering, faktisk i debug-modus.

Kapittel 11

To-flateblending

I boolske sum-flater blendes tre flater sammen til en flate. Vi skal nå se på en metode hvor vi kun bruker de to første flatene, de som er laget ved å blende kurver

Fra uttrykket (9.39), Coons patch - bikubisk blending, har vi de to første flatene $S_1(u,v)$ og $\overline{S}_2(u,v)$, begge laget av blending kurver. Disse to flatene er definert på et felles domene $U = [0,1] \times [0,1]$. Differanseflaten mellom disse to er

$$\widetilde{S}(u,v) = \overline{S}_2(u,v) - S_1(u,v). \tag{11.1}$$

Vi kan nå prøve lage en blendingsflate med B-funkjosn på vanlig måte

$$S(u,v) = S_1(u,v) + B(u,v)\ \widetilde{S}(u,v), \tag{11.2}$$

hvor $B(u,v)$ er en blendingsfunksjon i to variabler (2-p). Vi vet at i kapittel 7, definisjon 7.1, er B-funksjoner i én variabel (1-p) definert. I det følgende skal vi se på hvordan vi kan konstruere en 2-p B-funksjon, $B(u,v)$, med lignende egenskaper som en 1-p B-funksjon, dvs. at den intern må være minst C^d-glatt der d er ordenen til en 1-p B-funksjon som brukes til å bygge $B(u,v)$, og at alle deriverte opp til ordenen d er null ved rendene til domenet.

11.1 2-p B-funksjon

For å konstruere en 2-p B-funksjon $B(u,v)$ av orden ≥ 1, jfr. definisjon 7.1, starter vi med en funksjon som er en sammensatt invers og symetrisk B-funksjon (se definisjon 7.1)

$$g(u) = \begin{cases} 1 - \frac{1}{2}B(2u), & \text{hvis} \quad u \leq \frac{1}{2}, \\ 1 - \frac{1}{2}B(2-2u), & \text{ellers}, \end{cases} \tag{11.3}$$

med glatthet som samsvar med ordenen til B-funksjonen, og som er symmetrisk om $u = \frac{1}{2}$. Det følger videre at

$$g'(u) = \begin{cases} -B'(2u), & \text{hvis} \quad u \leq \frac{1}{2} \\ B'(2-2u), & \text{ellers} \end{cases} \qquad g''(u) = \begin{cases} -2\,B''(2u), & \text{hvis} \quad u \leq \frac{1}{2}, \\ -2\,B''(2-2u), & \text{ellers.} \end{cases} \tag{11.4}$$

Vi lager så to hjelpefunksjonene til

$$a(u) = 2u(1-u), \quad \text{hvor} \quad a' = 2(1-2u), \quad \text{og} \quad a'' = -4, \tag{11.5}$$

og

$$t(u,v) = \begin{cases} \frac{v}{a(u)}, & \text{hvis} \quad v < a(u) \\ \frac{1-v}{a(u)}, & \text{hvis} \quad v > 1-a(u) \\ 1 & \text{ellers} \end{cases} \tag{11.6}$$

Vi ser at $t(u,v)$ er kontinuerlig. Det følger at

$$t_u = \begin{cases} \frac{-v\ a'}{a^2}, & \text{hvis} \quad v < a \\ \frac{-(1-v)\ a'}{a^2}, & \text{hvis} \quad v > 1-a \\ 0 & \text{ellers} \end{cases}, \qquad t_v = \begin{cases} \frac{1}{a}, & \text{hvis} \quad v < a \\ \frac{-1}{a}, & \text{hvis} \quad v > 1-a \\ 0, & \text{ellers} \end{cases}$$

$$t_{uu} = \begin{cases} \frac{v(2(a')^2 - a''a)}{a^3}, & \text{hvis} \quad v < a \\ \frac{(1-v)(2(a')^2 - a''a)}{a^3}, & \text{hvis} \quad v > 1-a \\ 0, & \text{ellers} \end{cases}, \quad t_{uv} = \begin{cases} \frac{-a'}{a^2}, & \text{hvis} \quad v < a \\ \frac{a'}{a^2}, & \text{hvis} \quad v > 1-a \\ 0, & \text{ellers} \end{cases} \tag{11.7}$$

og $t_{vv} = 0$.

Til slutt kan vi konstruere en blendingsfunksjon,

$$\hat{B}(u,v) = g(u)\ B \circ t(u,v), \tag{11.8}$$

hvor de partiellderiverte er

$$\begin{aligned} \hat{B}_u &= g'\ B + g\ B'\ t_u, \\ \hat{B}_v &= g\ B'\ t_v, \\ \hat{B}_{uu} &= g''\ B + 2g'\ B'\ t_u + g\ (B''\ t_u^2 + B'\ t_{uu}), \\ \hat{B}_{uv} &= g'\ B'\ t_v + g\ (B''\ t_u\ t_v + B'\ t_{uv}), \\ \hat{B}_{vv} &= g\ B''\ t_v^2. \end{aligned} \tag{11.9}$$

Et problem med blendingsfunksjonen $B(u,v)$ er at den ikke oppfører seg symmetrisk i den forstand at

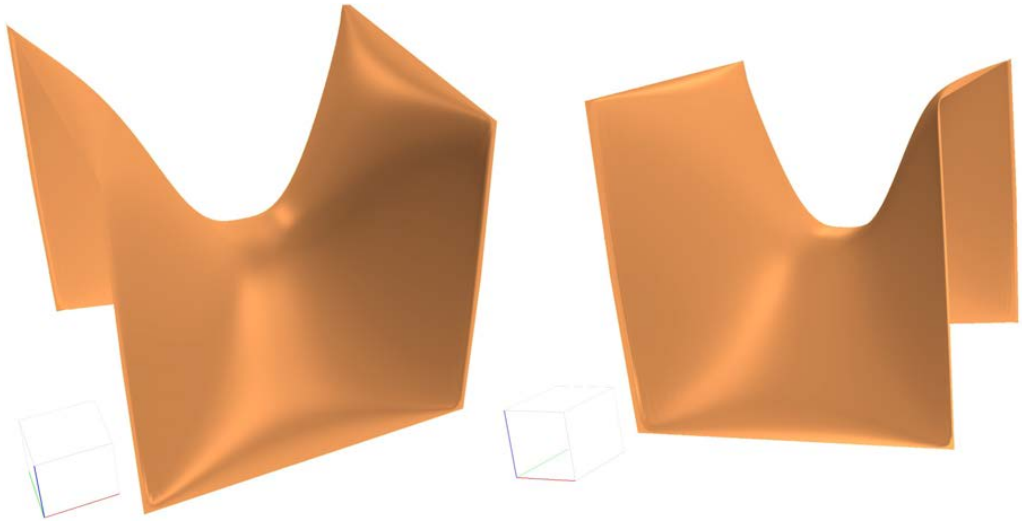
$$B(u,v) + B(v,u) = 1 \tag{11.10}$$

Man kan forvente at en blendingsfunksjon skal oppføre seg på samme måte i de to parameterretningene. For å endre en blendingsfunksjon til å oppføre seg slik, kan vi definere,

$$B(u,v) = \frac{1}{2}(1 + \hat{B}(u,v) - \hat{B}(v,u)). \tag{11.11}$$

Et plott av blendingsfunksjonene $B(u,v)$) som en flate kan sees i figur 11.1. For å få et bedre inntrykk av funksjonen vises den fra to sider. Hvis flaten snus opp ned og roteres 90°, vil det være den samme flaten.

I tillegg til å være symmetrisk i henhold til beskrevet ovenfor er $B(u,v)$ designet til å ha noen spesielle egenskaper, som for en gitt d, å være internt C^d-glatt og at alle deriverte opp



Figur 11.1: *Vi ser blendingsfunksjonen $B(u,v)$ definert i uttrykk (11.11) fra to forskjellige posisjoner. Den er fremstilt som en overflate. Det ser ut som at de deriverte langs kantene er null og at det er en slags diskontinuitet i de 4 hjørnene.*

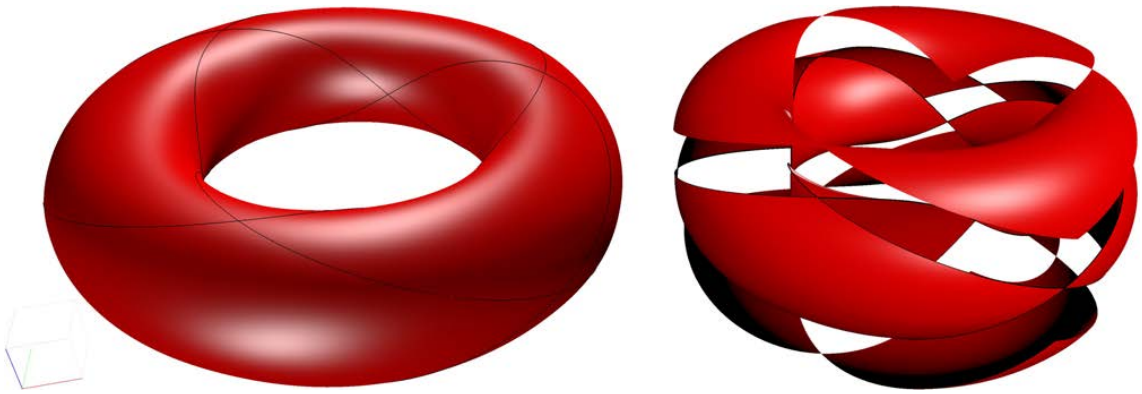
til orden d er null ved randa til domenet. Merk at B og $\hat{B}$ vil ha de samme egenskapene, så alle forslag som gjelder for $\hat{B}$ vil også gjelde for B, så vi kaller den derfor bare B. En oppsummering av egenskapene til $B(u,v)$ er:

- at verdien er 0 “internt" på to motsatte kanter (med konstant v-verdi),
- at verdien er 1 over hele de to andre kantene (med konstant u-verdi),
- den er symetrisk, dvs. $B(u,v)+B(v,u)=1$,
- den er “internt" C^d-glatt,
- at alle deriverte opp til orden d er 0 på kantene,
- og den er diskontinuerlig i alle hjørnene i “u-retningen"

I vedlegg C.9 er det et sett med bevis som viser at 2-p B-funksjonen $B(u,v)$ tilfredstiller alle disse egenskapene.

Denne 2-p B-funksjonen gjenspeiler egentlig egenskapene til en 1-p B-funksjon, i tillegg til det faktum at alle deriverte langs kantene er null i alle retninger. Imidlertid er denne 2-p B-funksjonen spesiell fordi selv om alle deriverte er kontinuerlige er selve funksjonen diskontinuerlig i hjørnene, dvs. den hopper fra 0 til 1 i “u-retningen", som påpekes av det siste punktet i lista av egenskaper.

Det siste punktet i oppsummering av egenskapene er spesielt, fordi det betyr at vi kun kan bruke $B(u,v)$ (11.11) til å blende to flater når de to flatene er like i de fire hjørnene - hvis resultatet skal bli C^d-kontinuerlig. Vi vil se nærmere på anvendelser av dette i neste seksjon.



Figur 11.2: *Til venstre ser vi et plott av 12 flater som sammen danner en C^1-glatt overflate, som er en approksimasjon av en torus. Grensekurvene som ligger på "torusen" er merket som tynne svarte kurver. De deriverte som er "ortogonale" til grensekurvene er også hentet ut fra en torus. De 12 flatene er laget ved å bruke Hermite 2-p blendingsflate. På høyre side ser vi de 12 flatene som tilsammen approksimerer en torus, og som her er flyttet litt fra hverandre slik at vi bedre kan se dem hver for seg.*

11.2 Hermite 2-p blendingsflate

Med "Coons patch" - bikubisk blending er det mulig å fylle et firkantet hull i en overflate på en glatt måte, se seksjon 9.6.2. Blendingen av to flater er imidlertid en alternativ metode. Vi starter med å lage to flater ved å bruke hermiteblending av kurver i to retninger, som i "Coons patch". Vi kaller disse $S_1(u,v)$ og $\overline{S}_2(u,v)$, og differanseflata for $\widetilde{S}(u,v) = \overline{S}_2(u,v) - S_1(u,v)$ (11.1). Vi bruker så disse i samsvar med (11.2), dvs.

$$S(u,v) = S_1(u,v) + B(u,v)\,\widetilde{S}(u,v),$$

hvor $\widetilde{S}(u,v)$ er definert i (11.1). De partiellderiverte blir da

$$\begin{aligned}
S_u &= S_{1u} + B_u\,\widetilde{S} + B\,\widetilde{S}_u,\\
S_v &= S_{1v} + B_v\,\widetilde{S} + B\,\widetilde{S}_v,\\
S_{uu} &= S_{1uu} + B_{uu}\,\widetilde{S} + 2\,B_u\,\widetilde{S}_u + B\,\widetilde{S}_{uu},\\
S_{uv} &= S_{1uv} + B_{uv}\,\widetilde{S} + B_u\,\widetilde{S}_v + B_v\,\widetilde{S}_u + B\,\widetilde{S}_{uv},\\
S_{vv} &= S_{1vv} + B_{vv}\,\widetilde{S} + 2\,B_v\,\widetilde{S}_v + B\,\widetilde{S}_{vv}.
\end{aligned} \tag{11.12}$$

Til venstre i figur 11.2 er det et plott av en approksimasjon av en torus med 12 sammenhengende flater laget av Hermite 2-p blendingsflater på samme måte som i "Coons-patch"-eksemplet i section 9.6.4. Til høyre i figuren er flatene flyttet litt fra hverandre slik at vi lettere kan se dem enkeltvis. Figur 11.2 kan sammenlignes med figur 9.15 da delflatene i begge flatene er laget på samme domener.

Metoden er ganske enkel å implementere, enklere enn for "Coons patch", men approksimasjonen gir et avvik som er litt større enn det vi så i "Coons patch"-eksemplet i Seksjon 9.6.4. I vedlegg C.10 er det et teorem som viser at det er mulig å fylle et firkantet hull med en flate slik at det totale resultatet får en ønsket grad av kontinuitet, som dermed betyr at Hermite 2-p blendingsflater kan erstatte "Coons patch".

Kapittel 12

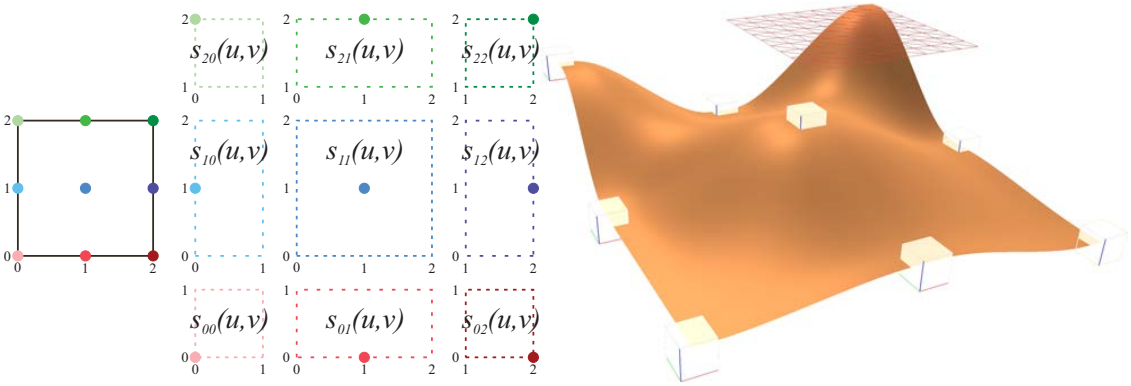
Blendingspline-tensorproduktflate

På samme måte som blendingsplinekurver, er blendingspline-tensorproduktflater 2.-ordens B-spline-tensorproduktflater med B-funksjoner, og hvor kontrollpunktene er erstattet med “lokale" flater. Vi har følgende generelle formel,

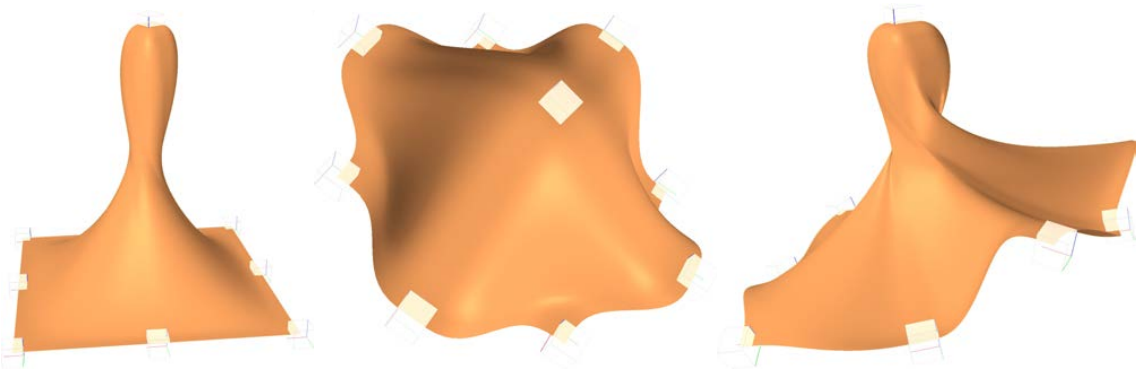
$$S(u,v) = \sum_{i=0}^{n_u-1} \sum_{j=0}^{n_v-1} s_{ij}(u,v) B_i(u) B_j(v), \tag{12.1}$$

hvor, for $i = 0,..,n_u-1,\ j = 0,..n_v-1,,\ s_{ij}(u,v)$ er lokale flater, og $B_i(u) = B \circ w_{1,i}(u)$, $B_j(v) = B \circ w_{1,j}(v)$ er B-splines med B-funksjoner, se (8.10). Formelen er i utgangspunktet den samme som formelen for vanlige polynomiske B-splineflater, bortsett fra at $s_{ij}(u,v)$ er flater istedenfor punkt. En blendingsplineflate kan derfor betraktes som en blending av “lokale flater". En åpen blendingsplineflate med $n_u \times n_v$ lokale flater kan dermed deles inn i $(n_u-1)\times(n_v-1)$ firkantede flatelapp, som hver er en blending av deler av 4 lokale flater. Det vil si de 4 lokale flatene koblet til sine respektive interpolasjonspunkt som er de fire hjørnene av flatelappen.

Et eksempel er gitt i figur 12.1. Til venstre ser vi parameterdomenet til en blendingsplineflate med $3 \times 3 = 9$ lokale flater. Flaten er en 2.-ordens B-splineflate med to “klemte"



Figur 12.1: *En blendingsplineflate med 9 lokale flater vises. Interpolasjonspunktene til venstre, i domenet, er kuber i $\mathbb{R}^3$ til høyre, og er farget lik de lokale parameterdomene i midten. Alle lokale flater er plane, en av dem er synlig.*



Figur 12.2: *Tre blendingsplineflater med samme domene, skjøtvektorer og "lokale" flater, som flaten i figur 12.1. Den eneste forskjellen er at de lokale flatene er flyttet og/eller rotert.*

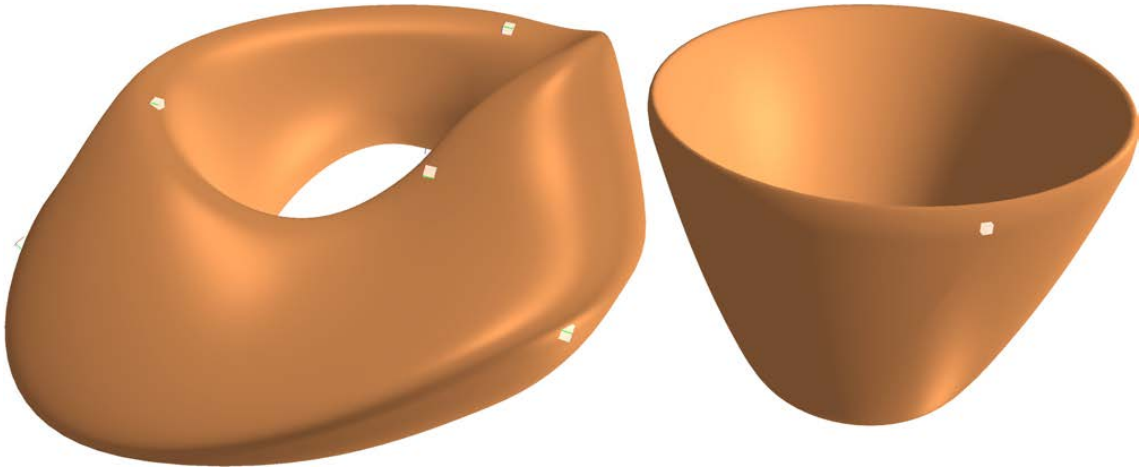
skjøtvektorer, u og v. Domenet til hver lokale flate $s_{i,j}$ er vist i midten av figur 12.1. Interpolasjonspunktene, punktene der den blendingsplineflaten (senere også kalt den globale flaten) interpolerer fullstendig (inkludert alle deriverte opp til orden lik ordenen til B-funksjonen som brukes) de lokale flatene, er plottet som kuber. Alle lokale flater i dette eksemplet er plane og parallelle med xy-planet, og på høyre side i figur 12.1 er en av de lokale flatene plottet. Skjøtvektoren **u** er $\{u_i\}_{i=0}^{4} = \{0,0,1,2,2\}$ og skjøtvektoren **v** er $\{v_i\}_{i=0}^{4} = \{0,0,1,2,2\}$. Som vi kan se til venstre i figur 12.1 dekker de lokale flatene i hjørnene bare $\frac{1}{4}$ av den globale flaten, og i dette eksempelet dekker de lokale flatene som er koblet til punktene på kantene halvparten av den globale flaten, og flaten som er koblet til midtpunktet av den globale flaten dekker hele den globale overflaten. I figur 12.2 vises tre flater med samme domene, skjøtvektorer og lokale flater, som flaten i figur 12.1. Den eneste forskjellen er at de lokale flatene er flyttet og/eller rotert. *Observasjonen er at vi nå har en B-splineflate med "kontrollpunkt" som interpolerer flaten totalt og som dermed også har en orientering, ikke bare en posisjon.*

I eksempelet i figur 12.1 er domenet $[u_1,u_3] \times [v_1,v_3]$. Dette er i samsvar med B-splines slik det er beskrevet i seksjon 6.2.2, hvor domenet for en åpen B-spline kurve er $[t_d,t_n]$, hvor $d = k-1$ er polynomgrad og n er antall kontrollpunkt. I eksemplet er $k-1 = 1$ og $n = 3$. I seksjon 8.2, er de lokale kurvene $c_i(t)$ i blandingssplinekurver definert med domenene (t_i,t_{i+2}). Det følger at for flater vil domenet til de lokale flatene $s_{i,j}(u,v)$ være $(u_i,u_{i+2}) \times (v_j,v_{j+2})$. Dette fremkommer også i skissen i midten av figur 12.1 hvor det er to like skjøtverdier i endene på begge skjøtvektorene.

12.1 Implementering av blendingsplineflater

En blendingsplineflate er en 2.-ordens B-splineflate i både u- og v-retning, og er, som vanlige B-spline-tensorproduktflater, å betraktes som en kurve (se seksjon 8.2) hvor koeffisientene er kurver, som beskrevet i section 9.5.3. Men her har de "indre kurvene" koeffisienter som er flater.

Vi vet fra seksjon 6.2.2 at B-splinekurver og flater kan være enten åpne eller lukkede i



Figur 12.3: *På venstre side ser vi en blendingsplineflate som i initialt er en kopi av en torus. Overflaten har* $3 \times 3 = 9$ *lokale flater hvor noen av dem har blitt flyttet og rotert. Flata til høyre er også en kopi av en torus, men her med* $1 \times 2 = 2$ *lokale flater. De to lokale flatene er så flyttet i motsatt retning fra hverandre.*

hver av parameterretningene. Normalt mener vi klemt når vi sier åpen. Husk at for en 2.-ordens B-spline er $k = 2$ og analogt med polynomgraden er $k - 1 = 1$. Nedenfor er en tabell hvor domenet for åpne, halvåpne og lukkede flater vises. Med halvåpen mener vi åpen i den ene parameterretningen og lukket i den andre. På høyre side av tabellen vises antall skjøtintervall, dvs. “firkantede" flatelapper.

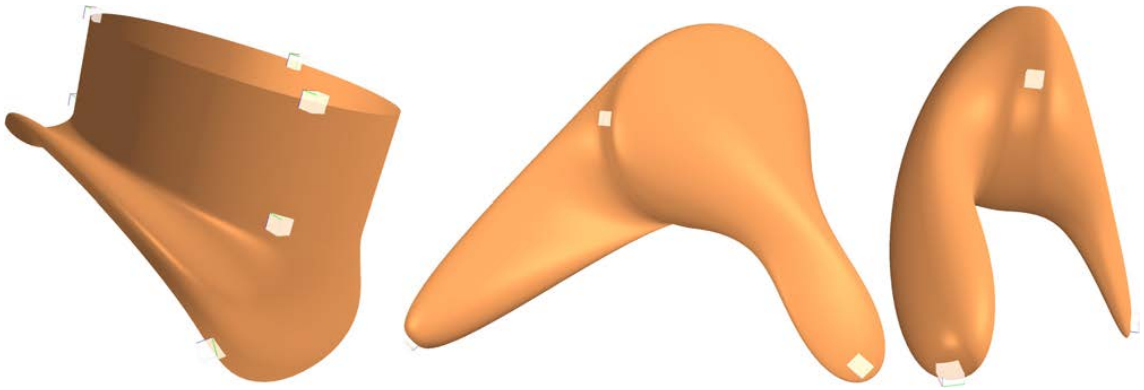
u - retning	v - retning	parameter domenet	antall flatelapper
åpen	åpen	$(u,v) \in [u_1, u_{n_u}] \times [v_1, v_{n_v}]$	$(n_u - 1) \times (n_v - 1)$
åpen	lukket	$(u,v) \in [u_1, u_{n_u}] \times [v_1, v_{n_v+1})$	$(n_u - 1) \times n_v$
lukket	åpen	$(u,v) \in [u_1, u_{n_u+1}) \times [v_1, v_{n_v}]$	$n_u \times (n_v - 1)$
lukket	lukket	$(u,v) \in [u_1, u_{n_u+1}) \times [v_1, v_{n_v+1})$	$n_u \times n_v$

Husk at en lukket skjøtvektor er en sykliskskjøtvektor, derav følger at kontinuiteten er den samme over slutt-skjøtensom over enhver annen skjøtverdi. Hvis vi bruker u-parameteren som eksempel, følger det at

$$\begin{aligned} \lim_{u \to u_{n_u+1}-} S_u^{(j)}(u,v) &= S_u^{(j)}(u_1,v), \quad \text{for} \quad j = 0,1,2,...,\mathscr{S}_u, \\ \lim_{u \to u_{n_u+1}-} S_v^{(j)}(u,v) &= S_v^{(j)}(u_1,v), \quad \text{for} \quad j = 0,1,2,...,\mathscr{S}_v, \end{aligned} \quad \text{og} \quad v \in [v_1, v_{n_v}],$$

hvor $\mathscr{S}_u$ og $\mathscr{S}_v$ er ordenen til de respektive B-funksjonene som er brukt.

I figur 12.3 ser vi to blendingsplineflater, som begge opprinnelig var kopier av en torus. De er følgelig begge lukket i begge retninger. Etter at en torusen er kopiert har de lokale flatene blitt flyttet og/eller rotert og resultatene ser vi i figur 12.3. I figur 12.4 er en sylinder og en kule først kopiert og deretter modifisert ved å flytte og rotere noen av de lokale flatene. Begge er halvåpne flater. Legg merke til at en kule enten er implementert topologisk lik en sylinder, hvor nord- og sør-polen er degenerert og hvor vi da må lage normaler med utgangspunkt i naboer, eller den kan være topologisk lik et plan der kantene kollapser til



Figur 12.4: *Til venstre ser vi en blendingsplineflate som initialt er en kopi av en sylinder men hvor 3 av 9 lokale flatene har blitt flyttet og rotert. Til høyre er det to plott av samme flate, som initialt er kopi av en kule med* $1 \times 3 = 3$ *lokale flater som så har blitt flyttet og rotert. Implementeringen vi ser er egentlig en modifisert sylinder, dvs. hvor punktene i de to endene til sylinderen er slått sammen til ett punkt i hver pol.*

ett punkt. Det første er mest vanlig. En typisk implementering er dermed en sylinder med en sammentrekning av begge de “åpne" rendene. Flaten er dermed degenerert i polene. Det er faktisk mulig å introdusere “sammentrekning" som en tilstand og på den måten håndtere degenererte poler, forhindre hull og å sikre en slags “kontinuerlig" normal (et enkelt kart for en kule er som kjent ikke mulig uten degenereringer). “Sammentrekning" er egentlig ikke en tilstand for en parameterretning, men en “sammentrekning" av alle funksjonsverdier langs en kurve i parameterplanet og hvor retningsderiverte i retningen til kurven i parameterplanet er en nullvektor. Imidlertid er en praktisk implementering en rett linje begrenset til å være parallell med en av koordinataksene i parameterplanet. For en sfære er det vanligvis en “sammentrekning" i start og sluttverdien til parameterne u eller v, mens den andre parameterretningen er lukket, dvs. syklisk.

For en lukket/syklisk parameter i en blendingsplineflate, er det i utgangspunktet bare nødvendig å ha samme antall skjøtintervall som antall lokale flater. Det vil si $n+1$ skjøtverdier, der n er lik antallet lokale flater. Men for å gjøre det konsistent med “lukkede" skjøtvektorer der vi har to like skjøtverdier ved start og slutt, legger vi en ekstra skjøtverdi først i skjøtvektoren slik at skjøtintervallet på slutten gjenspeiler begynnelsen av skjøtvektoren.

Et spørsmål er da, hva er minimum antall lokale flater vi kan ha i en parameterretning? Svaret er gitt i tabellen nedenfor.

Type av parameter	minimum antall lokale flater	størrelsen på skjøtvektoren
åpen	$n = 2$	$n+2$
lukket	$n = 1$	$n+2$

For å implementere / programmere blendingsplineflater, må en rekke spesifikke problemer løses, dette skal behandles i de følgende seksjonene.

12.2 Evaluator - beregne verdi og deriverte

Det generelle uttrykket for en blendingsplineflate er gitt ved (12.1), ie

$$S(u,v) = \sum_{j=0}^{n_v-1} \sum_{i=0}^{n_u-1} s_{i,j}(u,v)\, B_i(u)\, B_j(v), \tag{12.2}$$

Denne kan så forenkle. På grunn av den doble summen og tensorproduktkonstruksjonen, er det naturlige først å dele opp funksjonen i en indre og en ytre del, se (9.33). Den indre delen er, for $u \in [u_i, u_{i+1}]$

$$c_{i,j}(u,v) = \begin{pmatrix} 1-B_i(u) & B_i(u) \end{pmatrix} \begin{pmatrix} s_{i-1,j}(u,v) \\ s_{i,j}(u,v) \end{pmatrix}, \quad j = 0,1,...,n_v-1. \tag{12.3}$$

Det følger at formelen i (12.2) kan omformuleres ved å bruke (12.3). Vi får med det en formulering som erstatter (12.2) når $(u,v) \in [u_i, u_{i+1}] \times [v_j, v_{j+1}]$, og som er

$$S(u,v) = \begin{pmatrix} 1-B_j(v) & B_j(v) \end{pmatrix} \begin{pmatrix} c_{i,j-1}(u,v) \\ c_{i,j}(u,v) \end{pmatrix}. \tag{12.4}$$

Begge formlene (12.3) og (12.4) kan sammenlignes med kurveformelen (8.6) kombinert med (8.10). Når vi skal lage en algoritme kan vi derfor, med noen modifikasjoner, bruk samme algoritme som vi lagde for kurver.

Først, hvordan regne ut den indre delen (12.3):
Som i kurveeksemplet kan vi også her forenkle. Fra (8.7) og definisjon 8.1 får vi; For hvert skjøtintervall i v-retningen, $\Delta v_j = [v_j, v_{j+1}),\ j = 0,1,...,n$, hvor $n = n_v - 2$ for åpne flater og $n = n_v - 1$ for lukkede flater er:

$$\begin{aligned} &c_{i,j}(u_i,v) = s_{i-1,j}(u_i,v), && \text{for} \quad i = 1,2,\ldots,n_u \\ &c_{i,j}(u,v) = s_{i-1,j}(u,v) + \widehat{s}_{i,j}(u,v)B_i(u), && \text{når} \quad u_i < u < u_{i+1}, \end{aligned} \tag{12.5}$$

hvor $B_i(u) = B \circ w_{1,i}(u)$ og hvor

$$\widehat{s}_{i,j}(u,v) = s_{i,j}(u,v) - s_{i-1,j}(u,v). \tag{12.6}$$

For å beregne de partiellderiverte til (12.5) med hensyn på u, ser vi at for $u = u_i,\ i = 1,2,...,n_u$ er alle partielle derivater (med hensyn til u) lik de respektive deriverte til den lokale flaten, dvs.

$$D_u^{d_u} c_j(u_i,v) = D_u^{d_u} s_{i-1,j}(u_i,v), \quad \text{for } j = 0,...,n_v-1, \ \text{ og } \ d_u = 0,1,...,\mathscr{S}_u. \tag{12.7}$$

Så for $u_i < u < u_{i+1}$ får vi følgende formler for funksjonsverdien og de to første partiellderiverte i u retning,

$$\begin{aligned} c_{i,j}(u,v) &= s_{i-1,j}(u,v) + \widehat{s}_{i,j}(u,v)B_i(u), \\ D_u c_{i,j}(u,v) &= D_u s_{i-1,j}(u,v) + D_u\widehat{s}_{i,j}(u,v)B_i(u) + \widehat{s}_{i,j}(u,v)DB_i(u), \\ D_u^2 c_{i,j}(u,v) &= D_u^2 s_{i-1,j}(u,v) + D_u^2\widehat{s}_{i,j}(u,v)B_i(u) + 2D_u\widehat{s}_{i,j}(u,v)DB_i(u) + \widehat{s}_{i,j}(u,v)D^2B_i(u). \end{aligned} \tag{12.8}$$

Uttrykkene er i hovedsak de samme som de vi brukte for kurver (8.9). Dette er imidlertid bare en del av beregningen i den indre sløyfen. Vi må også beregne alt de partiellderiverte/kryssderiverte for $c_j(u,v)$ i v-retningen. Imidlertid er de deriverte i v-retningen enkel å beregne, fordi basisfunksjonene i (12.8) er uavhengig av v, og derfor er bare en av faktorene i alle termer avhengig av v. De deriverte av alle ledd er dermed bare summen av de deriverte av hvert ledd. Dette kan, som vi vil se, brukes til å utvide formel (12.8) til en vektor av vektorer i stedet for bare en vektor. For å utvide formelen (12.8) til også å inkludere de deriverte i v-retningen vi må først se på den første linjen i (12.8). Det følger at

$$D_v^d c_{i,j}(u,v) = D_v^d s_{i+1,j}(u,v) + D_v^d \widehat{s}_{i,j}(u,v) B_i(u), \quad \text{for } d = 1,2,...,$$

og da generelt for å beregne

$$D_u^{d_u} D_v^{d_v} c_{i,j}(u,v), \quad \text{for } d_u = 1,2,... \text{ og } d_v = 1,2,...,$$

på høyre side i uttrykket 12.8, må vi bare erstatte

$$\begin{aligned} &D_u^{d_u} s_{i+1,j}(u,v) \quad \text{med} \quad D_u^{d_u} D_v^{d_v} s_{i+1,j}(u,v) \quad \text{og} \\ &D_u^{d_u} \widehat{s}_{i,j}(u,v) \quad \text{med} \quad D_u^{d_u} D_v^{d_v} \widehat{s}_{i,j}(u,v). \end{aligned}$$

Egentlig må det totale resultatet av den indre sløyfen være en matrise av vektorer, der både matrisen og vektorene må ha samme dimensjon som “evaluatorene" fra de lokale flatene, og matrisen må inneholde posisjon og alle partiellderiverte. Vi klargjør dermed dette ved å introdusere strukturen til matrisen fra den indre løkken,

$$\mathbf{C}_{i,j,d_u,d_v}(u,v), \tag{12.9}$$

der hvert element i matrisen er en vektor $\in \mathbb{R}^n$, hvor n vanligvis er 3. Indeksen j er relatert til skjøtintervallet, og d_u angir antall deriverte i u-retningen, og d_v angir antall derivater i v-retningen. Et eksempel på en slik matrisen, hvor $d_u = 2$ og $d_v = 2$, er

$$\mathbf{C}_{i,j,2,2}(u,v) = \begin{bmatrix} c_j(u,v) & D_v c_j(u,v) & D_v^2 c_j(u,v) \\ D_u c_j(u,v) & D_u D_v c_j(u,v) & D_u D_v^2 c_j(u,v) \\ D_u^2 c_j(u,v) & D_u^2 D_v c_j(u,v) & D_u^2 D_v^2 c_j(u,v) \end{bmatrix}. \tag{12.10}$$

Notasjonen til den ekvivalente matrisen fra de lokale flatene er

$$\widetilde{\mathbf{S}}_{i,j,d_u,d_v}(u,v), \tag{12.11}$$

hvor i og j er i indeksene til den lokale patchen, og d_u og d_v angir antall derivater som skal beregnes i de respektive u- og v-retningene. Disse matrisene er også organisert som (12.10).

For å oppsummere før vi lager en algoritme: Den første kolonnen i (12.9–12.10) er lik venstre side av (12.8), og bare elementer fra den første kolonnen i (12.11) brukes på høyre side av (12.8). For å beregne de andre kolonnene må vi erstatte element fra den første kolonnen i (12.11) med respektive element fra de andre kolonnene. Konklusjonen er derfor at vi bare trenger å erstatte enkeltelement fra den første kolonnen av (12.11), det vil si på høyre side av (12.8), med de respektive radene av (12.11), for å utvide (12.8) til å returnere (12.9).

Merknad 12.1. *Det er selvfølgelig mulig og ikke uvanlig å lage en "evaluator" som kun returnerer øvre venstre trekant av matrisen (12.9). Fordelen med å gjøre det er å redusere antall beregninger og på den måten optimalisere koden. Det kan man gjøre fordi man i en del tilfeller bare trenger den øvre venstre delen av matrisen. Algoritmen som vil bli presentert her er imidertid laget for å beregne hele matrisen fordi i en del tilfeller er dette nødvendig. Det er imidertid enkelt å modifisere algoritmen til bare å beregne den øvre venstre trekanten av matrisen.*

Algoritmen blir nå ganske lik algoritme 8 for kurver. Den avhenger av en B-funksjons-evaluator og evaluatorer for lokale flater. Som for kurver, antar vi at flatene er i et euklidsk rom, normalt R^3, men det kan også være noe annet, så vektortypen er derfor betegnet som vektor, selv om en av dem er et punkt (den øvre venstre er posisjonen). Den store forskjellen fra kurver er imidlertid at algoritmen returnerer en matrise (av vektorer) i stedet for en vektor (av vektorer), og at evaluatoren for lokale flater også returnerer matriser. Vi kan nå introdusere en algoritme for den indre sløyfen til en tensorproduktflate-evaluator som beregner (12.3):

Algoritme 9. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.)*
Algoritmen lager matrisen $\mathbf{C}_{i,j,d_u,d_v}(u,v)$ definert i (12.9). Evaluator til en B-funksjon må være tilgjengelig. Dette gjelder også for lokale flater hvor evaluatorene må returnere en matrise $\widetilde{\mathbf{S}}_{i,j,d_u,d_v}(u,v)$ definert i (12.11). Skjøtvektorene $\{u_i\}_{i=0}^{n_u+1}$ og $\{v_i\}_{i=0}^{n_v+1}$ må også være til tilgjengelig. Innputt er: $u \in [u_1, u_{n_u}]$, $v \in [v_1, v_{n_v}]$ og $i_u :\backslash;\ u_{i_u} \le u < u_{i_u+1}$, $i_v :\backslash;\ v_{i_v} \le v < v_{i_v+1}$, og $d_u \in \{0,1,2,...,\mathscr{S}_u\}$ (antall derivater i u retning) og $d_v \in \{0,1,2,...,\mathscr{S}_v\}$ (antall derivater i v retning), der $\mathscr{S}_u$ og $\mathscr{S}_v$ er avhengig av B-funksjonen. Returen er en matrix⟨Vector⟩*, dvs. et sett med vektorer vanligvis i $\mathbb{R}^3$, husk likevel at "matrix[0][0]" faktisk er et punkt.*

```
matrix⟨Vector⟩ C( double u, double v, int i_u, int i_v, int d_u, int d_v )
    matrix⟨Vector⟩ C_0 = S̃_{i_u,i_v,d_u,d_v}(u,v);   // "evaluering" av første lokale flate
    if (u == u_{i_u})   return C_0;                  // returnerer kun lokal flate, se (12.5)
    Matrix⟨Vector⟩ C_1 = S̃_{i_u+1,i_v,d_u,d_v}(u,v); // "evaluering" av andre lokale flate
    vector⟨double⟩ a(d_u + 1);                       // for "Pascals trekant" - tall
    vector⟨double⟩ B = {B_i^(j)}_{j=0}^{d_u};        // verdier fra B-funksjonen, se(8.10).
    C_1 -= C_0;                                      // matrisen c_0 er nå ĉ_0, se (12.6)
    for ( int i=0; i ≤ d_u; i++ )
        a_i = 1;
        for ( int j=i-1; j > 0; j-- )
            a_j += a_{j-1};                          // lage "Pascals trekant" - tall
        for ( int j=0; j ≤ i; j++ )
            C_{0,i} += (a_j B_j) C_{1,i-j};          // "rad += skalar×rad", se (12.8)
    return C_0;
```

Merk at algoritmen oppdaterer hele rader i matrisen C_0 ved å summere skalerte rader av matrisen $\widehat{C}_1$ til hver rad i C_0. Dette følger direkte av det faktum at formlene i(12.8) utvides til å beregne ikke bare den første kolonnen i (12.10), men alle kolonner, og at her vil B-splines (med B-funksjonen) bare være avhengig av u.

Det neste er, hvordan regne ut den ytre delen (12.4):
Vi skal nå se på hvordan selve "evaluatoren" til en blendingspline-tensorproduktflater blir. Formelen i (12.4) er lik (12.3). Dermed kan vi forenkle den første delen av formel (12.4) på samme måte som det ble gjort for (12.3). Det følger at i hvert skjøtintervall i u-retningen, dvs. for $u \in [u_i, u_{i+1}),\ i = 0, 1, \ldots, n$ (hvor $n = n_u - 2$ for åpne flater og $n = n_u - 1$ for lukkede flater) får vi:

$$\begin{aligned} S(u, v_j) &= c_{i,j-1}(u, v_j), && \text{for} \quad j = 1, 2, \ldots, n_v, \\ S(u, v) &= c_{i,j-1}(u, v) + \widehat{c}_{i,j}(u, v) B_j(v), && \text{når} \quad v_j < v < v_{j+1}, \end{aligned} \tag{12.12}$$

hvor $B_j(v) = B \circ w_{1,j}(v)$,

og hvor $\widehat{c}_{i,j}(u, v) = c_{i,j}(u, v) - c_{i,j-1}(u, v)$.

Når vi ser nærmere på de partiellderiverte av (12.12) kun med hensyn på v, vil vi se at for $v = v_j,\ j = 1, ..., n$ er alle partiellderiverte av splineflaten med hensyn på v lik de respektive deriverte fra den indre sløyfen, dvs

$$D_v^{d_v} S(u, v) = D_v^{d_v} c_{i,j}(u, v), \qquad \text{for } j = 1, ..., n_u, \ \text{ og } \ d_v = 0, 1, 2, ...$$

så for $v_j < v < v_{j+1}$ får vi følgende formler for funksjonsverdien og deriverte i v-retningen,

$$\begin{aligned} S(u, v) &= c_{i,j+1}(u, v) + \widehat{c}_{i,j}(u, v) B_i(v) \\ D_v S(u, v) &= D_v c_{i,j+1}(u, v) + \widehat{c}_{i,j}(u, v) D B_j(v) + D_v \widehat{c}_{i,j}(u, v) B_j(v) \\ D_v^2 S(u, v) &= D_v^2 c_{i,j+1}(u, v) + \widehat{c}_{i,j}(u, v) D^2 B_j(v) + 2 D_v \widehat{c}_{i,j}(u, v) D B_j(v) + D_v^2 \widehat{c}_{i,j}(u, v) B_j(v). \end{aligned} \tag{12.13}$$

Ser vi nærmere på (12.13) ser vi at den er svært lik (12.8). Den indre sløyfen returnerer en matrise (12.10). "Evaluatoren", den ytre sløyfen må da returnere en matrise med samme dimensjon og struktur. Matrisen må da inneholde posisjonen og alle partiellderiverte. Vi kaller denne matrisen fra "evaluatoren" til en blendingsplineflate for

$$\mathbf{S}_{d_u, d_v}(u, v), \tag{12.14}$$

der hvert element i matrisen er en vektor $\in \mathbb{R}^n$ (det første elementer er da egentlig et punkt), hvor n er dimensjonen til det euklidske rommet flaten er i. Indeksen d_u angir antall deriverte vi ønsker i u retning, og d_v angir antall derivater vi ønsker i v retning. Et eksempel på en slik matrise, der $d_u = 2$ og $d_v = 2$, er:

$$\mathbf{S}_{2,2}(u, v) = \begin{bmatrix} S(u,v) & D_v S(u,v) & D_v^2 S(u,v) \\ D_u S(u,v) & D_u D_v S(u,v) & D_u D_v^2 S(u,v) \\ D_u^2 S(u,v) & D_u^2 D_v S(u,v) & D_u^2 D_v^2 S(u,v) \end{bmatrix}. \tag{12.15}$$

Til slutt, når vi skal lage hovedalgoritmen til en blendingsplineflate, dvs. den ytre sløyfen, ser vi at på venstre side av (12.13) har vi den første *raden*, ikke den første kolonnen i den resulterende matrisen (12.14-12.15). I tillegg, på høyre side av (12.13) finner vi bare elementer fra den første *raden* i matrisen $C_{j,d_u,dv}(u, v)$ (12.9-12.10).

Algoritmen er svært lik algoritme 8 for kurver, og da også algoritme 9 for den indre sløyfen. Den er avhengig av evaluatoren til B-funksjonen og den indre sløyfen. Som for algoritmen for den indre sløyfe, antar vi at flatene er i et euklidsk rom, f.eks. $\mathbb{R}^3$, men den kan også være et annet rom, så typen betegnes derfor som vektor, vanligvis en 3D-vektor. Den store forskjellen fra algoritmen til den indre sløyfen er at i nest siste linje i algoritmen summeres matrisekolonner i stedet for rader. Algoritmen er da en komplett evaluator, jmf. merknad 3.1.

Algoritme 10. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen beregner matrisen $\mathbf{S}_{d_u,d_v}(u,v)$ beskrevet i (12.14) og (12.15). Algoritmen forutsetter at algoritme 9 og "evaluator" til en B-funksjon er tilgjengelig. Skjøtvektorene $\{u_i\}_{i=0}^{n_u+1}$ og $\{v_i\}_{i=0}^{n_v+1}$ må også være tilgjengelig. Innputt er: $u \in [u_1, u_{n_u}]$, $v \in [v_1, v_{n_v}]$, og $d_u \in \{0,1,2,...,\mathscr{S}_u\}$ (antall deriverte i u-retningen), og $d_v \in \{0,1,2,...,\mathscr{S}_v\}$ (antall deriverte i v-retning), begrenset av ordenen til B-funksjonen som brukes. Retur er en* "matrise⟨Vector⟩"*, der Vector typisk er en 3D-vektor som samsvarer med $S(u,v)$, og hvor elementene i matrisen samsvarer med elementene i matrisen $C_{i,j,d_u,d_v}(u,v)$ beskrevet i (12.10).*

matrix⟨Vector⟩ **eval (** double *u,* double *v,* int d_u, int d_v **)**

```
int  i_u = i_u :\; u_{i_u} ≤ u < u_{i_u+1};          // Indeks til gjeldende skjøtintervall for u.
int  i_v = i_v :\; v_{i_v} ≤ v < v_{i_v+1};          // Indeks til gjeldende skjøtintervall for v.
matrix⟨Vector⟩  S_0 = C_{i_u,i_v,d_u,d_v}(u,v);      // Resultat fra den indre sløyfen - i_v.
if (v == v_{i_v})    return S_0;                     // Retur direkte fra indre sløyfe, se (12.12).
matrix⟨Vector⟩  S_1 = C_{i_u,i_v+1,d_u,d_v}(u,v);    // Resultat fra den indre sløyfen - i_v + 1.
vector⟨double⟩  a(d+1);                             // For "Pascals trekant" - tall.
vector⟨double⟩  B = {B_{i_v}^{(j)}}_{j=0}^{d_v};    // verdier fra B-funksjonen,, se (8.10).
S_0 -= S_1;                                         // C_0 er nå Ĉ_0, hele matrisen, se (12.7).
for ( int  i=0; i ≤ d_v; i++ )
    a_i = 1;
    for ( int  j=i-1; j > 0; j-- )
        a_j += a_{j-1};                             // lage "Pascals trekant" - tall.
    for ( int  j=0; j ≤ i; j++ )
        (S_1^T)_i += (a_j B_j)(S_0^T)_{i-j};        // "kolonne += skalar×kolonne", se (12.13).
return S_1;
```

Beregningskostnaden ved å evaluere en blendingspline-tensorproduktflate er å evaluere to B-funksjoner, fire lokale flater, og gå totalt tre ganger gjennom summeringsløkken i siste halvdel av både indre og ytre sløyfe. Den dyreste delen av beregningen er å evaluere de fire lokale flatene. Dette kan ta mer enn $\frac{9}{10}$ av tiden, avhengig av type lokal flate.

12.3 Bézierflater som lokale flater

Generelt er bézierflater svært anvendelige som lokale flater. Bézierflater er definert i seksjon 9.5.2, uttrykk (9.35). Det følger av bernsteinpolynomene at parameterdomenet for bézierflater er $[0,1] \times [0,1]$. Blendingsplines er 2.-ordens B-splines. Dermed må dome-

net til hver lokal flate spenne over to skjøtintervall i begge av parameterretningene. For en lokal flate $s_{i,j}(u,v)$ må parameterdomenet dermed være $[u_i, u_{i+2}] \times [v_j, v_{j+2}]$. I (8.13) er avbildningen $w_{2,i}$ og dets deriverte $\delta_{2,i}$ beskrevet. For eksempel vil $w_{2,i}(u)$ overføre $u \in [u_i, u_{i+2}]$ til $\overline{u} \in [0,1]$. Dermed får vi følgende formel når vi bruker lokale bézierflater,

$$s_{i,j}(u,v) = \sum_{r=0}^{d_u} \sum_{s=0}^{d_v} c_{r,s} \;\; b_{d_v,s} \circ w_{2,j}(v) \;\; b_{d_u,r} \circ w_{2,i}(u), \tag{12.16}$$

når $(u,v) \in [u_i, u_{i+2}] \times [v_j, v_{j+2}]$, og der basisfunksjonene er bernsteinpolynomene, og $c_{r,s} \in \mathbb{R}^n$, $n > 0$ og vanligvis 3, er koeffisientene, dvs. kontrollpunkt.

Alle typer verktøy for beregning av bézierkurver er selvfølgelig også tilgjengelige for tensorprodukt bézierflater. I seksjon 4.4.3 ble bernstein/hermitematrise introdusert, og algoritme 2 er lager denne matrisen. I bernstein/hermitematrisen skaleres hver rad med δ^j, hvor potensen j er radnummeret i matrisen (starter med 0). Dette er $\delta_{2,i}$, den deriverte av translering og skaleringsfunksjonen $w_{2,i}$. Matrisen ser dermed slik ut:

$$\mathbf{B}_d(w_{2,i}(t), \delta_{2,i}) = \begin{pmatrix} \delta_{2,i}^0 D^0 b_{d,0} \circ w_{2,i}(t) & \dots & \delta_{2,i}^0 D^0 b_{d,d} \circ w_{2,i}(t) \\ \vdots & \ddots & \vdots \\ \delta_{2,i}^d D^d b_{d,0} \circ w_{2,i}(t) & \cdots & \delta_{2,i}^d D^d b_{d,d} \circ w_{2,i}(t) \end{pmatrix}. \tag{12.17}$$

I kurvetilfellet ble matrisen brukt både for (pre-)evaluering og hermiteinterpolasjon. Det er derfor naturlig å gjøre dette også for blendingsplineflater.

Dermed, ved å bruke denne matrisen, (12.17), kan vi lage en utvidet matriseversjon av formel (12.16) som ikke bare returnerer verdien/posisjonen, men også alle partiellderiverte fra 1.- til d.-orden, der d er polynomgraden. Vi får dermed:

$$\widetilde{\mathbf{S}}_{d_u,d_v}(u,v) = \mathbf{B}_{d_u}(w_{2,i}(u), \delta_{2,i}) \;\; C \;\; \mathbf{B}_{d_v}(w_{2,j}(v), \delta_{2,j})^T \tag{12.18}$$

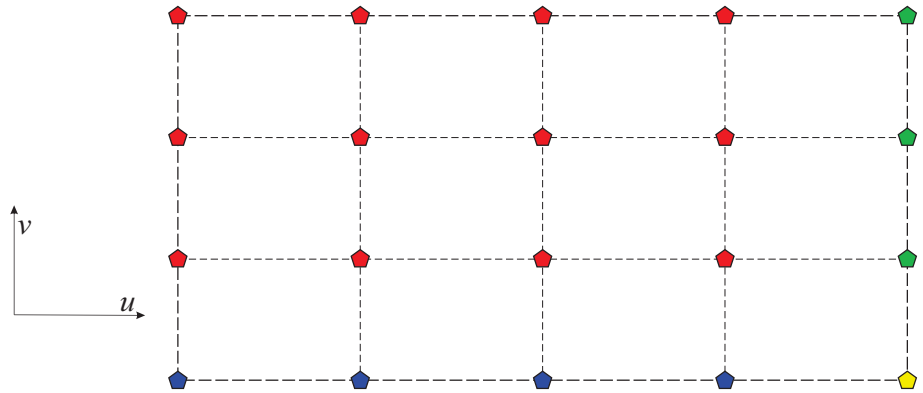
for $u_i \le u \le u_{i+2}$ og $v_j \le v \le v_{j+2}$, og hvor C er kontrollpunktene (matrisen), og $w_{2,i}(u)$, $w_{2,j}(v)$, $\delta_{2,i}$ og $\delta_{2,j}$ er fra (8.13), beskrevet i seksjon 8.2.2. Hvis $d_u = d_v = 2$ vil matrisen $\widetilde{\mathbf{S}}_{d_u,d_v}(u,v)$ bli,

$$\widetilde{\mathbf{S}}_{2,2}(u,v) = \begin{bmatrix} s(u,v) & D_v s(u,v) & D_v^2 s(u,v) \\ D_u s(u,v) & D_u D_v s(u,v) & D_u D_v^2 s(u,v) \\ D_u^2 s(u,v) & D_u^2 D_v s(u,v) & D_u^2 D_v^2 s(u,v) \end{bmatrix}. \tag{12.19}$$

Som man kan se, inneholder denne matrisen (12.19) posisjonen (elementet øverst til venstre) og alle partiellderiverte for parameterverdien (u,v) på flaten. Det følger også at denne matrisen beskriver den lokale flaten fullstendig.

12.3.1 Lokale bézierflater og hermiteinterpolasjon

Vi starter med å ta et tilbakeblikk på metoden gitt i seksjon 8.2.3 for så å tilpasse den til blendingsplineflater.



Figur 12.5: *En illustrasjon av parameterplanet til en flate $g(u,v)$, og to skjøtvektorer $\{u_i\}_{i=0}^{6}$ og $\{v_j\}_{j=0}^{5}$ som deler domenet i 12 deler. Ved alle interne skjøtverdier (u_i, v_j), $i = 1,\ldots,5$, $j = 1,\ldots,4$, er det polygon markert for å lage lokale flater med hermiteinterpolasjon. Fargene på polygonene illustrerer forskjellene mellom åpne og lukkede flater.*

1. Gitt en flate $g(u,v)$, $g : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^k$, hvor parameterdomenet $\Omega = [u_s, u_e] \times [v_s, v_e]$, og hvor $k > 0$, typisk 3.

2. Gitt også antall interpolasjonspunkt $n_u > 1$ og $n_v > 1$ og antall partiellderiverte $d_u > 0$ og $d_v > 0$ til bruk i hermiteinterpolasjonen. Merk at det i hvert enkelt interpolasjonspunkt er mulig individuelt å spesifisere antall partiellderiverte som skal brukes.

3. Vi generer så skjøtvektorene $\mathbf{u} = \{u_i\}_{i=0}^{n_u+1}$ og $\mathbf{v} = \{v_j\}_{j=0}^{n_v+1}$ med:
   – først å sette $u_1 = u_s$ og $v_1 = v_s$, dvs. starten av domenet til g i både u og v retning,
   – for så å sette $u_{n_u} = u_e$ og $v_{n_v} = v_e$, dvs. enden av domenet til g i både u og v retning.
   – Så, for $i = 2,3,\ldots,n_u - 1$ genererer vi u_i slik at $u_{i-1} < u_i$ (vanligvis uniformt fordelt), og for $j = 2,3,\ldots,n_v - 1$, generere v_j slik at $v_{j-1} < v_j$.
   – Til slutt setter vi u_0 og u_{n_u+1} og v_0 og v_{n_v+1} i henhold til reglene for "åpne/lukkede" skjøtvektorer for 2.-ordens B-splineflater.

4. Vi kan nå lage de lokale bézierflatene. Vi hermiteinterpolerer g i alle sett av interne skjøtverdier, dvs. (u_i, v_j), $i = 1,2,\ldots,n_u$, $j = 1,2,\ldots,n_v$. I figur 12.5 er interpolasjonspunktene markert med små fargede polygon. Hvis $g(u,v)$ er
   – åpne i begge parameterretninger. Da må lokale flater lages i alle polygonene,
   – åpen i u-retning og lukket i v-retning. Da må lokale flater lages i alle røde og grønne polygon, mens de blå og gule må gjenbruke de lokale flatene i øverste rad, jmf figur 12.5,
   – lukket i u-retning og åpen i v-retning. Da må lokale flater lages i alle røde og blå polygon, mens de grønne og gule må gjenbruke de lokale flatene laget langs venstre kant i figur 12.5,

– lukket inn i begge parameterretninger. Da må lokale flater lages i alle røde polygon, mens de blå må gjenbruke de lokale flatene laget langs toppen, og de grønne må gjenbruke de lokale flatene laget langs venstre kant, og den gule må gjenbruke den lokale flaten i øvre venstre hjørne.

Element 4 ovenfor forteller oss at vi må lage lokale flater med hermiteinterpolasjon. Husk at med B-funksjoner av tilstrekkelig høy orden og for $s_u = 0, ..., d_u$ og $s_v = 0, ..., d_v$ er

$$D_u^{s_u} D_v^{s_v} S(u_i, v_j) = \delta_{2,i-1,\bar{u}}^{s_u} \delta_{2,j-1,\bar{v}}^{s_v} \, D_u^{s_u} D_v^{s_v} s_{i,j} \circ (w_{2,i-1}(u_i), w_{2,j-1}(v_j)),$$

hvor $S(u,v)$ er blendingsplineflaten, $s_{i,j}(u,v)$ er for alle i, j lokale bézierflater og

$$w_{2,i-1}(u_i) = \frac{u_i - u_{i-1}}{u_{i+1} - u_{i-1}} \quad \text{og} \quad w_{2,j-1}(v_j) = \frac{v_j - v_{j-1}}{v_{j+1} - v_{j-1}},$$

er den lineære translasjons- og skaleringsfunksjonen fra definisjon 6.11 og

$$\delta_{2,i-1,\bar{u}} = \frac{1}{u_{i+1} - u_{i-1}} \quad \text{og} \quad \delta_{2,j-1,\bar{v}} = \frac{1}{v_{j+1} - v_{j-1}}$$

er den deriverte til den lineære translasjons- og skaleringsfunksjonen (6.13).

Formelen for hermiteinterpolasjon for en lokale bézierflate med indeksparet (i,j) og for $s_u = 0, ..., d_u$ og $s_v = 0, ..., d_v$ er,

$$\begin{aligned} D_u^{s_u} D_v^{s_v} g(u_i, v_j) &= D_u^{s_u} D_v^{s_v} S(u_i, v_j) \\ &= \delta_{2,i-1,\bar{u}}^{s_u} \delta_{2,j-1,\bar{v}}^{s_v} \, D_u^{s_u} D_v^{s_v} s_{i,j} \circ (w_{2,i-1}(u_i), w_{2,j-1}(v_j)) \\ &= \sum_{r=0}^{d_1} \sum_{s=0}^{d_2} c_{i,j,r,s} \; \delta_{2,i-1,\bar{u}}^{s_u} \, D^{s_u} b_{d_u,r} \circ w_{2,i-1}(u_i) \delta_{2,j-1,\bar{v}}^{s_v} \, D^{s_v} b_{d_v,s} \circ w_{2,j-1}(v_j). \end{aligned}$$

Dette er nesten det samme som formuleringen i (12.18), men vi har nå inkludert parametovergangen. Matriseformen nå er, for $i = 1, ..., m_u$, og $j = 1, ..., m_v$,

$$\mathbf{g}_{d_u,d_v}(u_i, v_j) = \mathbf{B}_{d_u}(w_{2,i-1}(u_i), \delta_{2,i-1,\bar{u}}) \; C_{i,j} \; \mathbf{B}_{d_v}(w_{2,j-1}(v_j), \delta_{2,j-1,\bar{v}})^T,$$

hvor

$$\mathbf{g}_{d_u,d_v}(u_i, v_j) = \begin{pmatrix} g(u_i, v_j) & \cdots & D_v^{d_v} g(u_i, v_j) \\ \vdots & \ddots & \vdots \\ D_u^{d_u} g(u_i, v_j) & \cdots & D_u^{d_u} D_v^{d_v} g(u_i, v_j) \end{pmatrix}.$$

Bézierflater og hermiteinterpolasjon

Det siste trinnet i genereringen av lokale bézierflatene er å snu formuleringen med hensyn på bézier-koeffisientene $\mathbf{C}_{i,j}$, kontrollpolygonet til den lokale flaten $S_{i,j}(u,v)$, dvs.

$$C_{i,j} = \mathbf{B}_{d_u}(w_{2,i-1}(u_i), \delta_{2,i-1,\bar{u}})^{-1} \; \mathbf{g}_{d_u,d_v}(u_i, v_j) \; \mathbf{B}_{d_v}(w_{2,j-1}(v_j), \delta_{2,j-1,\bar{v}})^{-T}. \quad (12.20)$$

Konklusjonen er at for å beregne koeffisienten til de lokale bézierflatene, dvs. løse (12.20), må man lage den utvidede bernstein/hermitematrisen med algoritme 2, og deretter invertere denne matrisen for så å multipliser den inverterte matrisen med matrisen fra “evaluatoren" til den opprinnelige flaten. Invertering av matriser vil ikke bli behandlet videre her,

men det er mange tilgjengelige programmeringsbibliotek med optimaliserte algoritmer for dette, se vedlegg B.

Merknad 12.2. *Merk at de tre matrisene på høyre side i 12.20 ikke er av samme "type". Den midterste, $\mathbf{g}_{d_u,d_v}(u_i,v_j)$ er en matrise av punkt/vektorer i $\mathbb{R}^n$, hvor n vanligvis er 3. Bernstein/hermitematrisen er en matrise der hvert element er en skalar. For bruk i både en flate-evaluatorer og i hermiteinterpolasjon er det derfor av stor interesse å implementere en matrise-type som har matrisemultiplikasjon som takler multiplikasjon mellom en matrise av skalarer og en matrise av vektorer/punkt. I template-programmering må da $*$ operasjonen mellom matrisene returnere en matrise av samme type som $*$ operasjonen mellom elementene i den første matrisen og elementene i den andre matrisen returnerer.*

Som i kurvetilfellet er det flere grunner til at det er fordelaktig å flytte koeffisientene i en lokal bézierflate kollektivt slik at interpolasjonspunktet blir det lokale origo. Da følger det at vi må trekke punktet $g(u_i,v_j)$ fra alle koeffisientene i kontrollpolygonet $C_{i,j}$, og at vi må motvirke dette med å sette inn den motsatte forflytningen i den høyre kolonnen i den homogene matrisen som bestemmer det lokale koordinatsystemet til bézierflaten. .

12.3.2 Eksempler på hermiteinterpolasjoner

I denne seksjonen skal vi se på eksempler på hermiteinterpolasjon av noen kjente parametriske flater ved blendingsplineflater som har lokale bézierflater. I noen av eksemplene vil også se noen av de lokale bézierflatene være plottet, og vi skal se nærmere på hvilke effekter hermiteinterpolasjonen gir.

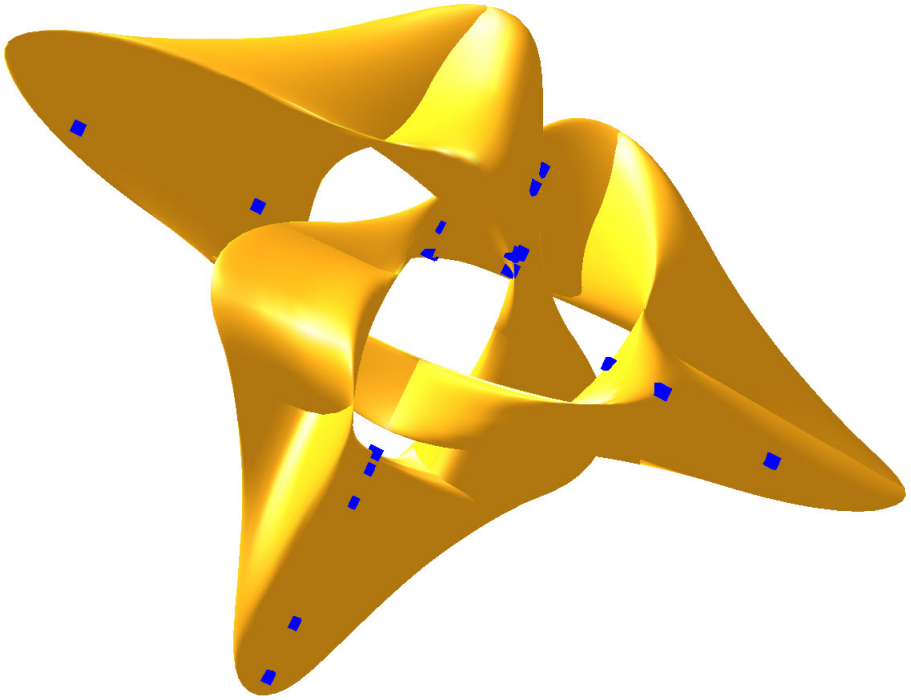
Det første eksemplet er basert på en flate kalt "Trianguloid Trefoil". Denne flaten ble lansert av Roger Bagula og kan finnes på [4]. Formelen er

$$s(u,v) = \begin{pmatrix} 2\frac{\sin(3u)}{2+\cos v} \\ 2\frac{\sin u+2\sin(2u)}{2+\cos\left(v+\frac{2}{3}\pi\right)} \\ \frac{(\cos u-2\cos(2u))(2+\cos v)\left(2+\cos\left(v+\frac{2}{3}\pi\right)\right)}{4} \end{pmatrix} \quad \begin{array}{l} \text{der} \quad u\in(-\pi,\pi], \\ \text{og} \quad v\in(-\pi,\pi]. \end{array} \tag{12.21}$$

Figur 12.6 viser et plott av en blendingsplineflate som interpolerer en "'Trianguloid Trefoil"-flate (12.21) i 5×5 punkt. I hvert punkt er posisjonen og 8 partiellderiverte brukt, dvs. matrisen $\mathbf{g}_{d_u,d_v}(u_i,v_j)$ definert i (12.17) har dimensjon 3×3. I figur 12.6 er interpolasjonspunkt markert som blå kuber, og de fleste av dem er synlige. Flaten er "lukket" i begge parameterretninger, og er ganske kompleks i formen, men rekonstruksjonen har holdt strukturen og formen svært lik orginalen.

Det andre eksemplet er basert på en flate kalt "Bent Horns". Den er også laget av Roger Bagula og man kan finne den på [5]. Formelen for denne flaten er

$$s(u,v) = \begin{pmatrix} (2+\cos u)\left(\frac{v}{3}-\sin v\right) \\ \left(2+\cos\left(u-\frac{2}{3}\pi\right)\right)(\cos v-1) \\ \left(2+\cos\left(u+\frac{2}{3}\pi\right)\right)(\cos v-1) \end{pmatrix} \quad \begin{array}{l} \text{der} \quad u\in(-\pi,\pi], \\ \text{og} \quad v\in(-2\pi,2\pi]. \end{array} \tag{12.22}$$



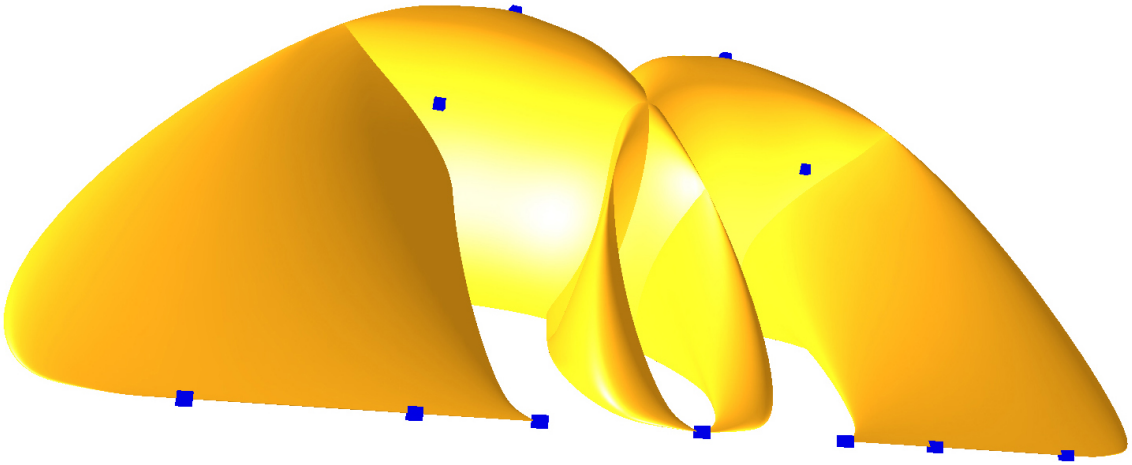
Figur 12.6: *Denne blendingsplineflaten er laget ved hermiteinterpolering av en "Trianguloid Trefoil"-flate [4] med å bruke* 5×5 *interpolasjonspunkt. Interpolasjonspunktene kan sees som blåe terninger.*

I figur 12.7 er det et plott av en blendingsplineflate som interpolerer en "Bent Horns"-flate (12.22) i 5×5-punkt. Også i dette eksemplet har vi brukt posisjonen og 8 partiellderiverte i hvert punkt, dvs. matrisen $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ definert i (12.17) har dimensjon 3×3. I figur 12.7 er interpolasjonspunktene markert som blå kuber, og noen av dem kan sees tydelig. Flaten er "lukket" i en parameterretning, men "åpen" i den andre. Dette sees ikke klart, fordi de to "åpne" endene er klemt sammen i to respektive kanter. De kan sees foran på hver side i figuren. På hver av dem er tre blå kuber markert. Disse kubene er egentlig 5 kuber, en på spissen mot midten, og to sammenfallende i hver av de to andre. Flaten er også uregelmessig i midten, hvor den kollapser til et punkt. Blendingsplineflaten har klart å beholde denne uregelmessigheten intakt fordi det er et interpolasjonspunkt i samme posisjonen, dvs. i parameterplanet er det faktisk 5 forskjellige punkt som gir samme punkt i rommet.

Det tredje eksemplet er basert på en sfære (kule), der formelen er,

$$s(u,v) = \begin{pmatrix} r\cos u\ \cos v \\ r\sin u\ \cos v \\ r\sin v \end{pmatrix} \qquad \begin{array}{l} \text{der} \quad u \in (0, 2\pi], \\ \text{og} \quad v \in (-\frac{\pi}{2}, \frac{\pi}{2}]. \end{array} \tag{12.23}$$

I formelen er r radiusen til kulen. Figur 12.8 viser 3 plott av en blendingsplineflate som interpolerer en kule (12.23) i 4×4 punkt. Også i dette eksemplet er posisjonen og 8 partiellderiverte brukt i hvert punkt, dvs. matrisen $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ definert i (12.17) har dimensjon 3×3. I figur 12.8 er interpolasjonspunktene markert som blå kuber. På toppen av "sfæ-



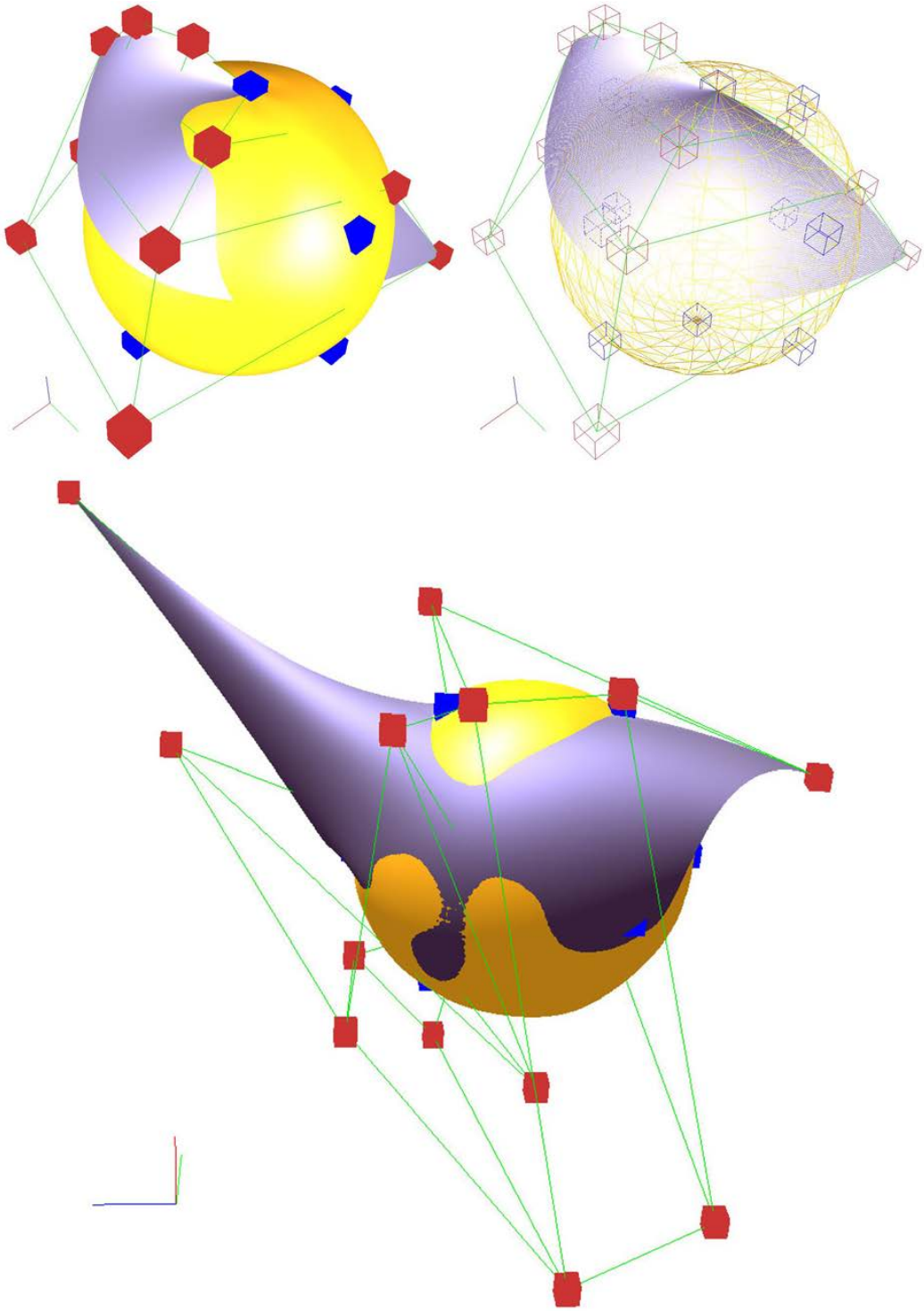
Figur 12.7: *Denne blendingsplineflaten er laget ved å hermiteinterpolere (posisjon, 1.- og 2.- deriverte) en "Bent Horns"-flate [5] i 5×5 punkt. Interpoleringspunktene er markert som blå terninger.*

ren" kan vi se en kube, mens det faktisk er 4 kuber i samme posisjon. Flaten er derfor uregelmessig, og kollapser til et punkt ved begge polene. I den øvre delen av figuren 12.8 er det et plott som inkluderer en av de lokale bézierflatene som ligger på "nordpolen". Det er en solid kule til venstre, og en trådmodell til høyre. Som vi kan se, spesielt i trådmodellen, har den lokale flaten bare to hjørner. De to andre hjørnene har kollapset til ett punkt og ligger på den tilsynelatende "glatte" kanten som går igjennom "nordpolen". Også kanten mellom de to hjørnene har kollapset til punktet. Blendingsplineflate har 8 lokale bézierflater, som alle er knyttet til de to polene. De er alle like i form sammenlignet med bézierflaten vist i figuren (men rotert og/eller translert). På den nedre delen av figur 12.8 er det et annet plott av den approksimerte "sfæren", denne gangen sammen med en av de lokale bézierflatene som ikke er koblet til polene. Figuren er rotert til venstre, slik at "nordpolen" er på venstre side. Den lokale flaten ser ganske kompleks ut, og muligens ulik det mange vil forvente. Det er totalt 16 lokale bézierflater knyttet til blendingsplineflaten, og 8 av de lokale flatene har samme form som denne.

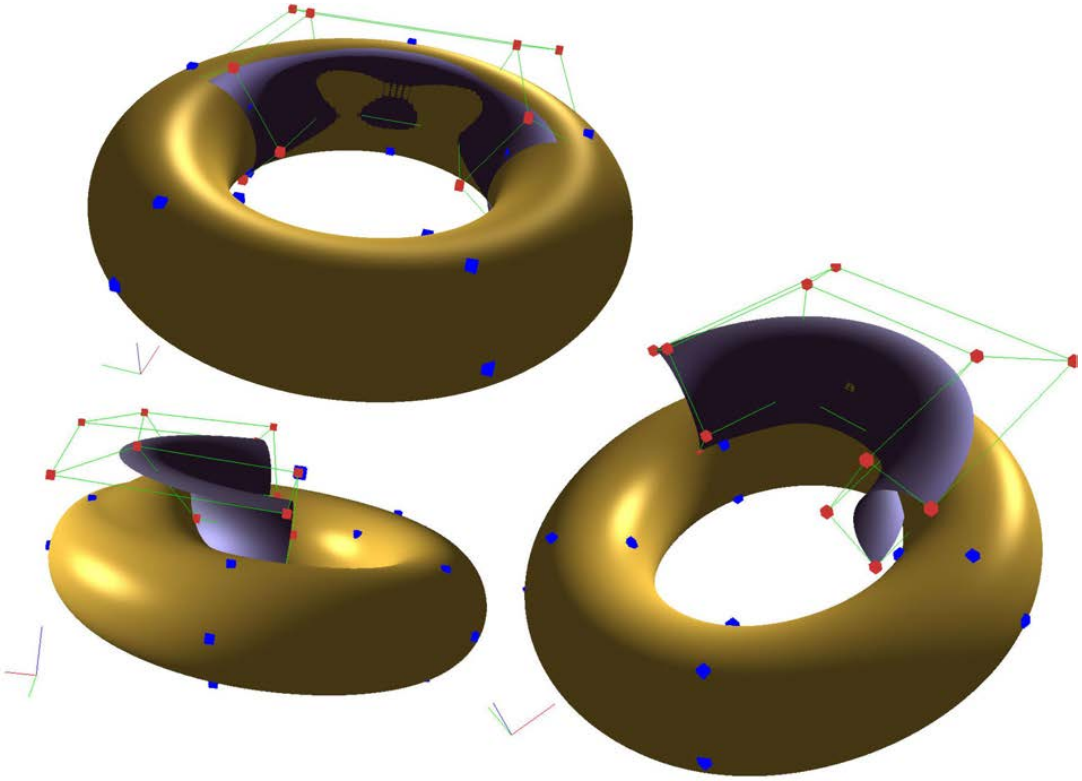
Det fjerde eksemplet er hermiteinterpolasjon av en torus hvor formelen er

$$s(u,v) = \begin{pmatrix} \cos u(R + r\cos v) \\ \sin u(R + r\cos v) \\ r\sin v \end{pmatrix} \quad \begin{array}{l} \text{der} \quad u \in (0, 2\pi], \\ \text{og} \quad v \in (0, 2\pi]. \end{array} \tag{12.24}$$

Her er r den lille radiusen, radiusen til røret, og R er den store radien, avstanden fra midten av røret til midten av røret. Den øvre delen av figur 12.9 viser en blendingsplineflate som interpolerer en torus (12.24) i 5×5 punkt. Som i de foregående eksemplene brukes posisjonen og 8 partiellderiverte i hvert punkt, dvs. matrisen $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ definert i (12.17) har dimensjon 3×3. I figur 12.9 er interpolasjonspunktene markert som blå kuber. En av de lokale bézierflatene er også plottet. Denne flaten modellerer en del av en torus ganske godt. Nederst i figuren er den lokale bézierflaten flyttet litt oppover, slik at vi kan se den klarere. Der er flatene plottet fra to forskjellige posisjoner, slik at det er mulig for oss å se



Figur 12.8: *Vi ser tre plott av en approksimert kule fra uttrykket (12.23). Øverst visen "kulen" med en av de lokale bézierflatene plassert på "nordpolen". Kontrollpolygonet til bézierflaten er markert som grønne linjer, og nodene er markert som røde kuber. Oppe til høyre vises en trådmodell av flata som er plottet på venstre side. Under vises en annen lokal bézierflate. Dette er en av flatene som ikke er plassert på polene. Alle de 8 lokale flatene som ikke er plassert ved nord- og sydpolen vil ha denne formen, men translatert og rotert.*



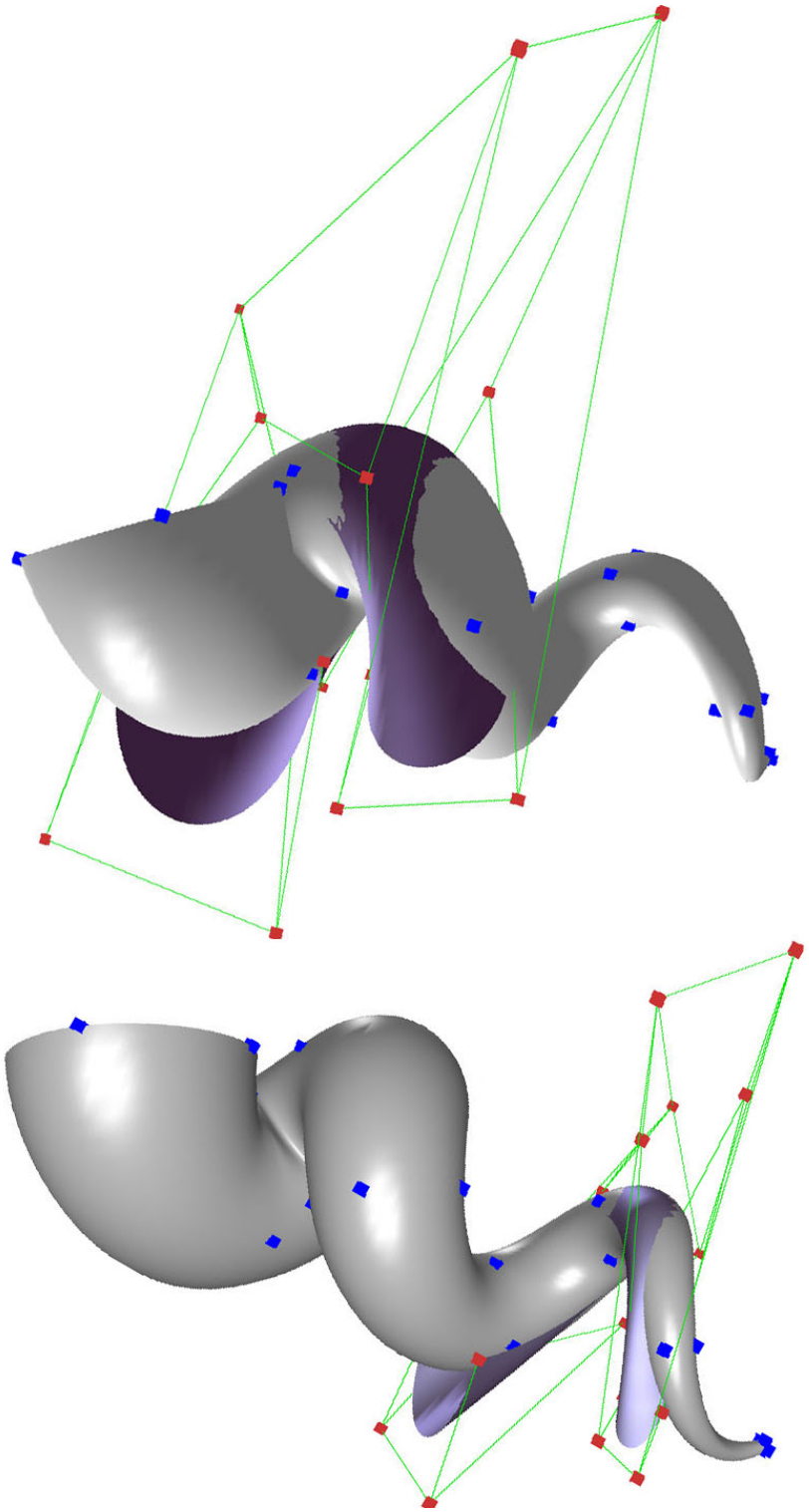
Figur 12.9: *Et plott av den approksimerte torusen fra (12.24). En av de lokale bézierflatene er også plottet. Også kontrollpolygonet til den lokale bézierflaten er plottet i grønt, mens kontrollpunktene er plottet som røde kuber. Nederst i figuren er den lokale bézierflaten flyttet litt oppover.*

den lokale flaten bedre. Som vi kan se til venstre følger blendingsplineflaten den lokale flaten når den flyttes, og endrer dermed formen til "torusen".

Det siste eksemplet er en flate kalt "sjøskjell", beskrevet av blant andre av Paul Bourke på [16]. Formelen for denne flaten er

$$s(u,v) = \begin{pmatrix} \cos v + \frac{v}{10}\left(\cos u \cos v + a \cos u \sin v\right) \\ \sin v + \frac{v}{10}\left(\cos u \sin v - a \cos u \cos v\right) \\ \left(b \sin u + \frac{6}{10}\right) v \end{pmatrix} \quad \begin{array}{l} \text{der} \quad u \in (0, 2\pi], \\ \text{og} \quad v \in (\frac{\pi}{4}, 5\pi]. \end{array} \tag{12.25}$$

I figur 12.10 vises et plott av en blendingsplineflate som interpolerer en "sjøskjell"-flate (12.25) med 4×8 punkt. I hvert punkt er posisjonen og 8 partiellderiverte brukt, dvs. matrisen $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ definert i (12.17) har dimensjonen 3×3 Interpolasjonspunktene er markert som blå kuber , og de fleste av dem er synlige. Som man kan se er flaten "lukket" i den ene parameterretningen, men "åpen" i den andre parameterretningen. På figuren kan flaten sees fra to forskjellige posisjoner. I begge plottene er også en av de lokale bézierflatene plottet, og den kan sees farget i lilla. Kontrollpolygonene til de lokale bézierflatene er også plottet. De kan sees som grønne nett, og deres kontrollpunkt er plottet som røde kuber.



Figur 12.10: *To plott av en blendingsplineflate laget ved å interpolere en "sjøskjell"-flate uttrykt i (12.25). De to plottene er sett fra forskjellige posisjoner. I hver av de to plottene er en av de lokale bézierflater også plottet. I det øvre plottet er en av de større lokale flatene plottet. I den nedre er en mindre lokal flate plottet. Vi kan også se kontrollpolygonet til begge de lokale bézierflatene. Den lokale flatene modellerer den globale flaten ganske godt lokalt.*

12.4 En delflatekonstruksjon

Som i kurvetilfellet kan vi konvertere en hvilken som helst parametrisk flate $\varphi(u,v)$ til en blendingsplineflate ved å legge til to skjøtvektorer, $\{u_i\}_{i=0}^{n_u+1}$ og $\{v_j\}_{j=0}^{n_v+1}$. Med det får vi et sett med overlappende delflater, som hver er originalflaten kun begrenset av en reduksjon av parameterdomene som hver dekker 2×2 skjøtintervall. Å bruke delflater som lokale flater betyr at en blendingsplineflate-kopi i utgangspunktet er identisk med originalflaten. Dette er fordi blending av en flate med seg selv gir selve flaten. Når man legger til affine transformasjoner (seksjon 8.2.1), vil en del-flate-algoritme i utgangspunktet være lik algoritmen for lokale bézierflater. I likhet med seksjon 12.2, og uttrykk (12.3), får vi for $i = 1,1,\ldots,n_u$ og $j = 1,1,\ldots,n_v$ og $(u,v)\in[u_i,u_{i+1}]\times[v_j,v_{j+1}]$,

$$\begin{aligned} C_{i,j}(u,v) &= \begin{pmatrix} 1-B_i(u) & B_i(u) \end{pmatrix} \begin{pmatrix} A_{i-1,j}\ \varphi(u,v) \\ A_{i,j}\ \varphi(u,v) \end{pmatrix} \\ &= \left(A_{i-1,j} + B_i(u)\ \Delta^i A_{i-1,j}\right)\varphi(u,v), \end{aligned} \tag{12.26}$$

der $B_i(u) = B\circ w_{1,i}(u)$, $\varphi(u,v)$ er orginalflaten, $A_{i,j}$ er homogene matriser beskrevet i seksjon 8.2.1 og $\Delta^i A_{i-1,j} = A_{i,j} - A_{i-1,j}$. Neste trinn er å omformulere (12.4) til

$$\begin{aligned} S(u,v) &= \begin{pmatrix} 1-B_j(v) & B_j(v) \end{pmatrix} \begin{pmatrix} C_{i,j-1}(u,v) \\ C_{i,j}(u,v) \end{pmatrix} \\ &= C_{i,j-1}(u,v) + B_j(v)\ \Delta^j C_{i,j-1}(u,v) \end{aligned} \tag{12.27}$$

hvor $\Delta^j C_{i,j-1}(u,v) = C_{i,j}(u,v) - C_{i,j-1}(u,v)$. Hvis vi setter (12.26) inn i (12.27) får vi:

En delflatekonstruksjon

$$S(u,v) = \mathbb{A}_{i-1,j-1}(u,v)\ \ \varphi(u,v) \tag{12.28}$$

hvor

$$\mathbb{A}_{i,j}(u,v) = A_{i,j} + B_{i-1}(u)\ \Delta^i A_{i,j} + B_{j-1}(v)\ \left(\Delta^j A_{i,j} + B_{i-1}(u)\ \Delta^{ij} A_{i,j}\right) \tag{12.29}$$

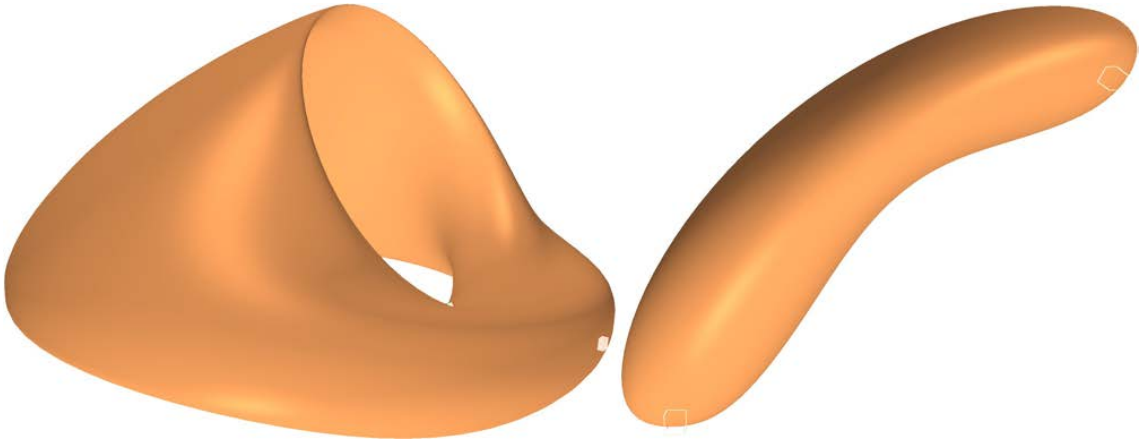
Delta-notasjonen i (12.29) kan ekspanderes på følgende måte

$$\begin{aligned} \Delta^i A_{i,j} &= A_{i+1,j} - A_{i,j} \\ \Delta^j A_{i,j} &= A_{i,j+1} - A_{i,j} \\ \Delta^{ij} A_{i,j} &= A_{i+1,j+1} - A_{i,j+1} - A_{i+1,j} + A_{i,j}. \end{aligned}$$

Alle flater vist i figurene 12.1, 12.2, 12.3, 12.4 og 12.11 er blendingsplineflater basert på delflater og dermed laget ved å bruke uttrykk (12.28) og (12.29).

12.5 Eksempler, formgiving med blendingsplineflater

I “Computer Aided Geometric Design" er objekt-formgiving et stor oppgave både i konstruksjoner for den virkelige verden (produkter), og design for virtuelle verdener (filmer, dataspill, VR/AR, etc.). Når det gjelder skulpturering, introduserte Barr så tidlig som i 1984 operasjoner for vridning, strekking, bøying og smalne flater rundt en sentral akse

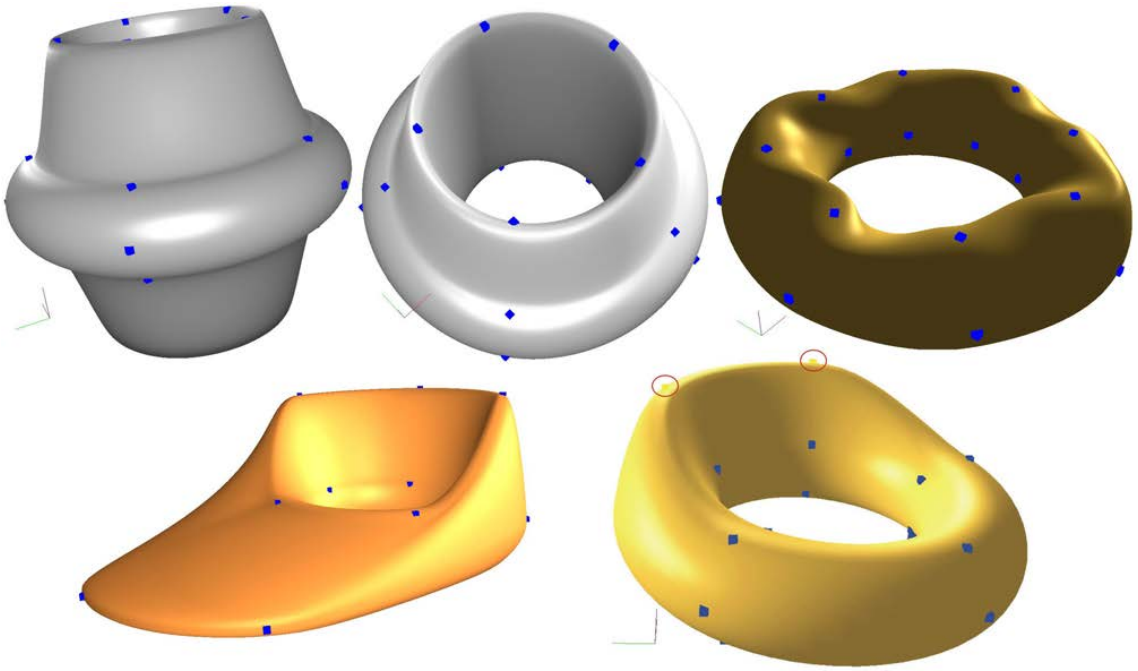


Figur 12.11: *To eksempler på blendingsplineflater fra en delflatekonstruksjon. På venstre side er en torus kopiert ved bare å bruke 2 lokale flater, den lokale flaten knyttet til interpolasjonspunktet inne i torusen er rotert* 60°*. Til høyre i figuren er en kule kopiert. Også her med kun bruk av 2 lokale flater. Disse 2 flatene er deretter flyttet fra hverandre og litt rotert.*

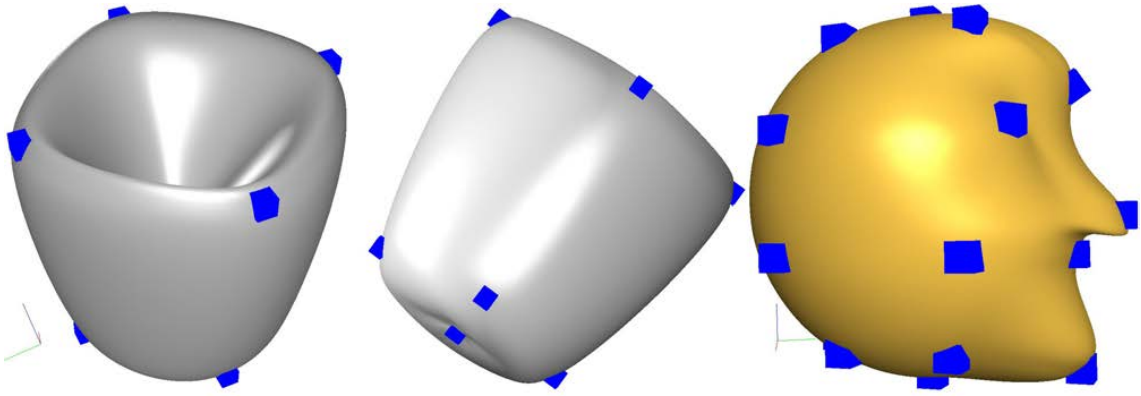
[8]. Dette ble fulgt av Sederberg og Perry som i 1996 introduserte en mer generell teknikk [145], kalt den frie formdeformasjonsmetoden, FFD. Denne metoden bygger objekter som skal deformeres inn i et 3D-gitter av kontrollpunkt som definerer et trevariabelt béziervolum. Deformasjoner kan dermed gjøres ved å deformere dette 3D-kontrollpolygonen og evaluere béziervolumet for å finne den nye posisjonen for de innebygde objektene. I de påfølgende årene har det blitt gjort en enorm mengde arbeid på dette området, ved bruk av mekanisk teknikk, flernivårepresentasjoner osv. (Eksempler på artikler er [27] [9] [84] [167]). Generelt har alle artikkler så langt slitt med den geometriske representasjonen. Blendingsplines tilbyr dermed en sterkere “formgivings"-funksjonalitet fordi den er en B-splines der alle typer affine operasjoner kan brukes på “kontrollpunktene".

I dette kapittelet har vi mange figurer som illustrerer mulighetene for kreativ design. Tidlig i dette kapittelet finner vi 4 figurer hvor alle eksemplene/plottene er basert på en delflatekonstruksjon. I figur 12.1 og 12.2 er en blendingsplineflate (et plan) deformert på forskjellige måter. Alle har 3×3 interpolasjonspunkt. I figur 12.3 er det en torus med 9 lokale flater og en med 2. Begge er deformerte, og vi kan se at antall interpolasjonspunkt er viktige for formmulighetene. I figur 12.4 ser vi en deformert sylinder, og et plott av en deformert kule sett fra to forskjellige posisjoner.

Figur 12.11 er også basert på en delflatekonstruksjon, og viser en torus- og en kule-kopi, begge med kun 2 lokale flater. Og i figur 12.12 er en torus endret på fire forskjellige måter. Den øverst til venstre er plottet fra 2 forskjellige posisjoner. De lokale flatene er bézierflater. I figur 12.13 er en kule endret til et krus, og en til et hode.



Figur 12.12: *Fire skulpturerte objekt, som alle i utgangspunktet var en torus-kopi. De er så redigert ved å flytte interpolasjonspunkt, bortsett fra objektet øverst til høyre, hvor punktene kun er rotert. Alle flater er blendingsplineflater med lokale bézierflater. Flaten øverst til høyre har* 8×4 *lokale flater, de andre har* 5×5.



Figur 12.13: *Til venstre er en kule endret til et krus bare ved å flytte interpolasjonspunktene. Objektet vises fra 2 forskjellige posisjoner. Til høyre i figuren kan du se et "hode". Det er en blendingsplineflate laget med lokale bézierflater ved først å interpolere en kule i* 6×6 *punkt (posisjon,* 1*te- og* 2*re-deriverte). Deretter flyttes noen av interpolasjonspunktene (fem av de blå kubene). Noen av dem, tre punkt ved øynene og nesen, er også rotert.*

12.6 T-kryss og Stjernekryss

Innen produktutvikling og design brukes solid-modellering hovedsakelig som verktøy. Rand-representasjon, forkortet B-rep, er en metode for å representere 3D-objekt (volum). Overflaten til et objekt og grenser mellom ulike materialer i et objekt er da en samling av sammenhengende flateelementer. Derfor er det viktig å kunne modellere komplekse flater, dvs. modellere flater av ulik opologi og med varierende grad av kompleksitet. Det betyr at vi må kunne håndtere “irregulær" flategeometri. Tradisjonelt har dette hovedsakelig vært gjort med trimmede flater med trimmekurver som er laget med boolske operasjoner og dermed flate/flate-skjæringer, se [119] og [90].

En viktig grunn til å håndtere komplisert flategeometri uten å bruke trimming er introduksjonen av isogeometrisk analyse som da er å erstatte de tradisjonelle “Finite Element"-funksjonsrommene med spline-rom, for med det å ha samme funksjonsrom (sett med basisfunksjoner) for både beregninger og formbeskrivelse, se [28].

Både T-splines, [144] og [143], LR-splines, [51], [95] og [128], og PHT-splines [158], er flatebeskrivelser som modifiserer B-splines til å håndtere noen former for uregelmessigheter. Derfor ble en beslektet løsning for å håndtere uregelmessigheter ved bruk av blendingsplineflater av tensorprodukttype beskrevet i [110]. Dette innebærer å introdusere T-kryss og stjernekryss, sammen med utvidede lokale flater som igjen er delt inn i delflater og noen ganger med re-parametrisering. Dette for å sikre kontinuitet (og glatthet) over skjøtene mellom ulike tensorproduktflater.

12.6.1 Avhengigheter av vertekser og indre kanter

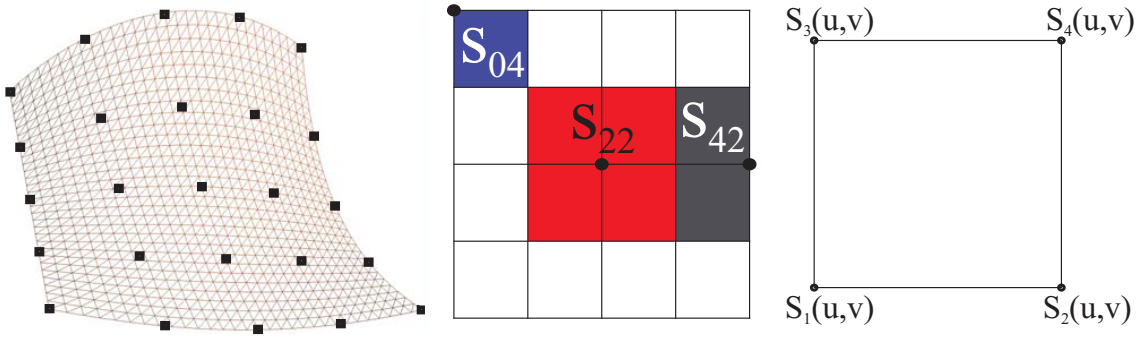
Parameterdomenet til en blendingsplineflate er partisjonert (delt opp) av skjøtvektorene, se figur 12.14. I skjæringspunktet mellom parameterlinjene har vi verteksene. Dette er punktene der de lokale flatene interpolerer den globale flaten. Det følger at domenet til de lokale flatene er partisjonene i domenet som omgir den aktuelle verteksen. For alle indre vertekser er dette fire kvadratiske partisjoner, for vertekser på kantene er det to partisjoner, og for verteksene i hjørnene er det en partisjon. Alt dette er tydelig illustrert i figur 12.14, men også skissert i figur 12.1.

I hver partisjon har vi delflater av de fire lokale flatene som dekker denne partisjonen. For å forenkle betrakter vi det lokale domenet til hver underflate som en enhetskvadrat $[0,1]\times[0,1]$. Formelen for en flate med dette domenet, lages ved å blande deler av fire lokale flater, hver koblet til ett av de fire hjørnene, jmf. høyre side i figur 12.14, er:

$$\begin{aligned} S(u,v) &= (1-B(v))((1-B(u))s_1(u,v)+B(u)s_2(u,v)) \\ &\quad +B(v)((1-B(u))s_3(u,v)+B(u)s_4(u,v)), \\ &= s_1(u,v)+B(u)(s_2(u,v)-s_1(u,v))+B(v)(s_3(u,v)-s_1(u,v)) \\ &\quad +B(u)B(v)(s_4(u,v)-s_3(u,v)-s_2(u,v)+s_1(u,v)). \end{aligned}$$

Vi hopper nå over parameterne (u,v) i flatebeskrivelsene. Det forenklede uttrykket blir

$$S = s_1+B(u)(s_2-s_1)+B(v)(s_3-s_1)+B(u)B(v)(s_4-s_3-s_2+s_1) \tag{12.30}$$



Figur 12.14: *Til venstre er en blendingsplineflate laget av et nett av 5×5 lokale flater. I midten er parameterplanet til flaten, der rutenettet vises. Tre eksempler på domener til lokale flater er illustrert. I et hjørne er flaten S_{04} i blått, i midten er S_{22} i rødt, og på venstre kant er S_{42} vist i svart. Til høyre i figuren ser vi èn partisjon, og de fire lokale flatene som er koblet til hvert sitt hjørne er markert.*

Vi undersøker så kanten bare på venstre side. De andre kantene vil vi ha lignende oppførsel. Vi har

$$S(0,v) = s_1 + B(v)(s_3 - s_1), \tag{12.31}$$

og 1.- og 2.-ordens partiellderiverte er

$$\begin{aligned}
S_u(0,v) &= s_{1u} + B(v)(s_{3u} - s_{1u}),\\
S_v(0,v) &= s_{1v} + B(v)(s_{3v} - s_{1v}) + B'(v)(s_3 - s_1),\\
S_{uu}(0,v) &= s_{1uu} + B(v)(s_{3uu} - s_{1uu}),\\
S_{uv}(0,v) &= s_{1uv} + B'(v)(s_{3u} - s_{1u}) + B(v)(s_{3uv} - s_{1uv}),\\
S_{vv}(0,v) &= s_{1vv} + 2B'(v)(s_{3v} - s_{1v}) + B(v)(s_{3vv} - s_{1vv}).
\end{aligned} \tag{12.32}$$

Følgende lemma viser interpolasjonsegenskapene på randa til en “flatelapp" (som ikke har interne skjøter) og som er laget ved å blende fire lokale flater koblet til hvert sitt hjørne. Dermed arver blendingsplineflaten oppførselen langs randa fra sine lokale flater.

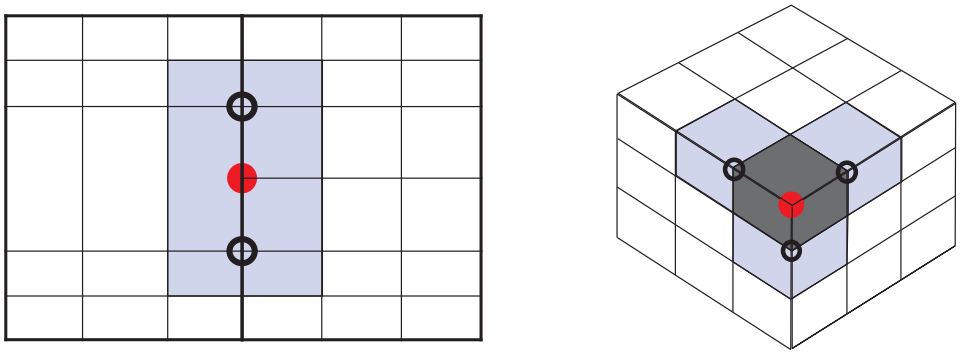
Lemma 12.1. *I de fire hjørnene har vi følgende egenskaper*

nedre venstre hjørne	$S(0,0)$	$S \equiv s_1$ *inkluderer alle dens deriverte*
nedre høyre hjørne	$S(1,0)$	$S \equiv s_2$ *inkluderer alle dens deriverte*
øvre venstre hjørne	$S(0,1)$	$S \equiv s_3$ *inkluderer alle dens deriverte*
øvre høyre hjørne	$S(1,1)$	$S \equiv s_4$ *inkluderer alle dens deriverte*

På de fire kantene har vi følgende egenskaper

venstre kant	$S(0,v) = s_1 + B(v)(s_3 - s_1)$	*bare avhengig av* s_1 *og* s_3
høyre kant	$S(1,v) = s_2 + B(v)(s_4 - s_2)$	*bare avhengig av* s_2 *og* s_4
nedre kant	$S(u,0) = s_1 + B(u)(s_2 - s_1)$	*bare avhengig av* s_1 *og* s_2
øvre kant	$S(u,1) = s_3 + B(u)(s_4 - s_3)$	*bare avhengig av* s_3 *og* s_4

Bevis. Det følger av B-funksjonens egenskaper og (12.30), (12.31) og (12.32). □



Figur 12.15: *Til venstre vises et T-kryss (rødt) med to termineringspunktene som er markert. De syv delområdene som er involvert i den "irregulære" blendingen er farget lyseblått. Til høyre illustreres et stjernekryss (rødt) som har tre markerte termineringspunkt. De involverte delområdene er tre mørkegråe og seks lyseblåe.*

12.6.2 Tensorproduktflater og irregulære grid

Irregulære grid (rutenett) kan være ganske komplekse. Det vi skal undersøke er hvordan en samling vanlige tensorproduktflater som er koblet sammen på en irregulær måte oppfører seg. Dette innebærer håndtering av T-kryss og Stjernekryss i koblingene. For å koble sammen flere blendingsflater til en stor flate, må vi bruke spesielle lokale flater for blendingen mellom naboflater. Disse lokale flatene må dekke nærmeste nabolag (i henhold til gridet) til alle flater som skal kobles sammen.

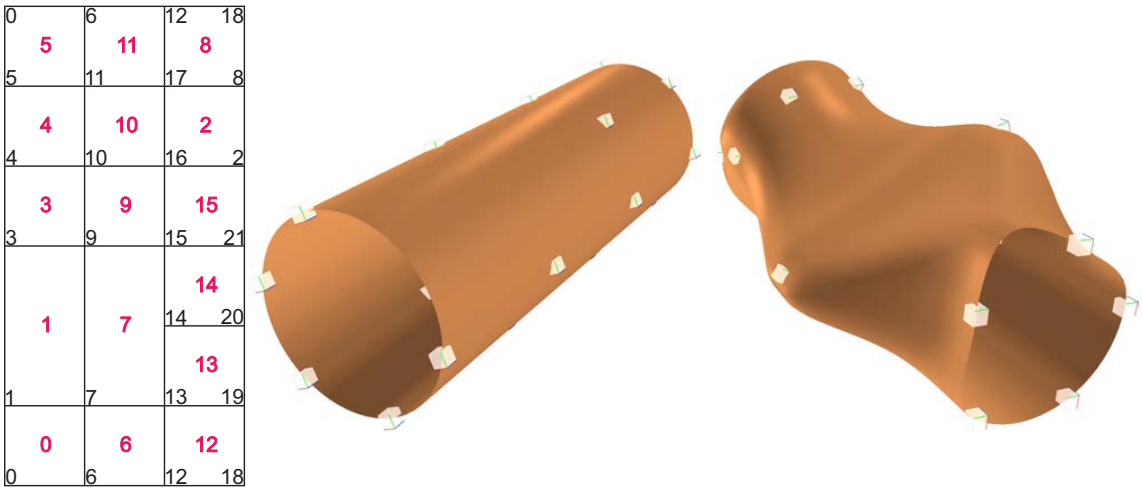
Til venstre i figur 12.15 ser vi et eksempel på et T-kryss, og til høyre et eksempel på et stjernekryss. Vi vil kalle punkt som avslutter irregulære områder for "termineringspunkt".

12.6.3 T-kryss

Et T-kryss er en gridlinje som ender i en ortogonal gridlinje. I figur 12.15 vises et eksempel på et T-kryss. T-kryss oppstår enten når skjøtvektoren i en flate ikke er den samme som skjøtvektoren i naboflaten eller hvis en skjøtverdi forsvinner ved passering av en gridlinje. Hvordan man håndterer T-kryss for å oppnå glatte overflater, er gitt av følgende teorem.

Teorem 12.1. *Hvis vi har et T-kryss eller en samling av koblede T-kryss, må de lokale flatene som er koblet til disse T-kryssene og til termineringspunktene til denne samlingen være deler av en felles flate, for å få en glatt og til og med C^∞-glatt blending.*

Bevis. Fra Lemma 12.1 følger det at i punktene/verteksene er resultatflaten identisk med de lokale flatene som er koblet til de respektive verteksene. Hvis vi har et T-kryss, er T-krysset et punkt på to naboflaterstykker på en felles lokal flate. På den andre siden av T-krysset er det et annen flatestykke fra samme lokale flate. På dette stykket er T-krysset ikke et punkt/verteks. For at en lokal flateå skal interpolere et internt punkt på en kant (med alle dens deriverte) følger det av Lemma 12.1 at de to flatene koblet til de to verteksene som definerer kanten må være deler av samme flate, og at flaten koblet til T-krysset også må være en del av den samme flaten. □



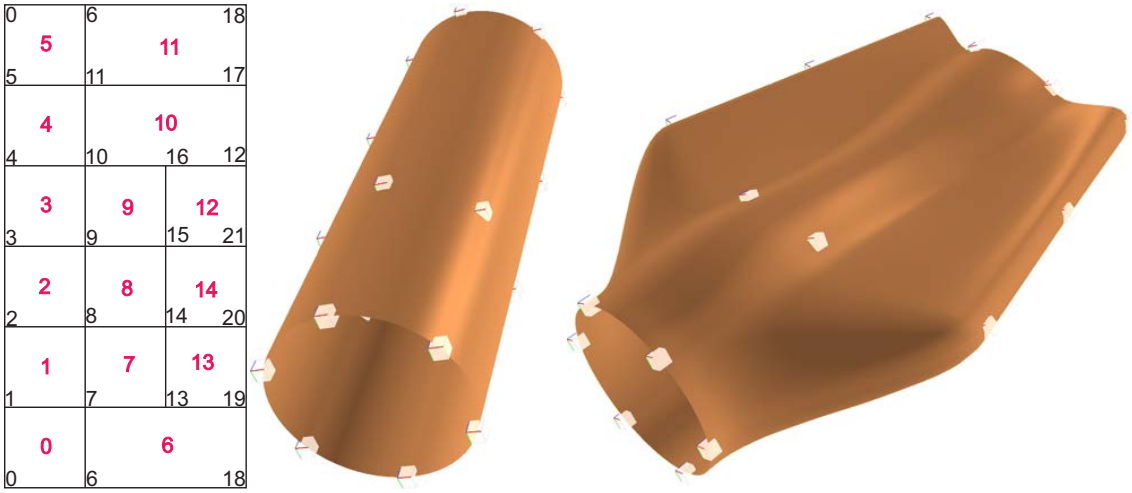
Figur 12.16: *Til venstre i figuren ser vi parameterplanet til en sylinder som er partisjonert av skøtvektorene* $\mathbf{u}$ *og* $\mathbf{v}$ *beskrevet i (12.33). Parameterlappene er merket med røde tall i figuren, og interpolasjonspunktene er merket med svarte tall. To punkt er fjernet. I figuren ser vi at punkt og parameterflater i øvre høyre del har overtatt indeksene (tallene) til de fjernede punktene og lappene. Til høyre i figuren ser vi først en blendingssylinder med spesifisert T-kryss. Punktene rundt området der interpolasjonspunktene er fjernet, flyttes så og roteres deretter. Resultatet ser vi til høyre i figuren.*

Til venstre i figur 12.15 vises et T-kryss markert med rødt og to termineringspunkt markert med svarte sirkler. De tre lokale flatene knyttet til de markerte verteksene og som dekker det ikke regulære området, er markert med lyseblått og er delt inn i syv lapper.

Figur 12.16 viser en implementering. Til venstre i figuren ser vi parameterplanet til en sylinder som er oppdelt av skjøtvektorene

$$\begin{aligned}\mathbf{u} &= \left\{-\frac{1}{2}, -\frac{1}{2}, -\frac{1}{6}, \frac{1}{6}, \frac{1}{2}, \frac{1}{2}\right\}, \quad \text{og} \\ \mathbf{v} &= \left\{-\frac{4}{3}\pi, -\pi, -\frac{2}{3}\pi, -\frac{1}{3}\pi, 0, \frac{1}{3}\pi, \frac{2}{3}\pi, \pi, \frac{4}{3}\pi\right\}.\end{aligned} \tag{12.33}$$

Implementeringen er basert på parameterlapper merket med røde tall i figur 12.16, og interpolasjonspunkt inkludert deres lokale flater som er merket med svarte tall i figuren. De er begge plassert i separate vektorer hvor tallene er indekser i sine respektive vektorer. Interpolasjonspunktene kan fjernes interaktivt ved å velge ett eller flere punkt og deretter fjerne dem. I figur 12.16 er dette gjort med to punkt. Dette er vist i figuren fordi punktene og lappene i øvre høyre del av parameterplanet har overtatt numrene til de fjernede punktene og lappene. Interpolasjonspunkt nummer 14 i figuren er nå et T-kryss, og følgelig må de lokale flatene til punktene 13, 14 og 15 kobles sammen i henhold til Teorem 12.1. Hvis vi bruker delflatekonstruksjonen må da de tre punktene ha en felles matrise. Til høyre i figur 12.16 ser vi først en blendingspline-sylinder med T-krysset i punkt 14. Verteksene rundt lappene der interpolasjonspunktene er fjernet, flyttes og roteres deretter. Resultatet kan sees til høyre i figur 12.16. Resultatet er fortsatt like glatt som den originale sylinderen.



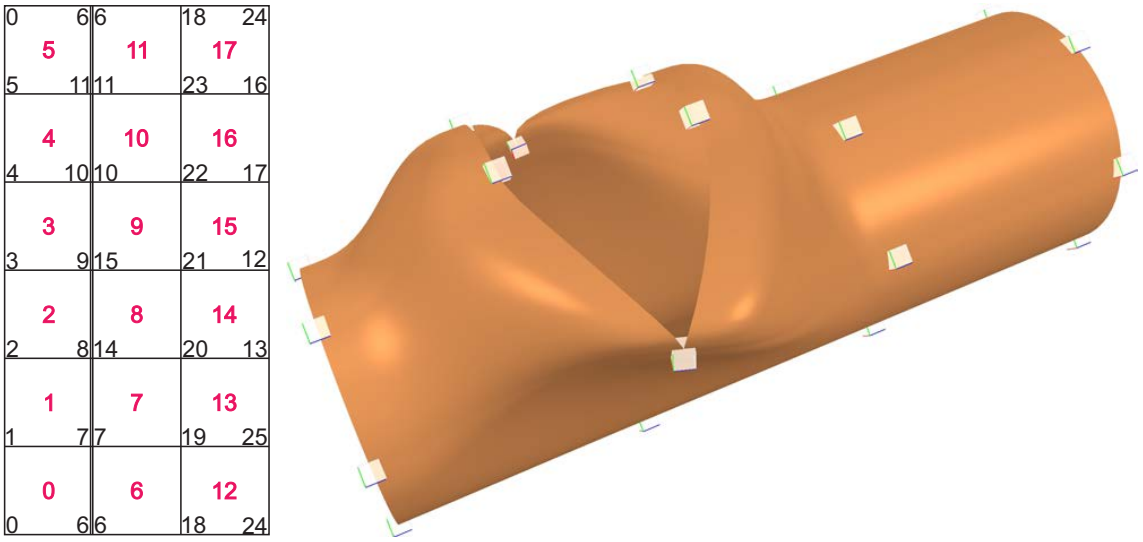
Figur 12.17: *Som i figur 12.16 er det her snakk om å fjerne interpolasjonspunkt. Til venstre i figuren ser vi parameterplanet som er partisjonert av skjøtvektorene* $\mathbf{u}$ *og* $\mathbf{v}$ *beskrevet i (12.33). Parameterlappene er markert med røde tall i figuren, og interpolasjonspunktene med svarte tall. To punkt er fjernet. Plasseringen av tallene er et resultat av algoritmen for å fjerne punktene og lappene. Til høyre i figuren ser vi først en blendingspline-sylinder med de angitte T-kryssene. Punktene rundt området der interpolasjonspunktene er fjernet, flyttes så og roteres deretter. Resultatet ser vi til høyre i figuren.*

I figur 12.17 er de fjernede punktene på hver side av den “sykliske grensen", på en lukket flate, og de er dermed interne. To punkt er fjernet. I figur 12.17 ser vi at det er punktet mellom lapp 10 og 11, og punktet mellom lapp 11 og 6. Figuren til høyre viser resultatet i parameterplanet etter at de to punktene er fjernet. Parameterlappene, merket med røde tall, holder styr på indeksene til interpolasjonspunktene merket med svarte tall. Interpolasjonspunktene holder også styr på parameterlappene som omgir dem. Antall parameterlapper reduseres med tre. Alle punktene rundt de fjernede punktene har blitt flyttet og rotert etter at de to punktene er fjernet. Resultatet ser vi til høyre i figur 12.17.

Hvis vi legger inn en multippel skjøtverdi, dvs

$$\mathbf{u} = \left\{ -\frac{1}{2}, -\frac{1}{2}, -\frac{1}{6}, -\frac{1}{6}, \frac{1}{6}, \frac{1}{2}, \frac{1}{2} \right\} \tag{12.34}$$

får vi en mulig geometrisk diskontinuerlig flate. I figur 12.18 er dette gjort. Samtidig har vi ved hjelp av T-kryss fjernet 4 interpolasjonspunkt/lokale flater, dette er punktene med indeksene 12, 13, 16 og 17. Resultatet kan sees til høyre i Figur 12.18. Der vi ikke har fjernet punktene med de multiple skjøtverdien, har vi nå et hull og der vi har fjernet punktene er overflaten kontinuerlig og glatt. Legg merke til at det fortsatt er en “en til en" avbildning mellom parameterplanet og flaten selv om flaten er diskontinuerlig. I Pedersen et al. [129], vises mer sofistikerte hull sammen med mulige anvendelser, spesielt knyttet til isogeometrisk analyse.



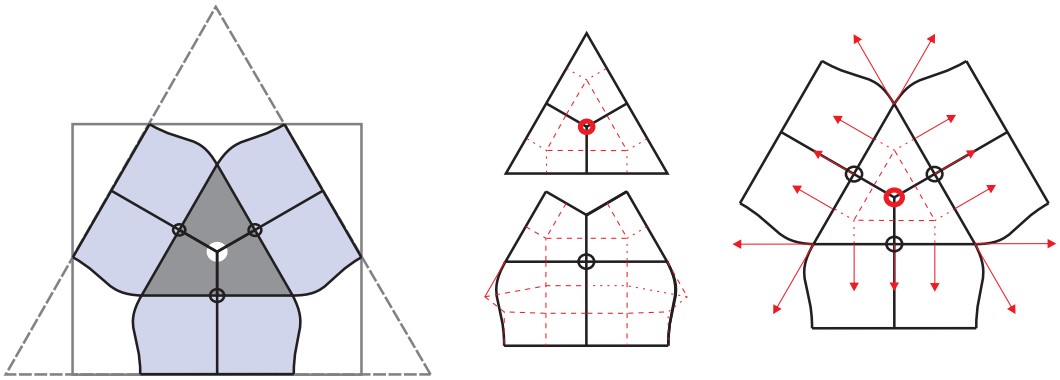
Figur 12.18: *I dette eksemplet har* **u***-vektoren, (12.34), en intern multippel skjøt, Til venstre er parameterplanet illustrert, og dobbelskjøten kan sees som to nesten sammenfallende vertikale linjer. 4 interpolasjonspunkt med tilhørende lokale flater er fjernet. Dette er punktene til høyre for 6, 7, 10 og 11. Flaten i* $\mathbb{R}^3$ *ser vi til høyre. Resultatet illustrerer at vi kan modellere hull uten å trimme flaten, dvs. at det er en "en til en" avbildning mellom parameterplanet og flaten.*

12.6.4 Stjernekryss

Et stjernekryss er et punkt der flere gridlinjer møtes på en "ikke-ortogonal" måte (antall liner $\neq 4$. Ett eksempel på et stjernekryss er gitt til høyre i figur 12.15, der tre gridlinjer møtes i ett punkt. Problemene knyttet til stjernekryss kommer tydelig frem når vi ser på figur 12.15. Linjer i parameterplanet hvor en parameter er konstant får en knekk når de passerer en av kantene som går ut av et stjernekryss. I det følgende lemmaet viser vi hvordan du håndterer stjernekryss i blending.

Teorem 12.2. *For å få en glatt blending når vi har et stjernekryss:*
a) må den lokale flaten koblet til stjernekrysset og de lokale flatene knyttet til termineringspunktene være delflater av en felles flate, men de lokale (del)flatene kan translateres individuelt (ikke roteres, skaleres, etc.),
b) og "glattheten" over kantene som går fra stjernekrysset til termineringspunktene vil være C^S*, mens "glattheten" over kanten ortogonalt på termineringspunktene (se figur 12.19) vil være avhengig av reparametriseringen som er benyttet.*

Bevis. Fordi de lokale flatene enten er deler av samme flate eventuelt bare er flyttet, er alle deriverte av alle ordener like på de to lokale flatene som deler en kant. Det følger av lemma 12.1 at posisjonen og alle retningsderiverte på den resulterende flaten må konvergere mot det samme når vi går mot kanten fra begge sider. Dermed blir kanten mellom stjernekrysset og dets termineringspunkt C^S-glatt. Ved de omkringliggende kantene, for eksempel trekanten vi ser i figur 12.19, vil glattheten kun avhenge av glattheten av reparametriseringen til den sammensatte lokale flaten. Dermed kan den resulterende flaten bare



Figur 12.19: *Til venstre vises parameterplanet til en flate (enten tensorproduktflate eller béziertrekant) for et stjernekryss og hvordan flatelappene er. I midten vises kontrollpolygonet til bézierflatene brukt i parameterplanet for reparametriseringen. Til høyre er det brukt en béziertrekant. Den er delt i 3 av reparametriseringen med bruk av bézierflater. I tillegg er 3 flater basert på hermiteblending av kurver knyttet til hver sin kant.*

være like glatt som reparametriseringen, jfr. eksemplet i figur 12.19. □

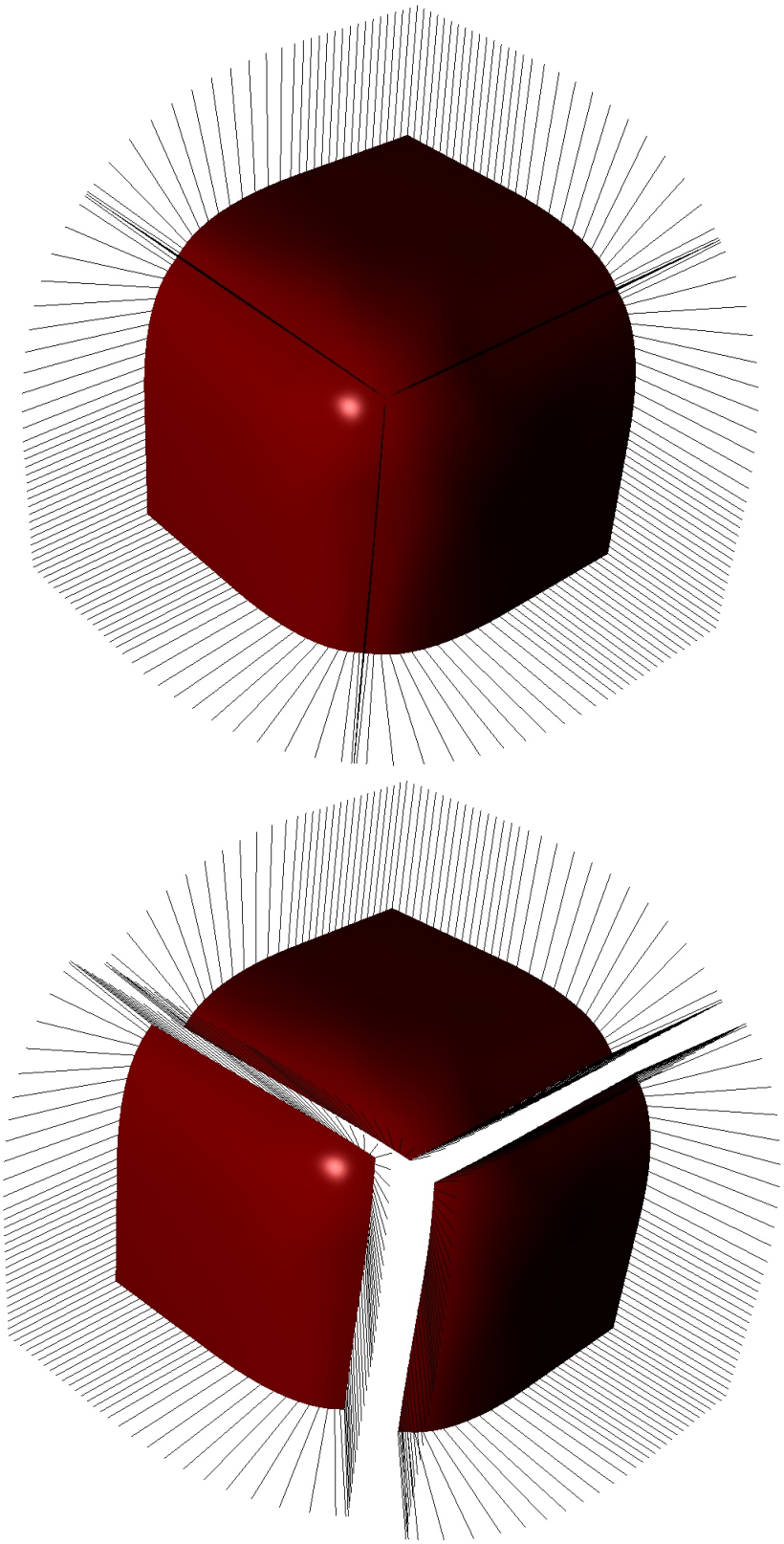
På venstre side i figur 12.19 er det et eksempel på et parameterplan til en flate som dekker det irregulære området til stjernekrysset. Det er ni paameterlapper, markert med grått og lyseblått, som til sammen definerer fire lokale flater. Punktet i midten av figuren viser den lokale flaten festet til selve Stjernekrysset og under ser vi en av de tre lokale flatene som er knyttet til et av termineringspunktene. Merk at den øvre delen av den nedre flaten er den samme som den nedre delen av den senterflaten.

I den øvre midtre trekanten i figur 12.19 er det en reparametriseringen i hver av de tre parameterlappene ved å bruke en bézier-avbildning, $\omega : \mathbb{R}^2 \to \mathbb{R}^2$. Hver av de tre parameterlappene er reparametrisert i begge retninger ved å bruke et 2.-grads bézier-avbildning med et 3×3 kontrollpolygon, på en “symmetrisk" måte. Den nedre flaten i midten har 4 deler. De to øvre delene er lik de to nedre delene i flaten over. I de to nedre delene ser vi i stiplet rødt en bézier-avbildning med 3×4 kontrollpunkt, dvs. 2.-grads horisontalt og 3.-grads vertikalt.

I den nedre parameterflaten i midten i figur 12.19 navngir vi kontrollpunktene på venstre side fra topp til bunn for $c_{i,j},\ i = 1,2,3$ og $j = 1,2,...,6$. Den øvre parameterlappen bruker så $c_{i,j},\ i = 1,2,3$ og $j = 1,2,3$ og den nedre lappen $c_{i,j},\ i = 1,2,3$ og $j = 3,4,5,6$. Det følger da at den lokale flaten er kontinuerlig. Hvis i tillegg $c_{i,4} - c_{i,3} = c_{i,3} - c_{i,2}$ så er den lokale flaten C^1-glatt over kanten.

På høyre side av figur 12.19 brukes en béziertrekant utvidet med hermiteblendede kurver koblet til hver av kantene på trekanten. Kurvene på den andre siden kan velges vilkårlig. Antallet deriverte-funksjoner som brukes i hermiteblendingen bestemmer kontinuiteten.

I figur 12.20 vises en flate med et stjernekryss hvor avbildningen som er forklart på høyre side i figur 12.19 er brukt. Flaten er C^1-glatt fordi bare én derivertefunksjon som er brukt for å lage flatene ved hermiteblendede kurver. Ved å bruke de to avbildningene som er beskrevet til venstre i figur 12.19 er resultatet likt det første eksemplet. Til venstre i



Figur 12.20: *Et eksempel på en glatt flate med et stjernekryss. Flaten er laget ved å blende plane flater og en béziertrekant utvidet ved å bruke hermiteblendede kurver fra de tre kantene.*

figur 12.20 er flaten vist med normaler plottet langs kantene også de tre kantene knyttet til Stjernekrysset. Figurene illustrerer kontinuiteten. Til høyre er de tre delene flyttet fra hverandre.

Kapittel 13

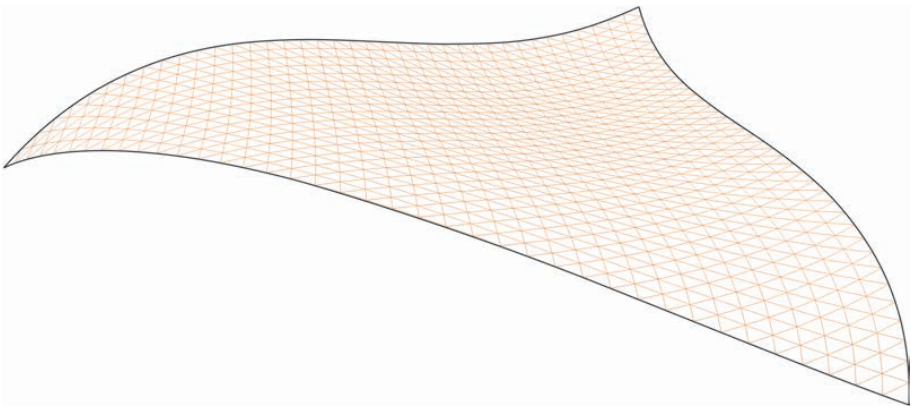
Trekantede flater

En trekant er en simpleks, og dermed en av grunnformene i geometrien. Det er et polygon med 3 kanter og 3 hjørner (vertekser). Plane trekanter har en del velkjente egenskaper som for eksempel at summen av vinklene alltid er 180°. Vi skal imidlertid her konsentrere oss om ikke-plane trekantede flater som figur 13.1 viser oss et eksempel av.

Det er vanlig å modellere et volum i $\mathbb{R}^3$ ved å lage den ytre randa, som for et volum er en overflate. En overflatene er begrenset, kompakt og sammenhengende med en topologiske genus g, dvs. antall hull/håndtak (torus har 1). Tessellering av overflater til et volum gjøres ofte med trekanter, da trekant-representasjon er enklere enn å bruke generelle polygon. Det viser seg at for enhver triangulering av en gitt overflate S er $\varkappa(S)$[1] et fast tall.

I første del av dette kapittelet vil vi konsentrere oss om de grunnleggende egenskapene til enkle trekantede ikke-plane flater. Videre skal vi se på triangulerte flater, dvs. flater som består av et sett med sammenkoblede trekantende flater som er mer eller mindre glatte i overgangen mellom trekantflatene.

[1] Euler-Poincaré-karakteristikken for en kompakt og sammenhengende flate S er $\varkappa(S) = F - E + V$, der F er antall trekanter, E er antall kanter og V er antall vertekser. Det er relatert til genius g til flaten på følgende måte, $\varkappa(S) = 2(1-g)$. Se Gauss-Bonnet i [49].



Figur 13.1: *En glatt og ikke-plan trekantet flate i* $\mathbb{R}^3$.

Den mest kjente ikke plane trekantede flaten er béziertrekanten. Den ble, ifølge Farin [64], først introdusert tidlig på 1960-tallet av de Casteljau i de interne tekniske Citroën rapportene [38] og [39]. Béziertrekanten vil bli gjenomgått i seksjon, 13.1.

Hovedmålet med første del av kapittelet er imidlertid å introdusere trekantflater basert på blending, og som er definert på et domene beskrevet av homogene barysentriske koordinater.

13.1 Béziertrekanter

En trekant er en 2-dimensjonal simpleks - Δ_2 definert i seksjon 2.7. På en simpleks kan vi bruke homogene barysentriske koordinater, seksjon 2.8. For en trekantet flate betyr dette

$$s(u,v,w) \quad \text{hvor} \quad u+v+w=1 \quad \text{og hvor} \quad u,v,w \geq 0.$$

En Δ_2 simpleks beskrevet med homogene barysentriske koordinater definerer domenet til en trekantet bézierflate. Over parameterdomet Δ_2 kan vi konstruere basisfunksjoner av enhver grad d på følgende måte,

$$(u+v+w)^d = 1, \quad \text{hvor} \quad u,v,w \geq 0 \quad \text{og} \quad d > 0.$$

Basisfunksjonene for béziertrekanter er bernsteinpolynomene av polynomgrad d, og disse er for $u+v+w=1$:

$$b_{d,i,j,k}(u,v,w) = \binom{d}{ijk} u^i v^j w^k, \quad \text{hvor} \quad i+j+k=d \quad \text{og} \quad i,j,k \geq 0.$$

For béziertrekanter har vi dermed følgende generelle formel,

$$S(u,v,w) = \sum_{\substack{i+j+k=d,\\ i,j,k\geq 0}} c_{i,j,k}\, b_{d,i,j,k}(u,v,w) \quad \text{for} \quad u+v+w=1,$$

hvor $c_{i,j,k} \in \mathbb{R}^n$ er koeffisientene, og der n normalt er 3.

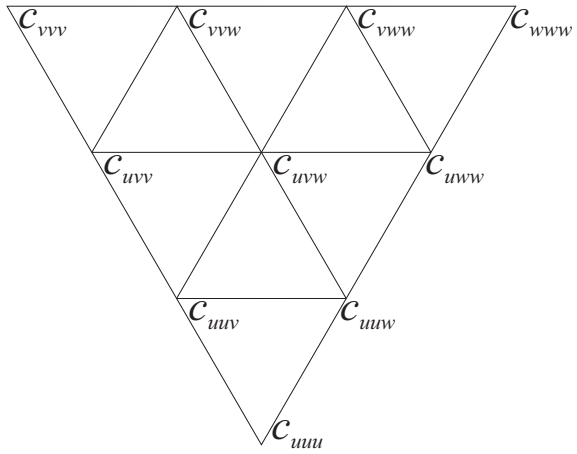
I avsnitt 4.4.2, definisjon 4.5, ble bernstein-faktormatriser $T_d(t)$ beskrevet. I disse matrisene summerer hver rad opp til 1 og er dermed egnet for barysentriske koordinater. Vi trenger kun å bytte t med $u = 1-t$ og $v = t$. Rune Dalmo har i [34] beskrevet disse matrisene for høyere dimensjoner. Han viste at $T_d(u_0,\ldots,u_k)$ er en $\binom{d+k-1}{k} \times \binom{d+k}{k}$ båndmatrise med kun $k+1$ elementer som er forskjellig fra null på hver rad. Matrisen er definert rekursivt som følger:

$$T_d(u_0,\ldots,u_k) = \left(\begin{array}{c|c} T_{d-1}(u_0,\ldots,u_k) & 0 \\ \hline T_{d^{diag}}(u_0) & T_d(u_1,\ldots,u_k) \end{array}\right).$$

Matriseformen til en 3.-grads béziertrekant blir da

$$s(u,v,w) = T_1(u,v,w)\; T_2(u,v,w)\; T_3(u,v,w)\; \mathbf{C}$$

hvor



Figur 13.2: *Notasjon og organisering av kontrollpunktene i en 3.-grads béziertrekant.*

$$T_1(u,v,w) = \begin{pmatrix} u & v & w \end{pmatrix}, \qquad T_2(u,v,w) = \left(\begin{array}{ccc|ccc} u & v & w & 0 & 0 & 0 \\ \hline 0 & u & 0 & v & w & 0 \\ 0 & 0 & u & 0 & v & w \end{array}\right)$$

$$T_3(u,v,w) = \left(\begin{array}{cccccc|cccc} u & v & w & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & u & 0 & v & w & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & u & 0 & v & w & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & u & 0 & 0 & v & w & 0 & 0 \\ 0 & 0 & 0 & 0 & u & 0 & 0 & v & w & 0 \\ 0 & 0 & 0 & 0 & 0 & u & 0 & 0 & v & w \end{array}\right)$$

og

$$\mathbf{C} = (c_{uuu}, c_{uuv}, c_{uuw}, c_{uvv}, c_{uvw}, c_{uww}, c_{vvv}, c_{vvw}, c_{vww}, c_{www})^T$$

Figur 13.2 viser notasjonen og plassering/organisering av kontrollpunktene $\mathbf{C}$. Merk at plasseringen av punktene i vektoren går fra bunn til topp og fra venstre til høyre i figuren. Hvis vi nå ruller ut formelen til en 3.-grads béziertrekant får vi

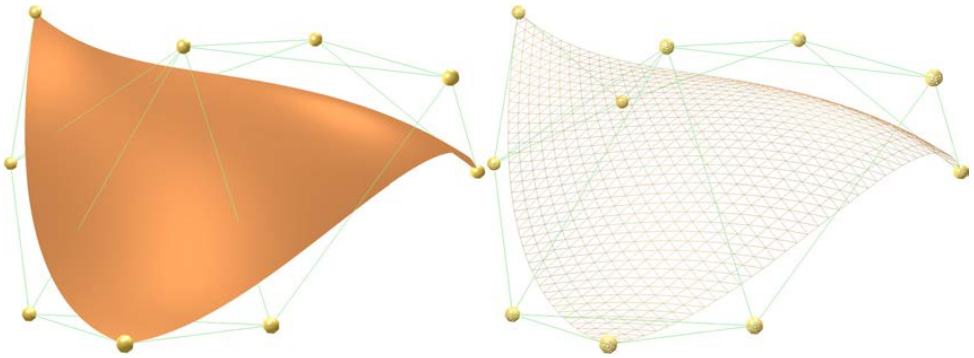
$$\begin{aligned} s(u,v,w) = {} & u^3 c_{uuu} + 3u^2 v c_{uuv} + 3u^2 w c_{uuw} + 3uv^2 c_{uvv} + uvw c_{uvw} \\ & + 3uw^2 c_{uww} + v^3 c_{vvv} + 3v^2 w c_{vvw} + 3vw^2 c_{vww} + w^3 c_{www} \end{aligned}$$

I homogene barysentriske koordinater har vi både partiellderiverte, S_u, S_v, S_w og retningsderiverte, $S_{\mathbf{d}}$. Fra (4.36) så vi at vi kan finne de deriverte ved å derivere den siste matrisen, og at radene i denne matrisen summerte opp til 0, dvs. de deriverte er vektorer. Med barysentriske koordinater ser vi at radene i derivertematrisene summere opp til 1, og dermed blir de partiellderiverte punkt. De deriverte vi trenger er derimot retningsderiverte i retning $\mathbf{d} = (r,s,t)$ hvor $r+s+t = 0$. De retningsderiverte er følgelig vektorer og kan beregnes på følgende måte

$$S_{\mathbf{d}}(u,v,w) = r\, S_u(u,v,w) + s\, S_v(u,v,w) + t\, S_w(u,v,w), \tag{13.1}$$

hvor $u+v+w = 1$ og $r+s+t = 0$. Parameterverdiene i hjørnene er $(1,0,0)$, $(0,1,0)$ og $(0,0,1)$, og vi kan bruke $d_1 = (-1,1,0)$ og $d_2 = (-1,0,1)$ for retningsderiverte, dvs.

$$s_{d_1}(u,v,w) = s_v(u,v,w) - s_u(u,v,w) \quad \text{og} \quad s_{d_2}(u,v,w) = s_w(u,v,w) - s_u(u,v,w),$$



Figur 13.3: *En 3.-grads béziertrekant inkludert dens kontrollpunkt og kontrollpolygon vises. Flaten er plottet både skyggelagt og som rutenett, det siste så vi kan se alle kontrollpunktene og hele kontrollpolygonet.*

og normalen er dermed

$$n = s_{d_1}(u,v,w) \wedge s_{d_2}(u,v,w).$$

Figur 13.3 viser en skyggelagt béziertrekant der normaler er brukt, og en som rutenett. Mer om den generelle teorien om Bernstein-béziertrekanter, inkludert algoritmer for beregning av posisjon og deriverte finnes for eksempel i [63] eller [64].

13.2 B-funksjon i homogene barysentriske koordinater

I seksjon 7.1 defineres en standard én-variabel B-funksjon. I **D5** i definisjon 7.1 defineres punktsymmetri, det vil si $B(t)+B(1-t)=1$. I 1-dimensjonale homogene barysentriske koordinater blir dette til $B(u)+B(v)=1$ og som når vi setter det sammen blir til $B(u,v) = (x_1,x_2)$ hvor $u+v=1$ og $x_1+x_2=1$.

Vi kan nå utvide B-funksjoner til å bruke homogene barysentriske koordinater og vi får:

Definisjon 13.1. *En B-funksjon* $B(u_0,u_1,\ldots,u_n)$ *hvor* $\sum_{i=0}^{n} u_i = 1$ *er:*

D1 *en homeomorfi (permutasjon)* $B : \Delta_n \to \Delta_n$, *hvor* Δ_n *er en n-simpleks*

– **D2** *dermed er* $B(\ldots,u_{i-1},0,\ldots) = \{\ldots,x_{i-1},0,\ldots\}$, i=0,1,...,n

– **D3** *og* $B(\ldots,u_{i-1},1,\ldots) = \{\ldots,x_{i-1},1,\ldots\}$, i=0,1,...,n

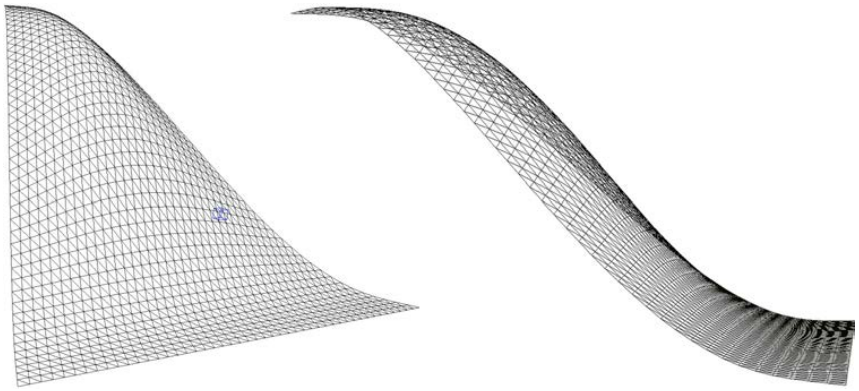
– **D4** *og som er monoton, dvs.* $\sum_{j=0}^{n} u_j \frac{\partial}{\partial u_i} B(\bar{u}) - \frac{\partial}{\partial u_i} B(\bar{u}) = \bar{x}$, $x_i \geq 0$, i=0,1,...,n

D5 *En B-funksjon er symmetrisk hvis* $B(\bar{u}) = \bar{x}$ *for alle felles permutasjoner i* $\bar{u}$ *og* $\bar{x}$

Merk at for å være monoton, ***D4****, følger det at for hvert koordinat må den retningsderivert fra et hvilket som helst punkt, i retning toppunktet til gjeldende koordinat være* ≥ 0 *(toppunktet er hvor gjeldende koordinatverdi er 1, se figur 13.4).*

En B-funksjon overfører $n+1$ koordinater. Med trekanter betyr det 3 koordinater slik at

$$B(u,v,w) = (x_0,x_1,x_2)\,,$$



Figur 13.4: *B-funksjonen $B_1(u_1,u_2,u_3)$ for triangler sett fra to forskjellige posisjoner.*

og vi kan følgelig angi hvert element med en indeks, dvs.

$$B_i(u,v,w) = x_i, \quad i = 0,1,2.$$

Som det viste seg når det gjaldt én-variabel B-funksjoner, definert i seksjon 7.1, har vi også mange måter å konstruere B-funksjoner i homogene barysentriske koordinater.

Vi skal se på to eksempler, først skal vi bruke en standard én-variabel B-funksjon i konstruksjonen.

Definisjon 13.2. *Gitt et punkt $(u_0,u_1,...,u_n) \in \Delta_n$ (i homogene barysentriske koordinater og som oppfyller konveksitetsegenskapen). En B-funksjon i Δ_n kan defineres som følger,*

$$B(u_0,u_1,\ldots,u_n) = \frac{1}{\sum_{i=0}^{n} B(u_i)} \left(B(u_0),B(u_1),\ldots,B(u_n)\right) \tag{13.2}$$

hvor $B(u)$ er en standard én-variabel B-funksjon som er beskrevet i definisjon 7.1.

For $n = 2$, dvs. en trekant, vil hver komponent i B-funksjonen bli

$$B_i(u_0,u_1,u_2) = \frac{B(u_i)}{B(u_0)+B(u_1)+B(u_2)} \quad \text{for} \quad i = 0,1,2.$$

Vi kan lett se at definisjon 13.2 oppfyller alle 5 punktene i definisjon 13.1.

I figur 13.4 er én koordinat til en 2-variabel B-funksjon plottet. I dette plottet er den eksporasjonale B-funksjonen av type 1 (7.31) brukt som standard én-variabel B-funksjonen i definisjon 13.2.

Som for béziertrekanten må vi bruke retningsderiverte , dvs. $\mathbf{d} \in \Upsilon_n$. De partiellderiverte må også beregnes, for $n > 0$ får vi følgende 6 uttrykk. Først, 1.-ordens partiellderiverte, for $i = 0,1,...,n$ er

$$D_{u_i}B_i(u_0,\ldots,u_n) = B'(u_i)\frac{\sum_{j=0}^{n} B(u_j) - B(u_i)}{\left(\sum_{j=0}^{n} B(u_j)\right)^2}, \tag{13.3}$$

og, for $j = 0, 2, ..., n$, med unntak for $j \neq i$,

$$D_{u_j} B_i(u_0, \ldots, u_n) = B'(u_j) \frac{-B(u_i)}{\left(\sum_{j=0}^{n} B(u_j)\right)^2}. \tag{13.4}$$

For 2.-ordens partiellderiverte får vi

$$D^2_{u_i} B_i(u_0, \ldots, u_n) = \left(B''(u_i) - \frac{2\left(B'(u_i)\right)^2}{\sum_{j=0}^{n} B(u_j)} \right) \frac{\sum_{j=0}^{n} B(u_j) - B(u_i)}{\left(\sum_{j=0}^{n} B(u_j)\right)^2}, \tag{13.5}$$

og for $j = 0, 1, ..., n$, med unntak for $j \neq i$ og for både

$$D^2_{u_j} B_i(u_0, \ldots, u_n) = \left(B''(u_j) - \frac{2\left(B'(u_j)\right)^2}{\sum_{j=0}^{n} B(u_j)} \right) \frac{-B(u_i)}{\left(\sum_{j=0}^{n} B(u_j)\right)^2}, \tag{13.6}$$

og

$$D_{u_i} D_{u_j} B_i(u_0, \ldots, u_n) = B'(u_i)\, B'(u_j) \frac{2B(u_i) - \sum_{j=0}^{n} B(u_j)}{\left(\sum_{j=0}^{n} B(u_j)\right)^3}, \tag{13.7}$$

og for $j = 0, 1, ..., n$, med unntak for $j \neq i$, og for $h = 0, 1, ..., n$, med unntak for $h \neq i, j$, får vi

$$D_{u_h} D_{u_j} B_i(u_0, \ldots, u_n) = B'(u_h)\, B'(u_j) \frac{2B(u_i)}{\left(\sum_{j=0}^{n} B(u_j)\right)^3}. \tag{13.8}$$

Merk at alle disse 6 partiellderiverte har en faktor med den valgte én-variabel B-funksjonen med samme deriverte-orden. Dermed vil hermiteordenen til den homogene barysentriske B-funksjonen være den samme som hermiteordenen til den valgte én-variabel B-funksjonen.

Nå, gitt en vektor $\mathbf{d} \in \Upsilon_n$, dvs.,

$$\mathbf{d} = \mathbf{u_1} - \mathbf{u_2} = (d_0, d_1, ..., d_n), \quad \text{hvor} \quad \mathbf{u_1} \text{ og } \mathbf{u_2} \in \Delta_n.$$

Husk at partiellderiverte i homogene barysentriske koordinater er punkt. For å beregne en flatenormal må vi ha vektorer, dvs. retningsderiverte for B-funksjoner i homogene barysentriske koordinater. Det vil si

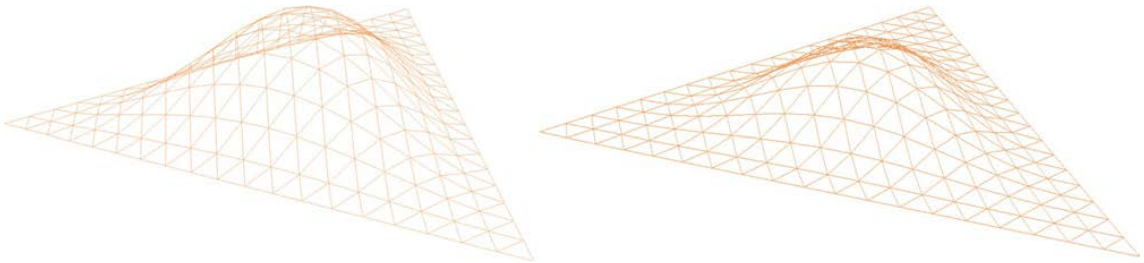
$$D_{\mathbf{d}} B_i(u_0, u_1, ..., u_n) = \sum_{j=0}^{n} d_j\, D_{u_j} B_i((u_0, u_1, ..., u_n)). \tag{13.9}$$

Det er praktisk å definere hovedretninger for deriverte. For trekanter kan vi bruke,

$$\mathbf{d}_1 = (-1, 1, 0), \quad \text{og} \quad \mathbf{d}_2 = (-1, 0, 1). \tag{13.10}$$

I disse to retningene er deriverte opp til 2.-orden for $B_i(u, v, w)$, $i = 0, 1, 2$:

$$\begin{aligned}
D_{\mathbf{d}_1} B_i(u, v, w) &= D_v B_i(u, v, w) - D_u B_i(u, v, w), \\
D_{\mathbf{d}_2} B_i(u, v, w) &= D_w B_i(u, v, w) - D_u B_i(u, v, w), \\
D^2_{\mathbf{d}_1} B_i(u, v, w) &= D^2_v B_i(u, v, w) - D_u D_v B_i(u, v, w) + D^2_u B_i(u, v, w), \\
D^2_{\mathbf{d}_2} B_i(u, v, w) &= D^2_w B_i(u, v, w) - D_u D_w B_i(u, v, w) + D^2_u B_i(u, v, w), \\
D_{\mathbf{d}_1} D_{\mathbf{d}_2} B_i(u, v, w) &= D_v D_w B_i(u, v, w) - D_u D_v B_i(u, v, w) - D_u D_w B_i(u, v, w) + D^2_u B_i(u, v, w).
\end{aligned}$$



Figur 13.5: *Plott av tetthetsfunksjoner tilsvarende Beta-funksjoner. Til venstre en orden 1 funksjon, $\Phi(u,v,w) = u^2v^2w^2$, og til høyre er en orden 2 funksjon, $\Phi(u,v,w) = u^3v^3w^3$.*

Det neste eksempelet er basert på massefordeling, da med utgangspunkt i observasjoner av enkelte én-variabel B-funksjoner. Beta-funksjoner (7.18) og type 1 ERB-funksjonen (7.31) er begge konstruert med denne teknikken, er det er derfor naturlig å utvide dette til B-funksjoner i homogene barysentriske koordinater også.

Definisjon 13.3. *Gitt et punkt $p = (u_0, u_1, ..., u_n) \in \Delta_n$ (i homogene barysentriske koordinater som oppfyller konveksitetsegenskapen). En B-funksjon i Δ_n kan defineres som følger,*

$$B(u_0, u_1, \ldots, u_n) = \frac{1}{mass(\Delta_n)} \left(mass(p, u_0), mass(p, u_1), \ldots, mass(p, u_n)\right) \qquad (13.11)$$

hvor masse(Δ_n) er massen til hele simpleksen, masse(p, u_i), $i = 0, 1, \ldots, n$ er massen til simpleksen definert av punktet p og sub-simpleksen Δ_{n-1} hvor $u_i = 0$.

For $n = 2$, dvs. trekanter har vi disse to eksemplene på tetthetsfunksjoner,

$$\Phi(u,v,w) = e^{-\frac{a}{uvw}}, \quad a > 0, \quad \text{og} \quad \Phi(u,v,w) = (uvw)^k, \quad k > 0.$$

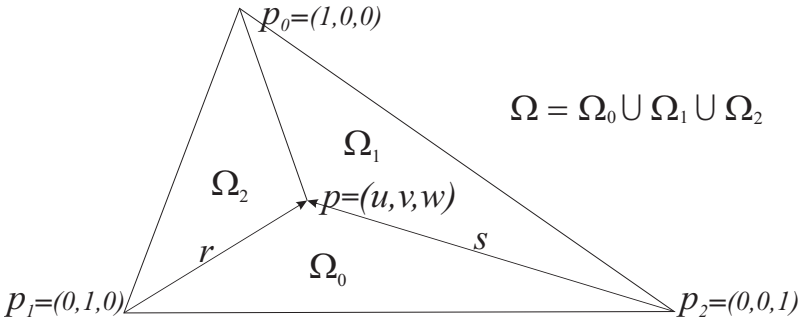
Det siste eksemplet, Beta-funksjonen, skal vi gå videre med og utvikle nye B-funksjoner. I figur 13.5 er det et plott av en 1.-ordens og en 2.-ordens tetthetsfunksjon av typen Beta-funksjon. Ordenen betyr, som for én-variabel B-funksjoner, antall deriverte som er null langs kantene. Ifølge definisjon 13.3 uttrykk (13.11) får vi

$$B(u,v,w) = \frac{1}{\int_\Omega \Phi(u,v,w) d\Omega} \left(\int_{\Omega_0} \Phi(u,v,w) d\Omega_0, \int_{\Omega_1} \Phi(u,v,w) d\Omega_1, \int_{\Omega_2} \Phi(u,v,w) d\Omega_2 \right)$$

hvor domenet Ω og underdomenene Ω_0, Ω_1 og Ω_2 er illustrert i figur 13.6. Punktet $p = (u,v,w)$ er "evaluerings"-punktet og koordinatene til de to vektorene i figuren er $r = p - p_1 = (u, v-1, w)$ og $s = p - p_2 = (u, v, w-1)$. Integralene blir da

$$\int_\Omega \Phi(u,v,w) d\Omega = \int_0^1 \int_0^{1-\mu} \phi(\mu, \nu, 1-\mu-\nu)\, d\nu d\mu$$

$$\int_{\Omega_0} \Phi(u,v,w) d\Omega_0 = \int_0^u \int_{\frac{v}{u}\mu}^{1-\frac{1-v}{u}\mu} \phi(\mu, \nu, 1-\mu-\nu)\, d\nu d\mu$$



Figur 13.6: *Vi ser et trekantet domene Ω og dets tre hjørnene p_0, p_1 og p_2. Et punkt p deler domenet i tre underdomener Ω_0, Ω_1 og Ω_2. De to vektorene r og s vises også.*

De fleste grensene i integralene er selvforklarende. Det som trenger forklaring er de nedre og øvre grensene til det siste integralet. De følger fordi vi bruker punktene p, p_1 og p_2 og vektorene r og s skalert med $\frac{\mu}{u}$ (se figur 13.6) . Dermed får vi

$$p_1 + \frac{\mu}{u}(p - p_1) = ((0,1,0) + \frac{\mu}{u}((u,v,w) - (0,1,0)) = \left(\mu,\ 1 - \frac{1-v}{u}\mu,\ \frac{w}{u}\mu\right)$$

hvor den andre komponenten er den øvre grensen, og

$$p_2 + \frac{\mu}{u}(p - p_2) = ((0,0,1) + \frac{\mu}{u}((u,v,w) - (0,0,1)) = \left(\mu,\ \frac{v}{u}\mu,\ 1 - \frac{1-w}{u}\mu\right)$$

hvor andre komponent er den nedre grensen. $\int_{\Omega_1} \Phi(u,v,w)d\Omega_1$ og $\int_{\Omega_2} \Phi(u,v,w)d\Omega_2$ kan deretter beregnes ved syklisk skifte av de barysentriske variablene u, v, w. Dette gir

1.-ordens Beta-funksjon for trekantede domener

$$B(u,v,w) = \big((6vw - 2u + 3)u^2,\ (6uw - 2v + 3)v^2,\ (6uv - 2w + 3)w^2,\big)$$

og 2.-ordens Beta-funksjoner for trekantede domener

$$\begin{aligned} B(u,v,w) = \big((&30vw(3vw - u + 1) + 6u^2 - 15u + 10)u^3, \\ &(30uw(3uw - v + 1) + 6v^2 - 15v + 10)v^3, \\ &(30uv(3uv - w + 1) + 6w^2 - 15w + 10)w^3\big) \end{aligned}$$

De partiellderiverte for både 1.- og 2.-ordens Beta-funksjoner er enkle å beregne. Og vi kan bruke samme teknikk som i det første eksemplet, definisjon 13.2, for å beregne retningsderiverte av forskjellig orden. Vi ser også at hermiteordenen S, definisjon 7.2, for Beta-funksjoner vil være potensen til tetthetsfunksjonene minus 1, og at ordenen er lik det vi ser for den én-variabel B-funksjonen av samme type.

Figur 13.7 viser komponentene til to Beta-funksjoner i homogene barysentriske koordinater, de tre 1.-ordenskomponentene til venstre og de tre 2.-ordenskomponentene til høyre. Husk at siden de er homogene barysentriske koordinater, summerer de alltid til 1. Ordenen S fungerer på samme måte som for én-variabel B-funksjoner, men hvor "start og slutt" er der alle retningsderiverte er null, dvs. hvor $u_i = 1$ og $u_i = 0$. Førstnevnte er et punkt, mens sistnevnte er en sub-simpleks Δ_{n-1}, dvs. en kant.

Figur 13.7: *Vi ser de tre komponentene til en Beta-funksjon i homogene barysentriske koordinater, tre 1.-ordensfunksjoner til venstre og tre 2.-ordensfunksjoner til høyre.*

13.3 Blendingstrekanter

Den generelle formelen for en blendingstrekant er

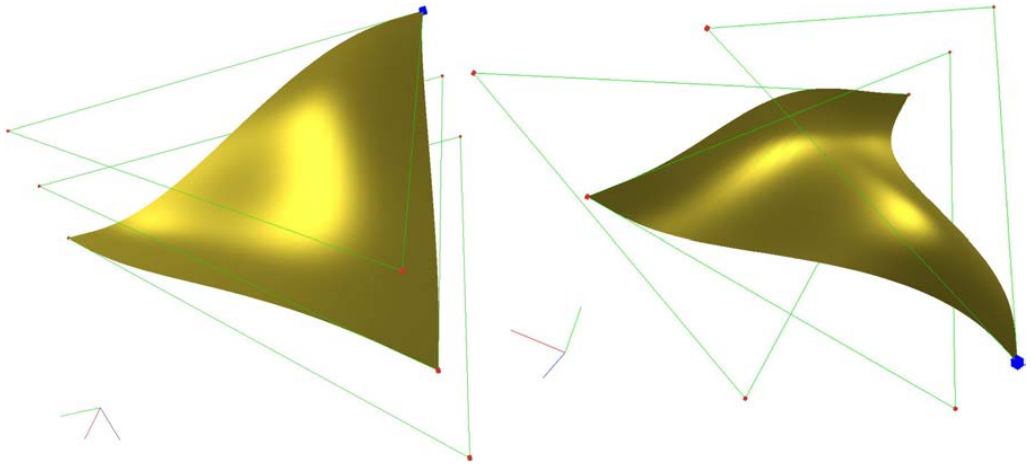
$$S(u,v,w) = \sum_{i=0}^{2} s_i(u,v,w)\, B_i(u,v,w), \quad \text{hvor} \quad u+v+w=1, \quad u,v,w \geq 0,$$

der $s_i(u,v,w),\ i = 0,1,2$ er lokale trekanter og B(u,v,w) er en B-funksjon i homogene barysentriske koordinater. I figurene 13.8 ser vi to plott av blendingstrekanter og deres respektive 3 lokale trekanter. Alle de lokale trekantene er plane i begge figurene, og vi kan se dem markert som grønne linjer. På figuren til venstre ser vi også at de er parallelle. Vi ser også at den globale blendingstriangelen interpolerer hver av de lokale trekantene i et av hjørnene, og vi vet fra egenskapene til B-funksjoner at interpolasjonen ikke bare involverer posisjonen, men også de deriverte opp til en gitt orden avhengig av ordenen til B-funksjonen som er brukt. Blendingsflaten til høyre i figur 13.8 er i utgangspunktet den samme som den til venstre. De lokale trekantene har imidlertid blitt flyttet og rotert. De partialderiverte beregnes ved produktregelen og de retningsderiverte ved å bruke (13.9) i retningene vi finner i (13.10).

Det er av flere grunner å foretrekke at de lokale flatene er orientert på en slik måte at den fulle støtten til den første parameteren (dvs. $u = 1$) er i interpolasjonspunktet til den globale blendingstrekanten. Dette forenkler konstruksjonen og organiseringen av de lokale trekantene, men det introduserer behovet for en ny type overgang av parameterne mellom den globale trekanten og de lokale trekantene for hver av hjørnepunktene. Denne overgangen er kun en syklisk rotasjon av parameterne. Denne rotasjonsavbildning har følgende uttrykk,

$$S(u,v,w) = s_0(u,v,w)\, B_0(u,v,w) + s_1(v,w,u)\, B_1(u,v,w) + s_2(w,u,v)\, B_2(u,v,w), \quad (13.12)$$

der $u+v+w=1, \quad u,v,w \geq 0, \quad$ og $\quad s_i,\ i = 0,1,2$ er lokale trekanter, orientert på en slik måte at den full støtte for den første parameteren (dvs. den lokale $u = 1$) er i interpolasjonspunktet med den globale blendingstriangelen. Derfor har vi rotasjonen av parameterne til de lokale trekantene i (13.12). Uttrykket er imidlertid lett å implementere i programeringssammenheng.



Figur 13.8: *To blendingstrekanter og deres tre lokale trekanter markert med grønne linjer. De lokale trekantene er alle 1.-grads béziertrekanter. Til venstre er de parallelle med hverandre, mens de til høyre er flyttet og rotert.*

13.4 Lokale béziertrekanter og hermiteinterpolasjon

I figur 13.9 vises tre blendingstrekanter, hver med tre lokale 2.-grads béziertrekanter. Figur 13.9 gir et lite innblikk i formingsmulighetene der både kontrollpunktene til béziertrekantene og også posisjonen og orienteringen til béziertrekantene kan endres.

For å lage en blendingstrekant ved å "kopiere" en del av en annen flate, må de lokale trekantede flatene lages ved å hermiteinterpolere hvert av hjørnene med en béziertrekant. I seksjon 12.3.1 er lokale bézier-tensorproduktflater laget med hermiteinterpolasjon i ett punkt. Teknikken for béziertrekanter er ganske lik, men må tilpasses trekanter og homogene barysentriske koordinater. Metoden er:

√ Gitt en flate $g : \Omega_g \subset \mathbb{R}^2 \to \mathbb{R}^n$, hvor typisk $n = 3$.

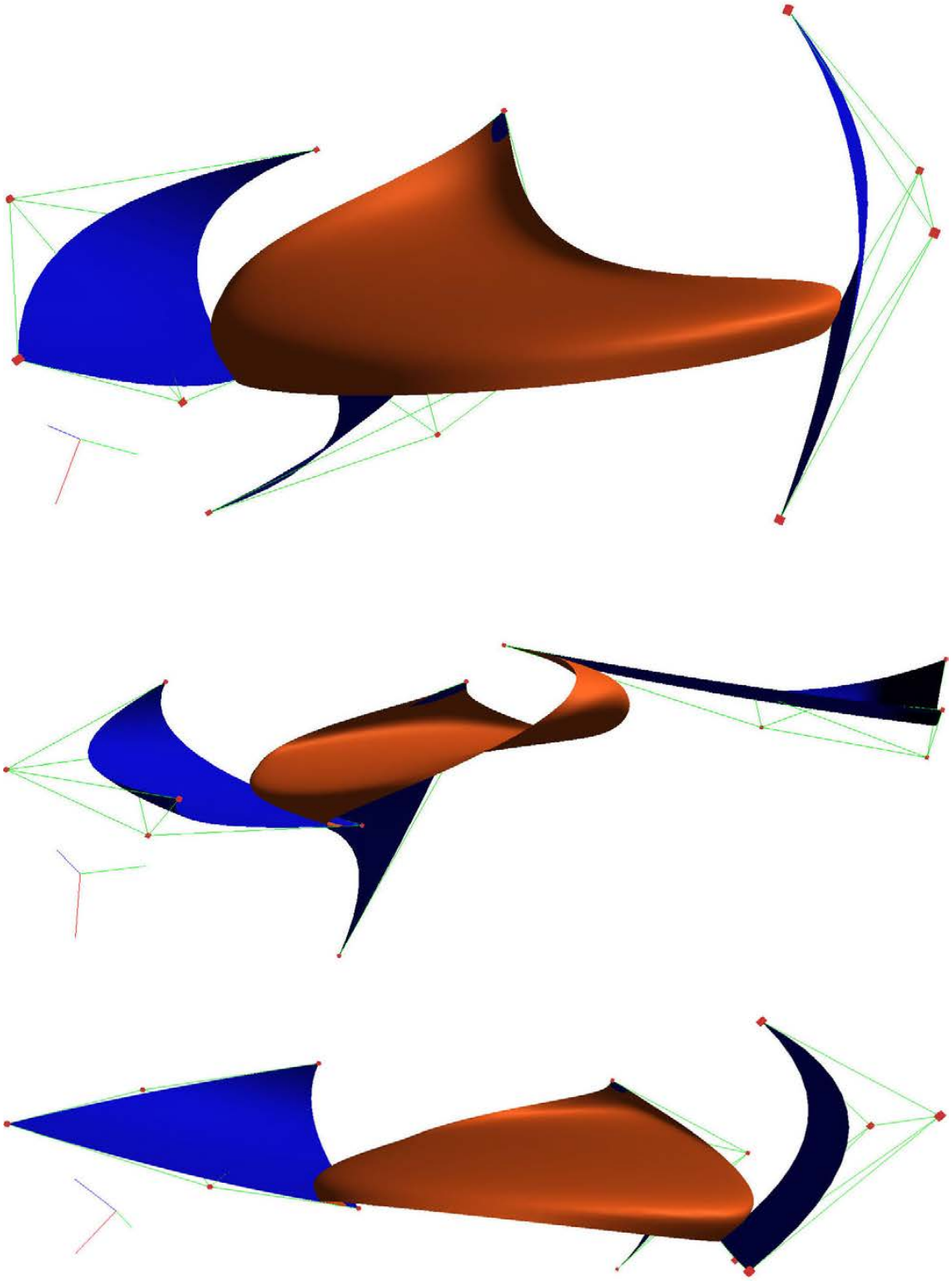
√ Gitt tre punkt i parameterplanet Ω_g; p_0, p_1 og p_2, hver assosiert med en lokal trekant. Polynomgraden til hver av de lokale béziertrekantene må også være gitt.

√ Lag så lokale béziertrekanter $s_i(u,v,w)$ av grad d_i ved å bruke posisjon og deriverte i parameterverdiene p_i, $i = 0,1,2$. Legg merke til det sykliske skiftet i uttrykkene nedenfor, jfr. (13.12). Hermitinterpolasjonen er basert på at følgende krav må være oppfylt for å kunne lage de tre lokale béziertrekantene, dvs.

$$
\begin{aligned}
D^i_{(-1,1,0)} D^j_{(-1,0,1)} s_0(1,0,0) &= D^i_{(-1,1,0)} D^j_{(-1,0,1)} S(1,0,0) = D^i_{p_1-p_0} D^j_{p_2-p_0} g(p_1), \\
D^i_{(-1,1,0)} D^j_{(-1,0,1)} s_1(1,0,0) &= D^i_{(0,-1,1)} D^j_{(1,-1,0)} S(0,1,0) = D^i_{p_2-p_1} D^j_{p_0-p_1} g(p_2), \\
D^i_{(-1,1,0)} D^j_{(-1,0,1)} s_2(1,0,0) &= D^i_{(1,0,-1)} D^j_{(0,1,-1)} S(0,0,1) = D^i_{p_0-p_2} D^j_{p_1-p_2} g(p_3),
\end{aligned}
$$

for alle kombinasjoner av i, j hvor $0 \le (i+j) \le d_k$, og $i, j \ge 0$, for $k = 0,1,2$.

Blendingstrekanter interpolerer sine lokale trekanter i hjørnene, med posisjonen og alle deriverte opp til hermiteordenen S. Dette ser vi tydelig i figur 13.9. Dette er grunnen til at



Figur 13.9: *Vi ser tre forskjellige blendingtrekanter. De lokale trekantene er 2.-grads béziertrekanter, og vi kan se dem farget i blått. Kontrollpolygonene til disse lokale béziertrekantene kan sees som grønne linjer, og kontrollpunktene deres som røde kuber.*

hermiteinterpolering av de lokale flatene kun må gjøres i ett punkt for hver lokale trekantede flate, der posisjonen og de deriverte fra den opprinnelige flaten g unikt bestemmer den lokale overflaten.

For å gjennomføre hermiteinterpolasjon av béziertrekanter, tar vi utgangspunkt i kravene på forrige side. Vi tar uttrykket til venstre som er posisjon og deriverte til béziertrekanter, og uttrykket til høyre som er de samme deriverte i samme punkt på den opprinnelige flaten, dvs.

$$D^i_{(-1,1,0)} D^j_{(-1,0,1)} s(1,0,0) = D^i_{\mathbf{d}_1} D^j_{\mathbf{d}_2} g(\mathbf{p}), \quad \text{for} \quad 0 \leq (i+j) \leq d, \quad i,j \geq 0, \quad (13.13)$$

der d er polynomgraden til béziertrekanten, $\mathbf{p}$ er punktet i Ω_g, og $\mathbf{d}_1, \mathbf{d}_2$ er vektorer $\in \mathbb{R}^2$ i punktet $\mathbf{p}$. Merk at antall uttrykk i (13.13) er

$$n_b = \sum_{i=1}^{d+1} i,$$

som følger av at summen av de mulige variantene, for $i+j=d$ er $d+1$ når $i,j \geq 0$. For $d=1$ får vi $n_b = 3$, for $d=2$ får vi $n_b = 6$, og for $d=3$ får vi $n_b = 10$. Dette samsvarer med forholdet mellom graden og antall kontrollpunkt til en béziertrekant. Formelen i (13.13) bestemmer derfor unikt béziertrekanten fra posisjonen og deriverte i den opprinnelige overflaten.

For å beregne uttrykket for hermiteinterpolasjonen bruker vi uttrykket for retningsderiverte for béziertrekanten. 1.-ordens retningsderiverte er gitt i (13.1). Alle disse beregningene er gjort i [102], side 157–161.

For 3 punkt $\mathbf{p}, \mathbf{p}_1, \mathbf{p}_2 \in \Omega_g$, og to vektorer $\mathbf{q}_1 = \mathbf{p}_1 - \mathbf{p}$ og $\mathbf{q}_2 = \mathbf{p}_2 - \mathbf{p}$, Er resultatene av hermiteinterpolasjonen av en parametrisk flate en 1.-grads béziertrekanten der

$$\begin{aligned} c_0 &= g(\mathbf{p}), \\ c_1 &= c_0 + dg_{\mathbf{p}}(\mathbf{q}_1), \\ c_2 &= c_0 + dg_{\mathbf{p}}(\mathbf{q}_2). \end{aligned}$$

Hermiteinterpolasjonen for å få en 2.-grads béziertrekant er litt mer komplisert, kontrollpunktene vil der bli

$$\begin{aligned} \mathbf{c}_1 &= g(\mathbf{p}), \\ \mathbf{c}_2 &= \mathbf{c}_1 + \frac{1}{2} dg_{\mathbf{p}}(\mathbf{q}_1), \\ \mathbf{c}_3 &= \mathbf{c}_1 + \frac{1}{2} dg_{\mathbf{p}}(\mathbf{q}_2), \\ \mathbf{c}_4 &= -\mathbf{c}_1 + 2\mathbf{c}_2 + \frac{1}{2} d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_1), \\ \mathbf{c}_5 &= -\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 + \frac{1}{2} d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_2), \\ \mathbf{c}_6 &= -\mathbf{c}_1 + 2\mathbf{c}_3 + \frac{1}{2} d(dg(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2). \end{aligned}$$

Hermiteinterpolasjonen for å få en 3.-grads béziertrekant gir kontrollpunktene

$$
\begin{aligned}
\mathbf{c}_1 &= g(\mathbf{p}),\\
\mathbf{c}_2 &= \mathbf{c}_1 + \frac{1}{3} dg_{\mathbf{p}}(\mathbf{q}_1),\\
\mathbf{c}_3 &= \mathbf{c}_1 + \frac{1}{3} dg_{\mathbf{p}}(\mathbf{q}_2),\\
\mathbf{c}_4 &= -\mathbf{c}_1 + 2\mathbf{c}_2 + \frac{1}{6} d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_1),\\
\mathbf{c}_5 &= -\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 + \frac{1}{6} d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_2),\\
\mathbf{c}_6 &= -\mathbf{c}_1 + 2\mathbf{c}_3 + \frac{1}{6} d(dg(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2),\\
\mathbf{c}_7 &= \mathbf{c}_1 - 3\mathbf{c}_2 + 3\mathbf{c}_4 + \frac{1}{6} d(d(dg(\mathbf{q}_1))(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_1),\\
\mathbf{c}_8 &= \mathbf{c}_1 - 2\mathbf{c}_2 - \mathbf{c}_3 + \mathbf{c}_4 + 2\mathbf{c}_5 + \frac{1}{6} d(d(dg(\mathbf{q}_1))(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_2),\\
\mathbf{c}_9 &= \mathbf{c}_1 - \mathbf{c}_2 - 2\mathbf{c}_3 + 2\mathbf{c}_5 + \mathbf{c}_6 + \frac{1}{6} d(d(dg(\mathbf{q}_1))(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2),\\
\mathbf{c}_{10} &= \mathbf{c}_1 - 3\mathbf{c}_3 + 3\mathbf{c}_6 + \frac{1}{6} d(d(dg(\mathbf{q}_2))(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2).
\end{aligned}
$$

Som vi ser ovenfor, for å finne kontrollpunktene til béziertrekantene, må vi finne posisjon og retningsderiverte i den originale flaten $g(u,v)$, både 1.-, 2.-, 3.-ordens, og kryssderiverte i og mellom de to retningene. Derfor trenger vi en posisjon $\mathbf{p} = (u_0, v_0)$ og to retningsvektorer, $\mathbf{q}_1 = (u_1, v_1)$ og $\mathbf{q}_2 = (u_2, v_2)$ i parameterplanet til den originale flaten g. 1.-ordens deriverte i de to retningene blir da

$$
\begin{aligned}
dg_{\mathbf{p}}(\mathbf{q}_1) &= u_1 g_u(\mathbf{p}) + v_1 g_v(\mathbf{p}),\\
dg_{\mathbf{p}}(\mathbf{q}_2) &= u_2 g_u(\mathbf{p}) + v_2 g_v(\mathbf{p}).
\end{aligned}
$$

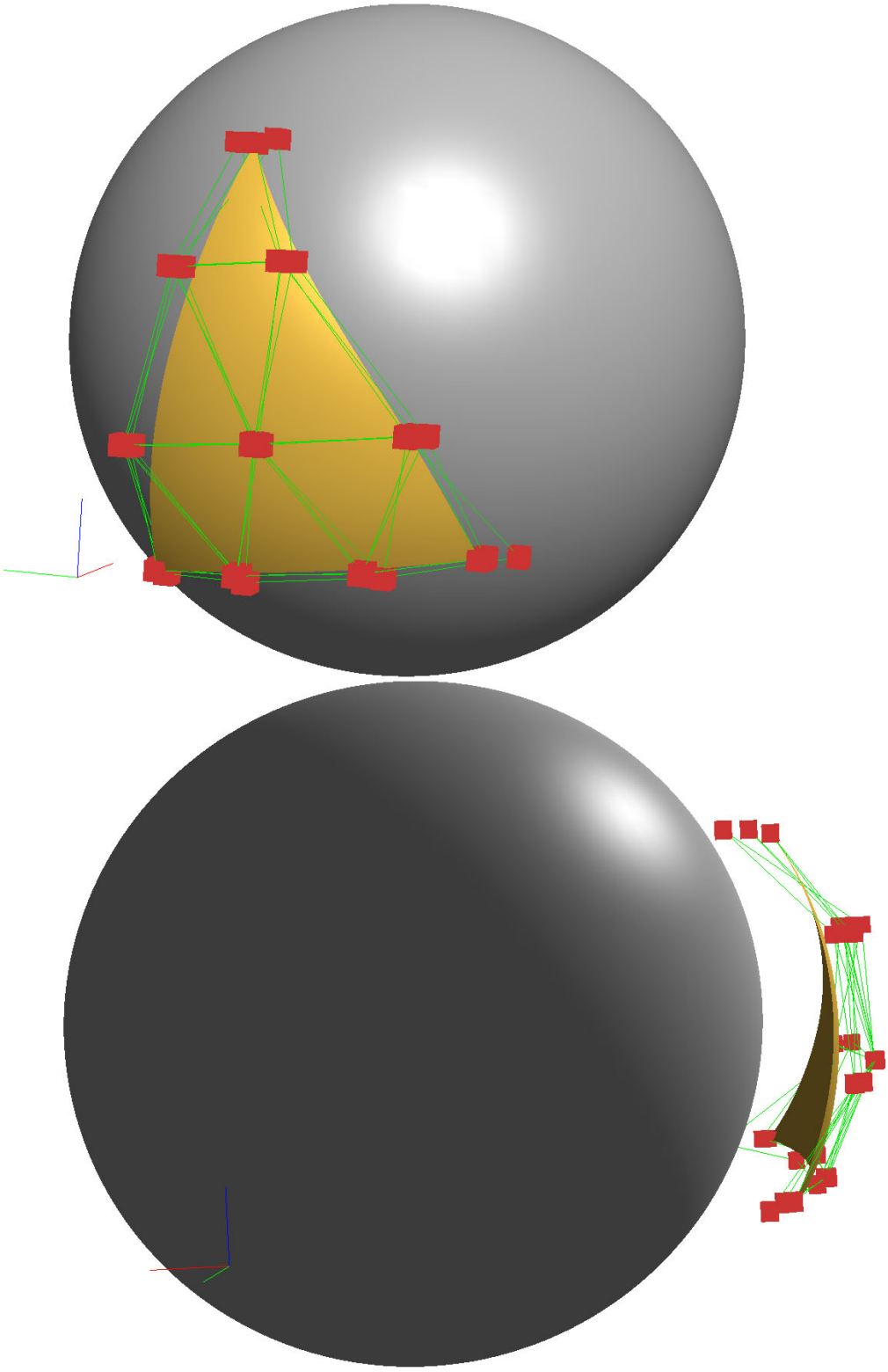
De 2.-ordens deriverte er

$$
\begin{aligned}
d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_1) &= u_1 d(g_u)_{\mathbf{p}}(\mathbf{q}_1) + v_1 d(g_v)_{\mathbf{p}}(\mathbf{q}_1) = u_1^2 g_{uu}(\mathbf{p}) + 2u_1 v_1 g_{uv}(\mathbf{p}) + v_1^2 g_{vv}(\mathbf{p}),\\
d(dg(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2) &= u_2 d(g_u)_{\mathbf{p}}(\mathbf{q}_2) + v_2 d(g_v)_{\mathbf{p}}(\mathbf{q}_2) = u_2^2 g_{uu}(\mathbf{p}) + 2u_2 v_2 g_{uv}(\mathbf{p}) + v_2^2 g_{vv}(\mathbf{p}),\\
d(dg(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_2) &= u_1 d(g_u)_{\mathbf{p}}(\mathbf{q}_2) + v_1 d(g_v)_{\mathbf{p}}(\mathbf{q}_2)\\
&= u_1 u_2 g_{uu}(\mathbf{p}) + (u_1 v_2 + u_2 v_1) g_{uv}(\mathbf{p}) + v_1 v_2 g_{vv}(\mathbf{p}),
\end{aligned}
$$

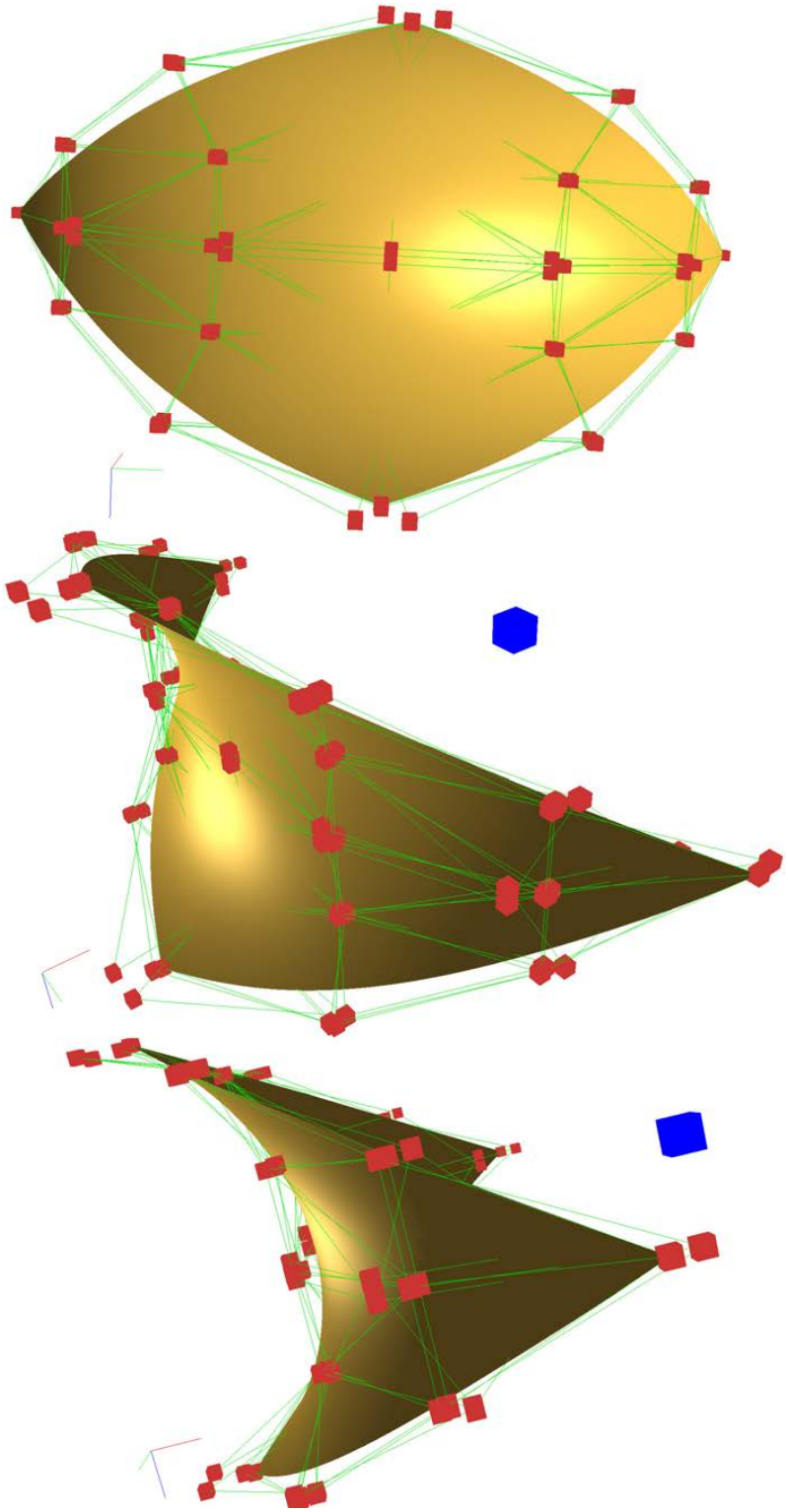
og de 3.-ordens deriverte er

$$
\begin{aligned}
d(d(dg(\mathbf{q}_1))(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_1) &= u_1^3 g_{uuu}(\mathbf{p}) + (3u_1^2 v_1) g_{uuv}(\mathbf{p}) + (3u_1 v_1^2) g_{uvv}(\mathbf{p}) + v_1^3 g_{vvv}(\mathbf{p}),\\
d(d(dg(\mathbf{q}_2))(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2) &= u_2^3 g_{uuu}(\mathbf{p}) + (3u_2^2 v_2) g_{uuv}(\mathbf{p}) + (3u_2 v_2^2) g_{uvv}(\mathbf{p}) + u_2^3 g_{vvv}(\mathbf{p}),\\
d(d(dg(\mathbf{q}_1))(\mathbf{q}_2))_{\mathbf{p}}(\mathbf{q}_2) &= u_1 u_2^2 g_{uuu}(\mathbf{p}) + a_1 g_{uuv}(\mathbf{p}) + b_1 g_{uvv}(\mathbf{p}) + v_1 v_2^2 g_{vvv}(\mathbf{p}),\\
d(d(dg(\mathbf{q}_1))(\mathbf{q}_1))_{\mathbf{p}}(\mathbf{q}_2) &= u_1^2 u_2 g_{uuu}(\mathbf{p}) + a_2 g_{uuv}(\mathbf{p}) + b_2 g_{uvv}(\mathbf{p}) + v_1^2 v_2 g_{vvv}(\mathbf{p}),
\end{aligned}
$$

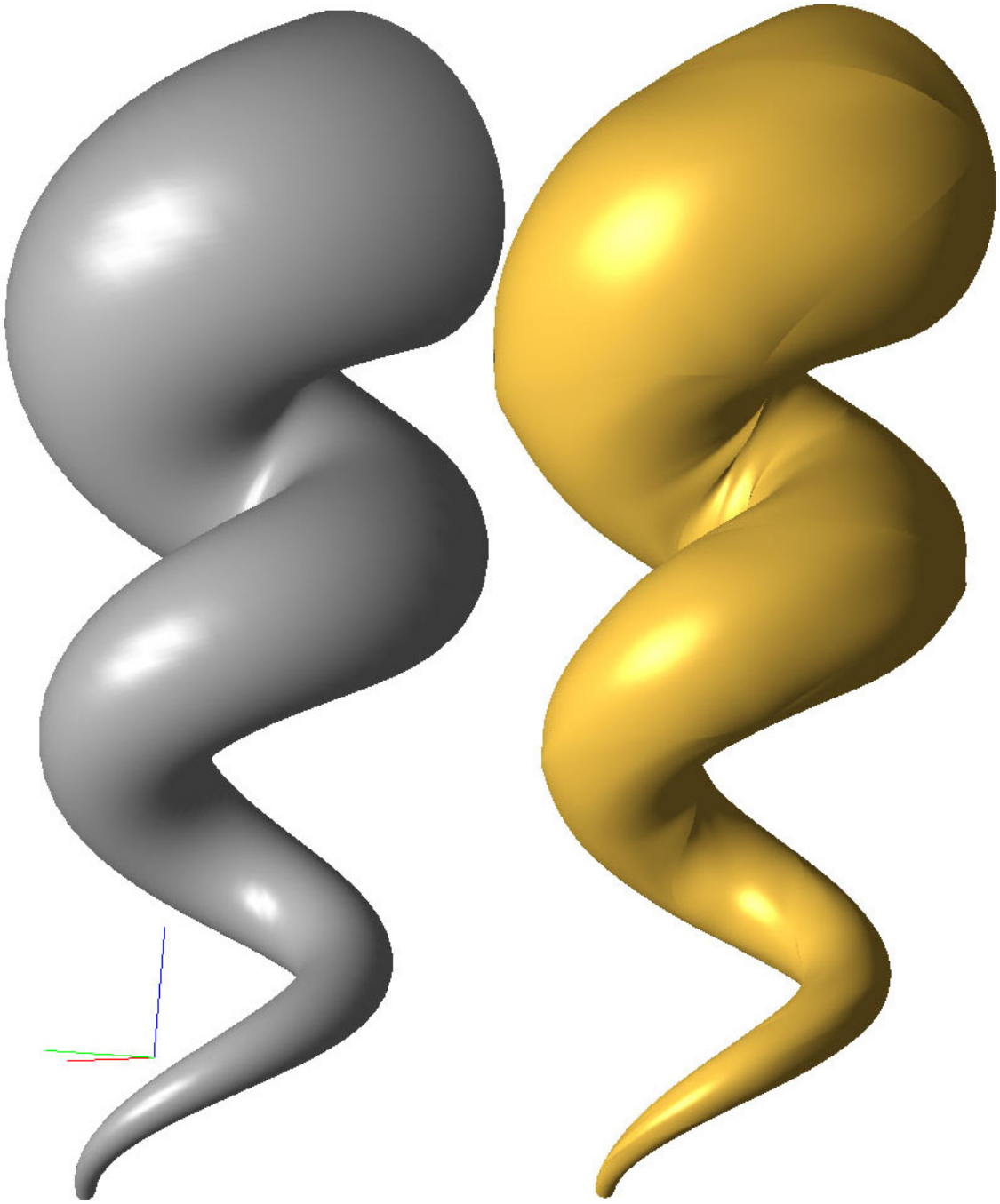
$a_1 = u_2^2 v_1 + 2u_1 u_2 v_2$, $b_1 = u_1 v_2^2 + 2u_2 v_1 v_2$, $a_2 = u_1^2 v_2 + 2u_1 u_2 v_1$, $b_2 = u_2 v_1^2 + 2u_1 v_1 v_2$.



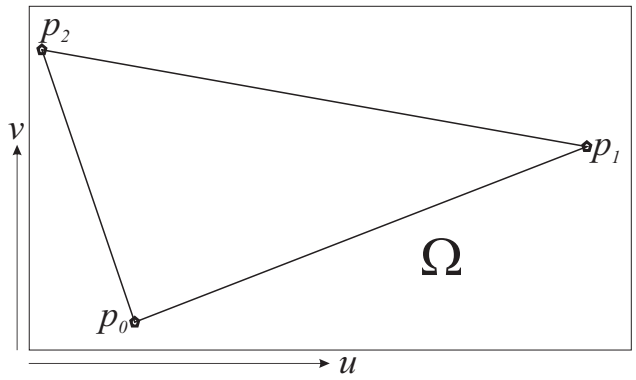
Figur 13.10: *To plott av en sfære og en blendingstrekant som interpolerer sfæren i de tre hjørnene. Blendingstrekanten er flyttet litt bort fra kulen. De lokale trekantene er 3.-grads béziertrekanter, og vi kan se kontrollpolygonene til alle de lokale béziertrekantene som grønne linjer, og kontrollpunktene som røde kuber.*



Figur 13.11: *Plott av en flate sett fra 3 forskjellige posisjoner. Flaten er satt sammen av fire blendingstrekanter som hermiteinterpolerer en del av en torus, formel (12.24), i 5 punkt. Totalt er det* 4×3 *lokale 3.-grads béziertrekanter. Vi kan se kontrollpolygonene deres som grønne linjer, og kontrollpunktene som røde kuber. Den blå kuben er midten representerer den originale torusen.*



Figur 13.12: *Til venstre er det et plott av en "sjøskjell"-flate, formel (12.25), og på høyre side er det et plot av et sett med 80 blendingstrekanter som hermiteinterpolerer en "sjøskjell"-flate i 44 punkt. De 80 blendingstrianglene er alle uavhengige av hverandre, men er "koblet kontinuerlig "; dette betyr at det ikke er hull i den sammensatte flaten etter interpolasjonene.*



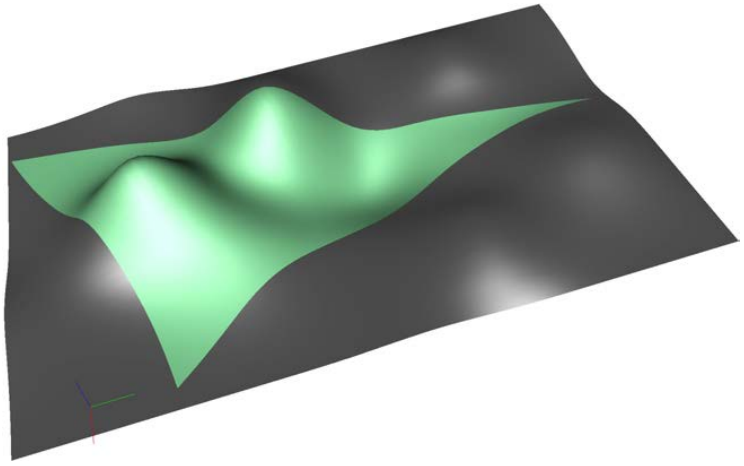
Figur 13.13: *Parameterplanet Ω til en flate $\mathbb{S} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^n,\ n > 0$. De tre punktene $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2 \in \Omega$ beskriver en trekant i parameterplanet.*

I figur 13.10 er en blendingstrekant laget ved å interpolere en kule i tre punkt. I hvert av hjørnepunktene er det så laget en 2.-grads béziertrekant ved hermiteinterpolasjon. I figuren er blendingstrekanten flyttet litt fra kulen slik at vi kan se den bedre. Vi kan også se kontrollpolygonene til hver av de lokale béziertrekantene som grønne linjer, og kontrollpunktene som røde kuber. Utgangspunktet for de tre interpolasjonspunktene er punkt i sfærens parameterplan, og utgangspunktet for de retningsderiverte er vektoren mellom disse punktene i parameterplanet. Parametriseringen av sfæren påvirker dermed trekantens form. Det kan sees at to av kantene danner en svak S. Midtpunktene til de tre kontrollpolygonene er nesten sammenfallende, og de tre kontrollpolygonene er svært like.

I det forrige eksemplet og de to neste eksemplene har vi brukte en ERB-funksjon med ∞ hermiteordren. Det neste eksemplet tar utgangspunkt i en torus, uttrykk (12.24). I figur 13.11 kan vi se en flate, sett fra tre forskjellige posisjoner. Flaten er egentlig fire forskjellige blendingstriangler som er laget ved hermiteinterpolasjon av deler av en torus i tilsammen fem forskjellige punkt, de fire hjørnene og et i midten. Sammenstillingen av de fire blendingstrekantene er klart kontinuerlig, men selv om de sammen ser ut som G^∞ (geometrisk uendelig glatte), er de ikke det. Dette kan vi tydelig se i det neste eksempelet, som er en hermiteinterpolasjon av en “sjøskjell"-flate, formel (12.25). Til venstre i figur 13.12 ser vi et plott av en ordinær “sjøskjell"-flate, (12.25), og på høyre side er det et plott av 80 blendingstrekanter som interpolerer “sjøskjell"-flaten i 44 punkt. Man kan tydelig se at komposisjonen er kontinuerlig, men den ser ikke ut til å være G^∞. Det er faktisk G^∞ kun i alle 44 interpolasjonspunkt, men over de 124 kantene ser det ut til å være bare kontinuerlig, G^0, selv om resultatet er relativt bra. En observasjon er at den ser ut til å være glatt i et punkt ca. midt på hver kant. Alt dette følger av egenskapene til B-funksjonen og at hvis et objektet er dekket av flere parametriseringer, må overgangen fra den ene parametriseringen til den andre også være kontinuerlig og glatt.

13.5 Deltrekanter fra enhver parametrisk flate

Det er mulig å trekke ut en trekantet flate $S(u,v,w)$ fra en vanlig flate som er parametrisert i kartesiske koordinater dvs. $\mathbb{S} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^n$, hvor n typisk er 3. $S(u,v,w)$ er dermed



Figur 13.14: *Den grå flaten er en B-spline-tensorproduktflate. Den trekantede grønne flaten er definert av tre punkt i parameterplanet til B-splineflaten, og dens domene er det minste konvekse settet som inkludert disse tre punktene (det vil si en trekant i det parametriske domenet).*

definert av de tre punktene $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2 \in \Omega$, i parameterplanet til den parametriske flaten $\mathbb{S}$, se figur 13.13.

For å tydeliggjøre notasjonen har vi en "vanlig" parametrisk flate $\mathbb{S}(\mathbf{p}) \in \mathbb{R}^n,\ \mathbf{p} \in \Omega \subset \mathbb{R}^2$ og differensialen, $d\mathbb{S}_{\mathbf{u}} = [\mathbb{S}_u\ \ \mathbb{S}_v]_{(}\,\mathbf{p}) \in \mathbb{R}^{n\times 2}$. En trekantflate S er da

$$S(u,v,w) = \mathbb{S}(\mathbf{u}), \quad \text{hvor} \quad \mathbf{u} = u\mathbf{p}_0 + v\mathbf{p}_1 + w\mathbf{p}_2,$$

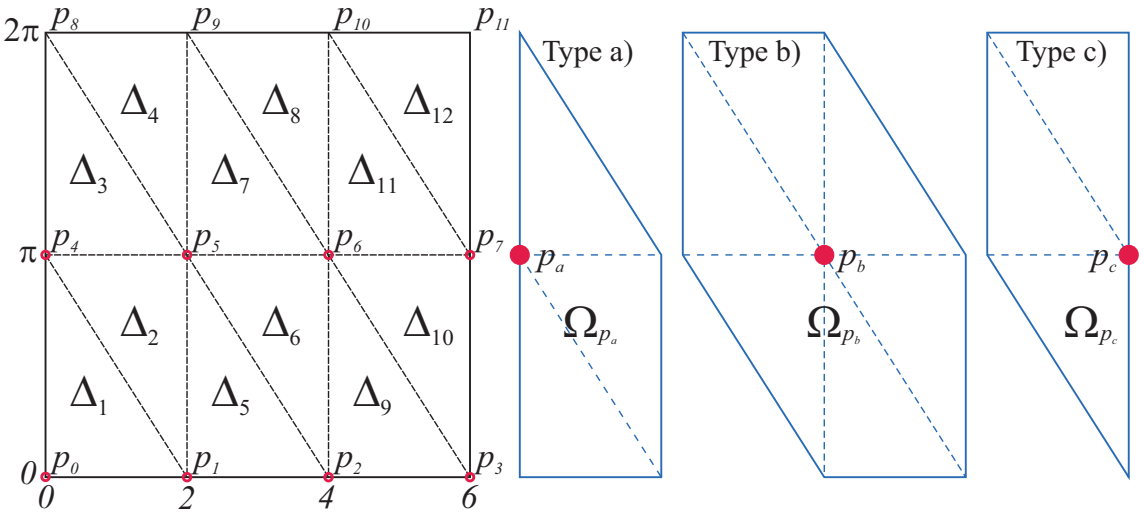
med tre hjørnepunkt $\mathbf{p}_i = \begin{pmatrix} u_i \\ v_i \end{pmatrix},\ i = 0,1,2$. 1.-ordens partiellderiverte er,

$$\begin{aligned}
S_u(u,v,w) &= d\mathbb{S}_{\mathbf{u}}(\mathbf{p}_0) = u_0\ \mathbb{S}_u(\mathbf{u}) + v_0\ \mathbb{S}_v(\mathbf{u}),\\
S_v(u,v,w) &= d\mathbb{S}_{\mathbf{u}}(\mathbf{p}_1) = u_1\ \mathbb{S}_u(\mathbf{u}) + v_1\ \mathbb{S}_v(\mathbf{u}),\\
S_w(u,v,w) &= d\mathbb{S}_{\mathbf{u}}(\mathbf{p}_2) = u_2\ \mathbb{S}_u(\mathbf{u}) + v_2\ \mathbb{S}_v(\mathbf{u}),
\end{aligned}$$

og 2.-ordens partiellderiverte er,

$$\begin{aligned}
S_{uu}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_0))_{\mathbf{u}}(\mathbf{p}_0) = u_0^2\ \mathbb{S}_{uu}(\mathbf{u}) + 2u_0v_0\ \mathbb{S}_{uv}(\mathbf{u}) + v_0^2\ \mathbb{S}_{vv}(\mathbf{u}),\\
S_{vv}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_1))_{\mathbf{u}}(\mathbf{p}_1) = u_1^2\ \mathbb{S}_{uu}(\mathbf{u}) + 2u_1v_1\ \mathbb{S}_{uv}(\mathbf{u}) + v_1^2\ \mathbb{S}_{vv}(\mathbf{u}),\\
S_{ww}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_2))_{\mathbf{u}}(\mathbf{p}_2) = u_2^2\ \mathbb{S}_{uu}(\mathbf{u}) + 2u_2v_2\ \mathbb{S}_{uu}(\mathbf{u}) + v_2^2\ \mathbb{S}_{vv}(\mathbf{u}),\\
S_{uv}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_0))_{\mathbf{u}}(\mathbf{p}_1) = u_0u_1\ \mathbb{S}_{uu}(\mathbf{u}) + (u_0v_1 + v_0u_1)\mathbb{S}_{uv}(\mathbf{u}) + v_0v_1\ \mathbb{S}_{vv}(\mathbf{u}),\\
S_{uw}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_0))_{\mathbf{u}}(\mathbf{p}_2) = u_0u_2\ \mathbb{S}_{uu}(\mathbf{u}) + (u_0v_2 + v_0u_2)\mathbb{S}_{uv}(\mathbf{u}) + v_0v_2\ \mathbb{S}_{vv}(\mathbf{u}),\\
S_{vw}(u,v,w) &= d(d\mathbb{S}(\mathbf{p}_1))_{\mathbf{u}}(\mathbf{p}_2) = u_1u_2\ \mathbb{S}_{uu}(\mathbf{u}) + (u_1v_2 + v_1u_2)\mathbb{S}_{uv}(\mathbf{u}) + v_1v_2\ \mathbb{S}_{vv}(\mathbf{u}).
\end{aligned}$$

For å beregne retningsderiverte og dermed normaler følger vi samme prosedyre som for béziertrekanter, se avsnitt 13.1. I figur 13.14 vises en trekantet flate ekstrahert fra en B-spline-tensorproduktflate. Domenet til denne trekantede flaten er vist i figur 13.13.



Figur 13.15: *Til venstre er hele parameterdomenet til en sylinder, $[0,6] \times [0,2\pi)$. 8 punkt er markert med rødt, og 4 punkt er ikke markert fordi de er de samme som de 4 punktene nederst. Parameterdomenet er triangulert. Til høyre ser vi de 3 typene deldomener som er koblet til de røde punktene.*

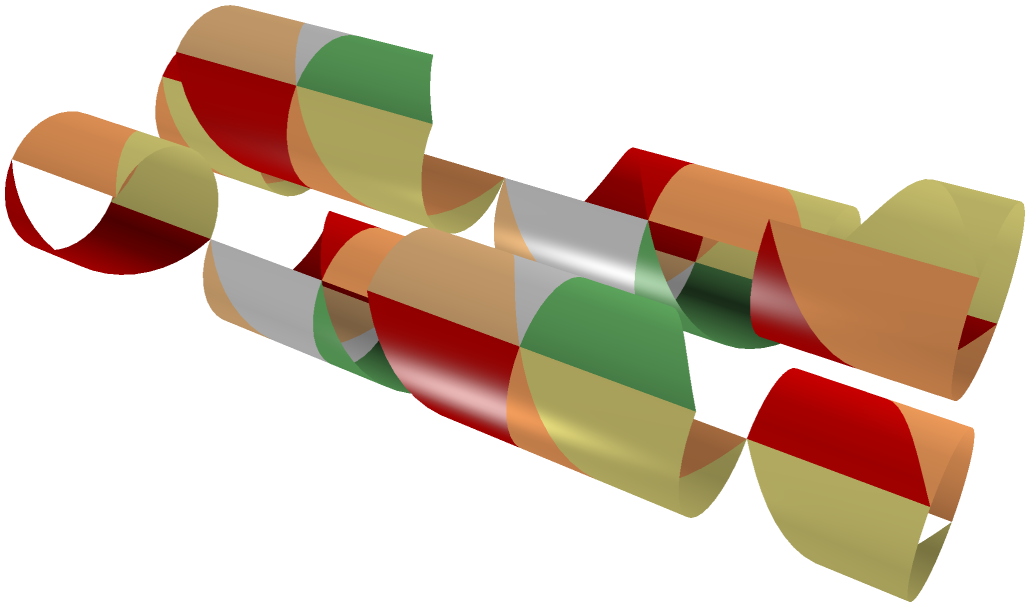
Vi kan følgelig kopiere en hvilken som helst parametrisk flate ved å triangulere domenet og bruke disse deltrekantene som lokale trekanter. Vær oppmerksom på at vi da må ha 3 like trekanter for å blende. Hvis alle trekanter som er koblet til ett toppunkt kan flyttes og/eller roteres, så har vi et verktøy for å formgi objekter. Dette er emnet for neste seksjon.

13.6 Flateapproksimasjon ved triangulering.

Enhver parametrisk flate kan trianguleres. Hvis vi har et sett med punkt (vertekser) i parameterplanet til en flate kan disse kobles sammen med linjer (kanter). Resultatet er en triangulert flate. Hvis flaten er lukket/syklisk i en eller begge retninger, må trianguleringen også kunne virke på sykliske domener. Vi tenker nå å lage en kopi av en flate med et sett med sammenkoblede trekantede blendingsflater. Konseptet er som følger,

1. Vi starter med et sett med punkt i parameterdomenet til en flate.
2. Vi triangulerer flaten ved å koble sammen to og to punkt med linjer (kanter).
3. Til hvert punkt tilordner vi et del-domene, det vil si et parameterområde som dekker alle trekanter i domenet, der punktet er et av hjørnene i trekanten. Et trekantdomene tilsvarer en trekantet delflate.
4. Til hvert punkt/del-flate tilordner vi en homogen matrise.
5. Vi finner nå hver trekantflate i 3 forskjellige del-flater. Disse tre trekantflatene kan nå brukes til å lage en blendingstrekant

I utgangspunktet får vi en nøyaktig kopi av en gitt flate, men formen kan senere endres ved å flytte/rotere punktene (delflatene) med den homogene matrisen som er tildelt punktet.



Figur 13.16: *Sylinderens 8 delflater med partisjonen angitt i figur 13.15 og tabellen nedenfor. Hver trekant av delflaten har egen farge.*

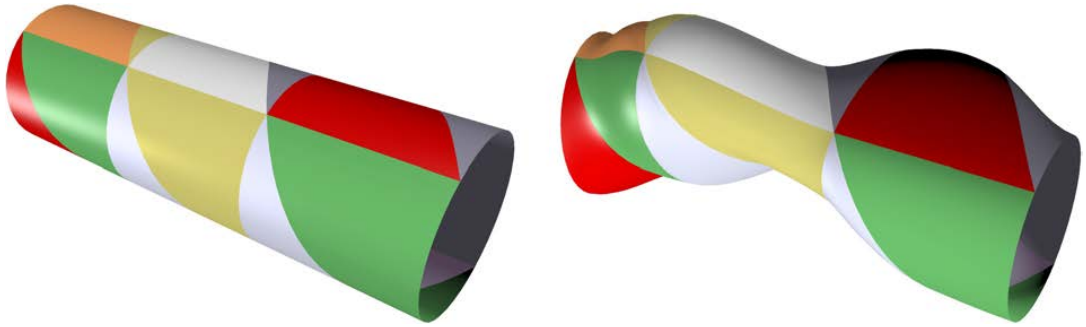
Vi skal nå se på to eksempler. Det første eksemplet er en sylinder, formel (9.2). Til venstre i figur 13.15 ser vi parameterdomenet med 8 punkt som er markert med rødt. Dette er "verteksene" for trianguleringen av domenet. De 4 punktene øverst er de samme som de 4 punktene nederst på grunn av den syklisk strukturen. Domenet er triangulert, og hver av de 12 trekantene er merket $\Delta_i,\ i = 1,2,\ldots,12$. Et underdomene Ω_{p_i} tilordnes også til hvert punkt $p_i,\ i = 0,1,\ldots,7$, se til høyre i figur 13.15. Disse underdomenene kan være av tre forskjellige typer, og i figuren er disse merket som type a), type b) og type c).

Parameterdomenet for en delflate er satt sammen av alle trekanter som har ett av hjørnene som det definerende punktet for parameterdomenet til delflaten. Tabellen nedenfor beskriver parameterdomenene til de 8 delflatene i sylindereksemplet.

delflatedomene	Trekantene som utgjør domenet	Type jmf. figur 13.15
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)

I kolonnen med trekantene i tabellen er det totalt 36 trekanter. Hver trekant er på nøyaktig tre forskjellige delflater. Trekanten Δ_1 er for eksempel i delflaten Ω_{p_0}, Ω_{p_1} og Ω_{p_4}, trekanten Δ_7 er i delflaten Ω_{p_1}, Ω_{p_5} og Ω_{p_6}.

Figur 13.16 viser de 8 delflatene i sylindereksemplet. Følgelig er de alle deler av en sylinder, og hver av dem er delt inn i trekanter som i figuren har sin egen farge. Sammen



Figur 13.17: *Til venstre ser vi en sylinder satt sammen av blendingstrekanter. Til høyre ser vi det samme settet med trekantede overflater, men nå er de deformert. Dette er fordi noen av delflatene er flytte i vertikale eller horisontale retninger.*

vil delflatene dekke sylinderen 3 ganger. Til hver delflate tilordnes en homogen matrise (se side 155). I praktisk implementering er delflatene bare representert som definisjonspunktet og et sett med trekanter i parameterdomenet samt den homogene matrisen. Hver av delflatene kan deretter flyttes, roteres, skaleres, ... (affine avbildninger, se side 16) med den homogene matrisen slik at for hver blendingstrekant får vi, på samme måte som uttrykk (13.12) (men uten det syklisk skifte) følgende uttrykk

$$S_i(\mathbf{u}) = B_0(\mathbf{u})\ H_0\ s_i(\mathbf{u}) + B_1(\mathbf{u}) H_1\ s_i(\mathbf{u}) + B_2(\mathbf{u}) H_2\ s_i(\mathbf{u})$$

hvor $\mathbf{u} = (u, v, w)$. Dette kan omformuleres til

$$S_i(\mathbf{u}) = H(\mathbf{u})\ s_i(\mathbf{u}) \tag{13.14}$$

hvor

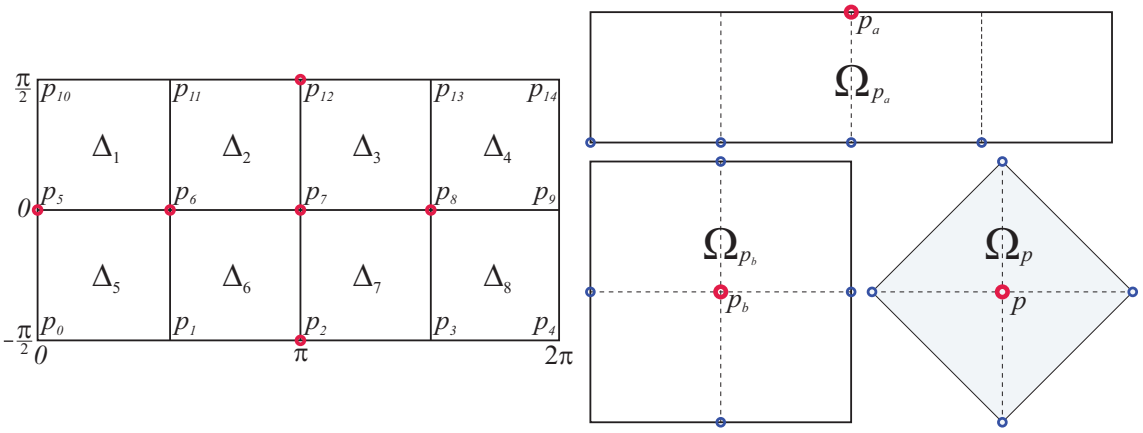
$$H(\mathbf{u}) = B_0(\mathbf{u})\ H_0 + B_1(\mathbf{u}) H_1 + B_2(\mathbf{u}) H_2. \tag{13.15}$$

og $s_i(\mathbf{u})$ er uttrykket for deltrekanten $\mathbb{S}(\Delta_i)$ for sylinderen $\mathbb{S}$. Her er H_0 matrisen koblet til punktet i hjørnet av trekanten der $u = 1$, H_1 er matrisen koblet til punktet i hjørnet av trekanten der $v = 1$, og H_2 er matrisen koblet til punktet i hjørnet av trekanten der $w = 1$.

Uttrykkene (13.14) og (13.15) er det generelle uttrykket for flate approksimasjon med triangulering. Konstruksjonen er i prinsippet den samme som konstruksjonen i seksjon 12.4, en delflatekonstruksjon som legger en editeringsstruktur på en flate. Til venstre i figur 13.17 er strukturen lagt på en sylinder. Det nye objektet er en nøyaktig kopi av den originale sylinderen. Konstruksjonen er imidlertid designet for endring av form. Til høyre i figur 13.17 er de homogene matrisene ikke lenger identitetsmatriser. Her er underflatene flyttet i vekslende vertikale og horisontale retninger. Men vi kan fortsatt gjenkjenne hver enkelt trekant og tydelig se deformasjonen av disse.

De deriverte er rett frem for å beregne:

$$\begin{aligned}
D_u S_i(\mathbf{u}) &= H_u(\mathbf{u})\ s_i((\mathbf{u})) + H(\mathbf{u})\ D_u s_i((\mathbf{u})) \\
D_v S_i(\mathbf{u}) &= H_v(\mathbf{u})\ s_i((\mathbf{u})) + H(\mathbf{u})\ D_v s_i((\mathbf{u})) \\
D_w S_i(\mathbf{u}) &= H_w(\mathbf{u})\ s_i((\mathbf{u})) + H(\mathbf{u})\ D_w s_i((\mathbf{u}))
\end{aligned}$$



Figur 13.18: *Til venstre ser vi parameterplanet til en sfære. Den er delt opp i 8 deler, som tilsynelatende ser ut som firkanter, men er relatert til trekanter fordi en kant (topp eller bunn) faktisk er ett punkt. I parameterplanet er 6 punkt markert med rødt, 4 rundt "ekvator", en øverst og en nederst. Til høyre ser vi parameterområdet for delflatene Ω_{p_a} og Ω_{p_b}, og transformasjonen av begge disse til en sammenslutning av trekanter er vist som en lyseblå rotert firkant som består av 4 trekanter.*

hvor

$$H_u(\mathbf{u}) = D_u B_0(\mathbf{u})\ H_0 + D_u B_1(\mathbf{u}) H_1 + D_u B_2(\mathbf{u}) H_2,$$

og hvor de andre deriverte kan beregnes på tilsvarende måte.

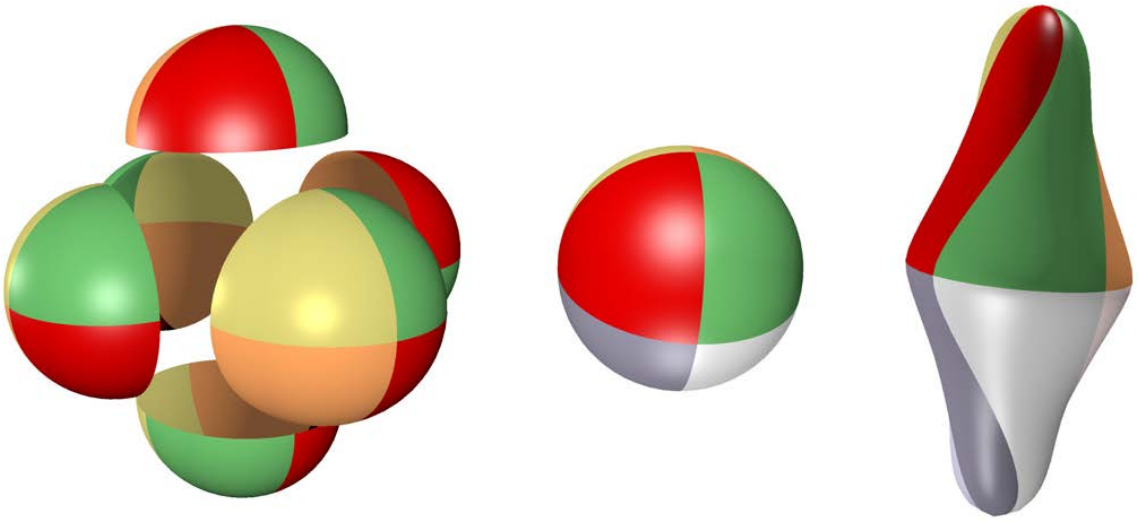
Det neste eksemplet er en sfære der vi bruker følgende formel,

$$\mathbb{S}(u,v) = \begin{pmatrix} \cos u\ \cos v \\ \sin u\ \cos v \\ \sin v \end{pmatrix}, \quad u \in [0, 2\pi), \quad v \in (-\pi, \pi). \tag{13.16}$$

Legg merke til at punktene på begge polene ikke er inkludert i formelen. De må legges til separat fordi formelen i disse punktene ikke er en en-til-en-avbildning mellom parameterplanet og $\mathbb{R}^3$, og at de er uregelmessige siden en av de partiellderivert er null og vi dermed ikke får et tangentplan og en normal.

Til venstre i figur 13.18 ser vi parameterplanet til en sfære med formelen (13.16). Parameterplanet er delt inn i 8 tilsynelatende firkanter, tilsynelatende fordi disse i rommet, $\mathbb{R}^3$, blir trekantede flater. Hele den øverste kanten og hele den nederst kanten er egentlig ett punkt hver, som markert i figur 13.18 med punktene p_2 og p_{12}. Høyre side av figur 13.18 viser de to typene av parameterdomener for delflatene, Ω_{p_a} og Ω_{p_b}. Det definerende punktet til domenene er markert med rødt, og det er 4 andre punkt (i blått) som definerer trekantene som til sammen danner domenet. Overgangen fra firkanter til trekanter er illustrert med den roterte lyseblå firkanten i figur 13.18. Overgangen fra trekant til firkant er:

$$g_i(\mathbf{u}) = g_i(u,v,w) = \begin{cases} p_k, & \text{hvis } u = 1 \\ w\ p_i + v\ p_j + u\left(\frac{w}{v+w} p_k + \frac{v}{v+w} p_h\right), & \text{ellers} \end{cases} \tag{13.17}$$



Figur 13.19: *Til venstre ser vi 6 delflater av en sfære. De er flyttet litt slik at vi lettere ser dem. I midten er en flate som er en samling av 8 blendingstrekanter og som er en nøyaktig kopi av en sfære. Til høyre ser vi samme flate, men hvor delflatene på nord- og sørpolen er flyttet fra hverandre og deretter rotert* 90° *rundt den vertikale senteraksen.*

der $u+v+w=1$, og hvor indeksene i,j,k,h er avhengig av partisjonen, for eksempel for Δ_1 får vi $(i,j,k,h)=(5,6,10,11)$ og for Δ_5 får vi $(i,j,k,h)=(6,5,1,0)$. Selv om vi ifølge figur 12.19 har to typer parameterdomener for delflatene, Ω_{p_a} og Ω_{p_b} , tilordnes disse til det samme, det vil si det lyseblå parameterdomenet Ω_p. delflatene er egentlig bare begrensninger i domenet. Hvis vi plotter dem, får vi 6 like flater (halvkule) som bare er orientert forskjellig. Dette er illustrert til venstre i figur 13.19. Flatene der er riktig orientert, men er blitt flyttet litt slik at de kan skilles fra hverandre. Hver liten trekant har egen farge, og hver har de små firkantede domene illustrert til venstre i figur 13.18.

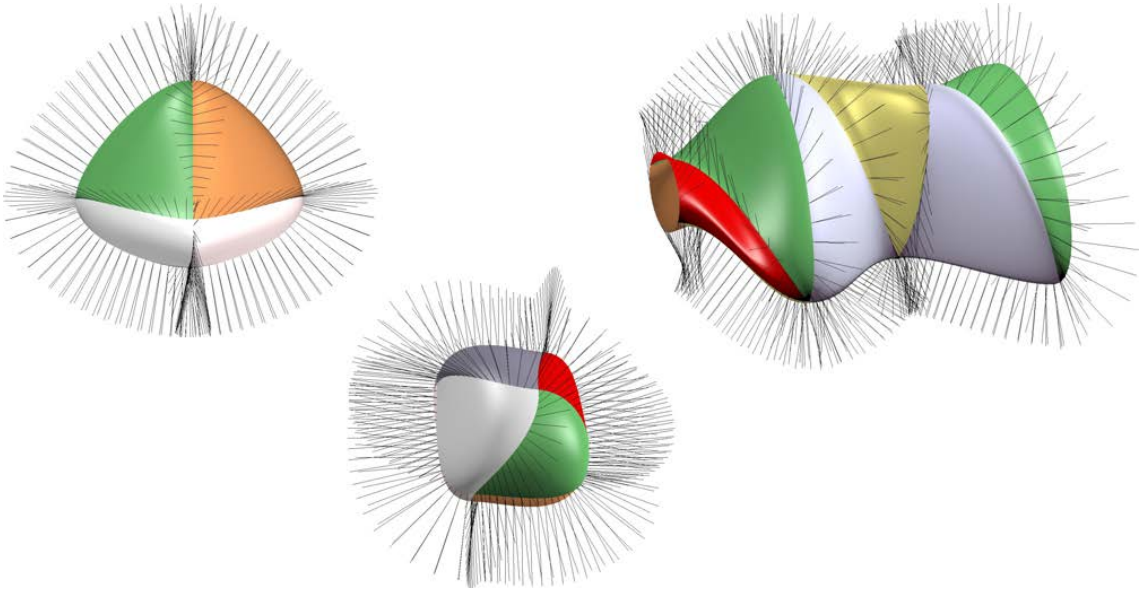
Formelen for deltrekanter som skal brukes i blendingen er

$$\begin{aligned} s_i(\mathbf{u}) &= \mathbb{S}\circ g_i(\mathbf{u}) \\ D_u s_i(\mathbf{u}) &= d\mathbb{S}_{\mathbf{u}}\left(\widetilde{w}p_k+\widetilde{v}p_h\right) \\ D_v s_i(\mathbf{u}) &= d\mathbb{S}_{\mathbf{u}}\left(p_j+\widetilde{u}\widetilde{w}\left(p_h-p_k\right)\right) \\ D_w s_i(\mathbf{u}) &= d\mathbb{S}_{\mathbf{u}}\left(p_i-\widetilde{u}\widetilde{v}\left(p_h-p_k\right)\right) \end{aligned}$$

hvor

$$\begin{array}{ll} \widetilde{u}=0, \quad \widetilde{v}=0, \quad \widetilde{w}=1 & \text{hvis } u=1, \\ \widetilde{u}=\frac{u}{v+w}, \quad \widetilde{v}=\frac{v}{v+w}, \quad \widetilde{w}=\frac{w}{v+w} & \text{ellers.} \end{array}$$

Legg merke til at $S_u(p_i)=(0,0,0)^T,\ i=0,1,2,3,4,10,11,12,13,14$, dvs. på nord- og sydpolen. Men siden "polpunktet" i hver trekant tilsvarer to punkt i parameterplanet, kan vi bruke S_v i begge punkt, følgelig kan vi sette $S_u(p_k)=S_v(p_h)$. Videre, for å lage en blendingstrekant bruker vi bare (13.14) og (13.15) der de homogene matrisene er koblet til 3 av de 6 punktene i midten av hver halvkule som vises som hjørner av trekanten. Vi har et eksempel på implementering. Midt i figur 13.19 er det 8 blendingstrekanter som til sammen utgjør en nøyaktig kopi av en sfære. På høyre side av figur 13.19 er halvkulene på Nord- og Sørpolen flyttet fra hverandre og rotert 90° rundt den vertikale sentertaksen.

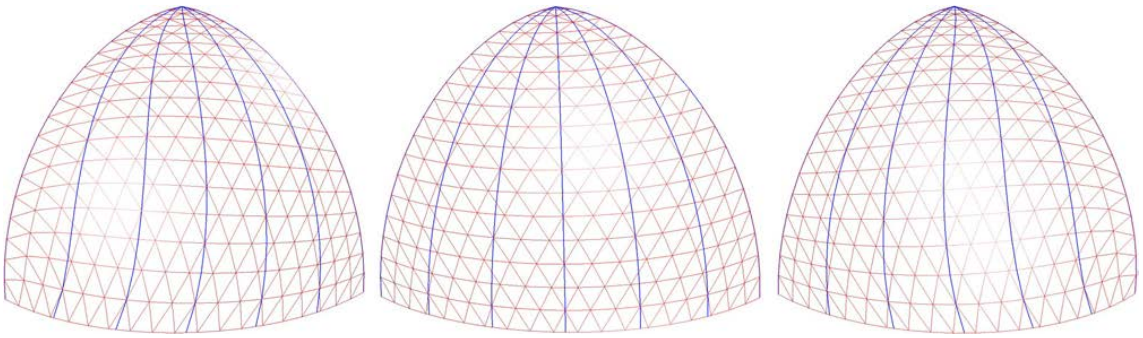


Figur 13.20: *Tre flater som alle er samlinger av trekantede blendingsflater. Normaler langs kantene på hver av trekantene i flatene er også vist. To av flatene er basert på en sfære og en er basert på en sylinder. Formålet med disse plottene er å visualisere kontinuitetsegenskapen.*

En viktig egenskap er kontinuitet. Konstruksjonen er åpenbart kontinuerlig, men når det gjelder deriverte, garanterer konstruksjonen bare kontinuitet opp til hermiteordenen til B-funksjonen, og dette bare i hjørnepunktene (punktene som delflatene og matrisene er koblet til). Dette følger av egenskapene til B-funksjonen. I figur 13.4 ser vi en trekantet B-funksjon. I toppunktet der verdien er 1, og på kanten motsatt til toppunktet, hvor verdien er 0 overalt, er alle deriverte også 0 opp til hermiteorden S. Spørsmålet er faktisk hvordan parametriseringen virker nær kantene. Fordi når trekanter deformeres, kan retningsderiverte ha forskjellig retning på hver side av en kant, og vi vil se at en knekk oppstår. Den eneste måten å forhindre dette på er å få en parametrisering som er slik at parameterlinjene (når forholdet mellom to parametere er låst) sammenfaller og helst er vinkelrett på kanten.[2] Figur 13.20 viser tre blendingsflater ved triangulering. Enhetsnormaler langs kantene på hver av trekantene er også plottet. To av flatene er konstruert fra en sfære der punktene etter opprettelsen er flyttet og/eller rotert (men forskjellig i de to eksemplene). Det tredje objektet er en deformert sylinder. En nærmere undersøkelse av sylindereksemplet viser oss at normalen sammenfaller i punktene og noen ganger i et punkt ca. midt på kantene. Det er her retningen til “parameterlinjene" sammenfaller på hver side av kanten. Det mest interessante er imidlertid de to sfæreeksemplene. De viser at på kantene mellom den nordlige og sørlige halvkule sammenfaller normalene alltid. Dette er faktisk et resultat av reparametriseringen i (13.17).

Figur 13.21 viser tre plott av samme trekantede flate (1/4 av en halvkule). 5 parameterlinjer (der én parameter varierer og forholdet mellom de to andre er konstant) vises i blått. Flatene, inkludert parameterlinjene, fra venstre mot høyre i figuren roteres flaten hver

[2] Dette ble først kommunisert som svar på et spørsmål fra Malcolm Sabin i mai 2007.



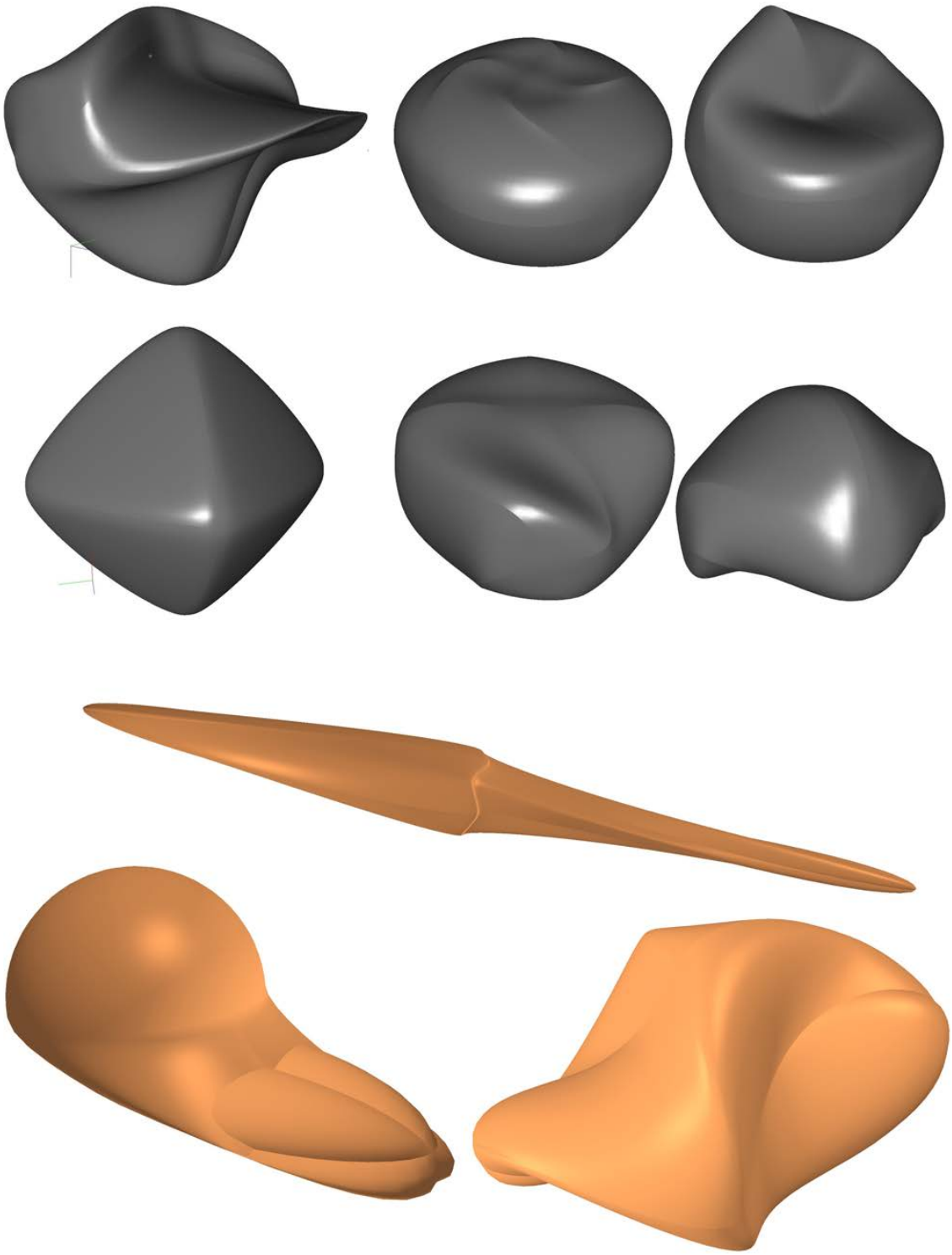
Figur 13.21: *Vi ser tre plott av samme trekantede flate (1/4 av en halvkule). Fra venstre mot høyre er flaten rotert* 120° *med klokken hver gang. Sju parameterlinjer vises i hvert av plottene. Vi ser at det bare er det midtre plottet av flaten at "parameterlinjene" kommer vinkelrett ned på kanten.*

gang 120° med klokken. Til venstre i figuren ser vi w-linjene, i midten u-linjene , og til høyre er det v-linjene. Vi ser at bare u-linjene er vinkelrett på kanten. Flaten i figur 13.21 er sfæren fra figur 13.19 og 13.20. u-linjene er de som kommer mot "ekvator", og det vi ser er da grunnen til at flaten er glatt over ekvator-kantene.

Mange har jobbet med å prøve å finne en generell metode for reparametrisering som gir i det minste gir glatte kanter. Så langt har ikke resultatene vært spesielt vellykkede. Det kan imidlertid være mulig å finne dette for spesielle trekantede flater i et delflatekonsept, men dette er fortsatt en åpen oppgave, og det er tvilsomt om det er teoretisk mulig, men denne forfatteren vet ikke om noe bevis på dette. Flateapproksimasjon med trekanter er muligens mest nyttig å bruke i forbindelse med friformmodellering i en kunstnerisk kontekst og/eller en mer spektakulær produktdesign. En metode er å starte med et topologisk objekt som samsvarer med ønsket resultat og deretter bestemme antall trekanter/"grad av finhet". Deretter kan vi lage en sammensatt flate og deretter endre form interaktivt ved å manipulere interpolasjonspunktene. Fra nå av er det to muligheter, enten å gjøre resultatet glatt ved å introdusere det duale settet med flater, se kapittel 14, eller å tessellere/generere en forfinet trekantstruktur som deretter går inn i en subdivisjonsalgoritme, dvs. bruke Loop-subdivisjonsskjema, se seksjon 10.1.3.

Figur 13.22 viser flere eksempler på modellering med flateapproksimasjon med trekanter. Alle eksemplene er basert på en sfære slik er definert tidligere. De 5 flatene øverst til høyre er hovedsakelig basert på rotasjoner av punktene, det vil si at rotasjon er å bruker de homogene matrisene H_i, beskrevet i (13.14) og (13.15). Den grå flaten nederst til venstre er bare at punktene som flyttes fra hverandre. De tre kobberfargede flatene er laget med større forflytninger og rotasjoner. Flaten nederst til venstre selvskjærer. Slik får vi den spesielle effekten på "nesa".

Mange har arbeidet med å sette sammen béziertrekanter på en glatt måte. Problemet er at det reduserer frihetsgrader mye, slik at konstruksjonen blir stiv og ikke veldig formbar. Du finner en teori omkring dette i [101]. Nylig har det blitt gjort arbeid med dette i samband med en isogeometrisk tilnærming.



Figur 13.22: *Alle flater vi ser i figuren, er basert på sfæren som er vist i figur 13.19. De eneste endringene er at interpoleringspunktene i $\mathbb{R}^3$ er flytte og rotert. Vær oppmerksom på at alle rotasjoner er utført rundt det aktuelle interpolasjonspunktet.*

Kapittel 14

En dual flatekonstruksjon

I forrige kapittel, Section 13.6, ble vi introdusert for *flateapproksimasjon ved triangulering*. Et konsept som er godt egnet for modellering og design, men som har én ulempe. Flatene er kontinuerlige, men det er ingen garanti for høyere ordens kontinuitet over kantene. I punktene er flaten derimot kontinuerlig opp til hermiteordenen til B-funksjonen som brukes. Men hvis vi ønsker en høyere grad av kontinuitet over kantene, kan dette gjøres ved å introdusere et dualt sett med firkantede flater over de kantene som vi ønsker å gjøre glatte. Dette ble introdusert i [109]).

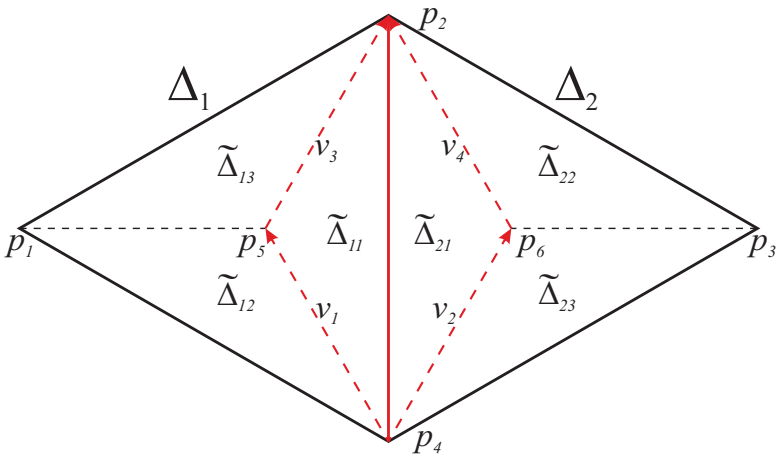
Hvis vi på flater som er satt sammen av et sett med trekantede flater har en kant vi vil gjøre glatt, da kan vi sette inn et punkt midt i parameterdomenet til hver av de to tilhørende trekantede flatene, det vil si $p_c = \frac{1}{3}(p_u + p_v + p_w)$, hvor p_u er punktet der $u = 1$, p_v er punktet der $v = 1$ og p_w er punktet der $w = 1$. I hver av de to originale trekantene får vi dermed tre mindre trekanter; $\Delta(p_u, p_v, p_c)$, $\Delta(p_c, p_v, p_w)$ og $\Delta(p_u, p_c, p_w)$, og hvor formelen for hver av disse blir:

$$
\begin{aligned}
s_{uvc}(u,v,w) &= s\left(u+\frac{w}{3}, v+\frac{w}{3}, \frac{w}{3}\right), \\
s_{cvw}(u,v,w) &= s\left(\frac{u}{3}, v+\frac{u}{3}, w+\frac{u}{3}\right), \\
s_{ucw}(u,,w) &= s\left(u+\frac{v}{3}, \frac{v}{3}, w+\frac{v}{3}\right).
\end{aligned}
\tag{14.1}
$$

Her er s en opprinnelige trekantede overflaten, og s_{uvc}, s_{cvw} og s_{ucw} er de trekantede flatene fra delingen. Husk at vi må bruke kjerneregelen for de partiellderiverte.

Delingen er illustrert i figur 14.1. Vi har parameterdomenene til to nærliggende trekanter som deler en kant som skal glattes. I begge trekantdomenene setter vi inn et nytt punkt i midten. Begge domenene blir så delt inn i 3 mindre trekanter. I midten av figuren har vi nå fått en rød stiplet firkant $\square(p_4, p_6, p_2, p_3)$. Diagonalen vises i heldekkende rødt, og er den kanten som skal glattes.

Husk at i den trekantede konstruksjonen har flaten en kontinuitet i punktene av orden lik heremiteordenen til B-funksjonen som brukes. Dette betyr at alle retningsderiverte av orden opp til heremiteordenen til B-funksjonen er like i alle trekantede flater som



Figur 14.1: *Figuren viser to trekanter, Δ_1 med punktene p_2, p_4, p_1 og Δ_2 med punktene p_2, p_3, p_4. Midtpunktene $p_5 = \frac{1}{3}(p_2+p_4+p_1)$ og $p_6 = \frac{1}{3}(p_2+p_3+p_4)$ er markert, og det samme gjelder vektorene v_1, v_2, v_3, v_4. De barysentriske koordinatene for punktene er $p_1 = (0,0,1)$ i Δ_1, $p_2 = (1,0,0)$ i både Δ_1 og Δ_2, $p_3 = (0,1,0)$ i Δ_2, $p_4 = (0,1,0)$ i Δ_1 og $p_4 = (0,0,1)$ i Δ_2. Alle seks deltrekantene er merket.*

deler et punkt i punktet. Dermed kan vi fylle området som er markert med røde stiplede linjer i figur 14.1, med en flate, enten ved hjelp av “Coons patch - bikubisk blending" vist i seksjon 9.6.2, hvis kravet er G^1-glatthet, eller ved hjelp av “to-flateblending", vist i kapittel 11, hvis det kreves en høyere orden av glatthet. Hvis det er en eller to kanter i en trekant som ikke skal glattes, bruker vi bare deltrekantene som er definert i (14.1).

Uavhengig av valg av flater, “Coons patch" eller “to-flateblending", trenger vi randkurver og vektorfunksjoner som beskriver retningsderiverte på tvers over rendene langs rendene. I figur 14.1 er både deltrekantene $\widetilde{\Delta}_{11}$ og $\widetilde{\Delta}_{21}$ og vektorene v_1, v_2, v_3 og v_4 markert.

14.1 Kurver og vektorfelt på trekantede flater

Figur 14.1 viser to trekanter, Δ_1 og Δ_2, som deler en kant. Disse to trekantene er domenene til to trekantede flater, S_1 og S_2. I trekant Δ_1 har vi to vektorer, $v_1 = p_5 - p_4$ og $v_3 = p_2 - p_5$. Videre definerer vi to kurver,

$$h_1(t) = p_4 + t\, v_1 \quad \text{og} \quad h_3(t) = p_5 + t\, v_3, \quad t \in [0,1]. \tag{14.2}$$

I trekant Δ_2 har vi to vektorer, $v_2 = p_6 - p_4$ og $v_3 = p_2 - p_6$. Videre definerer vi to kurver,

$$h_2(t) = p_4 + t\, v_2 \quad \text{og} \quad h_4(t) = p_6 + t\, v_4, \quad t \in [0,1]. \tag{14.3}$$

Husk at i trekantens parameterdomene er både punktene og vektorene i barysentriske koordinater som er relatert til hjørnene. Vi vet videre at i dette eksemplet er $u = 1$ i p_2 i begge trekanter, noe som ikke alltid vil være tilfelle. Hvis du bruker denne plasseringen, $p_4 = (0,1,0)$ i Δ_1, og $p_4 = (0,0,1)$ i Δ_2. Dette tilsvarer sfæreeksemplene i seksjon 13.6.

Hvis vi nå bruker barysentriske koordinater i kurveformuleringene, får vi

$$h_1(t) = \frac{1}{3}\begin{pmatrix} t \\ 3-2t \\ t \end{pmatrix}, \quad h_2(t) = \frac{1}{3}\begin{pmatrix} t \\ t \\ 3-2t \end{pmatrix}, \quad h_3(t) = h_4(t) = \frac{1}{3}\begin{pmatrix} 1+2t \\ 1-t \\ 1-t \end{pmatrix},$$

og fra (14.2) og (14.3) følger det at $h_1' = v_1$, $h_2' = v_2$, $h_3' = v_3$, og $h_4' = v_4$.

Det neste trinnet er avbildningen til $\mathbb{R}^3$. Vi har de trekantede flatene $S_j : \Delta_j \subset \mathbb{R}^2 \to \mathbb{R}^3$, j = 1,2. Kurvedefinisjonen vil være,

$$c_i(t) = S_j \circ h_i(t), \quad j = 1 \text{ og } i = 1,3 \quad \text{eller} \quad j = 2 \text{ og } i = 2,4, \tag{14.4}$$

og det følger at 1.- og 2.-deriverte for alle de fire kurvene er

$$\begin{aligned} c_i'(t) &= d(S_j)_{h_i(t)}(h_i'(t)), \\ c_i''(t) &= d(d(S_j)(h_i'(t)))_{h_i(t)}(h_i'(t)) + d(S_j)_{h_i(t)}(h_i''), \\ &= d(d(S_j)(h_i(t)'))_{h_i(t)}(h_i'(t)), \end{aligned} \tag{14.5}$$

fordi $h'' = 0$, og hvor

$$\begin{aligned} dS &= [S_u\ ,\ S_v,\ S_w]\,, \\ d(dS(h')) &= d\left([S_u\ ,\ S_v,\ S_w]\,(h')\right) \\ &= \left[[S_{uu}\ ,\ S_{uv},\ S_{uw}]\ h'\ ,\ [S_{vu}\ ,\ S_{vv}, S_{vw}]\ h',\ [S_{wu}\ ,\ S_{wv}, S_{ww}]\ h'\right]. \end{aligned}$$

Vektorfunksjoner på trekantede flater er neste emne. Vektorfunksjonene i parameterplanene som er relatert til de fire randkurvene, er(for $t \in [0,1]$) ,

$$\begin{aligned} g_1(t) &= \tilde{v}_2(t) + B(t)\,(v_3(t) - \tilde{v}_2(t))\,, \quad \text{i } \Delta_1, \\ g_2(t) &= \tilde{v}_1(t) + B(t)\,(v_4(t) - \tilde{v}_1(t))\,, \quad \text{i } \Delta_2, \\ g_3(t) &= v_1(t) + B(t)\,(\tilde{v}_4(t) - v_1(t))\,, \quad \text{i } \Delta_1, \\ g_4(t) &= v_2(t) + B(t)\,(\tilde{v}_3(t) - v_2(t))\,, \quad \text{i } \Delta_2, \end{aligned} \tag{14.6}$$

der $B(t)$ er en én-variabel B-funksjon med samme heremiteorden som er brukt i blendingstrekantene. $\tilde{v}$ betyr at vektoren må uttrykkes i koordinater som er koblet til domenet til nabotrekanten.

For å finne koordinatene til et punkt i en trekant som ligger i en annen trekant, bruker vi koordinatene til punktene p_i, $i = 1,2,3,4,5,6$ i parameterdomenet til den underliggende delflate. I Δ_2 er $p_6 = \frac{1}{3}(p_2 + p_3 + p_4)$, mens i Δ_1 er

$$\begin{aligned} u\,p_2 + v\,p_4 + (1-u-v)p_1 &= p_6, \\ (p_2 - p_1)u + (p_4 - p_1)v &= p_6 - p_1, \end{aligned}$$

og det følger at de barysentriske koordinatene til p_6 med hensyn til Δ_1 er

$$u = \frac{(p_6 - p_1) \wedge (p_4 - p_1)}{(p_2 - p_1) \wedge (p_4 - p_1)}, \quad v = \frac{(p_2 - p_1) \wedge (p_6 - p_1)}{(p_2 - p_1) \wedge (p_4 - p_1)} \quad \text{og} \quad w = 1 - u - v,$$

der $a \wedge b$ er kileproduktet i $\mathbb{R}^2$, beskrevet i slutten av seksjon 2.1.

De fire vektorverdifunksjonene (derivertefunksjoner) i $\mathbb{R}^3$ er da, for $j = 1$ og $i = 1,3$ eller $j = 2$ og $i = 2,4$

$$\begin{aligned} e_i(t) &= d(S_j)_{h_i(t)}(g_i(t)), \\ e_i'(t) &= d(d(S_j)(h_i'(t)))_{h_i(t)}(g_i(t)) + d(S_j)_{h_i(t)}(g_i'(t)), \end{aligned} \tag{14.7}$$

der $g(t)$ er definert i (14.6), $h(t)$ i (14.2) og (14.3). Legg merke til at sammenlignet med 2.-deriverte i (14.5) er den andre termen i 1.-deriverte i (14.7) inkludert fordi $g_i(t)$ ikke er en lineær funksjon på grunn av B-funksjonen.

14.2 En utfyllingsflate

Det neste trinnet er å opprette en flate som passer inn i det firkantede området som vises som stiplede røde linjer i figur 14.1. Vi trenger dermed rand-kurvene og -funksjonene som beskriver de deriverte “ortogonalt" over rendene.

I figur 14.1, og fra forrige seksjon ser vi at kurvene er organisert slik at $c_1(t)$ og $c_4(t)$ er på motsatte kanter av den firkantede flaten. Sammen med vektorfunksjonene $e_1(t)$ og $e_4(t)$, beskrevet i (14.7)), kan de brukes til å opprette en flate ved å blende kurver, som beskrevet i seksjon 9.4. Vi bruker så hermiteinterpolering som vist i (4.18) og (4.19). Resultatet er

$$S_1(u,v) = c_1(v)\, H_1(u) + c_4(v)\, H_2(u) + e_1(v)\, H_3(u) + e_4(v)\, H_3(u). \tag{14.8}$$

I den andre retningen er $c_2(t)$ og $c_3(t)$ på motsatte kanter, og sammen med $e_2(t)$ og $e_3(t)$ kan de da brukes til å lage en flate ved å blende kurver ved hjelp av hermiteinterpolering, og resultatet her er

$$S_2(u,v) = c_2(u)\, H_1(v) + c_3(u)\, H_2(v) + e_2(u)\, H_3(v) + e_3(u)\, H_3(v). \tag{14.9}$$

Det finnes nå minst to mulige måter å lage den endelige flaten på, enten ved å bruke “Coons Patch - bikubisk blending", beskrevet i seksjon 9.6.2, eller ved å bruke “to-flateblending", beskrevet i kapittel 11. I begge metodene antas det at hjørnene er konsistente sett fra begge sider, og at vi kan verifisere dette med å sammenligne (14.5) med (14.7). Husk at randkurvene følger vektorene v_1, v_2, v_3 og v_4 som vises i figur 14.1, og de har også samme retning. Husk også egenskapene til B-funksjonen, dvs. $B(0) = 0$, $B(1) = 1$ og $B'(0) = B'(1) = 0$. En undersøkelse ved hvert hjørnepunkt viser at:

I punktet p_2, der $c_3(1) = c_4(1)$ ser vi, ved å sammenligne (14.5) og (14.7) at

a) $c_3'(1) = e_4(1)$ fordi $h_3' = v_3$ og $g_4(1) = \tilde{v}_3$,

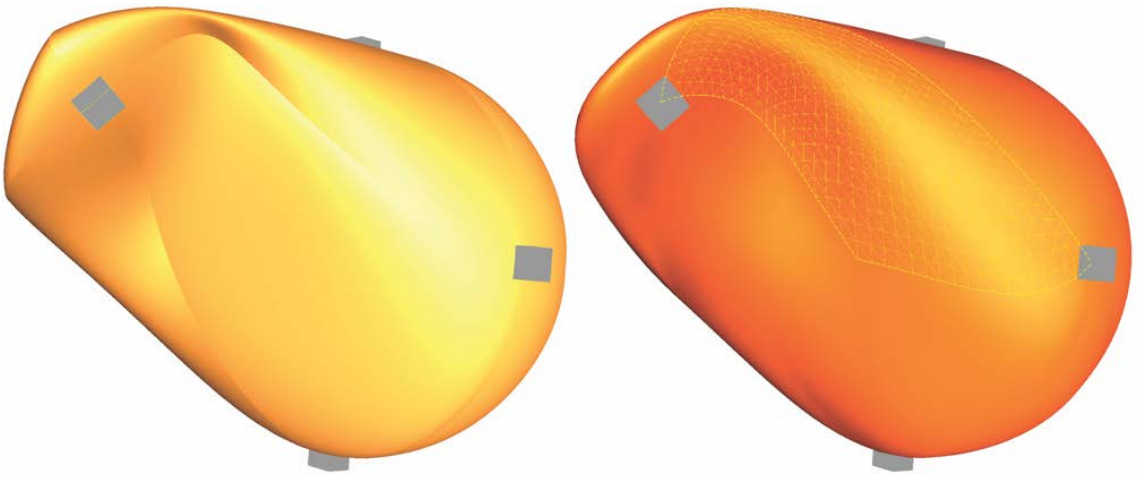
b) $c_3''(1) = e_4'(1)$ av samme grunn som punktet ovenfor, og at $B'(1) = 0$,

c) $c_4'(1) = e_3(1)$ fordi $h_4' = v_4$ og $g_3(1) = \tilde{v}_4$

d) $c_4''(1) = e_3'(1)$ av samme grunn som punktet ovenfor, og at $B'(1) = 0$,

I hjørnepunktet p_6, hvor $c_2(1) = c_4(0)$ ser vi at

a) $c_2'(1) = e_4(0)$ fordi $h_2' = v_2$ og $g_4(0) = v_2$,



Figur 14.2: *Vi ser to ganske like flater. Den til venstre er en "flateapproksimasjon ved triangulering". Flaten er i utgangspunktet en sfære, (13.16), med en triangulering som beskrevet i (13.17). Etter genereringen er interpoleringspunktene flyttet og rotert. Interpoleringspunktene vises som grå kuber. I flaten til høyre er kantene glattet med teknikken som er beskrevet i dette kapittelet. Et svakt synlig gult rutenettmønster skisserer den firkantede flate mellom to av interpoleringspunktene.*

b) $c_2''(1) = e_4'(0)$ av samme grunn som punktet ovenfor, og at $B'(0) = 0$,

c) $c_4'(0) = e_2(1)$ fordi $h_4' = v_4$ og $g_2(1) = v_4$

d) $c_4''(0) = e_2'(1)$ av samme grunn som punktet ovenfor, og at $B'(1) = 0$,

Ved de to andre hjørnepunktene p_4 der $c_1(0) = c_2(0)$, og p_5 der $c_1(1) = c_3(0)$, får vi et lignende resultat En annen ting er at 1.-deriverte fra (14.7) faktisk er den kryssderiverte i hjørnepunktet, det vil si at å endre den 1.-deriverte når vi beveger oss i den andre retningen. Derfor, i hjørnepunktene, må vi sørge for at disse deriverte er like fra begge sider. I p_4 er $e_1'(0) = e_2'(0)$. Ved å bruke (14.7) og finne de faktiske vektorene får vi $d(d(S_1)(v_1))_{p_4}(\tilde{v}_2) = d(d(S_2)(v_2))_{p_4}(\tilde{v}_1)$. Husk at dette er en vanlig delflate som gjør at alle retningsderiverte i S_1 og S_2 er like, og at differensialen er symmetrisk, det vil si $S_{uv} = S_{vu}$. Det samme vil vi observere i alle hjørnepunktene. Dette viser at systemet er konsistent i hjørnepunktene.

Dette lille beregningsstuntet som er gjordt for å vise konsistensen av punktene, viser egentlig bare egenskapen til konstruksjonen "flateapproksimasjon ved triangulering", som er at den totale flaten interpolerer hjørnepunktene med posisjon og alle deriverte opp til hermiteordenen til B-funksjonen som er brukt.

De to metodene er som følger:
"Coons patch - bikubisk blending" som er $S(u,v) = S_1(u,v) + S_2(u,v) - S_3(u,v)$ der S_1 er (14.8) og S_2 er (14.9), og S_3 er en Heremite-tensorproduktflate, se Avsnitt 9.5.1, hvor vi

må fylle en 4×4 matrise, dvs.

$$M = \begin{bmatrix} c_1(0) & c_1(1) & c_1'(0) & c_1'(1) \\ c_4(0) & c_4(1) & c_4'(0) & c_4'(1) \\ c_2'(0) & c_3'(0) & e_1'(0) & e_1'(1) \\ c_2'(1) & c_3'(1) & e_4'(0) & e_4'(1) \end{bmatrix}.$$

Den andre metoden er “to-flateblending" der vi bare bruker $S_1(u,v)$ og $S_2(u,v)$ sammen med en to variabel B-funksjon $B(u,v)$ beskrevet i seksjon 11.1.

Et eksempel er gitt i figur 14.2. Det er sfæreeksemplet fra seksjon 13.6. Noen av de 6 interpoleringspunktene er flyttet og rotert. Til venstre i figur 14.2 ser vi resultatet. Vi ser også klart kanter som ikke er glatte. Til høyre i figuren ser vi resultatet etter glatting av kantene. Ganske svakt kan vi også se et eksempel på en firkantet patch over en kant laget med “to-flateblending". Patchen er markert med et gult rutenettmønster. Interpoleringspunktene vises som grå kuber.

Vedlegg

Vedlegg A

Beregne ERB-funksjonen av type 1

Dette vedlegget er en sammenfatning og forenkling av kapittel 3 i [102], som kan lastes ned fra http://urn.nb.no/URN:NBN:no-15022

En “evaluator" for en ekspo-rasjonal B-funksjon av type 1 er å beregne $B(t)$ og dermed integralet i (7.31). Det vil si at vi må integrere eksponentialfunksjonen $\phi(t)$ som er definert i (7.27),eller $\phi(t;\gamma)$ definert i (7.38), eller $\phi(t;\gamma,\mu)$ definert i (7.39), eller $\phi(t;\gamma,\mu,\alpha,\beta)$ definert i (7.44).

I tillegg må vi også beregne de deriverte, $B^{(j)}(t)$ for $j = 1,2,...,d$ for d i alle fall opp til 3 og gjerne høyere. Derivasjon er beskrevet i seksjon 7.7.4, og der kommer det fram at beregningen må inkluderer $f_j(t)$, definert i (7.47). Alt dette innebærer håndtering av overflyt og underflyt og da også deling med null i brøker. Det innebærer stabil og presis numeriske integrasjoner, og metoder for å gjøre beregningene betydelih raskere.

Først må vi ta hensyn til implementering og da programmering, samt problemer relatert til flyttallsystemet på datamaskinen. I den første delen vil vi undersøke kravene til en pålitelig algoritme. Spørsmålene er overflyt, underflyt og divisjon med null, og vi vil undersøke disse med utgangspunkt i “IEEE binære flyttall"-standardiserte enheter. Den andre delen omhandler algoritmer for deriverte, og i den tredje delen, A.3, skal vi se på en algoritme for numerisk integrasjon av integralet

$$\int_{s=0}^{t} \phi(t;\gamma,\mu,\alpha,\beta)ds, \quad \text{hvor} \quad 0 < t \leq 1,$$

I [44] ble flere numeriske metoder vurdert for behandling av ERBS, som jo er det samme som type 1 ekspo-rasjonale B-funksjon. Her skal vi imidlertid bare se på metoden vi fant mest fordelaktig fordi vi ville ha en algoritme med god kontroll over presisjonen og som er enkel å implementere.

I den fjerde delen skal vi se på en konkret implementasjon og en test av en presis og ekstremt rask evaluator. Denne evaluatoren er basert på preevaluering samt hermiteinterpolasjon.

A.1 Pålitelighet i beregninger

Påliteligheten til en algoritme er avheng av sjansene for overflyt, underflyt og deling med null. Vi starter derfor med å se på hva disse tre fenomenene er og når de oppstår. IEEE-standarden for binær flyttallsaritmetikk [153] beskriver flyttallsformatene. I henhold til gjeldende standard skal binære flyttall være på formen

$$(-1)^s \, 2^E \, (b_0.b_1b_2\ldots b_{p-1}), \tag{A.1}$$

hvor p er presisjonen og

$$\begin{array}{ll} s \in \{0,1\} & \text{(binær)}, \\ E \in \{E_{min},\ldots,E_{max}\} & \text{(heltall med fortegn)}, \\ b_i \in \{0,1\} & \text{(binær)}. \end{array}$$

Tabellen nedenfor beskriver hvordan bittene i tallene er fordelt.

type presisjon	fortegn	signifikante bits (presisjon)	bits for eksponenten	sum bits
enkel	1	p = 23	8	32
dobbel	1	p = 52	11	64

(A.1) definerer de såkalte normalverdiene. I tillegg angir IEEE-standarden følgende spesialverdier for tall. ± 0 (null med fortegn), subnormale verdier, $\pm\infty$ og "signaliserende" og "stille" NaN (Not a Number). Vanligvis er den første signifikante bit b_0 1, fordi hvis den ikke er det kan man, for normalverdier, få det ved å redusere eksponenten E. For subnormaleverdier er ikke dette tilfelle, fordi vi nå allerede bruker $E_{min}-1$. Derfor, for tall med subnormale verdier, er den første signifikante biten alltid 0. Dette faktum gjør det mulig å hoppe over første bit, og dermed øke presisjonen (antall signifikante bits), fordi separasjonen mellom normale og subnormale verdier er godt definert uten første signifikante bit (se tabellen nedenfor). Tabellen nedenfor (hentet fra [75]) viser oss hvordan de 5 forskjellige typene av verdier kan skilles.

type	verdi	implementert eksponent	implementert presisjon
spesialverdi	± 0	$E = E_{min}-1$	$b = 0$
subnormalverdi	$\pm 0.b \times 2^{E_{min}}$	$E = E_{min}-1$	$b \neq 0$
normalverdi	$\pm 1.b \times 2^{E}$	$E_{min} \leq E \leq E_{max}$	
spesialverdi	$\pm\infty$	$E = E_{max}+1$	$b = 0$
spesialverdi	s/q NaN	$E = E_{max}+1$	$b \neq 0$

Ved hjelp av denne forbedrede presisjonen (å hoppe over første bit), får vi følgende antall signifikante sifre i desimaltallsystemet. For normale verdier er den største verdien for enkel presisjon (float) $2^{24}-1 = 16777215$, dvs. si mer enn 7 signifikante sifre, og for dobbel presisjon er den $2^{53}-1 = 9007199254740991$, dvs. omtrent 16 signifikante sifre. For subnormale verdier reduseres antall signifikante sifre gradvis til det til slutt bare er ett binært siffer igjen.

For å beskrive hva overflyt er og hvordan det skjer, skal vi først se på de maksimale normalverdiene,

$$\text{enkel - } 1.11\ldots11 \times 2^{2^{8-1}-1} = \left(2-2^{-23}\right)2^{127} \approx 3.4028237\, e+38.$$
$$\text{dobbel - } 1.11\ldots11 \times 2^{2^{11-1}-1} = \left(2-2^{-52}\right)2^{1023} \approx 1.7976931348623159\, e+308$$

Tall som blir større enn dette, settes til $\pm\infty$ avhengig av fortegnsbiten. For å beskrive hva underflyt er og hvordan det skjer, skal vi først se på minimum til normalverdiene.[1]

$$\text{enkel - } 1.00\ldots00 \times 2^{2-2^{8-1}} = 2^{-126} \approx 1.1754944\, e-38$$
$$\text{dobbel - } 1.00\ldots00 \times 2^{2-2^{11-1}} = 2^{-1022} \approx 2.22507385850720138\, e-308$$

Verdier som er mindre enn dette, er subnormalverdier med lavere presisjon. Legg merke til at den signifikante biten vil være $0.11\ldots11$ for den første subnormale verdien, og $0.00\ldots01$ for den siste subnormale verdien. Derfor er minimum subnormalverdi (uten fortegn),

$$\text{enkel - } 2^{2^{-126-23}} \approx 1.4012984\, e-45,$$
$$\text{dobbel - } 2^{2^{-1022-52}} \approx 4.9406564584124654\, e-324.$$

Tall som er mindre enn dette settes til ±0, avhengig av fortegnsbit. Når vi ser på tallene ovenfor, kan vi se at både for enkel og dobbel presisjon er

$$\frac{1}{\text{min normalverdi}} < \text{max normalverdi}.$$

Det følger av dette at hvis nevneren i en brøk har en normalverdi, og telleren er ≤ 1, vil ikke brøken produsere en overflyt.

Hvis du vil ha en nærmere studie av hvordan tallsystemet påvirker algoritmer, anbefaler vi [75]. Du finner en redigert versjon på https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html.

Oppsummering:

- Overflyt produserer et signal og $\pm\infty$, som ikke "lovlig" kan brukes videre i de fleste beregninger.
- Underflyt produserer først subnormalverdier og til slutt ±0.
- Brøker kan produsere en signalisert overflyt.
  - Deling med null gir klart en signalisert overflyt.
  - Hvis telleren i en brøk er ≤ 1, vil det aldri bli overflyt hvis nevneren er en normalverdi.

[1] Årsaken til at vi totalt bruker 3 færre tall i eksponenten enn det som er tilgjengelig, er at en brukes for null og 2 brukes henholdsvis til spesialverdi og subnormalverdi som vist i den forrige tabellen.

Så hvordan lage en pålitelig algoritme for en "ERB-evaluator"? En kritiske del er beregningen av brøken i uttrykkene i henholdsvis (7.27),(7.38), (7.39) og (7.44). Den siste er

$$-\gamma\frac{|t-\mu|^{\alpha+\beta}}{t^{\alpha}(1-t)^{\beta}}, \quad \text{hvor} \quad \gamma,\alpha,\beta>0, \quad \text{og} \quad 0<\mu<1 \quad \text{og} \quad t\in[0,1], \tag{A.2}$$

og senere også brøken $f_{j,k}(t),\ j=2,3,...$, beskrevet i seksjon 7.7.4. Når vi ser på telleren til brøken i (A.2), ser vi at $|t-\mu|<1$, fordi $0<\mu<1$ og $t\in[0,1]$, og dermed er

$$|t-\mu|^{\alpha+\beta}<1.$$

Vi får så følgende bemerkning og deretter algoritme.

Merknad A.1. *Det følger at det ikke er mulig å få overflyt når nevneren er en normalverdi. Den eneste kritiske delen i algoritmen og da beregningen av $\phi(t)$, og dermed uttrykket (A.2), er underflyt i nevneren, det vil si at tallet i nevneren ikke er en normalverdi.*

I det følgende gir vi først en algoritme for settet med alle iboende parametere, og deretter for standardsettet, det vil si når $\gamma=\alpha=\beta=1$ og $\mu=\frac{1}{2}$.

Algoritme 11. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen beregner $\phi(t;\gamma,\mu,\alpha,\beta)$ fra uttrykket (7.44), der γ, μ, α og β er tilgjengelige som tilstander, og $t\in[0,1]$ er innputt. Først følger en generell funksjon, så er det en optimalisert funksjon for standardsettet med iboende parametere, (7.27).*

double ϕ(double t)
    double $d = t^{\alpha}(1-t)^{\beta}$;
    if ($d \neq$ *normalverdi*) return 0.0; // *test om det er underflyt*
    else return $e^{-\gamma\frac{|t-\mu|^{\alpha+\beta}}{d}}$;

double ϕ(double t)
    double $d = t(1-t)$;
    if ($d \neq$ *normalverdi*) return 0.0; // *test om det er underflyt*
    else return $e^{-\frac{(t-\frac{1}{2})(t-\frac{1}{2})}{d}}$;

Husk at $\phi(t)$ er 1.-derivaterte til $B_d(t)$, som er en ERB-funksjonen av typen 1, (7.31). Formelen for 2.-deriverte er presentert i (7.27). I [102], side 51, vises det at det er mulig å beregne opptil 49 deriverte av ERB-funksjonen av type 1 med standardsettet med iboende parametere og da bare teste for underflyt for å få et gyldig og pålitelig resultat.

En generell funksjon som regner ut verdien og deriverte opp til orden 7 følger i neste seksjon. For praktisk bruk trenger vi imidlertid en mye raskere algoritme. Dette kan gjøres med å bruke preevaluering, numerisk integrasjon og heremiteinterpolasjon. Dette er temaet i seksjon A.4.

A.2 ERB-evaluering - verdi og deriverte

Nedenfor finner du en pålitelig algoritme som beregner verdien og inntil 6 deriverte av ERB-funksjonen, type 1 med standardsettet av iboende parametere.

Algoritme 12. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen beregner $D^jB(t),\ j=0,1,...,d,$ der $\phi(t)$ er definert i (7.27). Det er en implementering basert på seksjon 7.7.4. Algoritmen forutsetter at det er tilgjengelig funksjoner som beregner både $\phi(t)$, og $\int_{s=0}^{t}\phi(s)ds$, $t\in[0,1]$. Innputt er: $t\in[0,1]$ og $d\in\{0,1,\ldots,6\}$ (antall derivater som skal beregnes). Returen er en* "vector⟨double⟩", *der det første elementet inneholder $B(t)$, og deretter de d deriverte $B'(t),\ldots,B^{(d)}(t)$.*

vector⟨double⟩ **B (** double *t,* int *d* **)**
  vector⟨double⟩ $R(d+1)=0.0$; *// returvektor, størrelse d+1, alle elementene er 0.*
  double $\phi=\phi(t)$; *// kaller algoritme 11, part 2.*
  if *($\phi\neq$ normalverdi)* return *R;* *// test om underflyt.*
  $R=\mathbb{S}$; *// alle $d+1$ elementer settes lik $\mathbb{S}$, (7.32).*
  double $\widetilde{t}=(1-2t)^2$;
  switch *(d)*
    case *6:* $R_6*=\big(\big(\big(\big(\big(\frac{45}{2}\widetilde{t}+\frac{135}{2}\big)\widetilde{t}-\frac{765}{4}\big)\widetilde{t}+75\big)\widetilde{t}+66\big)\widetilde{t}-\frac{85}{2}\big)\widetilde{t}+\frac{15}{4}$; *// se*[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:* $R_4*=\big(\big(3\widetilde{t}+\frac{3}{2}\big)\widetilde{t}-5\big)\widetilde{t}+\frac{3}{2}$; *// se seksjon 7.7.4*
    case *3:* $R_3*=\frac{3}{2}\widetilde{t}^2-\frac{1}{2}$;
  $R_0*=\int_{s=0}^{t}\phi(s)ds$; *// se algoritme14 i neste seksjon.*
  $R_1*=\phi$;
  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;*

Denne algoritmen er for standard iboende parametere, det vil si $\phi(t)$ definert i (7.27). Hvis vi fritt vil velge iboende parametere, må vi bruke $\phi(t;\gamma,\mu,\alpha,\beta)$ definert i (7.44). Algoritmen blir da litt mer kompleks. I algoritme 12, $f_j(t)$, $j=1,2,...$ som er definert i seksjon 7.7.4 er formelen dissekert, og telleren og nevneren blir beregnet separat. I en generell algoritme med ikke-standard iboende parametere, vil ikke dette være formålstjenlig. Side 52-57 i [102] viser algoritmer for evaluering av $f_2(t)$ og $f_3(t)$ og $B_d(t,\gamma,\mu,\alpha,\beta)$ med 3 deriverte. I samme seksjon diskuteres restriksjoner på de iboende parameterne som må til for å oppnå en kontinuerlig og dermed korrekt B-funksjon, og også for å vise kombinasjoner av verdier på iboende parameter der asymptotisk oppførsel på deriverte oppstår.

I det følgende skal vi se nærmere på $f_2(t)$. I [102] vises det at den er kontinuerlig hvis $\alpha+\beta>1$ og at algoritmene kan håndtere asymptoter. Husk også de opprinnelige begrensningene for de iboende parameterne fra (7.44). Ved å kombinere dette med formelen

[2]Formlene for "case:" 2, 3 og 4 er vist i seksjon 7.7.4 og formler for "case:" 5 og 6 finnes i [102], side 33, som kan lastes ned fra http://urn.nb.no/URN:NBN:no-15022.

på sidene 28, 29 og 53 i [102] får vi

$$f_2(t) = \begin{cases} 0, & \text{hvis } t = \mu, \\ \mathbb{S}\, x_2(t)\zeta(t), & \text{ellers.} \end{cases} \tag{A.3}$$

hvor $\zeta(t)$ er eksponenten i (7.44) og

$$x_2(t)\zeta(t) = -\gamma \frac{\alpha+\beta}{t^\alpha(1-t)^\beta}\left(\frac{t-\frac{\alpha}{\alpha+\beta}}{t(1-t)}|t-\mu| + \text{sign}(t-\mu)1\right)|t-\mu|^{\alpha+\beta-1}. \tag{A.4}$$

Husk fra (7.32) at $\mathbb{S} = \left[\int_0^1 \phi(t;\gamma,\mu,\alpha,\beta)dt\right]^{-1}$. Nå følger en algoritme som beregner $f_2(t)$.

Algoritme 13. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen beregner $f_2(t)$ for ERB-funksjonen med ikke-standard iboende parametere. γ, μ, α og β må være tilgjengelig, og $\mathbb{S} = \mathbb{S}(\gamma,\mu,\alpha,\beta)$ må være pre-evaluert. Algoritmen er implementeringen av (A.3) og (A.4). Innputt må være $t \in [0,1]$, og det er slik at den andre linjen i algoritmen skal garantere at t i beregningen av (A.4) har en normalverdi på det åpne segmentet $(0,1)$.*[3]

```
double f2( double t )
    if ( t < 2.3e-308 || t == μ || t == 1 )    // Se (A.3), øvre del.
        return 0.0;
    double h = (t - α/(α+β)) / (t(1-t)) |t-μ|;  // første del av andre faktor i (A.4)
    if ( t < μ )   h -= 1;                      // siste del av andre faktor i (A.4)
    else           h += 1;
    h* = -S γ α+β / t^α (1-t)^β;                // setter inn S og første faktor i (A.4)
    if ( α+β < 1 )                              // vi har en asymptoten når t = μ
        double g = |t-μ|^(1-α-β) / h;           // den inverse av (A.4).
        if ( g ≠ normalverdi )
            return 0.0;
        else
            return 1/g;
    else if (α+β > 1)                           // ordinær løsning.
        return h |t-μ|^(α+β-1);
    else                                        // diskontinuitet ved t = μ.
        return h;
```

En kommentar til den foregående algoritmen; Hva gjør vi med en asymptote når $\alpha+\beta < 1$? Spørsmålet er hva vi bør gjøre når $t = \mu$ og vi ikke har noen verdi å returnere? Det logiske er å returnere 0, som vi vil gjøre i alle tilfeller når vi har en verdi.

[3] Garantien innebærer at vi må innføre praktiske begrensninger på de iboende parameterne på grunn av det binære tallsystemet.

A.3 Bruk av rombergintegrasjon i evaluering

Hoveddelen av evalueringen av ERB-funksjonen $B_d(t)$ definert i (7.31) er integrasjonen av $\phi(t)$ definert i (7.27). Her skal vi se nærmere på en pålitelig og kontrollerbar numerisk integrasjon, nemlig rombergintegrasjonen, se [137] og [25]. Rombergintegrasjonen er basert på gjentatte richardsonekstrapoleringer for å fjerne feil-ledd, se [92]. Bakgrunnen for algoritmen er idxEuler-MacLaurins integreringsformelen, se [157]. Den sier; gitt en funksjon f som er $C^{\infty}[a,b]$, da vil feilen fra en trapesapproksimasjon $T_n(f)$ i forhold til integralet $I(f)$ være

$$I(f) - T_n(f) = \frac{1}{n^2}\sum_{j=0}^{\infty} A_j^{(0)} \frac{1}{n^{2j}} = \frac{1}{n^2}A_0^{(0)} + \frac{1}{n^4}A_1^{(0)} + \frac{1}{n^6}A_2^{(0)} + \ldots, \tag{A.5}$$

hvor $A_j^{(0)}$ er konstanter. For eksempel i feilformelen til Trapes- og Simpson-metoden er

$$\begin{aligned} A_0^{(0)} &= \tfrac{-(b-a)^2}{12}(f'(b) - f'(a)), \\ A_1^{(0)} &= \tfrac{(b-a)^4}{180}(f^{(3)}(b) - f^{(3)}(a)). \end{aligned}$$

Richardsonekstrapolering kan brukes sammen med (A.5) for å eliminere termer i feiluttrykket. Richardsonekstrapolering er generelt en metode for å forbedre en approksimasjon ved å kombinere to uttrykk med forskjellige trinn. Hvis vi lager to forenklede versjoner av (A.5), bruker trinnstørrelse h i stedet for antall trinn n, og så bruker halve trinnstørrelsen, h/2 i den andre formuleringen, får vi

$$\begin{aligned} I(f) &= T_h(f) + A_0 h^{2j} + O(h^{2j+1}), \\ I(f) &= T_{h/2}(f) + A_0 \left(\tfrac{h}{2}\right)^{2j} + O(h^{2j+1}). \end{aligned}$$

Hvis vi deler uttrykk to med $\left(\frac{h}{2}\right)^{2j}$ og trekker det fra det første uttrykket, får vi

$$I(f) = B(h) + O(h^{2j+1}).$$

hvor

$$B(h) = \frac{2^{2j}T_{h/2}(f) - T_h(f)}{2^{2j} - 1}, \tag{A.6}$$

Hvis vi også beregner $B(h/2)$ kan vi bruke $B(h)$ og $B(h/2)$ i et neste trinn (i (A.6)) for å redusere feilledd. Dette kan gjentas i en iterativ prosess til det fjernede feilleddet er mindre enn en gitt toleranse.

Vi kan stille tre spørsmål når det gjelder bruk av rombergintegrasjon for å integrere $\phi(t)$,

- ✓ pålitelighet,
- ✓ effektivitet,
- ✓ presisjon.

kurvens navn		**a**	**b**	**c**	**d**	**e**	**f**
øvre grense		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	$1.7e-16$					

Tabell A.1: *Tabellen viser sammenhengen mellom antall trinn/evalueringer og presisjonen i rombergalgoritmen. Det er 6 senario som er beregnet. Hvert senario er en integrasjon fra 0 til øvre grense (andre rad). Kolonnen til venstre viser antall trinn i integrasjonen, den neste kolonnen viser antall nye beregninger av $\phi(t)$ som må gjøres på gjeldende trinn. Tallene i de resterende kolonnene er korreksjonen som er gjort i det aktuelle trinnet, og som da indikerer gjenstående feil.*

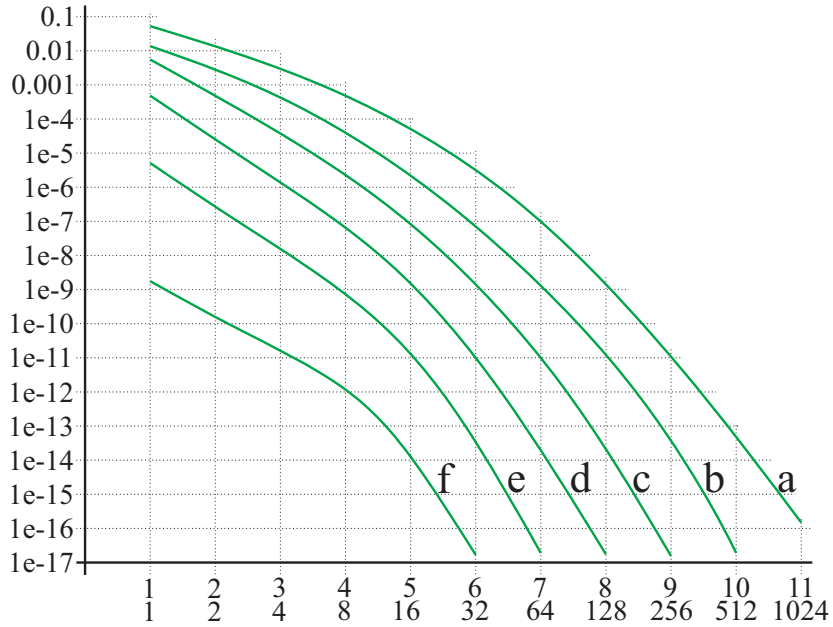
Det vil ikke bli gjort forsøk på å sammenligne rombergintegrasjon med andre metoder, men det er gjort en undersøkelse av hvor godt rombergintegrasjon fungerer for integrasjoner av $\phi(t)$, og tester er gjort på flere integrasjonsintervaller for å finne ut:
- hvor raskt kvadraturprosessen konvergerer,
- antall beregninger som er brukt på $\phi(t)$,
- og de resterende feilene.
Resultatet finner vi både i tabell A.1 og i figur A.1.

Først er påliteligheten ikke bare avhengig av at domenet til $\phi(t)$ er avgrenset til $[0,1]$, men også av det faktum at vi ikke får en overflyt når vi beregner $\phi(t)$ ved hjelp av algoritme 11, slik merknad A.1 påpeker. Siden integrasjonsintervallet er innenfor $[0,1)$, vil beregningen av integralet alltid gi en normalverdi og er med det pålitelig. Graden av "pålitelighet", feilen, vil bli nærmere diskutert nedenfor.

I tabell A.1 og i figur A.1 kan vi se sammenhengen mellom antall trinn og antall beregninger av $\phi(t)$ i rombergintegrasjonen, og den sist feilen som er fjernet. Det er foretatt integrasjon over 6 forskjellige intervall, dvs.

$$\int_{s=0}^{t} \phi(s)ds, \quad \text{for} \quad t = \left\{ \frac{1}{2^k} \right\}_{k=1}^{6}.$$

Hver integrasjon er navngitt med en bokstav, **a** betyr at integreringsintervallet er $\left[0, \frac{1}{2}\right]$, som er integrasjonen over det største intervallet. Deretter halveres intervallet fra integrasjonen til integrasjonen, til den siste integrasjonen kalt **f**, der intervallet er $\left[0, \frac{1}{64}\right]$. I



Figur A.1: *Figuren er et plott av tabellen A.1. Det er 6 integral,* **a, b, c, d, e** *og* **f**. *Integralene er over intervallene for* **a** $[0,\frac{1}{2}]$, **b** $[0,\frac{1}{4}]$, **c** $[0,\frac{1}{8}]$, **d** $[0,\frac{1}{16}]$, **e** $[0,\frac{1}{32}]$ *og for* **f** $[0,\frac{1}{64}]$, *Den vertikale aksen angir feilen, og den horisontale aksen angir øverst antall trinn, og nederst antall nye beregninger.*

tabellen A.1 finner vi verdien til siste fjernede “feiltermen" for de forskjellige trinnene til det ikke er “flere bits å fjerne". Verdiene er da forskjellene mellom resultatet fra dette trinnet og resultatet fra forrige trinn. Som vi kan se i tabell A.1 er en verdi i tabellen klart større enn summen av alle verdiene nedenfor i samme kolonne. Dette betyr at verdiene faktisk er større enn “gjenværende feil". Vi kan også se at de siste verdiene (siste i hver kolonne) er tall “utenfor kanten av signifikante bits" når resultatene er nær 1. I figur A.1 er verdiene fra tabellen A.1 tegnet inn som 6 kurver. Nederst på den horisontale aksen ser vi antallet nye beregninger av $\phi(t)$. Den forteller oss at beregningskostnadene dobles for hvert nytt trinn i integrasjonen.

Dybden i den iterative prosessen avhenger av størrelsen på integrasjonsintervallet. Integrasjonen av et intervall på 0,5 krever opptil 11 trinn, som betyr $2 \times 1024 = 2048$ beregninger av $\phi(t)$. Dette er en tidkrevende prosess. Dette betyr at man ikke bør bruke høyere toleranse enn nødvendig. Når man integrerer domener som er større enn $\frac{1}{2}$, bør man bruke speiling og å integrere fra høyre i domenet, dvs. $B(t) = 1 - \mathbb{S}\int_{s=t}^{1}\phi(s)ds$, når $t > \frac{1}{2}$.

Algoritmen som følger er spesialtilpasset for å integrere fra 0 til t. Hvis du ønsker å lage en algoritme for generelle intervall, for eksempel $[a,b]$, må endringer gjøre på linje 3, der det da må stå $\frac{\phi(a)+\phi(b)}{2}$, og på linje 9, som må være s += $\phi(a+j*t)$. I tillegg må selvfølgelig en ny deklarering “double $t = b-a$;" inkluderes. Algoritmen er optimal i den betydning at antall “evalueringer" av $\phi(t)$ er minimert. Vi vet hva $\phi(0)$ er og i noen tilfeller vet vi hva $\phi(t)$ er. Dermed kan algoritmen tilpasses enten ved å bruke tilstandsvariabler eller ved bruk av innputtparametere.

Algoritme 14. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen beregner integralet* $\widetilde{B} = \int_{s=0}^{t} \phi(s)ds,\ t \in (0,1]$*, der* $\widetilde{B}$ *er innenfor den angitte toleransen* ε*. Verdien til innputtvariabelen* ε *anbefales å være i området* $[10^{-3}, 10^{-15}]$ *(i henhold til tabell A.1).*

```
double integrate ( double t, double ε )
    double M[16][16];
    double sum = φ(t)/2;                    // kall til algoritme 11
    M_{0,0} = t * sum;
    for ( int i=1; i < 16; i++ )
        double s = 0;
        int k = 2^i;                        // C++ implementasjon: k = 1 << i
        t /= 2;
        for ( int j=1; j < k; j+=2 )  s += φ(j*t);    // kall til algoritme 11
        M_{0,i} = t * (sum += s);
        for ( int j=1; j ≤ i; j++ )
            double c = 4^j;                 // C++ implementasjon: c = 1 << (j << 1)
            M_{j,i-j} = (c*M_{j-1,i-j+1} - M_{j-1,i-j}) / (c-1);   // Richardson ekstrapolasjonsskjema
        if ( |M_{i,0} - M_{i-1,0}| < ε )  return M_{i,0};
    return M_{15,0};
```

A.4 En rask ERB-evaluator basert på approksimasjon

En rask "evaluator" for ERB-funksjoner av type 1 er absolutt nødvendig for at ERB-funksjonen skal være anvendelig til å lage kurver og flater. Det betyr at vi ønsker å ha en rask, men samtidig enkel algoritme med et enkelt brukergrensesnitt. Spesielt viktig er det fordi blendingsplines er spesielt egnet for interaktiv design i grafikkmodus, for direkte simuleringer der formendring er en viktig faktor, og i tilfeller der hermiteinterpolering er enten nødvendig eller å foretrekke.

Selv om den må være raskt og enkel å bruke, må den også være pålitelig og presist. Vi skal nå se nærmere på en metode som baserer seg på forhåndsevaluering (preevaluering) i gitte sampelverdier, dvs. verdi og 1.-deriverte. Vi deler domenet inn i for eksempel 1024-intervall og definerer deretter et 3.-grads polynom i hvert intervall ved å bruke hermiteinterpolering.

I praktisk implementering bør hele "evalueringssystemet" pakkes inn i et "object" (C++ -klasse eller tilsvarende). Fordelen med det er at det tar seg av alle tilstander som for eksempel de iboende parameterne $\phi(t;\alpha,\beta,\gamma,\lambda)$ jmf. (7.44), skaleringsfaktoren $S(\gamma,\mu,\alpha,\beta)$ og selvfølgelig antall sample-intervall, nedenfor angitt med m, og alle sampelverdier, mer enn $6m$ totalt, som beskrevet nedenfor. Derfor vil en evalueringsobjekttype i det følgende bli definert. Vi vil kalle den for "ERB-evaluator" og den inneholder følgende:

"ERB-evaluator"

Følgende tilstandsvariabler er tilgjengelige:

$\gamma,\ \mu,\ \alpha,\ \beta$	// Iboende parameter.
m	// Antall sampleintervall, antall sampelverdier er $m+1$.
$\Delta t = \frac{1}{m}$	// Intervallet mellom hvert sampel (også skaleringsfaktor),
	// sampel-vektoren $\{t_i\}_{i=0}^{m}$ definert via $t_i = i * \Delta t$.
$S = \mathbb{S}(\alpha,\beta,\gamma,\lambda)$	// Skaleringsfaktor, $\mathbb{S}(\gamma,\mu,\alpha,\beta) = \int_0^1 \phi(s;\gamma,\mu,\alpha,\beta)ds$.
$\mathbf{b} = \left\{\int_0^{t_i} \phi(s)ds\right\}_{i=0}^{m}$	// Vektor for lagring av integralet $\int_0^{t_i} \phi(s)ds$, hvor $t_i = i * \Delta t$.
$\mathbf{a}$	// En matrise med dimensjon $m \times 5$ (egentlig m vektorer).
	// Hver av vektorene $\{\mathbf{a}_i\}_{i=0}^{m}$ lagrer hermitekoeffisientene
	// a_0, a_1, a_2, a_3 og a_4 for hvert sampelintervall (se A.12).

Det vil selvfølgelig være nødvendig å ha "constructorer", "destructor", innstillinger etc. De vil ikke bli håndtert her, men de viktige utad tilgjengelige funksjonene er:

$initiate(\gamma,\mu,\alpha,\beta,m)$	// Endre tilstander og dermed beregne S, $\mathbf{b}$, $\mathbf{a}$ og Δt på nytt.
$B(t,d)$	// for en gitt $t \in [0,1]$, d - og antall deriverte,
	// analogt med algoritme 12, beregner $D^j B(t),\ j = 0,\ldots,d$.

For internt bruk (brukes bare av initiate( )) trenger vi følgende funksjoner (de tre siste funksjonene er definert i tidligere):

$interpolate(i,\phi_0,\phi_1,\phi_0',\phi_1')$	// Beregne $\mathbf{a}_{i,0},\ \mathbf{a}_{i,1},\ \mathbf{a}_{i,2},\ \mathbf{a}_{i,3}$ og $\mathbf{a}_{i,4}$ (med (A.12)).
$\phi(t)$	// Bruk algoritme 11.
$f_2(t)$	// Bruk algoritme 13.
$integrate(t_0,t_1,\phi_0,\phi_1,\varepsilon)$	// Modifisert versjon av algoritmen 14. Denne versjonen
	// kjenner start og sluttverdien for intervallet, dvs.
	// $\phi(start)$ og $\phi(end)$, for å minimere antall evalueringer.

Et betimelig spørsmål er; hvordan skal vi bruke "ERB-evaluatoren"? Et mulig sett med svar på dette er derfor listet opp nedenfor.

- ✓ For et gitt sett med iboende parametere og et gitt "akseptabelt avvik", lag en instans av en "ERB-evaluator". Alle kurver og flater som bruker dette settet med parametere, kan nå bruke dette objektet for å "evaluere" en ERB-funksjon.
- ✓ Parameterne eller "akseptabel avvik (toleranse)" kan når som helst endres for en eller flere spesifikke kurver/flater ved å lage en ny "ERB-evaluator", eller ved å endre parameterne i den gamle "ERB-evaluator".
- ✓ Det er mulig å ha mer enn én "ERB-evaluator" tilgjengelig samtidig.
- ✓ Det er mulig å ha flere instanser av samme kurve/flate, men med forskjellige "ERB-evaluatorer" som kan ha forskjellige iboende parametere.

Det er en åpenbar kostnad ved bruk av ERBS-evaluatorer, og det er bruk av minne. Dette er imidlertid relativt små tall, fra omtrent 1,8 til 48 kB for hver instans av et evaluatorobjekt, avhengig av antall sampelverdier, og med det minste tillatte avvik/feil. Sampeldataene består av vektoren **b**, som lagrer alle inkrementelle integreringsdataene, og matrisen **a**, som lagrer hermiteinterpoleringskoeffisientene i hvert sampelintervall. Bare de fire første verdiene på hver linje, $\mathbf{a}_{i,0}$, $\mathbf{a}_{i,1}$, $\mathbf{a}_{i,2}$, $\mathbf{a}_{i,3}$, er egentlig hermitekoeffisientene. Den siste, $\mathbf{a}_{i,4}$ er integralet over hele sampelintervallet av hermiteinterpolanten. Årsaken til å ha dette siste elementet er å enten integrere fra 0 til ≤ 0.5 eller fra > 0.5 til 1, for å redusere feilen. Formlene for interpoleringen "mellom 0 og 1" vil nå kort bli diskutert (vi skal etterpå se på skalering av deriverte på grunn av domeneskalering). Vi starter med en generell 3.-grads polynomformulering,

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3, \tag{A.7}$$

og dens 1.-deriverte,

$$f'(x) = a_1 + 2a_2 x + 3a_3 x^2. \tag{A.8}$$

Å integrere(A.7) fra 0 til en gitt verdi $\widehat{t} \in (0,1)$ gir følgende resultat

$$\int_{x=0}^{\widehat{t}} f(x) = \widehat{t}\left(a_0 + \widehat{t}\left(\frac{a_1}{2} + \widehat{t}\left(\frac{a_2}{3} + \widehat{t}\left(\frac{a_3}{4}\right)\right)\right)\right), \tag{A.9}$$

og å evaluere (A.7) i $\widehat{t}$, samt å nøste uttrykket gir

$$f(\widehat{t}) = a_0 + \widehat{t}\left(a_1 + \widehat{t}\left(a_2 + \widehat{t}\,(a_3)\right)\right). \tag{A.10}$$

Å evaluere og å nøste (A.8) gir

$$f'(\widehat{t}) = a_1 + \widehat{t}\left(2a_2 + \widehat{t}\,(3a_3)\right). \tag{A.11}$$

Hvis vi antar $f(0)$, $f'(0)$, $f(1)$ og $f'(1)$ som kjent, kan vi løse systemet med hensyn på $\{a_i\}_{i=0}^{3}$. Hvis vi i tillegg også inkluderer a_4 (integralet over hele intervallet, det vil si $\int_0^1 f(x)dx$), blir, for sampelintervall j, hele j-te rad i **a** matrisen):

$$\begin{aligned} a_0 &= f(0), \\ a_1 &= f'(0), \\ a_2 &= 3\left(f(1) - f(0)\right) - f'(1) - 2f'(0), \\ a_3 &= -2\left(f(1) - f(0)\right) + f'(1) + f'(0), \\ a_4 &= \tfrac{f(0)+f(1)}{2} + \tfrac{f'(0)-f'(1)}{12}. \end{aligned} \tag{A.12}$$

Det neste er å bruke de fire uttrykkene (A.9), (A.10), (A.11) og (A.12) i evaluatoren. Husk at skalering av domenet påvirker deriverte og antideriverte (integral). Anta at det reelle intervallet er Δt mens uttrykket ovenfor er laget over domenet 1. På grunn av regelen om skalering av domenet må de deriverte/antideriverte skaleres, se [49]). Dermed må vi følgende tre justeringer gjøres:

- Innputt: - deriverte $f'(0)$ og $f'(1)$ må begge skaleres med Δt,
- Utgang: - integralet $B(t)$, må skaleres med Δt.

- Utgang: - deriverte (som er 2.-deriverte $D^2B(t)$) må omvendt skaleres med Δt.

For å fullføre beskrivelsen av "ERB-evaluatoren" er det fortsatt tre algoritmer som ikke er beskrives. Den første algoritmen er initialisering.

Algoritme 15. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen beregner de intern tilstandsvariablene* Δt, $S = \mathbb{S}(\alpha,\beta,\gamma,\lambda)$, $\mathbf{b} = \{B(t_i)\}_{i=0}^{m}$ *og matrisen* $\mathbf{a}$ *(ved hjelp av en interpolasjonsfunksjon). Algoritmen forutsetter at det finnes algoritmer for å beregne* $\phi(t)$ *og* $\phi'(t)$ *(kan bruke* $f_2(t)$ *hvis* $\mathbb{S} = 1$*) og* $\int_{s=0}^{t}\phi(s)ds$, $0 \leq t \leq 1$. *Innputt er: De iboende parameterne* γ, μ, α, β, *og m (antall sampelintervaller). Det er ingen returverdier her.*

void **initiate** (double γ, double μ, double α, double β, int m)

double ϕ_0, ϕ_1;	// Verdi i starten og slutten av hvert intervall
double f_0, f_1;	// start og slutt-deriverte i hvert intervalll
set($\gamma,\mu,\alpha,\beta,m$);	// lagre iboende parametere og m i objektet
Tildel minne for **b**, **a**;	// Set størrelse = m+1 *for* **b**, *og* $m \times 4$ *for* **a**
$\Delta t = \frac{1}{m}$;	// Setter intervallstørrelse
$S = 1$;	// midlertidig bruke $\phi'(t) = f_2(t)$
$\mathbf{b}_0 = \phi_0 = f_0 = 0.0$;	// Definert til null (grunnleggende egenskap)
for (int i=1; $i < m$; i++)	// For hvert sampelintervall
double $t = i * \Delta t$;	// Sampelvektoren t_i i definisjon
$\phi_1 = \phi(t)$;	// Algoritme 11
$f_1 = \phi_1\ f_2(t)$;	// Algoritme 13
$\mathbf{b}_i = integrate(t-\Delta t, t, \phi_0, \phi_1, 10^{-16})$;	// Algorithm 14 (bruker toleranse 10^{-16})
interpolate($i-1, \phi_0, \phi_1, \Delta t f_0, \Delta t f_1$);	// Lager koeffisientene $\{\mathbf{a}_{i-1,k}\}_{k=0}^{3}$
$\phi_0 = \phi_1$;	// Klargjøre for neste trinn
$f_0 = f_1$;	
$\mathbf{b}_m = integrate(\Delta t(m-1), 1, \phi_0, 0, 10^{-16})$;	// Algoritme 14 (bruker toleranse 10^{-16})
interpolate($m-1, \phi_0, 0, \Delta t f_0, 0$);	// Lager koeffisientene $\{\mathbf{a}_{m-1,k}\}_{k=0}^{3}$
for (int i=1; $i \leq m$; $i \ll$=1)	// Disse tre følgende linjene introduseres
for (int j=m; $j \geq i$; $j-=1$)	// fordi det i linje 13/17 ikke brukes $+=$
$b_j += b_{j-i}$;	// Se under for forklaring
$S = \frac{1}{b_m}$;	

I linje 13 og 17 kunne **b** += *integrate*(...), inkrementell addisjon, vært brukt. Men det ville ha medført tap av minst én signifikant bit på grunn av at vi da legge et lite tall til et større (og økende) tall. For å unngå dette summerer de neste tre siste linjene på en binær måte. Algoritmen summerer naboelementer som er nesten like. Algoritmen er derfor konstruert for å være så presis som mulig. I algoritmen kan man se at både "integrate()" og "interpolate()" kalles to ganger. Dette gjøres fordi vi vil unngå å kalle $\phi(t)$ og $f_2(t)$ for $t = 0$ og $t = 1$, fordi de der er definert til å være 0. Vi må så i tillegg huske skaleringen av de deriverte: de er alle, i henhold til skaleringsreglene, skalert med Δt i innputt til interpolate()-funksjonen i linje 14 og 18.

Den neste algoritmen, interpoleringen, er en veldig enkel algoritme som beregner alle hermiteinterpolasjonskoeffisientene (se (A.12)).

Algoritme 16. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen beregner koeffisientene fra hermiteinterpoleringen* $\mathbf{a}_{i,0}$, $\mathbf{a}_{i,1}$, $\mathbf{a}_{i,2}$, $\mathbf{a}_{i,3}$ *og* $\mathbf{a}_{i,4}$, *og er en implementering av (A.12).*

void **interpolate (** int *i,* double f_0, double f_1, double f_0', double f_1' **)**

$\mathbf{a}_{i,0} = f_0$;
$\mathbf{a}_{i,1} = f_0'$;
$\mathbf{a}_{i,2} = 3(f_1 - f_0) - f_1' - 2f_0'$;
$\mathbf{a}_{i,3} = -2(f_1 - f_0) + f_1' + f_0'$;
$\mathbf{a}_{i,4} = \frac{f(0)+f(1)}{2} + \frac{f'(0)-f'(1)}{12}$;

Den tredje og siste funksjonen er hovedfunksjonen for evalueringen av $B()$, og som tilsvarer algoritmen 14. Vær oppmerksom på at $B()$ er en B-funksjon som ikke er koblet direkte til en kurve eller en flate men kun til en ERB-evaluator. Hvis det er en kurve eller flate som kallet en ERB-evaluator, må tilordningen $B \circ w_{1,i}(t)$ gjøres først, se (8.4) og (6.11). Og etter retur fra funksjonskallet må resultatet skaleres med $\delta_{1,i}^{j}$, der j er ordenen til de deriverte, se (6.13). Husk at indeksen i i skjøtvektoren er bestemt av $t_i \leq t < t_{i+1}$.

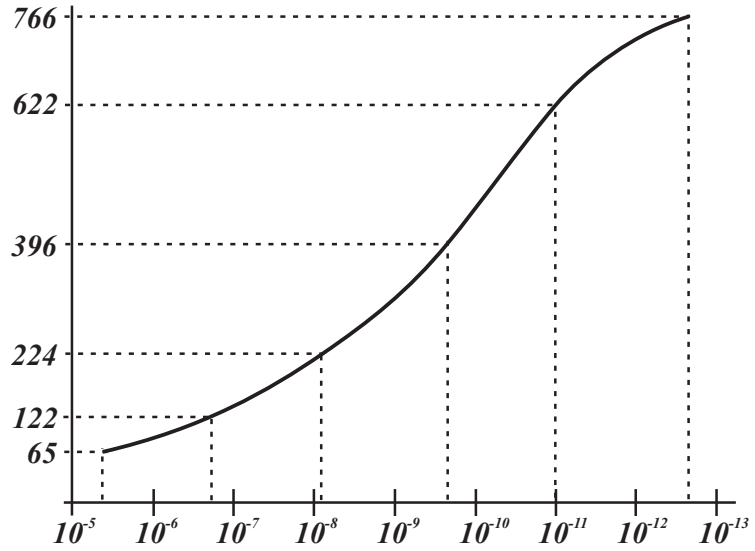
Algoritme 17. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen beregner* $D^j B(t), j = 0,1,2$ *med settet av iboende parametere som er brukt ved initialiseringen av objektet. Algoritmen forutsetter da at objektet er initialisert. Innputt er:* $t \in [0,1]$, *og* $d \in \{0,1,2\}$ *som er antall deriverte som skal beregnes. Returen er en* "vector⟨double⟩", *der det første elementet inneholder* $B(t)$, *og deretter følger ,avhengig av d,* $B'(t)$ *og* $B''(t)$.

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

vector⟨double⟩ $R(d+1) = S$; *// Lager retur vektor, størrelse d+1, alle settes til S*
int j = min(int$(t * m), m-1)$; *// j ikke lik m, på grunn av speiling rundt 0.5*
double $dt = \frac{t - j*\Delta t}{\Delta t}$; *// Mapper fra hele* $[0,1]$ *til* $[0,1]$ *på sampelintervall*
switch *(d)*
case *2:* $R_2 * = \frac{\mathbf{a}_{j,1} + dt\left(2\mathbf{a}_{j,2} + dt\ 3\mathbf{a}_{j,3}\right)}{\Delta t}$; *// Er (A.11), skalert med* $\frac{1}{\Delta t}$
case *1:* $R_1 * = \mathbf{a}_{j,0} + dt\left(\mathbf{a}_{j,1} + dt\left(\mathbf{a}_{j,2} + dt\ \mathbf{a}_{j,3}\right)\right)$; *// Er (A.10), uten skalering*
if $(dt > 0.5)$ *// Integrere: dt - 1 (A.12)*
$R_0 * = b_{j+1} - \Delta t\left(\mathbf{a}_{j,4} - dt\left(\mathbf{a}_{j,0} + dt\left(\frac{\mathbf{a}_{j,1}}{2} + dt\left(\frac{\mathbf{a}_{j,2}}{3} + dt\frac{\mathbf{a}_{j,3}}{4}\right)\right)\right)\right)$ *// Skalert - Δt*
else *// Integrere: 0 - dt (A.12)*
$R_0 * = b_j + \Delta t\ dt\left(\mathbf{a}_{j,0} + dt\left(\frac{\mathbf{a}_{j,1}}{2} + dt\left(\frac{\mathbf{a}_{j,2}}{3} + dt\frac{\mathbf{a}_{j,3}}{4}\right)\right)\right)$; *// Skalert - Δt*
return *R;*

Det neste spørsmålet er "evalueringen" av ERB-evaluatoren selv. Det er to forskjellige vurderinger som må gjøres: evaluering av effektiviteten og av presisjonen.

Effektiviteten er helt klart grunnen til å lage hele systemet. På grunn av hermiteinterpoleringen og bruk av numerisk integrasjon i initieringa betyr bruke "ERB-evaluatoren" tidsmessig det samme som å beregne en 4.-grads polynomfunksjon, inkludert deriverte. Dette er tidsmessig på en måte en optimal løsning. I figur A.2 vises en graf av forholdet mellom bruken av algoritme 12 (med to deriverte) og "ERB-evaluatoren" med algoritme

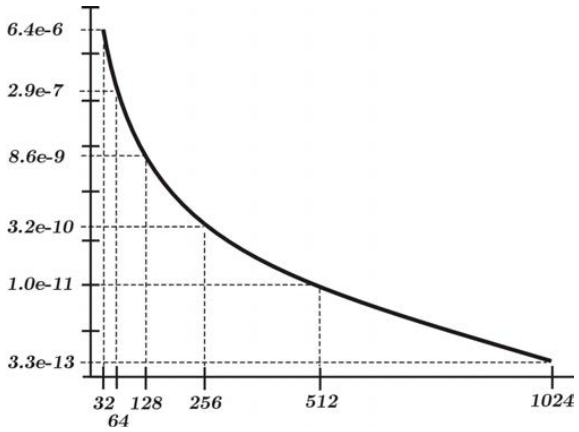


Figur A.2: *Figuren viser Forholdet mellom presisjonen (dvs. toleranse) og beregningshastighet til $B()$ i "ERB-evaluatoren". Den vertikale skalaen viser tidsbruken relativt mot beregning av $B()$-funksjonen i algoritme 12 fra seksjon A.2. Med en sampling på 1024, som gir en presisjon på $3,3\ 10^{-13}$, er "ERB-evaluatoren" 766 ganger raskere enn evaluatoren i Algoritme 12, ved beregning av funksjonsverdi og to deriverte.*

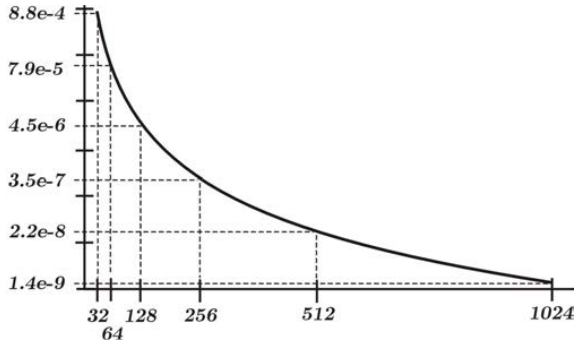
17,. Algoritmen 12 bruker standardsettet med iboende parametere, og er med det relativ rask. Hastigheten til $B()$ i "ERB-evaluatoren" er i seg selv uavhengig av samplingsfrekvensen, men presisjonen er imidlertid svært avhengig av samplingsfrekvensen. Funksjonen $B()$ i Algoritmen 12 er svært avhengig av presisjonen, spesielt integreringsdelen dvs. rombergintegrasjon. Dette kan tydeligvis sees i figur A.2. På den horisontale aksen vises presisjonen, dvs. det største avviket til den virkelige funksjonsverdien $B(t)$, $t \in [0,1]$. På den vertikale aksen kan du se forholdet mellom hastigheten til $B()$ i "ERB-evaluator" og $B()$ fra algorime 12. Man kan tydelig se at forskjellen i hastighet er enorm. Vi ser er at det kan ta opptil 766 *ganger* lengre tid å bruke $B()$ i algoritmen 12 enn $B()$ i "ERB-evaluatoren". Figuren indikerer også at når vi passerer toleransen 10^{-13} nærmer vi oss optimal hastighetsforbedring ved å bruke algoritme 17.

Presisjon er et mer komplekst problem å håndtere, spesielt siden presisjonen blir dårligere med økt orden på de deriverte. Årsaken er egentlig lett å se, fordi $B(t)$ er integralet til $B'(t)$, og videre fordi $B'(t)$ er integralet til $B''(t)$. Vær oppmerksom på at en forenklet beregning av et maksimalt avvik i et integral av en funksjon over et sampelintervall er omtrent: $\frac{2}{3} \times$ det største avviket til en funksjon $\times \frac{1}{m}$ (sampelintervallet). Forskjellen i avvik mellom en approksimasjon av en funksjon og en approksimasjon av deriverte til funksjonen er derfor nær 10^3 for en samplerate på $m = 1024$. Det er selvfølgelig metoder for å forbedre dette, men de kan redusere hastigheten eller fleksibiliteten (dette vil bli diskutert videre senere).

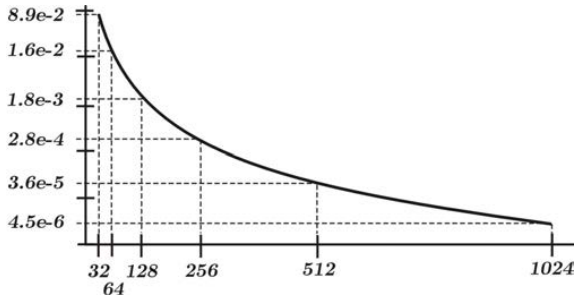
Vi får et innblikk i presisjonen/avviket til "ERB-evaluatoren" i tabell A.2. Tabellen viser sammenhengen mellom antall sampelintervall og feilen i ERB-funksjonen $B(t)$, samt 1.- og 2.-deriverte $B'(t)$ og $B''(t)$. Feilen vises i tre forskjellige normer. Disse normene er $L^1[0,1]$ som representerer en aritmetisk middelverdi, $L^2[0,1]$ som representerer en



Figur A.3: *For en "ERB-evaluator" - ser vi et plot av forholdet mellom antall sampelintervall og presisjonen (maks-norm) for $B(t)$, funksjonsverdien. Med 32 sampelintervall er maks avvik $6,4\ 10^{-6}$, og for 1024 sampelintervall er maks avvik $3,3\ 10^{-13}$.*



Figur A.4: *For en "ERB-evaluator" - ser vi et plot av forholdet mellom antall sampelintervall og presisjonen (maks-norm) for $B'(t)$, 1.-deriverte. Med 32 sampelintervall er maks avvik $8,8\ 10^{-4}$, og for 1024 sampelintervall er maks avvik $1,4\ 10^{-9}$.*



Figur A.5: *For en "ERB-evaluator" - ser vi et plot av forholdet mellom antall sampelintervall og presisjonen (maks-norm) for $B''(t)$, 2.-deriverte. Med 32 sampelintervall er maks avvik $8,9\ 10^{-2}$, og for 1024 sampelintervall er maks avvik $4,5\ 10^{-6}$.*

intervall		1024	512	256	128	64	32
$B(t)$	$L^1[0,1]$	$7{,}6\ 10^{-15}$	$2{,}4\ 10^{-13}$	$7{,}6\ 10^{-12}$	$7{,}7\ 10^{-10}$	$8{,}5\ 10^{-9}$	$2{,}7\ 10^{-7}$
	$L^2[0,1]$	$2{,}9\ 10^{-14}$	$9{,}5\ 10^{-13}$	$3{,}0\ 10^{-11}$	$9{,}6\ 10^{-10}$	$3{,}3\ 10^{-8}$	$9{,}7\ 10^{-7}$
	$L^\infty[0,1]$	$3{,}3\ 10^{-13}$	$1{,}0\ 10^{-11}$	$3{,}2\ 10^{-10}$	$8{,}6\ 10^{-9}$	$2{,}9\ 10^{-7}$	$6{,}4\ 10^{-6}$
$B'(t)$	$L^1[0,1]$	$4{,}9\ 10^{-11}$	$7{,}8\ 10^{-10}$	$1{,}3\ 10^{-8}$	$2{,}0\ 10^{-7}$	$3{,}5\ 10^{-6}$	$5{,}6\ 10^{-5}$
	$L^2[0,1]$	$1{,}7\ 10^{-10}$	$2{,}7\ 10^{-9}$	$4{,}3\ 10^{-8}$	$6{,}7\ 10^{-7}$	$1{,}2\ 10^{-5}$	$1{,}7\ 10^{-4}$
	$L^\infty[0,1]$	$1{,}4\ 10^{-9}$	$2{,}2\ 10^{-8}$	$3{,}5\ 10^{-7}$	$4{,}5\ 10^{-6}$	$7{,}9\ 10^{-5}$	$8{,}8\ 10^{-4}$
$B''(t)$	$L^1[0,1]$	$1{,}9\ 10^{-7}$	$1{,}5\ 10^{-6}$	$1{,}2\ 10^{-5}$	$9{,}7\ 10^{-5}$	$8{,}4\ 10^{-4}$	$6{,}5\ 10^{-3}$
	$L^2[0,1]$	$5{,}9\ 10^{-7}$	$4{,}7\ 10^{-6}$	$3{,}8\ 10^{-5}$	$2{,}9\ 10^{-4}$	$2{,}6\ 10^{-3}$	$1{,}9\ 10^{-2}$
	$L^\infty[0,1]$	$4{,}5\ 10^{-6}$	$3{,}6\ 10^{-5}$	$2{,}8\ 10^{-4}$	$1{,}8\ 10^{-3}$	$1{,}6\ 10^{-2}$	$8{,}9\ 10^{-2}$

Tabell A.2: *Tabellen viser sammenhengen mellom antall sampelintervall og feilen til de tre funksjonene $B(t)$, $B'(t)$ og $B''(t)$. Det er brukt tre forsjkellige normer, og for hver av de tre funksjonene er det tre rader, en for hver norm - $L^1[0,1]$, $L^2[0,1]$ og $L^\infty[0,1]$.*

geometrisk middelverdi, og $L^\infty[0,1]$, en maks-norm, som gir oss den garanterte toleransen. De tre figurene A.3, A.4 og A.5, viser grafer over avviket, dvs. maks-normen, til "ERB-evaluatoren" i forhold til antall sampelintervall. Figur A.3 viser feilen til funksjonsverdien $B(t)$. Den varierer fra $6{,}4\ 10^{-6}$ for 32 intervall, til $3{,}3\ 10^{-13}$ for 1024 intervall. Dette kan betraktes som et ganske godt resultat. I figur A.4 er feilen for den 1.-deriverte $B'(t)$ plottet. Den varierer fra $8{,}8\ 10^{-4}$ for 32 intervall, til $1{,}4\ 10^{-9}$ for 1024 intervall. I mange sammenhenger kan dette fortsatt betraktes som et akseptabelt resultat. I figur A.5 vises feilene for den 2.-deriverte $B''(t)$. Den varierer fra $8{,}9\ 10^{-2}$ for 32 intervall, til $4{,}5\ 10^{-6}$ for 1024 intervall. Dette resultatet er egentlig på kanten av det som er akseptabelt, men kan være greit i en del tilfeller.

Konklusjonen er dermed at for verdien, $B(t)$, og for den 1.-deriverte, $B'(t)$, er feilene akseptable. Mens feilen for den 2.-deriverte , $B''(t)$, er på kanten. Hvis vi har behov for en mer presis 2.-derivert og muligens også deriverte av høyere orden kan vi gjøre nen tilpasninger. Dette kan gjøres både for standardsettet med iboende parametere og også for et generelt sett med iboende parametere.

For standardsettet kan vi gjøre noen endringer av algoritme 17. I algorime 18 som kommer under er forbredelsene til dette gjordt. I siste delen av algoritmen er det klargjordt for opptil seks deriverte. Dette med utganspunkt i formel (7.50) og formler i [102]. I algoritme 18 er utgangspunktet den 2.-deriverte fra algoritme 17. Dette kan vi forbedre med å erstatte det røde feltet i algoritmen med et kall til den andre funksjon i algoritme 11, dvs. $\phi = \phi(t)$. Dette kan vi gjøre med en relativ liten merkostnad, se forøvrig merknadene i algoritme 18.

Når det gjelder $B(t)$ med vilkårilge iboende parametre er dette mere komplisert fordi $g(t)$ er mere komplisert. En nærmere studie av dette finnes i [102].

I C++ -programmering vil det være naturlig å lage en klasse som arver fra ERB-evaluator-klassen som er beskrevet på side 291. Det meste kan gjenbrukes, men en ny $B(t,d)$ kan

lages. Algoritmen for evaluering er **B** (*double* t, *int* d) for en ERB-funksjon med standardsettet med iboende parametere er:

Algoritme 18. *(Forklaring av notasjonen, se avsnitt "Algoritmisk språk", side 6.) Algoritmen beregner $D^j B(t), j = 0,1,\ldots,d$ for d opp til 6. Men dette gjelder bare med default sett av de iboende parameterene. Algoritmen forutsetter også at objektet er initialisert. Innputt er: $t \in [0,1]$, og $d \in \{0,1,2,3,4,5,6\}$ som er antall deriverte som skal beregnes. Returen er* "vector⟨double⟩", *der det første elementet inneholder $B(t)$, og deretter følger ,avhengig av d, $B'(t)$, $B''(t)$, ..., $B^{(d)}(t)$*

vector⟨double⟩ **B** (double *t*, int *d*)
  vector⟨double⟩ $R(d+1) = S$; // *Lager returvektor, størrelse d +1, alle settes til S*
  int j = min(int$(t*m), m-1)$; // *j ikke lik m, på grunn av speiling rundt 0.5*
  double $dt = \frac{t-j*\Delta t}{\Delta t}$; // *Mapper fra hele $[0,1]$ til $[0,1]$ på sampelintervall*
  if $(dt > 0.5)$ // *Integrere: dt - 1 (A.12)*
    $R_0 *= b_{j+1} - \Delta t \left(\mathbf{a}_{j,4} - dt\left(\mathbf{a}_{j,0} + dt\left(\frac{\mathbf{a}_{j,1}}{2} + dt\left(\frac{\mathbf{a}_{j,2}}{3} + dt\frac{\mathbf{a}_{j,3}}{4}\right)\right)\right)\right)$; // *Skalere Δt*
  else // *Integrere: 1 - dt (A.12)*
    $R_0 *= b_j + \Delta t\ dt\left(\mathbf{a}_{j,0} + dt\left(\frac{\mathbf{a}_{j,1}}{2} + dt\left(\frac{\mathbf{a}_{j,2}}{3} + dt\frac{\mathbf{a}_{j,3}}{4}\right)\right)\right)$; // *Skalere Δt*
  if $(d > 0)$
    double $\phi = \mathbf{a}_{j,0} + dt\left(\mathbf{a}_{j,1} + dt\left(\mathbf{a}_{j,2} + dt\ \mathbf{a}_{j,3}\right)\right)$; // *se*[4]
    $R_1 *= \phi$; // *(A.10) - ingen skalering*
    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]Det er også mulig å kalle algoritmen 11, del 2 direkte, dvs. double $\phi = \phi(t)$; Feilen vil bli betydelig forbedret, til bedre enn 10^{-15}. Men kostnaden, i form av tidsbruk, for bare denne linjen er omtrent 9:22. Det er omtrent 2,5 ganger så mye tid. Men vær oppmerksom på at dette bare gjelder denne ene linjen. For hele funksjonen betyr det en økning i tidsbruk fra ca. 1,4 til 1,1 avhengig av antall deriverte som skal beregnes, størst for bare 2 deriverte.

Vedlegg B

Programmeringsbibliotek

Å løse lineære systemer, som noen ganger er ganske store, er en ressurskrevende jobb for datamaskinen. Derfor er det lurt å bruke ferdige optimaliserte subrutiner til dette. Det er et definert programmeringsgrensesnitt for dette, BLAS, som er beskrevet under. Vedlegget har også en liste over komplette rutiner som er BLAS-kompatible.

Et annet problem knyttet til ressurskrevende program er å kunne bruke alle tilgjengelige ressurser på datamaskinen. Dette kalles “Heterogen databehandling"og sammen med parallellisering er dette behandlet i avsnittet etter BLAS.

B.1 BLAS, grunnleggende subrutiner for lineær algebra

Grunnleggende subrutiner for lineær algebra - BLAS (fra engelsk “Basic Linear Algebra Subprograms") er de facto API (grensesnittstandard for applikasjonsprogrammering) for programmeringsbibliotek til grunnleggende lineæralgebraoperasjoner som vektor- og matrisemultiplikasjon. Det ble lansert i 1979 og brukt til å bygge større pakker som LAPACK og Armadillo[1]. Kommersielle applikasjoner som MATLAB bruker også BLAS. BLAS er mye brukt i høyytelses-databehandling, svært optimaliserte implementeringer av BLAS-grensesnittet er utviklet av maskinvareleverandører som Intel, AMD og NVIDIA for CPU og GPU, men også av andre, se nedenfor. BLAS sin hjemmeside finner du på http://www.netlib.org/blas/.

BLAS-funksjonalitet er kategorisert i tre “nivå", som tilsvarer både den kronologiske rekkefølgen av definisjon og publisering, samt graden av kompleksiteten til algoritmer, $\mathcal{O}(n^j),\ j = 1,2,3$:

Nivå 1 BLAS-operasjoner av $\mathcal{O}(n)$. Dette nivået består av alle rutinene beskrevet i den opprinnelige presentasjonen av BLAS (1979), [112] som kun definerte vektoroperasjoner på stridede arrays: prikkprodukter, vektornormer, et generalisert vektortillegg på formen $y \leftarrow \alpha x + y$.

[1]LAPACK, et programvarebibliotek skrevet i Fortran http://performance.netlib.org/lapack/ og LAPACK++ i C++ https://math.nist.gov/lapack++/. Armadillo er et annet C++ -bibliotek http://arma.sourceforge.net/. Alle tre er gratis programvare

Nivå 2 BLAS-operasjoner av $\mathcal{O}(n^2)$. Dette nivået inneholder matrise-vektor operasjoner inkludert, blant annet, en generalisert matrise-vektor multiplikasjon $y \leftarrow \alpha Ax + \beta y$ og å løse for x, $Tx = y$ med T som triangulær. Design av dette nivået ble laget i 1984 - 1988, [53].

Nivå 3 BLAS-operasjoner av $\mathcal{O}(n^3)$, formelt publisert i 1990, [52]. Inneholder matrise-matrise-operasjoner, inkludert en generell matrisemultiplikasjon, av formen $C \leftarrow \alpha AB + \beta C$, hvor A og B valgfritt kan transponeres eller hermitekonjugeres inne i rutinen, og alle tre matriser kan ha vilkårlig steglengde (engelsk - strides). Den ordinære matrisemultiplikasjonen AB kan utføres ved å sette α til 1, og C til en null matrise av passende størrelse. Vi finner også rutiner for $B \leftarrow \alpha T^{-1}B$ der T er en trekant-matrise. Det finnes også andre funksjoner.

Moderne BLAS-implementeringer støtter vanligvis alle tre nivåene.

Nedenfor følger en kort (ikke komplett) liste over bibliotek med BLAS-grensesnitt:

Accelerate - Apple's rammeverk for macOS og iOS,
https://developer.apple.com/accelerate/

AOCL - er et sett med numeriske bibliotek spesifikt for AMD CPU-er
https://developer.amd.com/spack/amd-optimized-cpu-libraries/

ATLAS - Åpen kildekode implementasjon - APIer for C og Fortran77,
http://math-atlas.sourceforge.net/

BLIS - for AMD, et BLAS-lignende "komplett" lineæralgebrabibliotek,
https://developer.amd.com/amd-aocl/blas-library/

cuBLAS - Grunnleggende lineæralgebra på NVIDIA GPUer,
https://developer.nvidia.com/cublas

NVBLAS - For NVIDIA GPUer, men bare nivå 3-funksjoner,
https://docs.nvidia.com/cuda/nvblas/index.html

clBLAST - er et OpenCL BLAS-bibliotek skrevet i C++ 11,
https://rocmdocs.amd.com/en/latest/ROCm_Tools/clBLA.html

Eigen BLAS - er et C++ templatebibliotek (fri programvare)
http://eigen.tuxfamily.org/index.php?title=Main_Page

GSL - GNUs vitenskapelige bibliotek for C og C++ (fri programvare),
http://www.gnu.org/software/gsl/

Intel MKL - "oneAPI Math Kernel Library" for Intel CPU-er
http://software.intel.com/en-us/intel-mkl

Netlib BLAS - fritt tilgjengelig i Fortran, for C se CBLAS,
https://www.netlib.org/blas/

OpenBLAS - er basert på GotoBLAS2 1.13 BSD. OpenBLAS er et åpent kildekode-prosjekt støttet av "Lab of Parallel Software and Computational Science",
http://www.openblas.net/

rocBLAS - En implementering på toppen av AMDs Radeon Open Compute ROCm kjøretid og verktøykjeder. rocBLAS er implementert i programmeringsspråket HIP og optimert for AMDs siste diskrete GPUer.
https://rocblas.readthedocs.io/

SurviveGotoBLAS2 - er også basert på GotoBLAS2 1.13 BSD, lisensiert under AGPL-3. http://prs.ism.ac.jp/~nakama/SurviveGotoBLAS2/

uBLAS - et C++ -template-klassebibliotek som gir nivå 1, 2, 3 funksjonalitet for tette, pakkede og tynne matriser.
https://www.boost.org/doc/libs/1_72_0/libs/numeric/ublas

ViennaCL - et gratis lineæralgebrabibliotek med åpen kildekode for beregninger på GPUer og flerkjernede CPUer. Biblioteket er skrevet i C++ og støtter CUDA, OpenCL og OpenMP. Den inneholder kjernefunksjonalitet og mange andre funksjoner, inkludert støtte for BLAS nivå 1-3 og også iterative løsere.
http://viennacl.sourceforge.net/

Det er mange andre implementeringer å finne på internett ved søk. Dessuten er det mange større programbibliotek å finne. Tidligere er LAPACK++ og Armadillo nevnt. Et spesielt bibliotek som kan anbefales er Blaze som er et høy ytelse C++ template-basert bibliotek som kan lastes ned fra https://bitbucket.org/blaze-lib/.

B.2 Heterogen databeregninger og parallellisering

I heterogen databehandling bruker vi systemer som bruker mer enn èn type prosessor eller kjerner. Dette kan typisk være å kombinere den/de sentrale prosesseringsenhetene (CPU) med grafikkprosessorenheter (GPU). Sammen med parallellisering, dvs. bruk av mange like enheter/kjerner, er dette en viktig måte å øke beregningshastigheten på, spesielt når store matriser er involvert.

CUDA - "Compute Unified Device Architecture", er en parallell dataplattform og API-modell opprettet av Nvidia. Vi kan bruke en CUDA-aktivert GPU til generell databehandling, dvs. CUDA-plattformen er et programvarelag som gir direkte tilgang til GPU-ens virtuelle instruksjonssett og parallelle beregningselementer, for utførelse av databehandlingskjerner (shadere). Den kan brukes sammen med C, C++ og Fortran. CUDA-drevne GPU-er støtter også programmeringsrammeverk som OpenMP, OpenACC og OpenCL, og også HIP ved å kompilere slik kode til CUDA.
https://developer.nvidia.com/about-cuda

HIP - Det HPC-klare universelle språket i kjernen av AMDs helt åpne ROCm-plattform. HIP kan kjøre på både AMD- og Nvidia GPU-er. HIP API-syntaksen ligner veldig på CUDA API, og abstraksjonsnivået er det samme, som betyr at "porting" mellom de to er enkelt. https://rocmdocs.amd.com/

OpenMP - OpenMP API støtter parallellprogrammering med delt minne på flere plattformer i C/C++ og Fortran. Den består av et sett med kompilatordirektiver, bibliotekrutiner og miljøvariabler som påvirker kjøretiden. Kjerneelementene i OpenMP

er konstruksjonene for trådoppretting, arbeidsbelastningsdistribusjon (arbeidsdeling), datamiljøadministrasjon, trådsynkronisering, kjøretidsrutiner på brukernivå og miljøvariabler. https://www.openmp.org/

OpenCL - Den åpne standarden for parallellprogrammering av heterogene systemer. Det er et rammeverk for å skrive programmer som kjører på tvers av heterogene plattformer som består av sentrale prosesseringsenheter - CPUer, grafikkbehandlingsenheter - GPU-er, digitale signalprosessorer -DSPer, feltprogrammerbare portmatriser - FPGAer og andre prosessorer eller maskinvareakseleratorer. OpenCL angir programmeringsspråk (basert på C99, C++14 og C++17) for programmering av disse enhetene og apiene for programmering av programmer for å kontrollere plattformen og kjøre programmer på de forskjellige enhetene. OpenCL gir et standardgrensesnitt for parallellisering ved hjelp av oppgave- og databasert parallellitet. http://www.khronos.org/opencl/

Vedlegg C

Diverse bevis

C.1 Newton- og lagrangepolynomene, bevis lemma 5.1

Lemma 5.1 sier at newton- og lagrangepolynomene bare er to forskjellige representasjoner av det samme polynomet. Her følger beviset.

Bevis. Vi bruker induksjon. Vi starter med å vise at det stemmer for et 1.-grads polynom

$$f(t) = \sum_{i=0}^{1} a_i n_i(t) = f(t_0) + (t - t_0)\left(\frac{f(t_0)}{(t_0 - t_1)} + \frac{f(t_1)}{(t_1 - t_0)}\right) = f(t_0)\frac{(t_0 - t_1) + (t - t_0)}{t_0 - t_1}$$
$$+ \frac{t - t_0}{t_1 - t_0} f(t_1) = \frac{t - t_1}{t_0 - t_1} f(t_0) + \frac{t - t_0}{t_1 - t_0} f(t_1) = \sum_{i=0}^{1} f(x_i)\, L_{1,i}(t),$$

følgelig er 1.-grads newton- og lagrangepolynomene det samme polynomet. Så hvis vi har et lagrangepolynom av grad $d-1$ og deretter legger til ett interpolasjonspunkt som et

newtonpolynomledd ved å bruke (5.7) får vi,

$$
\begin{aligned}
f(t) &= \sum_{i=0}^{d-1} f(t_i)\, L_{d-1,i}(t) + f[t_0,\ldots,t_d] \prod_{i=0}^{d-1}(t-t_i) \\
&= \sum_{i=0}^{d-1} \left(f(t_i) \prod_{j=0,j\neq i}^{d-1} \frac{(t-t_j)}{(t_i-t_j)} \right) + \sum_{i=0}^{d} \frac{f(t_i)}{\omega_d'(t_i)} \prod_{j=0}^{d-1}(t-t_j) \\
&= \sum_{i=0}^{d-1} \left(f(t_i) \frac{\displaystyle\prod_{j=0,j\neq i}^{d-1}(t-t_j)(t_i-t_d)}{\displaystyle\prod_{j=0,j\neq i}^{d-1}(t_i-t_j)(t_i-t_d)} \right) + \sum_{i=0}^{d} \left(f(t_i) \frac{\displaystyle\prod_{j=0}^{d-1}(t-t_j)}{\displaystyle\prod_{j=0,j\neq i}^{d}(t_i-t_j)} \right) \\
&= \sum_{i=0}^{d-1} \left(f(t_i) \frac{\displaystyle\prod_{j=0,j\neq i}^{d-1}(t-t_j)(t_i-t_d) + \prod_{j=0,j\neq i}^{d-1}(t-t_j)(t-t_i)}{\displaystyle\prod_{j=0,j\neq i}^{d}(t_i-t_j)} \right) + f(t_d) \prod_{j=0}^{d-1} \frac{(t-t_j)}{(t_d-t_j)} \\
&= \sum_{i=0}^{d-1} \left(f(t_i) \frac{\displaystyle\prod_{j=0,j\neq i}^{d-1}(t-t_j)(t_i-t_d+t-t_i)}{\displaystyle\prod_{j=0,j\neq i}^{d}(t_i-t_j)} \right) + f(t_d) \prod_{j=0}^{d-1} \frac{(t-t_j)}{(t_d-t_j)} = \sum_{i=0}^{d} f(x_i)\, L_{d,i}(t).
\end{aligned}
$$

Vi ser at dette gir et lagrangepolynom av grad d, som dermed fullfører beviset. □

C.2 Kommutattivitetsrelasjoner mellom $T_d(t)$ og dens deriverte

$T_d(t)$-matrisene er matriser som bare inneholder lineære element eller null. Det betyr at T_d', som den deriverte av $T_d(t)$, er en matrise av bare konstanter.

Gitt at vi har en B-spline av grad 3. Vi vil da se at uansett hvilken av de 3 matrisene vi deriverer så blir resultatet det samme, dvs.

$$T_1(t)\; T_2(t)\; T_3' = T_1(t)\; T_2'\; T_3(t) = T_1'\; T_2(t)\; T_3(t),$$

og vi skal også se at det samme gjelder uansett grad.

Lemma C.1. *Derivasjon av en multiplikasjon mellom to $T(t)$-matriser er kommutativ. Det vil si for alle $d > 0$ og for en gitt skjøtsekvens $t_{j-d}, t_{j-d+1}, \ldots, t_{j+d+1}$ og $t_i \leq t < t_{i+1}$ er*

$$T_d(t)\; T_{d-1}' = T_d'\; T_{d-1}(t). \tag{C.1}$$

Bevis. Vi starter med å beregne venstre side av C.1. I hver linje i matrisene $T_d(t)$ er bare to påfølgende elementer forskjellige fra null. Vi multipliserer disse to elementene med undermatrisen til T_{d+1}' som er den delen av matrisen som gir noe annet enn null, og får

$$\begin{pmatrix} 1-w_{d,j}(t) & w_{d,j}(t) \end{pmatrix} \begin{pmatrix} -\delta_{d+1,j-1} & \delta_{d+1,j-1} & 0 \\ 0 & -\delta_{d+1,j} & \delta_{d+1,j} \end{pmatrix} = \\ \begin{pmatrix} -(1-w_{d,j}(t))\delta_{d+1,j-1} & (1-w_{d,j}(t))\delta_{d+1,j-1} - w_{d,j}(t)\delta_{d+1,j} & w_{d,j}(t)\delta_{d+1,j} \end{pmatrix}. \tag{C.2}$$

Så gjør vi det samme på høyre sidee av C.1,

$$\begin{pmatrix} -\delta_{d,j} & \delta_{d,j} \end{pmatrix} \begin{pmatrix} 1-w_{d+1,j-1}(t) & w_{d+1,j-1}(t) & 0 \\ 0 & 1-w_{d+1,j}(t) & w_{d+1,j}(t) \end{pmatrix} = \\ \begin{pmatrix} -(1-w_{d+1,j-1}(t))\delta_{d,j} & (1-w_{d+1,j}(t))\delta_{d,j} - w_{d+1,j-1}(t)\delta_{d,j} & w_{d+1,j}(t)\delta_{d,j} \end{pmatrix}. \tag{C.3}$$

vi ekspanderer så første uttrykket i (C.2) og første uttrykket i (C.3),

$$(1-w_{d,j}(t))\delta_{d+1,j-1} = \frac{t_{j+d}-t}{(t_{j+d}-t_j)(t_{j+d+1}-t_{j-1})},$$
$$(1-w_{d+1,j-1}(t))\delta_{d,j} = \frac{t_{j+d}-t}{(t_{j+d+1}-t_{j-1})(t_{j+d}-t_j)},$$

og ser at de er like. Vi ekspanderer så siste uttrykket i (C.2) og (C.3),

$$w_{d,j}(t)\delta_{d+1,j} = \frac{t-t_j}{(t_{j+d}-t_j)(t_{j+d+1}-t_j)}, \quad w_{d+1,j}(t)\delta_{d,j} = \frac{t-t_j}{(t_{j+d+1}-t_j)(t_{j+d}-t_j)},$$

som viser at de også er like. Midtelementet i (C.2) og (C.3) er en kombinasjon av det første og det siste elementet, og dermed er de også like. Lemmaet er dermed bevist. □

Hovedkommutativitetsteoremet følger nå.

Teorem C.1. *Derivasjon av en multiplikasjon av et sett av $T(t)$-matriser er kommutativ. Det vil si at for alle $d > 0$ og en gitt skjøtvektor kan en hvilken som helst matrise være derivertematrisen T'. dvs.*

$$T_d(t)\, T_{d+1}(t)\, \cdots\, T_{d+j-1}(t)\, T'_{d+j} = T'_d\, T_{d+1}(t)\, \cdots\, T_{d+j-1}(t)\, T_{d+j}(t). \tag{C.4}$$

Bevis. Dette følger av Lemma C.1 utvidet med en induksjon. □

C.3 Beta-funksjoner, bevis lemma 7.1

Lemma 7.1 sier følgende:

Den regulariserte ufullstendige betafunksjonen $I_t(a,b)$ har følgende egenskaper:

I	Null ved $t=0$	$I_0(a,b) = 0$,		
II	En ved $t=1$	$I_1(a,b) = 1$,		
III	Hermiteorden	$\frac{d^j}{dt^j} I_0(a,b) = 0$,	$j = 1,2,...,a$,	i.e. orden a ved start
		$\frac{d^j}{dt^j} I_1(a,b) = 0$,	$j = 1,2,...,b$,	i.e. orden b i enden

IV	Antisymmetrisk	$I_t(a,b) = 1 - I_{1-t}(b,a),$	i.e. symmetrisk hvis a=b
V	monoton	$\frac{d}{dt} I_t(a,b) > 0, \quad 0 < t < 1 \quad \text{og} \quad \frac{d}{dt} I_t(a,b) = 0, \quad t = \{0,1\}.$	
VI	Rekursiv	$I_t(a,b) = t\, I_t(a-1,b) + (1-t)\, I_t(a,b-1).$	
		$I_t(a,b) = I_t(a-1,b) - \frac{t^a(1-t)^{b+1}}{a\ \mathscr{B}(a-1,b)} = I_t(a,b-1) + \frac{t^{a+1}(1-t)^b}{b\ \mathscr{B}(a,b-1)},$	

Bevis. **I** følger direkte av (7.18), **II** følger av (7.19) fordi $\mathscr{B}(1;a,b) = \mathscr{B}(a,b)$. For å bevise **III** begynner vi med å differensiere (7.18), $\frac{d}{dt}\mathscr{B}(t;b,a) = t^a(1-t)^b$. Det følger at de etterfølgende deriverte er en sum der hvert ledd inneholder en faktor $t^j, \ \ j > 0$ for alle deriverte av orden opp til a, og en faktor $(1-t)^j, \ \ j > 0$ for alle deriverte av orden opp til b. Det følger at hvis $t = 0$ så er alle deriverte opp til ordre a null, og hvis $t = 1$ så er alle deriverte opp til ordre b null, noe som fullfører beviset av **III** .

For å bevise **IV** begynner vi med å regne ut

$$\begin{aligned}\mathscr{B}(1-t;b,a) &= \int_0^{1-t} x^b(1-x)^a dx \\ &= \int_t^1 (1-x)^b(1-(1-x))^a dx \\ &= \int_t^1 x^a(1-x)^b dx.\end{aligned}$$

Det følger at $\mathscr{B}(t;a,b) + \mathscr{B}(1-t;b,a) = \mathscr{B}(a,b)$, som sammen med (7.19) fullfører beviset av **IV**.

Å beviset **V** følger av at differensiering av (7.18) gir $\frac{d}{dt}\mathscr{B}(t;b,a) = t^a(1-t)^b$, og at den deriverte er positiv over $0 < t < 1$ og null ved $t = 0$ og $t = 1$.

For å bevise **VI**, rekursjonen, begynner vi å differensiere kjernen til (7.18),

$$\frac{d}{dx} x^a(1-x)^b = a\, x^{a-1}(1-x)^b - b\, x^a(1-x)^{b-1}.$$

Deretter går bakover med antideriverte (integrasjon) og deler begge sider med ab,

$$\frac{1}{ab} x^a(1-x)^b = \frac{a}{ab}\int_0^x t^{a-1}(1-t)^b dt - \frac{b}{ab}\int_0^x t^a(1-t)^{b-1} dt. \tag{C.5}$$

Vi legger deretter til $(a+b)x^a(1-x)^b$ på begge sider og omorganiserer så høyre side

$$\begin{aligned}\tfrac{a+b+1}{ab} x^a(1-x)^b = \ & \tfrac{a}{ab}\left(x^a(1-x)^b + \textstyle\int_0^x t^{a-1}(1-t)^b dt\right) + \\ & \tfrac{b}{ab}\left(x^a(1-x)^b - \textstyle\int_0^x t^a(1-t)^{b-1} dt\right).\end{aligned}$$

Vi integrerer så en gang til,

$$\frac{a+b+1}{ab}\int_0^x t^a(1-t)^b = \frac{1}{b} x \int_0^x t^{a-1}(1-t)^b dt + \frac{1}{a}(1-x)\int_0^x t^a(1-t)^{b-1} dt.$$

Til slutt multipliserer vi begge sider med $\frac{(a+b)!}{(a-1)!(b-1)!}$,

$$\frac{(a+b+1)!}{a!b!}\int_0^x t^a(1-t)^b = \frac{(a+b)!}{(a-1)!b!}x\int_0^x t^{a-1}(1-t)^b dt + \frac{(a+b)!}{a!(b-1)!}(1-x)\int_0^x t^a(1-t)^{b-1}dt.$$

som sammen med(7.17) og (7.19) fullfører beviset på den første delen av rekursjonen. Beviset for de to siste delene er bare en omorganisering av (C.5), som avslutter beviset. □

C.4 Rasjonale B-funksjoner, bevis teorem 7.4

Teorem 7.4 sier at en rasjonal B-funksjon, dvs en RB-funksjon er en ekte B-funksjon. Det følgende beviser teorm 7.4

Bevis. Vi følger definisjon 7.1. Funksjonen er helt klart en permutasjonsfunksjon (**D1**). Hvis vi beregner (7.21) ser vi at $B_{a,b}(0) = 0$ (**D2**) og $B_{a,b}(1) = 1$ (**D3**). Den 1.-deriverte av (7.21) er

$$B'_{a,b}(t) = (a(1-t)+bt+1)\frac{t^a(1-t)^b}{(t^{a+1}+(1-t)^{b+1})^2},$$

Det er tydelig at $B'_{a,b}(t) \geq 0$, fordi t og $1-t \geq 0$ for $t \in [0,1]$, og det følger at alle termer og faktorer er større eller lik null. Dermed er RB-funksjonen monoton (**D4**).

Vi omformulerer til $B_{a,b}(t) = \frac{\alpha(t)}{\beta(t)}$. Den deriverte $B^{(j)}_{a,b}(t)$ er en brøk der telleren er en sum der hvert ledd har faktorene $\alpha^{(r)}(t), r \leq j$ og $\beta^{(s)}(t), s \leq j$ der j er ordenen til den deriverte, og nevneren er $\beta(t)^{2j}$. Legg merke til at $\beta > 0$ når tß$[0,1]$.

a) Hvis $j \leq a$ i $B^{(j)}_{a,b}(t)$ så har hvert ledd i telleren en faktor t^{a+1-r}, $r \leq j$. Det følger at $B^{(j)}_{a,b}(0) = 0$ (hermiteorden på venstre side).

b) Fordi $\lim_{t\to 1}(1-t) = 0$, følger det at hvis $j \leq b$ så er $\lim_{t\to 1}\beta^{(j)}(t) = \alpha^{(j)}(t)$. Så hvis $j \leq b$ så er $B^{(j)}_{a,b}(1) = 0$ (hermiteorden høyre side).

Til slutt er RB-funksjonen $B_s(t)$ symmetrisk (**D5**) fordi

$$B_S(t) + B_S(1-t) = \frac{t^{S+1}}{t^{S+1}+(1-t)^{S+1}} + \frac{(1-t)^{S+1}}{(1-t)^{S+1}+t^{S+1}} = 1,$$

som sluttfører beviset. □

C.5 ERB-funksjoner, bevis teorem 7.5

Teorem 7.5 sier at $B_d(t)$ definert i (7.31), $B_y(t)$ definert i (7.33), $B_x(t)$ definert i (7.34) og $B_z(t)$ definert i (7.35) alle er symmetriske ekspo-rasjonale B-funksjoner (ERB-funksjoner) og at de er komplette B-funksjoner, definisjon 7.3.

Det følgende beviser teorem 7.5

Bevis. Vi følger definisjon 7.1. De fire funksjonene (7.31), (7.33), (7.34) og (7.35) er alle permutasjonsfunksjoner (**D1**). Dette følger av at de neste tre punktene er oppfylt. Vi ser at $B(0) = 0$ (**D2**) og $B(1) = 1$ (**D3**). Den 1.-deriverte $B'_d(t),\ t \in (0,1)$ i (7.36) er alltid positiv fordi eksponentialfunksjoner alltid er positive, og det samme argumentet gjelder for $B'_x(t)$ og $B'_z(t)$ i (7.37) også. Vi ser at $B'_y(t)$ i (7.36) også er positiv for $t \in (0,1)$ fordi nevneren i brøken alltid er positiv og telleren også alltid er positiv fordi $(1 - B_y(t)) > 0$ siden $B_y(t) < 1,\ t \in (0,1)$. Dermed er de alle funksjonene monotone (**D4**).

Symmetrien (**D5**) følger av:

Fordi $\left(t - \frac{1}{2}\right)^2 = \left(1 - t - \frac{1}{2}\right)^2$ er $B_d(t)$ symetrisk fordi

$$B_d(t) + B_d(1-t) = S_d \int_0^t \phi(s)ds + S_d \int_t^1 \phi(s)ds = 1.$$

De to neste funksjonene er konstruert til å være symmetriske,

$$B_y(t) + B_y(1-t) = \frac{\Psi(t)}{\Psi(t) + \Psi(1-t)} + \frac{\Psi(1-t)}{\Psi(1-t) + \Psi(t)} = 1,$$

$$B_x(t) + B_x(1-t) = \frac{1}{2}\left(1 - \Psi(1-t) + \Psi(t)\right) + \frac{1}{2}\left(1 - \Psi(t) + \Psi(1-t)\right) = 1,$$

og for $B_z(t)$ får vi

$$B_z(t) + B_z(1-t) = \frac{\theta(1-t)}{\theta(1-t) + \theta(t)} + \frac{\theta(t)}{\theta(t)) + \theta(1-t)} = 1,$$

som fullfører første del av beviset. Det vil si at $B_d(t)$, $B_y(t)$, $B_x(t)$ og $B_z(t)$ er alle symmetriske B-funksjoner.

Den andre delen, at $B_d(t)$, $B_y(t)$, $B_x(t)$ og $B_z(t)$ er komplette ERB-funksjoner, det vil si at hermiteordenen er ∞, henger på to av egenskapene til eksponentialfunksjonen, $(e_x)' = e_x$ og $\lim_{x\to+\infty} x_n e^{-x} = 0$ for alle n. For alle ERB-funksjoner vil eksponentialfunksjonen være en faktor på hvert ledd av alle deriverte. Sammen med det vi observerte i figur 7.16, det vil si mappingen fra $\mathbb{R}$ til $(0,1)$, vil dermed overstyringen av eksponentialfunksjonen sikre at alle deriverte er null ved $t = 0, 1$. Dette sluttfører beviset. □

C.6 Ordenssymmetri til en B-funksjon, bevis teorem 7.6

Teorem 7.6 sier at Beta-funksjoner og RB-funksjoner er ordenssymmetriske. Samt at ERB-funksjonene $B_d(t;\alpha,\beta)$ og $B_z(t;\alpha,\beta)$ er "ordenssymmetrisk" i den forstand at de er symmetrisk iht α og β.

Det følgende beviser teorem 7.6.

Bevis. Beviset for teorem 7.6 er delt inn i tre deler, en for hver type B-funksjon.

Beta-funksjoner: Beta-funksjonen er den regulariserte ufullstendige beta-funksjonen definert i (7.18). Hvis vi først ser på nevneren til brøken, ser vi at $\mathscr{B}(a,b) = \mathscr{B}(b,a)$ fordi det følger av (7.17). Dermed har vi en fellesnevner og vi kan derfor summere tellerne. Hvis vi setter (7.18) inn i (7.51) får vi

$$\mathscr{B}(t;a,b) + \mathscr{B}(1-t;b,a) = \int_0^t x^a(1-x)^b dx + \int_t^1 (1-x)^b x^a dx = \int_0^1 x^a(1-x)^b dx.$$

Vi ser nå at telleren og nevneren er like, og det betyr at summen er 1, og at Beta-funksjonene er ordenssymmetriske.

RB-funksjoner: RB-funksjonen er definert i teorem (7.21). Hvis vi setter (7.21) inn i (7.51) får vi

$$B_{a,b}(t) + B_{b,a}(1-t) = \frac{t^{a+1}}{t^{a+1} + (1-t)^{b+1}} + \frac{(1-t)^{b+1}}{(1-t)^{b+1} + t^{a+1}} = 1,$$

som verifiserer at RB-funksjonene er ordenssymmetriske.

ERB-funksjonene $B_d(t;\alpha,\beta)$ og $B_z(t;\alpha,\beta)$: $B_d(t;\alpha,\beta)$ er definert i(7.31) og (7.44). Hvis vi bytter α med β og erstatter t med $1-t$ i (7.44), viser det seg at $\phi(1-t;\beta,\alpha) = \phi(t;\alpha,\beta)$. Og dermed er

$$B_d(t;\alpha,\beta) + B_d(1-t;\beta,\alpha) = S_{\alpha,\beta} \int_0^t \phi(s;\alpha,\beta) ds + S_{\beta,\alpha} \int_t^1 \phi(s;\beta,\alpha) ds = 1.$$

som bekrefter at $B_d(t;\alpha,\beta)$ er ordenssymmetrisk i den forstand at den er symmetriske i henhold til α og β. Når det gjelder $B_z(t;\alpha,\beta)$ får vi,

$$B_z(t;\alpha,\beta) + B_z(1-t;\beta,\alpha) = \frac{e^{\frac{1}{(1-t)^\beta}}}{e^{\frac{1}{(1-t)^\beta}} + e^{\frac{1}{t^\alpha}}} + \frac{e^{\frac{1}{t^\alpha}}}{e^{\frac{1}{t^\alpha}} + e^{\frac{1}{(1-t)^\beta}}} = 1,$$

dvs. den er ordenssymmetrisk i den forstand at den er symmetriske i henhold til α og β. □

C.7 Balansesymmetri til en B-funksjon, bevis teorem 7.7

Teorem 7.7 sier at Rμ-funksjoner og ERB-funksjoner er balansesymmetriske.

Det følgende beviser teorem 7.7.

Bevis. Beviset for teorem 7.7 er delt i to deler, en for hver type B-funksjoner.

Rμ-funksjoner: Rμ-funksjonen er definert i Korrolar 7.1, (7.22). Hvis vi setter (7.22) inn i (7.52) får vi

$$B(t;\mu) + B(1-t;1-\mu) = \frac{(1-\mu)t^{S+1}}{(1-\mu)t^{S+1} + \mu(1-t)^{S+1}} + \frac{\mu(1-t)^{S+1}}{\mu(1-t)^{S+1} + (1-\mu)t^{S+1}} = 1,$$

som verifiserer at Rμ-funksjoner er balansesymmetriske.

ERB-funksjoner: $B_d(t;\mu)$, $B_x(t;\mu)$, $B_y(t;\mu)$ og $B_z(t;\mu)$ er definert i seksjon 7.7.2. Hvis vi erstatter μ med $1-\mu$ og t med $1-t$ i (7.39) ser vi at

$$\phi(t;\mu) = e^{-\frac{(t-\mu))^2}{t(1-t)}} \quad \text{og} \quad \phi(1-t;1-\mu) = e^{-\frac{((1-t)-(1-\mu))^2}{(1-t)t}} = e^{-\frac{(\mu-t))^2}{(1-t)t}},$$

dvs. de er like. Det følger at $S_\mu = S_{1-\mu} = \int_0^1 \phi(s;\mu)ds$, og vi får

$$B_d(t;\mu) + B_d(1-t;1-\mu) = S_\mu \int_0^t \phi(s;\mu)ds + S_{1-\mu} \int_t^1 \phi(s;\mu)ds = 1,$$

som bekrefter at $B_d(t;\mu)$ er balansesymmetrisk. Vi erstatter så μ med $1-\mu$ og t med $1-t$ i (7.43), og får

$$B_x(t;\mu) + B_x(1-t;1-\mu) = \mu\left(1 - e^{\frac{-2}{1-t}e^{\frac{-1}{t}}}\right) + (1-\mu)e^{\frac{-2}{t}e^{\frac{-1}{1-t}}} + (1-\mu)\left(1 - e^{\frac{-2}{t}e^{\frac{-1}{1-t}}}\right) + \mu e^{\frac{-2}{1-t}e^{\frac{-1}{t}}} = 1.$$

som bekrefter at $R_x(t;\mu)$ er balansesymmetrisk. Deretter erstatter vi μ med $1-\mu$ og t med $1-t$ i (7.42), og får

$$B_y(t;\mu) + B_y(1-t;1-\mu) = \frac{(1-\mu)\varphi(t)}{(1-\mu)\varphi(t) + \mu\varphi(1-t)} + \frac{\mu\varphi(1-t)}{\mu\varphi(1-t) + (1-\mu)\varphi(t)} = 1,$$

som bekrefter at $R_y(t;\mu)$ er balansesymmetriske. Til slutt erstatter vi μ med $1-\mu$ og t med $1-t$ i (7.41), og får

$$B_z(t;\mu) + B_z(1-t;1-\mu) = \frac{e^{\frac{\mu}{(1-t)}}}{e^{\frac{\mu}{(1-t)}} + e^{\frac{1-\mu}{t}}} + \frac{e^{\frac{1-\mu}{t}}}{e^{\frac{1-\mu}{t}} + e^{\frac{\mu}{(1-t)}}} = 1,$$

som bekrefter at $R_z(t;\mu)$ er balansesymmetrisk. Dette avslutter beviset. □

C.8 Samtidig orden og balansesymmetri, bevis teorem 7.8

Teorem 7.8 sier at følgende B-funksjoner er ordenssymmetriske og balansesymmetriske samtidig: Rμ-funksjonene, ERB-funksjonen B_d og ERB-funksjonen B_z.

Bevis. Beviset for teorem 7.8 er delt i to deler, en for hver av typene B-funksjoner.

Rμ-funksjoner: Rμ-funksjonen er definert i (7.22) i korrolar 7.1. Hvis vi bytter a og b, erstatter μ med $1-\mu$ og t med $1-t$ får vi

$$B_{a,b}(t;\mu) + B_{b,a}(1-t;1-\mu) = \frac{(1-\mu)t^{a+1}}{(1-\mu)t^{a+1} + \mu(1-t)^{b+1}} + \frac{\mu(1-t)^{b+1}}{\mu(1-t)^{b+1} + (1-\mu)t^{a+1}},$$

som klart summerer opp til 1. Dette viser at Rμ-funksjonene samtidig er balanse- og ordenssymmetriske.

$B_d(t,\mu,\alpha,\beta)$**-funksjoner:** Kjernen til $B_d(t,\mu,\alpha,\beta)$ er definert i (7.44). Hvis vi bytter α og β, erstatter μ med $1-\mu$ og t med $1-t$ i (7.44) får vi

$$\phi(t;\mu,\alpha,\beta)=e^{-\frac{|t-\mu|^{\beta+\alpha}}{t^{\alpha}(1-t)^{\beta}}} \quad \text{og} \quad \phi(1-t;1-\mu,\beta,\alpha)=e^{-\frac{|(1-t)-(1-\mu)|^{\beta+\alpha}}{(1-t)^{\beta}t^{\alpha}}}=e^{-\frac{|\mu-t|^{\beta+\alpha}}{t^{\alpha}(1-t)^{\beta}}},$$

som viser at $\phi(t;\mu,\alpha,\beta)=\phi(1-t;1-\mu,\beta,\alpha)$, som igjen betyr at $S_{1-\mu,\beta,\alpha}=S_{\mu,\alpha,\beta}$ og vi får

$$B_d(t;\mu,\alpha,\beta)+B_d(1-t;1-\mu,\beta,\alpha)=$$

$$S_{\mu,\alpha,\beta}\int_0^t\phi(s;\mu,\alpha,\beta)ds+S_{1-\mu,\beta,\alpha}\int_t^1\phi(1-s;1-\mu,\beta,\alpha)ds=S_{\mu,\alpha,\beta}\int_0^1\phi(s;\mu,\alpha,\beta)ds=1,$$

som bekrefter at $B_d(t,\mu,\alpha,\beta)$ samtidig er Balanse- og ordenssymmetriske.

$B_z(t,\mu,\alpha,\beta)$**-funksjoner:** Kjernen til $B_z(t,\mu,\alpha,\beta)$ er definert i (7.46).

Hvis vi erstatter μ med $1-\mu$ og t med $1-t$ i (7.41) får vi

$$B_z(t;\mu,\alpha,\beta)+B_z(1-t;1-\mu,\beta,\alpha)=\frac{e^{\frac{\mu}{(1-t)^{\beta}}}}{e^{\frac{\mu}{(1-t)^{\beta}}}+e^{\frac{1-\mu}{t^{\alpha}}}}+\frac{e^{\frac{1-\mu}{t^{\alpha}}}}{e^{\frac{1-\mu}{t^{\alpha}}}+e^{\frac{\mu}{(1-t)^{\beta}}}}=1,$$

som bekrefter at $B_z(t,\mu,\alpha,\beta)$ samtidig er Balanse- og ordenssymmetriske. Dette sluttfører beviset. □

C.9 Egenskaper til 2-p B-funksjoner $B(u,v)$

I seksjon 11.1 er en 2-p B-funksjon $B(u,v)$, $(u,v)\in[0,1]\times[0,1]$ definert, se (11.11). En B-funksjon $B(u,v)$ av hermiteorden d har følgende egenskaper:

- at verdien er 0 “internt" på to motsatte kanter (med konstant v-verdi),
- at verdien er 1 over hele de to andre kantene (med konstant u-verdi),
- den er symetrisk, dvs. $B(u,v)+B(v,u)=1$,
- den er “internt" C^d-glatt,
- at alle deriverte opp til orden d er 0 på kantene,
- og den er diskontinuerlig i alle hjørnene i “u-retningen"

Under følger 2 proposisjoner med bevis som statfester lista med egenskaper,

Proposisjon C.1. *Ved to motsatte kanter er funksjonsverdien 0 (merk at det ikke inkluderer hjørnene), ved de to andre kantene er funksjonsverdien 1 (inkludert hjørnene). dvs.*

$$B(0,v) = B(1,v) = 1, \quad v \in [0,1], \tag{C.6}$$
$$B(u,0) = B(u,1) = 0, \quad u \in (0,1). \tag{C.7}$$

Videre, i alle hjørner og langs alle kanter har alle deriverte opp til orden d verdien null,

$$\begin{aligned} D_u^{(i)} D_v^{(j)} B(0,v) = D_u^{(i)} D_v^{(j)} B(1,v) = 0, \quad v \in [0,1], \quad 0 \le i,j \le d, \quad i+j > 0, \\ D_u^{(i)} D_v^{(j)} B(u,0) = D_u^{(i)} D_v^{(j)} B(u,1) = 0, \quad u \in (0,1), \quad 0 \le i,j \le d, \quad i+j > 0. \end{aligned} \tag{C.8}$$

Bevis. I seksjon 11.1 har vi $g(u)$ (11.3), $a(u)$ (11.5), $t(u,v)$ (11.6) og $B(u,v)$ (11.8), og

hvis $u = 0$ og $v \in [0,1]$	da er $a(0) = 0$,	$g(0) = 1$	og $t(0,v) = 1$	og $B(0,v) = 1$,
hvis $u = 1$ og $v \in [0,1]$	da er $a(0) = 0$,	$g(1) = 1$	og $t(1,v) = 1$	og $B(1,v) = 1$,
hvis $v = 0$ og $u \in (0,1)$	da er $t(u,0) = 0$			og $B(u,0) = 0$,
hvis $v = 1$ og $u \in (0,1)$	da er $t(u,1) = 0$			og $B(u,1) = 0$,

som bekrefter (C.6) og (C.7).

Vi ser fra tabellen ovenfor at $g(0) = g(1) = 1$ og $t(0,v) = t(1,v) = 1, t(u,0) = t(u,1) = 0$. Fra egenskapene til en B-funksjon (7.1), følger det at (ved endene) $B^{(j)}(0) = B^{(j)}(1) = 0$, $j = 1,2,...,d$ hvor d er ordenen til B-funksjonen, og vi vet også fra definisjonen (11.4) at $g^{(j)}(0) = g^{(j)}(1) = 0,\ j = 1,2,\ldots,d$. På grunn av regelen om derivering av et produkt vil alle partiellderiverte til $B(u,v)$ opp til orden d inneholde et sett med termer, der hver term inneholder et produkt av enten g og noen deriverte av B (som er 0 fordi $B^{(j)} = 0,\ j = 0,1,\ldots,d$), eller $g^{(j)}$ og B (som er 0 fordi $g^{(j)}(0) = g^{(j)}(1) = 0,\ j = 1,2,\ldots,d$). Dette bekrefter (C.8), som igjen betyr at proposisjon C.1 er bevist. □

Domenet til $S(u,v)$ er $U = [0,1] \times [0,1] \subset \mathbb{R}^2$. For å bevise at $B(u,v)$ er C^d-glatt på domenet bortsett fra hjørnene $p_1 = (0,0)$, $p_2 = (1,0)$, $p_3 = (0,1)$, og $p_4 = (1,1)$, har vi følgende proposisjon.

Proposisjon C.2. *2-p blending funksjonen $B(u,v)$ (11.11) er $\in C^d(V)$, $V = U \backslash (0,0) \cup (1,0) \cup (0,1) \cup (1,1)$, og hvor $U = [0,1] \times [0,1] \subset \mathbb{R}^2$.*

- *B er diskontinuerlig i $p_1 = (0,0) \in U$ i retning $v_1 = (1,z),\ z \in [0,2)$*
- *B er diskontinuerlig i $p_2 = (1,0) \in U$ i retning $v_1 = (-1,z),\ z \in [0,2)$*
- *B er diskontinuerlig i $p_3 = (0,1) \in U$ i retning $v_1 = (1,-z),\ z \in [0,2)$*
- *B er diskontinuerlig i $p_4 = (1,1) \in U$ i retning $v_1 = (-1,z),\ z \in [0,2)$*

Bevis. 2-p blendingsfunksjonen $B(u,v) = g(u)\ B \circ t(u,v)$ består av B-funksjonen $B(t)$ beskrevet i seksjon 7.1, funksjonen $g(u)$ definert i (11.3) og funksjonen $t(u,v)$ definert i (11.6). Hver av disse faktorene vil bli analysert separate og deretter analysert sammen. Egenskapene til $B(t)$ er viktige og overføres også til $B(u,v)$. Beviset er delt i 4 deler, 1) handler om $g(u)$, 2) handler om $t(u,v)$, 3) handler om $B \circ t(u,v)$ og 4) ser på helheten.

1) Fra (11.3) ser vi at $g(u)$ er symmetrisk om $u = \frac{1}{2}$. Det følger at antallet etterfølgende deriverte som er null i $u = \frac{1}{2}$ bestemmer kontinuitetsnivået til g. Det følger at $g^{(j)}(u)|_{u=\frac{1}{2}} = B^{(j)}(u)|_{u=\frac{1}{2}}$. Følgelig får vi fra definisjon 7.2 at

$$g(u) \in C^d([0,1]).$$

2) Ved å analysere $t(u,v)$ i uttrykk (11.6) ser vi at:

- for en låst u-verdi $\bar{u} \in [0,1]$, er $\widetilde{t}(v) = t(\bar{u},v) \in C^0([0,1])$ og $\widetilde{t}(v)$ er stykkevis lineær.
- for en låst v-verdi $\bar{v} \in (0,1)$, er $\widehat{t}(u) = t(u,\bar{v}) \in C^0([0,1])$ og $\widehat{t}(u)$ er stykkevis glatt.
- For $v = 0$ ser vi at $\widehat{t}(u) = t(u,0) = \frac{0}{a(u)} = 0$ når $0 < u < 1$, se (11.5) og (11.6).
- For $v = 1$ ser vi at $\widehat{t}(u) = t(u,1) = \frac{1-1}{a(u)} = 0$ når $0 < u < 1$, se (11.5) og (11.6).

Det følger at t er kontinuerlig på V (V definert i proposisjon C.2). Vi ser at i de fire hjørnepunktene er t kontinuerlig i v-retningen, men diskontinuerlig i u-retningen.

- En nærmere undersøkelse av hjørnepunktene viser følgende:

  a) Fra formelen (11.6) får vi $t(p_1) = 1$.

  b) Fra (11.6) følger det at $t(u,v) = 1$ mellom de to kurvene $v = a(u)$ og $v = 1 - a(u)$ (11.5) i parameterplanet, og at $t(u,v) < 1$ ellers. Det følger at i starten av kurven $v = a(u)$ er $(v',a')|_{p_1} = (1,2)$. Når vi ser på retningen $(1,z)$, $0 \le z < 2$ ser vi at

  $$\lim_{u \to 0+} t(u, z\,u) = \frac{z}{2}. \tag{C.9}$$

  I motsetning til paragraph a) over er $\lim_{u\to 0+} t(u,z\,u) < 1$. Dermed er $t(u,v)$ diskontinuerlig fra p_1 i retning $v_1 = (1,0)$, $0 \le z < 2$. Hvis vi bruker symmetri følger det at $t(u,v)$ er diskontinuerlig fra hjørnene

  - $p_2 = (1,0)$ i retning $v_2 = (-1,z)$, $0 \le z < 2$,
  - $p_3 = (0,1)$ i retning $v_3 = (1,-z)$, $0 \le z < 2$,
  - $p_4 = (1,1)$ i retning $v_4 = (-1,-z)$, $0 \le z < 2$.

3) Siden $t(u,v)$ er kontinuerlig på V (V definert i proposisjon C.2), og diskontinuerlig i de fire hjørnene som beskrevet ovenfor, følger det at $B \circ t(u),v)$ er det samme.

Vi ser at $t(u,v)$ er delt i tre deler av to kurver i parameterplanet, $v = a(u)$ og $v = 1 - a(u)$. Hver del er internt glatt, har kontinuerlige 1.-, 2.-, ... deriverte. På de to kurvene ser vi da følgende

$$\begin{aligned} t(u,a(u)) &= t(u,1-a(u)) = 1, \\ B^{(j)}(t(u,a(u))) &= B^{(j)}(t(u,1-a(u))) = 0, \quad j = 1,2,...,d. \end{aligned}$$

Det følger at $B \circ t(u,v)$ ikke bare er kontinuerlig på V, men er $C^d(V)$. Dette fordi at på hvert sted en derivert av en eller annen orden er diskontinuerlig på t, er verdien av t lik 1 og alle deriverte opp til gjeldende orden er null (dette er illustrert for B i (11.9)).

4) Det følger at $B(u,v)$ vil arve alle diskontinuiteter fra både g og $B \circ t(u,v)$, derfor er $B(u,v)$ $C^d(V)$), mens den er diskontinuerlig fra punktene p_1 i retning v_1, p_2 i retning v_2, p_3 i retning v_3 og p_4 i retning v_4, som fulfører beviset . □

C.10 To-flateblending og kontinuitet

Følgende teoremer vil vise at det er mulig å fylle et firkantet hull i en større flate med en flate laget med to-flateblending slik at resultatet er en flate ønsket grad av kontinuitet.

Teorem C.2. *Gitt to flater $S_1(u,v)$ og $S_2(u,v)$ definert over samme domene $U = [0,1] \times [0,1] \subset \mathbb{R}^2$ og et tall d som bestemmer graden av kontinuiteten mot eventuelle naboflater. Da vil en flate $S(u,v) = B(u,v)\, S_1(u,v) + (1 - B(u,v))\, S_2(u,v)$ være $C^d(U)$ og alle deriverte på kantene (randa) er bestemt hvis,*
a) *blandingsfunksjonen $B(u,v)$ fra seksjon 11.1 er $\in C^d(V)$, der V er definert i proposisjon C.2,*
b) *og de to flatene $S_1(u,v)$ og $S_2(u,v)$ er begge $\in C^d(U)$ og alle deriverte på to motsatte kanter, vekselsvis i hver av flatene er bestemt,*
c) *og at i de fire hjørnene er verdien og alle deriverte opp til ordre d like i begge flatene S_1 og S_2.*

Bevis. Fra proposisjon C.2 og begrensningen **b)** ovenfor på overflatene S_1 og S_2, følger det at $S \in C^d(V)$, for V definert i proposisjon C.2.

Proposisjon C.2 forteller oss at B er diskontinuerlig fra de fire hjørnepunktene, $p_i,\ i = 1,2,3,4$, i gitte retninger $v_i,\ i = 1,2,3,4$. Imidlertid er oppførselen i de fire hjørnene symmetrisk eller antisymmetrisk rundt midtpunktet på overflaten. Vi trenger derfor kun å se på ett av hjørnepunktene. Derfor bruker vi hjørnepunktet $p_1 = (0,0)$ for undersøkelsen.

- Vi ser på verdien i hjørnepunktet p_1 ved å bruke (11.2), fra (C.6) vet vi at $B(0,0) = 1$, dvs.
$$S(0,0) = S_1(0,0) + 1(S_2(0,0) - S_1(0,0)) = S_2(0,0). \tag{C.10}$$
For å finne grenseverdien når vi på flaten S beveger oss i retning v_1 mot hjørnepunktet p_1 ser vi på
$$S(u,zu) = S_1(u,zu) + B(u,zu)\, \widetilde{S}(u,zu), \quad 0 \le z < 2.$$
Det følger fra (11.8) at
$$B(u,zu) \le 1, \quad 0 \le z < 2,$$
og sammen med begrensningen **c)** i teorem C.2 følger det at
$$\lim_{u \to 0} |\widetilde{S}(u,zu)| = 0.$$
Dermed er
$$\lim_{u \to 0} S(u,zu) = S_1(u,zu), \tag{C.11}$$
Det følger dermed fra (C.10) og (C.11) at S er kontinuerlig på $U = [0,1] \times [0,1] \subset \mathbb{R}^2$.

- Vi undersøker deretter 1.- og 2.-ordens deriverte i hjørnet p_1,
$$S_u(0,0) = S_{1u}(0,0) + 1(S_{2u}(0,0) - S_{1u}(0,0)) = S_{2u}(0,0), \tag{C.12}$$
$$S_v(0,0) = S_{1v}(0,0) + 1(S_{2v}(0,0) - S_{1v}(0,0)) = S_{2v}(0,0). \tag{C.13}$$
For å finne grenseverdien når vi på flaten S beveger oss i retning v_1 mot hjørnet p_1, skiller vi uttrykk (11.2), dvs.
$$S_u(u,zu) = S_{1u}(u,zu) + B_u(u,zu)\, \widetilde{S}(u,zu) + B(u,zu)\, \widetilde{S}_u(u,zu),$$
$$S_v(u,zu) = S_{1v}(u,zu) + B_v(u,zu)\, \widetilde{S}(u,zu) + B(u,zu)\, \widetilde{S}_v(u,zu),$$

hvor $0 \leq z < 2$. Videre følger det fra (11.8) at

$$B(u,zu) \leq 1, \quad 0 \leq z < 2,$$

og sammen med begrensningen **c)** i teorem C.2 følger det at både

$$\lim_{u\to 0+} |B(u,zu)\ \widetilde{S}_u(u,zu)| = 0,$$
$$\lim_{u\to 0+} |B(u,zu)\ \widetilde{S}_v(u,zu)| = 0.$$

Fra (11.3) ser vi at

$$\lim_{u\to 0+} g(u) = 1,$$
$$\lim_{u\to 0+} g^{(j)}(u) = 0, \quad j = 1,2,...,d,$$

og fra uttrykket (C.9) følger det at

$$\lim_{u\to 0+} B\circ t(u,zu) = B\left(\frac{z}{2}\right),$$
$$\lim_{u\to 0+} B^{(j)}\circ t(u,zu) = B^{(j)}\left(\frac{z}{2}\right),$$

og fra (11.7),

$$t_u(u,zu) = \frac{zu\ 2(1-2u)}{(2u(1-u))^2} = \frac{1-2u}{(1-u)^2}\ \left(\frac{z}{2}\right)\ \frac{1}{u},$$
$$t_v(u,zu) = \frac{1}{2(1-u)} \qquad = \frac{1}{1-u}\ \left(\frac{1}{2}\right)\ \frac{1}{u}.$$

Dermed følger det at

$$\begin{aligned}
\lim_{u\to 0+} B_u(u,zu)\ \widetilde{S}(u,zu) &= \lim_{u\to 0+}\Big((g'B + gB't_u)\ \widetilde{S}(u,zu)\Big) \\
&= B'\left(\frac{z}{2}\right)\ \frac{z}{2}\ \lim_{u\to 0+}\left(\left(\frac{1-2u}{(1-u)^2}\right)\frac{\widetilde{S}(u,zu)}{u}\right), \\
&= B'\left(\frac{z}{2}\right)\ \frac{z\sqrt{1+z^2}}{2}\ \lim_{u\to 0+}\frac{\widetilde{S}(u,zu)}{u\sqrt{1+z^2}}, \\
&= k\ d\widetilde{S}_{p_1}(\widehat{v}_1),
\end{aligned}$$

hvor $k = B'\left(\frac{z}{2}\right)\ \frac{z\sqrt{1+z^2}}{2}$ og $\widehat{v}_1 = \frac{v_1}{|v_1|}$.

$d\widetilde{S}_{p_1}(\widehat{v}_1)$ er retningsderiverte i p_1 i retning $\widehat{v}_1$. Fra restriksjon **c)** i teorem C.2 følger det at hvis $d > 0$ så har alle deriverte av orden 1 de samme verdiene i de to flatene S_1 og S_2 . Dermed må alle retningsderiverte av orden 1 i flaten $\widetilde{S}$ i p_1 være null-vektorer. Derfor er

$$\lim_{u\to 0+} |B_u(u,zu)\ S(u,zu)| = 0, \quad d > 0. \tag{C.14}$$

Vi bruker samme metode for å behandle den partielle deriverte B_v.

$$\lim_{u\to 0+} B_v(u,zu)\,\widetilde{S}(u,zu) = \lim_{u\to 0+}\left(gB't_v\,\widetilde{S}(u,zu)\right)$$
$$= k\,d\widetilde{S}_{p_1}(\widehat{v_1}),$$

hvor $k = B'\left(\frac{z}{2}\right)\ \frac{\sqrt{1+z^2}}{2}$ og $\widehat{v_1} = \frac{v_1}{|v_1|}$.

$d\widetilde{S}_{p_1}(\widehat{v_1})$ er retningsderivertene i punktet p_1 i retning $\widehat{v_1}$. Det følger her også at

$$\lim_{u\to 0+} |B_v(u,zu)\,S(u,zu)| = 0, \quad d > 0. \tag{C.15}$$

Det følger fra(C.14) og (C.15) at hvis $d > 0$ er de partiellderiverte

$$\lim_{u\to 0+} S_u(u,zu) = S_{1u}(u,zu), \quad 0 \le z < 2, \tag{C.16}$$
$$\lim_{u\to 0+} S_v(u,zu) = S_{1v}(u,zu), \quad 0 \le z < 2. \tag{C.17}$$

Som en konklusjon på denne undersøkelsen ser vi på grunn av begrensning **c)** i teorem er C.2 $S_{1u}(0,0) = S_{2u}(0,0)$ og at $S_{1v}(0,0) = S_{2v}(0,0)$ at det følger fra (C.12), (C.13), (C.16), (C.17) at i det minste er $S \in C^1(U),\ U = [0,1]\times[0,1] \subset \mathbb{R}^2$.

- For høyere ordens deriverte er argumentet analogt med førsteordens deriverte (dog litt mer komplisert), og det gjelder så lenge begrensning c) i teoremet er gyldig.

Dette sluttfører beviset. □

Bibliografi

[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 théoòreme 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 [138, 139]. *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. A method for sparse-matrix computation of b-spline curves and surfaces. In Ivan Lirkov, Svetozar Margenov, and Jerzy Waśniewski, editors, *Large-Scale Scientific Computing*, volume 5910 of *Lecture Notes in Computer Science*, pages 796–804. Springer Berlin Heidelberg, 2010.

[104] 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.

[105] 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.

[106] 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.

[107] 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.

[108] A. Lakså, B. Bang, and A. R. Kristoffersen. GMlib, a C++ library for geometric modeling. Technical report, Narvik University College, Narvik, Norway, 2006.

[109] 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.

[110] 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.

[111] 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.

[112] 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.

[113] Xin Li and Jianmin Zheng. Interproximate curve subdivision. *Journal of Computational and Applied Mathematics*, 244:36 – 48, 2013.

[114] C. T. Loop. Smooth Subdivision Surfaces based on Triangles. Master theses, University of Utah, Utha, USA, 1987.

[115] Charles Loop. A G^1 triangular spline surface of arbitrary topological type. *Comput. Aided Geom. Design*, 11(3):303–330, 1994.

[116] T. Lyche. *Discrete polynomial spline approximation methods*, volume 501/1976 of *Lecture Notes in Mathematics*, pages 144–176. Springer, Berlin/Heidenberg., 1976.

[117] T. Lyche and K. Strøm. Knot insertion for Natural Splines. *Annals of Numerical Mathematics*, 3:221–246, 1996.

[118] O. L. Mangasarian and L. L. Schumaker. Discrete Splines via Mathematical Programming. *SIAM Journal on Control*, 9:174–183, 1971.

[119] Martti Mäntylä. *An introduction to solid modeling*, volume 13. Computer Science Press, Incorporated, 1988.

[120] 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.

[121] L. Maurer. Über the Mittelwerte der Funktionen einer reellen Variablen. *Math. Ann.*, 47:263–280, 1896.

[122] D. S. Meek and D. J. Walton. Blending two parametric curves. *Computer-Aided Design*, 41:423–431, 2009.

[123] E. Mehlum. Nonlinear splines. *Computer Aided Geometric Design*, pages 173–207, 1974.

[124] 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.

[125] E. Meijering. A chronology of interpolation: From ancient astronomy to modern signal and image processing. In *Proceedings of the IEEE*, pages 319–342, 2002.

[126] J.Cotrina Navau and N.Pla Garcia. Modelling surfaces from planar irregular meshes. *Computer Aided Geometric Design*, 17(1):1 – 15, 2000.

[127] H. Olofsen. Blending functions based on trigonometric and polynomial approximations of the fabius function. In *Open Journal Systems*, Norsk Informatikk Konferanse, 2019.

[128] 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.

[129] 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.

[130] Jörg Peters and Ulrich Reif. The simplest subdivision scheme for smoothing polyhedra. *ACM Trans. Graph.*, 16(4):420–431, October 1997.

[131] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition, 1992.

[132] L. Ramshaw. Blossoming: A Connect-the-Dots Approach to Splines. Report 19, Digital Systems Research Center, Palo Alto, CA., 1987.

[133] 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.

[134] L. Ramshaw. Blossoms are polar forms. *Computer Aided Geometric Design*, 6(4):323–359, 1989.

[135] Ulrich Reif. A unified approach to subdivision algorithms near extraordinary vertices. *Computer Aided Geometric Design*, 12(2):153–174, 1995.

[136] R.F. Riesenfeld. On chaikin's algorithm. *Computer Graphics and Image Processing*, 4(3):304 – 310, 1975.

[137] W. Romberg. Vereinfacthe numerische integration. *Det Kongelige Norske Vid. Selsk. Forl.*, 28:30–36, 1955.

[138] 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.

[139] 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.

[140] I. J. Schoenberg. On spline functions. *Inequalities*, pages 255–291, 1967.

[141] I. J. Schoenberg. Cardinal Spline Interpolation. *CBMS-NSF Series in Applied Mathematics*, 12, SIAM, 1973.

[142] L. L. Schumaker. *Spline Functions: Basic Theory*. A Wiley-interscience publication. John Wiley & Sons Inc., New York, 1981.

[143] 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.

[144] T. W. Sederberg, J. Zheng, A. Bakenow, and A. Nasri. T-splines and T-nurces. *ACM Transactions on Graphics*, 22(3):477–484, 2003.

[145] 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.

[146] 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.

[147] 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.

[148] W. F. Sheppard. Central-difference formula. *Proceedings of the London Mathematical Society*, 31:449–488, 1899.

[149] 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.

[150] A. Sommerfeld. Eine besonders anschauliche ableitung des gaussischen fehlergesetzes. *Festschrift Ludwig Boltzman gewidmet zum 60. Geburtstage, 20. Februar 1904*, pages 848–859, 1904.

[151] M. Spivak. *A Comprehensive Introduction to Differential Geometry I*. Publish or Perish, Inc., Houston, Texas, USA, second edition, 1979.

[152] Jos Stam. Exact evaluation of catmull-clark subdivision surfacessubdivision surfaces at arbitrary parameter values. In *Proceedings of SIGGRAPH*, pages 395–404, 1998.

[153] 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.

[154] M. Szivasi-Nagy and T.P. Vendel. Generating curves and swept surfaces by blended circles. *Computer Aided Geometric Design*, 17(2):197–206, 2000.

[155] 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.

[156] T. N. Thiele. *Interpolationsrechnung*. B. G. Teubner, Leipzig, Germany, 1909. In German.

[157] I. Vardi. The Euler-Maclaurin Formula. *Computational Recreation in Mathematica*, pages 159–163, 1991.

[158] 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.

[159] W. Wang, B. Jüttler, D. Zheng, and Y. Liu. Computation of rotation minimizing frames. *ACM Trans. Graph.*, 27:2:1–2:18, March 2008.

[160] E. Waring. Problems Concerning Interpolations. *Philosophical Transactions of the Royal Society of London*, 69:59–67, 1779.

[161] J. Warren. Blending Algebraic Surfaces. *ACM Trans. Graph.*, 8(4):263–278, 1989.

[162] J. Warren. Barycentric coordinates for convex polytopes. *Advances in Computational Mathematics*, 6(1):97–108, 1996.

[163] E. W. Weisstein. Rose. http://mathworld.wolfram.com/Rose.html, 2006.

[164] H. Wenz. Interpolation of curve data by blended generalized circles. *Computer Aided Geometric Design*, 13(8):673–680, 1996.

[165] 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.

[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.

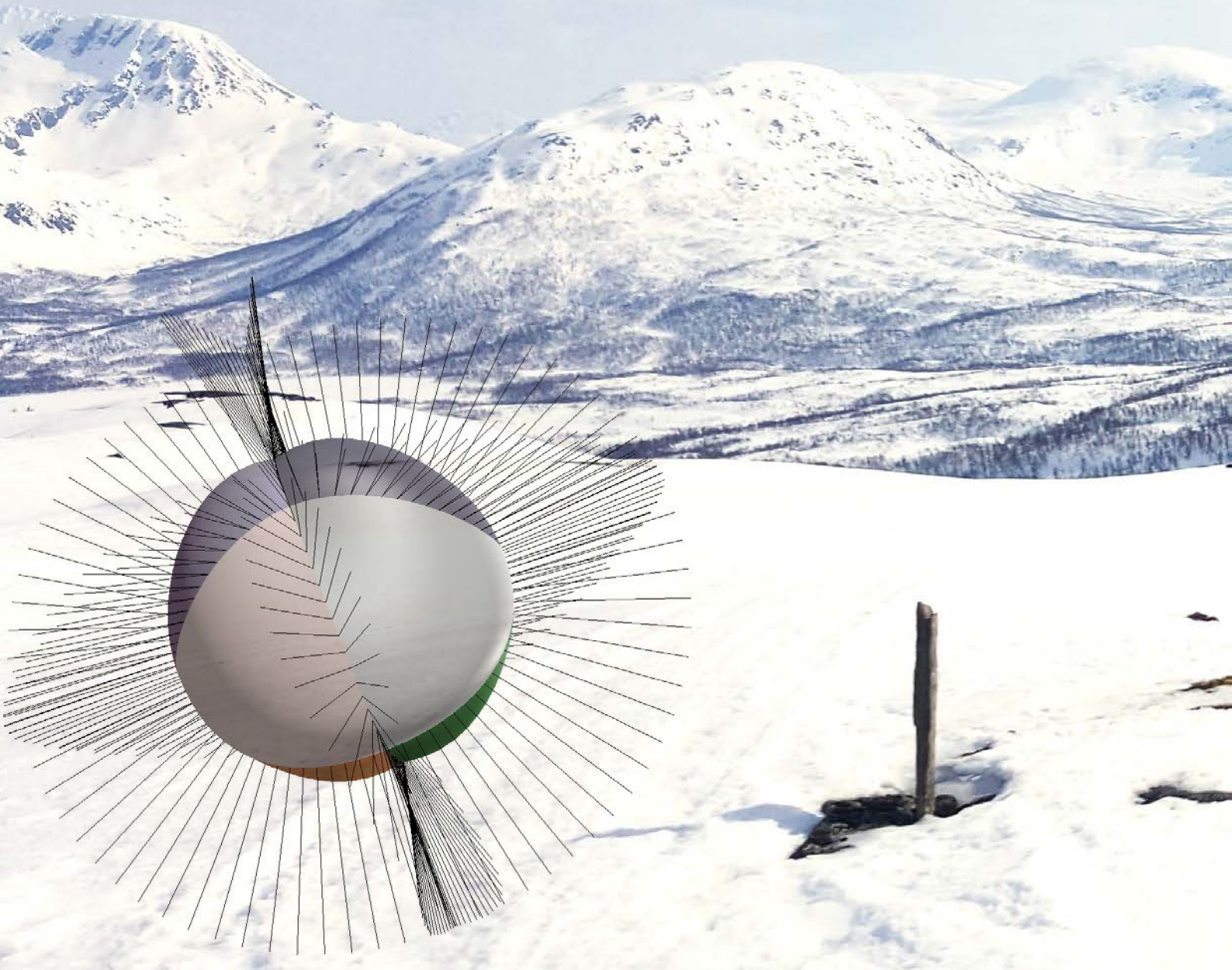
Liste av akronymer

2D	–	Beskriver objekter i det euklidske rommet $\mathbb{R}^2$
3D	–	Beskriver objekter i det euklidske rommet $\mathbb{R}^3$
API	–	Applikasjonsprogrammeringsgrensesnitt
C++	–	Objektorientert programmeringsspråk
CAGD	–	Datastøttet geometrisk design
ERB	–	Ekspo-rasjonale B-funksjoner
ERBS	–	Ekspo-rasjonale B-splines
eVITA	–	e-Science, inkluderer materiale fra både Computational Science og Software Engineering samt andre emner som grafikk, virtuell virkelighet, generell informatikk
FFD	–	Friform deformasjonsmetode
GMlib	–	C++ bibliotek for geometrisk modellering og simuleringer, utviklet av FoU-gruppen Simuleringer ved UiT Norges arktiske universitet
GM_Wave	–	C++ wavelet-biblioteket, utviklet av FoU-gruppen Simuleringer ved UiT Norges arktiske universitet
GPU	–	Grafikkprosessorenhet
GPGPU	–	GPU-programmering for generelt bruk
GUI	–	Grafisk brukergrensesnitt
IEEE	–	Institutt for elektrisk og elektronisk utvikling.
NaN	–	Ikke et tall
NURBS	–	Ikkeuniforme rasjonale B-splines
NUERBS	–	ERBS analogen til NURBS
ODE	–	Ordinær differensialligning
OpenGL	–	Programvaregrensesnitt til grafisk maskinvare
PDE	–	Partiell differensialligning
SINTEF	–	Stiftelsen for industriell og teknisk forskning
SISL	–	SINTEFs splinebibliotek
UIO	–	Universitetet i Oslo
UiT	–	Norges arktiske universitet

Register

Geometri – som betyr jordmåling på gammelgresk – har vært en viktig faktor i utviklingen av vitenskapen og dermed også industri, design, konstruksjon og produksjon og da industri/samfunnsutvikling generelt.

I dag er anvendt/industriell geometri en svært viktig faktor i blant annet produktutvikling, virtuelle systemer, systemer for gjenkjenning og orientering og kunstig intelligens.



GEOFO
Geometriforlaget
The geometry publishing house

Støttet av
NORGES ARKTISKE UNIVERSITET • UIT •

9 788269 306507
ISBN 978-82-693065-0-7