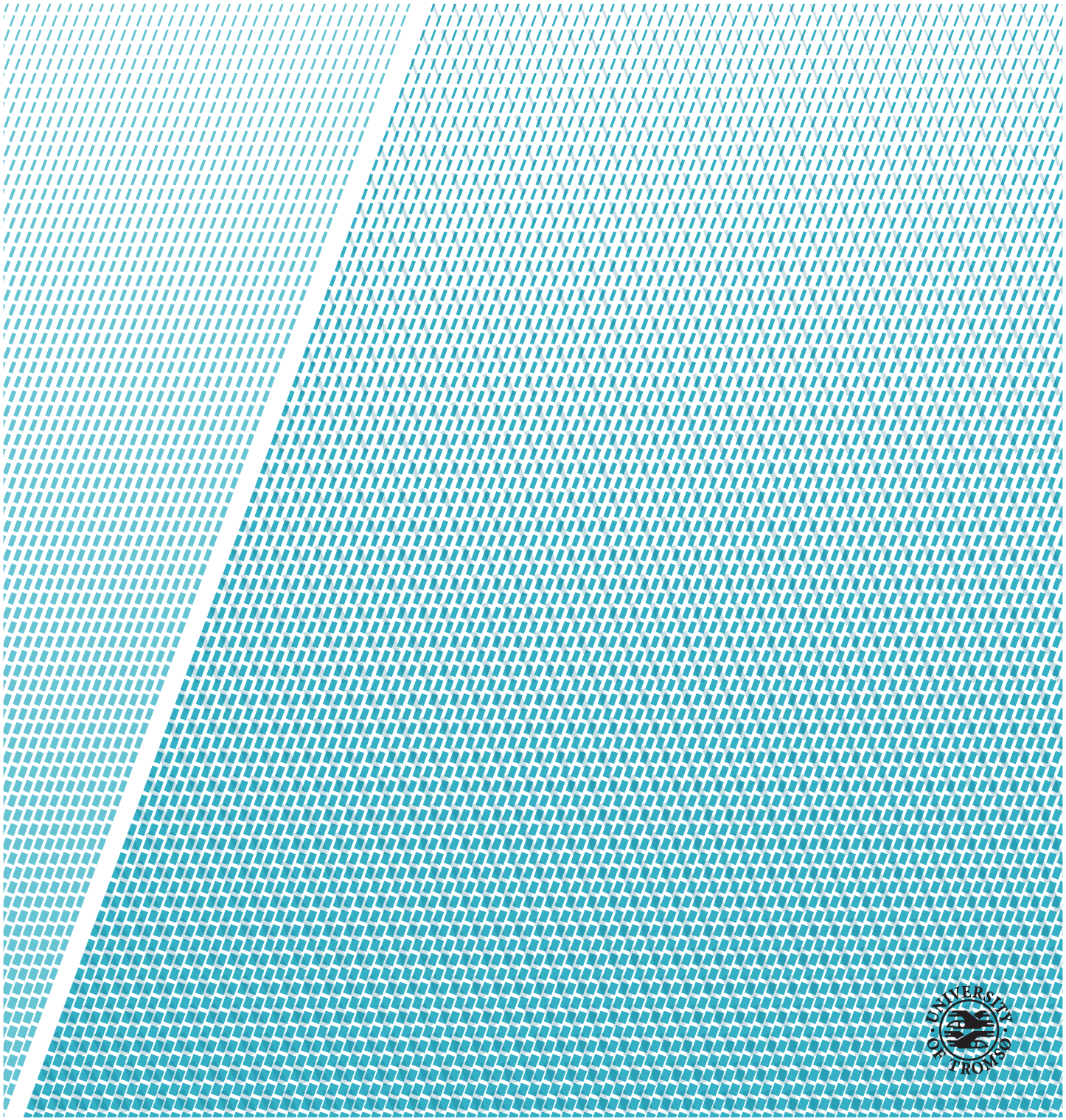
UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Science and Technology
Department of Computer Science

Neo: Virtual Object Modeling using Commodity Hardware

Thomas Bye Nilsen

INF-3990 - Master's Thesis in Computer Science
15th of May 2019



“You are the universe expressing itself as a human for a little while.”
–Eckhart Tolle

“It pays to keep an open mind, but not so open your brains fall out.”
–Carl Sagan

“How come you feel so alone
Is it the rage inside?”
–Anders Fridén

Abstract

Recent developments in augmented reality technology have paved way for new applications in a wide range of areas. These include the commercial markets, medicine applications, military applications and education. The technology provides immersive images to enhance our perception of the world. Augmented reality addresses challenges related to problem-solving by seamlessly integrating digital images into real-world images.

In the context of construction and maintenance industry, project inspections can be time-consuming and tedious. These inspections involve usages of expensive and specialized hardware. Some inspections even use physical blueprints and drawings along with standardized measurement tools. This approach can pose practical challenges and be prone to errors.

In this thesis we present Neo, a surface reconstruction system on commodity hardware. It utilizes augmented reality technology by scanning physical surroundings and reconstructs them as virtual objects. They are displayed on top of the camera's live preview of the real world. By using a pipeline architecture we model the physical surroundings in terms of their shapes and visual appearances. Cyber-physical information about the reconstructed virtual models are annotated in real-time. Evaluations of the system show us potentials to create realistic copies of physical objects.

Acknowledgements

First and foremost I want to thank my supervisor Robert Pettersen for his guidance, advice and continuous feedback throughout writing this thesis. I also want to thank the members of the IAD research group for social meetups, Monday meetings, inputs and feedback.

Further I want to thank my parents for their support and patience throughout my time at University of Tromsø. I want to thank my friends for providing me with friendships and social company. My classmates have been very important for me to achieve this. A special thanks goes to Robert Nilsen and Therese Wikbo. Long drives into the night are still to come.

I would also like to thank Nikolai Magnussen for dabbing throughout the year and Ole Jakob Hegelund for supplying me with unhealthy amounts of Red Bull.

Lastly, I would like to thank the staff at the Department of Computer Science at University of Tromsø for providing a fantastic learning environment!

Let these words be remembered.

Contents

List of Figures

List of Tables

List of Code Listings

List of Definitions

1 Introduction

In recent years we have seen an increased interest in Augmented reality (AR) technology. The technology has seen a growth in usage in several areas – commercial business, entertainment, military applications and medical applications. One of the reasons for this is the support for AR on off-the-self mobile hardware, as opposed to stationary hardware.

Another area that can benefit from AR technology is the Architectural, Engineering, Construction, and Facility Management (AEC/FM) industry. Today the industry faces challenges related to constructing new buildings and performing maintenance on existing buildings. Personnel in this area need access to detailed information regarding construction projects. The access to the information varies – physical blueprints, construction drawings and data in different formats maintained by different systems. This poses several challenges to the personnel working in these environments.

Performing construction site assessments can also be a tedious and time-consuming process with visual inspections. The physical drawings can be out-of-date and the measurement process using conventional tools can be prone to errors [1]. Mentally bridging the gap between physical 2D drawings and facility site objects can be a challenge of its own. Making informed decisions require precise data with a certain level of detail.

Therefore, researchers are adopting emerging technologies in the AEC/FM industry. Digitized construction models, or Building Information Models (BIMs),

have been used extensively during construction projects [2]. Rich information about building sites helps personnel follow up on construction projects. In light of the importance of performing construction projects and facility inspections accurately, we see a need to remove error-prone labor and to improve the effectiveness of construction projects.

Given the challenges on following up and inspecting construction projects, we see a potential for using AR in the field of AEC/FM. A motivation is that commodity hardware is becoming increasingly available and cheaper.

This thesis presents Neo, an AR architecture providing surface reconstruction capabilities on off-the-shelve commodity hardware. The architecture is intended to be realized on inexpensive mobil devices, addressing the issues related to building-site inspections. We propose that Neo will overcome these issues by utilizing AR in an agile and practical manner.

1.1 Problem Definition

Currently, the challenge of performing surface reconstruction is approached using specialized hardware. The hardware can potentially be expensive and require trained personnel. Some attempts require deploying markers in the physical environment before running the system. We wish to investigate an approach to the same problem using inexpensive commodity hardware in an marker-less approach. We formalize the problem statement as follows:

This thesis will design, implement and evaluate surface reconstruction capability on commodity hardware.

1.2 Scope and Limitations

This thesis shall specify requirements, design and implementation of a system that performs surface reconstruction in real-time on commodity hardware. The implementation will focus on commodity hardware that consists of a camera and a touchscreen display. The system shall capture virtual 3D models based on physical surroundings.

The system will render the virtual models in real-time on an overlay on the display. It provides cyber-physical information about the reconstructed surfaces by annotations. The information will be computed using the data captured from the camera. We focus on mobile platforms because they have sufficient

support for Mobile Augmented Reality (MAR) applications.

We require the system to support small surface areas. These surfaces are limited to be planar horizontal and vertical surfaces. We will consider simple curved surfaces in a limited manner. We limit the curved surfaces to not contains holes. The inner structures underneath surfaces are ignored as well.

We do not store the virtual models or process them any further beyond the time of capturing. The issue of scalability will also remain beyond the scope of this thesis. However, we defer these problem to future work.

1.3 Methodology

In the report final report of the Task Force on the Core of Computer Science, the discipline of computer science is described. A commonly accepted definition of computing as a science is "the discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application" [3]. This description of computing as a discipline was presented in 1989 by the *Task Force on the Core of Computer Science*, formed by ACM and the IEEE Computer Society. The report also outlines three major paradigms as a part of the field.

Theory is rooted in mathematics. The objects of study and their relationships to each other are characterized through a definition. Their behvaior is described by forming hypotheses. These hypotheses are subsequently proven or falsified to develop coherent, valid theories.

Abstraction is rooted in the experimental scientific method. It is based on four steps – forming a hypothesis, specifications, design and experiments, and finally analyzing the results.

Design is rooted in engineering and consists of four steps – state the requirements, specifications, design and implement the systems and finally test the system. These steps are iterated if the system does not meet the requirements.

This thesis is rooted in *design*. We describe the requirements Neo needs to meet in order to solve our problem statement. We specify the specifications of Neo using knowledge about AR and commodity hardware. From this, we design a prototype to develop and test. We perform experimental evaluation to verify that our system solves the problem statement.

1.4 Context

This project is written as a part of the Information Access Group at University of Tromsø. The group targets research on large-scale information access applications. The targeted topics range from security, fault-tolerance and privacy. The systems on which application run are public and private cloud computing environments. This section gives a brief summary of the research group's previous scientific work.

The group has done extensive research in sports science. One of the project is Bagadus [4], a prototype system used for capturing soccer player statistics and human expert annotations. The system allows for extracting video playbacks from tagged game events or player statistics. The system employs stitching of multiple video sources to form a full view of the soccer pitch. The system integrates with the ZXY Sport Tracking by automating some of the manual labor related to analyzing the athletes' performance. The system is deployed on Alfheim Stadium in Tromsø, Norway. The project was developed in collaboration with Simula Research Laboratory in the University of Oslo.

The ability to zoom in and pan on details using the video sub-system in Bagadus is found in [5]. From the initial cylindrical view, the image can be zoomed in for greater details. Another part of Bagadus, Muithu [6] is a touch-based system used to perform real-time annotations on handheld devices. In the context of Bagadus, the annotations capture events using the video-playback pipeline. We relate Neo to the two project involving capturing imagery in real-time and annotations using handheld devices with touch-displays.

1.5 Outline

The remainder of this thesis is structured as follows:

Chapter 2 contains the required background information regarding AR and mobile platforms. We look at how mobile platforms use AR technologies. Additionally, related work to our thesis is presented.

Chapter 3 outlines the design and architecture of Neo. By considering the problem statement and using the abstractions provided in Appendix A, we design the architectural components. Each of these components aims at solving a sub-set of the overall problem. Then, we describe how the we connect the components to solve the problem.

Chapter 4 describes the implementation of Neo, giving details of all the com-

ponents in the architecture. We highlight the major implementation details of our system.

Chapter 5 contains an experimental evaluation of the system. We perform experiments on Neo to investigate the capabilities of Neo to solve our problem statement. We perform qualitative and quantitative experiments using an experimental environment. A discussion of the experimental results and a summary follows.

Chapter 6 concludes our thesis and outlines future work.

2 Background

The field of Augmented reality (AR) have seen new development and innovation over the recent years. One of these developments is that AR support is becoming increasingly available on inexpensive commodity hardware. Hand-held devices, in particular smartphones, have been equipped with enhanced hardware and software supporting AR. Our thesis aims to determine the extent to which surface reconstruction can be performed using commodity hardware, and whether the reconstructed 3D models can be rendered on an overlay of the physical world.

This chapter covers the basic principles of AR technology. First, key terms and concepts are introduced. Then, we look at related work we draw inspiration from.

2.1 Augmented reality

Azuma [7] and Milgram [8] define AR as the ability to combine real-world imagery with computer-generated graphics in real time. The Reality-Virtuality continuum [8] depicted in Figure 2.1 spans between the real world and a completely synthetic and virtual world. AR is closer to the real world in the continuum because the real world is the predominating source of perception.

An AR experience involves immersing the user with virtual objects in the

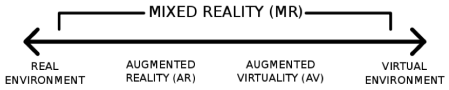


Figure 2.1: RV continuum coined by Milgram [8].

real world. In order to immerse the user, these objects need to be aligned and positioned with respect to physical surfaces. One of the goals of AR is to enhance the user's perception by integrating virtual objects with the surroundings. Computer vision renders 3D virtual objects from the same viewpoint from which the images of real scenes are being taken by a tracking camera [7].

Several different techniques are used to achieve this, namely tracking and reconstruction. First, tracking involves recognizing fiducial markers, optical images or feature points to estimate the device's pose in the world space. Then, based on data obtained from tracking, a world coordinate system is constructed. The world coordinate system is used to logically define virtual objects to be superimposed.

AR overcomes the problem of users having to mentally bridge the gap between physical reality and relevant digital information. AR is used in several areas, ranging from education, entertainment, medicine, military application and architecture [9]. Relevant domain-specific information is annotated to enhance the user's perception.

There are three main paradigms that realize AR technology from a hardware perspective [9]. The first paradigm involves a Head Mounted Device (HMD); a device attached to a user's head such as AR-glasses that can project computer generated graphics on semitransparent mini-monitors placed between the eyes and lenses. Examples on HMD hardware components are Google Glass [10] and Microsoft Hololens [11]

The second paradigm uses mobile devices which encompasses all the needs of AR; camera, display, computing power and software. This is contrary to the first paradigm in which the user is to a greater extent immersed in AR because the display is positioned close to the eye. The third paradigm is monitor-based, and is comparable to the previous, but camera, device and computing power is not necessarily encompassed into one single device.

2.1.1 AR tracking

There are two categories of tracking in AR, namely marker-based and marker-less tracking [12]. Marker-based tracking involves attaching markers in the

physical world prior to augmentation. An example is Fiducial Marker Based (FMB) tracking which requires the markers to be continuously visible throughout the augmentation. These markers can be QR-codes [13].

Marker-less tracking on the other hand, is based on the natural characteristics of the physical environment. These visual features are highly distinguishable properties of physical objects' appearances. Mathematical algorithms are used to formally determine these features [12]. Recent development has led to a shift from marker-based tracking to marker-less tracking [14].

2.1.2 Physical environment

A fundamental basis for conducting research of AR is an understanding of the physical world in which AR is applied. The physical environment subjected for AR can vary widely in characteristics depending on usage. The physical environments can contain an arbitrarily number of shapes of different categories, such as planes and curved surfaces. For our contribution we expect such surfaces to occur in the physical environment. Construction sites and common workplace rooms are examples of such environments. The shapes of the surfaces in the physical environment have certain characteristics associated with them, such as colors of the surfaces and the surrounding light conditions. The surfaces vary in size, color, orientation and position.

We base our perception of physical objects on their surface. From this we gain a visual understanding of their size, color, orientation and position. When we move the physical objects around we use the surfaces as a basis for their placements on a location. Using this property we allow the surfaces along with their characteristics, to be a basis for our understanding of the physical environment.

2.1.3 AR on smartphones

In recent years AR on smartphones has become more available in mainstream markets [15]. Smartphones are the dominant platform for providing Mobile Augmented Reality (MAR) experiences to users.

MAR has been used in several areas. In 2016, the mobile application Pokémon Go used AR technology to display virtual Pokémons combined with real-time images of the real world [16], using location-based techniques to determine position of the phone before rendering the statically defined virtual objects.

AR Browser Software Development Kits (SDKs) [12] use geo-located augmenta-

tion similar to Pokémon Go. However, they use marker-based tracking. ARmedia SDK [12] use feature-based visual tracking to understand the physical environment. It utilize Infrared camera (IRC), depth perception and inertial sensors for coherent augmentation [17]. Both planar images and more complex 3D objects, regardless of their size and geometry, can be scanned. This is contrary to earlier methods in which the physical environment was prepared prior to augmentation.

Since Pokémon Go arrived, several commercial mobile AR applications have been developed. Nike Fit uses AR to measure the end-users' feet to find suitable shoes [18]. Posten Norge uses AR to visualize the exact measurements of a recipient's package. The intention is to assist the recipient in figuring out whether the package can fit into their means of transportation [19]. IKEA Place uses AR to visualize furniture in a physical space [20]. The 3D models of furniture are projected onto surfaces to see how they fit in with the rest of the environment. Common for above-mentioned applications are that they aim to solve practical every-day challenges using AR technology.

2.2 Mobile AR platforms

There are a multitude of mobile platforms available for research and commercial purposes. Android is one of the most widely used platforms on the markets. It is an open source, Linux-based software stack created for mobile phones and other embedded systems [21, 22]. The foundation for Android is the underlying Linux kernel providing security, low-level memory management, network stack and driver model. An overview of the Android architecture can be seen in Figure 2.2

Applications running on top of the software stack utilize the Java Application Programming Interface (API) framework. The feature-set of the Android Operating System (OS) is exposed through the API. The Android Runtime (ART) executes the Dalvik Executable format and Dex bytecode specification [24].

2.2.1 Google Tango

Google Tango is an AR computing platform developed by Google. It utilizes computer vision to enable mobile devices to detect their position in physical environment without using Global Positioning System (GPS) or external sensors [25]. It enables 3D mapping, indoor navigation, AR and physical space measurement on inexpensive specialized hardware devices. It is built on the concepts of motion tracking, depth perception and area learning. An IR depth

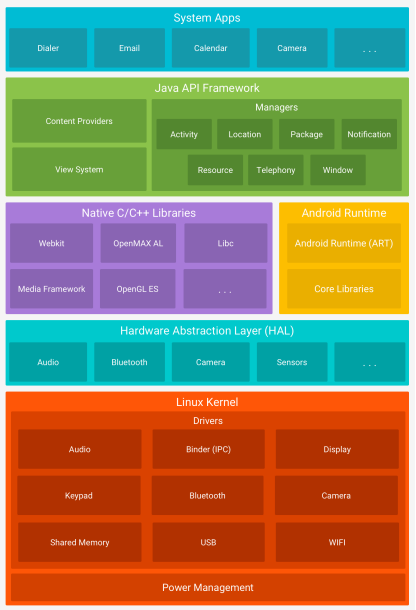


Figure 2.2: High-level overview of the Android architecture [23].

sensor, a gyroscope, an acceleratormeter and image processing algorithms are combined to make Tango understand the physical world [26]. As of writing, Google Tango has been discontinued by Google, and Google is advocating ARCore as its replacement.

2.2.2 Google ARCore

Google ARCore is a framework used to build AR applications on the Android platform [27]. It uses marker-less tracking to calculate the smartphone's pose. The framework enables a smartphone to perform motion tracking, environmental understanding and light estimation. Google ARCore tracks visual features in the physical world and build its understanding from them. Orientation and distances are measured as well [27].

A key difference between Google ARCore and Google Tango is that the latter requires specialized hardware [28]. As stated in [26], Google Tango is able to acquire point clouds from surface scans and construct polygon meshes from them. A point cloud is a set of points in a three-dimensional space that represents a physical space [29]. When enough points are brought together, interesting features can begin to arise. Google ARCore provides debugging features for acquiring point clouds [30], however at the time of writing polygon meshing are not supported.

2.2.3 Apple ARKit 2

Apple ARKit 2 is an AR platform aimed for mobile iOS devices. It uses device motion tracking, camera scene capture and image tracking to enable AR experience. ARKit 2 has extended the features provided by Apple ARKit 1 by enabling image recognition. It also adds the ability to recognize known objects, like sculptures and furniture [31]. While Apple ARKit is aimed specifically at the iOS platform, Google ARCore supports the Apple platform as well [32].

2.3 Related work

This section contains work that is related to this thesis. We look at interesting findings from the projects and position our contribution to theirs.

2.3.1 Augmented reality in AEC/FM

In the Architectural, Engineering, Construction, and Facility Management (AEC/FM) industry AR has been used to visualize 3D models acquired from scans of physical surfaces. [33] proposes HD4AR, a model-based system used to reconstruct models of physical construction objects based on photographs and 3D point clouds. The construction objects can be walls, doors, ceilings and floors.

When users want to query cyber-information of a construction object, photographs is submitted to the system. The system is able to recognize construction objects by applying feature detection between photographs and the point cloud. From this, the camera's pose is determined to find the position of the user. The recognition of construction and annotated cyber-information are overlaid on top of the photographs.

Users can query relevant information from an existing Building Information Model (BIM), a digital representation of a physical building [34]. By tapping a photograph on a display, the information is annotated. Annotation of data about the objects and use of point clouds are interesting for our thesis. However, we do not involve BIM models, we consider the virtual objects directly when we want to query relevant cyber-information.

[35] uses the Kinect v2 sensor for RGBD data acquisition to create a model of the Construction Information Technology (CIT) Lab at Cambridge University. A corresponding BIM is manually created from the same room using a desktop application. When the Kinect v2 device has reconstructed the environment to

its greatest extent, the corresponding BIM was transparently overlaid. The result shows the planned construction provided by the BIM, combined with the virtual models provided by the data acquisition of the physical construction objects.

For the data acquisition and displaying virtual objects we use a similar approach. The environment is scanned for both planar surfaces and point clouds using a marker-less tracking approach. However, we do not involve back-end services to accomplish similar results.

Constructor Developer Tool [36] is a Tango-based app that allows the mobile device to scan meshes of physical environment and export the data into different file formats. Wayfair™ View [37] is another mobile application built on Google Tango that can render 3D models of furniture in the physical world. Common for these two applications is the ability to create a 3D model of the world. Figure 2.3 shows the Constructor Developer Tool application in use.

We aim to accomplish similar effects as these applications, but we strive to implement software that can run on off-the-self commodity hardware. Google Tango requires specialized hardware and software.

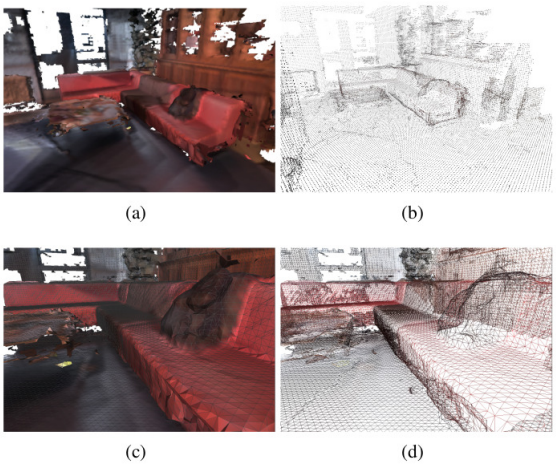


Figure 2.3: a) Global view of the 3D mesh. b) Point cloud of the mesh. c) Zoom on the textured triangulated surface. d) Colored wireframe model [26].

2.3.2 Mesh generation

[38] created a system for mesh generation using Mobile Edge Computing (MEC). As a part of the project, they use smartphones compatible with Google Tango [25]. The smartphones collected point clouds and offloaded them to

MEC units. They used the Point Cloud Library (PCL) to reconstruct a mesh from the point clouds.

PCL is a library used to perform surface reconstruction from point clouds [39]. It use a set of underlying libraries to triangulate and visualize reconstructed surfaces, and is available on several platforms.

Our thesis aims to perform mesh generation using a simpler point cloud meshing algorithm. We also aim to accomplish this on off-the-self commodity hardware, and avoid the requirements of specialized hardware in Google Tango. However, the principle remains the same; reconstructing physical objects with a wide range of surfaces to consider.

3

Design and Architecture

This chapter outlines the overall architecture of Neo. The problem at hand is to test the ability for commodity hardware to perform scanning of physical surfaces. We design Neo to scan physical surfaces and reconstruct them as virtual entities on commodity hardware. We design the individual components to be modular which makes the application easier to develop, extend and maintain.

On Figure 3.1 we see the how the commodity hardware gains an environmental understanding from interpreting the raw imagery from the on-board camera. We represents the interpretation of the surroundings as abstractions and uses them to reconstruct the surfaces. Then, they are rendered on the display.

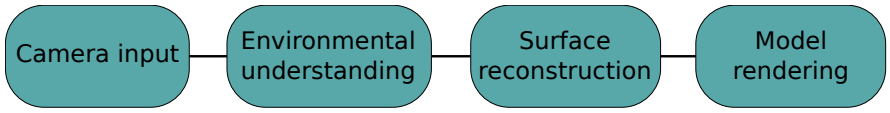


Figure 3.1: High-level overview of the architecture.

The complexity of the problem requires a divide-and-conquer approach. From the design components we design a pipeline architecture. A motivation for using a pipeline architecture is to achieve a stepwise approach to solving the problem. The complexity of problem requires it to be broken down into smaller, manageable parts.

Each pipeline step has a well-defined area of responsibility to reduce com-

plexity and elevate the abstraction level. The loosely connected components enable data to be processed independently. The modularity adheres to Single responsible principle (SRP) [40], and makes it easier to detect bottlenecks and parallelize the execution of individual steps if need arise.

3.1 Design components

The design components are loosely connected abstractions, each of which encapsulates data and operations on that data. They provide means of communication through well-defined interfaces. Separating data and operations on them enables a clearly defined area of responsibility each component must provide. Each component consider only parts of the overall problem. A side-effect of using SRP is reuse of components. We design the components to enable modularity between them.

We follow Dependency inversion principle (DIP) [40] by designing the interfaces in the design components. The design is based on abstractions provided in Appendix A. Changes in implementations in one design component do not result in side-effects in other design components.

3.1.1 Plane detection

Plane detection is a procedure in which the planar surfaces of physical objects are detected and represented as an abstraction. The representation of a plane is adheres to Definition A.2. The notion of a plane is the abstraction through which physical planar surfaces are interacted with. In particular, the vertices and centroid have well-defined positions in the world space.

The process can be represented as a linear timeline where planes are defined at arbitrary points in time. As soon as a plane is defined it is accessible as an entity as illustrated in Figure 3.2.

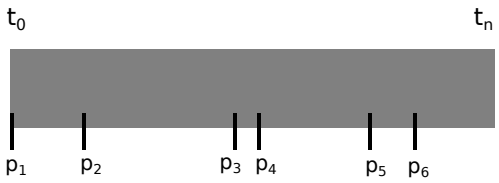


Figure 3.2: The timeline of planes being detected as time indefinitely progresses.

As the camera hovers in proximity of physical surfaces, the plane detection system improves its understanding. As time progresses, new planes are defined

and existing planes expanded. The planes are visualized on an overlay on top of the camera's view of the physical environment. Figure 3.3 shows a display with a graphical representation of a plane on it. Rendering a virtual plane is required because the user should be able to see the current state of the plane detection. Hovering the camera in uncharted areas expands the rendered plane.

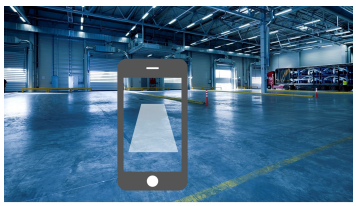


Figure 3.3: The display renders a transparent polygon stretched along the physical floor [42].

There exists an arbitrary number of planes at the same time, each of which behaves independently. The planes can expand as time progresses indefinitely. As stated in Definition A.2, a plane can subsume another plane. This occurs when the one plane extends into the area covered by another plane.

We assume a plane following Definition A.2 is defined when its vertices fit the edges and vertices of the physical object. Asserting for whether they fit or not is made by visualizing the plane relative to the physical surface on the hardware display. Because the plane detection process is a best-effort understanding of the physical environment, we do not extend the logic any further. Determining whether a plane is sufficiently defined is a qualitative decision based on its size relative to the corresponding planar physical surface.

3.1.2 Plane selection

This pipeline step is invoked whenever a plane is selected at an arbitrary point in time. The rendering of planes enables the end-user to see the plane's size and position, relative to the physical surface. When the user decides the plane rendered on the screen fits the edges of the physical surface, the plane is selected.

The selection is made by a touch gesture on the touchscreen. There are two possible situations plane selection occurs in. Either the *floor plane* or any other plane are defined. The floor plane is the plane which corresponds to the floor in the physical environment. The floor plane is the plane used to compare the difference in height to other planes. This is needed for determining the volume of virtual copies.

Determining which physical surface is the floor plane is a qualitative decision made by the user. From this, there is no formal definition of a floor plane. The floor plane needs to be horizontal. Any other planes will behave equally as the floor plane. In addition, it is possible to position nodes anywhere, relative to the floor plane.

3.1.3 Triangle construction

As claimed in Definition A.4, there exists a set of homogeneous triangles that makes up a polygon. The homogeneity is that all triangles follow the same properties specified in Definition A.3. To construct a triangle adhering to Definition A.3 using a general approach we assign values to its numerical attributes. More precisely, a triangle can be constructed using any two adjacent vertices and the centroid of a *plane*. Figure 3.4 shows a plane with five vertices and a centroid. Using the centroid and two vertices we create a triangle.

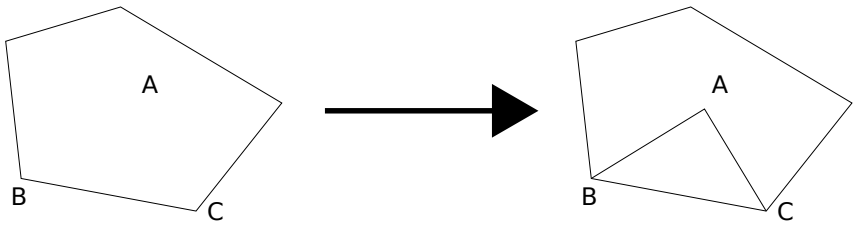


Figure 3.4: Illustration of how a triangle can be created using the polygon's vertices and its centroid. The polygon's centroid becomes one of the triangle vertex.

In the polygon to the left, A is considered a centroid. In the polygon to the right, A is both the polygon's centroid and a triangle's vertex. To begin with, the position of Vertex A is positioned at the plane's centroid. Vertex B and C are positioned on two adjacent vertices. Consequently, the triangle overlaps a subset of the plane's interior. In addition, a texture needed for rendering is supplied upon rendering a triangle.

3.1.4 Texture construction

The foundation for drawing textures on a triangles are Definitions A.1, A.2 and A.3. For each invocation, exactly one triangle is considered. The live camera preview draws the texture of its current view of the physical world, given a position and direction in world space.

To create the texture of a triangle, we make use of the live camera preview. The image contains at least a plane's entire view surface. Consequently, there exists a subset that can be used as textures to all triangles of a polygon. For

each triangle, the image needs to be cropped to extract the texture contained in the image.

At the time of capturing, it visualizes the physical surface as seen from the camera. Any movements of the camera after creation will not change the texture when the triangle has been rendered.

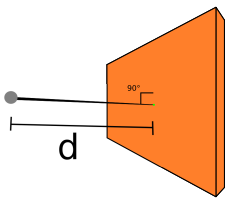
However, movements will change the texture's appearance. This is because the triangle remains stationary in the world, relative to movements of the smartphone camera. This effect mimics the visual appearance of a physical surface when an observer moves relative to it. In particular, the texture remains stationary in the same way the visual appearance of a physical surface remains stationary.

The quality of the texture depends on the angle of view relative to the surface. If the screenshot of the physical surface is captured when the camera view is perpendicular relative to the surface, the texture is not distorted. However, as the angle between the camera and the surface approaches a straight alignment relative to each other, the texture becomes distorted. The distortion originates from the keystone effect, in which there is a misalignment between the surface and the projection of an image [43]. Here, we capture an image from a misaligned surface. However, the misalignment remains the same.

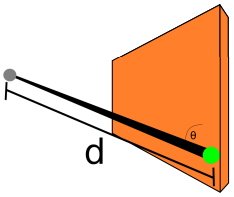
The width of the view also affects the quality. If the physical object is relatively large, the texture quality closer to the edges can decrease. This effect comes from the fact that the view of the surface becomes more distorted as the camera views it from a near-parallel angle.

The illustrations in Figure 3.5 demonstrate how the effect behaves, as seen from the camera's position relative to the surface. The black line depicts the ray coming from the camera. Here, the point at the surface seen by the user intersects with the ray. Figure 3.5a depicts a situation in which the quality is optimal with a distance of d. In this case, the portion of the surface covered by the center of the line starting at the camera lens is optimal. A perpendicular angle at which the camera views the surface is optimal.

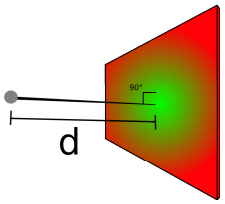
Figure 3.5b shows that distortion can occur by capturing an image from a sub-optimal angle. A sub-optimal angle of θ is an an angle whose value is significantly different from 90 degrees at some distance d. Figure 3.5c shows a Field of View (FOV) in which the image quality decreases depending on the size of the physical surface. The green area close to the straight ray coming from the view point O has better quality than the red areas further away. The distortion effect becomes greater as the surface is viewed from a more shallow angle.



(a) The point which the ray hits contains the most optimal portion of the texture as seen from the camera.



(b) θ is the angle between the surface and the ray from the camera. θ is always different from 90 degrees.



(c) The green area contains an optimal texture quality when seen from the camera, whereas the red area contains a distorted texture.

Figure 3.5: Keystone effect arising from capturing images from different angles.

This effect appears on both horizontal and vertical surfaces. As a requirement for optimal texture quality, the surface needs to be viewed at a 90 degree angle at an appropriate distance. The distance and light conditions can vary between the hardware being used.

3.1.5 Polygon construction

When a plane has been selected, it is possible to construct a polygon based on its vertices and edges. The fact that the plane is a convex polygon in its own right is fundamental to construct a distinct copy. The polygon shares the geometrical properties of the plane. From the plane we create a polygon by filling the interior of the plane with homogeneous triangles. The homogeneity originates from using Definition A.3. The triangles are encapsulated under the abstraction provided by the component. Figure 3.6 shows how a collection of triangles are fed into the interior of the plane.

3.1.6 Polygon mesh

To begin with, we acquire a point cloud. It maintains a set of points in 3D space which represents a position on any physical surface. A point in the point cloud is considered if and only if its confidence value is greater than zero. It

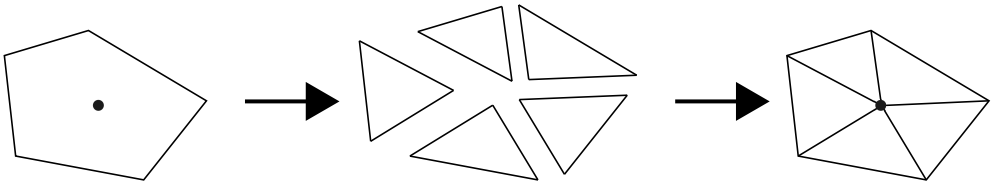


Figure 3.6: The transition from a plane into independent triangles. Together, the triangles make up a convex polygon similar to the plane.

is a realization of Definition A.7. Further, we create a polygon mesh from the point cloud.

3.1.7 Virtual copy construction

We assume a physical object of interest is to be modeled. To begin with, we allow the commodity hardware to hover in proximity of surfaces. Depending on the curvature of the surfaces, we utilize different design components. We encapsulate the different design components under the same abstraction.

Planar surfaces After setting floor plane is defined, the smartphone camera hovers in proximity of a flat physical surface.

The polygon construction pipeline steps produces a polygon that fills the plane. Along with the geometrical properties of the polygon, the texture is also applied such that the polygon is a best-effort representation of the physical surface. This procedure is applied for each physical surface of interest. The construction of virtual copies is performed by iteratively constructing polygons and then encapsulating the polygons into one coherent entity.

Now there is a set of independent polygons, each of which positioned in the world space at the same position as where they were constructed. To construct a virtual copy, each polygon is assembled such that they all behave as one entity. More precisely, all polygons should rotate, scale and move at the same time. When one polygon is altered, the changes propagate to all other polygons in the virtual copy. Transitively, the triangles in all the polygons are changed as well.

In the physical world, all object's surfaces move when at least one of them moves. The same holds for rotation of objects. The virtual copies aim to mimic the same behavior.

Curved surfaces For curved surfaces, a point cloud is required as input. The mesh consists of points in a three-dimensions space and edges between

them. Its data is encapsulated into the notion of a virtual copy. The edges and vertices are bundled into the scene. The polygon mesh's edges and vertices follow the same rotation, scaling and movements.

3.1.8 Managing virtual copies

This pipeline step enables the ability to manage the virtual copies in a physical environment. To manage a virtual copy means to move it in all three directions, scale it and rotate it. Displaying the virtual copies follows the same principle as with plane rendering. Displaying and interacting with the virtual copies is performed using the touchscreen.

3.2 Architecture

The design components defined in Section 3.1 provides data and operations on them encapsulated into different abstractions. Using these components enables us to represent models of physical objects as abstractions on commodity hardware.

We pass data between the modular design components in a particular direction. To create and manage a virtual copy depends on the data processed from other components. This approach incrementally builds a virtual copy. It aims to logically represent the appearance of a physical object.

A pipeline is well-suited for our application architecture because the surface reconstruction requires several operations. Each step requires an input, processing of data and yields output to the next step. Communication between the steps are realized by interfaces provided by the steps. This is motivated by encapsulating data and operations on the data. The order in which the pipeline steps are invoked arises naturally by looking at the services provided by them.

3.2.1 Plane reconstruction

The first pipeline step starts when the plane detection process starts. The steps are invoked at arbitrary points in time. The pipeline is illustrated in Figure 3.7, and shows hows how one virtual copy is created following the pipeline flow from left to right.

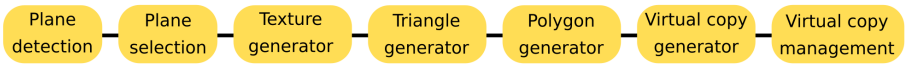


Figure 3.7: Polygon triangulation pipeline.

Once a plane has been expanded to fit a planar surface, it is selected by performing a touch gesture on the touchscreen. Then, a texture that visually represents the plane is created. From the shape of the plane, a texture is cropped to fit the triangles. When all triangles have been created using cropped parts of the screen texture they are used to envelop the interior of a polygon. When all necessary polygons have been created, a virtual copy is defined. To finalize the functionality, the virtual copy can now be managed.

It is possible to repeat any of the preceding horizontal pipeline components after invoking a component. For example, when a virtual copy has been created, any other arbitrary number of virtual copies can be constructed. The modularity of the components allows this.

3.2.2 Curved surface reconstruction

To create a virtual copy from the polygon mesh, four of the design components similar to the pipeline described in Subsection 3.2.1 are used – plane detection, plane selection, virtual copy generation and virtual copy management. Although some of their parameters are different, the principle remains. When creating a virtual copy for a polygon mesh, the points and edges in the mesh are encapsulated into the same abstraction.

Figure 3.8 shows the pipeline for creating a virtual copy from the polygon mesh. After completion of one pipeline step, it is possible to invoke any other. This is enabled by modularity of the design components.

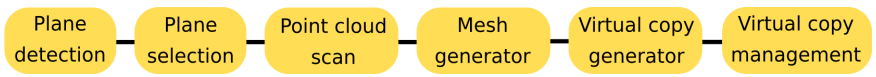


Figure 3.8: Polygon meshing pipeline.

4

Implementation

To realize Neo's architecture described in Chapter 3, an implementation is described in this chapter. We implement a mobile application for the Android platform, using Google ARCore to enable Augmented reality (AR) features on off-the-shelf commodity hardware.

We begin by outlining the implementation details of the architectural components. We use the Application Programming Interface (API) provided by Google ARCore to enable AR features and Sceneform API for rendering graphics on top of the camera's view of the world. Then, we combine Neo's software components to implement the two pipelines.

4.1 Environment

To implement the architecture on commodity hardware we use the feature-set provided by Android Operating System (OS). The framework is accessible through a Java API which offers needed functionality. For commodity hardware we rely on a smartphone with enhanced camera supporting AR. To utilize the camera's functionalities we use Google ARCore, and Sceneform API for graphics rendering on top of the live camera preview.

We chose a Huawei P20 smartphone as commodity hardware for developing the application. We also use a smartphone emulator provided by Android Studio

to ease the development process. The emulator provide a Virtual Reality (VR) space in which the application can be tested. However, we assume only a successful implementation when it runs on commodity hardware. Figure 4.1 shows the Android emulator.

Figure 4.1: Google Pixel 2 emulator provided by Android Studio.

Sceneform API is used as a rendering library. It is necessary to ensure the provided hardware is able to run Google ARCore and Sceneform by testing some of their fundamental functionalities. We test our choice of hardware and development environment by running example projects provided by the Google ARCore Software Development Kit (SDK) repository on Github [44].

4.2 3D reconstruction pipeline

The pipeline is implemented as a series of classes, each of which defines operations and data required to perform surface reconstruction. The pipeline steps employ classes and methods declared in the Android API and Google ARCore.

We define some important concepts used througout this chapter.

Definition 4.4.1 - Initialization gesture

This is a hand gesture performed when holding the smartphone in proximity of surfaces. It is needed for Google ARCore to detect and track physical planes. This gesture is necessary to perform prior to scanning planes.

Definition 4.4.2 - Plane detection spotlight

The portion of a plane being rendered is covered by the plane detection spotlight. When the smartphone hovers, the spotlight renders the planes that are in the Field of View (FOV). The plane detection spotlight can be adjusted.

Its value is measured in meters.

Definition 4.4.3 - Abstract sphere

A sphere in a three-dimensional space with a position p and a radius r.

4.2.1 Plane detection

After performing the initialization gesture the plane detection starts. It runs on a separate worker thread and renders planes on the UI thread. As the phone hovers in proximity of planes, Google ARCore recognizes them and renders them in the plane detection spotlight. Figure 4.2 shows a red-yellow texture projected onto the floor. This is the plane detection spotlight rendering the interior of a plane.

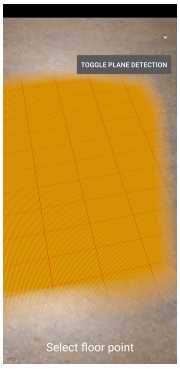


Figure 4.2: The plane detection spotlight by Google ARCore used to render planes.

4.2.2 Plane selection

Selecting a plane is done by tapping on the portion of the screen that renders it. The first plane to be tapped needs to be horizontal. This is referred to as the *floor plane*. The floor plane is used to calculate the volume of a virtual copy. All consecutive planes that are selected can have horizontal or vertical orientation.

4.2.3 Triangle

The implementation of a triangle is a realization of Definition A.3. Each triangle vertex is associated with a position, a normal and a UV coordinate. The normal

of a vertex is upward-facing relative to the plane from which the triangle is created. A UV coordinate is a 2-tuple which signifies the position on a texture rendered by a vertex. The UV coordinates maps the texture onto a triangle by enclosing the texture within the vertices. Using three vertices along with UV coordinates, the triangle's interior renders the part of the texture contained within the three vertices. The UV coordinates are each in the range 0 to 1. Formally, one triangle's texture is an element in the set of a polygon's texture. When it is applied to the triangle, it is cropped. An image of `Triangle` object being rendered on the live camera preview is seen in Figure 4.3. The description of a triangle is given in Code Listing 4.1.

Figure 4.3: A triangle being rendered on a wall with a texture that blends in with the rest of the surface.

`Node` is an abstraction in Google ARCore that provides the dynamic behavior we need. It offers functionality to achieve movements, scaling and rotations using hand gestures on the touchscreen [45, 46]. The `Node` in `Triangle` is used to connect the triangles together into a polygon. Movements, scaling and rotations are propagated into the node from its parent.

Code Listing 4.1: Traingle class

```
public class Triangle{
  static class Position {
    private float x, y, z;
  }
  private Position p1, p2, p3;
  private UV uv1, uv2, uv3;
  private Texture texture;
  private Node node;
}
```

4.2.4 Texture

Creating a texture requires multiple procedures, all of which are described below. Our motivation is the need to crop the screenshot such that it fits in a polygon.

To construct a texture we take a screenshot of the view from the camera. The screenshot represents the visual appearance of the surface. The screenshot is cropped to fit within a polygon created from a plane. In turn, all triangles inside a polygon will have a sub-set of the texture applied to the triangle.

The world space is used to represent a plane. When constructing a polygon with a texture, the plane's properties are used. The points in the world space we are interested in are a plane's vertices and the center of a plane. We want to apply the screenshot of a physical object onto a polygon created from the same plane. To do this, we need to find which screen coordinates each of the plane's vertices correspond to.

We perform a linear transformation from three-dimensional world space to two-dimensional screen space. From this, we find on-screen coordinates of the plane's vertices. By linearly transforming the world space to screen space, we are able to crop the screenshot using the vertices' positions in screen space.

Each vertex in terms of the screen space is normalized to the range 0 to 1, inclusively. Then, they are used as UV coordinates to crop the screenshot. The cropped portion of the screenshot fills the interior of the polygon. Each UV coordinate associated with a vertex which was transformed from world space is associated with the triangle's vertex. This logic is applied to all triangles in a polygon.

4.2.5 Polygon

To realize a polygon we use `Node` from Google ARCore. A summary of the class can be seen in Code Listing 4.2. As stated in Subsection 3.1.5, a convex polygon consists of a set of triangles.

Code Listing 4.2: Polygon class

```
public class Polygon {
  private Plane plane;
  private Node node;
}
```

`Plane` is an abstraction provided by Google ARCore that represents a plane in the physical world. To create a polygon that envelops a plane's interior, the `Plane`'s collection of vertices is iterated.

For each iteration, the two vertices v_i and v_{i+1} and the center of the plane are used to create a `Triangle`. The texture is cropped into individual portions when creating triangles. By adding nodes from all triangles into the polygon's set of nodes, they will behave similarly to their parent. This parent-child relationship provides the encapsulation needed. Figure 4.4 shows a polygon created from triangles enclosed in a plane. The green spheres signify the vertices of the plane. Each triangle is made up by two triangles and the white centroid in center of the plane.

Figure 4.4: A polygon being created from triangles. Each triangle are made up by two green spheres on the outer edge and the plane's white centroid.

4.2.6 Virtual copy

To create a virtual copy, one or multiple all nodes from polygons are appended to the set of virtual copy's set of children. By adding polygons' nodes to the the set, we encapsulate polygons into a virtual copy. This is similar to how triangles are encapsulated into polygons. The result is that a node in a virtual copy is the root in a tree of nodes. Transitively, the movements will propagate to the triangles as well. Each node provides movements, scaling and rotations we need. An outlining of a virtual copy is summarized in Code Listing 4.3.

Code Listing 4.3: Virtual copy class

```
public class VirtualCopy {
  private Node node;
}
```

4.2.7 Polygon meshing

To create a polygon mesh we implement it using `PointCloud` provided by Google ARCore. From the point cloud we create `MeshPoint`. An outlining can be seen in Code Listing 4.4. Each `MeshPoint` created from a point cloud consists of two values – a 3-tuple position (x, y, z) and a confidence value. The confidence value is a number which signifies the reliability of the point.

Code Listing 4.4: Mesh point

```
public class MeshPoint {
  static class Position {
    float x, y, z;
  }

  public Positon p;
  public float confidenceValue;
}
```

We implement an algorithm for creating a polygon mesh from a point cloud. The algorithm requires a collection of `Triangles` and `radius` as parameters. The radius defines the granularity of the polygon mesh.

For each triangle, the center of its circumsphere in world space is found. Let `circumSphere` be the position of the circumsphere. Then, two abstract spheres are perpendicularly positioned on opposite sides of the triangle. The radius of an abstract sphere is `r`.

If the triangle's vertices intersect with the abstract spheres and if at least one of the two abstract spheres do not contain vertices of other triangles, the triangle is valid and straight edges can be drawn between the vertices. The algorithm is summarized in pseudo-code in Code Listing 4.5.

Code Listing 4.5: Polygon mesh computation

```
public void mesh(List<Triangle> triangles, float r) {
  // valid triangles for the mesh
  ArrayList<Triangle> res = new ArrayList<Triangle>();
  for (Triangle t : triangles) {
    Position circumSphere = getCircumSphere(t);

    // center of abstract spheres
```

```
    Position centerA = getAbstractSpherePos(t, 1);
    Position centerB = getAbstractSpherePos(t, 2);

    // iterates the point cloud to test for all points
    if (noOtherPointsInSpheres(t, centerA, centerB)) {
      res.add(t);
    }
  }
}
```

4.3 Application

The mobile application consists of two Android activities. The activity objects implement the two pipelines described in Chapter 3. Using the design components outlined in Section 4.2 we build the mobile application. We use Android activities because they provide abstractions needed to run applications on the Android platform [47]. This section outlines how the pipeline steps are put together to create a working Android application.

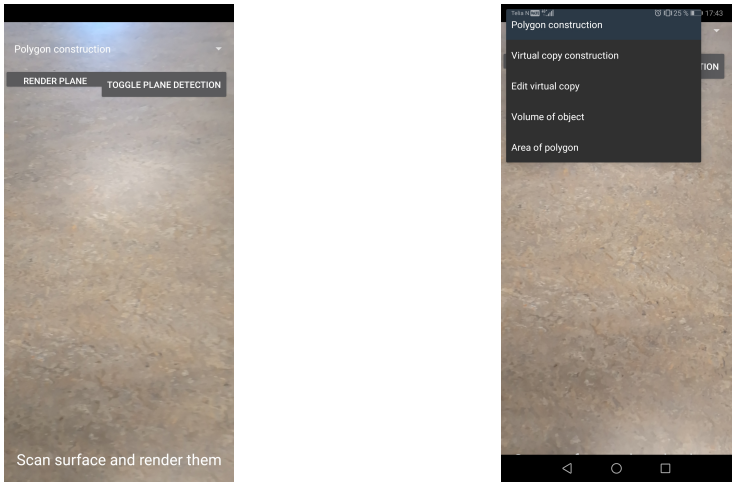
To begin with, the application assert for whether or not Google ARCore is installed. It exits with an error message if it is not. Permission for using the camera is prompted. If granting permission fails, the application exits with an error message.

4.3.1 Planar surface reconstruction

A plane of interest is scanned by hovering the smartphone in proximity of it. We capture a screenshot of the smartphone camera's FOV. Using the screenshot we create a polygon rendered on top of the physical surface. The `Polygon` facilitates `Triangle` objects to render a virtual representation of the physical surface.

After `Polygons` have been created, we create a `VirtualCopy` to encapsulate the `Polygons` into it. Finally, we are able to move, scale and rotate the `VirtualCopy` objects.

The spotlight radius is set to 100 meters. This is needed because the spotlight area must be larger than the area visible in the camera's view. This means that the outer edges of a `Plane` is rendered. This enables a precise visualization of the extent of a `Plane` relative to the edges of a physical surface.



(a) Live camera images of the physical surroundings with a simple UI.

(b) A drop-down menu when interacting with Neo.

Figure 4.5: The user interface of Neo to reconstruct physical planes.

To use Neo we set a User Interface (UI) used for interactions. After starting the UI the live camera preview and graphics rendering are enabled.

4.3.2 Polygon meshing

As the smartphone is being hovered in proximity of physical surfaces, Google ARCore is acquiring a point cloud from the physical surroundings. To render the points in the point cloud, `MeshPoints` in Code Listing 4.4 is used.

To generate a polygon mesh, the collection of mesh points is converted into all possible triangles. Using this approach, all possible triangles are considered. Each triangle consists of three vertices. Then they are passed into `mesh()` in Code Listing 4.5. We assign the granularity constant `meshGranularity` a value of 0.20 m. It is used as the radius of the abstract spheres in the meshing algorithm. This is a statically assigned variable because we limit ourselves to an experimental setting. The granularity constant signifies the minimum distance between two points needed before lines are drawn.

The output of the meshing algorithm is a polygon mesh consisting of points described by `MeshPoint` in Code Listing 4.4 and straight edges between the points. Using Sceneform API we render the edges and vertices of the mesh. Figure 4.6 shows a screenshot of the UI when rendering mesh points. The green spheres are `MeshPoints` and the orange sphere is a `Node` from Google ARCore used for drag gestures of the polygon mesh.

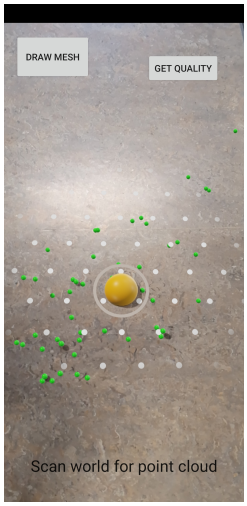


Figure 4.6: The user interface of Neo to reconstruct curved surfaces.

We have used ProGAL API [48] for solving the problem of finding the circumsphere of a triangle. ProGAL is a library for performing computational geometry with an emphasis on characterizing and representing protein structures. This has increased the Trusted Computer Base (TCB) to a slight degree. However, we claim that the usage is justifiable because it is used in the same field, namely computational geometry.

5

Evaluation

To investigate whether our thesis is able to perform surface reconstruction we designed Neo, a pipeline architecture described in Chapter 3. We implemented it as a mobile application in Chapter 4. This chapter experimentally evaluates its ability to perform surface reconstruction on commodity hardware.

Neo leverages low-level abstractions by encapsulating them into higher-level modules. By capturing immersive images using the Google ARCore framework, Neo facilitates them to create virtual copies projected onto live images of the physical world.

5.1 Experimental environment

The motivation for performing the experiments is to evaluate the architecture. There are several qualitative and quantitative experiment to conduct, each of which seeks to describe how the architecture behaves under different conditions. Along with the architecture, the tools and experimental setup stated in this chapter allow the experiments to be reproduced. The experiments can be replicated to other types of commodity hardware as well.

5.1.1 Android Platform

For development purposes and to perform experiments, a Huawei P20 was used. Huawei provides support for Android and Google ARCore. It uses a Kirin 970 CPU consisting of 4 x Cortex A73 2.36 GHz + 4 x Cortex A53 1.8 GHz cores, 4 GB RAM and 128 GB ROM. The dual-lens rear camera consists of 12 MP (RGB, f/1.8 aperture [1]) + 20 MP (Monochrome, f/1.6 aperture [1]) [51]. Figure 5.1 shows how an illustration of the phone.

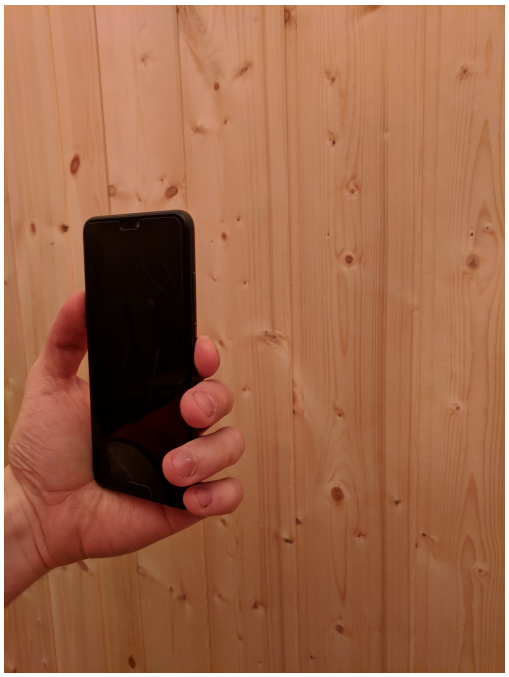


Figure 5.1: A Huawei P20 hovering in proximity of a vertical surface. The way it is held is applicable to all experiments in our evaluation.

5.1.2 Tools

A lux-meter is used to measure the illuminance emitted from a surface. Lux is a SI-derived unit of illuminance emitted from a surface when one lumen is evenly distributed over an area of one square meter [52]. We use it as quantity of measuring the amount of artificial lightning emitted from a surface. Google ARCore uses light conditions to build its understanding of the world [27].

We used a Hagner ScreenMaster to measure the illuminance of a surface. The sensitivity is in the range 0.1 lx to 200 000 lx. The accuracy is greater than

1. Aperture is a hole within a lens, through which light travels into the camera body.

±3%. For the last digit it is ±1% [53]. Figure 5.2 shows the lux-meter used to find the illuminance of a surface.

Figure 5.2: A lux-meter used to read illuminance of a surface. For the sake of completeness, we demonstrate how it reads illuminance in the image to the left. An example reading of 33.0 lx is shown.

We used a tripod to position the smartphone in a stationary position. Figure 5.3a shows the tripod with the smartphone attached to it. We used a dimmable lamp to adjust the illuminance a surface emits. Figure 5.3b depicts the light source used.

5.1.3 Surfaces

Table 5.1 lists the vertical surface used in the experiments, along with the labels used for reference. They are used in the experiments regarding initialization gesture, area calculation and volume calculation. Figure 5.4 shows the surfaces used in the experiments. The surface area is the total area of the object.

Table 5.1: Surfaces used for initialization gesture experiment.

Label	Surface	Area
A	Black plastic	123 cm x 115.5 cm
B	Wooden wall	3 m x 2.7 m
C	Styrofoam	120 cm x 120 cm
D	Red and green color sheets	100 cm x 140 cm
E	Orange tape with textures	73.5 cm x 45.5 cm

In Figure 5.5 we see the object we want to use as subject in the point cloud

(a) A tripod with a Huawei P20 smartphone attached to it.

(b) A lamp with a switch for dimming.

Figure 5.3: Experimental tools

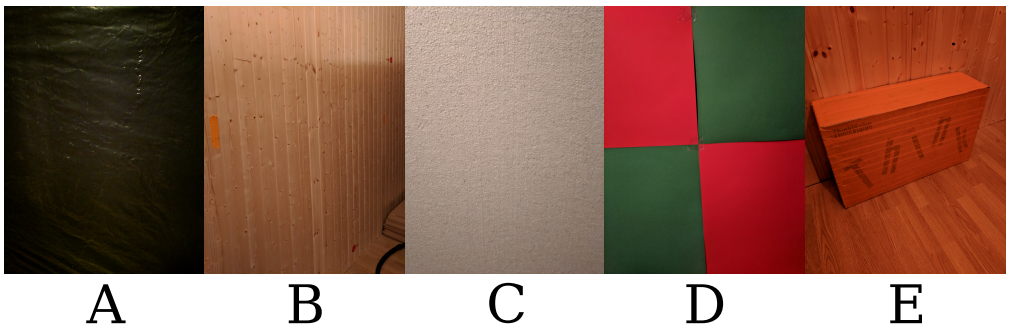


Figure 5.4: The vertical surfaces subjected for experiments.

experiment.

5.1.4 Scope/Limitation of experiments

There are many ways the design and implementation can be tested. This is based on the observation that Augmented reality (AR) is used in dynamic environments [14].

The environments of the experiments are limited to a selection of cases in which the application is expected to be used. Practicalities force us to limit the tests to vertical surfaces. This is due to the challenges associated with hovering a smartphone potentially high above a horizontal surface. Throughout the project there have been no observations in which detection of vertical planes has differed significantly from detection of horizontal planes.

We do not attempt to find the upper values of illuminance and distance due to the practical challenges that come from setting up the experiments. As the

Figure 5.5: The car tire used to create point cloud meshes.

Figure 5.6: A wooden wall used to create polygon textures.

distance from the smartphone to the surface increases, the Field of View (FOV) increases as well. Objects from the surrounding environment might obscure the tests which introduce noise into the results. Finding the upper limit of illuminance is also challenging because it is difficult to reliably block solar rays. The lamps we used as lightning sources did not provide an illuminance beyond the capabilities of Google ARCore.

Although the fluctuations in the environmental conditions can be significant, we try to the greatest extent to limit them. Although AR is used in dynamic environments like construction sites [14], we limit them nonetheless. In experiments where it is feasible we use a tripod to keep the environmental conditions constant.

5.2 Experiments

This section describes the experiments and their results. The experiments are conducted separately to ensure reliable results. For each experiment, its

setup is explained followed by the results and conclusion we can draw from them.

To evaluate whether Neo can be used to perform surface reconstruction using commodity hardware, we seek to answer a series of questions in four categories.

Environmental Conditions We need to measure the required conditions for Google ARCore to detect planes, to retain position coherence and to find point clouds. The conditions are determined by illuminance, angle and distance between the smartphone and the surface texture. Neo and Google ARCore should overcome challenging light conditions. Construction sites are examples of dynamic places in which AR technology is used [14]. In our context, we want to see the light conditions undergo significant changes which can be expected at construction sites.

The results are used as parameters in the other experiments to eliminate fluctuations that could occur from poor environmental conditions. We pass the parameters to the experiments because Neo is dependent on Google ARCore's ability to understand the environment. To create virtual copies, planes need to be detected. As for point cloud meshes, point clouds need to be created. Specifically, we measure the completion time used to acquire references to `PointCloud` objects and `Plane` objects because they represent Google ARCore's understanding of the physical environment. We arrive at the following questions regarding the conditions:

Question 1. What are the required conditions for the initialization gesture to complete?
Question 2. What are the required conditions to achieve position coherence of virtual objects?
Question 3. What are the required conditions for acquiring a point cloud?

Geometrical properties We wish to find the accuracy of the measurements provided by Neo, given a surface texture and illuminance. By comparing Neo's result to the real values, we see the viability to measure objects' geometrical properties. We pose the following questions:
Question 4. To which degree is it possible to measure the area of planar surfaces?
Question 5. How accurate is the architecture to measure volumes of virtual objects?

Texture quality We look at the texture quality to visually confirm Neo's ability to create realistic textures. We want to visually inspect if Neo can create

textures from different angles. From this, we get the following:
Question 6. What do the textures look like when created at different angles?

Point cloud mesh quality We want to see if Neo is able to create polygon meshes under different lightning conditions, given a physical object consisting of a curved surface. Despite that the point cloud detection feature provided by Google ARCore is a debugging feature at the time of writing [30], we still want to evaluate its quality. Regardless of the size of the point cloud, all points still represent a presence of a surface. We get the following question to answer:
Question 7. To which extent is the architecture able to create polygon meshes from curved surfaces?

5.2.1 Initialization gesture

To answer **question 1**, we seek to find the optimal environmental conditions for the mandatory initialization gesture. The experimental procedure is used to measure the minimum required values for distance and illuminance. These environmental conditions are used to measure the shortest execution time.

Setup The surfaces we scan are surface **A** through **E** in Figure 5.4. We use the light source and lux-meter to adjust and measure illuminance of the surfaces.

Experiment procedure We measure the minimum required illuminance and distance by hovering the smartphone close to the surface. The illuminance is initially pitch dark. The smartphone is gradually hovered further away from the surface while increasing the illuminance until there is a reference to a `Plane`. We follow the gesture movements illustrated in Figure 5.7. The position of the smartphone is perpendicular to the surface. We measure the distance using a folding rule. We also measure the illuminance using the lux-meter at that distance.

The minimum required values for illuminance and distance are derived from the experimental procedure. Using these conditions we measure the time Google ARCore use to detect a Plane.

Results Table 5.2 shows the results of the experiments. The columns for distance and illuminance show the minimum required values to complete the initialization gesture in the shortest possible time.

From these results there are several things worth noting. First, scanning surface

Figure 5.7: The initialization gesture provided by Google ARCore.

Table 5.2: Minimum required distance and illuminance for the initialization gesture to complete in the shortest amount of time.

Label	Minimum distance	Minimum lux	Completion time
A	49.0 cm	25.6 lx	8.26 sec
B	64.5 cm	3.8 lx	3.09 sec
C	71.0 cm	4.6 lx	4.21 sec
D	217.0 cm	6.9 lx	∞
E	29.0 cm	13.1 lx	5.18 sec

B led to the shortest completion time, possibly due to the surface's contrasts provided by the vertical lines between the wood boards. In addition, this surface required the lowest illuminance at 3.8 lx. However, this comes at the cost of the minimum distance of 64.5 cm. This is significantly greater than the minimum distance required when using surface **E**.

As is the case with surface **B**, surface **E** has patterns in it. This enables the procedure to complete within a reasonable amount of time. However, the required illuminance of surface **E** is somewhat stronger than the required illuminance of **B**. This might be due to the fact that the wooden wall has narrow gaps between the wood boards. The gaps provide more contrast to the surface to make the initialization gesture complete faster.

Surface **A**, the black plastic bag, required the longest completion time and illuminance. This is possible due to the black-body phenomenon in which the surface absorbs visible light [54]. However, as the light source kept getting brighter, more light was reflected from it. These results indicate a limitation

with respect to which surface textures Google ARCore can scan. When looking at the completion time and illuminance, an interesting observation is that the distance required is the second lowest.

The completion time and illuminance required when scanning surface **C**, the styrofoam plates, are among the lowest. However, the distance is relatively high compared to the other surfaces. To complete the gesture, physical surroundings in the camera's FOV was needed. The styrofoam surface contains relatively few patterns which might be the cause of this.

Another observation worth discussing is the scan of surface **D**. Google ARCore was not able to complete the gesture without the wooden floor in its FOV. This is depicted in Figure 5.8. In this situation, we also observe a limitation of Google ARCore's ability to interpret flat surfaces correctly. The plane detection spotlight is horizontally projected onto surface **D**. The wooden floor might be a reason as to why Google ARCore interprets the vertical surface to be horizontal.

It seems to be the mix of green and red color that obscures Google ARCore's correct understanding of the physical surface. Both the initialization gesture and interpretation of the surface **D** suffer from this.

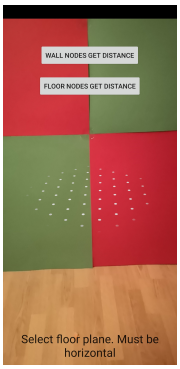


Figure 5.8: The vertically aligned paper sheets and the plane detection spotlight misinterpreting the surface to be horizontal. The wooden surface below surface **D** is a horizontal floor.

To conclude, it seems to be a trade-off between distance, illuminance and completion time required for the procedure to successfully complete. The factor that has the most impact on these environmental factors is the surface being scanned. From what we have seen, a surface's color and texture highly determine Google ARCore's ability to understand flat surfaces.

It appears to be a connection between the amount of contrasts on the surface and Google ARCore's ability to complete the procedure. This is supported

by the experiment using surface **C** and **D** – the styrofoam plates and the color sheets. The ability to complete the initialization gesture is dependent on surface textures, their sizes, distance to them and illuminance. In some cases, the surrounding environment has a significant impact.

5.2.2 Position coherence

In **Question 2** we focus on the required conditions for Google ARCore to retain coherent positions of virtual objects in the world space. We wish to measure the minimum required illuminance for rendering to occur. At the same time, we visually observe the ability to retain positions across periods of low illuminance.

Setup We position two physical stripes on surface **B** at a distance of 2 m between them. Then, we position one virtual sphere on each of the stripes. The visual appearances of the spheres relative to the stripes are used to ensure the accuracy of the positioning. Figure 5.9a shows the 2 m stretch between the tapes. The stretch starts at the right-most edge of the stripe to the left and stops at the left-most edge on the stripe to the right.

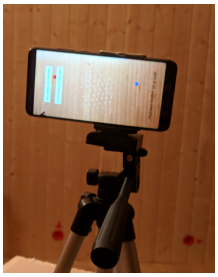
The initial illuminance must be above 3.8 lx, a value derived from Table 5.2. For this experiment we choose 9.3 lx. The smartphone is positioned in a tripod at a distance of 2.3 m from the surface with both spheres in its FOV. Figure 5.9b shows the smartphone in the tripod.

Experimental procedure With the smartphone in the tripod at a distance of 2.3 m from the wall and the virtual spheres anchored to the wall, Neo calculates the Euclidean distance between them. Then, the lightning source is dimmed down until they are no longer rendered. The illuminance emitted from the surface is measured at 2.3 m from the wall. The illuminance is increased until the spheres are rendered again. Then, the illuminance is measured again at 2.3 m from the wall. Finally, the Euclidean distance is calculated and their positions relative to each other are visualized.

Results When the spheres stopped being rendered, the illuminance was 0.1 lx. When they were rendered again the illuminance was 5.2 lx.

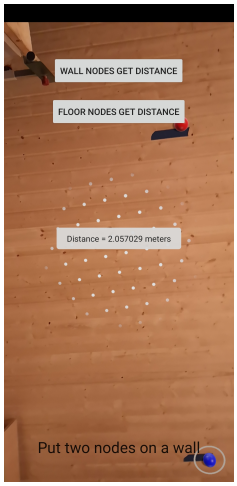
We compare Figure 5.10a and Figure 5.10b. Before dimming the light, the distance between the spheres is 2.0249 m. After dimming the light, the distance is 2.0570 m. As the pictures show, Google ARCore is not quite able to retain the same position on the wooden wall across significant changes in lightning conditions. The spheres have drifted slightly apart from their initial positions.

(a) Close-up images of two spheres positioned two meters apart from each other.

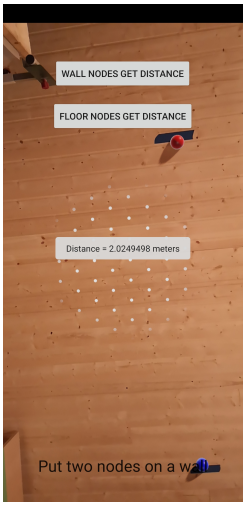


(b) The Huawei P20 smartphone in the tripod while rendering the spheres.

Figure 5.9: Virtual spheres used in experiments.



(a) The spheres positioned on the wall before dimming the light source.



(b) The spheres positioned on the wall after dimming the light source.

Figure 5.10: Screenshots of the camera view as seen from the tripod.

The numerical distance between them also changed. The reason for this might be the loss of visual features on which Google ARCore depends [27]. Any obstruction in tracking can cause loss in its ability to retain coherency in positioning of virtual objects. On the other hand, as the spheres visually drifted apart from each other, the distance increased too.

5.2.3 Point cloud completion time

To answer **Question 3** we setup the experiment to measure the shortest completion time to find a point cloud.

Table 5.3: Minimum require distance and illuminance required to complete to find a point cloud scan in the shortest amount of time.

ID	Test object	Minimum distance	Minimum lux	Response time
1	Car tire	56 cm	56 lx	2.04 sec

Setup We use the car tire in Figure 5.5 because provides contrasts on its surface. We use the wooden texture seen in surface **B** as background. The floor has a similar wooden texture.

Experimental procedure We measure the minimum required illuminance and distance between the smartphone and the car tire. We position the smartphone close to the surface at a 45 degree angle in a pitch dark environment. The light source is dimmed upwards while hovering the smartphone gradually further away at a 45 degree incline. This is done until a reference to a `PointCloud` is found. Using environmental conditions measured in this procedure we measure shortest execution time Google ARCore use to find a `PointCloud` reference.

Result The results in Table 5.3 shows that acquiring a `PointCloud` requires less illuminance than finding a `Plane` reference, as seen in Table 5.2. From this, it appears that Google ARCore's understanding of curved surface and flat surfaces do not differ significantly from each other. Although the surfaces are different, it is worth pointing out this observation. A potential source of inaccuracy is that the surrounding floor was in the camera's FOV.

5.2.4 Area calculation

In this experiment we answer **Question 4** by measuring the area of a flat surface, given a surface texture and illuminance. We compare the result to the actual area of the surface.

Setup We use surface **E** in Figure 5.4. We set the surface's illuminance to 30.1 lx to remove the difficulties in performing plane detection in a dimmed environment. Intuitively, this seems like a appropriate value above 13.1 lx derived from Table 5.2. The wooden wall surface is used as background to create an environment with evenly distributed illumination. The real value of the surface's area is measured by hand. Its value is $0.332\,15\,\mathrm{m}^2$.

Experimental procedure After completing the initialization gesture we hover the smartphone in proximity of the surface. Figure 5.11a shows how the plane is expanding. The red-yellow texture of the surface is the plane detection

(a) The yellow texture drawn on the surface is the plane detection spotlight rendering a `Plane`'s extents.

(b) A polygon with texture rendered on top of the surface from which it is created.

Figure 5.11: Setup of area calculation experiment.

spotlight covering the interior of a plane. When the surface has been scanned Neo creates a polygon, as depicted in Figure 5.11b.

Results After positioning the smartphone in the tripod, as seen in Figure 5.12, there were relatively small fluctuations in the area measurements. There were made three readings from the same position.

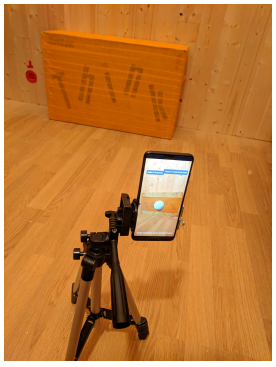


Figure 5.12: The smartphone stationary in the tripod while measuring the area of a surface.

As seen in Table 5.4, there are small fluctuations in the area calculated, even when the smartphone remains stationary in the tripod and the illuminance is constant. This indicates that Google ARCore's numeric values fluctuate close to the real value. As seen here, this is the case for Euclidean distance. The foundation for measuring distance is the concept of `Nodes` provided by Google

Table 5.4: Measuring the area of the polygon. The percentage signifies the increase from the actual value.

Acutal area	Area calculated	Percentage increase
0.332 15 m^2	0.353 763 22 m^2	6.109515850 %
0.332 15 m^2	0.353 763 3 m^2	6.109537083 %
0.332 15 m^2	0.353 763 5 m^2	6.109590164 %

ARCore. Observing the fluctuations is interesting because the environmental conditions on which Google ARCore depends are constant. For similar experiments, the measured value might settle close to the actual value after some time when Google ARCore has gained a greater understanding.

An observation is that the polygon does not fit the rectangular shape of the physical surface. This is due to the lack of Google ARCore's understanding of the physical surface. This is a somewhat inaccurate measurement, mostly due to the gap between the edges of the `Plane` and the edges of the surface. An interesting observation is that the calculated area is greater than the real value. This means that the plane provided by Google ARCore extends beyond the edges of the surface. The origin of this problem is that Google ARCore can interpret other surrounding surfaces as a part of the surface we initially scan.

5.2.5 Volume calculation

To answer **Question 5** we measure the volume of a physical object. The calculated volume provided by Neo is compared to the actual value of the object's volume.

Setup To calculate the volume of a virtual copy, we use the cardboard box depicted in Figure 5.4 as surface **E**. For this experiment we consider the entire object. The real value to which we compare the result is 0.068 678 4 m^3.

Experimental procedure After completing the initialization gesture, we find the floor plane on which the cardboard box stands is set. Then, the upper-most horizontal plane of the cardboard box is scanned. The result form scanning the plane is depicted in Figure 5.13.

Result Neo calculates the volume to be 0.073 820 08 m^3. This is a 7.48 % increase from the actual value. As seen on Figure 5.13, the polygon rendered on top of the horizontal surface does not quite align to the edge of the surface. The inaccuracy of the volume probably stems from that Google ARCore misin-

Figure 5.13: The uppermost horizontal polygon of the object. In addition, the volume in cubic meters is displayed.

terprets the flat horizontal surface. This is an interesting finding, and is similar to the observation made in the previous experiment regarding area calculation. In both experiments, there is an increase in calculated value, compared to the actual value. Figure 5.13 shows that the edges of the polygon extends beyond the edges of the cardboard box. This misalignment is the most likely source of the percentage increase.

5.2.6 Texture quality

To answer **Question 6** we are interested in Neo's ability to create realistic textures of physical surfaces.

Setup We use surface **B** and surface in Figure 5.6 to create a polygon. We consider only parts of the surface to focus only the visual effects. We do not need to consider the entire surface.

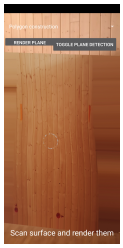
Experimental procedure After completing the initialization gesture we point the smartphone towards a surface. We hover the smartphone in proximity of the surface to trigger Google ARCore to detect it. For each image capture we create vertically aligned polygons from different angles. We inspect them

to qualitatively determine their appearance.

Results The results show the visual results from dynamically creating textures. We see that different visual effects occur when capturing the images from different angles.

(a) A polygon created from the wooden wall at a distance of 2.3 meters. The angle is perpendicular.

(b) A polygon created from the wooden wall at a non-perpendicular angle.



(c) A polygon created from the wooden wall where the plane's extends is outside the smartphone camera's field of view.

(d) The texture of a polygon whose texture consists of wooden boards. Compared to the wooden boards on the wall, they do not align correctly.

Figure 5.14: Polygons created from different angles and distances. Their visual appearances show different visual effects.

The images show different visual effects under different conditions. Figure 5.14a displays a near-optimal texture quality compared to surface **E**. Figure 5.14b shows the keystone effect being triggered from capturing the screenshot at a non-perpendicular angle. Here, the edges of the triangles in the polygon's interior are visible. This implies that there is a shortcoming in Neo's ability to perform a successful texture construction. Compared to Figure 5.14a, it differs largely in appearance. Despite the distortion, there is a clear resemblance of the real surface.

Figure 5.14c displays a case in which the texture closer to the edge of the smartphone's display lack the correct pixels. Although the assumption that the camera's FOV contains the entire plane is violated, it is interesting to see the

immediate effects when it does not hold. The portions of the texture closer to the vertical edges of the screen are distorted because they lack the pixels that would normally be contained in the camera's FOV. Here, the smartphone camera is moved closer to the surface after expanding the Plane.

Figure 5.14d displays the case where the keystone effect comes into play. the polygon is created at a distance of 3 m. The wooden boards depicted on the polygon's texture do not align with the wooden boards on the wall. This effect becomes stronger on the texture further away from the center of the camera's FOV.

5.2.7 Point cloud

To answer **Question 7** we measure the quality of a point cloud. First, we formally define its quality. By taking into consideration the number of points and the sum of their confidence values we arrive at the following formula:

$$f(x, y) = -\log \frac{1}{x} \cdot y \tag{5.1}$$

where

$$y = \sum_{i=1}^{x} \alpha_i^2 \tag{5.2}$$

Here, $x, y \geq 1$ and α_i is the confidence value for the point p_i.

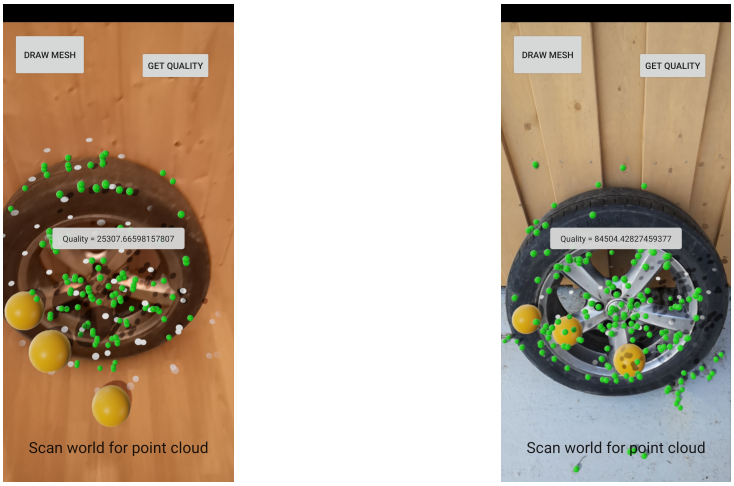
Setup For this experiment we scan the car tire depicted in Figure 5.5 in two scenarios. We do not render the edges between the points to generate a polygon mesh to better visualize the distribution of points. Regardless of enabling mesh rendering, the numeric results remain the same.

Experimental procedure First, we scan the object indoors with wooden texture in the background. Then we scan it outdoors with gray concrete and wooden panels as texture background. The illuminance is 43.3 lx, a reasonable value more than 2.04 lx derived from Table 5.3.

Result The results are listed in Table 5.5. Notice the difference in illuminance between test 1 and 2. From the difference in quality, it is reasonable to claim that illuminance can affect the quality of the point cloud. An enforcement of

Table 5.5: Required minimum values for distance and illuminance to detect acquire a point cloud.

ID	Lux	Quality
1	43.3 lx	25307.66
2	4590 lx	84504.42



(a) A point cloud from a car tire captured indoors with 43.3 lx as seen at a distance of ~50 cm.

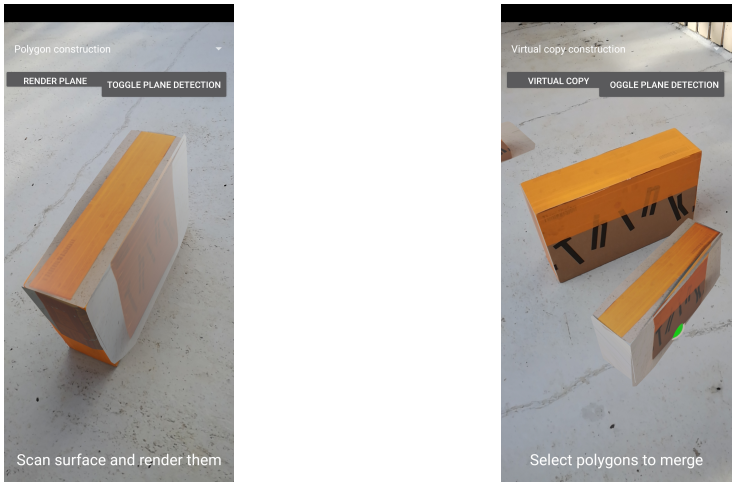
(b) A point cloud from the same car tire captured outdoors with 4590 lx as seen at a distance of ~50 cm.

Figure 5.15: Screenshots of the point clouds created from scanning the car tire.

this claim comes from visually comparing Figure 5.15a and Figure 5.15b. A source of noise in the result is seen in Figure 5.15b – there exist points not attached to the car tire.

5.3 Qualitatively assessment of Neo

We have seen promising results in our experimental evaluation. After finding affirmative answers to the questions regarding environmental conditions, accuracy and texture quality, we want to see if our architecture and implementation meet our expectations. This section look at qualitative assessments of Neo by capturing two virtual copies, each of which with different curvatures.



(a) A virtual copy that envelops the cardboard box.

(b) A virtual copy placed beside the cardboard box. Notice its orientation and size relative to the cardboard box.

Figure 5.16: Creating a virtual copy from a cardboard box.

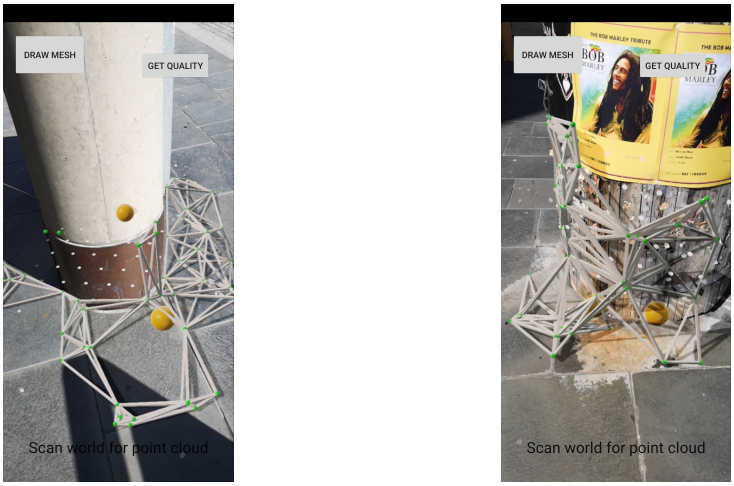
5.3.1 Planar surface

To confirm that creating a virtual copy with planar surfaces is possible, we scan all five visible surfaces of the cardboard box depicted in Figure 5.4. Figure 5.16a shows a virtual copy that envelops the cardboard box. Figure 5.16b shows the virtual copy placed beside the cardboard box. Notice that it has been scaled and rotated. The textures of the polygons contain parts of the concrete floor. This demonstrates shortcoming in the texture creation.

5.3.2 Curved surface

To complete our evaluation, we demonstrate Neo's ability to create a polygon mesh from a point cloud. After scanning an object we arrive at the two illustrations in Figure 5.17. The yellow sphere is rendered to make the polygon mesh manageable. In Figure 5.17a we see the mesh mostly extending to the ground. An important observation is the straight lines extending vertically from the ground. In Figure 5.17b we see the mesh following the curvature of the wooden column.

Some lines are drawn stretching from the ground to the wooden column that do not align with a surface. This is due to the granularity parameter. It needs to be adjusted to a lower value for the mesh to be more realistic.



(a) A polygon mesh created from acquiring a point cloud surrounding a concrete column.

(b) A polygon mesh created from acquiring a point cloud surrounding a wooden column.

Figure 5.17: Creating meshes from two columns.

5.4 Summary

Throughout our evaluation we have used Neo and Google ARCore in different environmental conditions. We began by evaluating the required conditions to track planes, the ability to retain coherency of tracking and to acquire a point cloud. Here, we measured the minimum required distance and illuminance. We saw that the environmental conditions were dependent on the surface texture. We also saw the need to have the surrounding environment in the camera's FOV.

To measure minimum required distance and illuminance we devised a set of experimental procedures. These parameters were used in the remaining experiments to overcome challenging environmental conditions.

We used Neo to measure area and volumes of virtual copies. In all tests we a slightly higher value in the computed result when comparing to the actual value. The shortcomings came from Google ARCore's ability to misinterpret the physical surfaces. Consequently, the planes represented by Google ARCore were expanded beyond the boundaries of the physical surfaces and Neo computed a result unequal to the actual value.

Neo created textures of polygons by capturing the camera's view of the physical surfaces. We saw interesting effects arising from capturing the images from different angles and distances.

We developed a means of measuring a point cloud in terms of quality. Experiments on a point cloud concluded that illuminance impacts the results to a significant degree. Lastly, we saw the architecture of Neo in a practical scenario by scanning two physical objects and displaying their virtual copy.

The methodology and equipment have provided useful results. We have seen both abilities and limitations of Neo and Google ARCore. As stated earlier, there are many ways in which our architecture can be evaluated. We have chosen a sub-set of interesting ways to test it. Despite the limitations, Neo and Google ARCore have shown promising results. We saw best-practices of using Google ARCore. We observed that high-contrast surfaces with a certain amount of illuminance give the best environmental conditions.

We set out to investigate our thesis. Our problem statement is to investigate the ability to perform perform surface reconstruction on commodity hardware. To draw a conclusion, we see from the evaluation that Neo is able to create virtual models of physical objects. In our experimental evaluation we have seen under which conditions it is feasible for Neo to accomplish this.

6 Conclusion

This chapter will conclude our thesis, summarize our contributions and results and relate to future work. We reiterate our thesis statement:

This thesis will design, implement and evaluate surface reconstruction capability on commodity hardware.

6.1 Achievements

In this thesis we designed and implemented Neo, a pipeline architecture for surface reconstruction using commodity hardware. We began by introducing a set of abstractions which provide us with a useful interpretation of lower-level mechanisms and concepts. Using these abstractions we created design components, each of which is separate entities. We chained the design components together to create two separate pipelines. These pipelines perform surface reconstructions on different surfaces. We implemented Neo on the Android platform using Google ARCore to enable Augmented reality (AR) features.

6.2 Findings

We saw that a pipeline architecture was suitable to approach the thesis statement. We assigned each architectural component a well-defined set of tasks with modularity in mind. From a high-level point of view we saw that a stepwise approach to the thesis statement.

From the evaluation in Chapter 5 we observed the ability of Neo to perform surface scans and visualizing them according to the characteristics of AR. Rendering the *virtual copies* bears some resemblance to the related work. Both qualitative and quantitative testing metrics were used. We also saw some limitations which pave the way for opportunities and future work.

The evaluation procedures we devised showed the environmental conditions used in the experiments. These procedures were useful to find a best-practice usage of Neo and Google ARCore.

6.3 Future work

Neo has potential to be further developed. This section highlights the aspects that can be subject for future work.

Surface reconstruction accuracy As seen in Chapter 5, the polygons are not perfectly aligned to the surface from which it is created. Notably, they extend beyond the extents of the edge of a planar surface. A suggestion for a future project is to improve Google ARCore's understanding of the physical environment. As an alternative, Neo can be improved by selecting a subset of the planes provided by Google ARCore. As long as the planes are aligned with the extends of a surface, selecting the sub-set is achievable.

Virtual copy storage The virtual copies can be stored in a repository along with metadata to be used for modeling purposes. The data can be shared among several participants for collaboration. The virtual copies can be fetched to be modeled on a desktop computer, using Virtual Reality (VR) or other types of commodity hardware supporting AR.

Texture quality We saw in Chapter 5 that the textures were not completely optimal. We suggest that efforts should be taken to improve their quality. Similar to [4], we suggest that an approach based on stitching would be beneficial. Multiple images of a surface stitched together would improve the quality because they can be taken at a near-perpendicular angle. This approach

could address the issues related to keystone effect.

Polygon mesh The polygon meshes we have created have only consisted of the edges and vertices. We suggest to extend the polygon meshes to include a solid surface as well.

BIM integration We see potential to integrate Neo into an existing Building Information Model (BIM) application. It is possible to overlay the virtual copies of physical objects with a BIM and comparison of the information about the objects. The integration could simplify inspection routines on construction projects.

Scalability Our implementation of Neo aims at processing few surfaces with small area. We propose to scale the processing capabilities to handle larger surfaces and areas. This approach could involve storage and offloading to external computing units.

A

Virtual space abstractions

This chapter details important Augmented reality (AR) abstractions used throughout the thesis. The abstractions formally represent high-level concepts, each of which maintains a set of properties. We remove the lower-level details by considering them as coherent entities. Some concepts contains properties which are made up of lower-level concepts.

A.1 World space

The foundation for creating virtual copies is based on observations of physical objects. Formally, physical objects and abstractions representing them occur in the **world space**. The world space is a Euclidean $\mathbb{R}^3$ vector space. Interactions with the abstractions representing the physical world occur here. The world space is a vector space in which copies of physical objects have the potential to be created and manipulated. More precisely, copies of the physical objects are created by explicitly describing them using the abstractions provided.

Furthermore, we define a point of origin in the vector space $world\_origin = (0, 0, 0)$. Its position relative to physical objects is arbitrarily chosen. This means that several positions in the physical world are candidates for being marked as the origin. We assume that the position of *world_origin* is automatically chosen and will never change.

A.2 Nodes

We define the concept of a **node**. It represents a point of reference in a Euclidean vector space. A node N has certain assumptions and properties associated with it. Formally, we define N as follows:

Definition A.A.1 - Node

1. N can have either zero or one parent. Such a parent *N.parent* has the same properties as N. By default, *N.parent* is undefined.

2. N has a 3-tuple position $N.world\_position \in \mathbb{R}^3$. The position is relative to *world_origin*.

3. N has a $\mathbb{R}^3$ Euclidean vector space. We refer to this space as the Local Vector Space (LVS) of N. The 3-tuple origin *N.lvs.origin* of this local vector space is positioned at *N.world_position*. The value of *N.lvs.origin* is always $(0, 0, 0)$.

4. N has a 3-tuple position $N.local\_position \in \mathbb{R}^3$. This is a position in the *N.parent.lvs* vector space. *N.local_position* is relative to *N.parent.lvs.origin*.

5. N has a world scaling factor $N.world\_scaling\_factor \in \mathbb{R}$. The default value is 0.

6. N has a local scaling factor $N.local\_scaling\_factor \in \mathbb{R}$. The default value is 0.

7. N has a rotation quaternion *N.world_rotation*. It is defined as the 4-tuple (x, y, z, w). Its default value is $(0, 0, 0, 0)$.

8. N has a rotation quaternion *N.local_rotation*. It is defined as the 4-tuple (x, y, z, w). The rotation is relative to the rotation quaternion of *N.parent*. If no such parent exists the rotation quaternion defaults to the value of *N.world_rotation*. It has the default value of $(0, 0, 0, 0)$.

9. N has a set of child nodes $N.children = \{N_1, N_2, \ldots, N_n\}$ where $n \geq 0$. *N.children* can be an empty set.

10. Assuming $N.parent$ is defined, N will follow the movements, rotation and scaling of $N.parent$.

A.3 Plane

In addition to nodes we need to detect surfaces of physical objects. Given any physical object of interest, we assume its physical surfaces together make up the object. Although the inner structures, if any, are a part of the physical object they are omitted. The material and density are also ignored. Although the surfaces of the physical object can extend arbitrarily in any direction, it is allowed to partially or fully omit some surfaces. It is possible to detect the entire physical object or only smaller sections of it. It is assumed that physical surfaces are limited to be either horizontal or vertical. We define a **plane** as a two-dimensional planar surface object with an arbitrary number of vertices and straight edges. It is a logical representation of a surface of a physical object. More formally, we define a plane P to have the following properties:

Definition A.A.2 - Plane

1. P is an approximate logical representation of a flat physical surface.
2. P is either horizontal or vertical relative to the physical world's upward direction.
3. The area of P can only grow in size; it can never shrink.
4. P has a 3-tuple position in world space $center\_point \in \mathbb{R}^3$.
5. P is co-planar because all its vertices are contained within it.
6. P has m edges and m vertices where $m \geq 3$.
7. The plane P_i can be subsumed by P_j.
8. P has a 3-tuple coordinate $P.center\_coordinate \in \mathbb{R}^3$ which is relative to $world\_origin$.
9. P has a model matrix $matrix\_model$ given below.

$$model\_matrix = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

The scalars in the matrix are provided by the underlying plane detection system.

10. P has a set of coordinates $P.local\_coordinates = \{x_1, z_1, x_2, z_2 \ldots x_m, z_m\}$ where $m \geq 3$ which defines the vertices of the plane. The elements in $P.local\_coordinates$ are relative to $P.center\_point$ such that the y component is always zero. Implicitly, they are omitted from the set.

11. P has a set of coordinates $P.world\_coordinates = \{x_1, y_1, z_1, x_2, y_2, z_2 \ldots x_m, y_m, z_m\}$ where $m \geq 3$. The elements in $P.world\_coordinates$ are transformed from $P.local\_coordinates$.

12. Given any 2-tuple $(x_j, z_j) \in P.local\_coordinates$, a 3-tuple $(x_i, y_i, z_i) \in P.world\_coordinates$ is given by the following linear transformation:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x_j \\ 0 \\ z_j \\ 1 \end{bmatrix}$$

    where $i = j$ and $i, j \leq m$.

13. All elements in the sets $P.local\_coordinates$ and $P.world\_coordinates$ can be changed at any moment. Due to the linear transformation, any changes in j$-th$ 2-tuple in $P.local\_coordinates$ imply that i-th 3-tuple in $P.world\_coordinates$ is subject for change.

14. The vertices of P will never move towards $P.center\_point$. Implicitly, the area of P can only increase.

15. P is not self-intersecting and always convex.

Figure A.1 shows how a plane can look like. Notice that the plane is a convex

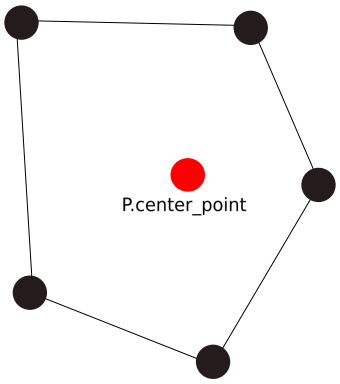


Figure A.1: An example of a convex plane with five vertices.

polygon. A plane is not limited to a fixed number of vertices, although this particular plane contains five edges and five vertices.

A.4 Triangle

The cardinality of $P_i.world\_coordinates$ and $P_i.local\_coordinates$ can be larger than 9 – the number of coordinates needed to construct one single triangle. This implies that it is possible to construct a polygon that is not limited to a triangular shape. This property allows planes with an arbitrary number of vertices and edges. We also observe that the edges between the vertices are not curved; they are straight lines. For a plane that follows Definition A.2 it is possible to construct m triangles that together make up a convex polygon. A definition of a triangle T below is given below.

Definition A.A.3 - Triangle

1. It is classified as either an equilateral, isosceles or scalene triangle.
2. The three vertices are denoted $T.a$, $T.b$ and $T.c$.
3. There exists exactly one straight line between any two vertices.
4. There exists a node $T.node$ at vertex $T.a$.
5. Each vertex has a 3-tuple normal $normal \in \mathbb{R}^3$ and a 3-tuple position $position \in \mathbb{R}^3$. The position is relative to the origin in $T.node.lvs$.
6. Each vertex has a 2-tuple UV-coordinate pair $uv \in [0, 1]$. The UV-coordinate describes the texture mapping from a *bitmap texture* to a triangle.
7. T has a texture associated used for rendering.

Figure A.2 shows the different triangles. Each triangle is a valid element in the interior of a Plane.

A.5 Polygon

The foundation for creating polygons is the notion of planes that follows Definition A.2. P is a convex polygon in its own right. Generally, polygon is an

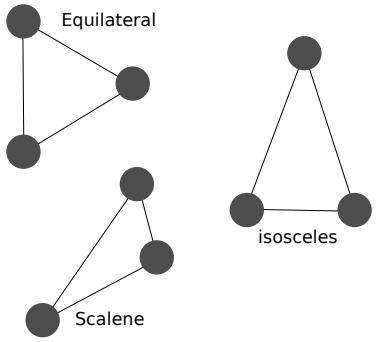


Figure A.2: Examples of triangles we consider, each in their own category.

object that is a independent copy of a plane under Definition A.2. Logically, they are two different objects. However, they share the fact that they represent physical surfaces. Because all triangles in a polygon follow Definition A.3, the polygons are homogeneous themselves. A way of interpreting a polygon is by considering as a set of such triangles. However, the abstraction provided by a *polygon* makes it possible to consider the set of triangles as one entity.

By using the Definition A.1 and Definition A.3 a definition of a polygon G is given below:

Definition A.A.4 - Polygon

1. G has m edges and vertices.
2. G is convex and not self-intersecting.
3. G has a set of triangles $G.triangles = \{t_1, t_2, t_3, \cdots, t_m\}$ where $m \geq 1$.
4. G has a node *G.center_node* positioned at the center of itself.
5. G has a volume $G.area \in \mathbb{R}$.

By extending Figure A.1, 5 lines stretching from $P.center\_point$ have been added. Similar to Figure A.1, it is still a polygon. However, this image shows how the edges between vertex a and b of any triangle make up the interior of the polygon. As illustrated, the union of all the triangles makes up the polygon. Although there are a fixed number of triangles in this example, this logic is applicable to any number of edges and vertices.

Figure A.3 shows how the polygon on Figure A.1 can be made from independent triangles. Notice that the order of vertices b and c is clockwise for each triangle

when viewed from a.

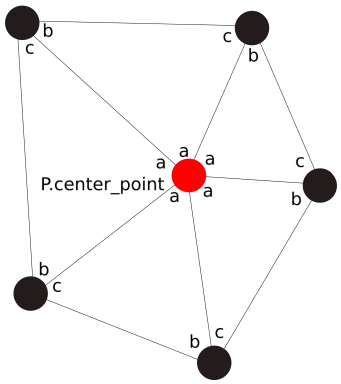


Figure A.3: A set of triangles that fills the interior of a convex plane.

Logically, *G.center_node* maintains a tree of triangles. After merging m triangles into its collection of nodes, Figure A.4 displays the underlying tree structure.

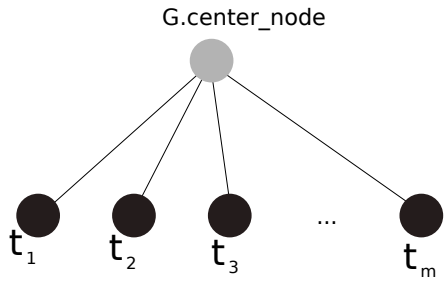


Figure A.4: Tree structure maintained by a polygon.

A.6 Virtual copy

A virtual copy is a best-effort estimation of the geometrical and graphical properties of a physical object. Its attributes are derived from the properties listed in Definitions A.1, A.2 and A.3. More formally, we give a definition of a virtual copy C below.

Definition A.A.5 - Virtual copy

1. C has a set of polygons $C.polygons = \{G_1, G_2, G_3, \cdots, G_n\}$ where $n \geq 1$.

2. C is made up of all the triangles that make up the elements in *C.polygons*.

3. C has a node $C.center\_node$.

4. C has a volume $C.volume \in \mathbb{R}$.

The underlying tree structure of a virtual copy is illustrated in Figure A.5. triangle

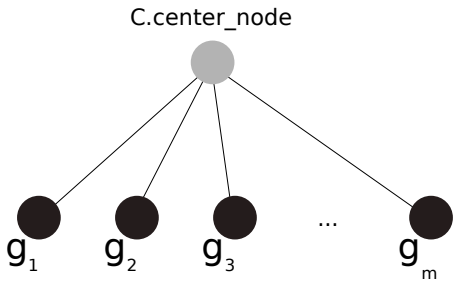


Figure A.5: Logical tree structure of a virtual copy. Each leaf node is a polygon g_i maintaining a tree of triangles.

A.7 Graphical texture

The graphical texture is used to draw a triangle. While the numerical attributes in Definition A.3 logically defines a triangle, we also need a way to define how it is rendered. To give a best-effort representation of physical objects, their visual appearance in the physical world is used. The portion of a physical surface that is covered by a triangle is used as a graphical texture. More precisely, the image of the physical object provided by the live camera feed is used to draw the triangle.

Formally, let the screen texture $texture$ be a visual representation of a smartphone's screen captured at any given moment in time. The following definition of it is given below.

Definition A.A.6 - Texture

1. $texture$ consists of a bitmap. The bitmap contains pixels that make up a polygon as seen from the camera's view. They visually represent a planar physical surface.

2. $texture$ is assumed to completely envelop the extends of the plane.

(a) The triangle and bitmap before being cropped. uv1, uv2 and uv3 signify the coordinate on the bitmap. Here, the positions of the UV coordinates are arbitrarily chosen.

(b) The triangle with a cropped portion of the bitmap as its texture.

Figure A.6: Texture operation

Figure A.6a shows how the image [1] to be cropped. Figure A.6b shows the triangle's texture that fit within its interior.

A.8 Properties of the virtual copies

There are multiple mathematical properties of a virtual copy that can be found. Some interesting properties are area of a polygon and the volume of a virtual copy. These properties can be found using the notions of *nodes* and *planes*.

Implicitly, the formula can only be applied to convex planes. To find the area of a polygon G_i with m sides, we apply Gauss's area formula described in Equation A.1. Let x_i and y_i be two elements in $G_i.world\_coordinates$ for $1 \leq i \leq m$ and A be the area of G_i.

$$A = \frac{1}{2}\left|\sum_{i=1}^{m} x_i y_{i+1} + x_m y_1 - \sum_{i=1}^{m} x_{i+1} y_i - x_1 y_m\right| \tag{A.1}$$

Let G_i be a polygon that represents the uppermost surface of a physical object. Its orientation in world space must be horizontal. Furthermore, let $floor\_node$ be a node that is anchored on a **floor plane**. Assume that the vertical surfaces of the physical object are planar. From this, the height of the physical object is

1. Original photo is found on https://commons.wikimedia.org/wiki/File:Flickr_-_Nicholas_T_-_Bus_Stop.jpg. It is licensed under Creative Commons Attribution 2.0 Generic.

given by Equation A.2.

$$d_y = G_i.center\_node.world\_position.y - floor\_node.world\_position.y \tag{A.2}$$

The absolute value of the subtraction is used in case $world\_origin$ is positioned above $floor\_node$. Finally, the volume of a virtual copy is given by Equation A.3.

$$V = Ad_y \tag{A.3}$$

If one or more vertical surface(s) are not planar, the formula fails to give an accurate estimate of the volume. This formula does not require the vertical planes beneath G_i to be defined. The height between the floor and G_i is already given by Equation A.2.

The values computed in Equations A.1, A.2 and A.3 are estimates of the real values. The parameters of the formulas originate from the underlying plane detection framework. From this fact, the numbers provided are the best effort there is to compute the values.

A.9 Point cloud

A point cloud is a set of points in world space. Such a point is a position in the world space located on a physical surface. The definition of a point cloud and a point is given in Definitions A.7 and A.8, respectively.

Definition A.A.7 - Point cloud

1. *Point* has a set of points $points = \{c_1, c_2, c_3, \ldots c_n\}$ for n points where $n > 0$.

2. Insertions and deletions of $points$ is maintained by the underlying plane detection system.

Definition A.A.8 - Mesh point

1. p has a 3-tuple position in world space $world\_position \in \mathbb{R}^3$.

2. p represents a position in the world space that is located on a physical surface.

3. p has a confidence value $\alpha \in [0, 1]$.

4. p is visualized if $\alpha > 0$.

5. p has a unique ID id.

The point is attached on a physical surface. The point itself does not contain information about the texture, shape or the extends of the physical surface. As the number of points in a point cloud grows large they will create a set of vertices in a polygon mesh.

Because the point has no surface, it does not depend on the any alignment of the surface to which it is attached. This property allows for attaching points on surfaces not limited to horizontal and vertical surfaces. The surfaces represent any smooth manifolds in the physical world. For non-planar surfaces, the points can visualize how physical surfaces curve.

It is the underlying plane detection system that maintains the point cloud. It is a best-effort understanding of the positions of physical surfaces. As the camera hardware hovers in proximity of a physical surface, it manages the point clouds by tracking individual feature points on the surface. The tracking of feature points is continuous, which is similar to the plane detection.

A.10 Polygon mesh

A polygon mesh is a set of edges and vertices that define the shape of a polyhedral object. The mesh's vertices are a point cloud's points and the edges are straight lines between them. The polygon mesh is a best-effort representation of a collection of curved physical surfaces. Formally, a polygon mesh is defined below.

Definition A.A.9 - Polygon mesh

1. M has a set of mesh points $M.points = \{p_1, p_2, p_3, \cdots, t_j\}$ where $j \geq 3$.

2. M has a node $G.center\_node$ positioned at a Plane.

3. M can be moved, scaled and rotated following the logic from Definition A.5.

Bibliography

[1] L. Hou, Y. Wang, X. Wang, N. Maynard, I. T. Cameron, S. Zhang, and Y. Jiao, "Combining photogrammetry and augmented reality towards an integrated facility management system for the oil industry," *Proceedings of the IEEE*, vol. 102, no. 2, pp. 204–220, 2014.

[2] K. Ammari and A. Hammad, "Collaborative bim-based markerless mixed reality framework for facilities maintenance," *Computing in Civil and Building Engineering*, pp. 657–664, 2014.

[3] P. J. DENNING, D. E. COMER, D. Gries, M. C. MULDER, A. Tucker, J. Turner, and P. R. YOUNG, "The final report of the task force on the core of computer science presents a new intellectual framework for the discipline of computing and a new basis for computing curricula. this report has been endorsed and approved for release by the acm education board," 05 2019.

[4] H. K. Stensland, V. R. Gaddam, M. Tennøe, E. Helgedagsrud, M. Næss, H. K. Alstad, A. Mortensen, R. Langseth, S. Ljødal, O. Landsverk, C. Griwodz, P. Halvorsen, M. Stenhaug, and D. Johansen, "Bagadus: An integrated real-time system for soccer analytics," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 10, pp. 14:1–14:21, Jan. 2014.

[5] V. R. Gaddam, R. Langseth, H. K. Stensland, P. Gurdjos, V. Charvillat, C. Griwodz, D. Johansen, and P. Halvorsen, "Be your own cameraman: Real-time support for zooming and panning into stored and live panoramic video," in *Proceedings of the 5th ACM Multimedia Systems Conference*, MMSys '14, (New York, NY, USA), pp. 168–171, ACM, 2014.

[6] M. Stenhaug, Y. Yang, C. Gurrin, and D. Johansen, "Muithu: A touch-based annotation interface for activity logging in the norwegian premier league," in *MultiMedia Modeling* (C. Gurrin, F. Hopfgartner, W. Hurst, H. Johansen, H. Lee, and N. O'Connor, eds.), (Cham), pp. 365–368, Springer International Publishing, 2014.

[7] J. Carmigniani and B. Furht, *Augmented Reality: An Overview*, pp. 3–46. 07 2011.

[8] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino, "Augmented reality: A class of displays on the reality-virtuality continuum," *Telemanipulator and Telepresence Technologies*, vol. 2351, 01 1994.

[9] A. Sanna and F. Manuri, "A survey on applications of augmented reality," *Advances in Computer Science : an International Journal*, vol. 5, no. 1, pp. 18–27, 2016.

[10] M. C. Leue, T. Jung, and D. tom Dieck, "Google glass augmented reality: Generic learning outcomes for art galleries," in *Information and Communication Technologies in Tourism 2015* (I. Tussyadiah and A. Inversini, eds.), (Cham), pp. 463–476, Springer International Publishing, 2015.

[11] G. Evans, J. Miller, M. I. Pena, A. MacAllister, and E. Winer, "Evaluating the microsoft hololens through an augmented reality assembly application," in *Degraded Environments: Sensing, Processing, and Display 2017*, vol. 10197, p. 101970V, International Society for Optics and Photonics, 2017.

[12] D. Amin and S. Govilkar, "Comparative study of augmented reality sdks," *International Journal on Computational Science & Applications*, vol. 5, no. 1, pp. 11–26, 2015.

[13] T.-W. Kan, C.-H. Teng, and W.-S. Chou, "Applying qr code in augmented reality applications," in *Proceedings of the 8th International Conference on Virtual Reality Continuum and Its Applications in Industry*, VRCAI '09, (New York, NY, USA), pp. 253–257, ACM, 2009.

[14] H.-L. Chi, S.-C. Kang, and X. Wang, "Research trends and opportunities of augmented reality applications in architecture, engineering, and construction," *Automation in construction*, vol. 33, pp. 116–122, 2013. pp. 119.

[15] H. Chen, Y. Dai, H. Meng, Y. Chen, and T. Li, "Understanding the characteristics of mobile augmented reality applications," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 128–129, IEEE, 2018.

[16] J. Paavilainen, H. Korhonen, K. Alha, J. Stenros, E. Koskinen, and F. Mayra, "The pokémon go experience: A location-based augmented reality mobile game goes mainstream," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, (New York, NY, USA), pp. 2493–2498, ACM, 2017.

[17] "Augmented reality sdk comparison : Vuforia sdk vs armedia sdk." https://socialcompare.com/en/comparison/vuforia-ar-sdk-vs-armedia-sdk-3h8ymnz0. [Online; accessed 12-May-2019].

[18] "Nike fit mobile application." https://www.engadget.com/2019/05/09/nike-fit-augmented-reality-right-fit-size-shoes/. [Online; accessed 11-May-2019].

[19] "Posten labs ar." https://www.posten.no/labs. [Online; accessed 11-May-2019].

[20] "Ikea place mobile application." https://highlights.ikea.com/2017/ikea-place/. [Online; accessed 11-May-2019].

[21] M. Lettner, M. Tschernuth, and R. Mayrhofer, "Mobile platform architecture review: android, iphone, qt," in *International Conference on Computer Aided Systems Theory*, pp. 544–551, Springer, 2011.

[22] "Android platform architecture." httpshttps://www.sciencedirect.com/topics/engineering/blackbody://developer.android.com/guide/platform, 2018. [Online; accessed 26-April-2019].

[23] "Android architecture." https://developer.android.com/guide/platform/images/android-stack_2x.png. [Online; accessed 12-May-2019].

[24] "Art and dalvik." https://source.android.com/devices/tech/dalvik/index.html, 2018. [Online; accessed 26-April-2019].

[25] M. Froehlich, S. Azhar, and M. Vanture, "An investigation of google tango® tablet for low cost 3d scanning," 07 2017.

[26] A. Diakité and S. Zlatanova, "First experiments with the tango tablet for indoor scanning," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. III-4, pp. 67–72, 06 2016.

[27] "Arcore environmental understaning." https://developers.google.com/ar/discover/, 2018. [Online; accessed 25-April-2019].

[28] "Arcore developer preview 2'." https://www.blog.google/products/arcore/arcore-developer-preview-2/, 2016. [Online; accessed 2-May-2019].

[29] "Point cloud." https://www.sciencedirect.com/topics/engineering/

point-cloud, 2016. [Online; accessed 2-May-2019].

[30] "Google arcore point cloud'." https://developers.google.com/ar/reference/java/arcore/reference/com/google/ar/core/Frame#acquirePointCloud(), 2018. [Online; accessed 3-May-2019].

[31] "Apple arkit." https://developer.apple.com/arkit/. [Online; accessed 11-May-2019].

[32] "Google arcore for ios." https://developers.google.com/ar/reference/ios/. [Online; accessed 11-May-2019].

[33] H. Bae, M. Golparvar-Fard, and J. White, "High-precision vision-based mobile augmented reality system for context-aware architectural, engineering, construction and facility management (aec/fm) applications," *Visualization in Engineering*, vol. 1, p. 3, Jun 2013.

[34] "Bim model." https://www.autodesk.com/solutions/bim. [Online; accessed 10-May-2019].

[35] M. Kopsida and I. Brilakis, "Markerless bim registration for mobile augmented reality based inspection," in *16th International Conference on Computing in Civil and Building Engineering (ICCCBE2016)*, 2016.

[36] "Constructor developer tool." https://play.google.com/store/apps/details?id=com.projecttango.constructor, 2017. [Online; accessed 29-April-2019].

[37] "Wayfair's augmented reality technology now available in its popular mobile shopping app on the asus zenfone ar." https://investor.wayfair.com/investor-relations/press-releases/press-releases-details/2017/Wayfairs-Augmented-Reality-Technology-Now-Available-in-its-Popular-Mobile-Shopping-App-on-the-ASUS-ZenFone-AR/default.aspx, 2017. [Online; accessed 29-April-2019].

[38] D. Varga and S. Laki, "Scalable surface reconstruction in the mobile edge," in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, SIGCOMM '18, (New York, NY, USA), pp. 84–86, ACM, 2018.

[39] R. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," 05 2011.

[40] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[41] D. R. J. Mitchell, *Managing Complexity in Software Engineering*. Peter Petergrinus Ltd., 1990. pg. 5.

[42] "Warehouse floor image." https://c.pxhere.com/photos/e9/b7/industrial_hall_toore_warehouse_industry_industrial_door_spiral_gates_fast_closing_doors_hall_doors-518913.jpg!d. [Online; accessed 12-May-2019].

[43] M. D. Yadav and M. S. Agrawal, "Keystone error correction method in camera-projector system to correct the projected image on planar surface and tilted projector," *International Journal of Computer Science & Engineering Technology*, vol. 4, no. 2, pp. 142–146, 2013.

[44] "Google arcore sdk for android." https://github.com/google-ar/arcore-android-sdk. [Online; accessed 5-May-2019].

[45] "Google arcore node." https://developers.google.com/ar/reference/java/sceneform/reference/com/google/ar/sceneform/Node. [Online; accessed 12-May-2019].

[46] "Google arcore basetransformablenode." https://developers.google.com/ar/reference/java/sceneform/reference/com/google/ar/sceneform/ux/BaseTransformableNode. [Online; accessed 12-May-2019].

[47] "Introduction to activities." https://developer.android.com/guide/components/activities/intro-activities. [Online; accessed 13-May-2019].

[48] "Progal api." http://hjemmesider.diku.dk/~rfonseca/ProGAL/. [Online; accessed 6-May-2019].

[49] "Enable arcore." https://developers.google.com/ar/develop/java/enable-arcore, 2018. [Online; accessed 25-April-2019].

[50] "Enable sceneform." https://developers.google.com/ar/develop/java/sceneform/, 2018. [Online; accessed 25-April-2019].

[51] "Huawei p20 specifications." https://consumer.huawei.com/en/phones/p20/specs/, 2018. [Online; accessed 5-May-2019].

[52] "Lux." https://www.britannica.com/science/lux, 2018. [Online; accessed 5-May-2019].

[53] "Hagner screenmaster." http://www.hagner.se/combination-instruments-1/screenmaster/. [Online; accessed 5-May-2019].

[54] "Black-body radiation." https://www.sciencedirect.com/topics/engineering/blackbody. [Online; accessed 5-May-2019].