Aalto University
School of Science
Master's Programme in Mathematics and Operations Research

Olli Niskanen

An approach for automated rigging to facilitate 3D modeling in mobile augmented reality

Master's Thesis
Espoo, June 6th, 2019

Supervisor: Professor Antti Punkka
Advisor: Kristo Lehtonen M.Sc., M.Ec.

Aalto University
School of Science
Master's Programme in Mathematics and Operations Research

ABSTRACT OF
MASTER'S THESIS

Author	Olli Niskanen		
Title	An approach for automated rigging to facilitate 3D modeling in mobile augmented reality		
Major	Systems and Operations Research	**Code**	SCI3055
Supervisor	Professor Antti Punkka		
Advisor	Kristo Lehtonen M.Sc., M.Ec.		
Date	June 6th, 2019	**Pages**	v + 49

3D modeling has become more popular among novice users in the recent years. The ubiquity of mobile devices has led to the need to view and edit 3D content even beyond the traditional desktop workstations. This thesis develops an approach for editing mesh-based 3D models in mobile augmented reality.

The developed approach takes a static 3D model and automatically generates a rig with control handles so that the user can pose the model interactively. The rig is generated by approximating the model with a structure called sphere mesh. To attach the generated spheres to the model, a technique called bone heat skinning is used.

A direct manipulation scheme is presented to allow the user to pose the processed model with intuitive touch controls. Both translation and rotation operations are developed to give the user expressive power over the pose of the model without overly complicating the controls.

Several example scenes are built and analyzed. The scenes show that the developed approach can be used to build novel scenes in augmented reality. The implementation of the approach is measured to be close to real time with the processing times around one second for the used models. The rig generation is shown to yield semantically coherent control handles especially at lower resolutions. While the chosen bone heat skinning algorithm has theoretical shortcomings, they were not apparent in the built examples.

Keywords	3D modeling, 3D animation, linear blend skinning, computer graphics, direct manipulation, sphere mesh, bone heat skinning
Language	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Matematiikan ja operaatiotutkimuksen maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä	Olli Niskanen		
Työn nimi	Menetelmä 3D-mallin animointijärjestelmän automaattiseen luomiseen älylaitteella lisätyssä todellisuudessa		
Pääaine	Systems and Operations Research	**Koodi**	SCI3055
Valvoja	Professori Antti Punkka		
Ohjaaja	DI, KTM Kristo Lehtonen		
Päiväys	6. kesäkuuta 2019	**Sivumäärä**	v + 49

3D-mallinnus on kasvattanut suosiotaan ei-ammattimaisten käyttäjien keskuudessa viime vuosina. Mobiililaitteiden yleistyminen on johtanut tarpeeseen katsella ja muokata 3D-malleja myös perinteisten työasemien ulkopuolella. Tämä diplomityö kehittää menetelmän verkkorakenteisten 3D-mallien muokkaamiseen lisätyssä todellisuudessa mobiililaitteilla.

Kehitetty menetelmä luo staattiselle 3D-mallille animaatiojärjestelmän ohjauskahvoineen automaattisesti. Näin käyttäjä voi interaktiivisesti muuttaa 3D-mallin asentoa. Animaatiojärjestelmä luodaan muodostamalla mallille likiarvoistus pallomallirakenteella. Luodut pallot kiinnitetään malliin nk. luulämpöpinnoitusmenetelmällä.

Mallin asennon muokkaamiseksi esitellään suorakäyttöjärjestelmä, jossa käyttäjä voi käsitellä mallia helppokäyttöisin kosketusnäyttöelein. Työssä kehitetään sekä siirto- että pyöritysoperaatiot, jotta käyttäjä voi muokata mallia monipuolisesti ja vaivattomasti.

Menetelmän toimivuuden osoittamiseksi työssä luodaan ja analysoidaan esimerkkejä, jotka eivät olisi mahdollisia ilman menetelmän hyödyntämistä. Menetelmän tekninen toteutus on mittausten perusteella lähes tosiaikainen ja käytettyjen mallien käsittelyajat ovat lähellä yhtä sekuntia. Luodut animaatiojärjestelmät ovat semanttisesti merkittäviä erityisesti alhaisemmilla tarkkuuksilla. Vaikka luulämpöpinnoitukseen liittyy teoreettisia ongelmia, ne eivät näkyneet luoduissa esimerkeissä.

Asiasanat	3D-mallinnus, 3D-animointi, lineaarisekoituspinnoitus, tietokonegrafiikka, suorakäyttö, pallomalli, luulämpöpinnoitus
Kieli	Englanti

Acknowledgements

This thesis would not have been possible without all the brilliant people around me. I would like to thank my supervisor, professor Antti Punkka, for the many insightful comments and your patience when guiding me on this scientific journey. This thesis is truly better for all your effort.

My friends and coworkers at 3DBear deserve a special mention. This thesis is inspired by the work we do together daily, and it has been my pleasure to bring a bit of magic to the classrooms all around the world together with you all. I am also grateful to my instructor Kristo, who between managing a startup and closing an investment round has found the time to support me. Thank you for all the guidance, encouragement and passion you've given me.

Both my family and the friends I've made in Aalto have played a pivotal role in my path here. A huge thank you to Pentti, Tuija, Elina and Aino! Special thanks to my roommates at BY, my fellow board members in Guild of Physics years 2011 and 2015 and to all the wonderful people in Rissittely!

Finally, I must thank my partner Katri. You've had my back through my studies at Aalto. You are my love, inspriration and best friend.

Espoo, June 6th, 2019

Olli Niskanen

Contents

Chapter 1

Introduction

The use of 3D modeling has spread to a wide variety of fields, and to users from professionals to casual users. Novice users can already design their homes in 3D using relatively easy desktop software like SketchUp (Trimble Inc., 2018). The advent of non-professional 3D design is democratizing and personalizing manufacturing via 3D printing. This has led users from all backgrounds to take interest in 3D modeling and design.

Since being among the first to adopt use of personal computers in the 1980s, schools and other educational institutions have been in the forefront of technological adoption (Tatnall and Davey, 2012). The same effect with educational institutions leading a mass-market adoption can be observed with 3D design and 3D printers. While professional use and games enjoy the widest usage, a new wave of users is currently being introduced to 3D design in schools all over the world.

Augmented reality (AR) is a technology, which allows a user to see a modified version of their surroundings. Interest in AR has spiked in the wake of the virtual reality industry maturing. AR technology has become more accessible to consumers and early adopters with solutions like Apple's ARKit (Apple Inc., 2018b), Google's ARCore (Google Inc., 2018), and Microsoft's Hololens (Microsoft Inc., 2019) leading the way into ubiquitous computing.

Education, in particular, can benefit from using augmented reality by improving learning outcomes (Akçayır and Akçayır, 2017), along with student motivation and engagement (Bacca et al., 2014; Akçayır and Akçayır, 2017). The possibility to show virtual objects in real environments brings tangibility to abstract ideas. Being able to walk around a virtual solar system can bring clarity to an otherwise difficult to grasp and abstract idea (Kerawalla et al., 2006). Augmented reality has also been shown to assist in learning spatial reasoning (Kaufmann et al., 2005).

The adoption of 3D technologies in the education industry has been slowed by technical and pedagogical difficulties (Kerawalla et al., 2006; Wu et al., 2013). There is a gap on the mobile 3D application market for a creation tool aimed at 3D modeling novices. The current tools are limited to professional users. By combining expressiveness with an easy interface, we can expand the audience for 3D creation. One major issue encountered in creation focused augmented reality applications is that composing a believable scene is difficult due to the 3D models being either static or looping through pre-determined animations that contain only some of the desired poses. The current way to work around this issue is to export the 3D model and open it in a dedicated desktop 3D modeling application, make edits, and bring the edited model back to the AR application. The process is lengthy and complex, which can be off-putting for novice users. Even experienced users lose the benefit of real-time feedback.

This thesis develops a method of processing and editing 3D models in augmented reality, directly addressing the scene composing challenges present in current mobile augmented reality creation tools. The editing method chosen is *posing*. When editing by posing the user moves control handles, which affect the pose of the model. This capability to make instant changes to any 3D model, whether pre-made by professionals or self-made and imported, allows quick iteration.

Specifically, the approach developed in this thesis has two main components. First the model is analyzed and a control rig is fitted to the model. This

allows manipulating the object in question through a limited amount of control handles, rather than moving vertices individually. This process is called *rigging*, and it is usually conducted by a professional artist. The optimization method presented in this thesis uses a *spherical error metric* (Thiery et al., 2013) to automatically find a good approximation of the model. The approximated model is then mapped back to the original using a technique called *linear blend skinning* (Magnenat-Thalmann et al., 1988; Lewis et al., 2000). This determines how much each vertex in the original object is deformed when moving an individual control handle.

The second component of the developed approach allows the user to pose the model using the created rig. Posing the model is done by translating and rotating the control handles in three dimensions, which leads to six degrees of freedom per control handle. Fully controlling and manipulating even the simplified rig is cumbersome for a trained professional equipped with a mouse and keyboard. This difficulty is further increased, as the target users for the application are novices on mobile devices, where input precision is limited. This thesis presents one way of structuring the input method so that novice users face an easy learning curve, and are able to do modeling tasks that would not otherwise be possible for them.

This thesis is limited to the implementation, evaluation and discussion of a proof of concept level tool showing the benefits of the approach. The end result should allow a product development team building a mobile 3D creation tool to verify that the proposed approach provides new capabilities for the end users, and that an implementation can be made without compromises to usability. The approach is developed for 3DBear Oy, a Finnish company in the field of educational technology. In order to assess real-world use cases and actual technical hurdles, the implemented tool is integrated into the mobile augmented reality creation tool of the same name developed by the company. In order to ensure practical relevance, this thesis discusses both the usability and reliability perspectives necessary for deploying the application to end users. The discussion is limited to a non-exhaustive analysis of potential issues and improvements.

This thesis is structured as follows. Chapter 2 reviews the existing research on the topic. It introduces basic concepts of 3D models, and how the traditional method of rigging 3D models works. The most common methods for automatically rigging and skinning 3D models are also presented in this chapter. An introduction on augmented reality is given to give the reader an understanding of the context the approach is used in. Finally, a brief review of usability principles is given to give background for the implementation choices. Chapter 3 describes the approach for automated rigging developed in this thesis. The key algorithms are presented and implementation specific concerns are discussed along with mentions of the major third party components used. The experimental results are reviewed in Chapter 4. Example scenes are presented with before and after photographs showcasing the application of a tool that incorporates the automated rigging approach. Found shortcomings of the tool are documented and explained. Chapter 5 contains discussion on how well the implemented approach meets expectations. The found results are given context, along with the author's predictions on the future of the approach. The shortcomings are individually analyzed for potential mitigations and improvements. In addition, the quality and performance of both the rigging and skinning components are analyzed. Finally, Chapter 6 summarizes the main contributions and results of this thesis along with future research directions.

Chapter 2

Background

This chapter introduces the relevant literature and background knowledge used throughout the rest of the thesis. Section 2.1 gives the necessary concepts to understand the computer graphics processing pipeline and 3D animation, both of which are essential for understanding the rest of the thesis. Section 2.2 builds on this knowledge and explains necessary steps for automatically building an animation rig for posing a character. The most important methods of rigging and skinning are presented and compared. Section 2.3 provides an overview of the mobile augmented reality environment and the underlying concepts of tracking and feature points. It also describes the system the user interacts with in order to understand the context the user will use the developed approach in. Finally, Section 2.4 highlights the background behind the chosen interaction model and the usability evaluation criteria used to guide implementation and analyze the results.

2.1 Computer graphics and 3D animation

The end goal of computer graphics work and applications is to communicate visually via a computer's display (Hughes et al., 2014), usually by producing

an image on a screen. The pipeline that processes a 3D scene and eventually produces a visible 2D image is complex and nuanced if all the details are taken into account. On a more general level, taking the components as black boxes, the process is fairly straightforward. The basic building blocks of the scene are mesh data, texture data, lighting data and the current view.

The triangular mesh representation is the industry standard (Hughes et al., 2014), and this thesis adheres to the standard. As any polygonal mesh can be transformed into a triangular mesh using polygon triangulation (Lee and Schachter, 1980), the rest of this thesis considers 3D meshes as triangle meshes in order to keep the methods free from needless complexity. Based on this choice, we can define a mesh as a construct consisting of a set of vertices V and a set of triangular faces T those vertices form. Each vertex j in V has a position v_j. Figure 2.1 shows an example triangular mesh in the shape of a dolphin.

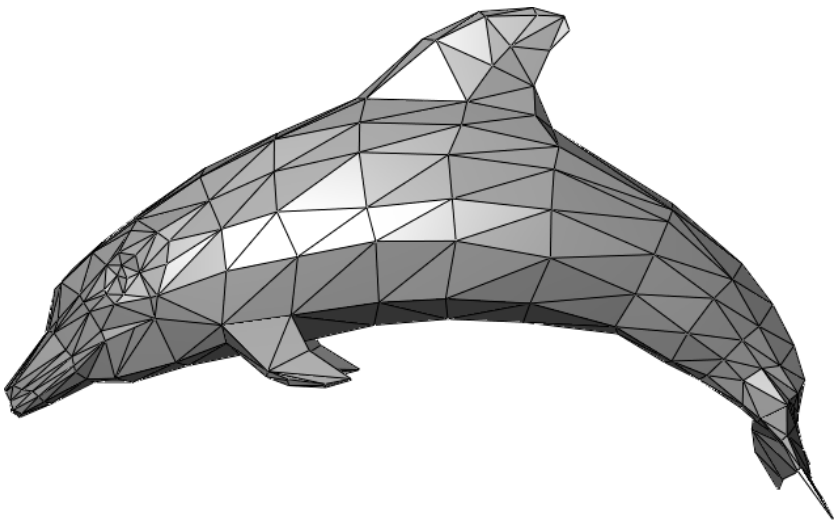


Figure 2.1: A triangular mesh representing a dolphin

In the pipeline described by Hughes et al. (2014), texture data at its simplest defines the base color of the meshes on the scene. It is usually formatted as 2D pictures depicting the surface color of each face. When drawing the face, lighting data is also considered, providing shading, highlights and other variations depending on the shading program used. The view is often referred

to as the virtual camera, providing a window to the scene. It is represented as a matrix transform that allows the graphics card to calculate the correct positions for each face in the scene.

Figure 2.2 shows a high level overview of how the basic building blocks of a 3D scene are processed by the graphics pipeline, which generates the resulting image. The 3D application pushes data consisting of polygonal meshes, texture data, the current view and lighting data on to a graphics card. On the graphics card, the pipeline starts with a perspective transformation in order to align the vertices of each mesh based on the view. Each mesh, now in the correct perspective, is then colored based on texture and lighting data. The final image is then assembled and displayed.

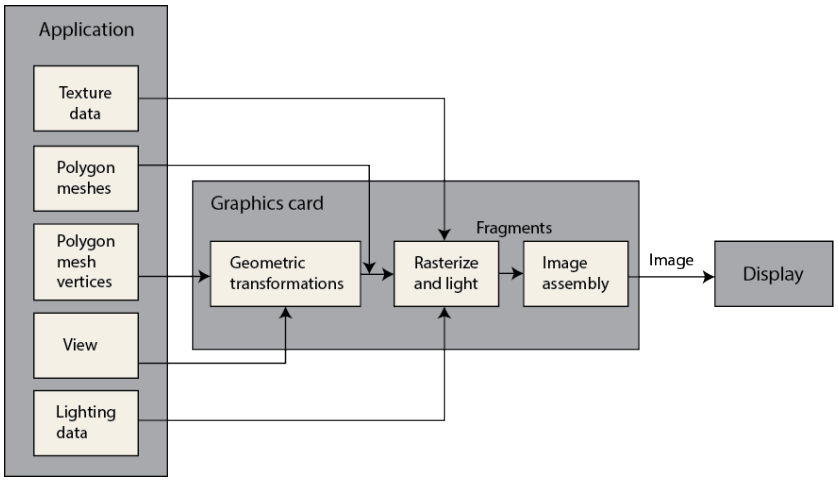


Figure 2.2: The computer graphics pipeline (Hughes et al., 2014)

The pipeline as such is used for acquiring a single image of the scene in question. This is fine for some purposes, like creating digital art or creating an image of a new product for marketing. Many applications using computer graphics, however, rely on application logic to modify the pipeline inputs in order to create a dynamic sequence of rendered images. When shown in a rapid sequence, the contents of the images blend into each other, and they are perceived as an animation rather than individual frames.

The multitude of ways of modifying a scene ranges from varying the view to create the appearance of the virtual camera being moved, to gradually increasing light intensity to simulate a sunrise. As such, it is not possible to give an exhaustive listing of the different animation methods used by the industry in the scope of this thesis. For a more complete discussion of the ways to create dynamic effects, the reader is referred to a computer graphics handbook such as Hughes et al. (2014) or Akenine-Moller et al. (2018).

One of the more important and popular systems for creating motion in an otherwise static scene is rig-based animation, also called *skeletal subspace deformation* (Lewis et al., 2000). It is often used to manipulate especially characters, but the technique is universally applicable to any mesh-based 3D model. The method deforms an input mesh by applying transformations to a simpler *control rig*, sometimes called the skeleton. The rig consists of elements called *bones*, sometimes also referred to as *joints* or *handles* in other literature. This thesis uses the term *bone* for the components of the control rig and the term *handle* for the user-interactable visualization of the bones.

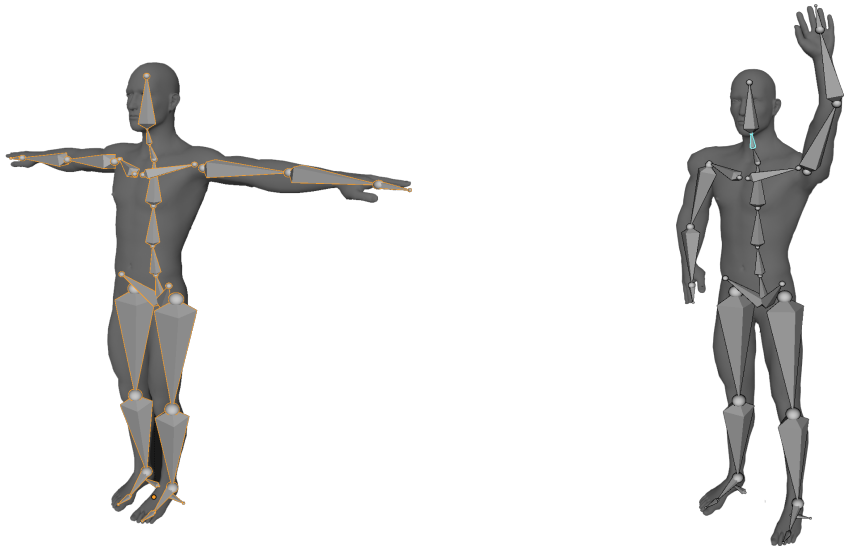
The magnitude of the effect each bone in the control rig has on a given vertex is controlled by a *bone-vertex weight*. The final position $\boldsymbol{v}'_j$ of each vertex j is given by a weighted sum of all the bone transforms

$$\boldsymbol{v}'_j = \sum_i w_j^i T_i(\boldsymbol{v}_j), \tag{2.1}$$

where w_j^i is the weight of bone i for vertex j and T_i is the transformation of bone i (Lewis et al., 2000).

Figure 2.3a shows an example with a humanoid 3D model and a simple animation rig to control its pose. By applying geometric transformations like translating, rotating or scaling to the bones, an artist is able to deform the humanoid to poses other than standing straight with the arms spread out. Figure 2.3b shows how the created rig can be used to pose the character to stand in a more relaxed way and wave. Creating these animation rigs is

a functionality found only in advanced 3D modeling software. Popular tools used by professional animators include 3D Studio Max, Maya and Blender (Beane, 2012).



(a) Default T-pose (b) Posing the humanoid to wave

Figure 2.3: A humanoid 3D model superimposed with its animation rig

Beane (2012) explains that the process of creating an animation rig is the job of a rigging artist in a 3D animation production. The artist's work starts by creating the skeleton. Each bone is individually placed into the object being rigged. Handles for manipulating the bones are created to help the animator pose the object. The bones are connected with the geometry by applying a *skinning deformer*, which is provided with the bone-vertex weights by the artist. This completes the functionality of the rig. The mesh combined with the rig and the skinning deformer can then be posed by applying transformations on the bones. This process is usually conducted by an animator.

2.2 Automated rigging of 3D models

When automating the rigging process, the steps traditionally done by a professional artist need to be recreated with minimal human intervention. Jacobson (2014) divides the process into three separate, but interconnected parts: defining the bones, defining the weights and applying transformations. The process follows a similar progression as the manual one described in Section 2.1.

The first part is defining the bones that control the mesh. This is comparable to the skeleton creation in manual rigging. There are several methods for automatically creating the bones. Au et al. (2008) describe a medial-axis method that iteratively contracts the mesh in the direction of the curvature-flow normal. As a result, the method produces a curved skeleton. Since the most common skinning techniques depend on a simple hierarchy of linear and rigid bones, allowing curved bones can be impractical.

Another common approach to generating the bones is segmenting the mesh and deriving the bones from the connected segments (Katz and Tal, 2003; Bharaj et al., 2012; Thiery et al., 2013). For example, the algorithm presented by Katz and Tal (2003) finds cuts to segment the model. The concavities present in the mesh are used to identify suitable places for the cuts. The bones generated by the method are placed at the center of the cuts and connected based on the hierarchy of the segmentation.

The second part of the automated rigging process is defining the bone-vertex weights. As shown in equation (2.1), the final position of each vertex is determined by a weighted sum, where the elements are the original vertex position $\boldsymbol{v}_j$ transformed by a transform applied to bone i, $T_i(\boldsymbol{v}_j)$. The task for the algorithm is, given the original 3D mesh and the hierarchy of generated bones, to find appropriate bone-vertex weights for each possible pair of vertex j and bone i. Jacobson (2014) discusses desirable qualities for skinning weights in conjunction with the different available methods for calculating

them. He lists the following eight criteria for evaluating the quality of a weight generation algorithm:

- Interpolation: the weights at handle i should be one and the weights at all other handles should be zero. This criterion allows *direct manipulation*.
- Partition of unity: the weights at each vertex should sum up to one. If this criterion is violated, basic geometric operations behave erratically. Even applying the identity transform causes the vertices to move.
- Derivative continuity C^1: the weights should maintain smoothness in order for the deformed mesh to also maintain any existing smoothness.
- Shape awareness: the weights should decay with respect to geodesic distance on or within the shape rather than a simple Euclidean distance. This criterion means that vertices that are close to each other, but connected only through a long graph of intermediate vertices should have different weights, as they are semantically different parts.
- Non-negativity: weights should be constrained to be non-negative. Together with the partition of unity constraint, the resulting range is $[0, 1]$, as weights above one would need to be counterbalanced by negative weights. Negative weights would cause the vertices to move opposite to the action taken by the user, which can be counterintuitive.
- Monotonicity: weights should monotonically decrease based on the geodesic distance from the bone. Again, geodesic distance is a good approximation for semantic closeness, so the weights should reflect that.
- Locality: weights should be zero over most of the shape. This is more of an implementation concern, as many skinning deformer implementations have a limit on how many bones can affect the position of a specific vertex.

- Speed: a faster processing time is better, all other things being equal. In cases where there is a trade-off between speed and quality, application specific concerns need to be taken into account. For example, when the processing time is just a single up-front delay, a longer time is tolerable. When iterating quickly, a near real-time feedback loop is preferable.

Jacobson (2014) compares different skinning techniques on the criteria above. All closed form approaches, such as weights based on the inverse Euclidean distance, fail on the shape awareness criterion. The best approaches work by numerically minimizing some chosen energy value. Out of the energy minimizing approaches, notable highlights are the *constrained biharmonic weight* method (Jacobson et al., 2011), and the *bone heat diffusion* method (Baran and Popović, 2007).

The constrained biharmonic weight algorithm performs well when it is evaluated against the criteria above. It maintains interpolation, partition of unity, derivative continuity, non-negativity, monotonicity, locality and shape awareness, that is, it produces the highest quality skinning result possible on the chosen criteria. The drawback is the speed of processing, as the algorithm requires iterative solving of convex optimization problems. In addition the implementation requires a volumetric discretization of the mesh. This is done, for example, with the finite element method, which has a notoriously slow runtime (Jacobson et al., 2011).

The bone heat diffusion method fails on the monotonicity, locality and derivative continuity metrics, satisfying all other binary criteria. As such the skinning results can show unintuitive artifacts due to improper weight distribution. The bottleneck in the algorithm is calculating the discrete surface Laplacian containing a bone visibility calculation that needs to be done per vertex. The visibility calculations are parallelizable, and can be offloaded to a GPU which excels in a task of this sort. The processing time is therefore shorter, though quantifying the exact benefit would require empirical testing.

2.3 Mobile augmented reality

Azuma (1997) defines augmented reality as having three different characteristics: it combines real and virtual, it is interactive in real time and it is registered in 3D. Mobile augmented reality creates the augmented reality experience using a combination of sensors present in the mobile device. The most important one is the camera, which is used to capture reality. The augmentations, meaning the virtual objects and environments, are drawn on top of the video feed from the camera. Figure 2.4 shows an example, where a virtual Mars colony is created on a real sandbox.

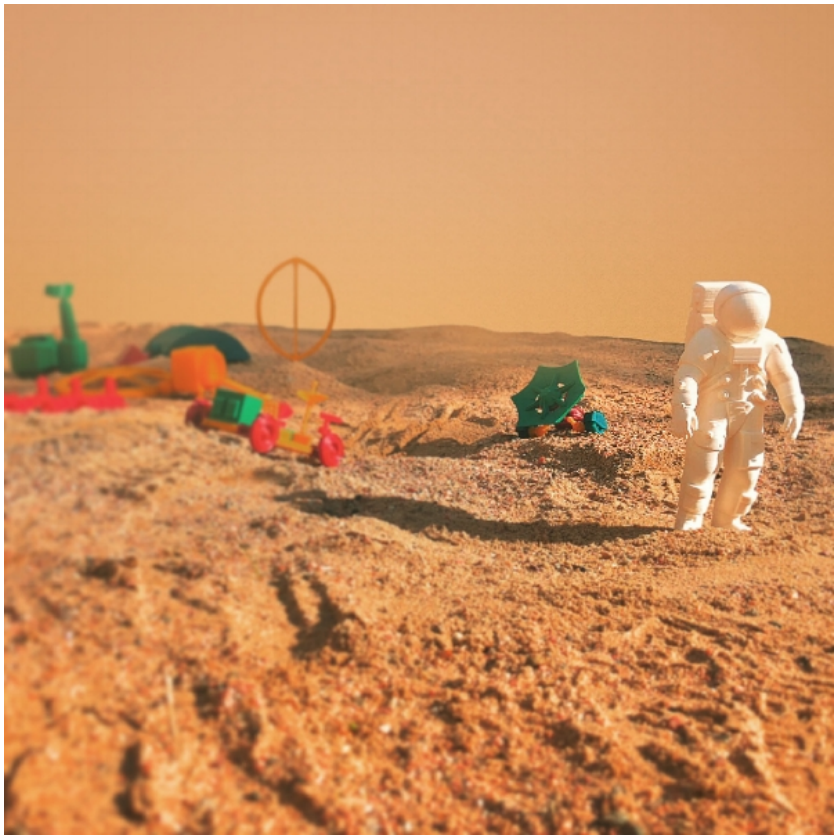


Figure 2.4: An augmented reality image showing a virtual Mars colony

The illusion of virtual objects appearing correctly aligned to the real world is called *registration* (Holloway, 1997). Aligning virtual objects to a fixed

point of view, *static registration*, was fairly well achieved already in the late 1990's (You et al., 1999). A manual calibration of an augmented reality system can create virtual scenes that are "good enough" for visualization or story telling purposes (You et al., 1999). Some scenes may even be nearly indistinguishable from reality on a glance. Statically registered systems can be useful for taking still images. To create a more immersive experience, we need the illusion of objects staying in place when viewed from multiple angles in real time. This is hampered by drift inherent to the motion tracking systems. The solution is called *dynamic registration*, which requires *tracking* (Schmalstieg and Hollerer, 2016).

On mobile devices the implementation of tracking relies heavily on applying computer vision algorithms to the camera images captured by the device (You and Neumann, 2001). In such *optical tracking*, the captured image is first analyzed to find features or landmarks that are used to relate the successive images to each other. The features can be predetermined markers, such as lights or shapes arranged in a specific pre-determined relation to each other. Alternatively, natural features present in the surroundings can be derived from the images and tracked. This thesis focuses on markerless, natural-feature based tracking, as it requires no setup and is easier to get started with for a novice user. For a review of the marker based approach, see the survey conducted by Lepetit et al. (2005) or the paper by Hoff et al. (1996).

There exist several algorithms for extracting feature points from an image (Moravec, 1977; Harris and Stephens, 1988; Förstner, 1994). The algorithms used in Apple's ARKit and Google's ARCore, which are the target AR platforms for this thesis, are not public. Thereby, this thesis is unable to give a more detailed description of the image recognition process used. Both companies' documentation indicates that the tracking is based on a combination of feature points and inertial sensor data (Google Inc., 2018; Apple Inc., 2018a). This combined method is called *visual-inertial odometry*.

In visual-inertial odometry, the camera provides a ground-truth for the tracking algorithm (You et al., 1999). To provide real-time registration, the optical

tracking needs to be augmented with data from the device's accelerometer, gyroscope, compass and GPS sensors through sensor fusion. The more sensors are available, the more accurate the process can be made (Schall et al., 2009). Using both the camera and inertial sensors provides a quality of tracking that people are impressed by (Jung et al., 2016). However, the experience of just walking around in an augmented reality scene grows old quickly, and many mobile AR applications allow the user to interact with their virtual surroundings as well.

The screen is a central input device in an augmented reality experience. It often has a high resolution, with the latest flagship models offering enough pixels to be indistinguishable with the human eye at the usual arms-length distance. The input resolution of the multi-touch display is similar, however it is reduced by the fact that the devices are mostly operated by fingers rather than styli. The average 2 degrees of freedom touch interaction resolution is around 1 centimeter (Wang and Ren, 2009). Most mobile touch screens are designed with a single user in mind and thus allow for 10 simultaneous touches, each with two degrees of freedom (Wang and Ren, 2009). In addition the user needs at least one hand to hold the device in an augmented reality experience, which leaves a single free hand and the thumb of the hand supporting the device for interaction.

Hinrichs and Carpendale (2011) researched how museum visitors interacted with a multi-touch display in an exhibition. The by far most common way to perceive the gesture is as a single unit – either a single finger or the whole hand. Multi-hand and multi-finger gestures were somewhat common especially for rotation and scaling. As the smartphone producing the AR experience needs to be held, this finding shrinks the likely natural touch screen interactions to one finger drags, and possibly two finger scaling and rotation.

The augmented reality tracking provides another important method of input. As we are able to track the position and the orientation of the device, those can be used as another 6 degrees of freedom input (Henrysson and

Billinghurst, 2007). In practice the input design is constrained by the fact that the same input is already being used to drive the camera into the virtual world, so requiring full six degrees of freedom motions might clash with the user's desire to view a specific portion of the scene at the same time.

2.4 Direct manipulation and ease of use

Designing and implementing an interactive 3D tool is more difficult than a similar tool that is limited to two dimensional objects and content (Herndon et al., 1994). Many of the common navigational problems in virtual environments, such as maintaining spatial orientation or steering over-control (McGovern, 1989) are not issues in augmented reality, as they happen in the real world and users are already accustomed to walking around in their surroundings.

Shneiderman et al. (2016) mention that while augmented reality is still an evolving field with many different application areas, interface designers can be aided by the existing large body of research done on more traditional direct manipulation interfaces. Direct manipulation is characterized by three principles (Shneiderman et al., 2016):

1. Continuous representations of the objects and actions of interest with meaningful visual metaphors
2. Physical actions or presses of labeled interface objects (i.e., buttons) instead of complex syntax
3. Rapid, incremental, reversible actions whose effects on the objects of interest are visible immediately

There exist multiple different methodologies for evaluating the usability of an interface. Nielsen (1994) divides the methods to four different areas: *automatic* methods utilize a computer program to calculate usability measures,

empirical methods assess the interface in use by actual end users, *formal* methods use some sort of model to calculate a usability measure and *informal* methods rely on heuristics and an expert evaluator. Formal and automatic methods are still mostly used in specific settings and in research settings (Nielsen, 1994; Shneiderman et al., 2016) with some notable limited use exceptions, such as the GOMS method (John and Kieras, 1996).

Both Nielsen (1994) and Shneiderman et al. (2016) deem that the most common real-life analysis is a combination of empirical and informal evaluation. Shneiderman et al. also note, that the choice of evaluation method depends, among other factors, on the following: stage of design, novelty of the project, number of expected users, criticality of the interface, costs of the product and finances allocated for testing, time available, experience of the design and evaluation team, and environment where interface is used.

Heuristic evaluation is an informal evaluation technique, where an expert or a group of experts critique an interface to determine conformance with a short list of design heuristics (Shneiderman et al., 2016). An example list of heuristics given by Shneiderman et al. (2016) is the Eight Golden Rules, listed in Appendix A. As a summary, the rules guide design to take the limitations of humans into account. We humans are not perfectly knowledgeable and our memory is fallible. We need feedback and our errors need to be accounted for, either by preventing them in the first place or by allowing actions to be easily reversed. Consistency and clarity are key for a good experience.

Chapter 3

Approach for automated rigging

In this chapter, the chosen approach for automated rigging is explained. The reasoning behind the choice of algorithms and design is given. Both the third party components used in the implementation and the ones developed by the author for this thesis are presented.

Section 3.1 gives the high level overview of the developed method. Sections 3.2 and 3.3 detail the automatic rig generation and skinning, respectively. Section 3.4 explains how the direct manipulation model is used for posing in the approach.

3.1 Overview of the approach

As explained in Section 2.2, three separate, but interconnected components are required for the automated rigging process. In the developed approach, the rig generation algorithm is based on sphere mesh approximation, skinning is done using a bone heat diffusion method and posing is done by direct manipulation on the touchscreen.

The mesh processing pipeline built for the developed approach can be seen in Figure 3.1. As shown in the figure, the automated parts of the approach require the desired number of handles and the 3D mesh as inputs from the user. A mesh with control handles and an appropriate skinning deformer is produced as an output. The user is then able to interactively pose the model using the generated handles to control the bone positions. If the user wishes, changing the number of handles and posing again is supported, although the already done changes are lost. Once the user is ready, they can continue building the scene either by deselecting the model or switching to another tool.

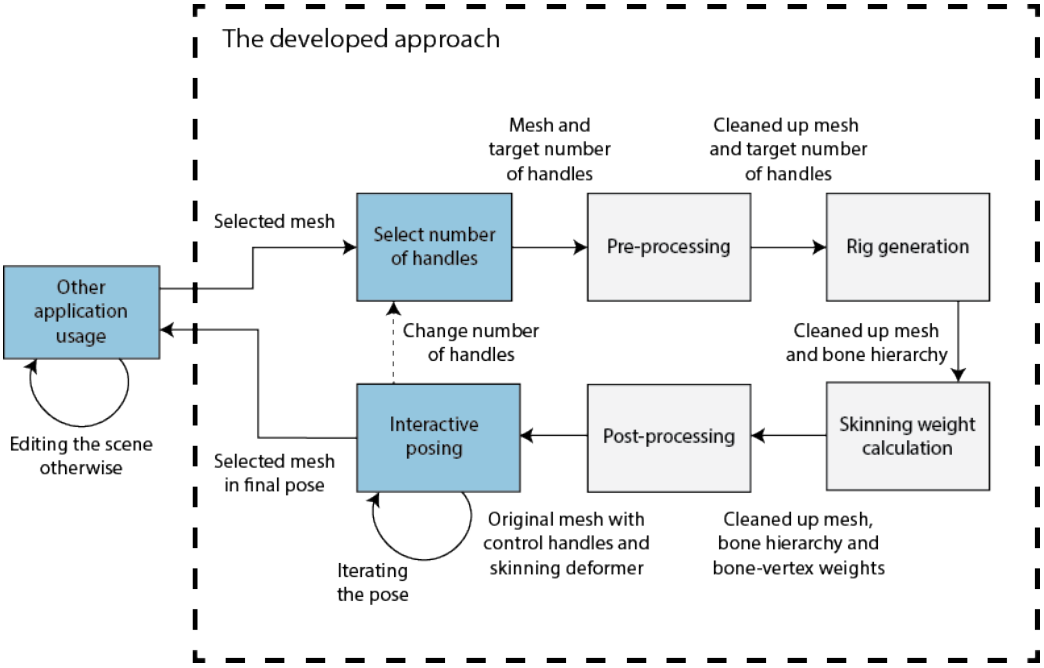


Figure 3.1: The developed mesh processing pipeline (the phases with user interaction are drawn in blue and automated phases in light gray)

To evaluate the approach, a tool implementing it was created as part of the 3DBear mobile application. Many of the core 3D functionalities used in the tool are provided by the host 3DBear application. The application is built using the Unity game engine, which provides basic 3D engine features, such as 3D rendering, importing 3D models and the regular translate, rotate

and scale operations. The Unity ARKit plugin is used in the application to incorporate high quality AR tracking, the camera background and model lighting estimation. The bulk of the implementation work was done in C# in order to seamlessly integrate with the Unity engine objects.

The pre-processing step of the approach consists of some technical operations ensuring that the mesh is easy to process later on in the pipeline. To clean up the mesh, we start by centering the model, applying unit scale and default rotation, while saving the original values in order to restore them in the post-processing step. This allows us to work directly in the mesh coordinate system rather than keep track of the otherwise necessary world-to-mesh coordinate system transformations in all steps. Next, any existing Unity renderers are removed in order to later replace them with the one containing the skinning information. Last, as Unity meshes often contain duplicate vertices to separate sharp and rounded corners, a vertex de-duplication process is run. The end goal is to not affect the vertex count of the mesh, so a mapping between the original and de-duplicated vertices is also generated in order to later apply deformations on the actual vertices rather than the de-duplicated ones.

The cleaned up mesh and user-chosen number of control handles are then used as inputs for the automated rig generation. The rig generation process is detailed in-depth in Section 3.2. As an output, we get the hierarchy of bones that together form the skeleton for the chosen mesh. Both the cleaned up mesh and the newly generated bone hierarchy are then given for the skinning algorithm as inputs. The algorithmic details of the skinning process are explained in-depth in Section 3.3. The process yields an array of bone-vertex weights for each bone in the hierarchy.

The heavy processing steps are then complete, and all that remains is to generate the result for the user in the post-processing step. First, the object is returned to its initial position, rotation and scale. A transparent material is swapped in place of the regular one to allow the user to see the handles inside the mesh. The renderer is created and initialized with a skinning

deformer. The bone hierarchy and bone-vertex weights are applied on the skinning deformer. Last, for each bone, a user-visible and interactable handle is created. As a result, the user sees the 3D model with the automatically generated handles, and can start posing the model as further explained in Section 3.4.

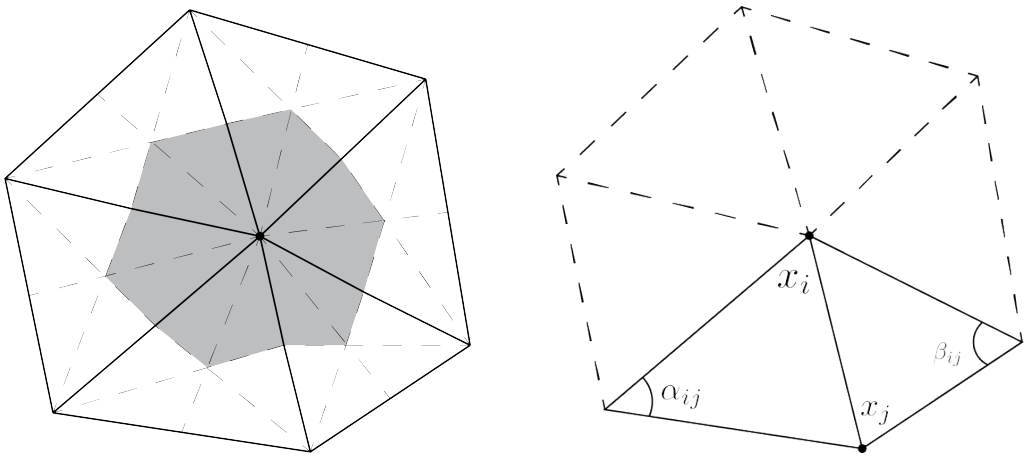
3.2 Rig generation with a sphere mesh approximation

The sphere mesh approximation approach introduced by Thiery et al. (2013) was chosen for rig generation. Medial axis methods (Au et al., 2008) were initially considered, but the sphere mesh algorithm was deemed more suitable in two ways: efficiency and multi-resolution capability. Mobile devices require an efficient algorithm. Medial axis methods can be fairly efficient, but their core implementation still consists of two steps: shifting vertices towards the center and then decimating them. The sphere mesh based approach relies entirely on decimation, leaving the iterative contraction phase out and saving computational cycles. An empirical analysis would be necessary to authoritatively assert the difference in speed.

The intermediate approximations provided by the sphere mesh algorithm can be useful. The medial axis method is useful to find a skeleton shaped control rig, but is limited to a single resolution. The sphere mesh based approach is naturally multi-resolution. By allowing the user to choose a comfortable level of detail for the approximation, the rig generation can target anything from just a few control handles, up to returning a control handle for each vertex on the mesh. This allows the user to shift from high-level overarching changes to editing small details.

The rig generation algorithm consists of an initialization phase, a main loop and a finalization phase. In the initialization phase, the required data structures are created for further processing. The vertices of the mesh are first

transformed into an initial set of approximating spheres. For each vertex, a sphere is initialized to the position of the corresponding vertex, with a radius of zero. In addition, each sphere is assigned the corresponding vertex's *barycentric cell* as an initial set of faces it currently approximates. Such barycentric cell consists of a third of all the triangles in the T_1 neighborhood of the vertex. The T_1 neighborhood contains the triangles that include the vertex. Figure 3.2a shows an illustration of the barycentric cell of a vertex and Figure 3.2b shows the T_1 neighborhood of a vertex.



(a) The barycentric cell of a vertex, shown in gray. The medians of the neighboring triangular faces (dashed lines) form the cell.

(b) Vertex $\mathbf{x}_i$ along with its N_1 and T_1 neighborhoods. Also pictured are angles α_{ij} and β_{ij} corresponding to the edge $(\mathbf{x}_i, \mathbf{x}_j)$

Figure 3.2: Vertex properties

The spheres are connected based on the edges between the vertices of the mesh. For each edge uv connecting the vertices u and v, the corresponding spheres s_u and s_v are combined into a pair. These pairs $\{s_u, s_v\}$ are candidates for decimation. The decimation is guided by a *spherical quadratic error metric*. The metric is based on the signed distance from a sphere to an oriented plane. The metric is defined (Thiery et al., 2013) as

$$\mathbf{Q}(\mathbf{s}, \mathbf{p}, \mathbf{n}) = \frac{1}{2}\mathbf{s}^T A(\mathbf{n})\mathbf{s} - \mathbf{b}(\mathbf{p}, \mathbf{n})^T\mathbf{s} + c(\mathbf{p}, \mathbf{n}),$$

where

$$A(\mathbf{n}) = 2 \begin{bmatrix} \mathbf{n}\mathbf{n}^T & \mathbf{n} \\ \mathbf{n}^T & 1 \end{bmatrix},$$

$$\mathbf{b}(\mathbf{p}, \mathbf{n}) = 2(\mathbf{n} \cdot \mathbf{p}) \begin{bmatrix} \mathbf{n} \\ 1 \end{bmatrix},$$

and

$$c(\mathbf{p}, \mathbf{n}) = (\mathbf{n} \cdot \mathbf{p})^2.$$

$\mathbf{s} \in \mathbb{R}^4$ is the vector representation of a sphere:

$$\mathbf{s} = \begin{bmatrix} \mathbf{q} \\ r \end{bmatrix} \in \mathbb{R}^4.$$

$\mathbf{p} \in \mathbb{R}^3$ and $\mathbf{n} \in \mathbb{R}^3$ are 3D representations of a point the plane intersects, and the normal of the plane we calculate the error for, respectively:

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix},$$

and

$$\mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}.$$

The metric takes into account the direction of the plane, which is useful when calculating errors for boundary representations of a volume. It handles

convex and concave regions in an intuitive way and yields logical results for the value of the error.

The cost of a sphere s approximating a barycentric cell of an individual vertex v_i is calculated as the weighted sum for the error of each plane in the barycentric cell:

$$\mathbf{Q}_i(s) = \sum_{t_j \in T_1(v_i)} \frac{area(t_j)}{3} \mathbf{Q}(s, p_{t_j}, n_{t_j}),$$

where $T_1(v_i)$ denotes the triangles adjacent to v_i.

In order to keep track of the best available collapse operation, a priority queue is initialized. As the priority queue is on one of the hot paths of the algorithm, a fast implementation is necessary to prevent delays for the user. An open source C# priority queue implementation by Pflughoeft (2013) is used to ensure good performance. The priority of collapsing an edge $\{s_u, s_v\}$ is the cost of the resulting union of faces being approximated by a single sphere:

$$\mathbf{Q}_{uv} = \min_s \{\mathbf{Q}_u(s) + \mathbf{Q}_v(s)\}.$$

The priority queue nodes are constructed to use vertex index based comparisons. As a necessary part of the initialization, the costs of each known collapse operation are calculated and the resulting optimal sphere is retained until the collapse is realized. For the cost calculations, the open source spherical quadratic error metric implementation by Thiery et al. (2013) is used.

Inside the main rig generation loop, the end condition is set to the number of resulting control handles requested by the user. The main processing part of the loop consists of taking the top node of the priority queue, and collapsing it into a single new sphere. Then, for all the nodes remaining in the queue, the just collapsed spheres are replaced with the new one. The spherical quadratic error measure is re-evaluated for any changed nodes.

Once the loop reaches the desired level of decimation, the remaining nodes are processed into the control rig hierarchy in the finalization phase. The root of the hierarchy is chosen at random, and the connected nodes are recursively embedded into the hierarchy under the root node. In case the source mesh contains multiple disconnected graphs, the operation is repeated until all the remaining control points are placed in the hierarchy. This created hierarchy is the final control rig and we are able to start the skinning process.

3.3 Bone heat skinning

Based on the comparison conducted by Jacobson (2014), the initially chosen skinning algorithm was the bounded biharmonic weight skinning. Unfortunately, during implementation the drawbacks of the bounded biharmonic weights approach were evident: running the algorithm on a mobile device took too long. The problems were exacerbated by instabilities in finding a tetrahedralization for more complex models. Both model quality issues and large memory use when working on more complex meshes prevented the implementation from being used in a meaningful manner. The natural replacement for bounded biharmonic weights is the popular bone heat diffusion method mentioned in Section 2.2. While the resultant skinning weights are not artifact free, the process of calculating them is reliable and quick.

The bone heat skinning method described by Baran and Popović (2007) is heat equilibrium inspired. It is used to automatically calculate weights for a given mesh and bone hierarchy. According to the method, we apply weights by treating the volume bounded by the mesh as an insulated heat-conducting body. The weights for each bone are processed as an independent system. The physical analogy is a system, where we force the temperature of the chosen bone to 1 and the rest of the bones at 0. These boundary conditions along with the differential equation for heat diffusion (Luikov, 1968) yield an equilibrium we solve numerically. The vertex temperature in the solution is interpreted as the vertex weight.

As the method is used to set the weights for the vertices, modeling the heat transfer for the whole body is not required. Instead, we only solve the equation on the surface. This allows us to model the equilibrium for the bone i as

$$\frac{\partial \mathbf{w}^i}{\partial t} = \Delta \mathbf{w}^i + \mathbf{H}(\mathbf{p}^i - \mathbf{w}^i) = 0.$$

This can be written as

$$-\Delta \mathbf{w}^i + \mathbf{H}\mathbf{w}^i = \mathbf{H}\mathbf{p}^i, \tag{3.1}$$

where Δ is the discrete surface Laplacian, $\mathbf{p^i}$ is a vector with $p^i_j = 1$ if the nearest bone to vertex j is i and $p^i_j = 0$ otherwise, and $\mathbf{H}$ is a diagonal matrix with element H_{jj} being the heat contribution weight of the nearest bone to vertex j. Baran and Popović (2007) use $H_{jj} = \frac{1}{d(j)^2}$, when shortest path from vertex j to the nearest bone is contained in the mesh volume, and $H_{jj} = 0$ otherwise. Here $d(j)$ is the distance from vertex j to the nearest bone.

We calculate the Laplacian in equation (3.1) using the cotangent formula (Meyer et al., 2003):

$$\Delta \mathbf{w} = \frac{1}{2} \sum_{k \in N_1(j)} (\cot \alpha_{jk} + \cot \beta_{jk})(\mathbf{p}_j - \mathbf{p}_k),$$

where α_{jk} and β_{jk} are the two angles opposite to the edge in the two triangles sharing the edge ($\mathbf{p}_j$, $\mathbf{p}_k$) and $N_1(j)$ is the set of 1-ring neighbor vertices of vertex j. The 1-ring neighborhood N_1 contains all the vertices that share an edge with vertex j. N_1 and angles α_{jk} and β_{jk} are depicted in Figure 3.2b.

3.4 Posing by direct manipulation

Implementing a direct manipulation scheme (Shneiderman et al., 2016) for posing the 3D model is a necessity to keep the user experience consistent with other tools found in the 3DBear application. The control handles are visualized as red spheres to make them appear distinct and indicate to the user that they are interactable. The visibility of the handles is ensured by making the rest of the model semi-transparent. The drastic visual change also informs the user of the change in modality.

As the sphere mesh approximation yields spheres with both a position and a radius, we use the radius when generating the handles in order to better represent the target model. A minimum scale is enforced to keep the handles interactable. The spheres are also slightly downscaled in order to avoid overlap and help the user to select the desired handle.

Two modes were implemented for the manipulation. In the first one, dragging the handles when they are visible causes the corresponding rig bone to translate. This follows logic similar to how individual models are moved in the application. In contrast to the standard move tool found in 3DBear, the plane the movement is constricted to face the camera, rather than being the horizontal plane. The change was necessary, as many posing actions require vertical movement, whereas object placement is quite naturally restricted to the ground.

The full 3D freedom in translation is provided in two ways. First, while engaged in a translate action, the user is able to “push” and “pull” the handle by moving the AR device forwards or backwards. This is a natural way to introduce the third axis of control, but might be difficult to discover for users. The alternative way to perform a move depth-wise is to change the perspective. By walking around the model, the space of the available actions is rotated along with the camera. This leads to a rather natural physically inspired interface that is hopefully intuitive for the end users.

The second mode causes the selected handle to rotate around the camera Z-axis, pivoting around its parent. Again, the full three degrees of freedom is provided by the user being able to walk around and view the model from different directions. This is at first a less intuitive mode of operation, but the real world parallels suggest that it is worth investigating. In animals with skeletons, the joints tend to rotate rather than extend, much less translate sideways. For example, if an arm is extended out and the wrist is moved half a meter in any direction while keeping the elbow from turning or moving otherwise, a broken forearm is the only possible result. The combination of these two modes allow both for shape changing operations by translating and adjusting the pose by rotating the joints.

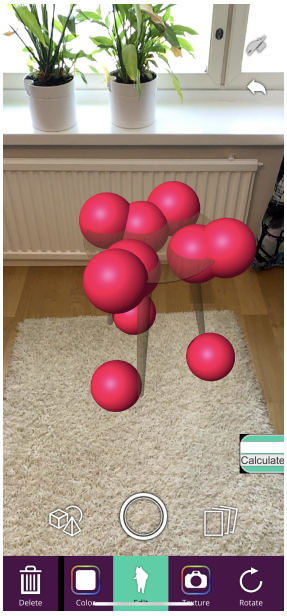
Chapter 4

Examples of application

The approach outlined in Chapter 3 was tested by building three simple scenes using an Apple iPhone XS Max running a custom build of the 3DBear application. The first scene consists of a stool from the Home collection in 3DBear. The unedited stool is shown in Figure 4.1a. The tool was configured to provide 10 control handles. After a very brief processing time, the tool provided the requested amount of control handles. The resulting handles are shown in Figure 4.1b.

As can be seen in Figure 4.1b, the points correspond to the shapes of the stool in a natural way. The points at the top follow along the outer rim of the seat and there is one control point in each end of the three feet, which are the most natural points for controlling the placement. The top of the stool is crowded with a total of seven control points in close vicinity to each other. Selecting and moving each point proved possible due to the fact that the user is able to get closer and further away by moving the mobile device in real life. A handful of translation operations were performed on the control points resulting with the modified model shown in Figure 4.1c. Several peculiarities, that violate the least surprise principle from the user's point of view, were discovered in the deformed object.

(a) Unedited



(b) With control handles

(c) After editing

Figure 4.1: 3D model of a stool in augmented reality

(a) Stretched texture

(b) Hard angles in the leg

(c) Rotated legs cause less artifacts

Figure 4.2: Translating the handles of the stool can cause artifacts

Figure 4.2a shows a close up of the top of the stool, where the model was deformed by stretching. The blurry, diagonal lines are a result of texture stretching. The blurriness is caused by the original artist-created texture being stretched over a larger area than intended. In addition, diagonal cut in the uniform wood grain pattern makes it very visible that the geometry has been altered.

Figure 4.2b shows a close-up of one of the legs in the stool. The midpoint shows two distinct cutoffs in the influence function of the two control handles controlling the leg. At the bottom, the position of the leg's vertices is fully determined by the lower control handle. The same happens at the top with the higher control handle. In the middle, there appears to be a linear transition bridging the differing positions of the control handles. To contrast the harsh angles in Figure 4.2b, another variation of the stool was created by rotating the legs instead of translating them. As can be seen in Figure 4.2c, this version shows much less unwanted deformation.

Another built example scene includes a 3D model of a Tyrannosaurus rex. Figure 4.3a shows the Tyrannosaurus in its default pose, which looks somewhat out of place and static. A more natural pose is shown in Figure 4.3b. This pose was created in 3DBear using 20 handles with the goal of showing the Tyrannosaurus terrorizing the neighborhood. Slight adjustments to the limbs and head were done to help bring life and believability to the otherwise dull scene. Figure 4.3c shows the Tyrannosaurus eating a lamb in order to showcase that editing actions can be done purposefully.

The third example features an animated 3D model of a Pterodactyl. Figure 4.4 depicts the original flight animation of the Pterodactyl flying. The existing animation rig is removed and a new one is generated using the tool. The resulting Figure 4.5 shows the Pterodactyl in a pose created using 20 handles, landed on the ground.

(a) Default pose

(b) A more natural standing pose

(c) Eating a lamb

Figure 4.3: Tyrannosaurus in various poses

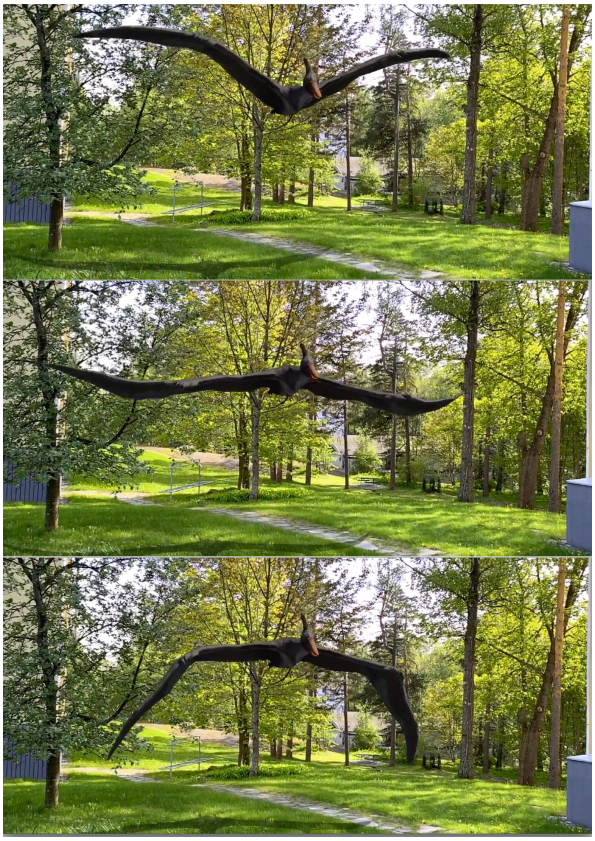


Figure 4.4: Three frames of the original professionally created Pterodactyl flight animation

Figure 4.5: An edited Pterodactyl model showing the reptile on the ground

Chapter 5

Evaluation and discussion

5.1 Analysis of the example scenes

The results demonstrated in Chapter 4 show promise. As mentioned in the motivation for this thesis in Chapter 1, the current tooling on the market lacks an easy tool for 3D modeling on a mobile device. The approach developed in this thesis provides an example for application developers, hopefully influencing the future versions of modeling applications to allow anyone to edit models in an approachable way.

The technology adoption model (Davis, 1989) suggests that in order for new information technology to be accepted and adopted by users, two criteria need to be clear to the user: the technology needs to be useful and perceived to be easy to use. As shown in Chapter 4, giving the user the ability to pose 3D models as they wish allows them to build meaningful augmented reality scenes. Although previous tooling has been useful, this approach allows the user to build something that would have required help from a 3D artist.

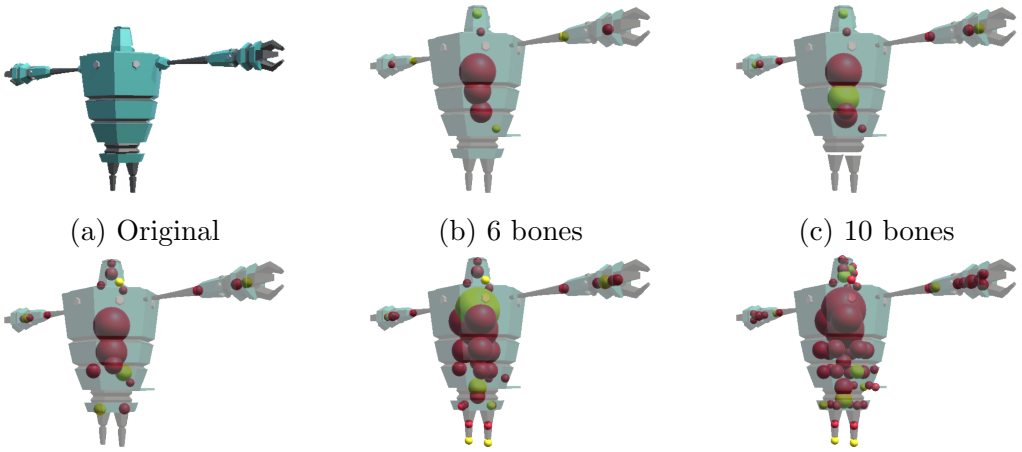
The second requirement for adoption is that the users need to perceive the technology to be easy to use. The tool built in this thesis functions on a proof-of-concept level, and requires more development work in order to fully

be embraced by casual users. As is evident from the shortcomings highlighted in Chapter 4, there are still issues to overcome before the method can reasonably be expected to be adopted en-masse. For example, the texture stretching caused by a geometry configuration that was unexpected by the original artist, shown in Figure 4.2a, is a challenging problem to address. The current iteration of the tool already allows for large modifications to the geometry if the object in question has a monotone texture. For objects with a regular micro-pattern, there are very promising developments (Portilla and Simoncelli, 2000; Wei and Levoy, 2000). The shortcoming demonstrated by the wood grain in Figure 4.2a also contains a macro-pattern in the nearly-parallel stripes, which is a challenging problem to overcome. Efros and Freeman (2001) present a promising quilt-based synthesis method that retains regular patterns. Gatys et al. (2015) show that there is ongoing neural network development work into texture synthesis with macro-patterns. However, the use case presented in this thesis expects near real-time feedback for the user, while a convolutional neural network texture generation algorithm might be out of reach for a mobile processor acting in a real-time environment.

One of the key components in a successful automatic rigging tools is the skinning component. In this thesis, the used skinning algorithm was chosen for its stability and ease of implementation. The shortcomings of the algorithm can be seen in Figure 4.2b, where the skinning weights are distributed with an artificially clear-cut drop-off that resembles a clamped linear function. The state-of-the-art automated skinning algorithms can handle bone weight interpolation in a smoother manner (Le and Deng, 2012), spread weights across volumes rather than just surfaces (Dionne and de Lasa, 2013) and ensure that seemingly unrelated bones do not contribute to the position of the vertices (Jacobson et al., 2011). Incorporating some or even all of these improvements into the skinning algorithm used may make the tool more intuitive and easier to approach.

5.2 Quality and performance analysis of the rigging approach

In this section, we further analyze the rig generation process to assert the correct functioning of the approach. Figures 5.1 and 5.2 show the generated bones at different levels of detail for two 3D models, a robot and a Tyrannosaurus rex. An image of the original model is included for both for clarity. The yellow colored spheres denote root nodes in the bone hierarchy and the red spheres are non-root bones.

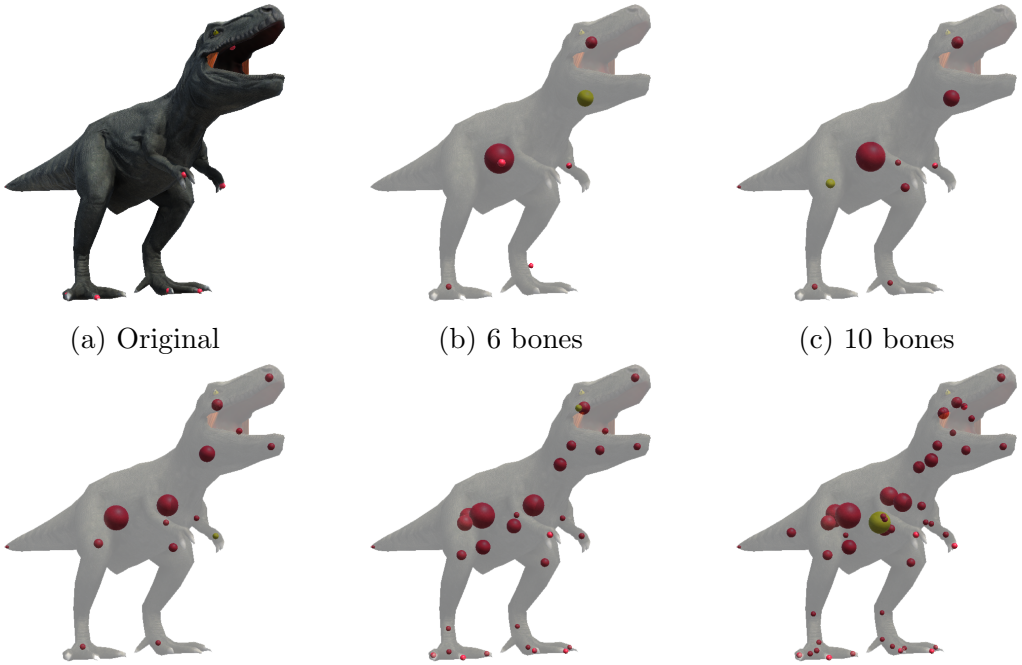


(a) Original (b) 6 bones (c) 10 bones

(d) 20 bones (e) 50 bones (f) 100 bones

Figure 5.1: A 3D model representing a robot

Both examples show that the rig generation algorithm places bones along the entire volume of the targeted 3D model. As can be seen in the legs of the robot in Figures 5.1b and 5.1c, the algorithm can leave large parts of the model completely without a bone. On the other hand, Figures 5.1f and 5.2f show a very dense distribution of bones, making the selection and manipulation of an individual handle cumbersome and error prone. The best bone count is situational, with lower and higher counts having different tradeoffs. This is where the user's capability to configure different resolutions is important.



(a) Original (b) 6 bones (c) 10 bones

(d) 20 bones (e) 50 bones (f) 100 bones

Figure 5.2: A 3D model representing a Tyrannosaurus rex

Figures 5.3a to 5.3c show the performance of the bone heat skinning algorithm when applied on the Tyrannosaurus model. Figure 5.3a shows a natural segmentation of the model into its semantic parts: legs, feet, arms, tail, body, neck and head. Figure 5.3b with its larger number of bones has more blended weights, which denote vertices controlled by several bones. This leads to a more nuanced posing process, but requires the user to do several manipulations in order to see the desired change. Smaller details are editable at this resolution. For example, the individual toes each have a bone assigned to them for fine tuning of the pose. Opening and closing the jaw is also possible. Figure 5.3c shows the influence of an individual bone from the rig containing a total of 10 bones. Although the bone heat diffusion algorithm does not guarantee the locality of the weights, in this case the weights are highly local.

(a) 10 bones

(b) 50 bones

(c) The influence of an individual bone out of a rig with ten bones, shown in red

Figure 5.3: The influence of different bones visualized with separate colors

Algorithm runtime benchmarks for the robot and the Tyrannosaurus models for different target bone counts are listed in Table 5.1. The benchmarks were ran on a 2,7 GHz Intel Core i7 processor. As can be seen in the table, the majority of the algorithm's runtime is due to the rig generation using sphere meshes and the bone heat skinning process. The runtime grows as a function of the bone count, which is mostly caused by the skinning process running longer. This is expected, as the skinning algorithm needs to do a specific amount of work for each bone, while the sphere mesh approximation algorithm actually stops earlier and iterates less for larger bone counts.

Table 5.1: Benchmarks for the runtime of the implemented approach (in seconds)

	Robot (# of bones)					Tyrannosaurus (# of bones)				
Phase	6	10	20	50	100	6	10	20	50	100
Pre-processing	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Rig generation	0.39	0.38	0.38	0.40	0.47	0.63	0.67	0.63	0.63	0.66
Skinning	0.49	0.53	0.60	0.75	1.05	0.37	0.41	0.46	0.58	0.93
Post-processing	0.00	0.01	0.01	0.02	0.04	0.01	0.01	0.01	0.02	0.03
Total	0.89	0.92	0.99	1.18	1.53	1.00	1.08	1.10	1.23	1.63

Chapter 6

Conclusion

The objective of this thesis was to develop an approach for novice friendly 3D modeling on a mobile device. The chosen approach for automatic rig generation used a spherical quadratic error metric to approximate the 3D model, yielding a naturally multi-resolution control rig. The generated rig was then attached to the original 3D model using a technique called bone heat skinning.

The need for the developed approach came from the limitations present in the current modeling tools available on the mobile market. Using practical examples and qualitative analysis, the approach was shown to at least partially meet the need of novice users. The built example scenes were previously impossible to create without the help of professional 3D artists.

In the approach, the rig generation and skinning components were based on previous research by Baran and Popović (2007) and Thiery et al. (2013). The novel contribution in this thesis was to incorporate these methods into the augmented reality context. In the process, inventing an appropriate interaction model was a necessity. The approach used a simple direct manipulation scheme for interacting with the processed models.

As a part of the evaluation process, the implemented approach was also integrated to the augmented reality application developed by 3DBear Oy. Both the resulting scenes and the level of interaction fit the application well. As is common in a proof-of-concept level implementation, the user experience showed shortfalls. Investigating and fixing these would be a natural way to continue the research into the topic of this thesis.

Further research into improving the operation of the approach is warranted. For instance, allowing the user to seamlessly switch between different editing resolutions without losing work would ease more complex tasks. This can be achieved by combining a system such as the one presented by Kobbelt et al. (1998) and committing the pose between changes of resolution. A further improvement could be achieved by using an efficient in-memory representation of the different intermediate steps of the generated rig (De Floriani et al., 2004). It would be beneficial to also guide the resolution selection towards interesting areas through an importance criterion, such as the difference of collapse operation priorities between steps. A final improvement suggested here is incorporating an inverse kinematics setup (Girard and Maciejewski, 1985) for the rig. Inverse kinematics allows calculating a total pose based on the pose of a single bone. Especially for deeper bone hierarchies, grabbing a single control handle and seeing the others follow in a natural manner could be magical.

Appendix A

Golden rules

Listing A.1: Eight Golden Rules (Shneiderman et al., 2016)

- *Strive for consistency.* Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus, and help screens; and consistent color, layout, capitalization, fonts, and so on, should be employed throughout. Exceptions, such as required confirmation of the delete command or no echoing of passwords, should be comprehensible and limited in number.

- *Seek universal usability.* Recognize the needs of diverse users and design for plasticity, facilitating transformation of content. Novice to expert differences, age ranges, disabilities, international variations, and technological diversity each enrich the spectrum of requirements that guides design. Adding features for novices, such as explanations, and features for experts, such as shortcuts and faster pacing, enriches the interface design and improves perceived quality.

- *Offer informative feedback.* For every user action, there should be an interface feedback. For frequent and minor actions, the response can be modest, whereas for infrequent and major actions, the response should be more substantial. Visual presentation of the objects of interest pro-

vides a convenient environment for showing changes explicitly.

- *Design dialogs to yield closure.* Sequences of actions should be organized into groups with a beginning, middle, and end. Informative feedback at the completion of a group of actions gives users the satisfaction of accomplishment, a sense of relief, a signal to drop contingency plans from their minds, and an indicator to prepare for the next group of actions. For example, e-commerce websites move users from selecting products to the checkout, ending with a clear confirmation page that completes the transaction.

- *Prevent errors.* As much as possible, design the interface so that users cannot make serious errors; for example, gray out menu items that are not appropriate and do not allow alphabetic characters in numeric entry fields. If users make an error, the interface should offer simple, constructive, and specific instructions for recovery. For example, users should not have to retype an entire name-address form if they enter an invalid zip code but rather should be guided to repair only the faulty part. Erroneous actions should leave the interface state unchanged, or the interface should give instructions about restoring the state.

- *Permit easy reversal of actions.* As much as possible, actions should be reversible. This feature relieves anxiety, since users know that errors can be undone, and encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data-entry task, or a complete group of actions, such as entry of a name-address block.

- *Keep users in control.* Experienced users strongly desire the sense that they are in charge of the interface and that the interface responds to their actions. They don't want surprises or changes in familiar behavior, and they are annoyed by tedious data-entry sequences, difficulty in obtaining necessary information, and inability to produce their desired result.

- *Reduce short-term memory load.* Humans' limited capacity for information processing in short-term memory (the rule of thumb is that people can remember "seven plus or minus two chunks" of information) requires that designers avoid interfaces in which users must remember information from one display and then use that information on another display. It means that cellphones should not require reentry of phone numbers, website locations should remain visible, and lengthy forms should be compacted to fit a single display.

References

M. Akçayır and G. Akçayır. Advantages and challenges associated with augmented reality for education: A systematic review of the literature. *Educational Research Review*, 20:1–11, 2017.

T. Akenine-Moller, E. Haines, and N. Hoffman. *Real-time rendering*. AK Peters/CRC Press, 2018.

Apple Inc. Understanding world tracking in ARKit. Webpage, 2018a. `https://developer.apple.com/documentation/arkit/understanding_world_tracking_in_arkit`. Accessed November 18, 2018.

Apple Inc. About augmented reality and arkit. Webpage, 2018b. `https://developer.apple.com/documentation/arkit/about_augmented_reality_and_arkit`. Accessed June 3, 2018.

O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee. Skeleton extraction by mesh contraction. *ACM transactions on graphics (TOG)*, volume 27, page 44. ACM, 2008.

R. T. Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.

J. Bacca, S. Baldiris, R. Fabregat, S. Graf, et al. Augmented reality trends in education: a systematic review of research and applications. 2014.

I. Baran and J. Popović. Automatic rigging and animation of 3d characters. *ACM Transactions on Graphics (TOG)*, 26(3):72, 2007.

A. Beane. *3D animation essentials*. John Wiley & Sons, 2012.

G. Bharaj, T. Thormählen, H.-P. Seidel, and C. Theobalt. Automatically rigging multi-component characters. *Computer Graphics Forum*, volume 31, pages 755–764. Wiley Online Library, 2012.

F. D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, pages 319–340, 1989.

L. De Floriani, P. Magillo, E. Puppo, and D. Sobrero. A multi-resolution topological representation for non-manifold meshes. *Computer-Aided Design*, 36(2):141–159, 2004.

O. Dionne and M. de Lasa. Geodesic voxel binding for production character meshes. *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 173–180. ACM, 2013.

A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM, 2001.

W. Förstner. A framework for low level feature extraction. *European Conference on Computer Vision*, pages 383–394. Springer, 1994.

L. Gatys, A. S. Ecker, and M. Bethge. Texture synthesis using convolutional neural networks. *Advances in neural information processing systems*, pages 262–270, 2015.

M. Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures. *ACM SIGGRAPH Computer Graphics*, volume 19, pages 263–270. ACM, 1985.

Google Inc. ARCore fundamental concepts. Webpage, 2018. `https://developers.google.com/ar/discover/concepts`. Accessed November 18, 2018.

C. Harris and M. Stephens. A combined corner and edge detector. *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

A. Henrysson and M. Billinghurst. Using a mobile phone for 6 dof mesh editing. *Proceedings of the 8th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: design centered HCI*, pages 9–16. ACM, 2007.

K. P. Herndon, A. van Dam, and M. Gleicher. The challenges of 3d interaction: a chi'94 workshop. *ACM SIGCHI Bulletin*, 26(4):36–43, 1994.

U. Hinrichs and S. Carpendale. Gestures in the wild: studying multi-touch gesture sequences on interactive tabletop exhibits. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3023–3032. ACM, 2011.

W. A. Hoff, K. Nguyen, and T. Lyon. Computer-vision-based registration techniques for augmented reality. *Intelligent Robots and Computer Vision XV: Algorithms, Techniques, Active Vision, and Materials Handling*, volume 2904, pages 538–549. International Society for Optics and Photonics, 1996.

R. L. Holloway. Registration error analysis for augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):413–432, 1997.

J. F. Hughes, A. Van Dam, J. D. Foley, M. McGuire, S. K. Feiner, D. F. Sklar, and K. Akeley. *Computer graphics: principles and practice*. Pearson Education, 2014.

A. Jacobson. Part ii: Automatic skinning via constrained energy optimization. *SIGGRAPH Course*, 2014:1–28, 2014.

A. Jacobson, I. Baran, J. Popovic, and O. Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78–1, 2011.

B. E. John and D. E. Kieras. The goms family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(4):320–351, 1996.

T. Jung, M. C. tom Dieck, H. Lee, and N. Chung. Effects of virtual reality and augmented reality on visitor experiences in museum. *Information and communication technologies in tourism 2016*, pages 621–635. Springer, 2016.

S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics (TOG)*, volume 22, pages 954–961. ACM, 2003.

H. Kaufmann, K. Steinbügl, A. Dünser, and J. Glück. General training of spatial abilities by geometry education in augmented reality. *Annual Review of CyberTherapy and Telemedicine: A Decade of VR*, 3:65–76, 2005.

L. Kerawalla, R. Luckin, S. Seljeflot, and A. Woolard. "making it real": exploring the potential of augmented reality for teaching primary school science. *Virtual reality*, 10(3-4):163–174, 2006.

L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 105–114. ACM, 1998.

B. H. Le and Z. Deng. Smooth skinning decomposition with rigid bones. *ACM Transactions on Graphics (TOG)*, 31(6):199, 2012.

D.-T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980.

V. Lepetit, P. Fua, et al. Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and Trends® in Computer Graphics and Vision*, 1 (1):1–89, 2005.

J. P. Lewis, M. Cordner, and N. Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. *Proceedings of the 27th annual conference on Computer graphics and interactive*

techniques, pages 165–172. ACM Press/Addison-Wesley Publishing Co., 2000.

A. v. Luikov. *Analytical heat diffusion theory*. Elsevier, 1968.

N. Magnenat-Thalmann, R. Laperrire, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. *In Proceedings on Graphics interface'88*. Citeseer, 1988.

D. E. McGovern. Experiences in teleoperation of land vehicles. 1989.

M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. *Visualization and mathematics III*, pages 35–57. Springer, 2003.

Microsoft Inc. Microsoft hololens. Webpage, 2019. `https://www.microsoft.com/en-CY/hololens`. Accessed June 2, 2019.

H. P. Moravec. Techniques towards automatic visual obstacle avoidance. 1977.

J. Nielsen. Usability inspection methods. *Conference companion on Human factors in computing systems*, pages 413–414. ACM, 1994.

D. Pflughoeft. A heap-based C# priority queue optimized for A* pathfinding. Webpage, 2013. `http://www.blueraja.com/blog/356/a-heap-based-c-priority-queue-optimized-for-a-pathfinding`. Accessed April 25, 2019.

J. Portilla and E. P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1):49–70, 2000.

G. Schall, D. Wagner, G. Reitmayr, E. Taichmann, M. Wieser, D. Schmalstieg, and B. Hofmann-Wellenhof. Global pose estimation using multi-sensor fusion for outdoor augmented reality. *2009 8th ieee international symposium on mixed and augmented reality*, pages 153–162. IEEE, 2009.

D. Schmalstieg and T. Hollerer. *Augmented reality: principles and practice*. Addison-Wesley Professional, 2016.

B. Shneiderman, C. Plaisant, M. Cohen, S. Jacobs, N. Elmqvist, and N. Diakopoulos. *Designing the user interface: strategies for effective human-computer interaction*. Pearson, 2016.

A. Tatnall and B. Davey. Reflections on the history of computer education in schools in victoria. *Reflections on the History of Computing*, pages 243–264. Springer, 2012.

J.-M. Thiery, É. Guy, and T. Boubekeur. Sphere-meshes: shape approximation using spherical quadric error metrics. *ACM Transactions on Graphics (TOG)*, 32(6):178, 2013.

Trimble Inc. Sketchup website. Webpage, 2018. `https://www.sketchup.com/`. Accessed June 3, 2018.

F. Wang and X. Ren. Empirical evaluation for finger input properties in multi-touch interaction. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1063–1072. ACM, 2009.

L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488. ACM Press/Addison-Wesley Publishing Co., 2000.

H.-K. Wu, S. W.-Y. Lee, H.-Y. Chang, and J.-C. Liang. Current status, opportunities and challenges of augmented reality in education. *Computers & education*, 62:41–49, 2013.

S. You and U. Neumann. Fusion of vision and gyro tracking for robust augmented reality registration. *Proceedings IEEE Virtual Reality 2001*, pages 71–78. IEEE, 2001.

S. You, U. Neumann, and R. Azuma. Hybrid inertial and vision tracking for augmented reality registration. *Proceedings IEEE Virtual Reality (Cat. No. 99CB36316)*, pages 260–267. IEEE, 1999.