Sketch-based 3D Modeling in Virtual Reality

Floor Verhoeven

Master Thesis
April 2018

Supervisors:
Dr. Roi Poranne
Prof. Dr. Olga Sorkine-Hornung

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

igl
INTERACTIVE GEOMETRY LAB

Abstract

The recent advancements in virtual reality (VR) have resulted in many new virtual reality applications. One group of applications targets 3D modeling in VR. This thesis addresses the development of a novel sketch-based 3D modeling tool in VR. Sketch-based modeling aims to provide the user with a very intuitive and simple way of creating 3D models, and is based on the way that humans draw 2D shapes with pen and paper.

The developed system is based off the sketch-based modeling tool FiberMesh [Nealen et al. 2007] and essentially brings it to the Oculus Rift. The system is built into the mesh viewer of libigl [Jacobson et al. 2017], which is also ported to the Oculus Rift as part of this thesis. With our software the user can draw an outline, which will then inflate to a rough 3D model. The drawn strokes stay on the model surface and serve as deformation handles. The user can add and remove extra control curves, as well as deforming them. Finally the user can edit the mesh topology by performing cuts or extrusions.

We show that our software is very easy to learn and intuitive, and that novice users can quickly generate 3D models with it. Also we show that this application greatly benefits from the 3D setting that comes with VR, since it allows users to do out-of-plane editing in a faster and more intuitive way.

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

igl
INTERACTIVE GEOMETRY LAB

Bachelor/Master Thesis

3D Modeling in Virtual Reality

Introduction

High-end virtual reality headsets provide a feeling of full immersion in a virtual 3D environment, and enable users to explore and interact with virtual 3D objects by precisely tracking the position and orientation of the user's head and the controller(s) placed in the users' hands. Virtual reality eliminates the cognitively difficult translation between a 2D projection and the actual 3D object that plagues traditional 2D display technology. At the same time, applications must fulfill strict real time requirements in order to avoid motion sickness and other types of discomfort caused by lag. In this thesis we wish to research how various interactive 3D shape modeling and animation algorithms, originally developed for standard display-mouse-keyboard user interface, translate to the virtual reality setting.

Task Description

We will be working with the Oculus Rift VR headset and its touch controllers. All necessary hardware will be made available to the student for the duration of the project. The concrete methodology to explore will be sketch-based modeling (FiberMesh). The challenges in this project are the conversion of the traditional user interface to the VR setting, adaptation of the shape modeling algorithm to work with this new type of input, porting existing geometry processing and modeling code to VR headset, addressing the stringent efficiency requirements, and more.

Here are the suggested steps:

1. Read the FiberMesh paper carefully, and check the papers referenced there as well to make sure everything about the method is clear, discuss with advisor.

2. Implement a functional version of FiberMesh in libigl, targeted at keyboard/mouse or keyboard/stylus interface. The definition of "reasonable" is up for interpretation, but the minimum should be: being able to draw silhouette curves (and inflate), draw/erase additional curves on the surface, grab-and-deform any curve (and the surface shape follows, as described in the paper), smudge/smoothing of curves, extrusion curves. Creating tunnels, sharp features – can be optional although very nice to have. Additional the typical I/O functionality: load/save the created models + curves, and export to common mesh formats.

3. Devise a plan how to port the controls you implemented for desktop/mouse/keyboard into Oculus VR with touch controls, while keeping the same functionality. Try to answer the following questions: How does the UI need to be changed? Can you think of new features or different

features that were not possible or difficult in 2D display mode? Are there things that are harder/unreasonable without a mouse or stylus? Do you need some helper tools/UI in VR, for example a "wallpaper" or "guide" image that you can trace curves from? (this could be useful in 2D as well). Research related literature and ongoing projects in this direction.

4. Port FiberMesh to Oculus VR setup, implementing the UI ideas you developed above. Collaborate with other students working at IGL regarding the interface of libigl and Oculus SDK – by then there should be a reasonable libigl mesh viewer running available.

5. Test your app, invite others to test, think of particular questions/experiments/hypotheses you would like to check.

6. Write thesis and prepare the final presentation (usually takes about 3-4 weeks).

Grading scale

The thesis will be graded using the following scale:

6.0	Work and results are equivalent to publication quality of international workshops/conferences
5.5	Thesis quality significantly exceeds expectations
5.0	Thesis meets expectations
4.5	Thesis partially meets expectations, minor deficits
4.0	Thesis meets minimal quality requirements; it has major deficits and it is clearly below expectations

Prof. Dr. Olga Sorkine-Hornung

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

SKETCH-BASED 3D MODELING IN VIRTUAL REALITY

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
VERHOEVEN	FLOOR

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Chur, 27-03-2018	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Contents

List of Figures

List of Tables

1

Introduction

Digital 3D shape modeling is a field that has received a lot of attention over the past years as a result of increasing rendering possibilities and a growing demand for 3-dimensional digital content due to the digitalization of movie-making, fabrication, engineering, architecture, product design, medicine and many other disciplines. The recent surge in interest for VR (Virtual Reality) has driven the development of better GPUs (Graphics Processing Unit) even more, while also further increasing the demand for digital 3D content. The process of 3D shape modeling has so far been almost exclusively suitable for trained professionals or users with a lot of experience. While 3D modeling software generally comes with an abundance of options and possibilities, it also usually has a steep learning curve. This causes 3D modeling to become inaccessible to novice users who would like to create (simple) 3D models, but who do not have the time to learn how to use these programs.

Sketch-based 3D modeling mostly seeks to simplify the process of 3D modeling in order to make it more accessible for this user group, but also to provide professional users with a means of quickly making rough drafts or simple base models to which details can be added by other software, or to experiment. Its goal is to provide the user with intuitive ways to interact with and edit the mesh that is being modelled. Previous work regarding sketch-based modeling tools has produced multiple software products which present the user with this intuitive method of 3D modeling. However, these have only been made available for use with a PC and mouse or on mobile [Gravity Sketch 2018], and not for usage in VR.

Recently a variety of art creation tools for usage in VR have been created. These applications range from painting in 3D to voxel modeling and 3D sculpting. VR gives artists the possibility to look at what they are creating from a literally new perspective. Directly modeling in 3D can give an improved perception of the proportions of different parts of a model and the relations between them.

The goal of this thesis is to develop a sketch-based 3D modeling program explicitly created for usage in a VR setting. This combines the simplicity and intuitiveness of sketch-based designing with the immersiveness of modeling in actual 3D space and the possibility of viewing the results immediately in 3D. The software is targeted to users that are new to 3D modeling and should therefore be straightforward to use and easy and quick to learn.

Chapter 2 discusses previous work that is done in the fields of sketch-based 3D modeling and give several examples of software for 3D modeling in virtual reality.

In Chapter 3, we give a detailed description of the VR sketch-based 3D modeling program that was developed as part of this thesis, and discuss how it differs from previous work. It describes the algorithms that were used and gives a description of the user interface including the rationale behind the design decisions that were made.

Next, in Chapter 4 we let target users test our software and summarize the feedback that they have given. We also compare the VR sketch-based 3D modeling tool with a version of the software that is developed for non-VR use with a computer and mouse.

Finally, Chapter 5 provides ideas for future improvements on the software and summarizes the findings of this thesis.

2

Related Work

A lot of software for the purpose of 3D modelin exists, with some of the best known ones being Maya, Blender, Autodesk 3ds Max and Zbrush [Maya 2018, Blender 2018, 3ds MAX 2018, ZBrush 2018]. Typically these software let their users manipulate meshes on vertex, edge or face level, resulting in very fine-grained control over the appearance of created models. Although this makes these programs very powerful and versatile for the creation and editing 3D models, they come with a very steep learning curve and are overcomplete for recreational users. This chapter instead focusses on describing some of the more intuitive and easy-to-use sketch-based 3D modeling software, as well as a couple of relatively new software for creating 3D content in VR.

2.1 Sketch-based Modeling

Sketch-based modeling is a modeling technique that aims to transfer the way that people draw shapes with pen and paper to a method of modeling 3D shapes on the PC with a mouse. With the mouse the user draws strokes on the screen, whose interior is subsequently meshed and then inflated to smooth rotund 3D meshes. Typically the user can then edit this initial mesh by specifying additional strokes that for example create extrusions or cut the mesh. The software that was written as part of this thesis also adopts the sketch-based modeling paradigm.

2.1.1 FiberMesh

FiberMesh is a system that allows modeling freeform surfaces by drawing and editing 3D curves [Nealen et al. 2007]. The user-defined curves are used to create a 3D model and stay

present on the model and can be used to edit the geometry. This allows for a very intuitive method of deforming and editing meshes after their initial creation. FiberMesh lets users define curves as smooth or sharp, add and remove control curves on the mesh and pull curves to deform the mesh. Additionally it allows the user to change the mesh topology by cutting parts of the mesh and creating extrusions or tunnels.

Algorithmically FiberMesh depends on two main steps, namely curve deformation and surface optimization. In addition to these two steps, there are also a mesh construction and remeshing step, which only occur after new mesh topology has been created (e.g. after creation, cut or extrusion).

For curve deformation they used a detail-preserving deformation method that combines differential coordinates [Sorkine 2006] with co-rotational methods [Felippa 2007]. A sequence of linear least-squares problems is solved, while satisfying the user-defined positional constraints on the drawn curves. The rotation matrices are explicitly represented as free variables, as they cannot be derived from the curves which are nearly collinear. In order to accommodate for large linear rotations, the gross rotation is computed by iteratively concatenating small delta rotations that are each obtained by solving a linear system. The linear system that is solved in each step can be seen in equation 2.1.

$$\arg\min_{\mathbf{v},\mathbf{r}} \left\{ \sum_i \left\| \mathbf{L}\left(\mathbf{v}_i\right) - \mathbf{r}_i \mathbf{R}_i \delta_i \right\|^2 + \sum_{i \in C_1} \left\| \mathbf{v}_i - \mathbf{v}_i^{'} \right\|^2 + \sum_{i,j \in E} \left\| \mathbf{r}_i \mathbf{R}_i - \mathbf{r}_j \mathbf{R}_j \right\|_F^2 + \sum_{i \in C_2} \left\| \mathbf{r}_i \mathbf{R}_i - \mathbf{R}_i^{'} \right\|_F^2 \right\} \tag{2.1}$$

Here $\mathbf{L}(\cdot)$ is the differential operator, $\mathbf{v}_i$ is the coordinates of vertex i, $\| \cdot \|_F$ is the Frobenius norm, E is the set of curve edges and C_1 and C_2 are the sets of constrained vertices, primed values are constraints, $\mathbf{R}_i$ represents the gross rotation (in the deformed curve) corresponding to vertex i obtained from the previous iteration, and finally $\mathbf{r}_i$ is a linearized incremental rotation for vertex i given by a skew symmetric matrix with 3 unknowns,

$$\mathbf{r}_i = \begin{bmatrix} 1 & -r_{iz} & r_{iy} \\ r_{iz} & 1 & -r_{ix} \\ -r_{iy} & r_{ix} & 1 \end{bmatrix}.$$

Rotations are updated by setting them to $\mathbf{R}_i = \mathbf{r}_i \mathbf{R}_i$ and orthonormalizing the result using polar decomposition [Fu et al. 2007]. In the iterative process of estimating rotations, they use first order differentials (L_0), and for computing the final vertex positions using this estimated rotations they use the second order differential (L_1).

$$L_0 = \mathbf{v}_i - \mathbf{v}_{i-1}, \quad L_1 = \mathbf{v}_i - \frac{1}{|N_i|} \sum_{j \in N_i} \mathbf{v}_j.$$

For surface optimization, FiberMesh solves 3 sparse linear systems in order to compute a smooth mesh surface that adheres to the user-defined constraint curves. The first system solves

for a set of smoothly varying Laplacian magnitudes $\{c_i\}$ which approximate the scalar mean curvature values. The least-squares minimization that it solves is as follows:

$$\arg\min_{c} \left\{ \sum_i \left\| \mathbf{L}\left(c_i\right) \right\|^2 + \sum_i \left\| c_i - c_i^{'} \right\|^2 \right\} \tag{2.2}$$

Where $\mathbf{L}\left(\cdot\right)$ denotes the discrete graph Laplacian, which is independent of the exact mesh geometry, allowing us to reuse it in multiple iterations. In the first iteration, the target Laplacian magnitudes are only set for the constrained curves by using the scalar mean curvature along the curve.

To obtain a geometry that satisfies these target Laplacian magnitudes, Nealen et al. use the uniform Laplacian as an estimator of the integrated mean curvature normal, which is computed by $\delta_i = A_i \cdot c_i \cdot \mathbf{n}_i$, where A_i is an area estimate for vertex i, c_i is the target Laplacian magnitude and $\mathbf{n}_i$ is the estimated normal from the current face normals. However the uniform Laplacian is not an accurate estimator of the integrated mean curvature normal when the incident edges to a vertex are not of equal length. To solve for this problem without using a geometry dependent discretization and thus avoiding recomputing the matrix for every iteration, they prescribe target edge vectors in an attempt to achieve equal edge length. This is done by solving the following linear system (which uses the same matrix as the system that solves for target Laplacian magnitudes) to obtain a smooth set of target average edge lengths e_i:

$$\arg\min_{e} \left\{ \sum_i \left\| \mathbf{L}\left(e_i\right) \right\|^2 + \sum_i \left\| e_i - e_i^{'} \right\|^2 \right\} \tag{2.3}$$

Again the first iteration is performed with only the edge lengths along the constrained curve. The target average edge lengths are then used to compute target edge vectors for a subset B of mesh edges (in the first iteration this subset only contains edges along the constrained curves, and afterwards it contains all edges incident to the constrained curves) as follows:

$$\eta_{ij} = \left(e_i + e_j\right)/2 \cdot \left(\mathbf{v}_i - \mathbf{v}_j\right) / \left\| \mathbf{v}_i - \mathbf{v}_j \right\|. \tag{2.4}$$

These target edge vectors are then used to solve a linear system that gives the updated vertex positions as follows:

$$\arg\min_{\mathbf{v}} \left\{ \sum_i \left\| \mathbf{L}\left(\mathbf{v}_i\right) - \delta_i \right\|^2 + \sum_{i \in C} \left\| \mathbf{v}_i - \mathbf{v}_i^{'} \right\|^2 + \sum_{(i,j) \in B} \left\| \mathbf{v}_i - \mathbf{v}_j - \eta_{ij} \right\|^2 \right\} \tag{2.5}$$

The systems for solving for target Laplacian magnitudes, average edge lengths and optimal vertex positions are solved iteratively until convergence (approximately 5-10 iterations). Since only a geometry independent discretization of the Laplacian is used, the system matrix can be reused between iterations, until the mesh topology is changed (for example by a cutting action). Because of this, the slow matrix factorization only needs to happen once for a given mesh topology, resulting in a fast algorithm that allows for interactive rates.

2.2 3D Modeling in Virtual Reality

Over the last couple of years, plenty of VR applications have been published that involve creating some type of digital 3D content. The software can roughly be split into 2 categories, namely for creating 3D paintings and for creating 3D models (assembled from a structure of vertices, edges and faces). Whereas the former category focusses on the artistic side of 3D content and usually does not result in work that can be exported to some standard format, the latter category more aims to provide a tool for producing 3D models that can be exported and then used in some other VR or 3D application. This section describes a variety of these 3D virtual reality applications, their interfaces and what kind of content users can make with them.

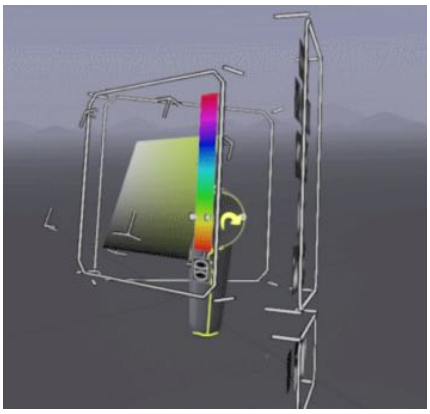
2.2.1 Google Tilt Brush

Google's Tilt Brush is available for the Oculus Rift and HTC Vive and allows the user to create room-scale art in the style of 3D paintings. The software provides different types of tools and brushes, including special effects tools like a fire or stars brush. Users cannot export their creations to typical model formats such as .obj or .off, but they can turn them into a GIF or upload them to Google Poly where other users can explore and enjoy their work. Figure 2.1 (a) shows a screenshot of example 3D artwork that has been made with Google Tilt Brush (the original work has stars that change their light intensity and seem to twinkle). In order to give the user easy access to the large set of different painting tools, Tilt Brush has created an embedded menu in the form of a painters palette that is attached to the controller in the user's non-dominant hand. Several of the menus allow for browsing through further submenus. The dominant hand is used to select a tool from the menu and paint with it. Figures 2.1 (b) and 2.1 (c) show two examples of what the hand-held palette menus looks like and you can see that the user can twist his hand in order to view more menu options on the backside of the palette.

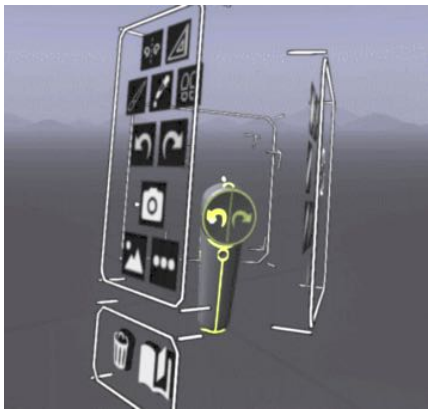
2.2.2 Oculus Medium

Oculus Medium is a 3D sculpting tool that is only available for the Oculus Rift. Conceptually the user can paint with volumetric brushes, much like sculpting with clay. For instance the user will draw a simple circle and this will result in a donut shape. With several tools the user can sculpt away or add extra clay, as well as assigning multiple colors to the surface. Unlike Google Tilt Brush, Oculus Medium does allow the user to store created models in .obj format, so they can be exported for use in other programs. The interface is quite intuitive since it shows a very strong connection to the method of creating 3D shapes with the use of clay in real life which most users have experience with. Users can quickly create rough shapes and refine them by adapting the brush size and adding small-scale details. The dominant hand of the user serves as the so-called "Tool hand" which can be used to sculpt with, while the non-dominant hand serves as the "Support hand" which can be used to open up several hand-held menus. Figures 2.2 (a) and 2.2 (b) show examples of complicated 3D models that have been made in Oculus Medium. One of the unique features of Oculus Medium is the functionality to create and use stamps, which helps easily create interesting textures on the models like the fur and grass in Figure 2.2 (a).

(a)



(b)



(c)

Figure 2.1: *Google Tilt Brush.* (a) *Recreation of Van Gogh's Starry Night (moving version available at https://poly.google.com/view/e-Zqenw7Dui).* (b) *Left-side view of the "hand-held" user interface of Google Tilt Brush.* (c) *Right-side view of the "hand-held" user interface of Google Tilt Brush.*

2.2.3 MasterpieceVR

MasterpieceVR brings a combination of the design paradigms used by Oculus Medium and Google Tilt Brush, letting the user mix volume sculpting with 3D painting. It is available for Oculus Rift, HTC Vive and Windows Mixed Reality, and for each of them it requires the accompanying controllers for input. One of MasterpieceVR's biggest selling points is that it allows multiple users to work on one model simultaneously, allowing interactive collaboration with direct feedback between artists. Like Oculus Medium, MasterpieceVR lets the user export models (including vertex colors) to .obj, as well as .fbx format. Additionally, users can load reference images or models into the program, which can greatly simplify modeling. MasterpieceVR implements menus in a similar fashion to Google Tilt Brush and Oculus Medium, with a menu of icons held in the user's non-dominant hand. Compared to these two, MasterpieceVR however has a larger and less intuitive menu layout. Figure 2.3 (a) shows a complex model created in MasterpieceVR and Figure 2.3 (b) shows the menu interface.

(a) (b)

(c) (d)

Figure 2.2: *Oculus Medium.* (a) *Model of a furry monster created in Oculus Medium by artist Goro Fujita.* (b) *Model of an African figure created in Oculus Medium by artist Goro Fujita.* (c) *Sculpting view in Oculus Medium.* (d) *Menu view in Oculus Medium.*

2.2.4 Google Blocks

Google Blocks is available for Oculus and HTC Vive and is Google's software for creating 3D objects in VR. From the programs mentioned in this section, Google Blocks seems the most similar to professional non-VR based modeling software like Blender [Blender 2018] or Maya [Maya 2018]. Users can start with several predefined shapes like spheres, cubes or cones and can edit the models on face, edge or vertex level. While this gives the user a lot of artistic freedom and extra modeling possibilities, it also makes its usage a lot less intuitive than the other VR 3D modeling software that is available. Although the user has low-level control, the created meshes are very low-poly compared to the models created with for example Oculus Medium. The user does not have control over the number of polygons, which results in less realistic and smooth models. Again the created models can be exported as .obj file. Figures 2.4 (a) and 2.4 (b) show a simple model of an anglerfish created in Google Blocks, and the simple Google Blocks interface.

(a) (b)

Figure 2.3: *MasterpieceVR.* (a) *Model of a warrior created in MasterpieceVR by artist Vladimir Ilic.* (b) *Interface of MasterpieceVR (model by Mary Ellis).*

(a) (b)

Figure 2.4: *Google Blocks.* (a) *Model of an anglerfish created in Google Blocks.* (b) *Interface of Google Blocks.*

3

System Description

This chapter gives a description of the VR sketch-based modeling software that was developed as part of this thesis and we have aptly named SketchMeshVR. The software aims to provide users with a simple and intuitive sketch-based 3D modeling tool within a VR environment. Specifically the software is developed for use with the Oculus Rift including one Touch controller and targets users that are new to 3D modeling. Compared to the VR modeling software that was discussed in Section 2.2, our system provides a novel method for creating 3D models. Whereas the existing modeling tools for VR utilize volumetric brushes or predefined 3D shapes, our tool lets the user define the outline of a 3D shape and then fills in the inside of this silhouette.

3.1 Algorithms

Algorithmically, SketchMeshVR largely follows the FiberMesh software that was described in Section 2.1.1. Similar to FiberMesh, users can draw strokes that define the 3D mesh surface, and these drawn strokes stay on the model surface to later function as deformation handles. Users can create an initial mesh, cut parts of the mesh away, create extrusions, add and remove additional control curves and deform control curves on the mesh. Unlike in FiberMesh, SketchMeshVR does not give users the option to choose between smooth and sharp constraint curves and instead all curves are treated as smooth. The functionality to create tunnels through meshes is also not yet supported in SketchMeshVR.

SketchMeshVR is built inside the framework of libigl [Jacobson et al. 2017] and uses Oculus C++ SDK in order to render everything to the Oculus Rift. The standard overlay menu that is included in libigl is disabled since the concept of overlays does not port well to the VR setting. Overlays are displayed too close to the eye, making it very difficult for the eye to focus on them,

which results in a highly unpleasant user experience.

The system is multithreaded such that the VR display keeps on being updated while the mesh computations are being executed. When multithreading is not enabled the screen will freeze during the mesh computations, and therefore will stop updating when the user moves his head, breaking immersion and likely inducing motion sickness.

Due to the direct availability of the actual 3D coordinates, some changes were made in comparison to SketchMeshVR's non-VR counterpart FiberMesh. These implementation differences are discussed below.

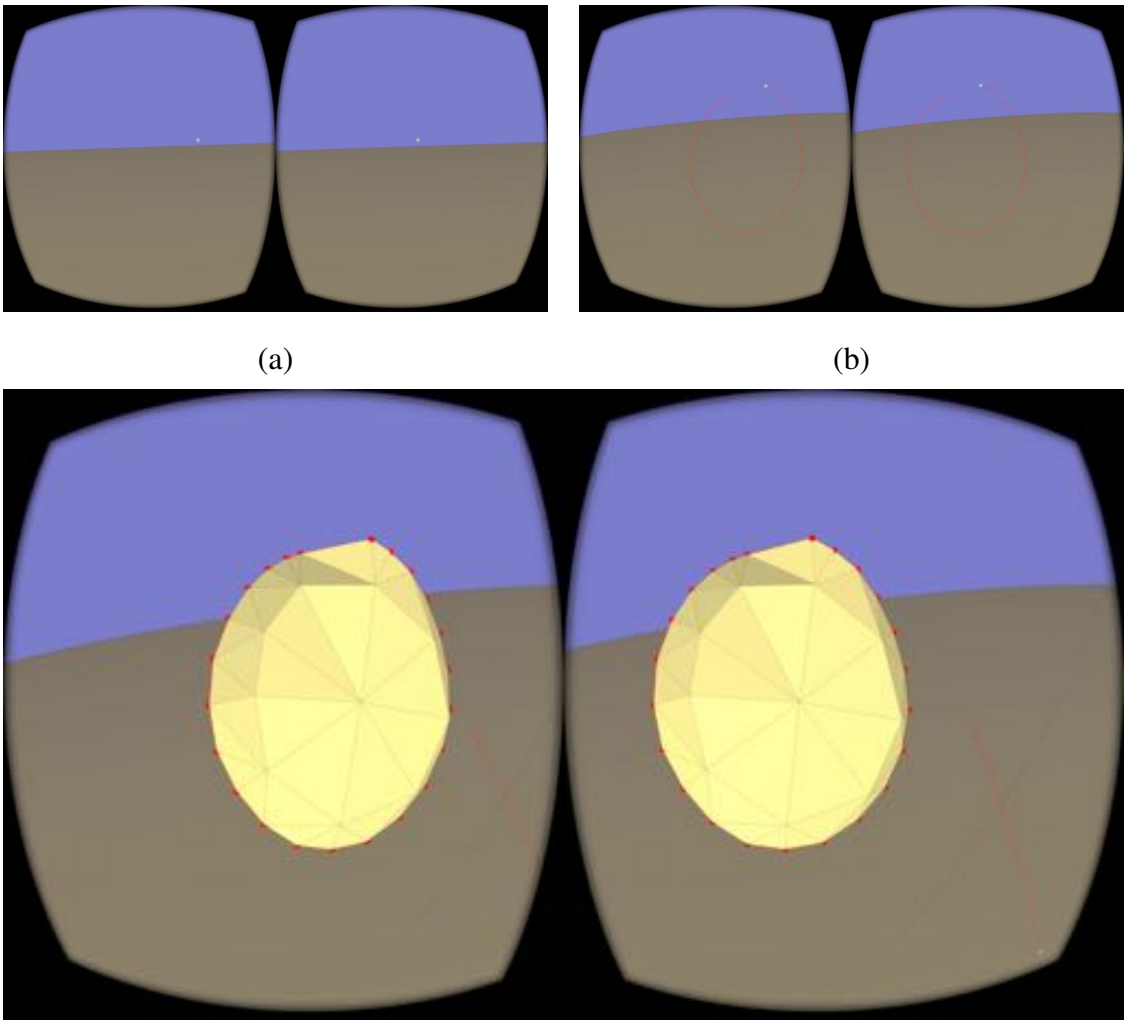
3.1.1 Drawing

The process of drawing the initial mesh curve stays mostly unchanged when converting to virtual reality. However, whereas the non-VR version in drawing mode starts by converting mouse positions to screen coordinates and then unprojects these to 3D coordinates with a z-value of 0, SketchMeshVR takes the actual 3D coordinates of the drawn stroke and in real time projects them to 2D only to allow for triangulation. The depth values that result from this projection are stored and after triangulation used to unproject the stroke points back to their original 3D positions. The interior mesh vertices are unprojected with a depth value that equals the mean depth value of the stroke points in order to get their x- and y-value. Their z-coordinate is determined by taking the mean z-coordinate of the stroke points, and adding an offset along the vertex normal to this. This ensures that the resulting mesh will be correctly inflated by the subsequent smoothing iterations. As a result of this method, users are able to create initial meshes that lie in actual 3D space, as opposed to lying in a plane. Figure 3.1 gives an example of what the drawing process looks like.

Headset movement will result in a changing view matrix, as the position of the headset represents the position of the camera in the scene. When projecting the stroke's positions to 2D in real time, this could result in a jagged 2D representation. To prevent this from happening and to generate a smooth result we use the view matrix that was in use when the user drew the first stroke point for subsequent projection of all the stroke points. In order to prevent motion sickness and to provide the user with an interface that is easier to use, the tracking origin is reset every time the user starts a drawing action. The position of the Touch controllers is tracked relative to the position of the headset, so resetting the tracking origin to the current headset position makes the drawing appear where the user would expect to see it in real life.

3.1.2 Adding and removing control curves

In the non-VR case, adding control curves on the mesh and later removing them is done by unprojecting the screen coordinates onto the mesh in order to get 3D positions and decide where the user wanted to add/remove a curve. In VR, instead of unprojecting the screen coordinates of the hand or taking the direct hand coordinates, we cast a ray from the hand position in the direction that the controller is being held. This intersection between this ray and the mesh is then used as the final 3D position for adding/removal. Figures 3.2 and 3.3 show what the curve adding and removal actions look like respectively.



(a) (b)

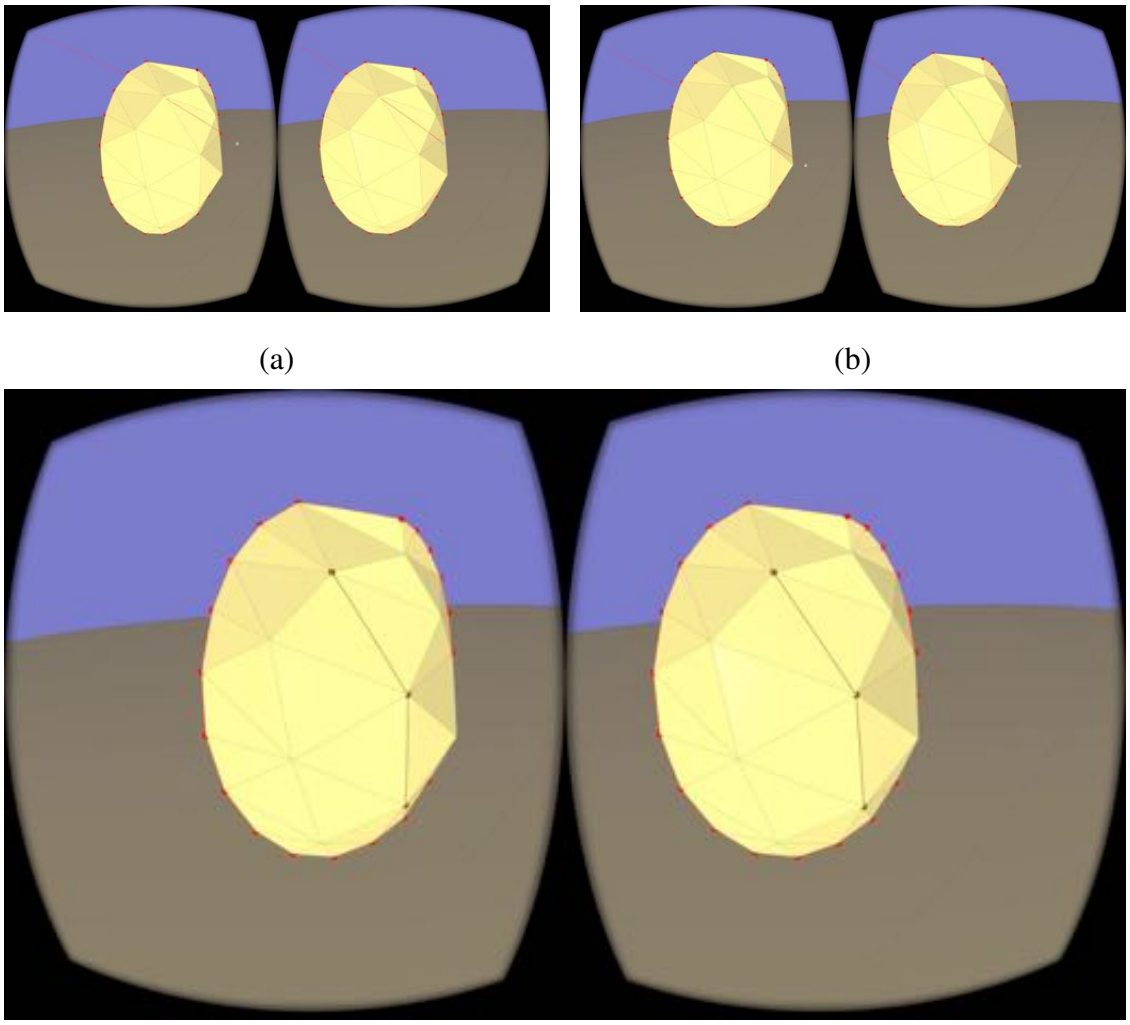
(c)

Figure 3.1: *Drawing action.* (a) *Before* (b) *During* (c) *After*

3.1.3 Cutting

When cutting in the non-VR program, the drawn stroke is interpreted purely in 2D screen coordinates. In order to create a loop on the front and back of the mesh surface, the algorithm checks which edges are crossed (in 2D) by a line segment between two consecutive stroke points. Since the same 2D points are used for creating a surface path on the backside of the mesh, this results in perpendicular cuts (meaning that the "cutting knife" always goes perpendicular through the screen, and never at an angle). On the other hand, when cutting in VR we take the coordinates of both the first (front side) and second (back side) intersection between the cutting ray and the mesh. By doing this, we enable the possibility for the user to cut the mesh diagonally, without first having to rotate the mesh.

In order to check which edges are crossed by the stroke segments, we initially check if the next stroke point lies either in the current polygon or in one of the adjacent ones. If this is not the case, we additionally check for intersections between the edges of the current point's polygon



(a) (b)

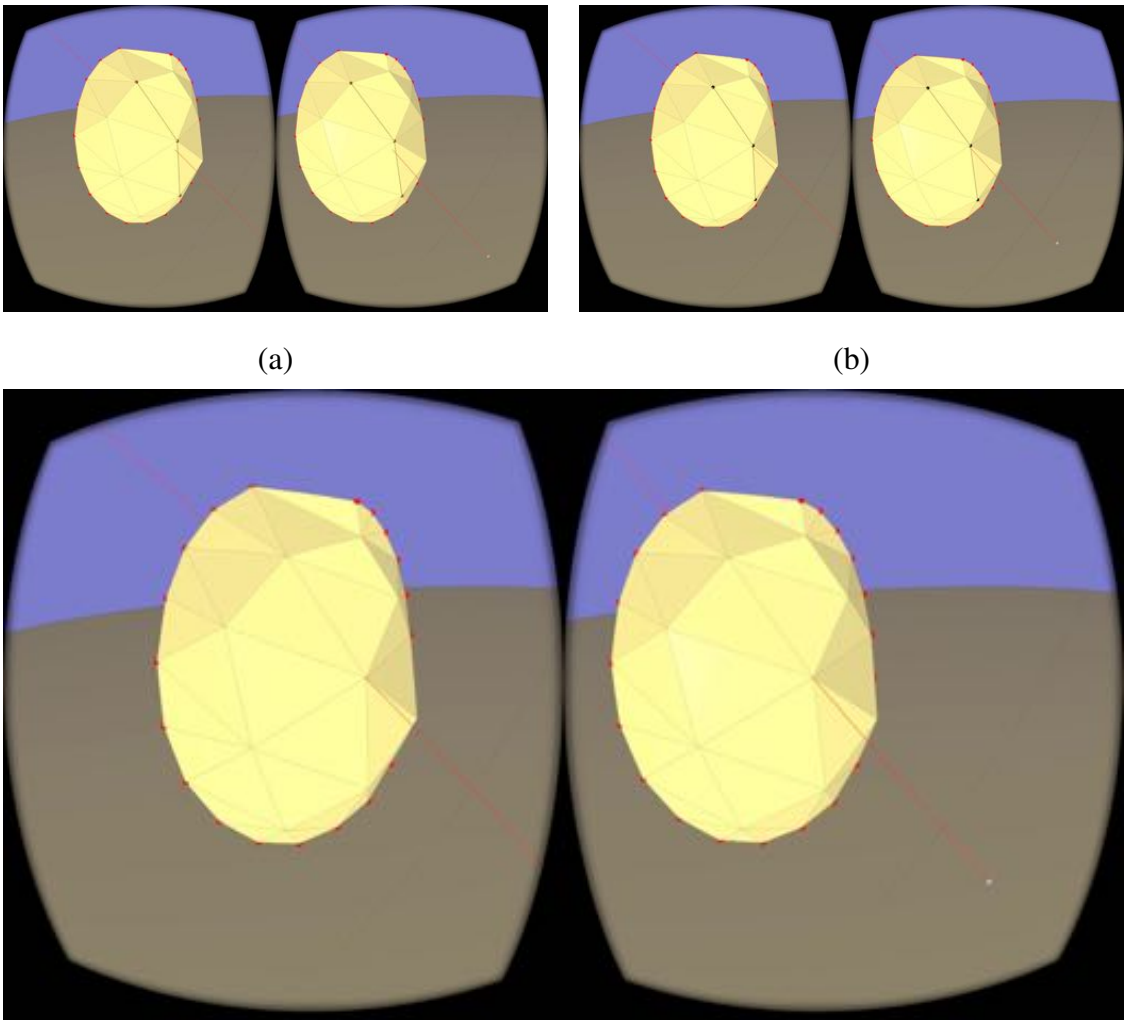
(c)

Figure 3.2: *Curve adding action.* (a) *Before* (b) *During* (c) *After*

and the plane that is formed by the two points that make up the stroke segment and the position of the hand at the time of drawing that stroke segment. We define the edges in the parametric form of a line and find the value t along the line where it is intersected by the plane. If an edge is intersected with $0.01 \leq t \leq 0.99$ (discarding 1% of the edge length on each side due to numerical imprecision), we know that the stroke segment crosses that edge.

One problem that arises due to the fact that we are using 3D intersection points between the cutting ray and the mesh, is that we cannot easily derive the start and end points of the cutting stroke. These points are the final point outside of the mesh before we start drawing on top of the mesh and the first point outside of the mesh after drawing on top of the mesh respectively. We need these points in order to be able to find the mesh boundary edges that we need to cross to wrap the surface path around to the backside of the mesh. As mentioned before, in the non-VR case we simply use 2D coordinates based off the screen coordinates of the mouse pointer. In VR we cannot take the position of the controller as the user is likely cutting from a distance.

What we have done in order to determine the start and end point is storing the controller position



(a) (b)

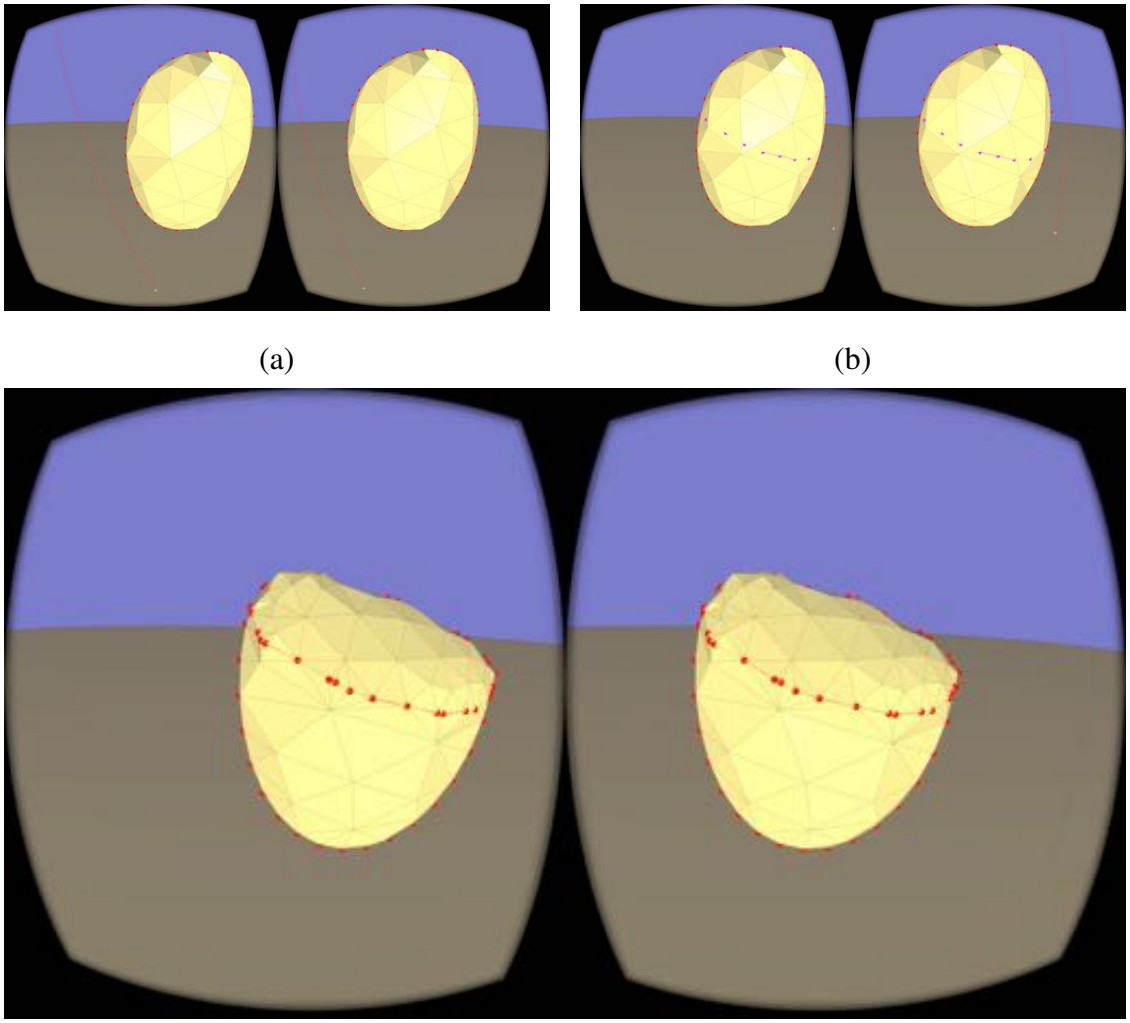
(c)

Figure 3.3: *Stroke removal action.* (a) *Before* (b) *During* (c) *After*

and direction for a start and end point. The start point is updated with every sampled point that does not intersect the mesh that is sampled before we have started going over the mesh. The end point is updated with every sampled point that does not intersect the mesh and is drawn after we have been on the mesh before. Then when the user releases the controller buttons after drawing the cut stroke, we take these positions and directions and along their resulting rays find the two points that are closest to the first and last point drawn onto the mesh. The two black points in Figure 3.5 represent the start and end point.

A problem that might arise when the user draws the cut stroke too fast (resulting in larger distances between the sampled stroke points) is that the algorithm fails to create a closed surface path over the mesh. This can happen when the resulting stroke points have at least one empty polygon between them, and the plane that is spanned by the two subsequent stroke points and the position of the controller at the time of drawing intersects the edge outside of the allowed range ($t \leq 0.01$ or $t \geq 0.99$). As a result of this, the algorithm is not able to find an edge to cross in order to move in the direction of the next stroke point and gets stuck. If this happens, a beep

will sound and the cut stroke will be drawn in black, similar to what is shown in Figure 3.7. Figure 3.5 gives a schematic overview of the problem that might arise with a cut stroke.



(a) (b)

(c)

Figure 3.4: *Cutting action.* (a) *Before* (b) *During* (c) *After*

3.1.4 Extrusion

The surface path for the extrusion base is created in a similar fashion to the cutting path, except that only the first hit between ray and mesh (front side intersection) is used. Drawing the extrusion silhouette in VR is significantly different from the way it is done in non-VR. When defining the silhouette stroke in non-VR, the user first has to rotate the mesh such that it is shown from a side view. Only then can the user sensefully specify the shape and depth of the extrusion silhouette, since we cannot specify any depth information from a frontal view of the extrusion base. The z-value of the silhouette stroke in this view is determined by projecting the silhouette stroke points onto one of the normal planes of the extrusion base. We select the plane that goes through the "left-most" (point with the smallest projected value onto a axis perpendicular to

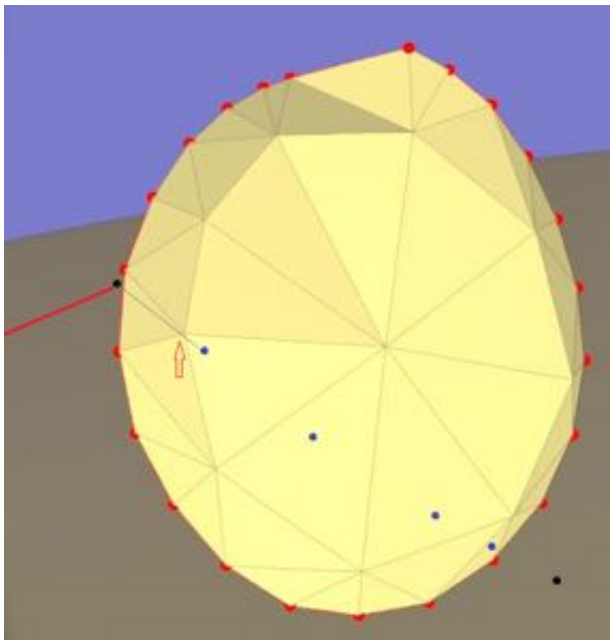
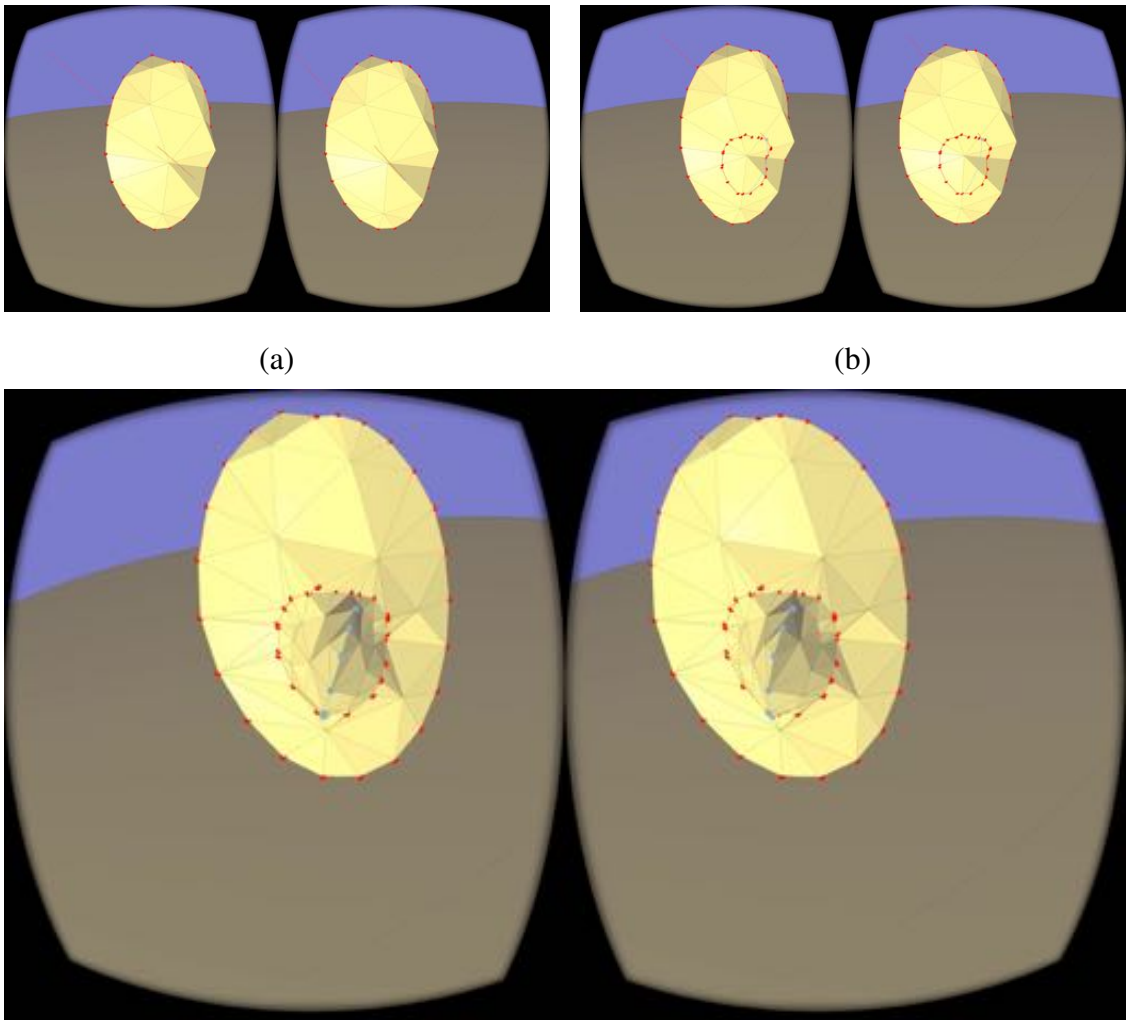


Figure 3.5: *Schematic drawing of a problematic cut stroke. The black off-mesh point and the left-most on-mesh point are separated by an empty polygon. The intersection with the edge that is indicated with the red arrow is discarded due to being out of range.*

the base normal) silhouette base vertex, which means that the silhouette points will lie in a line between the left-most and right-most base vertex. In VR on the other hand, we know the actual 3D positions of the controller and can therefore directly specify the extrusion silhouette in a frontal view of the extrusion base. If the user prefers to do this from a side view instead (to get additional visual feedback about the silhouette depth on top of the tactile feedback), this is also possible.

However, because the user also has control over the z-coordinate, the silhouette stroke is no longer guaranteed to lie on a line between the "left-most" and "right-most" base stroke points. This can give problems when projecting the silhouette points to 2D in preparation for triangulation. For the purpose of triangulation, the base stroke is divided in a left and right part, and together with the silhouette stroke each of these forms a half-dome. Both of these half domes are projected to 2D and then individually triangulated, unprojected back to 3D and finally stitched to each other. Because the silhouette is no longer guaranteed to lie on a straight line, it can happen that when we project the two half-domes onto the plane perpendicular to the base stroke normal, some of the silhouette points project to the wrong side of the base stroke points that complete the half-dome. Triangulating such a projection where the silhouette points flipped over to the other side will results in holes in the mesh. To overcome this problem, we project the points of each half-dome loop onto the plane that is perpendicular to the normals of each individual half-dome. This prevents the silhouette stroke points from projecting to the wrong side of the extrusion base stroke and thus will avoid meshes with holes. Figure 3.6 shows the mesh pre, during and post a successful extrusion operation.



(a) (b)

(c)

Figure 3.6: *Extrusion action.* (a) *Before* (b) *During* (c) *After*

3.1.5 Curve deformation

Finally there are some changes in the way that curve deformation is handled when moving from non-VR to a VR setting. Whereas in the non-VR version of SketchMeshVR we project all vertices on the constraint curves to 2D in order to find the closest vertex to the 2D screen position of our mouse pointer, in SketchMeshVR we simply compare the 3D position of the Touch controller with the 3D positions of all selectable vertices (vertices that are part of a constraint curve). If the closest vertex is within our threshold distance, we select it as the deformation dragging handle. In the case that the vertex is further away than the threshold, we instead go into navigation mode in the non-VR case and in VR we simply ignore the action. Directly using the vertex 3D coordinates instead of projecting vertex positions to 2D avoids ambiguity when multiple 3D coordinates with different depth values result in the same screen coordinates.

When determining where to move the selected handle vertex to, in non-VR we unproject the

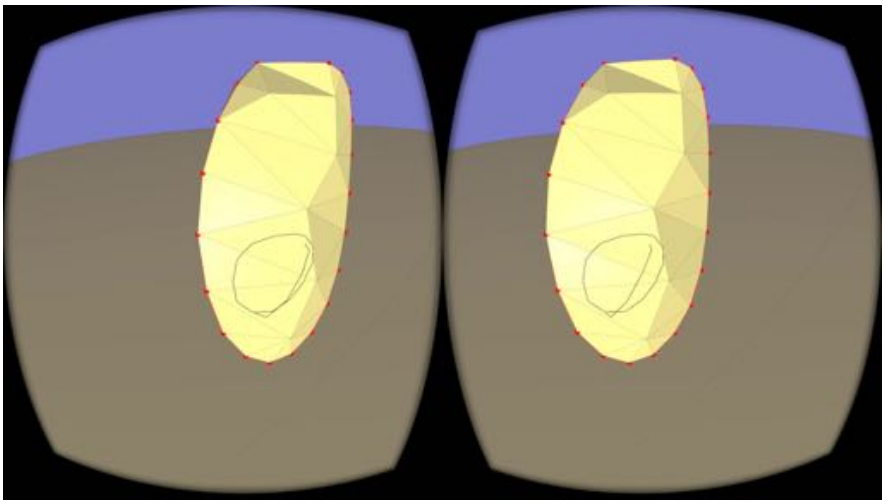


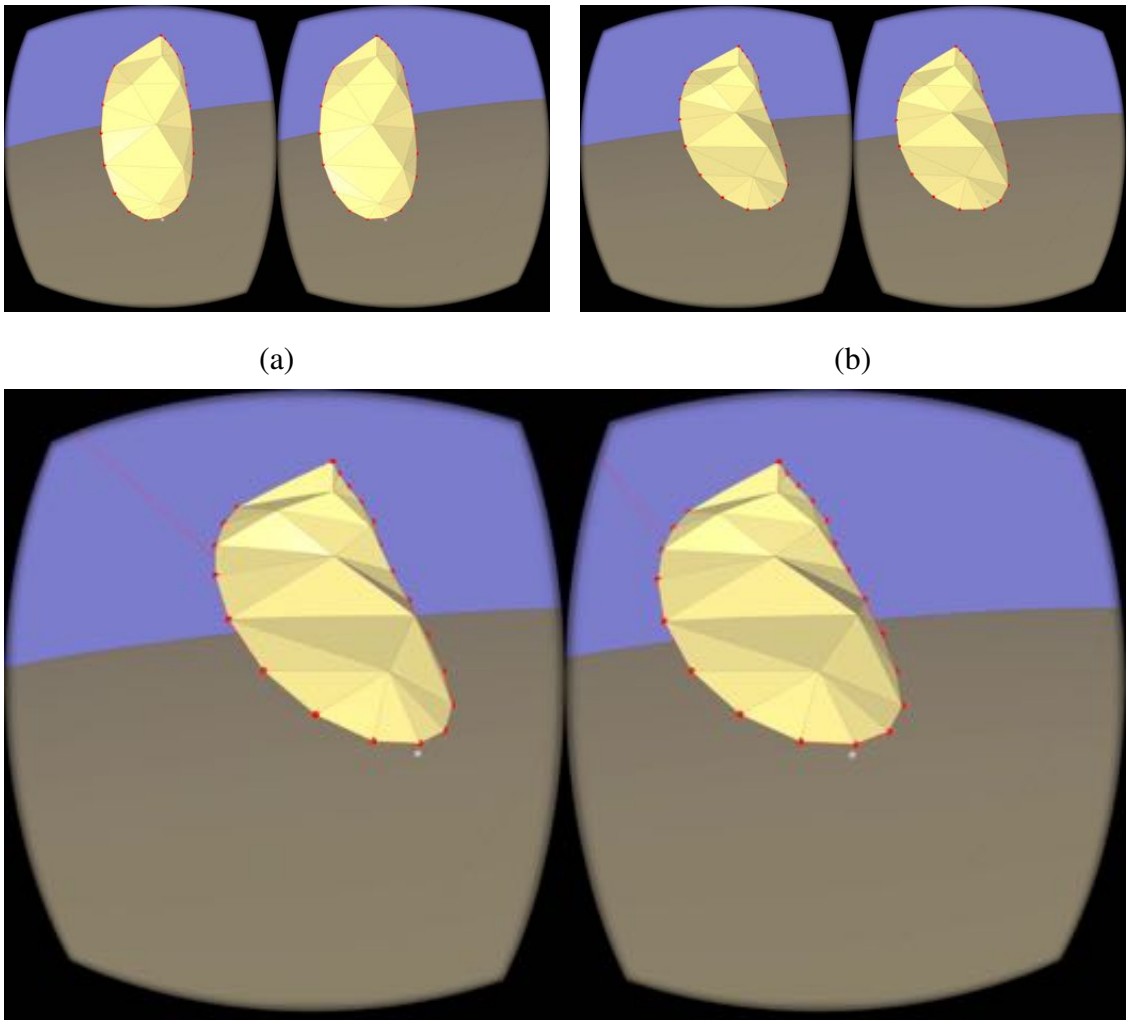
Figure 3.7: *Error screen as result of a problematic extrusion base stroke (it does not contain any vertex on its interior).*

screen coordinates to a 3D coordinate using the z-value belonging to the projection of the deformation handle vertex, resulting in a displacement in the XY-plane. In VR on the other hand, we simply take the actual 3D position of the Touch controller. Figure 3.8 shows an example of an out-of-plane curve deformation.

3.2 User Interface

For the user interface in virtual reality it was important to keep the controls for all actions very intuitive. Unlike the VR 3D modeling programs that were discussed in Section 2.2, SketchMeshVR does not have any menus in its interface. Instead all functionality in the program can be used with just the right Touch controller (in future versions the user should be able to choose whether to use the left or right controller as the primary one). To increase spatial awareness and simplify orientating the user's hand in the scene, a sphere is displayed at the location of the right Touch controller, together with a ray from the hand following the orientation of the controller. When the user is in draw or pull mode or is drawing the extrusion silhouette, the ray disappears and only the hand sphere shows. This is done in order to emphasize that the actual position of the hand is used to draw a stroke, instead of the intersection points of the ray and mesh. Next to that we added a base floor to the scene, giving the user a reference point for orientation and minimizing the chance of motion sickness.

Compared to drawing strokes with a mouse on a PC screen (which happens purely in a 2D plane), drawing strokes in virtual reality with the Oculus Touch controller allows the user to draw strokes directly in 3D. In SketchMeshVR we made use of this advantage whenever possible. However, when the user draws the stroke that is used to create the initial mesh, the software does work best when the stroke is drawn mostly on the plane that the user is looking at. This is due to the fact that the drawn points will be projected to this 2D plane before they are triangulated, and thus it will result in a more regular mesh geometry if the drawn stroke lies mostly on the plane. For subsequent cutting or extrusion operations there is no benefit to



(a) (b)

(c)

Figure 3.8: *Curve deformation action.* (a) *Before* (b) *During* (c) *After*

drawing strokes in the plane that is being viewed, as for these actions the actual 3D positions of the drawn points are used. The rest of this section describes how to employ each of the editing modes from within our application.

3.2.1 Drawing and mesh deformation

To create an initial mesh, the user has to simultaneously press and hold the grip and trigger buttons while drawing a stroke in the 3D space and subsequently release both buttons when the stroke is finished. If the drawn stroke results in a non-edge manifold mesh (for example if the stroke intersects itself), a beep will sound and the stroke is displayed in black. Otherwise the created mesh will be shown, with the originally drawn stroke overlayed.

In order to deform the mesh, a user can drag on all mesh constraint curves, which are indicated by a overlay that places coloured points on the vertices and coloured edges between them. The

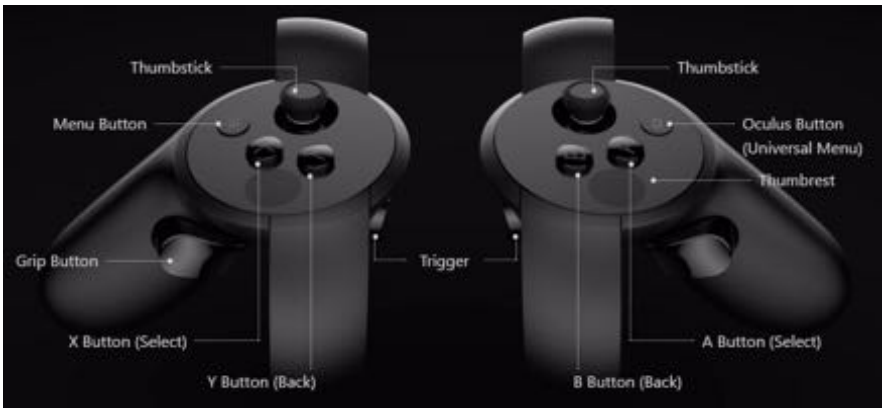


Figure 3.9: *Button layout of the Oculus Touch controllers.*

Action button	*Toggle button*	*Actions*
Grip	A button	Cut & extrude
Trigger	B button	Add & remove curves
Grip + Trigger	Thumbstick	Draw & pull

Table 3.1: *Button mappings for the different actions (the default action is mentioned first).*

user can toggle between drawing and deformation mode by pressing the B button, with draw mode being selected upon startup. To perform the dragging, the user has to simultaneously press and hold down the grip and trigger buttons while his hand is close to the constraint vertex that he wants to pull to a new position. Then, while still pressing the grip and trigger buttons, the user needs to move his hand to the desired new position, and release both buttons when the desired result is achieved. The new vertex position is interpreted directly as the 3D position of the Touch controller, therefore also allowing deformation outside of the viewing plane (as compared to the non-VR version, which only allows in-plane deformation).

The combination of grip and trigger buttons was chosen for these actions as they most resemble the feeling of holding a pencil and grabbing something in order to pull on it.

3.2.2 Adding and removing control curves

To add extra control curves to an existing mesh, the user has to press and hold the trigger button while drawing the stroke onto the mesh. All points that are outside of the mesh will be ignored and will not be added to the control curve.

In order to remove control curves (which is available on all constraint curves except for the curve that created the initial mesh), the user can toggle from stroke adding mode to stroke removal mode by pressing down the right thumbstick. Upon startup the stroke adding mode is selected. Whereas the drawing and deformation modes use the actual 3D position of the Touch controller, for adding and removing control curves the intersection between the hand ray and mesh are being used.

We decided to map the trigger button to the functions of adding and removing control curves since this feels like a natural way to select objects (removal) and to highlight areas (addition).

3.2.3 Cutting and extruding

Finally with the A button the user can switch between cut and extrusion modes. Upon startup the toggle is set to cutting mode, which allows the user to cut parts of the existing mesh away. To make a cut the user has to first point the laser ray to somewhere outside of the mesh, and then while holding down the grip button draw the cutting stroke over the mesh and finally releasing somewhere outside of the mesh again. In order to minimize unwanted behaviour, cut strokes should be drawn at a pace that is not too fast. If anything went wrong during the cutting procedure, the drawn stroke will show on the mesh in black and a beep will sound. When looking in the direction the stroke was drawn in, the part of the mesh that is on the left hand side of the stroke will be removed.

When in extrusion mode, the user has to start drawing the stroke on the mesh and then while pressing the grip button has to draw the entire extrusion base on the mesh and release the button when the stroke is complete. Just like in the control curve addition mode, any points that are drawn outside of the mesh will be ignored. To ensure a correct functioning of the algorithm, the drawn base stroke needs to have at least one vertex on its interior. Then to complete the extrusion, the user has to draw the extrusion silhouette stroke that will define the height of the extrusion. When drawing this stroke, the user has to press and hold the grip button again but this time the actual position of the Touch controller will be used instead of the ray-mesh intersection point. Releasing the grip button will start the extrusion computations.

When pressing only the grip button, the hand often naturally assumes a position that resembles the way we symbolize pistols with our hands. Intuitively this maps to the concept of shooting a laser ray from your hand, which would be a sensible tool to cut a mesh with. For the case of extrusion we can argue something similar, as part of the mesh topology is also cut away. For this reason the grip button was mapped to the cut and extrude modes.

4

Results

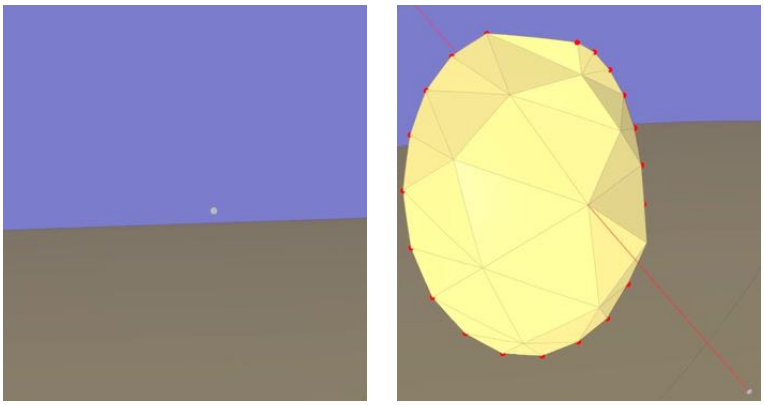
4.1 Interface

As mentioned in the previous chapter, SketchMeshVR does not make use of any menus and instead solely relies on different combinations of button presses to differentiate between the multiple available modeling actions. This does require more effort from the user in order to keep track of the selected editing modes, but at the same time keeps the interface cleaner. In order to guide the user when drawing strokes, SketchMeshVR provides visual feedback on the positions that will be used to create a stroke. In case of the drawing and curve deformation modes this means that the user sees the position of the controller and in case of all other modes the user will see the position of the controller plus a ray shooting from it (the ray ends at any intersections it has with the existing model). Figures 4.1(a) and (b) show what this looks like.

4.2 User review

Our program has been tested by users that had none to little prior experience with 3D modeling and also had none to little experience with VR. The test users were requested to first try and recreate one of the example models from Figures 4.2 or 4.3 using SketchMeshVR and then to recreate the same model in the non-VR version of the software. Users reported that although it took some initial effort to get acquainted with the controls, it was very easy to learn how to use SketchMeshVR. They also reported that the immersive nature of VR made the interaction with the created mesh much more impressive compared to the non-VR SketchMesh.

Although the users had experience with creating the given models when they modeled them in non-VR, it took them longer to recreate the model than when using SketchMeshVR. Especially



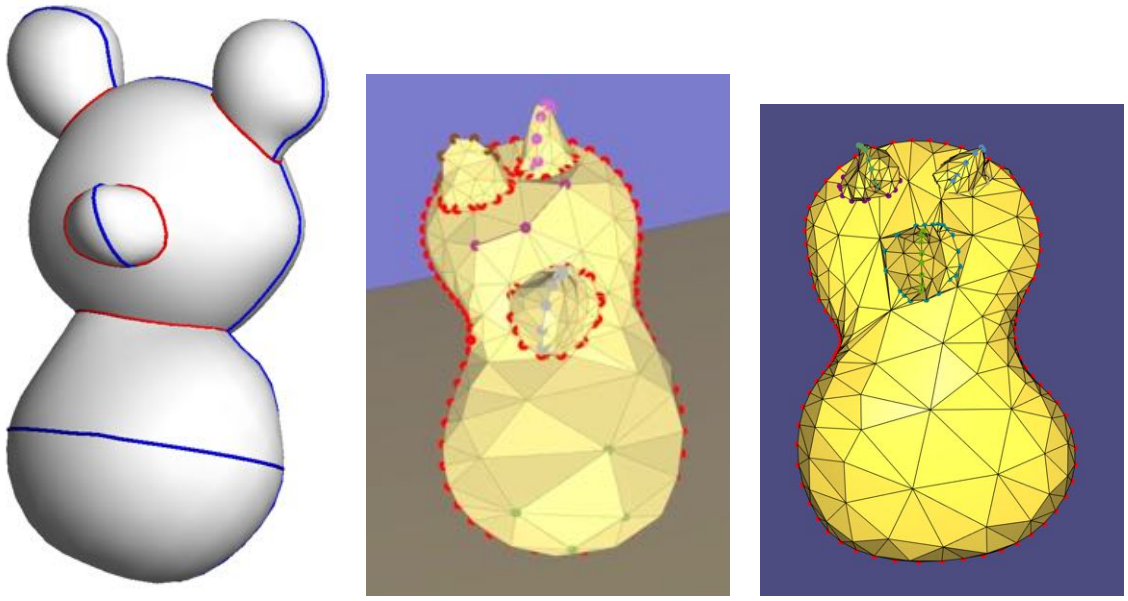
(a) (b)

Figure 4.1: *SketchMeshVR interface.* (a) *Controller reference point that is displayed in drawing and deformation mode.* (b) *Controller and ray reference that are displayed in all other editing modes.*

performing extrusions takes significantly less time in VR than in non-VR, due to the ability to specify silhouette strokes in 3D space. Curve deformation also proved easier and faster to perform in VR, especially when there are multiple control curves close to eachother (the projection method that is used in non-VR can lead to the selection of different curves from what the user expected). Figures 4.2 and 4.3 show side-by-side comparisons of the example models that were given to the test users, and the models that they created in both the non-VR and VR versions of our software. As you can see, the users managed to create models that were similar to the example models that were provided to them. The recreated teddy models do not show much difference between the VR and non-VR tool, apart from the time it took to create them. The recreated dolphin model does show a difference in detail between the version creaetd in VR and non-VR, with the non-VR version showing more resemblance with the example model. With SketchMeshVR users had difficulty creating the typical dolphin head shape and the sharp endings of the fins due to the lack of a sharp deformation tool. Users reported that the addition of this tool mode would greatly increase the artistic possibilities of SketchMeshVR.

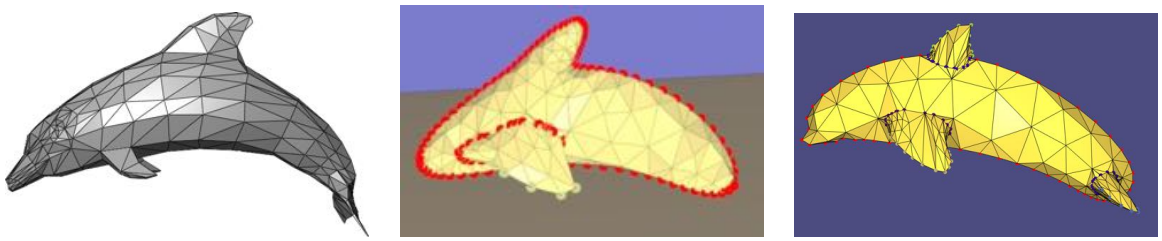
Figure 4.4 shows two further models that the test users created in SketchMeshVR, alongside the example models that were provided to them. While creating the turtle model, the user reported that the lack of a mesh navigation option was preventing them from adding the hind legs to the turtle since the user could not walk to the backside due to a desk being in the way. Future improvements of SketchMeshVR should definitely include this option in order to avoid this problem and also to simplify mesh editing when the mesh is either close to the physical floor or out of reach for the user.

While creating these models, users did not experience any fatigue in their hands. When using the software for a prolonged period though (15 minutes or more), users started to feel tiredness in their arms due to the relatively large body movement as compared to mouse movement. Users did not experience fatigue from the VR rendering while using the application and reported that they found the rendering to be very smooth. However, after prolonged wear of the headset some users reported an uncomfortable feeling due to the Oculus Rift headset pressing on the face.



(a) (b) (c)

Figure 4.2: *SketchMeshVR recreating models.* (a) *Example mesh of a simplified teddy bear.* (b) *Resulting recreated model made in VR (took 6 minutes to complete).* (c) *Resulting recreated model made in non-VR (took 8 minutes to complete).*



(a) (b) (c)

Figure 4.3: *SketchMeshVR recreating models.* (a) *Example simplified mesh of a dolphin.* (b) *Resulting recreated model made in VR (took 7 minutes to complete).* (c) *Resulting recreated model made in non-VR (took 12 minutes to complete).*

Also users reported that they found the modeling to be considerably more effortless in VR, as it requires less intermediate navigation of the mesh between modeling steps (for example before drawing the extrusion silhouette or making a diagonal cut there is no need to rotate the mesh in VR). However they did also remark that actions that required larger precision, such as defining additional control curves, were easier to perform with the mouse than in VR. Additionally users noted that the possibility to define strokes in an additional dimension provides added artistic freedom, for example when defining the extrusion silhouette the user can now draw a S-shaped curve and choose where it will be positioned inside the extrusion base, whereas the non-VR mode forces extrusion silhouettes to be straight lines.

Rendering in SketchMeshVR has been tested with meshes consisting of up to 5000 vertices, and no flickering, frame rate drops or other side effects have been observed during simple

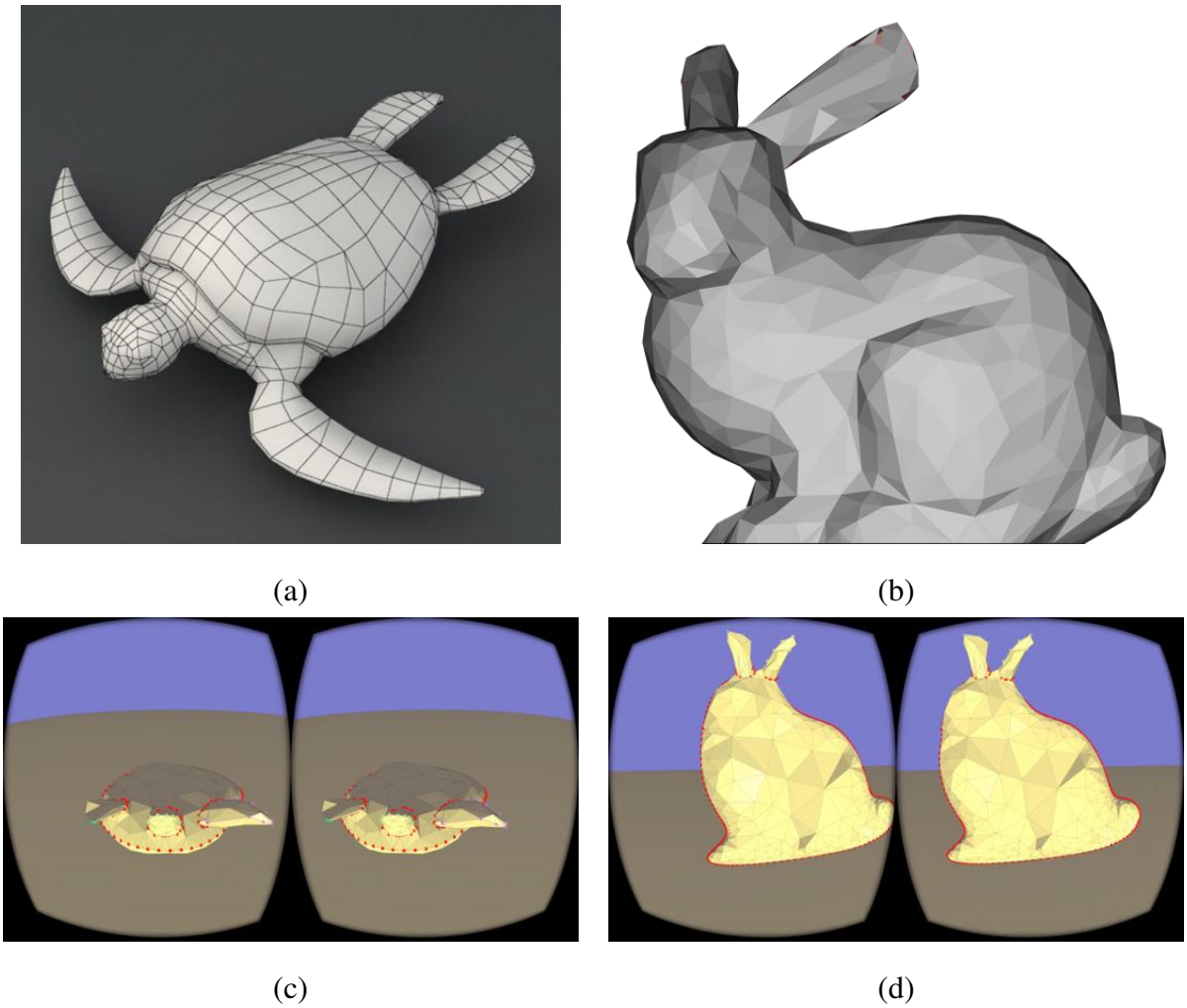


Figure 4.4: *SketchMeshVR recreating models.* (a) *Example mesh of a turtle.* (b) *Example mesh of the Stanford bunny.* (c) *Resulting recreated model made in VR (took 5 minutes to complete).* (d) *Resulting recreated model made in VR (took 9 minutes to complete).*

displaying of the mesh. Curve deformation poses the biggest performance bottleneck and can result in significant frame rate drops. Whereas during normal rendering the frame rate is at 90 Hz (without Oculus Asynchronous Spacewarp (ASW) enabled), it will show a significant drop during curve deformation. Our tests show that while deforming a mesh of 260 vertices the frame rate will drop to 70 Hz, with 640 vertices the frame rate will drop to 25 Hz and with 830 vertices it will drop to 20 Hz. All these frame rates are obtained with ASW disabled. When the frame rate drops below 90 Hz, ASW will bring the frame rate down to 45 Hz and interpolate between the frames to bring the rendering frequency back to 90 Hz again. Because of this a drop to 70 Hz is still recoverable, but frame rates below 45 Hz should be avoided. Possibly the frequency of intermediate mesh smoothing iterations while performing curve deformation could be lowered more in order to prevent frame rates from dropping below 45 Hz.

The precomputing of the linear system in draw mode poses another slow-down step. Precomputing allows us to solve the linear system much faster in subsequent smoothing iterations but does take significant time for the first iteration. The first smooth iteration takes 3 seconds for meshes of

1000 vertices, 11 seconds for 1500 vertices, 26 seconds for 2000 vertices and 69 seconds for 3000 vertices. Therefore, if the main goal is to provide an immersive and interactive application, it is advisable to generate relatively low-resolution meshes.

5

Conclusion and Outlook

5.1 Future work

The developed software still offers potential for future expansion and improvements. For instance, the software misses some of the useful tools that are present in other modeling software such as simple predefined shapes (like cubes or cylinders), mirroring and merging meshes. Using a second VR controller (e.g. the Oculus Touch controller) gives enough buttons to map these functions to, allowing us to implement them without the addition of a menu, thus staying with the principle of "intuitive" hand gestures. Adding an extra controller also allows for the user to switch between smooth and sharp curve deformation. The functionality for this is embedded in the software, but is not mapped to any button because all buttons of the first controller are already occupied by other functions.

Another functionality that would prove to be useful in many cases is the possibility to use blueprint images. When working with blueprint images, the user defines a series of different images of an object each taken from a different viewpoint. This allows them to trace these silhouettes from different angles and precisely recreate the object they want to model.

Additionally, the quality of the created meshes could greatly be improved by applying intermediate remeshing. As can be seen from the mesh examples, the triangle sizes differ greatly between triangles that are part of the initial created shape and triangles that are part of a subsequent cut surface or extruded part. This makes the appearance of the mesh rather chaotic and this could be avoided by remeshing every time a new surface is created. The purpose of the remeshing would be to make the edge lengths more uniform across the different parts of the mesh, resulting in a much more homogeneous mesh structure.

Final possibilities for improvements to the software lie in improving the interface. Implementing

hand avatars for hand orientation instead of a simple sphere could greatly increase the feeling of immersion. Furthermore, displaying the currently selected tool modes (for example cutting versus extrusion mode) is an addition that will greatly help the user in keeping track of the type of mode that is selected, especially when there is a lot of switching going on. This could be done either by including a HUD (head-up display), a distanced billboard or customized hand avatars for each tool as is done in Google Blocks (Figure 5.1 gives an example). Additionally enabling the user to navigate the mesh either with one of the thumbsticks or with hand gestures using the two controllers is a valuable extension that will improve the usability, especially for users who do not have a lot of space available to physically walk around the mesh.

Figure 5.1: *Google Blocks' customized hand avatar for the eraser mode.*

5.2 Conclusion

In this thesis we have presented a novel way of creating 3D models in virtual reality. Modeling 3D objects in VR in itself is not a novel concept, but to the best of our knowledge our software newly introduces the sketch-based 3D modeling paradigm to VR. Whereas other programs mostly work with volumetric brushes and tools to edit the created "clay-like" models, ours transfers the concept of sketch-based modeling to VR by letting the user define model silhouettes. Although volumetric brushes allow for more fine-grained control over the resulting model, we believe that our method provides a more accessible and effortless way of creating simple models. In our opinion, if the goal is to provide the user with an intuitive and uncomplicated way of creating 3D models in VR, SketchMeshVR is better suitable to do the task than other 3D modeling tools for VR like Google Blocks or Oculus Medium.

We can conclude that virtual reality provides a very useful and powerful extension to traditional 3D modeling and that bringing the sketch-based paradigm to a VR setup makes it even more versatile. Users reported that directly working in 3D especially made out-of-plane editing operations a lot more intuitive as compared to performing these operations in a traditional PC and mouse setting.

The source code for SketchMeshVR (and also its non-VR version), along with instructions for installation can be found online at `https://github.com/FloorVerhoeven/SketchMeshVR`.

Bibliography

3DS MAX, 2018. Autodesk Inc. https://www.autodesk.com/products/3ds-max/overview.

BLENDER, 2018. Blender Foundation. https://www.blender.org.

FELIPPA, C., 2007. Nonlinear finite element methods. www.colorado.edu/engineering/CAS/courses.d/NFEM.d/.

FU, H., AU, O. K. C., AND TAI, C. L. 2007. Effective derivation of similarity transformations for implicit laplacian mesh editing. *Computer Graphics Forum 26*, 1, 34–45.

GRAVITY SKETCH, 2018. Gravity Sketch. https://www.gravitysketch.com/app/.

JACOBSON, A., PANOZZO, D., AND OTHERS, 2017. libigl: A simple C++ geometry processing library. http://libigl.github.io/libigl/.

MAYA, 2018. Autodesk Inc. https://www.autodesk.com/products/maya/overview.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. FiberMesh. *ACM Transactions on Graphics 26*, 3.

SORKINE, O. 2006. Differential representations for mesh processing. *Computer Graphics Forum 25*, 4, 789–807.

ZBRUSH, 2018. Pixologic Inc. http://pixologic.com/zbrush/features/overview/.