

Realtime Editing in Virtual Reality for Room Scale Scans

2019

Charles Greenwood
University of Central Florida

Find similar works at: <https://stars.library.ucf.edu/etd>

University of Central Florida Libraries <http://library.ucf.edu>

 Part of the [Computer Sciences Commons](#)

STARS Citation

Greenwood, Charles, "Realtime Editing in Virtual Reality for Room Scale Scans" (2019). *Electronic Theses and Dissertations*. 6302.
<https://stars.library.ucf.edu/etd/6302>

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.

REALTIME EDITING IN VIRTUAL REALITY OF ROOM SCALE SCANS

by

CHARLES R. GREENWOOD
B.S. Illinois State University, 2000

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2019

Major Professor: J. LaViola

© Charles R. Greenwood, 2019

ABSTRACT

This work presents a system for the design and implementation of tools that support the editing of room-scale scans within a virtual reality environment, in real time. The moniker REVRSS (“reverse”) thus stands for Real-time Editing (in) Virtual Reality (of) Room Scale Scans. The tools were evaluated for usefulness based upon whether they meet the criterion of real time usability. Users evaluated the editing experience with traditional keyboard-video-mouse compared to a head mounted display and hand-held controllers for Virtual Reality. Results show that users prefer the VR approach. The quality of the finished product when using VR is comparable to that of traditional desktop controls. The architecture developed here can be adapted to innumerable future projects and tools.

This work is dedicated to my family for supporting my concentration on the work, and understanding my absence from family events, to my friends for moral support, and to everyone in the ISUE lab who helped out and made it fun.

ACKNOWLEDGMENTS

We would like to acknowledge here the Thesis Committee Dr. C. Hughes, Dr. M. Heinrich, and especially Dr. J. LaViola, committee chair. Acknowledgment also must go to the members of the ISUE lab, who helped show the way, provided support, and kept it real. We would also like to acknowledge our supervisors at Eastern Florida State College for allowing some flexibility in work schedule.

TABLE OF CONTENTS

LIST OF FIGURES	xi
LIST OF TABLES	xiii
LIST OF ACRONYMS AND ABBREVIATIONS	xiv
CHAPTER ONE: INTRODUCTION.....	1
Tools.....	2
Architecture.....	3
User Testing	4
Hypothesis.....	4
Document Guide	5
CHAPTER TWO: RELATED WORK.....	6
Industry Products for the Desktop.....	6
AutoCAD.....	7
Blender.....	8
SketchUp	9
Industry Products for VR systems.....	9
Kodon	10
TiltBrush.....	11
MeshLab – A Product of Academia.....	12

Scholarly Works	12
CHAPTER THREE: REVRSS ARCHITECTURE	15
Hardware & Software.....	15
Mesh Representation in Unity	16
Microsoft HoloLens.....	17
HTC Vive	18
The REVRSS HUB	19
Current Tool Interface	19
Mesh Management	20
Messages.....	20
Tool Control Panels.....	20
The REVRSS ROOM.....	21
ITEM	21
SCAN.....	21
QUAD.....	22
The REVRSS TOOL	23
The REVRSS USER.....	24
Control Activation	25
Control Indicators	25

Maneuvering	26
User Scale	26
CHAPTER FOUR: REVRSS TOOL SET	27
Mesh Appearance	27
Paint and Texture	28
Warehouse	28
Transforms	29
Vertex Selectors	30
Single Select	31
Spider Select	31
Sub-mesh Select	32
Whole Mesh Select	32
Enclosure Selectors	33
Surface Analyzer	34
Surface Deformation Tools	34
Vertex Moves	35
Hammers	35
Straightest Line	37
Mesh Topology Alteration	37

Cut, Copy, and Paste.....	37
Triangle Splitting.....	38
Sub-mesh Join.....	39
Other Triangle Modifications	40
Quad Creation.....	40
Walls, Floor, Ceiling.....	40
First: Find the Floor.....	41
Second: Mark the Corners	42
Third: Find the Ceiling	42
Fourth: Refine the Walls.....	43
Fifth: Refine the Selection.....	43
Sixth: Divide and Delete Quads	44
Seven: Finish	45
CHAPTER FIVE: USER STUDY	45
Experiment Design.....	45
User Experience Surveys	46
Performance Evaluation: Placement	49
Performance Evaluation: Time to completion	51
Comments and Observations.....	53

CHAPTER SIX: DISCUSSION AND FUTURE WORK.....	57
Expert Results	57
Measuring Successfulness.....	60
REVRSS in the Future.....	61
Decorating	61
Selection and Manipulation.....	62
Deformation.....	63
Topology Changes	64
Utilities	65
User Interaction	66
Notes For Developers.....	67
CHAPTER SEVEN: CONCLUSION	68
APPENDIX: IRB APPROVAL LETTER.....	70
LIST OF REFERENCES	73

LIST OF FIGURES

Figure 1	AutoCAD, from https://www.autodesk.com/products/autocad/overview	7
Figure 2	Blender, complex and customizable. From LO4D.COM.....	8
Figure 3	A screenshot from SketchUp including a car from their warehouse.....	9
Figure 4	Kodon, from the Steam Store. A clay modelling technique.....	10
Figure 5	Tilt Brush in use, from indianexpress.com.....	11
Figure 6	a screenshot from MeshLab. From “Basic MeshLab” at debian.net.....	12
Figure 7	Left: a wireframe mesh outline and rendered model car, a Unity standard asset. Right, texture UV mapped to a 3D object, from https://en.wikipedia.org/wiki/UV_mapping	16
Figure 8	A comparison of a HoloLens scan rendered in white, and with wireframe. The gold and green colors indicate that there are two separate mesh objects. Clearly noticeable are the overlaps between non-connected regions in the scan.	17
Figure 9	REVRSS Architecture: command and communication flow	19
Figure 10	Tool control panels, with the VR main control panel nearest the sun icon.	23
Figure 11	Scan fully rendered, wireframe, and color for number of sub-mesh triangles	27
Figure 12	a truncated cone has been added, and then rotated using the transforms	29
Figure 13	The original spider selector finding connected vertices out to three hops. Smaller spheres indicate greater number of hops from the graph center.....	32
Figure 14	Early highlighting, and simple vertex moves	35
Figure 15	a hammer in KVM, showing scale widgets and purple highlight	36
Figure 16	removing/copying a chair: selected, cut, copied.	38
Figure 17	Triangle splitting techniques	38

Figure 18	WFC steps 1 and 2. Left: Find the floor. Right, mark the corners	42
Figure 19	WFC steps 3 and 4. Left, Find the ceiling. Right, Refine the walls.....	43
Figure 20	WFC steps 5 and 6. Left, refine the selection. Right, new wall guides.....	44
Figure 21	WFC steps 6 and 7. Left, wall quads divided. Right, finished product.....	45
Figure 22	Survey responses by Task	47
Figure 23	All users preference for KVM or VR	48
Figure 24	RMS Placement Error, by Group and Task.....	49
Figure 25	rms placement error for each surface, by task.....	50
Figure 26	Mean time to completion.....	51
Figure 27	Feature completion time, per Task	52
Figure 28	expert results, niche and closet.....	57
Figure 29	expert results, closet, TV, door, window.....	58
Figure 30	expert results, blank wall and furniture.	58
Figure 31	typical user has gaps near details, and removed the TV and door	59
Figure 32	user image from near front door showing big gaps, loss of bed detail.....	60

LIST OF TABLES

Table 1	Time to identify co-located vertices by divisions per axis, in seconds.....	39
Table 2	Significance of responses, KVM vs VR.....	47
Table 3	Significance in placement error	50
Table 4	Significance in time to complete tasks	52

LIST OF ACRONYMS AND ABBREVIATIONS

3D	Three Dimensional
AR	Augmented Reality
CAD	Computer Aided Drawing/drafting
CAVE	Computer Assisted Virtual Environment
KVM	Keyboard, Video, Mouse. The standard desktop setup.
LIDAR	Light Detection And Ranging, a 3D scanning tool
REVRSS	Real-time Editing in Virtual Reality of Room Scale Scans
RTH	Runtime Transform Handles, a Unity asset
UV	a texture map, the coordinates of which are called u and v for clarity
VR	Virtual Reality
WFC	Walls, Floor, Ceiling. A tool in REVRSS

CHAPTER ONE: INTRODUCTION

The project described in this Master's thesis, Real-Time Editing in Virtual Reality of Room Scale Scans (REVRSS), is an effort to produce a VR system for making changes to room-sized scans so that they look and behave more like a real room. While there is no shortage of desktop-based editors that can achieve this goal, we feel that a more natural editing environment using virtual reality can be realized. We would like to produce tools and interfaces that are modular, so that any input/output devices could be accommodated, and any tools added or modified without rewriting the entire system.

If the end goal of users is a fully featured room, it would make sense to edit the scan from inside, as if it were a real room. Doing so will give the user a sense of scale and proportion unmatched by desktop monitors. Architects already display their work to prospective clients in 3D using head mounted displays in the form of goggles and a smart phone. The client may wish to make changes to the plan, and send it back to the architect for making changes to the technical drawings. Simple tools in the 3D environment will make this more natural for those who are not comfortable with 2D plans. Game developers may find a room which they can scan and modify, saving time in development. If the game is intended for VR, doing the editing work inside the virtual room saves the effort of designing on a desktop, checking it in VR, then going back to the desktop for changes.

A VR editor would consist of user output in the form of stereoscopic all-around viewing capability, some natural user input in the form of open air gestures, data gloves, or control devices held in the hands, and the software necessary to affect changes to the scanned room.

The possibilities for editing three-dimensional (3D) mesh-based content are nearly infinite, as evidenced by the profusion of desktop 3D graphical editors in industry. The desire for more natural control interfaces is evident in the array of 6-degree-of-freedom controllers used by drafting professionals. Expanding what was already a plentiful set of choices for designers, drafters, and hobbyists, the advent of economical and powerful Virtual Reality (VR) systems give the user a new way of operating the software tools for 3D modelling and editing.

This work has three main goals. First, the creation of tools for editing mesh models in a 3D environment. Second, the development of a system architecture that can easily accommodate new tools, to be operated by any user input device. Finally, user testing compares a set of tools used on a desktop system to the same set of tools used in a VR environment for time required, accuracy of work, and user preference.

Tools

The current set of tools in REVRSS includes methods for selecting portions of the mesh that makes up a 3D model. Selectors include single vertex, spider (connected vertices) and enclosures (shaped like a cube, cylinder, and sphere) selectors for sub-mesh, and whole mesh. Individual vertices or triangles can be selected for alteration with single select. An enclosure selects a group of vertices or triangles within some defined region. Vertices connected by triangle edges can be selected with a “spider” that traverses those edge.

Any editor must be capable of deforming the mesh surface through a number of methods driven by user actions, and REVRSS gives us tools for doing so. The surface can be altered to conform to a selected shape using hammers, or selected regions moved at will. A set of tools

may remove portions of the mesh, or duplicate selected regions of mesh. Most tools can be scaled and rotated by simple user actions.

Items can be added to the model, or removed from the model. An item might be a copied part of the original mesh, a new mesh from some other source, or a shape built on demand. Replacing items is useful when the scan is just too poor to correct any other way. Removing or adding items may be done if game development requires a room which is not quite like the scanned room, or by interior designers to show how different furniture can fit an existing room. Users can scale, rotate, and position items within the model. The beginnings of methods to color the surfaces in the model, and to apply textures, are under development. Adding color and texture to a blank white room will make it much more realistic.

Architecture

A major motivation was to build a system that was modular, allowing any user input device to be used with any tools, on any model. The driving notion was anonymity – no portion of the system needs an understanding or knowledge of other parts of the system. To that end, REVRSS is divided into main sections for the USER, the TOOL and the ROOM. The user section takes button presses, gestures, or spoken commands from the user input device, and converts them to a uniform set of action commands. These actions are routed to the currently active tool by a central HUB. Tools act through the hub on the mesh and items in the scanned room. Information about the room, and from the tools, travels through the hub to the user. A large amount of information is maintained in a mesh manager, to be accessed by the tools.

User Testing

We tested individual tools for applicability. A tool was chosen for inclusion in the tool set based on the function it provides, and whether it meets responsiveness conditions. Testing with live participants was done to compare the experience of using the tools on a desktop system (KVM) to the experience of using the tools with a head-mounted display and hand-held controllers (VR). A set of survey questions provides user feedback, and comments by the users and observations by the experimenters provide some context and subjective evaluation.

It is our contention that the users will strongly prefer working in VR. Further, we expect to show that the time required to complete tasks, and the quality of the final product, will be no worse in VR than with KVM. To show this, the tools in REVRRSS were instrumented to provide a time stamped file with locations of objects placed in the room. The time required and the placement of the items compared to a theoretical “perfect location” provides an objective measure of the suitability of the tools.

Hypothesis

Our hypothesis, then, is threefold. We contend that it is possible to build simple tools for editing mesh models in 3D; that it is possible to create a flexible architecture that accommodates any tool and any user input device; that users will show a preference for VR while producing work comparable in time and accuracy to that done with a traditional desktop system.

Document Guide

The remainder of this thesis will provide the reader with an understanding of the REVRSS system. Related Work in chapter 2 shows how it benefits from existing work and where it goes beyond them. In chapter 3 we describe the REVRSS architecture, and show how the parts work separately yet fit together. A thorough explanation of the tools currently available makes up chapter 4. The design of and results from the user study are in chapter 5. Chapter 6 will give some thoughts about a possible future for REVRSS. Finally, in chapter 7 we bring the thesis to a conclusion.

CHAPTER TWO: RELATED WORK

There are thousands of scholarly articles written on every aspect of 3D graphical representations, on modifying a mesh, and on user interfaces in general. There are also many commercially available products that employ these techniques. Products that alter meshes are available now for VR systems as well as for desktops. Countless how-to web sites have provided information, code samples, and inspiration for the tools implemented in REVRSS.

Industry Products for the Desktop

We feel it important to acknowledge here that there are many products available for three-dimensional design work. Most of these are expensive and complex, with rich features that could not be duplicated by a small team with limited time. A few notable desktop editors are freely available, and yet have an enormous and complicated set of features and capabilities. Some are beginning to see extensions that allow editing using a VR headset, while others remain thoroughly on the desktop. These products have influenced the creation of the tools for this thesis by providing examples of tools that are effective, work quickly, and are easy to use. In some cases, poor tools or confusing tools have spurred us to create more natural tools. The look and feel of indicators is, in some cases, emulated in REVRSS.

AutoCAD

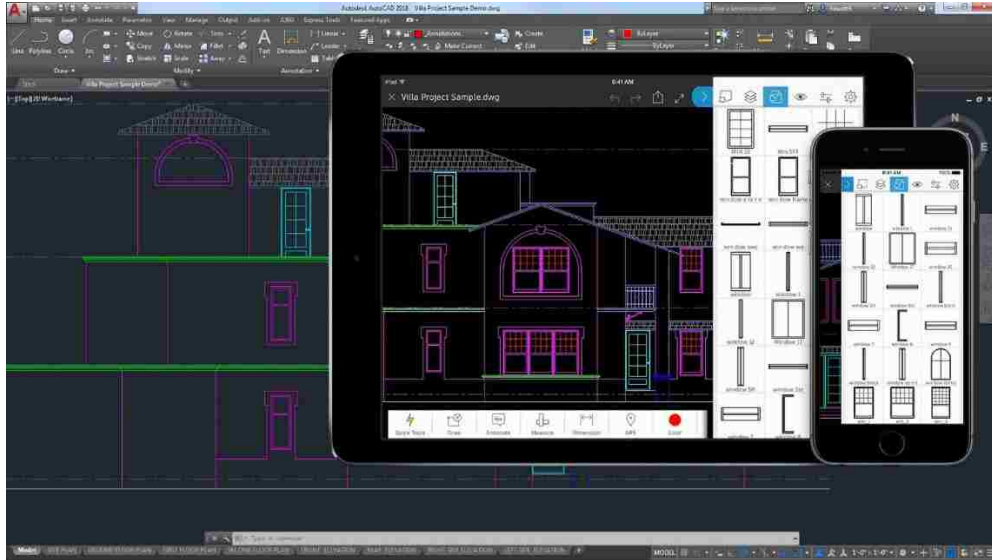


Figure 1 AutoCAD, from <https://www.autodesk.com/products/autocad/overview>

Perhaps the first product that comes to mind for drawing and design, AutoCAD is a large and complex set of tools used by industry professionals all over the world. AutoCAD's maker, Autodesk, has been in business for decades and has grown and changed over the years. This software can be used for 2D and 3D architectural and mechanical drawing, project cost estimation, and more. Other examples of large, complex commercial products widely in use are SolidWorks, Turbo CAD, Vectorworks, and many, many more.

Blender

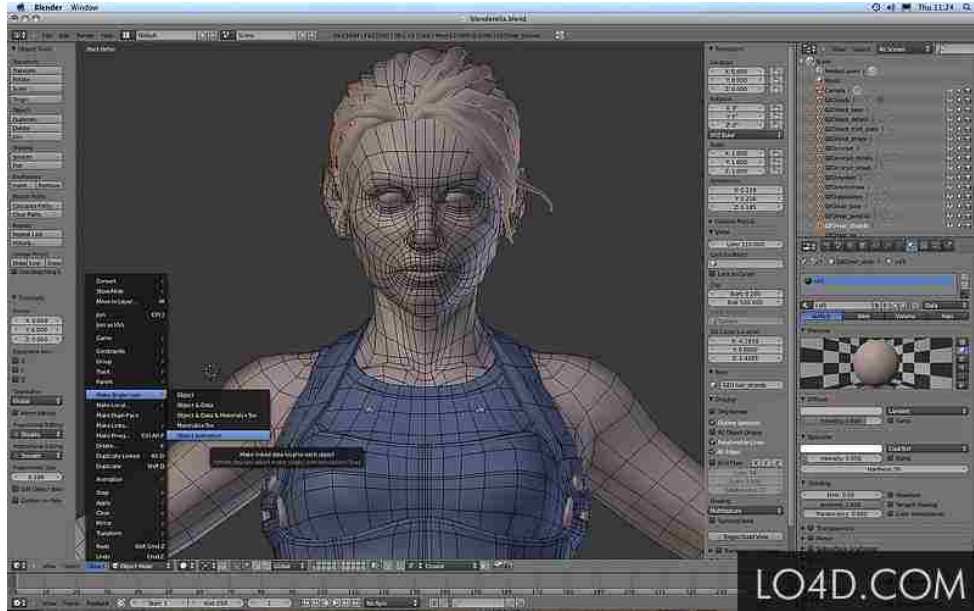


Figure 2 Blender, complex and customizable. From LO4D.COM

Blender is an open-source drawing and design tool that many use for creating and animating mesh-based 3D objects. Like most design software, the learning curve is steep and the possibilities are endless. One can make a quick 3D object, animate it, and make a movie or a game all in one package. As with the paid commercial applications, there are many tools available. With Blender, all are free and open source.

SketchUp

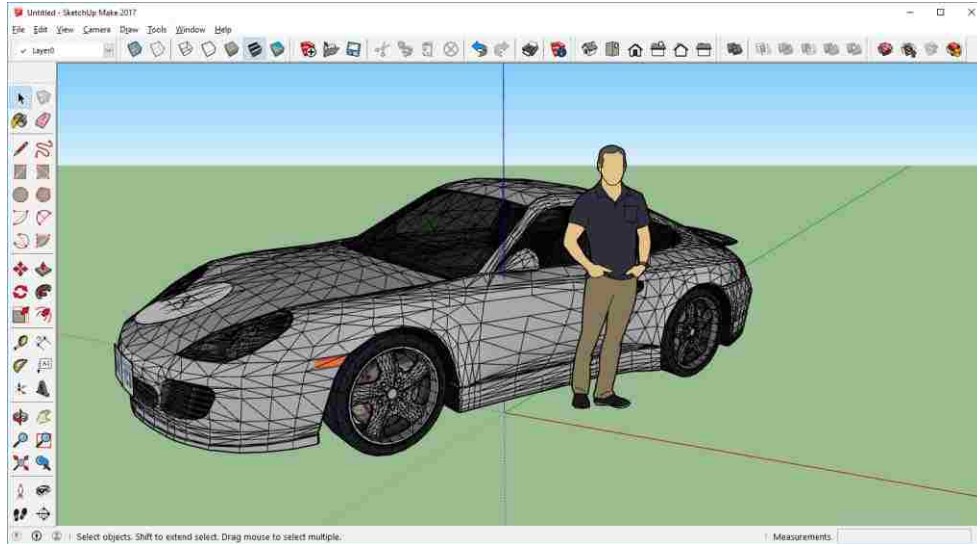


Figure 3 A screenshot from SketchUp including a car from their warehouse

SketchUp is a simple but capable tool that can generate 3D objects for use in architectural and mechanical drawings. Once a product of Google, it has been spun off into a separate product owned by Trimble, Inc. Despite having a small set of tools, impressive objects have been constructed by users, many of which are available in an online warehouse. These objects make good candidates for inclusion in the model during REVRRSS edits.

Industry Products for VR systems

When this project had its quiet beginning, there were only a couple of virtual reality and head mounted display productivity tools available. Now there are several products for VR available, with a wide variation in operating methods and purposes. Some of these were useful guides in designing and implementing the tools and user interface in REVRRSS.

Kodon



Figure 4 Kodon, from the Steam Store. A clay modelling technique

This is an interesting take on modeling tools. Kodon behaves like a clay modeling studio, with tools that mimic squashing, pinching, smoothing, and extruding a big blob of clay. Still in development, it makes for a standout by being different. Someday REVRSS may deform a surface as rapidly as Kodon. Despite some interface issues, it is a fun and useful tool.

TiltBrush



Figure 5 Tilt Brush in use, from indianexpress.com

Google's Tilt Brush is another outstanding art productivity tool. The user interface is clean, responsive, and straightforward. Using it is like having an unlimited supply of goeey paint, ribbons, and fire that stay put as the artist squirts them into the air.

MeshLab – A Product of Academia

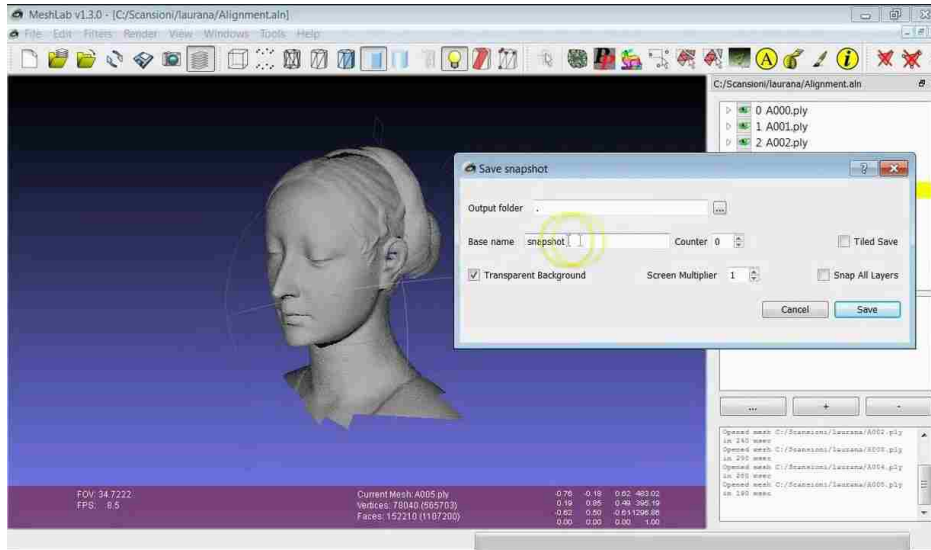


Figure 6 a screenshot from MeshLab. From “Basic MeshLab” at debian.net

Originally a course assignment at the University of Pisa in 2005, MeshLab [1] is an open source and free collection of tools for mesh creation, distortion, and modification. Written in C, with source code available, MeshLab is a gold mine of tools, good candidates for the future.

Scholarly Works

A number of journal articles had an influence on the creation of REVRSS. Some provided a basis for the tools and methods, and others point the way to future tools. Chief among the guides is the book 3D User Interfaces Theory and Practice. [2] Chapters 5 Selection and Manipulation, 6 Travel, 8 System Control provide a very good overview and introduction to the techniques used and troubles overcome in building REVRSS. In trying to come to grips with how mesh structures are represented, and how they can be manipulated, we found “What’s in a Mesh?” [3] to be very instructive.

During editing, the environment may become filled with objects such as furniture, and we'll need more capable selection techniques. There are numerous methods available for making selections, but none work well in all scenarios. Previous work at UCF [4] shows how the best among these for a given set of conditions can be applied automatically. Some of the selector methods [5] appear more suited to a desktop system than VR. Selecting and manipulating more than one object at a time is possible in REVRSS by making multiple single selections. Selecting objects which are out of reach of the user [6] is handled in our system by the pointing ray extended from the main hand,

Many of the more interesting mesh deformation techniques [7] operate too slowly for our VR system's high refresh rate requirement. Incorporating mesh simplification, deformation, and then reconstituting the original works well for large meshes where high frame rates are not as important, such as a graphics workstation. One useful approach [8] to speeding up this process is to limit the actions to a portion of the mesh. Their use of the term "submesh" to mean a chosen portion of a larger mesh should be distinguished from our use of "sub-mesh" here, in which we mean a section of mesh that is not connected to other parts of the same mesh data structure.

Correcting problems with the mesh from Hololens is a main goal. There are numerous holes, many disconnected single triangles, and the onerous difficulty of having the mesh divided into cubical regions that do not share vertices at the overlaps. Many of these problems can be corrected in MeshLab, or by an application of ReMesh [9] tools. The sheer number of polygons in even a small room had us looking for simplification algorithms [10] [11] but it was determined that we wished to keep detail, and instead replace areas with too high a vertex density with simple shapes, such as with our flat surface analyzer tool.

While considering how best to find connected vertices, we came across a recursive divide-and-conquer approach [12] which produces a binary tree. This is quite different from our method, which produces a graph representation by inspecting the mesh's triangle list.

To simplify the coding, often an object to be manipulated is made a child of a transform box, and user input is applied to the box instead of the object or objects. This is a somewhat similar notion to a real box [13] used in CAVE systems.

There seemed to be few published accounts of mesh editing systems for VR in existence, but we did find some interesting systems that incorporate force feedback and haptic interfaces [14] [15] to deform and paint a mesh. One system [16] operates on captured data from color and depth cameras, which would need to be converted to a mesh representation before editing in REVRSS. We feel that our system goes beyond what we have found in its plug-in component design, which makes changing user interfaces and adding tools relatively simple.

CHAPTER THREE: REVRSS ARCHITECTURE

The development of REVRSS followed an incremental design approach. Small, single-purpose tools were developed individually with the main goal of understanding how the 3D mesh works, and how it can be modified. Simple tools sparked ideas for more complex ones, and informed of the need to develop data about the mesh which could be used for even more powerful tools. In order to make changes to the scanned mesh, we needed to understand how meshes represent the model, how the HoloLens forms the scan of the room, and how the Unity game engine handles the data.

Soon it became apparent that we would need to develop an architecture that could manage the tools, the various forms of user input, and to keep data about the mesh ready for use by other parts of the system. Three main sections of the system architecture divide the functionality into sections related to the USER, the ROOM, and the TOOL. Central to all of these is the HUB which acts to route information and commands.

Hardware & Software

Due to the popularity and familiarity it enjoys in our lab, the Unity game engine was chosen for development, with code written in C-sharp. The Microsoft HoloLens was available in our lab, and therefore served as the source device for making scans. We chose the HTC Vive head worn display and hand controllers for the hardware portion of our user interface.

Mesh Representation in Unity

A mesh is a set of points in 3D space (vertices) connected by lines (edges) to form triangles (polygons) that lie on or near the surface of some object. Each mesh may be composed of sub-meshes that share a data structure, but have no edges connecting vertices between them. Each mesh data structure also stores information about the direction of a surface normal at the vertices to aid in the smooth lighting on the final rendered image. Each also has information that serves to identify a location in 2D images for color and texture information to be applied to the area surrounding each vertex. The Unity game engine uses these mesh data structures to provide information to the graphics card which renders the objects as graphical images on some viewing device. There is an unfortunate limitation in older graphics cards, and in earlier versions of Unity, which sets a maximum size of about 65,000 triangles per mesh. It was necessary to develop code that circumvented this limitation. Newer graphics cards and later versions of Unity may overcome these limitations, but the technique is still useful because it can handle cases where the environment is composed of many different parts from more than one source.

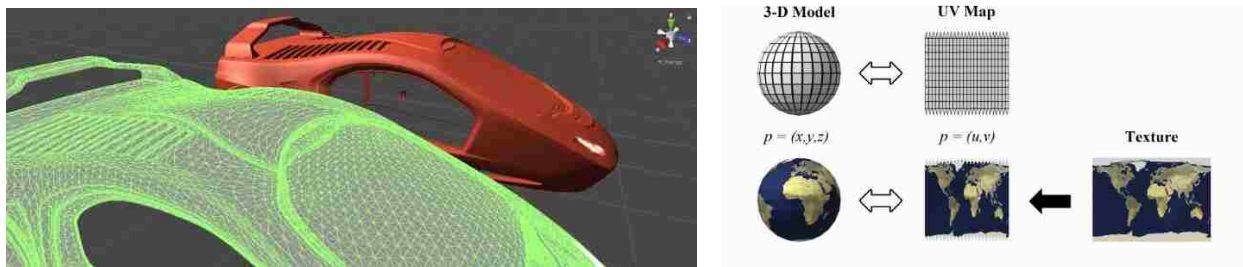


Figure 7 Left: a wireframe mesh outline and rendered model car, a Unity standard asset. Right, texture UV mapped to a 3D object, from https://en.wikipedia.org/wiki/UV_mapping

Microsoft HoloLens

The room scan used in this project originates from Microsoft HoloLens, an Augmented Reality (AR) headset device. The HoloLens overlays virtual objects on a real world, rather than building an entire virtual world. While in use, the HoloLens continuously scans the surrounding environment and compares the current scan to stored scans to determine the user's position and orientation within a space. To do this work quickly, the scans are purposely kept small and not terribly detailed. A scan produced by a LIDAR system stores millions of colored dots in a point cloud that when converted to a mesh may have millions of triangles. By contrast, the HoloLens scan of a living room size area may have around 100,000 points and triangles.

To speed the recognition of room features, the HoloLens scan is broken into cubes about two meters in size. The meshes at the adjoining edges of these cubical regions do not meet perfectly, they overlap slightly. There are also numerous smaller sub-meshes which may be updates to the scan. These features introduced a few problems to overcome.

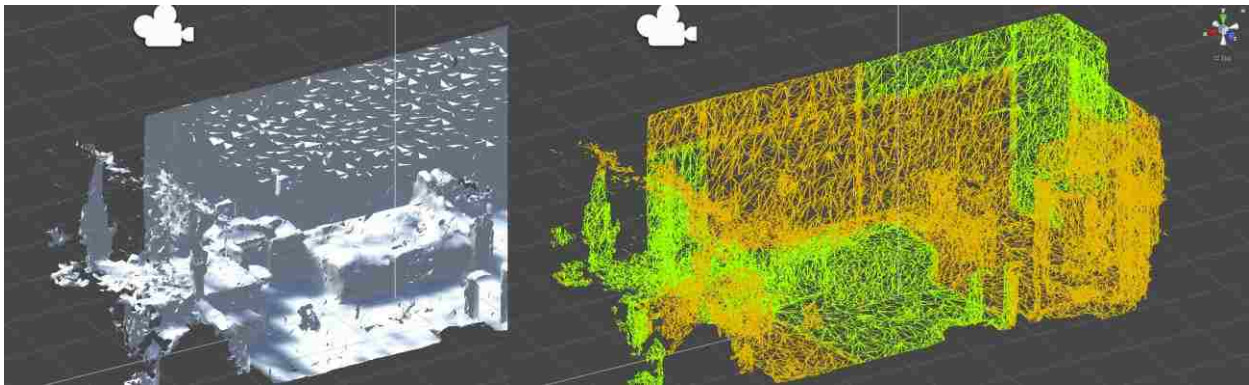


Figure 8 A comparison of a HoloLens scan rendered in white, and with wireframe. The gold and green colors indicate that there are two separate mesh objects. Clearly noticeable are the overlaps between non-connected regions in the scan.

HTC Vive

The Vive is a head-mounted display with a pair of hand controllers. It was chosen because of its quality, excellent hand controllers, and reasonable cost. Any decent gaming computer is fast enough to drive the Vive at rates needed to combat simulator sickness.

The Vive controllers are shaped somewhat like bicycle handgrips, with a pressure-sensitive trigger button at the user's index finger, grip buttons along the handle that can be operated with fingers and/or the thumb, a pair of small thumb-operated buttons on top (one of which is reserved for the Vive system, unfortunately) and a trackpad operated by the thumb.

The trackpad can detect touch, and the location of the touch, as well as respond to pressing like a button. Developers can use this control in an infinite variety of ways. Early in the development of this project, we designed a rotary menu system that used the trackpad like a knob on a washing machine that could be turned to select tools. Ultimately, this was too uncertain in operation, and users too easily changed tools when not intending to.

The REVRSS HUB

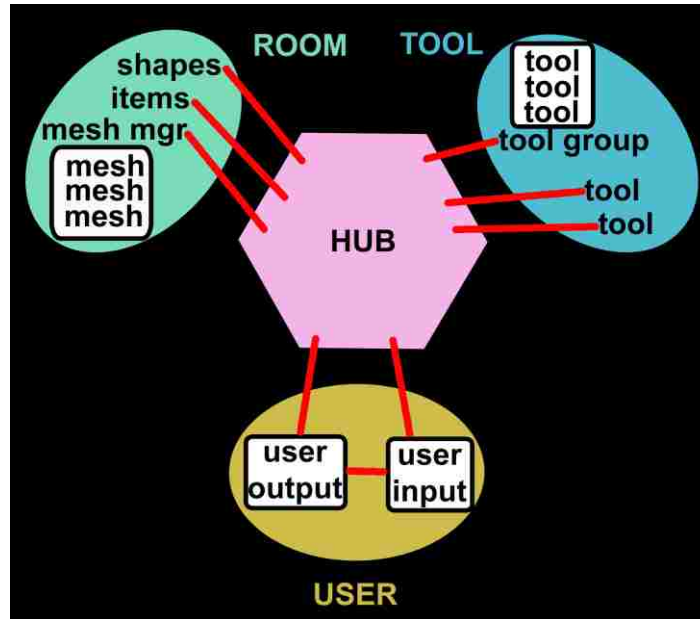


Figure 9 REVRSS Architecture: command and communication flow

The hub is a central component of the REVRSS architecture, though it takes no active role in the actions performed. Data is fed to the hub from the user, tool, and room sections. Those sections access the information as needed, without direct access to other sections. This supports anonymity, with no section reliant on an understanding of any other section.

Current Tool Interface

One critical component in the hub is the Current Tool Interface which routes user input to the tool currently in use – this is a C-sharp interface, and it behaves like a function pointer in C. When a call is made to `currentTool.action()` with `currentTool` set to `BoxSelector`, that call is routed directly from the user to the tool as if the call were `BoxSelector.action()`.

Mesh Management

Another critical section in the hub is for mesh management. A tool, or the user, does not access the mesh directly, but calls a function in the hub. These hub functions then make calls to each of the mesh objects that make up the scanned room. So, a request from the BoxSelector to highlight the vertices it surrounds is made to `hub.PreselectedVertexListFill()` and the hub iterates through a list of mesh managers, calling `mm.colorPreselect()` on each mesh. The mesh queries `hub.currentTool.isItInSelector(vertex)` which, in this example, asks the BoxSelector to determine if that vertex is inside the selector. The result makes its way back to the mesh, which sets the highlight status of the vertex appropriately.

Messages

The hub also stores text messages sent from the tools and room. The user section accesses those message only if it is desired to display them. Likewise, notes from any part of the system may be sent to the hub to be saved in a time-stamped text file. This is used for instrumenting the tools for user testing.

Tool Control Panels

The line between sections can be a bit blurry. The tool control panels are directly accessed by the user, and act directly on a tool. Logically, then, they are a part of the hub. Programmatically, it really makes no difference where the control panel code is placed, so scripts can be kept with the tools for convenience. For proper positioning for the user, the visual element of the control panel is a child object of the user.

The REVRRSS ROOM

The group of data and functions that most closely aligns with scanned meshes, inserted meshes, and primitive items is held in the room section of the system. Any objects created are kept in sub-sections of the room for easy differentiation: created shapes are kept as an ITEM; scanned meshes are held as a SCAN; replacements for flat surfaces are QUADS. Each of these components of the room has a set of behaviors that allow operations, and provide information.

ITEM

An ITEM can be moved, scaled, and rotated. An ITEM cannot be modified by the mesh deformation tools. An ITEM may be a shape built from scratch by computer code, or a mesh from a file. Items can be destroyed at will by simply grabbing them and then releasing them at a point behind the user's gaze in VR, or with a delete button in KVM.

SCAN

A SCAN can be deformed, have parts of it removed or duplicated, but cannot be scaled or rotated, as it was desired to preserve the size and aspect ratios of the original scan. The scan is a 3D mesh object which may be in one large piece, or several smaller pieces. The scan may even be composed of mesh objects from different sources. The REVRRSS mesh management system treats them all as one single large mesh, hiding the gymnastics required to overcome limitations imposed by Unity, Hololens, and the graphics card.

Numerous useful data structures are kept up-to-date for each mesh, and accessed through the hub. There are lists of vertices with coordinates transformed from mesh-specific internal

coordinates to world coordinates so that they need not be repeatedly transformed for every test. There are lists of vertices indicating their highlighted and selected status. For development, and for users who want to know, the number of vertices and triangles in a mesh, which mesh is currently hit by the main aiming ray, and other bits of information are kept about the mesh.

One very useful feature is a graph structure that represents connectivity among vertices. A list of lists, with the index of the outer list equal to the vertex index is first built. Then the list of triangles is read through. For each vertex of a triangle, the other two vertices are added to the inside list for that vertex. In this way, every vertex has a list of vertices it is connected to by an edge. The count of vertices gives the degree of connectivity for each vertex, should it be needed for some feature. This vert-graph is used by the selectors to highlight connected portions by the spider selector, detailed later.

This graph structure also generates a list of sub-meshes with lists of all vertices contained within, and a list of vertices showing the sub-mesh to which it belongs. Another list with vertex number as the index contains lists of triangle indices of which that vertex is a part. Another structure keeps information about edges as pairs of vertices. When an edge belongs to exactly one triangle, that edge is a boundary of the mesh.

QUAD

A quad is a flat plate with four vertices and two triangles. In REVRSS, quads are used as replacements for flat surfaces such as the walls, floor, and ceiling. Quads can be subdivided by a special tool, and removed by another. Quads are made by specifying all four corners, or dragging out a rectangle constrained to a surface, or automatically by the WFC toolset.

The REVRSS TOOL

In REVRSS, a tool is anything that can accept some user command and then affect changes upon the scanned room. Any tool may also send a command to another tool. Each tool has a control panel that initiates actions, is capable of displaying which tool is chosen, and what that tool is configured to do. The control panels are designed such that a user activates a control which then sends a command to the tool. The tool then sends a command back to the control to indicate that it has been activated. It is permissible to directly access the tool, command some action, and the tool will then activate the indicator on the control panel. It should be noted that the tools will do their work properly even if the control panels are hidden from the user. The actuation and indicator methods are not important to the usage of the tool.

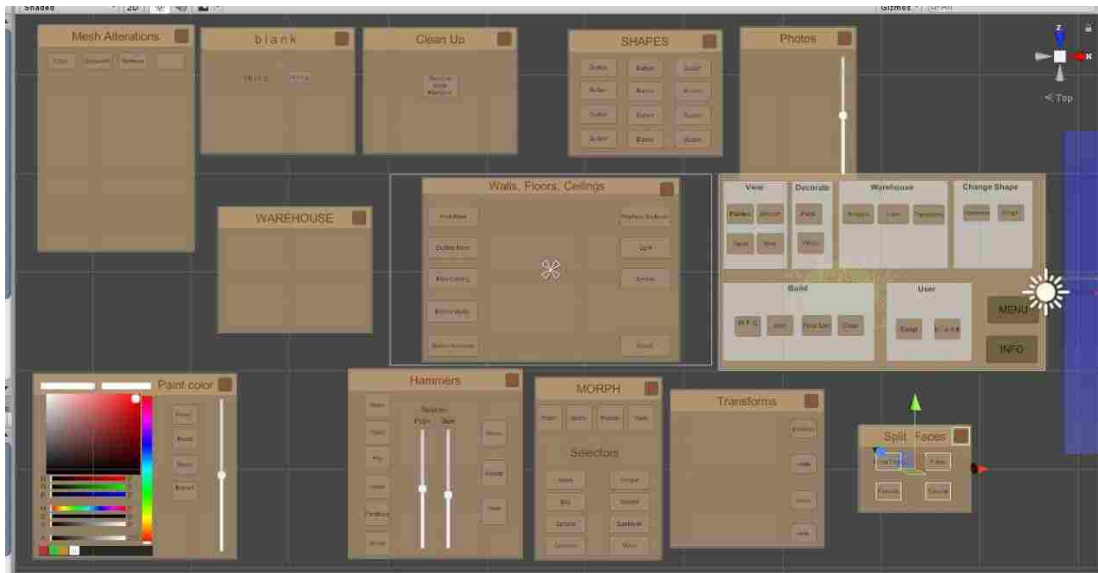


Figure 10 Tool control panels, with the VR main control panel nearest the sun icon.

There are currently tools for selecting vertices, deforming the mesh, splitting triangles, and choosing different “skins” for the mesh. There are tools for deleting, copying, and inserting sections of mesh. There are tools for painting and decorating the mesh. There are tools for removing small-vertex-count sub-meshes, for finding boundary edges, and for identifying co-located vertices. There are tools for creating and subdividing flat plate quads. There are tools for finding large flat surfaces, and marking room corners, and several other tool types. These are detailed in chapter 4, the REVRRSS tool set.

The REVRRSS USER

The user section of the architecture hides the actual actions of the user from the rest of the system. Any form of action the user may take, such as mouse click, gesture, or voice command is translated to action families, with each having a starting, doing, and ending segment.

Thus a CTRL-Left-Click on the mouse will generate three behaviors: CLC_down(), CLC_hold(), CLC_up(). The hold behavior will trigger repeatedly for each frame in game engine parlance – about 90 times per second with a good gaming computer. The action of squeezing the grips on a Vive controller may generate the same sequence of behaviors. A voice command or gesture could do the same.

The user section also pushes information up to the hub for use by other tools. A main aim ray stores the origin and direction of either the mouse’s onscreen location projected into the scene, or the user’s main hand origin and pointing direction in VR. On the main aim ray is a point, at a user-adjustable distance, called the main aim point. The main aim point is where tools are attached. In VR there are also aiming rays for the secondary hand, and the user’s view.

The Vive controllers are handed – that is, the software in Unity knows which hand is the left and which is the right. It would be unfair to left-handers everywhere to force them to perform actions with the wrong hand. REVRSS takes care of this by abstracting the idea into a main hand, and a secondary hand. A main hand takes input from either the left or right, whichever is the dominant hand for the user. The secondary hand takes its input from the other. There is a user control for swapping hands. The rest of the system has no interest in which hand is which, and takes its input from main or secondary as desired by the developers.

Control Activation

Activating a control on the desktop means clicking a mouse button or pressing a key. In VR, the user aims their secondary hand at a button, and pulls the trigger. Control activations could be done in any possible way; designers need only call the proper method in the tool control panel. The panel itself could be of any design, as long as it implements the control methods. The panel need not even be visible to the user, as would be the case for voice command with sound or voice response.

Control Indicators

Indicators displaying tool status are treated however the user interface designer wishes. In the current incarnation, REVRSS simply shows the control panels as movable dialog boxes on the desktop screen, and as floating widgets attached to the user's main arm in VR. Lighted buttons indicate which tool is in use, and which function that tool is engaged in. Messages, such as number of selected vertices from the scan, can be ignored or displayed in whatever way the

user interface designer desires – these are simply text message held in the hub and accessed or not by the user section.

Maneuvering

The user needs to see what they are doing while editing the scan. Corners, furniture, and distance can hide important features. The user must be able to move about the room. In the KVM system, the keyboard arrow keys provide for panning and tilting. The numeric keypad 2, 4, 6, 8 allow for translation down, left, up, and right. The numeric keypad - and + keys travel into and out of the screen. In VR, the Vive system allows for actual walking in a limited area (maximum of 5 meters by 5 meters). Beyond that, a teleport feature is available by pressing the touchpad on the secondary hand. Any locomotion technique can be used without regard for any other part of the REVRRSS system, as it acts only on the user section.

User Scale

For work in very large or very small scanned areas, it may be beneficial to change the size of the user. A giant can work quickly in a vast building without needing to move about as much, while a Lilliputian may be able to more delicately do fine work. While user scale is implemented in REVRRSS, there are currently no VR controls for it. Adding a control is trivial. User scaling, like maneuvering, is independent from all other parts of REVRRSS; it initiates in, and acts only on, the user. A user can change their location, orientation, or size and have no effect on any tool, and no effect on the room.

CHAPTER FOUR: REVRRSS TOOL SET

The tools available in REVRRSS fall into categories based on the severity of their impact on a mesh. Inconsequential changes are those where the surface appearance changes, or the object's position, scale, and rotation may change, but the mesh remains intact and does not deform. A simple change is one involving deformation of a surface. A complex change alters the mesh topology by adding or removing vertices and triangles.

Mesh Appearance

The View tools allow an inconsequential change in the appearance of the mesh. We can set the entire mesh to display as a wireframe image, which is handy for understanding the way the mesh is built, and how it is divided into sections and subsections. There is a setting used to show highlighted and selected portions of the mesh. The painted setting shows how the mesh has been decorated with paint and texture tools. Another setting, used only in testing, colors the sub-meshes according to how many vertices are contained within it. Any number of textures could be added to the set presented here.

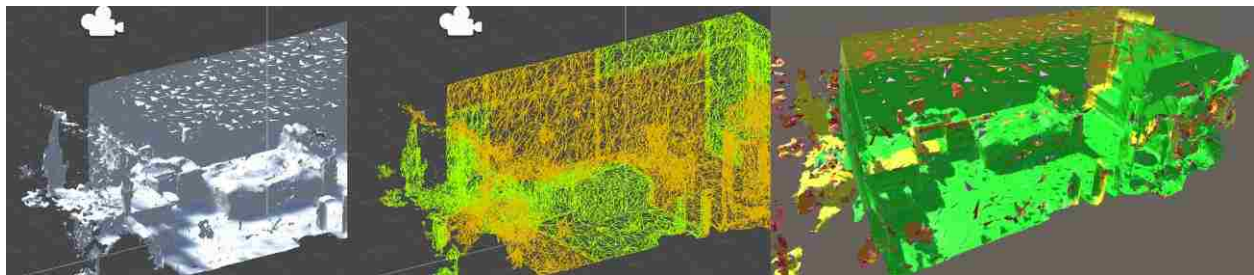


Figure 11 Scan fully rendered, wireframe, and color for number of sub-mesh triangles

Paint and Texture

The Decorate set of tools includes the ability to place stickers on any mesh, and to paint on its surface. This ability is courtesy of Unity Asset RealTime Painting. Decoration tools are less than useful at the moment, until a suitable method for “unrolling” the complicated mesh is implemented. It is trivial to flatten out a cube, fairly simple to flatten out spheres, cylinders, or capsules, or any regular geometric shape. Finding a way to flatten out the unruly mesh formed by scanning a real room has so far eluded us. Though not yet helpful, there is a set of painting tools that include a pencil and brush with adjustable size, a bucket for entire mesh coloring, and a sticker that places an adjustable size image on the mesh.

Warehouse

The Warehouse group gives us the ability to bring in objects, which may be meshes or shapes. Currently, the warehouse is a simple set of buttons in both VR and KVM. The shapes are a collection of primitive elements that can be created at run time. The cube, sphere, capsule, and plane are all part of the Unity primitive objects. Others, such as the cone, are created in software as needed. Any shape could be added. Transforms allow positioning, scaling, and rotating the warehouse objects.

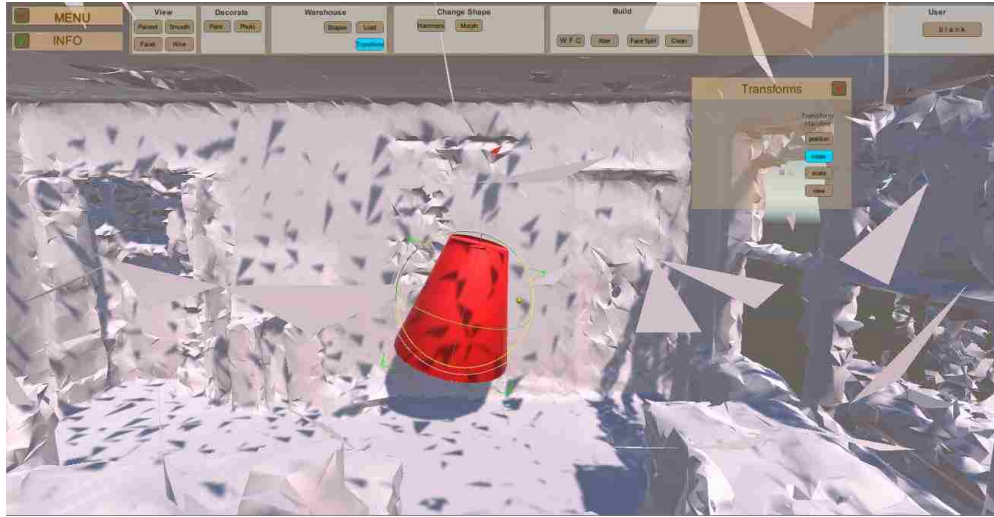


Figure 12 a truncated cone has been added, and then rotated using the transforms

Transforms

For the desktop, transforms are handled by a Unity asset called RTH, Runtime Transform Handles. In REVRRSS this is implemented so that when users hold the SHIFT key, a selected object is moved. Arrows appear to cue the user to the tool. To produce a rotation, users hold the CTRL key, and a set of circles appear. For scaling, the ALT key results in the appearance of the 3-cube scaling tool.

The VR version of REVRRSS handles rotation, scaling, and positioning a little differently. When the grip buttons on the main hand are held, a chosen object becomes attached to the hand so that moving and rotating happen together. Holding the grip buttons of both hands activates scaling, and by moving the hands together or apart, acts to squash or stretch the object in three dimensions. The RTH widgets appear to help the user understand the transform mode.

Vertex Selectors

While not terribly useful alone, selectors form an important part of other tools. Selectors in REVRSS were developed as need arose to choose vertices in greater numbers, and in connected groups. They were developed in roughly the order given in the explanations below. For each of these, users perform some action that causes the vertices to be added to a list kept by the mesh manager, and to be highlighted. An anti-action removes vertices from the list, and un-highlights them. The selectors are usually attached to the user's pointing device, so there is no special action for positioning. On the desktop, a regular left-click selects, and a right-click deselects. The same actions on the Vive are made by pulling the trigger on the main hand to select, and pushing the small top button to deselect.

The attachment point can be moved toward or away from the user by spinning the scroll wheel on the desktop, or by pressing on the track pad on the Vive controller. This behavior is commonly referred to as a "fishing reel" in the field. On the desktop, scaling is accomplished by holding the ALT key and dragging handles that look like small cubes. Rotations are performed by holding the CTRL key and dragging a virtual trackball. With the Vive controllers, the grip buttons grab the selectors for rotation. Both grip buttons can be held and the hands moved about to scale the selector.

Early work on the selectors placed spheres at the chosen vertices, but it was found that this technique was too slow with more than a few dozen vertices. Work to understand the mesh texture UV maps led to a giant improvement in highlighting. Now a simple texture map is used, and lists are kept that indicate the highlighted and selected status for each vertex. By adjusting

the UV coordinates for each vertex, the coloring for highlight and selected status can be shown in real time. Regions currently inside the selector are highlighted in purple, and regions that have been selected are colored gold.

Single Select

The single select is exactly what it sounds like. Users aim at the surface of a mesh, and trigger a selection action. The nearest vertex to the aiming point is highlighted. The anti-action will deselect a vertex. Holding the action or anti-action causes vertices to be continually added or removed from the appropriate list as the selector is swept over the surface.

Spider Select

We wanted a method to select vertices that were connected by edges, and to select the vertices connected to those by edges, and so on to some reasonable number of hops. Thus the “spider selector” was born. A graph structure to accomplish this is detailed above, in the mesh manager section. Named for the creepy crawly appearance as it is swept along the mesh as well as the web of connected edges that it forms, the spider selector works quite well. The user aims a pointer at a vertex, and the spider collects all the points within a specified number of hops. Distance (number of hops) from center is stored, so that a tool may deform the mesh into shapes based on distance from center if desired by implementers.

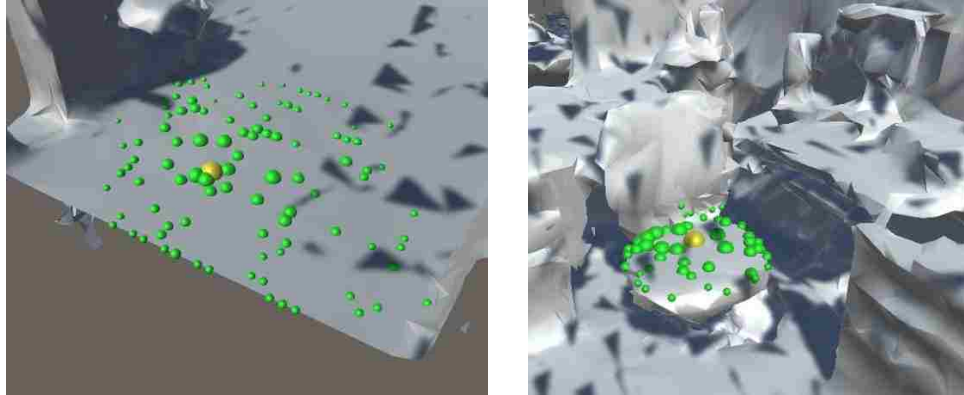


Figure 13 The original spider selector finding connected vertices out to three hops. Smaller spheres indicate greater number of hops from the graph center

Sub-mesh Select

Taken to its extreme, the spider selector can collect all vertices connected by any number of hops. This selects an entire sub-mesh. The sample room used for testing has thousands of sub-meshes, ranging in size from one triangle to thousands of triangles. Since the data list is kept in the mesh manager, and there is also a list of member vertices for each sub-mesh, the spider is not needed. The tool simply highlights all members of the sub-mesh list.

Whole Mesh Select

This tool selects an entire mesh – which may be the entire room, or just one of many scanned parts. The action of selecting sets all of the vertices in the selected list of the target mesh to highlighted. If the room is composed of more than one mesh, only one at a time will highlight using whole mesh select.

Enclosure Selectors

Choosing connected vertices will fail at sub-mesh boundaries, and users will wish to select an entire region even if not connected – even if in different meshes altogether. The enclosure selector family does this. There are currently enclosures shaped like cubes, spheres, and cylinders. For any of them, a simple test of every vertex in the mesh reveals whether that vertex is within a bounding box of the enclosure. For a cube, this is sufficient. For a sphere, cylinder, or most other shapes, a little math determines if the vertex is inside the enclosing shape.

To simplify the math, the vertices' world coordinates are taken from the mesh manager, and transformed to coordinates relative to the enclosure. Scaling the enclosure may change a cube to a thin plate, or a tall column, but still has internal max and min values of one unit in size. Rotations may change the orientation of the selector, but internally it is still a unit size sitting upright. A sphere needs a check for distance from center. A cylinder needs a check to see if the X and Z values are within the base circle of the cylinder. Other shapes require similar checks.

Selector tools do not act directly on the vertices. The selector asks the hub to command the meshes to iterate through all vertices and ask the selector if the vertex is inside the selector. Rotation, scaling, and positioning are as explained at the top of the Vertex Selector section here.

Any shape could be included, developers need only supply code to build the shape, and code to determine whether a coordinate lies within the intended shape, and to provide some simple controls for the new selector. These enclosure selectors all have a wireframe texture applied so that users can see where the shape lies, and what is in or near it. These shapes can select continuously, as does the single select, by repeating the select action.

Surface Analyzer

It should be noted that there is a built-in feature that can correct the user's placement of large flat surfaces. In order to determine the theoretically correct position of a large flat area for the WFC tool set (detailed later), the surface analyzer finds the geometric center of the area by averaging all the vertices within a thin box selector.

When the user gets close enough to the correct placement, the tool could be made to snap to the correct position by itself. The surface analyzer method also finds the average normal direction of the surface, which can be used to snap to the proper orientation of any surface that is not quite aligned to the cardinal directions. This can also be used to exclude vertices where a triangle's normal is not within some window angle around the average surface normal direction.

Surface Deformation Tools

Perhaps the most common action to perform on a mesh is to deform the surface – to change a lumpy appearance to flat, to correct the curvature of a surface, or to sharpen a corner. REVRSS has a few tools for manipulating vertices so as to change the contours of surfaces in the mesh. Surface Deformation is a simple edit, meaning that the mesh is not torn or altered topologically, but the vertices will change relative position within the mesh. After any deformation, some of the data held in the mesh manager must be updated. This is handled automatically by the mesh manager. As with the selectors, the surface deformation tools do not change the mesh, but only indicate to the mesh manager how the vertices are to be moved. This is usually done with vector deltas, which are added to each chosen vertex by the mesh manager.

Vertex Moves

The easiest deformation is to simply move the chosen vertices in lock-step with the user's pointing device. This is called "follow me" in REVRSS parlance. Another method is to move the chosen vertices along isometric axes or surfaces. On the desktop, these are performed with the positioning handles supplied by RTH; with VR a constrained movement technique is used.

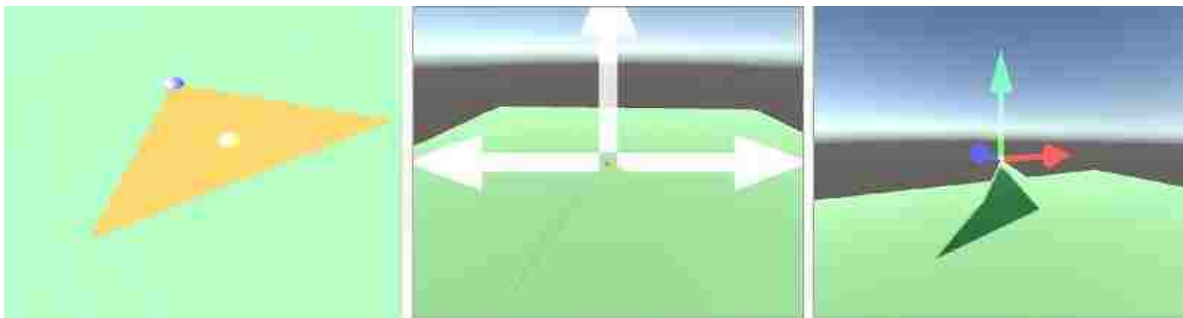


Figure 14 Early highlighting, and simple vertex moves

Hammers

Hammers are the result of experiments with tools that select mesh vertices and perform a deformation in one user action. Each has a short green cylinder that highlights the mesh within it, and a gold hammer head that shows the final shape after deformation. Any final shape could be added; in REVRSS we have a flat surface, a cone, hemisphere, paraboloid, and sinusoid. The hammers are always attached to the user's pointing device, and can be rotated and scaled in the same way that selectors are manipulated.

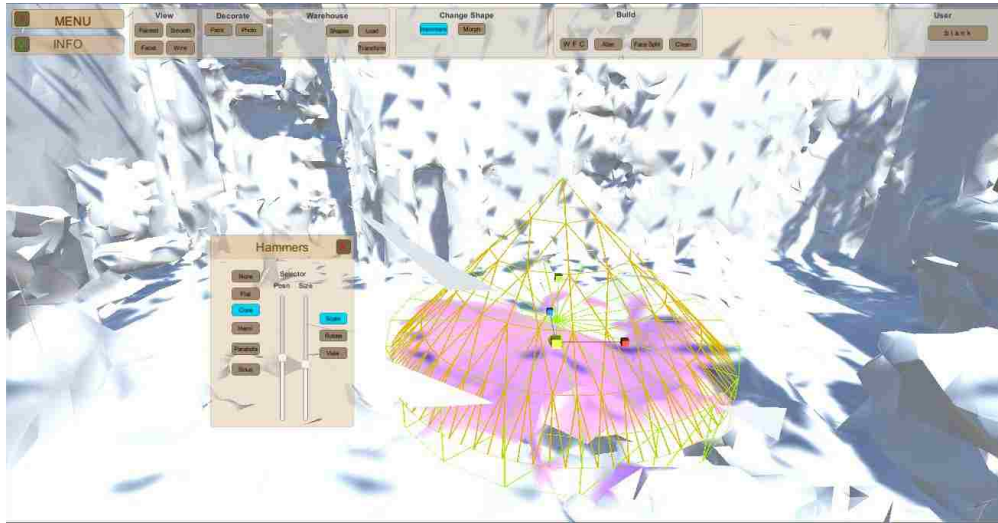


Figure 15 a hammer in KVM, showing scale widgets and purple highlight

The hammers are a descendant of an ironing board technique, which was designed so that a user could place a flat surface (the board) and then use a tool (the iron) to gradually smooth the surface. This is a good idea in need of faster technique, and it was not included because updates were too slow, resulting in a terrible white-out condition which was not pleasant for the user.

The desktop system has a control to set the separation between the selection disk and the final hammered shape, and to change the thickness of the selection disk. This feature proved problematic with the VR system, so has not yet been implemented.

The selection disk of the hammer is actually just a backdoor use of the selector tool, and operates in the exact same way. As with other tools, the mesh manager iterates through the vertices, checks for inclusion in the selector portion, and asks the hammer to update the position if so. The hammer action operates on local, inside-hammer, coordinates then transforms them back after manipulation. The vertices are first set so that the local y-component of the vertex is zero, then a simple bit of math specific to each hammer head moves the vertex up or down to the

surface indicated by the shape. The vertex coordinate is then transformed back to mesh-centered coordinates, and the mesh manager updates the mesh info.

Straightest Line

It was hoped that by using the vertex graph, we could find a shortest path through two vertices. The idea was to straighten the meandering intervening vertices by adjusting them to conform to the line between the chosen vertices. This project was stymied by the discontinuity at sub-mesh boundaries. It may be possible to resurrect this tool, once we have the ability to treat the mesh as a whole after joining sub-meshes.

Mesh Topology Alteration

Deforming the surface of a mesh keeps all of the triangles in the same relationship to one another, in the same way that crumpling a piece of paper keeps it intact. Ripping a hole in the paper, or adding to it, is analogous to removing or adding vertices and triangles to a mesh.

Cut, Copy, and Paste

As is the case with most computer software, Cut, Copy, and Paste actions are useful features. Here the user first selects a mesh region with any selector, or combination of selectors. Users then activate controls to remove the region, or to make a copy of the selected region while leaving the original in place. Users can grab and move the new pieces as desired. Transforms may be applied to rotate and scale the new mesh, too.



Figure 16 removing/copying a chair: selected, cut, copied.

Triangle Splitting

Splitting the triangles to add detail can be accomplished in several ways. Splitting the triangle at its geometric center is simple, and does not affect other triangles. Splitting a triangle on a single edge (usually the longest edge) requires splitting the adjacent triangle. Splitting a triangle on all three edges requires the adjacent triangles to be split as seen below. Splitting requires adding new vertices to the end of the vertex list, adding new triangles to the end of the triangle list, and then setting vertex indices in the correct triangle in the list.

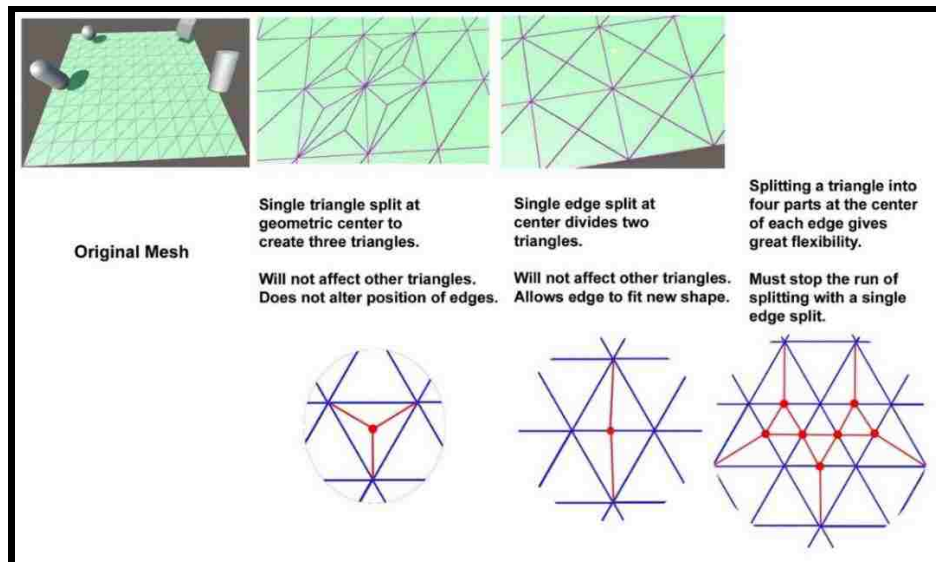


Figure 17 Triangle splitting techniques

Sub-mesh Join

While working to understand the mesh formed by Hololens, it was thought that there must be a large number of vertices sharing the same coordinates due to the sub-meshes, and the division of the whole mesh into parts. We wished to remove these redundant vertices and join together the sub-meshes. A brute force method with $O(n^2)$ would have taken many hours to complete. An octree could divide the vertices into sections, and make searches simpler.

As an approximation to the octree, we divided the mesh into a set of boxes such that each dimension was divided by twenty to seventy, making 8,000 to 343,000 boxed sections. Each vertex was assigned to the box surrounding its coordinates. Subsequent $O(n^2)$ searches could then be done on a small number of vertices within a box.

Table 1 Time to identify co-located vertices by divisions per axis, in seconds

# divisions /	Seconds to complete	# of compares
20	6.45	3,884,687
30	6.24	1,652,237
40	6.26	924,227
50	6.26	599,637
60	6.25	435,985
70	7.51	328,687

Surprisingly, this technique took only a few seconds to complete, and was relatively insensitive to the total number of boxes within a range. This may be due to the overhead of creating the boxes and assigning the vertices to them. The speedup was very good, but not suitable for a real-time interface. The co-located vertex finder is better suited to pre-processing

the mesh. Ultimately, the technique was deemed not yet useful, but may be resurrected for some future tool implementation.

Other Triangle Modifications

There is a control to remove all sub-mesh components formed by a single triangle. The result of this action is to De-clutter the room, and requires no user action except to push a button. The tool could be altered to remove sub-meshes up to a user-specified number of vertices.

Quad Creation

A four-vertex, two-triangle quad can be a useful thing. So far, the best use is for the WFC tool, described next. During development, a couple of methods of creating quads were built. These might be handy for some future application.

One method is to set four corners, and then the tool fills in the space with a quad. The finished product will not be flat if the four corners are not in a plane. The final shape need not be square, or even rectangular. As the name suggests, these are quadrangles. A tool able to snap to existing vertices or surfaces would be quite handy in placing quads.

A second method is the drag-out. Users activate a control, placing one corner, then drag across space, releasing the control to place the opposite corner. This tool is constrained to work on an imaginary plane, which could be defined in any way.

Walls, Floor, Ceiling

The scanned mesh from Hololens was designed to quickly locate the device's position in a real room. Any identifiable feature in the room becomes a vertex, and triangles join those

features. The result is that large flat areas can have many more vertices than needed for a model of the room. Removing unneeded vertices decreases the size of the mesh, makes large flat areas smoother, cleans up extra clutter, and makes editing faster. Removing extra vertices gives us space to add vertices where greater detail is desired, without slowing down the rendering.

The WFC tool, (Walls, Floor, Ceiling) is used to replace the six (or more) surfaces with simple quad objects. WFC is a compound tool made of other tools and slight variations on those tools. The control panel used in development has buttons for each of the sub-tasks needed to replace the walls. During user testing, a meta-control panel with a single “next” button operates the several parts of the WFC panel. This shows some flexibility in the architecture, and how it can hide complexity.

The WFC toolset guides users through a few tools in several stages to arrive at a finished result. The box selector is used to find flat surfaces. A corner-finding tool sets the rough outline of the room. The box selector is again employed, this time to refine the surfaces for replacement. Finally, quad building and dividing tools refine the new replacement surface, and are then replaced by finished surfaces.

First: Find the Floor

The box selector was modified with an additional texture to make it transparent, curing a problem where the green wireframe hid the user’s view. The selector appears near the floor, and users move it up and down to find the point where the floor is flooded with purple highlight, but without taking away wall area or furniture bases. On the desktop, users interact with the RTH

handles to move the selector. On the Vive in VR, the grip buttons on the main hand grab the selector, and moving the hand moves the selector.

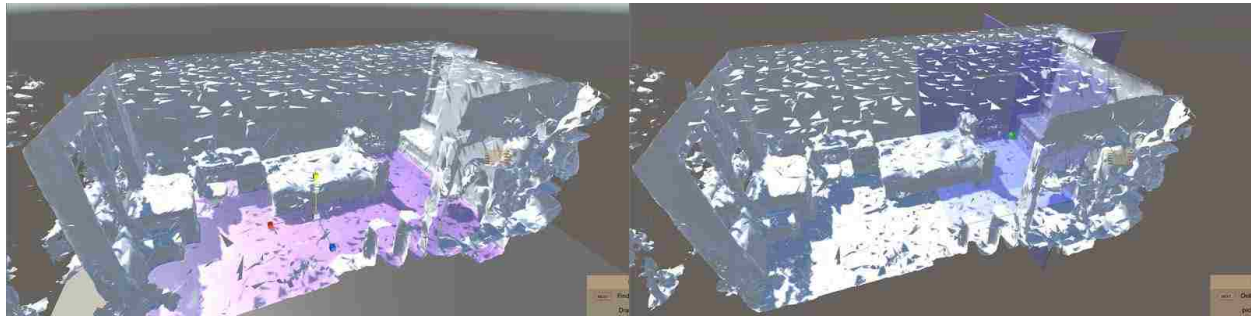


Figure 18 WFC steps 1 and 2. Left: Find the floor. Right, mark the corners

Second: Mark the Corners

A corner finder tool has a target box that is constrained to the new found floor, and a pair of transparent walls to help align it into the corner. Pulling the trigger or clicking the mouse drops a marker ball in the corner. There is no need to be exact, as positions will be refined later. Each subsequent corner mark also drops in a transparent tube connecting it to the previous corner. When all corners have been marked, users activate the “next” control.

Third: Find the Ceiling

The same tool that was used for finding the floor is used in the same way to mark the ceiling. The highlight floods the ceiling with purple, and users adjust the height to remove most of the ceiling without removing desired detail. When users activate “next” a cage of transparent blue tubes appears, outlining the room.

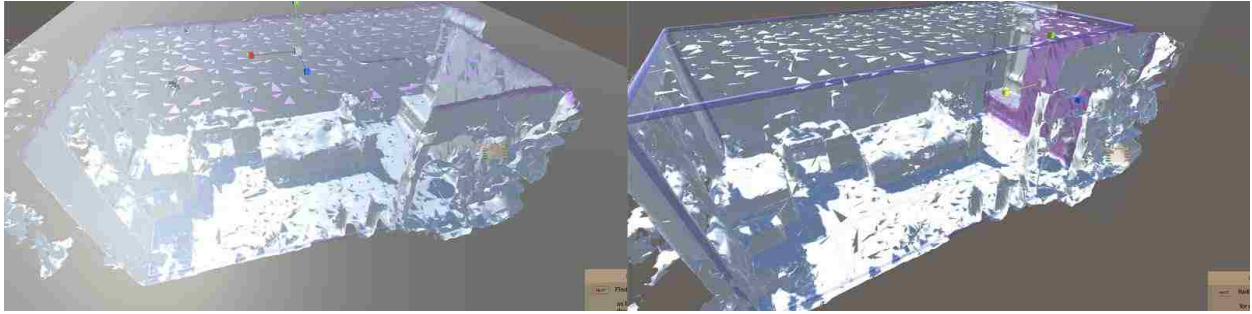


Figure 19 WFC steps 3 and 4. Left, Find the ceiling. Right, Refine the walls.

Fourth: Refine the Walls

Next, the floor and ceiling finder (box selector) is used in the same fashion, but sideways, to flood the walls with purple highlight while keeping furnishings unaffected. Users are guided wall by wall, and when all walls are complete they are given the chance to go around again. The updated cage is constrained to move along only the appropriate dimension. This refinement means that corners do not have to be placed exactly. A seasoned user will make use of this knowledge and place four corners in a few seconds. When the “next” control is activated, the corner markers disappear, and a gold highlight appears on the major surfaces, showing what has been selected for removal.

Fifth: Refine the Selection

There will be areas that have not been highlighted in gold, since the walls can be rough and have a thickness greater than the box selector. There will be regions of useful mesh that the user wishes to keep, so as not to unduly remove detail. An enclosure selector control panel appears, set at first to the unit box. As described earlier, users may click the mouse or pull the trigger to add to the gold selected portions of the mesh. To remove the gold selected marking,

users may right-click the mouse or hit the top button on the Vive wands. Seasoned users will know how much detail and trim area to leave intact around windows and other features to avoid gaps in the final model. Going ahead to the “next” part of the sequence deletes the selected mesh, and places one large quad on each of the flat surfaces defined earlier.

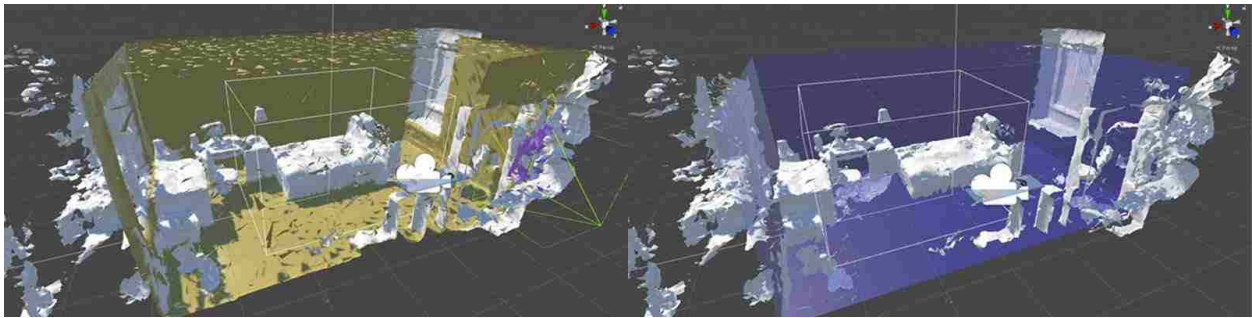


Figure 20 WFC steps 5 and 6. Left, refine the selection. Right, new wall guides.

Sixth: Divide and Delete Quads

The large quads will undoubtedly cover doors, windows, and other room features. A pair of tools will cut the big quads into smaller pieces, and delete the small quads that are not wanted. Users can freely switch between these two tools by activating a control, or do all of the dividing and then do the deleting, it makes no difference.

To divide, or split, a quad, a tool highlights the normally transparent blue quads in green when struck by the pointer. A black rod appears across the quad, always perpendicular to the edge nearest the pointer. Clicking the mouse or pulling the trigger splits the highlighted quad.

Deleting a quad uses another tool with a red box on the pointer. When aimed at a quad, the normally transparent blue quad highlights in green. Clicking the mouse or pulling the trigger deletes the highlighted quad.

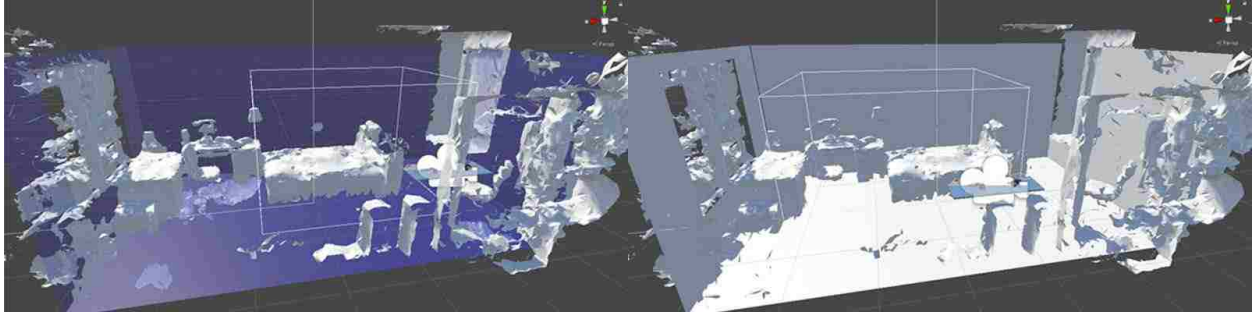


Figure 21 WFC steps 6 and 7. Left, wall quads divided. Right, finished product.

Seven: Finish

When the user is satisfied with the quads, the “next” control will change the texture on the transparent quads to the featureless white appearance, and the WFC replacement task is done.

CHAPTER FIVE: USER STUDY

Now we turn to the user study, where we test our tools and get some feedback from the users . Comparing a desktop interface with a VR interface made to use the same tools in the same task will determine which user prefer, and show that the time needed and quality of work in the two methods. We chose to test only a subset of the full tool suite. The “Walls, Floors, and Ceilings” (WFC) tool contained everything that we needed to discover about preference and applicability of VR as compared to the desktop environment.

Experiment Design

For our testing, we would have subjects replace the walls, floors, and the ceiling of a scanned room. The testing software was outfitted with the ability to record the time of several actions, and the final position of the flat surfaces in the room. Control panels were limited to a

single button labelled “Next.” Users would take a paper survey about their preference, and comfort, with the editor system. Each participant would try the desktop version and the VR version. Since the choice of scanned rooms is limited to one good one, the subjects would be split into those who did the VR first, and those who did the KVM first. We could collect meaningful between-subject information this way.

Testing occurred in the ISUE lab. After answering brief demographic questions, each participant was coached through the process by experimenters, and a set of text prompts on the control panel. After each trial, a set of questions was presented covering comfort and ease of use. After both trials were complete, the participant was presented with preference questions.

User Experience Surveys

A total of ten users ran a test of the WFC tool. Of the ten, only one was female, and all had ages in the range of 18 to 23. This makes for an unfortunately homogeneous group, but will suffice for these informative tests. Studies with a wider range of tools and greater variety of subject, and with many more subjects, may reveal more interesting results about preferences.

We derived statistic using SPSS for the five-point Likert scale surveys. We ran Wilcoxon Signed Rank Tests for significance at an (alpha) of 0.05. Type I errors inside the t-tests are controlled using Holm’s Sequential Bonferroni adjustment

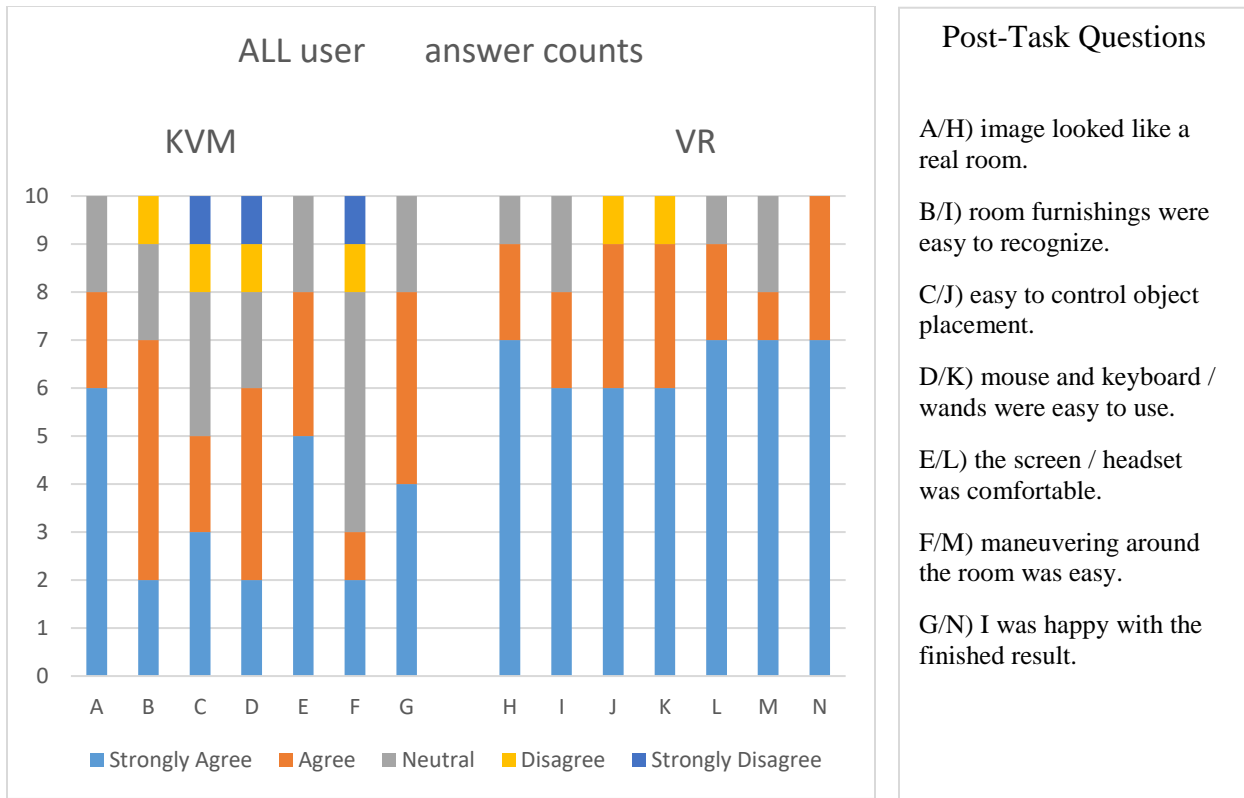


Figure 22 Survey responses by Task

Though the graph suggests that the VR experience was always rated more highly, the statistics package shows significance ($p < 0.05$) only with a few questions. This is likely due to the small sample size of ten subjects. More testing should be done to get more meaningful results.

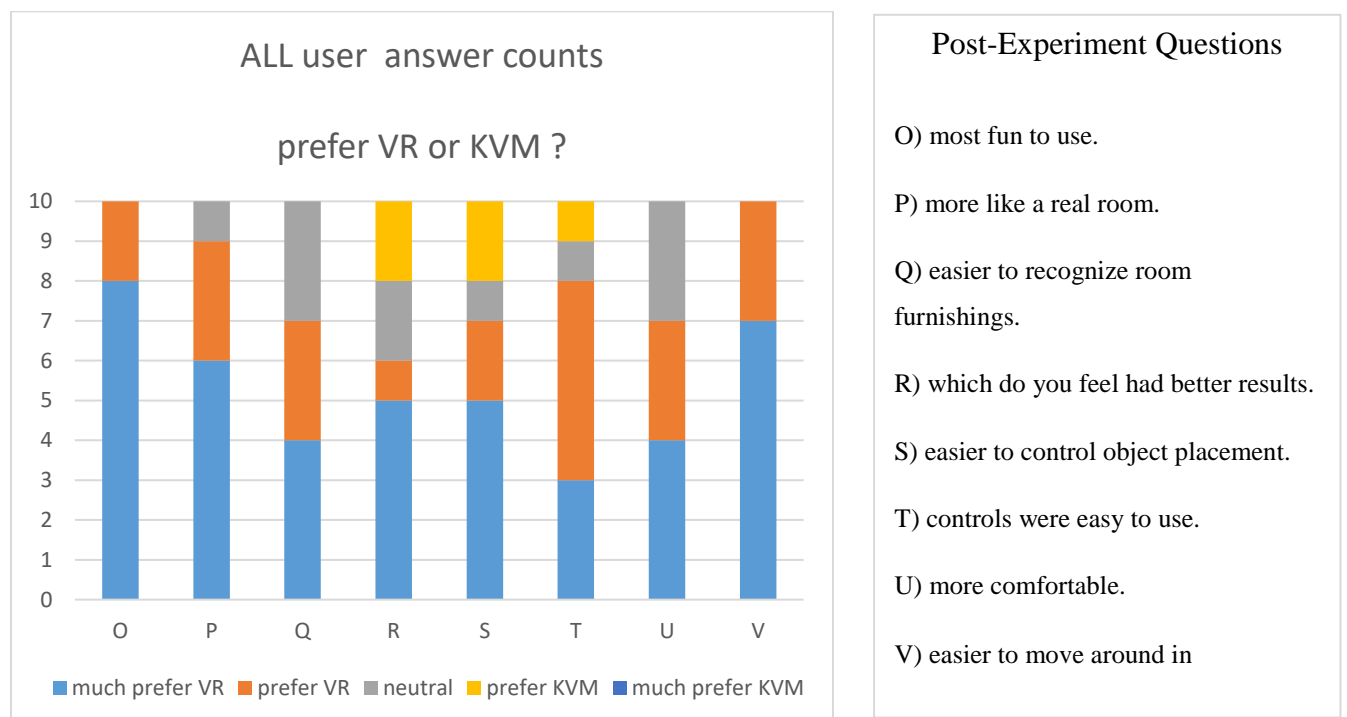
Table 2 Significance of responses, KVM vs VR

	A/H	B/I	C/J	D/K	E/L	F/M	G/N
Z	-1.414	-1.310	-1.496	-2.333	-1.000	-2.214	-2.236
p	0.157	0.190	0.135	0.020	0.317	0.027	0.025

Questions D/K with $p = 0.020$ show that the wands were better appreciated than the mouse and keyboard. Maneuvering, with a $p = 0.027$ in questions F/M, shows that users were happier

with the view and locomotion in VR. Respondents also showed in question G/N with a $p=0.025$ that they thought the finished product turned out better in VR.

After the users had completed both the VR method and the KVM method, a set of statements provoking a direct comparison were presented. Responses show that the users have a preference for the VR system for each of the following statements.



- Post-Experiment Questions
- O) most fun to use.
 - P) more like a real room.
 - Q) easier to recognize room furnishings.
 - R) which do you feel had better results.
 - S) easier to control object placement.
 - T) controls were easy to use.
 - U) more comfortable.
 - V) easier to move around in

Figure 23 All users preference for KVM or VR

The graph above shows that when forced to choose, an almost overwhelming preference for editing the mesh in VR as opposed to KVM. This supports our expectation that VR would be preferred. At least 60% of all the responses were favorable to VR, and “most fun” and “easiest to move around” gained 100% approval for VR. For “had better results” and “easier to control placement” and “controls were easy to use” we saw that only 20% of the subjects prefer KVM.

Performance Evaluation: Placement

A measure of placement accuracy was taken by comparing the geometric center of the user's final position of the walls, floor, and ceiling. This was compared to the geometric center of the scanned vertices replaced by those surfaces. The root-mean-square of the differences from the theoretically correct position is given here. These values are in "Unity Units" which are approximately one meter, meaning placements were off by no more than a few centimeters.

For measured placement errors, we again used the SPSS package. A Shapiro-Wilks Test of Normality showed that some of our metrics were not normally distributed ($p < .05$). For the parametric items, we ran pair-wise t-tests to test for statistical significance. For the non-parametric metrics, we used Wilcoxon Signed Rank Tests, summarized in the table below.

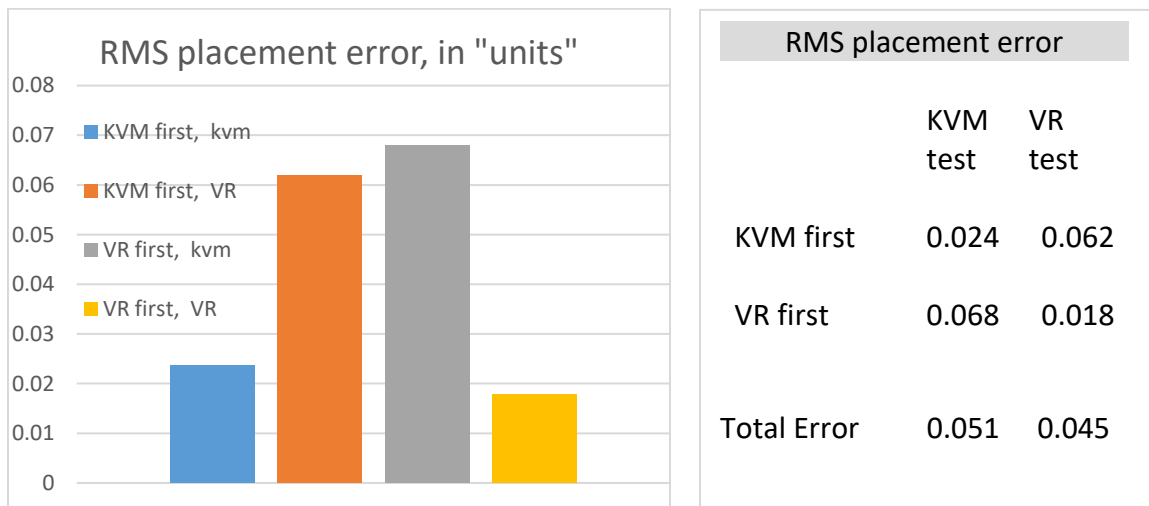


Figure 24 RMS Placement Error, by Group and Task

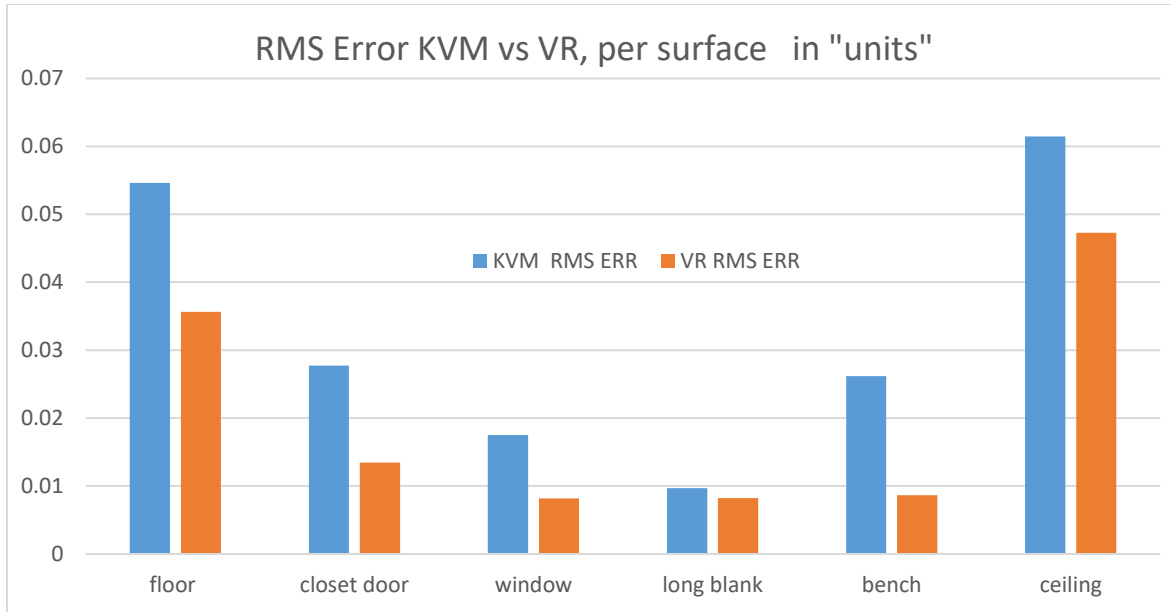


Figure 25 rms placement error for each surface, by task

Though the graph seems to show the VR was slightly better in placement, the SPSS statistics results shows no significant difference in accuracy between the KVM and VR tasks. Finding no significant difference ($p < 0.05$) in the two methods supports our contention that the VR is not worse than the KVM methods for accuracy. By incorporating the surface analyzer's snap-to feature, these errors could be reduced to zero, leaving us with no difference at all between KVM and VR, though testing with more subjects is advised to avoid error.

Table 3 Significance in placement error

	floor	Closet wall	Window wall	Blank wall	Bench wall	ceiling
Z			-0.356	-0.563	-0.771	-0.296
t	-1.181	0.631				
p	0.272	0.545	0.722	0.574	0.44	0.767
Adj.	0.0063	0.001	0.025	0.013	0.0083	0.05

Some odd problems crept in. There is evidence in the data that some users were tilting the surface finding tool, or moving it laterally. This is a behavior that can be forestalled by a little extra constraint in the code. One user teleported while carrying the tool, and moved it to a position far from ideal, which took a little time to correct.

Performance Evaluation: Time to completion

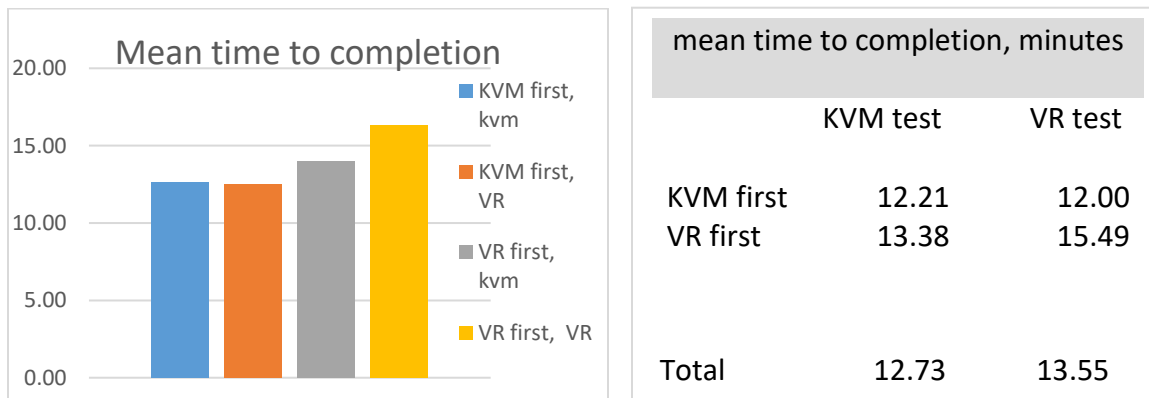


Figure 26 Mean time to completion

The users were timed while performing the task of cleaning up the scanned room. The program tracks time spent on each of the several actions needed to complete the work. The mean time to completion for the two groups (VR first, KVM first) is here plotted against the method used. The VR task probably gave users a chance to enjoy the view, and to be more cautious about placement, so looks a little slower here. A closer look at the times to complete sections of the task shows where the slowdown occurs.

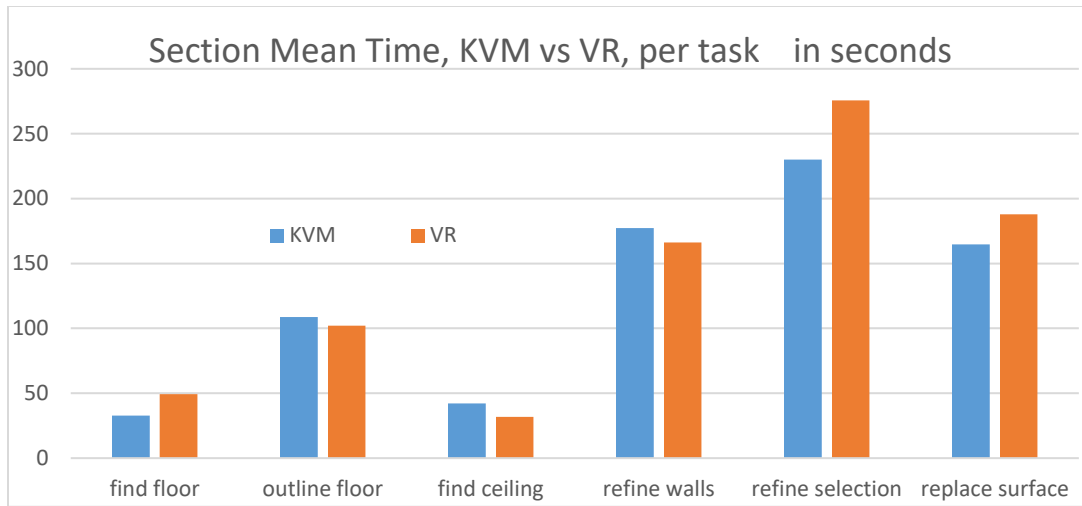


Figure 27 Feature completion time, per Task

The graph suggests that there is only a small difference in time required to complete each task for the two methods. The statistics show no significant difference for KVM and VR. A Shapiro-Wilks Test of Normality showed that only the “find ceiling” task showed significance. Significance for all times disappeared after applying Holm’s Sequential Bonferroni adjustment.

Table 4 Significance in time to complete tasks

	Find floor	Outline Floor	Find Ceiling	Refine Walls	Refine Sel’n	Repl. Surface	Total Time
Z	-1.362		-2.803			-2.666	
t		2.13		-0.449	-2.791		-1.096
p	0.151	0.066	0.0051	0.666	0.024	0.008	0.305
Adj.	0.0056	0.005	0.0038	0.017	0.0045	0.0042	0.0071

This agrees with our prediction that there would not be a big time difference. While VR seems faster for identifying items in the room, and navigating, users seem to be spending a little more time getting things just right when they can see better.

Comments and Observations

The comments made by the subjects tell us quite a bit about what they had trouble with, and what they liked. Experimenters made a few notes while the subjects were using REVRSS, and these notes will help refine what we've got, and suggest new things to try. Some notes are just interesting observations. These notes will help improve REVRSS in the future.

Most of the users worked the desktop controls from a point outside the room. Almost all of the VR work was done inside the room, but one VR user teleported outside of the room for a better view of a corner. There was quite a lot of maneuvering in both versions, something that an experienced user can limit by planning. Only two of the VR users did actual walking around, while the others preferred to stay in a small area and teleport to move about the room. This may be due to the concern about running into things in the real world. Perhaps a larger clear area would have spurred more people to walk. One user noted worry about tangling in the headset cable. A wireless headset system would be worth a try.

A few said that viewing the room in VR was so much easier when all you have to do is turn your head. One person went so far as to say "a million times better," another said, "wow, lots better view," while using VR. One claimed that splitting and deleting the wall quads was easier with the VR view.

A few kept a thumb on the teleport button, causing it to display all the time. This should be changed so that the teleport appears when the button is pressed, and teleports when released. Most had some trouble with the grip buttons – this is just an awkward design of the Vive wands. A couple of people noted that the VR wand sensitivity seemed high, that a shaky hand was enough to throw off their aim. Perhaps a bit of smoothing would help, but the users could simply stand closer to their work to reduce the lever arm. A few subjects poked at the “next” button when in VR, where the control is meant to operate by a trigger pull. Redesigning this interface to meet the user’s expectation is a very good idea.

One, clearly a gamer, noted that the keyboard controls did not follow a standard game layout, and attempted to use key combinations that were meaningless or harmful in REVRSS. Another thought that the keyboard navigation controls were hard to use, as they did not allow fine control like you could get in VR by moving your head or walking.

Users who seemed to “get it” were allowed a bit of freedom to try rotating and scaling the selection cube, and thought it was a great feature in VR. The rotate and scale features in KVM were not special enough to warrant comment, as they are like familiar software.

There was one user with poor spatial awareness who needed much more coaching than others. Two people needed almost no coaching, and seemed to understand what to do right away. Everyone was able to do better at placing objects after the first couple of tries, suggesting that even a little practice makes a big difference. The ceiling was placed poorly on a couple of occasions, and once the floor was out of place. It may be that users did not release a control before moving. In the end, nearly everyone did well.

A few people got an extra wall placement in VR, and one did with KVM. This tells us that better feedback is needed. Haptic feedback in VR would alert the user that a button has been pressed. Better markers, and maybe a counter to show how many corners have been placed, would help. There is an “undo” for the last corner placed, but users had already moved on to the next corner before the experimenter could coach them how the extra one could be removed. A feature that allows the users to drag the current corner marker exists, but was used only once.

Users spent a fair amount of time altering the selection of surfaces for replacement, but not much more than an experienced person would spend. One user enjoyed using the cube selector to “paint” on the model for highlighting, and to push and pull the cube for positioning. Those who were more adept were allowed to try scaling the selector cube, and found it very useful for finer selection work.

Splitting and Deleting Quads was a bit of a problem, both in operation and understanding. For the future, we should change the way the control works. Instead of pressing a button to switch between split and delete modes, there should be a button for each. Hitting the “next” button instead of swapping tools happened on at least one occasion. Understanding what would happen with a quad split, and why it was needed, took a fair bit of coaching. The instructions need revising, and maybe some visual example would help.

A few subjects found the front door in VR, where none did in KVM. The ceiling vent was noted by a couple users, with one identifying it as an attic access hatch. One user thought the rough wall surface might be shelves, another thought it may be a poster.

Most people were being careful, but a couple were sloppy and got poor placement, too much detail removal, or hit the wrong control. At least one was simply trying to go too fast. Most subjects erased much more detail than would an experienced user. One user placed the edges of a to-be-removed quad inside the closet door space, a better strategy than the others who made the quad opening too large by marking quad edges outside the door opening. About half deleted the front door detail. Most removed too much window detail. Most removed items that were in the room such as a lump that may be a vacuum cleaner near the front door, the bookcase and TV, or parts of the wall niche.

We think that nearly all of the poor finished work could be cured with a few practice sessions. Placement could be made perfect every time with the snap-to-surface analyzer.

CHAPTER SIX: DISCUSSION AND FUTURE WORK

So far we have described the REVRRSS architecture, its tools, and detailed the user study that was done. It is now time to compare an experienced user's results to those of the neophyte, discuss how we might make objective measures of the quality of the finished product, cast our gaze into the future, and leave a few notes to those who will take the project forward.

Expert Results

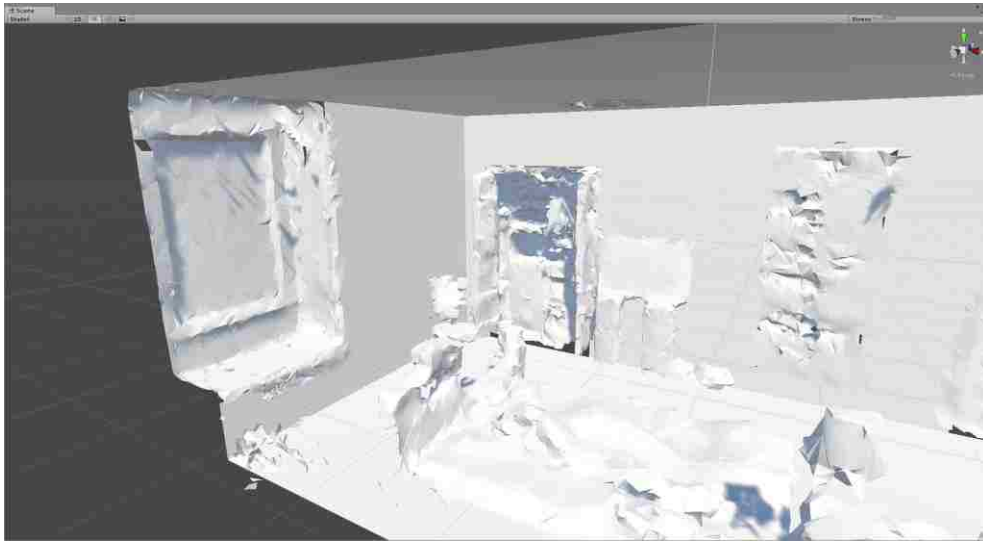


Figure 28 expert results, niche and closet.

Here we can see clearly the new smooth walls, floor, and ceiling after an experienced user has completed the study task. New walls meet the niche area on the left cleanly, and closely match the wall opening for the closet. The possible television atop two speakers or bookcases has been retained. The rough wall area that might be shelves or some wall hanging has been kept in the final product.



Figure 29 expert results, closet, TV, door, window.

From this angle we can see the front door has been kept largely intact, with just a little gap around the difficult floor and tight corner joint. The window frame neatly matches the new walls, too. Above the possible television, notice that a vent or attic access port is neatly fitted to the ceiling.

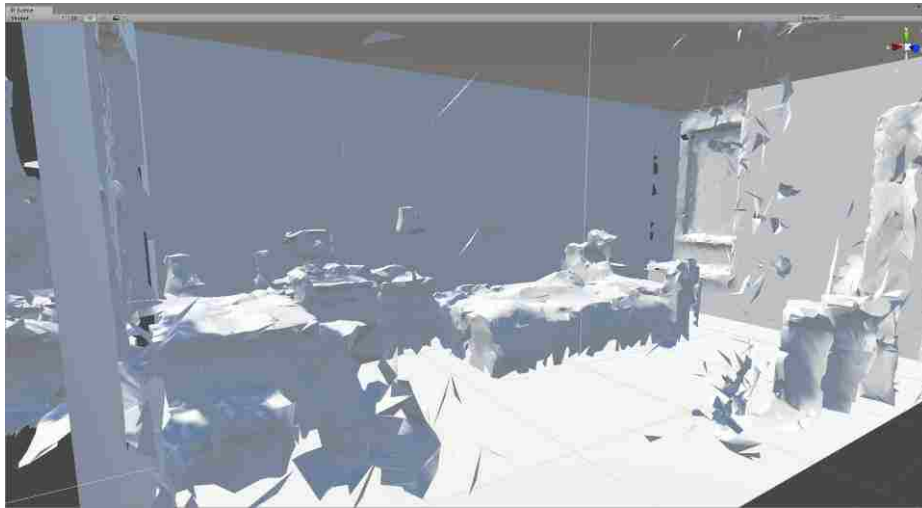


Figure 30 expert results, blank wall and furniture.

From a vantage point near the door, we can see that the bed coverings touch the floor, and that there is just a little gapping along the edge of the niche. The expert user has spent only 80 seconds to find the floor, mark the corners, find the ceiling, and refine the wall placements. The 265 seconds spent adjusting the selection to keep or remove surface was well spent perfecting the final product. Another 195 seconds was spent to split and delete quads. Total, the expert used 539 seconds, or just under nine minutes to finish the VR task.

The average test subject required 13 minutes to complete the same task, and spent nearly 4.5 times longer (80 seconds for expert, 355 seconds for average user) to find the floor, mark the corners, find the ceiling, and refine the wall placements. The average user needed slightly longer than the expert to refine the selection for removal. (265 vs 253 seconds)

Images of the finished product from users show large gaps around important features like the niche, closet, door, and window. Most users removed useful detail like parts of the window, closet door details, and the ceiling hatch. Most deleted front door detail and the TV.

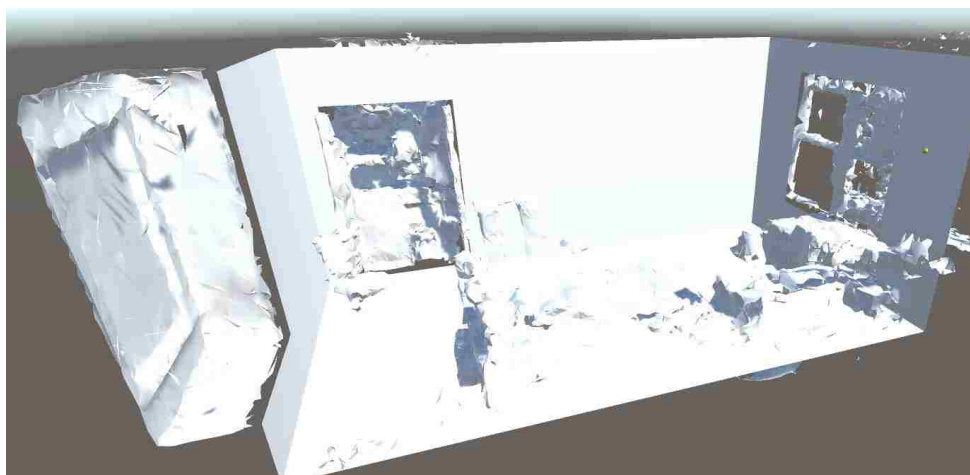


Figure 31 typical user has gaps near details, and removed the TV and door

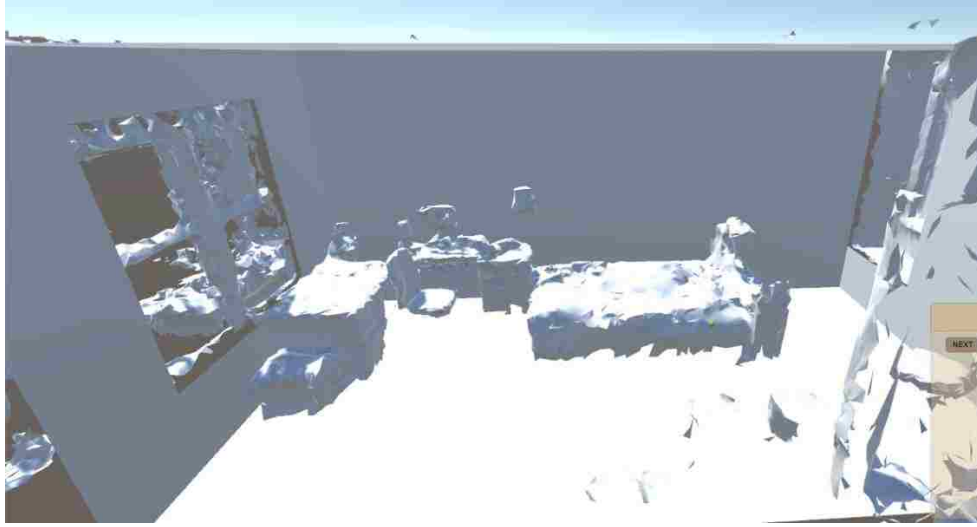


Figure 32 user image from near front door showing big gaps, loss of bed detail

A change to the method that removes triangles could improve these results. Instead of removing every vertex that has been marked, the tool could be made to remove triangles only if two or more of the vertices were marked, and keep all vertices needed for any remaining triangle.

Measuring Successfulness

The visual comparison of the expert's finished work to the typical user is a job for judges at this point, but it has us wondering if there may be some way to objectively measure the quality of the work. This question seems like a big one, a research topic for the future.

Perhaps shape recognizing software could be employed. There may be a way to measure the gap between mesh boundaries and the nearest edge of a quad, assuming that the sub-meshes can be combined so that the boundaries are sensible. We know of no method that may be able to determine what mesh details should remain as part of the room, and which should be removed as noise. This is why we have humans guiding the computer.

REVRSS in the Future

The state of REVRSS right now is similar to the karate student who has earned a black belt. Rather than an achievement of success, the black belt denotes the student who is now ready to begin learning the art. The architecture has begun to settle into a simple modular design, with a degree of anonymity in the way the parts interact. There is a reasonable degree of separation of the parts. There is much uniformity in the way information and commands flow. But, there are also quick cheats put in place to get around some sticky problem. There are some tools that do not yet fulfill their obligation to be uniform and anonymous. The time has come for another round of ferreting out what does not comply, and of better unifying the whole of the system.

Though REVRSS has a set of tools that can get the job done, there are a few that need further development. There are tools that we have imagined, but have not yet begun to develop. Some tools could work more efficiently to overcome slow response times.

Decorating

Painting the mesh is in development, and awaits some method of unrolling the mesh, so that it can be mapped to a flat texture image. Vertex coloring might make this simpler, but does result in lo-fi coloration, with triangles lacking detail. Mesh Colors [17] solves this by also coloring the triangle edges and faces. This is a likely solution to painting in REVRSS. A nice real-world spray can interface is presented in My Virtual Graffiti. [18]

Viewing the mesh as a wireframe can aid in understanding the bones of the scan. The wireframe texture in REVRSS is a modification of an existing shader. A much better approach [19] that can place wireframe lines on a fully painted render is worth a look.

Displaying the mesh as a faceted surface rather than a smoothly curving one reveals details that are otherwise hard to see. This is a useful feature in MeshLab, and one we would like to include. Surely this is just a matter of writing a shader that can be applied in Unity.

Selection and Manipulation

Several choices exist for removing triangles and vertices in MeshLab. One can choose to remove everything that has been touched, or only those areas that are completely enclosed by the selection method. One can also select edges, which REVRSS cannot do.

A contour sketching approach from GemSketch[20] allows the user to outline a shape on a 2D screen, so that the system can select the shape in a 3D environment. Adopting the technique so that a user can define a shape with the hands for selecting a region would be more natural than making multiple selections with simple-shape enclosure selectors.

We found a group selection technique [21] that considers gravity, following the innate expectation that picking up a platter also lifts the objects upon it. This could be readily included in our VR system. Such a technique improves the “actual room” feel of the VR environment.

Manipulating objects in REVRSS is still a bit non-uniform. Objects such as the original scanned room do not allow rotation, scaling, and positioning. This was done to preserve the original room as much as possible. We should allow, at the minimum, rotation to make the room conform to the main axes in Unity, and positioning so that the center of the floor can lie at the origin in the coordinate system. If we allow scaling to correct real-world size, a tape measure tool could be used to get dimensions of objects in the virtual room.

Deformation

The application Kodon, mentioned in the introduction, can deform and create surfaces in real time, and smoothly. This is highly desirable in REVRSS, and is a worthy goal to pursue. Making the mesh updates faster is the route to achieve this. It may be necessary to do the updates outside the Unity framework which is designed for game play, not for mesh editing.

Some deformation techniques were deemed too complex for easy implementation, such as those based on Laplacian smoothing,[22] [23] the Poisson equation, [24] or by filtering schemes on surface normals. [25] These are certain to be targeted for inclusion in future tools. Marking the region to be deformed by these methods is done by cutting planes or the like. A sketching interface [26] adapted for VR might be more natural, and is certainly more flexible.

The hammer tools in the desktop version allow the selector thickness and relative position to be adjusted for user preference. We must find some control technique that brings this feature to the VR version. Perhaps wand buttons can be used to indicate that we wish to change these parameters. If the tools more resembled some real-world device, controls could be placed on the tool for action by the user's secondary hand. This harkens back to our early experiments with a rotary menu selection that mimic radio selector knobs or the settings on a clothes dryer.

We wished to inflate a flat surface to more closely follow a rounded form, and were reminded of Teddy [27] which inflates childish drawings into puffy stuffed animal forms. This would fit in with our desire for simple, real-world tool analogs for making changes to the mesh. Matching the whimsical nature of Teddy, we would use a bicycle pump as a tool analog.

3D scanned rooms are often pierced by a number of holes, where the scanner could not see to place mesh on a surface, and gaps at every boundary between cubical scan sub-meshes. Ability to repair these problems is paramount in a full-fledged tool kit, and a set of slides [28] gives us a quick view of how we might proceed. A more detailed method for hole filling [29] may become a tool in the near future. The common tool metaphor for any of these would be a putty knife and a jar of spackle.

In REVRSS, we should allow mesh deformation of all objects, not just the original scan. If a new object does not quite match expectations, then users should be able to change it in any way possible. Treating every object the same makes them anonymous, as are the tools. If a truncated cone, used to replace a lamp shade, is not the correct slope, then we should have a tool to correct that.

Topology Changes

A method for combining the separate sub-mesh parts that Hololens creates seems a necessary tool. This would allow development of other tools, such as the straight line tool which would move a wavering row of vertices along a connected path into a straight line, enabling the sharpening of corners in the model. The boundary finding tool would become useful.

The flat surface analyzer is capable, but we desire an analyzer that can recognize other surface contours, too. Finding corners in between a wall and floor, picking out the spherical shape of a ball, or the truncated cone for a lampshade are examples of our mythical all-surface analyzer tool. We imagine that this could be an extension of line and curve analyzing for pen-based displays, used in gesture recognition.

Utilities

Future work could provide miniature 3D models to choose from the warehouse. An entire warehouse could be implemented as a world-in-miniature, or even in full size with the ability to quickly move to warehouse departments with the desired models.

The load control is intended to open any stored mesh, adding to or replacing the current one. One good use for the load control is to remove the mesh of a bed, and replace it with a clean bed object which has been created elsewhere. Another use could be to replace a badly scanned chair with a higher-quality scan of the same chair. A save control must be added allowing the user to keep their work. These features would need to be built from scratch or found somewhere, as Unity provides no method for this.

To use models from other sources may require methods to convert them to a format useful to REVRSS. A point cloud from a LIDAR system, for example, is a list of vertices with color information. Our system must either be able to handle that point cloud natively, or convert it to a mesh representation with texture files. Adding a networked connection to a collection of models, such as the SketchUp warehouse, would be a welcome addition to a finished product.

Some work has been done in recognizing objects in 3D meshes. We'd like to borrow that, so that a user can enclose a bed or chair or table, and the recognizer would know it as such. Methods could then be employed to display similar items, either for replacing the existing mesh, or applying texture to change the color of what is already there.

User Interaction

The Vive headset and hand controls seem a great improvement over the desktop experience to our test subjects, and also to the developers. We would like to see much more developing and testing done. There are so many possibilities here that this is a field all on its own. We have only barely used the capabilities of the Vive wand trackpads.

Maneuvering has a number of existing methods that could be included. We chose teleportation largely because it is simple to implement, and easy for newcomers to understand. Hand walking could be developed quickly. Changing the size of the user could allow quicker work in large spaces, and finer work in detailed areas.

Manipulation by grabbing an item is working well, but is a bit awkward due to the design of the grip handles in the Vive controllers. There may be better alternatives to this hardware, or some other method of grabbing with Vive.

The hand controls also suffer from too much sensitivity for some users, and maybe some motion filtering could be employed. A part of this problem is that users often manipulate selectors from a distance by reeling it away from their hand. If the users stepped closer, the motions would be finer. It is also possible to change the tool, so that instead of being attached to a long pointer, it moves by action at a distance, as is the case with the “follow me” vertex mover from early trials. It may be possible to use a degree of isomorphism, where the hand motions are magnified or decreased appropriately. We can see this at work in a mouse, where a quick flick moves the pointer a fair distance, while a slow motion is reduced to a small adjustment.

Notes For Developers

As we prepare to release REVRSS into the wild, we'd like to leave a few notes for future researchers and developers. We would like to leave developers with a system that relieves them of as much tedium as possible, so that adding a tool is a simple matter of meeting design standards and plugging it in. While the current state of the architecture isn't quite up to that standard, we are certain that adding functionality can be done with a little work.

There is a blank control panel, which serves to provide the basic code and visual elements for any control panel you may wish to create. A look at the tool interface files will reveal what methods must be implemented. Having a look at any of the selectors will provide examples for how they work.

User interfaces can follow the general design of those for the Vive and the KVM, which should be largely self-explanatory. Any new user interface need only be capable of sending a small number of actions to the tools through the hub, and of actuating some type of control panel to choose tools and operate features unique to the tool.

The mesh manager has grown large and untidy, but does contain enough commenting to understand the algorithms used for things like the vertex-graph data. No doubt better methods could be used for the data currently gathered. Any information about the mesh that would aid tools and techniques can be easily added to the pool already available. Methods of removing unwanted triangles could be extended to several "flavors" that keep more or less at the edges.

CHAPTER SEVEN: CONCLUSION

The goal of this work was to extend 3D editing from the desktop workstation to a VR environment. Professionals already use 6-degree-of-freedom controllers on the desktop, and large monitors for a better view. Giving professionals and casual users an entire world to work in, and tools that naturally fit the hand, is our underlying driving force.

The purpose of this work covers three main points: That we would like a simple set of tools for editing a 3D mesh that even a novice could use; that we would like to build a system architecture allowing easy inclusion of future tools and that could accommodate any user input device; and that the results of user studies would show that people will prefer working in VR, the results of their work will take no more time than on a desktop, and that the finished product will be comparable to the finished product obtained on a desktop system.

The set of tools currently available in REVRSS allows selection and manipulation of the mesh surface. Tools for deforming, deleting portions of the mesh, or copying a portion of it, exist. Tools to paint or texture the mesh surface are in development. Items such as simple shapes or 3D mesh models obtained elsewhere can be added to the room environment. Simple position, scale, and rotation of items and tools is possible. A set of tools allowing the main flat surfaces of the walls, floor and ceiling to be replaced by simple flat planes reduces vertex count dramatically, making the model simpler and smaller without reducing desired detail. While a reasonably good set of tools exists, there is always room for more, and our system makes it possible to add to the tool set without great difficulty.

The system architecture is capable of taking commands from any user input device and routing it to any tool, without regard for what the actual user action is, and without regard for what the tool accomplishes. The mesh management section treats any number of scanned segments as one big mesh, and maintains information about the mesh which is useful to the tools and to the user. The system has grown and changed, and could benefit from another refresh to more tightly integrate all the parts of it.

User testing has shown a strong preference for VR over the desktop KVM interface. Measurement of placement of walls, floor, and ceiling shows that the VR accuracy was near the same as placement made using KVM. Modifications to the tools could reduce any difference to zero. The time required to complete tasks in VR was essentially the same as the time required to accomplish those same tasks on the desktop in KVM.

With these several expectations met, it is possible to say that we have achieved our purpose. The future may see us meet our goals. There is room for improvement, and we hope that a new wave of researchers “come play in our sandbox” and make our work their own.

**APPENDIX:
IRB APPROVAL LETTER**



University of Central Florida Institutional Review Board
Office of Research & Commercialization
12201 Research Parkway, Suite 501
Orlando, Florida 32826-3246
Telephone: 407-823-2001 or 407-882-2276
www.research.ucf.edu/compliance/irb.html

Approval of Human Research

From: UCF Institutional Review Board #1
FWA00000351, IRB00001138

To: Charles Greenwood

Date: October 23, 2018

Dear Researcher:

On 10/23/2018 the IRB approved the following human participant research until 10/22/2019 inclusive:

Type of Review: UCF Initial Review Submission Form
Expedited Review
Project Title: Realtime Editing Virtual Reality Room Scale Scans
Investigator: Charles Greenwood
IRB Number: SBE-18-14462
Funding Agency:
Grant Title:
Research ID: N/A

The scientific merit of the research was considered during the IRB review. The Continuing Review Application must be submitted 30 days prior to the expiration date for studies that were previously expedited, and 60 days prior to the expiration date for research that was previously reviewed at a convened meeting. Do not make changes to the study (i.e., protocol, methodology, consent form, personnel, site, etc.) before obtaining IRB approval. A Modification Form **cannot** be used to extend the approval period of a study. All forms may be completed and submitted online at <http://iris.research.ucf.edu>.

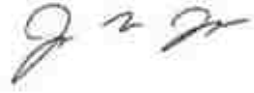
If continuing review approval is not granted before the expiration date of 10/22/2019, approval of this research expires on that date. When you have completed your research, please submit a Study Closure request in IRIS so that IRB records will be accurate.

Use of the approved, stamped consent document(s) is required. The new form supersedes all previous versions, which are now invalid for further use. Only approved investigators (or other approved key study personnel) may solicit consent for research participation. Participants or their representatives must receive a copy of the consent form(s).

All data, including signed consent forms if applicable, must be retained and secured per protocol for a minimum of five years (six if HIPAA applies) past the completion of this research. Any links to the identification of participants should be maintained and secured per protocol. Additional requirements may be imposed by your funding agency, your department, or other entities. Access to data is limited to authorized individuals listed as key study personnel.

In the conduct of this research, you are responsible to follow the requirements of the [Investigator Manual](#).

This letter is signed by:



Signature applied by Racine Jacques on 10/23/2018 04:00:51 PM EDT

Designated Reviewer

Page 2 of 2

LIST OF REFERENCES

- [1] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, G. Ranzuglia MeshLab: an Open-Source Mesh Processing Tool Sixth Eurographics Italian Chapter Conference, page 129-136, 2008.
- [2] D. Bowman, E. Kruijff, J. LaViola, I. Popyrev 3D User Interfaces Theory and Practice. Addison Wesley Pearson Education 2005
- [3] C. Gotsman, "What's in a Mesh? A Survey of 3D Mesh Representation Schemes," International Conference on Shape Modeling and Applications 2005 (SMI' 05), Cambridge, MA, 2005, pp. 2-2. doi: 10.1109/SMI.2005.51
- [4] Jeffrey A. Cashion ; Chadwick Wingrave ; Joseph J. LaViola Automatic 3D selection technique assignment using real-time scenario analysis 2013 IEEE Virtual Reality (VR) Year: 2013 Pages: 103 - 104
- [5] Tobias Rick ; Anette von Kapri ; Torsten Kuhlen GPU implementation of 3D object selection by conic volume techniques in virtual environments 2010 IEEE Virtual Reality Conference (VR) Year: 2010 Pages: 243 - 246
- [6] Daniel Mendes ; Daniel Medeiros ; Eduardo Cordeiro ; Maurício Sousa ; Alfredo Ferreira ; Joaquim Jorge PRECIOUS! Out-of-reach selection using iterative refinement in VR 2017 IEEE Symposium on 3D User Interfaces (3DUI) Year: 2017
- [7] C. Sha, B. Liu, Z. g. Ma and H. b. Zhang, "Multi-resolution Meshes Deformation Based on Pyramid Coordinates," Computer Graphics, Imaging and Visualisation, 2007. CGIV '07, Bangkok, 2007, pp. 200-204. doi: 10.1109/CGIV.2007.59
- [8] F. G. M. Silva and A. J. P. Gomes, "Interactive editing of multiresolution meshes," Computer Graphics and Image Processing, 2004. Proceedings. 17th Brazilian Symposium on, 2004, pp. 202-209. doi: 10.1109/SIBGRA.2004.1352962
- [9] M. Attene and B. Falcidieno, "ReMESH: An Interactive Environment to Edit and Repair Triangle Meshes," IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06), Matsushima, 2006, pp. 41-41.
- [10] M. Yirci and I. Ulusoy, "A comparative study on polygonal mesh simplification algorithms," 2009 IEEE 17th Signal Processing and Communications Applications Conference, Antalya, 2009, pp. 736-739. doi: 10.1109/SIU.2009.5136501
- [11] K. Kahler, C. Ross, R. Schneider, J. Vorsatz and H. P. Seidel, "Efficient processing of large 3D meshes," Shape Modeling and Applications, SMI 2001 International Conference on., Genova, 2001, pp. 228-237. doi: 10.1109/SMA.2001.923394

- [12] I. Ivriissimtzis ; C. Rossl ; H.-P. Seidel A divide and conquer algorithm for triangle mesh connectivity encoding 10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings. Year: 2002 Pages: 294 - 303
- [13] David J. Zielinski ; Derek Nankivil ; Regis Kopper 6 Degrees-of-freedom manipulation with a transparent, tangible object in world-fixed virtual reality displays 2017 IEEE Virtual Reality (VR) Year: 2017 Pages: 221 - 222
- [14] A. D. Gregory, S. A. Ehmann and M. C. Lin, "inTouch: interactive multiresolution modeling and 3D painting with a haptic interface," Virtual Reality, 2000. Proceedings. IEEE, New Brunswick, NJ, 2000, pp. 45-52.doi: 10.1109/VR.2000.840362
- [15] Adam Faeth ; Michael Oren ; Jonathan Sheller ; Sean Godinez ; Chris Harding. Cutting, Deforming and Painting of 3D meshes in a Two Handed Viso-haptic VR System 2008 IEEE Virtual Reality Conference Year: 2008 Pages: 213 - 216
- [16] I. Ha, Y. B. Lee and J. D. K. Kim, "3D editing system for captured real scenes," Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific, Hollywood, CA, 2012, pp. 1-5.
- [17] Cem Yuksel, John Keyser, Donald H. House Mesh colors April 2010 ACM Transactions on Graphics (TOG): Volume 29 Issue 2, March 2010 Publisher: ACM
- [18] Mei Yii Lim ; R. Aylett MY virtual graffiti system 2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763) Year: 2004 , Volume: 2 Pages: 847 - 850 Vol.2
- [19] Celes, Waldemar & Abraham, Frederico. (2010). Texture-Based Wireframe Rendering. 149 - 155. 10.1109/SIBGRAPI.2010.28.
- [20] Maghoumi, M., LaViola, J., Desingh, K., and Jenkins, O. "GemSketch: Interactive Image-Guided Geometry Extraction from Point Clouds", 2018 International Conference on Robotics and Automation (ICRA), May 2018.
- [21] Ji-Young Oh ; W. Stuerzlinger ; D. Dadgari Group Selection Techniques for Efficient 3D Modeling 3D User Interfaces (3DUI'06) Year: 2006 Pages: 95 - 102
- [22] O. K. C. Au, C. L. Tai, L. Liu and H. Fu, "Dual Laplacian editing for meshes," in IEEE Transactions on Visualization and Computer Graphics, vol. 12, no. 3, pp. 386-395, May-June 2006.doi: 10.1109/TVCG.2006.47
- [23] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, H.-P. Seidel Laplacian surface editing July 2004 SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing

- [24] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, Heung-Yeung Shum Mesh editing with poisson-based gradient field manipulation August 2004 ACM Transactions on Graphics (TOG): Volume 23 Issue 3, August 2004
- [25] H. Yagou, Y. Ohtake and A. Belyaev, "Mesh smoothing via mean and median filtering applied to face normals," Geometric Modeling and Processing, 2002. Proceedings, 2002, pp. 124-131.doi: 10.1109/GMAP.2002.1027503
- [26] Andrew Nealen, Olga Sorkine, Marc Alexa, Daniel Cohen-Or A sketch-based interface for detail-preserving mesh editing August 2007 SIGGRAPH '07: ACM SIGGRAPH 2007 courses Publisher: ACM
- [27] Takeo Igarashi, Satoshi Matsuoka, Hidehiko Tanaka Teddy: a sketching interface for 3D freeform design August 2007 SIGGRAPH '07: ACM SIGGRAPH 2007 courses Publisher: ACM
- [28] marcel campen, marco attene , leif kobbelt Practical Guide to polygon mesh repairing RWTH Aschen University Germany IMATI-GE, CNR, Italy Eurographics 2012 Tutorial
- [29] Xiaohe Li ; Lan Kang ; Hu Xiao Hole filling method of triangle mesh 2011 International Conference on Multimedia Technology Year: 2011 Pages: 3085 - 3089