University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Master's thesis

Testing environment for user studies of distorted triangle meshes in virtual reality

Plzeň 2017 Petr Šroub

Místo této strany bude zadání práce.

Declaration

I hereby declare that this master's thesis is completely my own work and that I used only the cited sources.

Plzeň, 16th May 2017

Petr Šroub

Acknowledgments

I would like to thank Doc. Ing. Libor Váša Ph.D. for an interesting and fun project to work on as well as his efforts, suggestions, enthusiasm and most importantly patience during the protracted development. I would also like to thank Palmer Luckey, John Carmack and the hundreds of other individuals responsible for bringing affordable virtual reality back from the realm of science fiction and to the forefront of public interest.

Abstract

The subject of this work is software for conducting subjective experiments in virtual reality on the subject of perception-oriented triangle mesh distortion metrics. The work entails the creation of this software and iterative improvement of its design in order to increase its usability. In this document, a background necessary for understanding the problem domain is provided, including the topics of virtual reality and perception-oriented metrics. The project goals and requirements are supplied. A detailed overview of the software's final implementation as well as several development iterations is provided. A description of the pilot study that was conducted to improve on the user interface along with the alternations to the software based on its results is given. Finally a conclusion about the final software is reached and possible future work is discussed.

Abstrakt

Tématem této práce je program pro provádění subjektivních experimentů ve virtuální realitě na téma percepčně-orientovaných metrik deformací trojúhelníkových sítí. Práce obsahuje vytvoření tohoto programu a iterativní vylepšování jeho designu pro zvýšení jeho použitelnosti. V tomto dokumentu je dodáno pozadí nutné pro pochopení problémové domény, zahrnující virtuální realitu a percepčně-orientované metriky. Dále jsou předloženy cíle a požadavky projektu. Je popsán detailní přehled o finální verzi programu a několika iteracích vývoje. Následuje popis provedené pilotní studie zaměřené na vylepšení uživatelského rozhraní a z jejích výsledků provedených změn. Nakonec je o programu podán závěr a je diskutováno o možné budoucí práci.

Contents

1 Introduction 9

2 Background 11
2.1 Distortion sources 11
2.1.1 Mesh simplification 11
2.1.2 Mesh compression 12
2.2 Human perception 13
2.3 Perception-oriented metrics 14
2.3.1 Static mesh metrics 15
2.3.2 Dynamic mesh metrics 17
2.4 Dihedral angle mesh error metric 18
2.5 Subjective test validation 18
2.5.1 Subjective tests 18
2.5.2 DAME subjective test 19
2.6 MeshTest 21
2.6.1 MeshTest usage 21
2.7 Virtual reality 23
2.7.1 History 23
2.7.2 Current state 23
2.7.3 Advantages 23
2.7.4 Issues 24
2.7.5 Health effects 25
2.7.6 Uses 26

3 Project goal 28

4 Project design requirements 29
4.1 Binary forced choice 29
4.2 Anchored scoring 30

5 Application use cases 31
5.1 Typical usage (experimenter) 31
5.2 Typical usage (test subject) 31
5.3 Conclusion 31
5.4 Result logging 32

6 Application design **33**
6.1 Hardware platform . . . 33
6.1.1 Technical specifications . . . 34
6.2 Engine . . . 35
6.2.1 Unity . . . 36
6.3 File storage . . . 37
6.4 Interactivity . . . 38
6.5 Experiment file format . . . 39
6.6 Log file format . . . 40
6.7 Recording file format . . . 40

7 Implementation details **41**
7.1 Singletons . . . 41
7.2 GlobalStateHolder . . . 42
7.3 User input system iterations . . . 42
7.3.1 Keyboard/mouse button iteration . . . 42
7.3.2 Gaze input iteration . . . 42
7.3.3 Worldspace cursor iteration . . . 43
7.3.4 Position tracked controllers iteration . . . 43
7.3.5 Final solution . . . 43
7.4 Model data . . . 43
7.4.1 wavefront .obj format . . . 44
7.4.2 application-specific .dat format . . . 44
7.4.3 Model loaders . . . 45
7.4.4 Postprocessing . . . 45
7.5 Screen system . . . 47
7.6 Individual screen implementations . . . 47
7.6.1 BaseMCOScreen . . . 47
7.6.2 MCOCommandlineScreen . . . 48
7.6.3 MCOCustomScreen . . . 49
7.6.4 MCODummyStartScreen . . . 49
7.6.5 MCOModelDisplayScreen . . . 49
7.6.6 MCOPersonalInfoScreen . . . 49
7.6.7 MCORecordingPlaybackScreen . . . 49
7.6.8 MCOTutorialScreen . . . 49
7.6.9 MCOCustomScreenEditorScreen . . . 50
7.6.10 BaseExperimentScreen . . . 50
7.6.11 MCOComparingExperimentScreen . . . 50
7.6.12 MCORatingExperimentScreen . . . 51
7.7 Worldspace UI implementation . . . 51

7.7.1 Dashboards 51
7.7.2 Model manipulation panel 53
7.7.3 Configuration UI 53
7.8 Model display objects 53
7.9 Experiment file format 54
7.9.1 Model file paths 54
7.9.2 Experiment log 55
7.9.3 User movement recording 56
7.9.4 Result e-mail 56
7.10 Abandoned features 57
7.10.1 Editor 57
7.10.2 Prop system 57

8 Pilot study 59
8.1 Study conduct 59
8.2 Form 59
8.3 Results 60
8.3.1 Changes enacted 60

9 Application usage 62
9.1 Launch options 62
9.2 Typical experiment example 62
9.2.1 Tutorial 62
9.2.2 Personal data collection 63
9.2.3 Additional explanations 63
9.2.4 Experiment 63
9.2.5 Finalization 63

10 Conclusion 65
10.1 Future work 65

1 Introduction

In computer graphics, 3D objects are defined using various means, such as mathematical object definitions, voxel arrays or point clouds. The most common method of 3D object representation are polygonal meshes (in most cases specifically triangle meshes), which offer significant advantages over the alternatives.

When manipulated these meshes are often unavoidably altered. However, in many cases these alternations are undesirable, as it is preferable or even necessary to preserve the original shape of the object.

In order to design and optimize algorithms that manipulate models it is necessary to measure the amount of introduced distortion. Depending on the use case, it may be necessary to preserve the model shape in the absolute sense (for example, in the case of CAD models it is critical not to distort the object past specified tolerances) or it may be desirable to preserve the models perceptible visual quality (such as when the models are meant to be viewed by humans). These two considerations are fundamentally different.

As part of their work on their work on human-perception oriented mesh simplification algorithms, Váša and Rus have noticed that for most sources of distortion (other than those commonly used as part of validation experiments, such as Gaussian noise or sine waves, which poorly model distortions introduced by mesh simplification or compression algorithms) there is only a weak correlation between the commonly used metrics of mesh distortion and the actual human observer-determined degradation of quality. As part of their paper on the proposal of a more perception-oriented metric (Váša and Rus 2012), the Dihedral angle mesh error (DAME) metric, they performed a validation subjective experiment.

This validation experiment was conducted using a purpose-built software named MeshTest, which presented test subject with model triplets (one model of which was designated as the original) and asked them to decide which of the two model copies was more similar to the original. Not only was this application initially intended for this singular purpose, developments since then have resulted in the base this software was built on (Microsoft XNA) becoming unsupported.

More recent developments have resulted in the resurfacing of virtual reality devices on the customer market for the first time in several decades. Thanks to advances in technology these devices are now of sufficient quality to provide for a viable alternative to traditional display methods, which

means that they represent an avenue worth exploring for perception-based metrics such as DAME. In order to determine the accuracy of DAME when models are viewed using a virtual reality device, a new subjective experiment must be conducted.

In order for this experiment to take place a new corresponding experiment software is required, since MeshTest is incapable of rendering to a virtual reality headset. The creation of that software is the subject of this work.

2 Background

In computer graphics objects are predominantly represented using triangle meshes: collections of vertices and their connectivity data. Since the speed and difficulty of mesh rendering is heavily related to its triangle count, there is usually a soft upper limit to the triangle count of any mesh intended for a given application, and by extension its maximum detail level. In order to obtain the highest possible quality level achievable within this limit, many techniques exist to simplify meshes. All of these affect the triangle mesh by distorting it, and therefore possibly negatively affect the perceptible visual quality of the mesh. Even for applications where this is not required, other processes may be applied with similar results. For example, mesh compression algorithms may be employed in order to reduce the file size. While in some cases this kind of distortion is a non-issue, for example when the data is intended for industrial use (such as CAD models or 3D), in other cases the model will be viewed by human observers (such as 3D games or movies) and as such it is desirable to minimize or ideally eliminate the perceived distortion.

2.1 Distortion sources

Algorithms and processes that distort triangular meshes are a pivotal part of computer graphics and cannot be avoided in all but the simplest use cases. While there are many such algorithms (such as mesh watermarking), two specific kinds stand out as highly explored regions of research.

2.1.1 Mesh simplification

Mesh simplification algorithms are used to reduce the triangle count of a model in a way that does not affect vertex positions. Usually this involves merging and replacing nearby triangles with consideration towards minimizing the error caused by the change, though numerous alternate approaches exist (Cignoni, Montani and Scopigno 1998), (D. P. Luebke 2001). Some of the reasons for using such an algorithm are:

Rendering complexity reduction Difficulty and speed of triangular mesh rendering is amongst other things (such as texture size, lightning model used, post processing effects) tightly correlated with triangle counts of

the meshes rendered. Simply put, the more triangles there are, the more times the renderer has to perform operations related to processing them. Therefore, reducing the triangle count in a realtime rendering situation can lead to increased framerates or the possibility of increasing the count of unique objects in a scene. Even when the rendering scenario is not realtime, such as in the case of pre-rendered videos, reducing the scene complexity results in less computing time and resources required and is thus desirable.

Level of detail Level of detail (LOD) implementations take advantage of the fact that distant objects often only occupy a small amount of screen space. In the case of these objects, the detail required when the object is close up becomes imperceptible as the objects draws further away from the camera, and therefore becomes a waste of rendering resources. In these cases, LOD systems replace objects with lower-quality versions. These LOD version can either be created by hand or generated on the fly. However, in most cases LOD variations are pre-generated and saved along with the full detail version.

Aesthetic choice In some cases, an intentional low-fidelity aesthetic is desirable (such as in the case of retro videogames or movies, or 3D presentations). While in most cases the better approach is to create the models used with such an aesthetic in mind from the start, sometimes there are situations where usage of preexisting models is necessary (such as legacy characters or real world object scans). While it is unrealistic to expect an algorithm to produce an aesthetically better low-polygon version of existing models than one produced by a talented artist, in some cases relying on such an approach can be more cost effective or even necessary (with scenes containing a large amount of unique objects).

2.1.2 Mesh compression

Mesh compression algorithms focus on reducing mesh file size through many different approaches (Peng, C.-S. Kim and Kuo 2005), from more space-efficient connectivity encoding to quantization of vertex data. Depending on the approach, this compression can be lossless or lossy. In the case of lossy compression, care must be taken not to degrade the mesh past the point of perceptability, or in the case of industrial applications, the point of error tolerance. Mesh compression algorithms are also sometimes paired in sequence with mesh simplification algorithms, as a reduction in triangle

count also leads to a reduction in data to compress. Some reasons for mesh compression algorithm usage include:

Limited storage space Applications that work with model data are often limited in the amount of storage space available to them (for example, a videogame is limited in storage space to the amount of data that can be stored on the disc it is stored on). Even when the actual limit is larger than the storage space required by the application, reducing its storage space footprint is often desirable.

File transfer speed In a similar vein, smaller files are easier to transfer in reasonable time over the internet. In addition, some mesh compression algorithms focus specifically on this usage and encode the data in such a way that a partially downloaded mesh data file can still be used, albeit with a corresponding decrease in detail quality.

GPU memory footprint While in use, meshes also use up memory on the graphical processing unit (GPU) used for rendering, which is typically far more limited than the primary storage medium of a computer. When this memory is full, the GPU must start switching data between its local memory and the computer's, which can lead to a drastic reduction in performance. Mesh compression algorithms that aim to alleviate this problem are geared towards producing encoded meshes that are either quickly decodeable or usable in GPU memory directly.

2.2 Human perception

In order to effectively and objectively measure the perceptible distortion of a mesh, an understanding of human perception is required. While there is a certain degree of subjectivity to the result, there are thresholds in difference of quality that most observers will agree upon. In order to model human perception and cognition, the human visual system (HVS) is described. When a thorough enough understanding of the HVS is reached, it can be artificially modeled and thus human perception data can be estimated without actual human involvement. This model can then also be used advantageously to mask imperfections or highlight specific details. The HVS perceives stimuli depending on many factors, including color, contrast, orientation, spatial distribution, etc. Some properties of the HVS are briefly explored below:

- There is a significant amount of existing work on the subject of the contrast sensitivity function (CSF) detection threshold, which determines

at which point an observer will fail to notice imperceptible information (such as a monochrome gradient becoming uniform gray given enough distance), and the creation of metrics for 2D image quality based on it. There are several ways of defining the CSF. Some examples include the CSF defined in the 1970 (Mannos and Sakrison 1974) for the development of the first image quality metric for encoded monochrome images. Another is a three parameter exponential function (Movshon and Kiorpes 1988). A model accounting for orientation, luminance, image size, eccentricity, and viewing distance was proposed by Daly in his Visual Difference Predictor (Daly 1992). Other, more complex models also exist.

- Some related works (Larabi, Brodbeck and Fernandez 2006) instead focus on determining the discrimination threshold, at which an observer can no longer distinguish between two similar stimuli. This is a more applicable metric when considering quality assessment tasks which rely on the observer comparing the examined visual stimulus to some reference (either another visual stimulus or an implicit reference given a lack of one). Little work has been dedicated to color-based CSFs. This is primarily thanks to the HVS being less sensitive to changes with its chromatic mechanisms as opposed to luminance ones. A suggested explanation (Fairchild 2013) is that this is caused by the lack of edge detection and enhancement in the color dimension.

- Visual masking defines the reduction of visibility of one visual stimulus due to the presence of another. This effect is strongest when the two stimuli share orientation, location and frequency (Pappas, Safranek and J. Chen 2000). It is widely used as a tool in many vision-related fields, such as image or video compression, watermarking, computer generated graphics, etc.

There are many other visual systems that are too numerous to elaborate upon in detail, such as visual saliency, luminance adaptation, binocular compensation, etc.

2.3 Perception-oriented metrics

Generally-used metrics for evaluating the effectiveness of mesh compression or simplification algorithms (for example the Mean-Squared Error (MSE) or Hausdorff distance) do have some advantages, such as ease of computation or predictability of results, but ultimately they can be proven (Váša and

Rus 2012) to be only weakly correlated to actual visual quality as judged by a human observer. Perception-oriented metrics attempt to rectify this issue, usually by incorporating some approximation of human perception as part of their definition.

Mesh distortion perception-oriented metrics can be divided into two main categories. First are the metrics focused on static 3D meshes, used mainly for measuring distortion introduced by mesh compression, watermarking or as a guiding data source for mesh simplification and perceptually based rendering. Second are metrics focused on dynamic 3D meshes which are focused on artefacts either introduced by or made visible by animating the mesh over time.

2.3.1 Static mesh metrics

Static mesh metrics can be further split into two categories: view-dependent and view-independent.

View-dependent metrics

View-dependent metrics rely on applying perceptual properties and principles of the HVS in image space to a still image (in the case of 3D models, rendered images of a 3D mesh). This approach cannot be described as fully reliable, as it has been demonstrated (Rogowitz and Rushmeier 2001) that the results of visual perception of a set of images of a 3D mesh are not equivalent to the results of actually observing that 3D mesh in a more robust manner (such as a 3D model viewer application). Some of the more pivotal view-dependent metrics are presented below.

CSF Used as part of one of the first perceptually driven rendering solutions (Reddy 1997), this metric analyzed the frequency content in several prerendered images to determine the best LOD to use in a realtime rendering system. It was later used as a basis for a simplified CSF metric (D. Luebke and Hallen 2001) which also accounted for silhouette changes and considered worst-case contrast and frequency results before determining whenever the results were imperceptible, which was itself later extended (Williams et al. 2003) to integrate texture and lightning effects into the calculation.

VDP Proposed (Ferwerda, Shirley et al. 1997) as an extension of the earlier Daly VDP image metric (Daly 1992), this metric was the first perception model that paid particular attention to visual masking. This

work was later used (Myszkowski 1998) as a basis of an algorithm that would produce a simplified mesh that would result in a visually identical rendered image. Similarly, another work proposed (Ramasubramanian, Pattanaik and Greenberg 1999) a VDP-based approach that would reduce the rendering power necessary for a global illumination rendering model. Another interesting application is a proposed (Dumont, Pellacini and Ferwerda 2003) real time rendering system capable of on-the-fly optimization of performance based on image quality and framerate.

Visual discrimination model (VDM) A simplified version of an earlier visual differences metric (Lubin 1995), this metric was used (Bolin and Meyer 1998) as a perceptual model for optimizing the sampling for ray-tracing algorithms. modified in order to account for chromatic aberration effects in color image renders. Later work (Qu and Meyer 2008) would expand upon this metric by taking into consideration the visual masking properties of texture maps and using them to allow simplification and remeshing of of textured meshes.

Visual equivalence A more recent work, this approach proposes the concept of visual equivalence: images being considered visual equivalent if they offer the same scene appearance impression, which is defined by visual perceptions of geometry, illumination, materials and other properties of the scene and affected by changes in environment lighting. Other works (Ferwerda, Ramanarayanan et al. 2008) written around the same time also deal with this concept. The concept of visual equivalence has also been used to investigate rendering artefacts seen in global illumination-based approximations of minerals and shapes (Křivánek, Ferwerda and Bala 2010).

View-independent metrics

In contrast to view-dependent metrics, view-independent (model based) metrics seek to analyze the geometry and other properties of a 3D model directly and from there predict any visual artefacting that may arise, with the purpose being elimination of the bias towards detecting only those artefacts that show up on still image renders (and may not even be apparent in motion), rather than artefacts revealed when the model is manipulated in real time. Some more relevant examples of this sort of metric are presented below.

Geometric laplacian First defined in order to evaluate the efficiency of a mesh compression algorithm (Karni and Gotsman 2000), this metric

measures the Geometric Laplacian, which relates to the visual smoothness of each vertex and from it derives a visual metric to compare two 3D objects. Even with a later proposal (Sorkine, Cohen-Or and Toledo 2003) of a modification that would use different parameter values, this metric correlates poorly with actual human perception.

Curvature-based Various works have explored the idea of basing a metric around the idea that human perception is sensitive to changes in curvature. The first metric based on this approach was the Discrete Differential Error Metric (S.-J. Kim, S.-K. Kim and C.-H. Kim 2002). Later, a similar metric (Howlett, Hamill and O'Sullivan 2004) that further emphasized visually salient features using an eye tracking was suggested. A similar approach that had the advantage of automatically extracting the visually salient areas using computed multi-resolution curvature maps surfaced not long after (C. H. Lee, Varshney and Jacobs 2005). Of high importance to this work is the Mesh Structural Distortion Measure metric (Lavoue, Denis and Dupont 2007), proposed for evaluation of mesh watermarking algorithms. This metric, based on an the concept of structural similarity first introduced (Wang et al. 2004) for the purposes of 2d image quality evaluation, computes differences of statistics local to corresponding windows on the compared meshes. A global difference is then defined as a Minkowski sum of the distances of the windows, with one window being considered per vertex. A later multi-resolution based improvement (MSDM2) (Lavoué 2011) provides better performance and allows comparison of meshes with differing connectivity.

Strain energy This metric (Bian, Hu and Martin 2009) is based on strain energy (energy which causes the deformation between the two compared meshes). The larger is the strain energy, the bigger is the perceivable distortion and by extension the larger are the odds that a human observer will notice the distortion. This metric is only suitable for relatively small distortions.

2.3.2 Dynamic mesh metrics

Unlike with static meshes, dynamic meshes enforce the consideration of the effect of perception shifting with time. A not insignificant number of distortion artefacts can be concealed by the still nature of a static mesh and brought to the forefront by animating it. Other distortions are only apparent with the passage of time, and are referred to as temporal artefacts.

An example of such a distortion would be the addition of a small amount of noise each frame: on a static mesh, this would only result in a slightly bumpy surface that is mostly unnoticeable, whereas on an animated mesh it will result in a clearly visible bubbling or rolling effect.

While static mesh metrics can be used on individual frames of the animation analyzed and their results then averaged, this approach inherits all the issues of static mesh metrics while failing to account for the peculiarities of dynamic mesh analysis. It is for this reason that metrics specifically geared towards dynamic meshes tend to lead to more accurate results.

2.4 Dihedral angle mesh error metric

The paper that this work is an extension upon (Váša and Rus 2012) describes a novel objective metric that can achieve higher correlation between its results and results of a subjective test experiment (Pearson > 90) and is about two orders of magnitude faster than the next best approach of the time, MSDM2 (Lavoué 2011).

This metric is based on measuring the distortion of dihedral angles, which is closely linked to quadratic bending energy functionals and is invariant with translations and rotations. It explicitly models the model masking effect by taking the original dihedral angle into account and assigning higher influence to areas which were originally close to flat.

2.5 Subjective test validation

In order to validate this metric, actual human observation data was also required. This data was collected using a subjective test.

2.5.1 Subjective tests

Subjective tests related to human perception, even regardless of the type of data being tested (3D models, image data, video, etc.) should all follow the same basic format:

- A database of objects to perform the experiment with is constructed, differentiating between reference objects and their distorted versions.

- The subjective experiment is conducted, with human observers giving their opinion on perceived distortions of the objects in the database.

- A mean opinion score (MOS) is then computed for each distorted object in the database. Afterwards it is normalized in order to equate the different rating scales used by the different test subjects. Finally, a filtering step is applied to eliminate potential outliers.

- A correlation is computed between the MOSs of the objects and the values computed using the tested metric. Usually, two correlation coefficients are considered: the Spearman Rank Order Correlation and the Pearson Linear Correlation Coefficient. The Pearson coefficient is calculated over a non-linear regression on the values calculated by the metric in order to improve the matching between the metric and the subjective opinion values.

Some other considerations must also be made during the test design, as the quality of results can be heavily dependent on several secondary factors (Ebrahimi 2009):

- The testing environment, such as the computer monitors used to display the objects, the viewing distance, and the room lighting conditions.

- The test object themselves should contain multiple variations with diametrically opposite qualities in order to generalize the test results. Selecting for specific kinds of test objects or scenarios should be avoided, and special care should be given to include worst-case scenarios.

- The methodology must be decided upon. For example, how the objects will be presented (on their own or with a reference version) or how they will be rated, such as on a numeric scale or using a preselected set of adjectives.

- The data analysis technique also plays a significant role.

2.5.2 DAME subjective test

The validation experiment for the DAME metric was conducted on a collection of five models:

Bunny The Stanford bunny, a widely used mesh built from a scan of a real object. Its shape is complex and well known to a human observer.

Jessi An artificially created model of a female human head.

James An artificially created model of a male human figure.

Nissan An artificially created model of a car's industrial design.

Helix An artificial, computer generated model of an abstract shape.

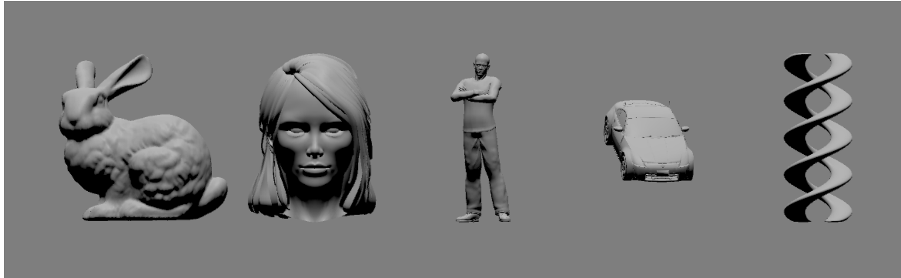

Figure 2.1: Models used in the experiment

The models were then distorted using several distortion processes or compression algorithms:

- Random noise in each axis
- Random noise with uniform distribution in each axis
- Sine signal in the X axis
- Random noise with Gaussian distribution applied to each axis
- Uniform quantization of coordinates
- quantization of coordinates scaled by the model's bounding box
- Random affine transformation
- Smoothing
- Angle preserving mesh noise
- Normal preserving mesh noise
- Result of the angle analyzer mesh compression algorithm (H. Lee, Alliez and Desbrun 2002)
- Result of the spectral mesh compression algorithm (Karni and Gotsman 2000)
- Result of the high-pass encoding mesh compression algorithm (Sorkine, Cohen-Or and Toledo 2003)

The experiment was conducted using a known-original forced-choice task with a hidden anchor. The users were presented with triplets of models. One was the original and designated as such. It was accompanied by two distorted versions, displayed at once on a single monitor to eliminate the effect of differing viewing conditions. The subjects were then tasked with selecting which of the two distorted versions was a better match to the original. The distorted versions were never identical, but could contain the original itself as a hidden anchor, in order to evaluate the level of distortion imperceptible by human observation.

While ideally this experiment would be conducted on every pair possible from the generated combinations, in practice this proved to be far too many comparisons to make without the subject losing concentration on the task. The final experiment was conducted using 40 comparison pairs per user.

2.6 MeshTest

The experiment was conducted using the purpose-built software MeshTest. MestTest was implemented using Microsoft XNA and the C# programming language. The software was iterated upon during development in order to maximize usability and ease of use, reaching final version after eight iterations. A pilot study was also conducted in a usability lab at the Czech Technical University in Prague.

It allowed for simple model manipulation using the keyboard and mouse, allowing for translational and rotational movement as well as model scaling, all of which was shared between all models to maintain identical conditions. The models were displayed in a pyramid formation with the original model on top. It used the default lighting and shading setup, with specular and ambient illumination disabled and models being rendered with anti-aliasing enabled.

While the MeshTest software was perfectly functional at its time of use, developments since then (Microsoft has discontinued support for their XNA platform) coupled with the lacking source code documentation and code practices have have made further modifications difficult.

2.6.1 MeshTest usage

The following is a run-through of a MeshTest experiment:

Introduction The user is presented with an introductory test describing the experiment. It specifies the amount of model triplets and the

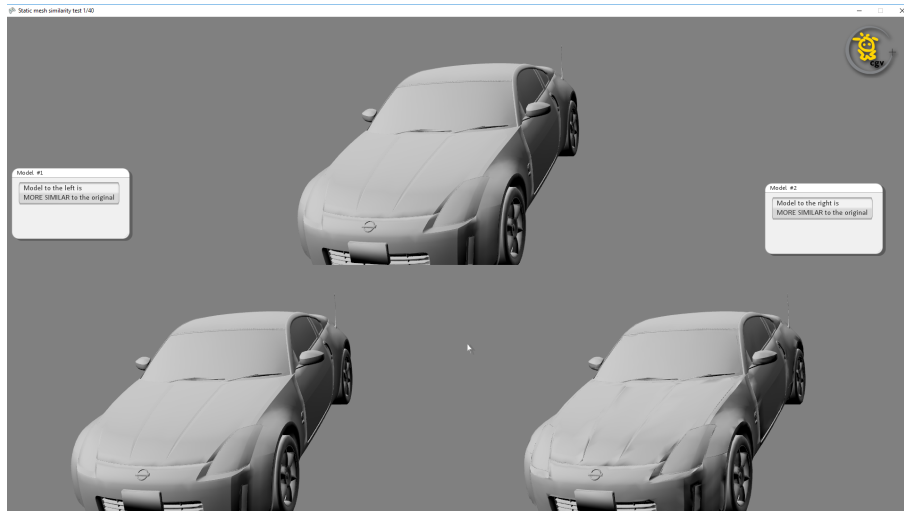

Figure 2.2: MeshTest software

intended comparison criteria. It also specifies that there is no time limit, but the experiment is expected to take about 25 minutes.

Mouse control - moving A visual tutorial explains to the user that they can move the model around the screen by dragging the mouse with the right mouse button pressed.

Mouse control - rotation The tutorial continues to explain that the model can be rotated by dragging the mouse with the left mouse button pressed.

Mouse control - magnification The tutorial explains that the model can be rescaled by using the mousewheel and reset to a neutral size using the spacebar.

Experiment clarification The comparison criteria is explained again, with an example image to increase clarity.

Tutorial finalization The tutorial concludes by clarifying what to do if the user cannot decide between the models presented (pick at random) and that the experiment is voluntary, anonymous and can be terminated at any time by closing the window.

Sex choice The user is presented with a binary choice of their sex for the purposes of data gathering.

Experiment The user is presented with 40 model triplets in sequence and asked to select either the right or left model from each as the more

similar to the central one.

2.7 Virtual reality

Virtual Reality can be defined as an immersive simulation of a three-dimensional environment, created using interactive software and hardware, and experienced or controlled by movement of the body.

2.7.1 History

While the term "virtual reality" was first coined in 1987 by Jaron Lanier (founder of the visual programming lab) (Lanier 1992), devices that were in some form related to the concept have existed for decades. One early example is the Headsight HMD developed for military use in 1961, which allowed an operator to move a remote camera via head movement.

"The ultimate display" concept, written about in 1965 by Ivan Sutherland, was the first description of the modern concept of immersive virtual reality. "The sword of Damocles" also developed by Ivan Sutherland along with his student Bob Sproull was the first head-mounted motion tracking device that used computer generated graphics for the displayed environment.

In 1995 the first consumer-grade virtual reality device was released, the Nintendo Virtual Boy. The technology at the time was however still too primitive and the product was a failure, being taken off-market in only six months after release. For the next two decades, the Virtual Boy would serve as a prime example of why virtual reality devices are considered unfeasible.

2.7.2 Current state

In recent years costumer-grade hardware has finally achieved resolution and synchronization speeds comfortable for human usage and advances in motion-sensing technology has allowed for near-perfect tracking of positions of objects in reality and matching of their location in the virtual space (in most cases this means the headset and controllers).

2.7.3 Advantages

Virtual reality offers some advantages over a standard monitor + controller setup.

Stereoscopic vision Unlike with a traditional computer monitor, virtual reality headsets allow for stereoscopic vision. This allows the user to better judge distances and better navigate the virtual world.

Immersion With the headset covering the users entire field of vision, there is nothing in the visual space that can distract them from the virtual world. Coupled with a position audio system, this leads to a much higher amount of immersion compared to a traditional setup.

Presence Presence is defined as the sense of being physically present in a virtual world. Closely related to immersion, it is however distinct as a user can be immersed in a situation without necessarily placing themselves within it. Positionally tracked controllers help with this, as they are often used to simulate interactions analogous to those a user would be capable of performing with their own hands. Some examples include firing a bow by drawing and releasing a virtual bowstring or picking up objects by moving the controller into the right space and squeezing a grip button.

Ease of use The position tracked headsets allow the user to interact with the virtual space in a much more natural fashion. For example, in order to look at something, the user simply has to face in its direction. In a more traditional control scheme, the user would have to move a camera or change the facing of their avatar using a controller or a mouse. In the case of room-sized experiences this extends to movement, as instead of moving an avatar the user can simply walk to the desired location. With position tracked controllers, interacting with objects in the virtual space also becomes more in line with a users natural expectations.

2.7.4 Issues

There are however also some disadvantages.

Inability to see In order to facilitate proper immersion, the headset has to be opaque and cover the users entire field of view. The obstructed vision can result in accidental collisions.

Controller limitations For any controller that is not position tracked (such as the standard keyboard and mouse) interaction can become rather difficult as the user cannot actually see what they are doing.

Screen door effect Displays with insufficient resolution or built for a different purpose can cause the screendoor effect, wherein the user can distinguish the individual pixels on the headset screens and thus can see a screendoor-like covering over their entire field of view.

Lens distortion With the lenses of virtual reality headsets being located so close to the users eyes, some distortion is inevitable, as a set of lenses capable of perfectly distorting a flat screen display to the human eye's field of view would require a set of nine lenses (Abrash 2014) that would be too bulky to fit within any reasonable headset design. Careful positioning, purpose-built lenses and countermeasures implemented in the software layer can minimize this effect, but it cannot be eliminated completely.

Limited field of view The lack of a field of view provided by virtual reality headsets equivalent to a users actual field of view results in limited peripheral vision.

Loss of tracking When the headset loses tracking, the displayed world ceases maintaining synchronization with the users movement. This can be uncomfortable or render the application unusable, or in extreme cases even temporarily blind the user (until the headset is removed).

Real-world mismatch Care must be given when constructing virtual experiences so that they match the users space as closely as possible given the limited real world information available. For this reason virtual reality experiences often rely on the user either staying in a seated position or, when movement is necessary, limiting the interaction to a predetermined area that matches the users available space, otherwise problems can occur (such as accidentally walking into a table that fails to be presented in the virtual space). A different consideration of the same problem is the possibility of an user relying on the presence of a virtual construct that does not exist in reality (such as by attempting to sit in a virtual chair).

2.7.5 Health effects

It must be noted that virtual reality has been known to cause health issues in some users, such as temporary nausea. Depending on the individual user there can also be significant differences of both the onset time and severity of such effects.

Due to these issues it is generally recommended to keep modifiers to the users position to an absolute minimum, limit sessions to an hour or less, and to remove the headset as soon as discomfort is experienced.

Simulator sickness, Motion sickness, Virtual reality sickness While significant progress has been made, there are still some problems caused by the difference between the virtual reality perceived by sight and the actual reality perceived by other senses. These closely-related but distinct effects are known as simulator sickness, motion sickness, and virtual reality sickness. Virtual motion is a typical culprit of causing nausea and discomfort, as the body senses no motion is occurring regardless of the velocities experienced because of the scene presented by the VR device. The opposite is also true, as it can prove very unpleasant if the application restricts the camera despite the users movement.

Headset A more mundane issue is the weight of the headset itself and the straps necessary for it to stay firmly attached to the head as well as the bright screens focused into the eyes from a distance of a few centimeters causing discomfort due to prolonged usage.

Vergence-accommodation conflict The Vergence-accommodation conflict (Hoffman et al. 2008) is caused by the fact that unlike with the real world, where the eye's focal distance and vergence distance match for any given observed object (after a short accommodation period), with virtual devices the actual distance to the screen is always the same regardless of the virtual scene's perceived distance. While the human eye is capable of adjustment, long term use can cause eye strain and fatigue. While great strides have been made in the field of reducing this effect (Huang, K. Chen and Wetzstein 2015), completely eliminating them is difficult if not impossible.

2.7.6 Uses

Virtual reality can be used in many varied applications from various fields.

Entertainment Virtual reality can be used in the entertainment sector in many ways. Virtual reality games take the lessons learned from the decades of game development and apply them to virtual reality, though this is still a rapidly developing field with many issues yet to be solved, as many of the traditional videogame paradigms translate to virtual reality poorly at best. Virtual reality experiences focus on the possibility of virtual reality to simulate locations that the user

might not be capable of traveling to, wherever for monetary reasons (a virtual trip to Paris), simply because they are incapable of it (the frozen summit of Mount Everest), because the means of travel does not exist in the first place (a crater on Mars, or the night sky as seen from one of the moons of Jupiter) or even because such a location is fictional (visiting Helm's Deep from the Lord of the Rings series). 3D movies, when experienced in virtual reality, allow for an entirely new level of immersion, as the user is essentially fully surrounded by the experience. While with recordings of the real world an impressive amount of expensive technology is required to record the necessary 3D data, an avenue worth considering is computer animation movies.

Medical In the medical sector, virtual reality has been used as a tool to allow telepresense of surgical machine operators over vast distances. In a similar vein, 3D scans of patient bodies and cadaver slice scans can be more naturally observed in virtual reality as opposed to 2D slice displays. While human body simulation is not yet quite sophisticated enough to offer perfectly accurate results, virtual reality can also be used for medical training without risking the life of an actual patient.

Military The military sector has been experimenting with virtual reality uses for decades. Simulators are a well-established use of virtual reality, allowing for vehicle pilot or driver training without risking the lives of the operators or the expensive vehicles in real training exercises. They also allow for exposing the operator to situations that would in real life prove fatal. Other uses have also been explored, such as troop training simulations or remote control of turret weapons and vehicles.

3 Project goal

This project has three primary goals:

Create a VR application that is equivalent to MeshTest Given the increased ease of interaction and user immersion as well as the increase in field of view, refresh rate, and other such statistics that affect the quality of perception of a virtual reality-based environment as opposed to a typical monitor display, it is of interest what effects will a virtual reality-based experiment environment have on the results of a study conducted in a manner equivalent to MeshTest. Hopefully, the results will be close to identical, thus proving the accuracy of the DAME metric regardless of the display method used. If this proves not to be the case, an alternate set of metrics and perception-oriented algorithms may be required for use in virtual reality applications.

Additionally, the ability to observe meshes up close and from different angles with ease should allow for a more natural interaction of using the users own body to move around the environment as opposed to controller or keyboard-and-mouse based interaction, thus reducing the amount of preexisting experience and time needed to participate in an experiment.

Explore the problem space As a part of this project comes determining the requirements of a project of this nature for future efforts of a similar nature. The knowledge gained in its development will hopefully lead to faster and easier development should a similar virtual reality project be required in the future.

Draw conclusions After the project is finished, it is to be analyzed. Conclusions are to be drawn as to its viability as a MeshTest alternative in specific and as a data collection tool in general. Depending on its relative usability, further experiments of a similar nature may also make use of virtual reality technology.

4 Project design requirements

The goal of this project is to create a virtual reality-based application that serves the same purpose as the original MeshTest software, with some additional improvements as detailed in the project's initial description.

The following is a listing of project-critical requirements:

- Implement functionality identical to MeshTest.
- Expand on functionality with required additional features (such as experiment definition files and result logging).
- Perform pilot study to verify functionality.

In addition, there are these requirements:

- Expand on functionality with quality-of-life features (such as further experiment customization, user recordings, or an experiment editor application).
- Improve ease of use and clarity of experiment task.
- Iterate on user interface design based on user feedback.
- Explore the results of a VR interactive environment on the experiment results.

Otherwise, in order to allow an accurate comparison, the application must be as similar to MeshTest as possible.

There are two kinds of experiments that must be supported: Binary forced choice and Anchored scoring.

4.1 Binary forced choice

In this experiment, the user is tasked with choosing which of the two options presented better match the original. The original is also presented for comparison. The two options may be identical, or contain the original. This kind of experiment focuses on enabling the subject to decide between relative options instead of assigning an absolute rating value to any specific model variation. This leads to a result set where it is possible to broadly sort the subject's opinions on models from best to worst without ever requiring the subject to sort them in such a fashion themselves, at least given the assumption every model pair was presented.

4.2 Anchored scoring

In this experiment, the user is presented with an original and a copy and is tasked with rating how closely the copy matches the original on a predetermined scale. The copy may be identical to the original. This kind of experiment allows for a more granular result, but is susceptible to the problem of rating compression: since the subject initially has no frame of reference, they may assign the minimum value to a model they consider awful, only to then not be able to assign a lower value to a later, much worse model (or vice versa with high-quality matches).

5 Application use cases

The application has two distinct categories of users: there are users *performing* experiments (experimenters), and there are users *participating* in experiments (test subjects).

5.1 Typical usage (experimenter)

Define the experiment The experimenter defines the experiment to be run, using either a purpose-built editor or an experiment definition file.

Perform the experiment The experimenter runs the experiment on one or more test subjects.

Collect the results The experimenter collects the results of the experiment, collates the data and draws conclusions from it.

5.2 Typical usage (test subject)

Learn how to interact with the VR environment The subject learns how to participate in the experiment with no prior experience and minimal assistance.

Perform the experiment The subject performs his run of the experiment.

5.3 Conclusion

Based on the typical usages, it is clear that experimenter users will interact with the application mostly by manipulating experiment data files, while the bulk of VR interaction will be performed by the test subject users. Given this and the isolating nature of a VR environment (which completely obscures the users vision), it makes sense to focus the VR environment on ease of use by users with no prior experience with the application itself, and in extreme cases computer use in general.

5.4 Result logging

While it is certainly possible for the experimenter to manually log the results of the experiment, noting down the user choices between model pairs as they make them, this is impractical given the length of an individual experiment. A much better approach is for the result logging to be automated, printing the subject's choices to the command line as they are made or saving them to a file for later processing.

Additionally, while logging the results of the subject's choices is sufficient for the purposes of the experiment, an extension on the concept is logging the user's interactions with the application itself. Not only does this ease debugging during development, the resulting data on user movement and interaction choices can also be used to further improve the user experience and offers at least some insight into the subject's thought process.

In order to simplify data gathering, automated sending of the results is a desirable feature. A simplistic implementation approach is to e-mail the results to a provided e-mail address. In order to extend the supported experiment time, the file size of the resulting e-mail attachment must be kept below the threshold limit at which significant numbers of e-mail clients restrict file attachments of that size.

6 Application design

The following components are necessary to implement this project:

Hardware platform The subset of virtual reality devices and their associated supporting hardware platforms that will be used by the application.

3D graphics engine A 3D graphics engine that supports both the VR device and its platform that can be used to develop the application.

File storage A method of data file storage, either offline on-disk or cloud-based online.

Additionally, the following must be decided upon:

Method of interaction A system for user interactivity that incorporates aspects unique to virtual reality spaces.

Experiment definition file format A file format for the files that define experiments.

Experiment log file format A file format for the logged results of an experiment.

Experiment recording file format A file format for the recording of a user's movement during an experiment.

6.1 Hardware platform

Since the field of customer-grade virtual reality devices is still in its infancy, the quality of presentation as well as user interaction devices vary greatly with device manufacturer and model due to a lack of standardization. The following is a brief overview of VR devices and their associated platforms available.

Oculus Rift DK2 (PC) The most recent device available at the start of development. Despite being a development kit, the DK2 is a fully capable virtual reality device that remains supported to this day. Its main disadvantages lie in the lack of motion tracked controllers and somewhat lower resolution.

Oculus Rift CV (PC) The consumer release version of the Oculus Rift, featuring an improved screen resolution that drastically reduces the screen-door effect and more accurate head-tracking. There are now also motion-tracked controllers designed to work with the Rift, but they are not a part of the default package. Unfortunately a Rift CV was not available for development or testing.

Playstation VR (PS4) The Playstation VR headset only works with the Playstation 4 game console. This, coupled with the fact its motion-tracked controllers are also optional (and less accurately tracked), as well as the fact that none were available, made it a poor choice.

Google Cardboard, others (smartphones) Smartphone-based platforms like Google Cardboard are designed to be a more affordable alternative to full-featured VR devices. As such, they lack positional tracking and their resolution and processing power is limited to the smartphone used.

HTC Vive (PC) The HTC Vive was released at nearly the same time as the Oculus Rift CV, but unlike the Rift it contains the motion controllers as part of the initial package (and thus they can be relied on to be present for every user). In addition, its Lighthouse-based motion tracking allows for room-sized interactive experiences as opposed to the seated-position restriction of the alternatives. Out of the options, the Vive proved to be the best final target platform.

6.1.1 Technical specifications

A brief comparison of technical specifications of the mentioned devices:

Device	Resolution	Refresh rate	FOV	Head track.	Cont. track.
Rift DK2	960x1080 per eye	75 Hz	100	Yes	No
Rift CV	1080x1200 per eye	90 Hz	110	Yes	Optional
PS VR	960x1080 per eye	90 Hz	100	Yes	Optional
HTC Vive	1080x1200 per eye	90 Hz	110	Yes	Yes
Cardboard	Varies	Varies	Varies	No	No

Table 6.1: Device specifications comparison.

6.2 Engine

A 3D graphics engine is an API layer that handles the rendering of 3D objects and the displaying thereof on a display device (usually a computer monitor screen, but other options are also possible, such as a virtual reality headset) through interfacing with the operating system. A typical 3D engine usually takes care of interaction with more low-level graphics APIs such as OpenGL or DirectX, and through them with the GPU itself. Additionally, it simplifies the handling of data such as vertex and index arrays for triangular meshes, texture coordinates and images, and other associated data. It also provides related functionality, such as transformation matrix operations, 3D projection helper functions, or physics simulation.

Videogame 3D engines are more specialized engines. In addition to the functionality mentioned above, a modern videogame engine usually includes a 3D positional audio system, an input handling system capable of handling both keyboard-and-mouse and controller input, a networking layer, and a realtime physics simulation system. They are usually designed to handle a steady framerate. While they are meant for a different class of application, well designed ones are also perfectly capable of supporting simulations and any other purpose of a similar nature.

The choice of an underlying engine is an important one, as by their very nature engines dictate a significant portion of the application's internal design. The following options were considered:

From-scratch solution This approach would entail implementing an application specific engine from scratch on top of a bare graphics processing API layer, such as OpenGL or DirectX. While this does result in the highest possible amount of control over the application, it is a project unto itself and a very time consuming and unecessarily error-prone solution for a problem with already existing fully-featured solutions. It would also necessitate interfacing with the virtual reality device using its raw API instead of a prebuilt engine-specific interface package.

Unreal engine Unreal engine is a time proven high quality engine with many web-based resources available, both for tutorials on programming with it and assets. It is however very game-oriented. It also uses C++ as its scripting language, which is a powerful but difficult to master programming language.

VTK While VTK better fits the problem domain, during initial develop-

ment it was not yet capable of VR integration. Its module-based system is also an unnecessary complication in this projects case.

Unity While Unity is primarily a videogame engine, its implementation is generic enough to be easily usable for this project. This coupled with its very early implementation of native Oculus Rift integration as well as usage of the C# language as its scripting language ultimately made it the best choice.

6.2.1 Unity

Unity (also known as Unity3D) is a videogame engine first released in June 6, 2005. While initially only supporting Mac OS X, the engine quickly expanded support to Windows and web browsers, later also expanding to the iPhone, Adobe Flash, Linux and Android.

One of the major reasons for Unity's success (Haas 2014) was its support for independent developers who did not have the resources to license expensive game development technology. Until the more recent release of Unreal Engine 4 in 2015, there were precious few options for developers looking for a professional grade engine affordable on a budget. Combined with Unity's quick rate of improvement, extensive documentation, up-to-date video tutorials and an open approach to developer feedback-based improvements this has made it one of the most used engines by independent game developers.

The strongest feature of Unity is its editing interface, the Unity Editor. In comparison to alternatives at the time, the Unity Editor is both powerful and incredibly easy to use, supporting such conveniences as drag-n-drop, file handling and automatic importing of model, texture and image data. Object manipulation is handled through handle controls similar to that of 3D modeling software, albeit simplified for ease of use.

In Unity, scripting is handled through usage of a powerful strongly-typed component system, with support for two languages: Unity JavaScript an C#. Historically, Boo was also supported, but it was dropped due to the minimal amount of developers using it. Other features include a robust animation system (Mecanim), a standardized particle system and a code-controlled wireframe rendering system intended for debugging. The inspector for game object components is automatically dynamically generated from their class files and allows for object specific direct variable entry, though a custom inspector can also be implemented on a class specific basis.

Unity also supports live debugging, with the possibility of changing game object positions, rotations, hierarchy, variable data, and even recompiling of

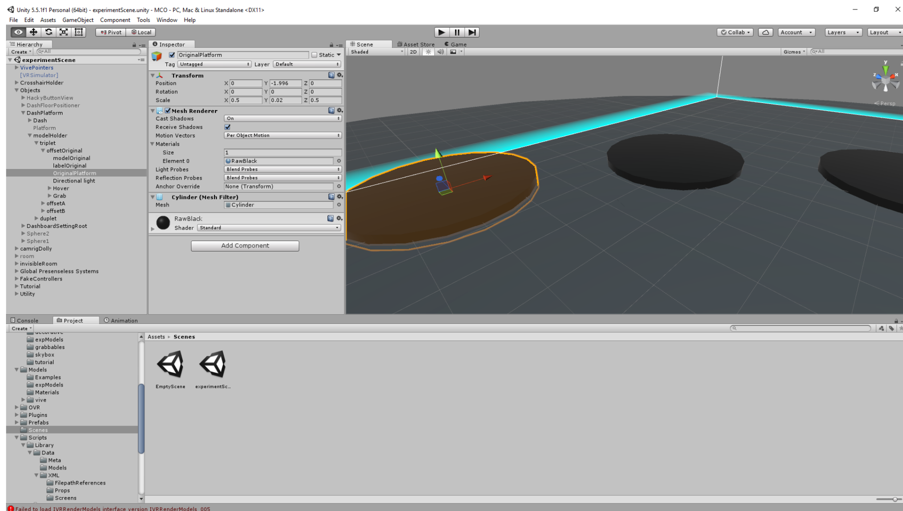

Figure 6.1: Unity editor

the scripts themselves during gameplay. It also contains a powerful profiler, allowing a developer to monitor the application's memory and CPU time usage, as well as framerate and other miscellaneous statistics. The profiler can also be used to track down problematic issues such as frame drops, down to the individual method calls causing them.

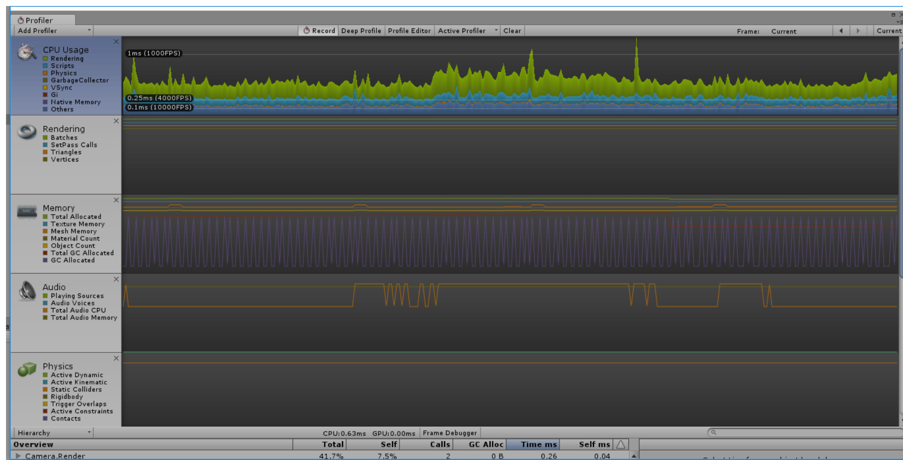

Figure 6.2: Unity profiler

6.3 File storage

The application must be capable of loading, editing or saving the following data files:

- Triangular mesh models
- Experiment definition
- Experiment log
- Experiment recording

There are several options for file storage:

Disk storage Simply storing the data files directly on disk. A simple solution that is easy to debug and practically necessary for the application to be usable.

Online dedicated storage Storing the data files on a dedicated server accessible by internet connection and transmitted over FTP. This option would allow remote storage and acquisition of experiment data.

Online cloud storage Using a cloud-based file storage solution, such as Google Drive or Dropbox. This option is simpler to set up than a specialized server while still offering most of its advantages, but it relies on a third party provided service.

Since online storage is not part of the project requirements, it is a optional feature. Disk storage is trivial to implement and required for any degree of iterative testing and is therefore the most sensible initial choice.

6.4 Interactivity

In order for the application to be interactive, it must be capable of receiving user input in some form. The following options were considered:

Mouse + keyboard A traditional input method which uses the mouse as a pointing device and the keyboard as a general input device. This input method presents the issue of users losing hold of their mouse/keyboard and having difficulty regaining it due to obscured vision.

Classical controller While controllers have adequate amounts of inputs (including analog axis movement that can be mapped to model rotation), they tend to require memorization of button location and do not provide as quick an input as the alternatives.

Leap Motion controller While this hand-tracking input device offers some intriguing options for design, its requirement of an additional API integration module as well as its relative rarity in the general user base left it as a last-resort option.

Gaze controls This option would eschew any external input devices in favor of using the users vision directly as an input.

Position tracked controllers Controllers whose positions are tracked in the real world in real time, allowing them to be represented by their virtual counterparts inside the virtual space.

6.5 Experiment file format

In order to store the experiment between application uses, its definition must be able to be saved to a file. This file must be easily editable, preferably without the usage of a specialized editor and should ideally be of a small size. There are several options available in order to achieve this, each with its own advantages and disadvantages:

Raw binary file While an application-specific binary file does offer the best possible file size, this is of little importance for this particular problem. The disadvantages include susceptibility to complete loading failure or corruption due to errors caused by a few misplaced data points, the necessity of implementing an editor to edit the file, and the lack of human readability.

Unity asset file Unity provides asset export functionality, including reusable prefabricated assets (prefabs) created an runtime. These prefabs could be made to contain an experiment definition. This could be used to export experiments created using an editor included in the application. However, while this is the solution that is most native to Unity, the lack of an ability to quickly edit the file is an unnecessary hindrance.

Raw text file A text-based application specific file (such as a CSV file) does provide increased human reliability and can be edited using a standard text editor of the users choice, but remains prone to errors.

XML file Using the standardized XML file format offers several advantages. Text editors with syntax highlighting or similar features will usually include XML in their supported languages. The file format is

also somewhat resistant to errors. Furthermore, C# natively supports object serialization into XML files, massively simplifying their definition and eliminating any accidental differences between the file format and the actual supported experiment definitions caused by any additional layers between a file loader and the application objects themselves. The verbose nature of XML does however lead to comparatively larger file sizes.

In the case of the experiment definition file, the XML file is the best fit.

6.6 Log file format

The options for the log file format are the same as those for the experiment file format, but the considerations are different. The file must primarily be easily human-readable. Additionally, it must be simple to parse in order to allow automatic experiment result data entry. Given these, a raw text file is the most appropriate choice.

6.7 Recording file format

In the case of the recording file, the primary consideration is file size as the recording must keep track of position and rotation data of multiple objects at sixty data points (corresponding to sixty fixed physics frames) per second. The file should also be simplistic in order to allow better file compression. Therefore, the best option is a specialized binary file format.

7 Implementation details

The project is split into two primary code portions: the part that interfaces with Unity classes and the part that interfaces with native C#, though there is of course some overlap. The major distinction that necessitates this divide is that Unity objects can only be manipulated by the Unity primary thread, and cannot be serialized.

7.1 Singletons

In order for the two parts of the application to interact, some sort of interface was necessary. The application uses singleton objects as this interface. During initialization, a singleton object attached to a Unity gameObject assigns itself into its related globally accessible static variable which can then be accessed by classes which are not part of the Unity system themselves. While there are alternate solutions such as searching for specific gameObjects by name or class, they are comparatively slower and less reliable (as it is possible for more than one object with the same name or component to exist). The singletons used by the applications are:

UnityInterface The primary singleton that holds references to most of the gameObjects used by the application and all utility methods used to manipulate the world state.

DebugToggles This contains contains a few binary switches that control various debug options in the application. It is a singleton simply because it is easier to change values through the the inspector rather than edit class files or setting files and force recompiling the application. They can also be changed during runtime.

DashController This singleton controls the dashboard user interface, enforcing that always one and only one dashboard is visible at all times.

MaterialsHolder This singleton holds references to the materials used for rendering models, allowing them to be quickly and easily changed through the inspector.

PrefabHolder This singleton holds references to various prefabs (prebuilt gameObjects that can be created on demand).

TutorialStages This singleton holds references to the gameObjects that represent the various stages of the VR device-specific tutorials and controls the displaying thereof while the tutorial is running.

7.2 GlobalStateHolder

The GlobalStateHolder is a static class that contains references to all globally accessible singletons as well as utility methods for accessing the currently loaded experiment, the currently active screen, and the next screen in the experiment screen list. It also contains the application exit method.

7.3 User input system iterations

The user interactivity implementation was iterated upon during development. The following are the major revisitions presented in order of implementation time:

7.3.1 Keyboard/mouse button iteration

The most simplistic and thus initial solution was to simply assign buttons on the keyboard and mouse to the interaction required (specifically, confirmation of instructions and choosing between two options, for a total of two key bindings), binding mouse movement to model rotation and the mousewheel to model scale. While this solution was functional, it was prone to accidental input, lack of intuitiveness and losing hold of the mouse/keyboard.

7.3.2 Gaze input iteration

In this version, instead of clicking buttons on the mouse or the keyboard, the user could simply focus their vision on a virtual button using a worldspace cursor in the center of their vision, and after a short interval a click would be triggered. While this did reduce the need for keyboard or mouse input, it did not eliminate it entirely as the mouse was still required to rotate or scale the models. Additionally, the need to hold the users head almost perfectly still while facing specific objects, even for short periods of time, proved to eventually lead to neck cramps and was much slower than the previous solution.

7.3.3 Worldspace cursor iteration

Using parts of both previous approaches, the mouse was now used in a more typical fashion, moving the cursor implemented for the gaze input in a rectangle projected ahead of the user and locked to the users field of vision, with mouse buttons triggering clicks as normal. Model rotation and scaling was moved to sliders on a worldspace panel located in front of the user, which allowed for a more precise manipulation with no accidental input. This solution proved as a more natural approach.

7.3.4 Position tracked controllers iteration

After shifting development to the HTC Vive, it seemed prudent to take advantage of its position tracked controllers. Instead of projecting a cursor from the user themselves, there are laser pointer-like cursors projected from the front of the controllers which can be used to interact with the virtual world much like one would with a pointing finger. Model rotation using the controllers directly was also implemented (mimicking the user grabbing the object and rotating it by hand), though the manipulation panel remains as a more precise option.

7.3.5 Final solution

The finalized solution is the position tracked controllers iteration combined with a fallback of the worldspace cursor iteration in case the VR device in use does not use position tracked controllers. In order to improve ease of use, a highlight system was added that dynamically displays 3D hint objects when the user moves a controller near to an object with interactivity capabilities.

7.4 Model data

While the Unity engine and editor do have functions for loading and processing triangular mesh model data, they are not available at runtime, as Unity processes any such data into Unity-specific assets which then become part of the application's resources. However, since the .obj and .dat formats are relatively simply defined, it is not difficult to implement a solution for loading the raw data separately. Once loaded, Unity has functions available for turning raw triangular mesh data into a Mesh object, which can then be used by the Unity engine. Simplified definitions of the formats used are as follows:

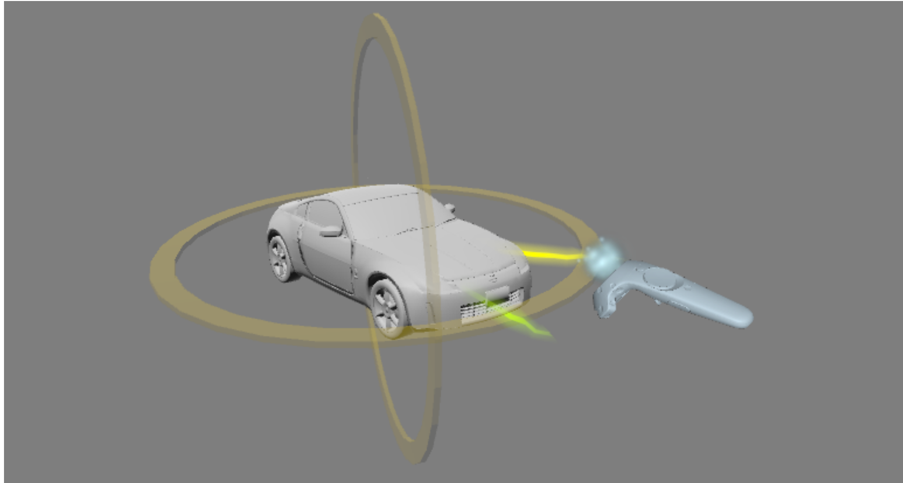

Figure 7.1: Controller gripping a model

7.4.1 wavefront .obj format

One data point per line. Data separated by spaces, with the first entry specifying data type.

- Vertex definition: v x y z
- Vertex normal definition: vn x y z
- Face definition: f v1 v2 .. .. vn
- Face with normals definition: f v1//n1 v2//n2 .. .. vn//nn

Note that this is only a small subset of the format. In order to process all possible files, the .obj loader must also handle cases where the data contains texture coordinates data.

7.4.2 application-specific .dat format

The .dat format, used originally by MeshTest, reduces the file size by limiting itself to only to vertices, triangle faces, and vertex normals, as well as using a binary file as opposed to a text file. The definition is as thus:

- Triangle faces count represented by a single 32 bit integer
- Index data, represented by three 32 bit integers per face, one for each vertex in the face
- Vertex count represented by a single 32 bit integer

- Vertex data, represented by six 64 bit doubles, one per each coordinate and one per each vertex normal coordinate

As with all simplistic binary files, the data must be in this exact order or the file will fail to load or result in corrupt data.

7.4.3 Model loaders

There are three model loader classes. Two of them (ObjLoader and DatLoader) correspond to their respective model data formats. The third (QuickLoader) skips loading entirely, instead creating a placeholder object that only contains the model's file path. This loader and placeholder objects are used during model preloading. All three loaders also inherit from the abstract class AbstractLoader which implements their common functionality.

7.4.4 Postprocessing

Once the data is loaded, there are only a few things left to do.

Coordinate system mismatch Since the files use different coordinate systems, they must first be matched (specifically, by inverting the z coordinate of .dat files).

Recentering In order to keep models in the same spot when switched to, all mesh data is recentered so that its averaged center lies at (0,0,0). This is done simply by adding up all vertices, then dividing by their count and substracting the resulting offset from every vertex in the mesh.

Splitting Unity has a maximum limit of 65535 vertices in a single mesh. When a model with too many vertices is imported into Unity using the usual means, the import process will automatically split the mesh into submeshes to avoid this issue. However, when meshes are generated dynamically (or in this application's case, created from raw data without involving Unity code), this check is bypassed. Since the importer code is not exposed, the process must be reimplemented. Due to the relative rarity of this issue, a naive approach to solving it is sufficient. The application simply adds faces to a submesh until it nears the limit, at which point all vertices used by this subset of the mesh faces are duplicated into a separate array that the submesh then uses as its vertex array. This process is repeated until all mesh triangles have been duplicated into submeshes.

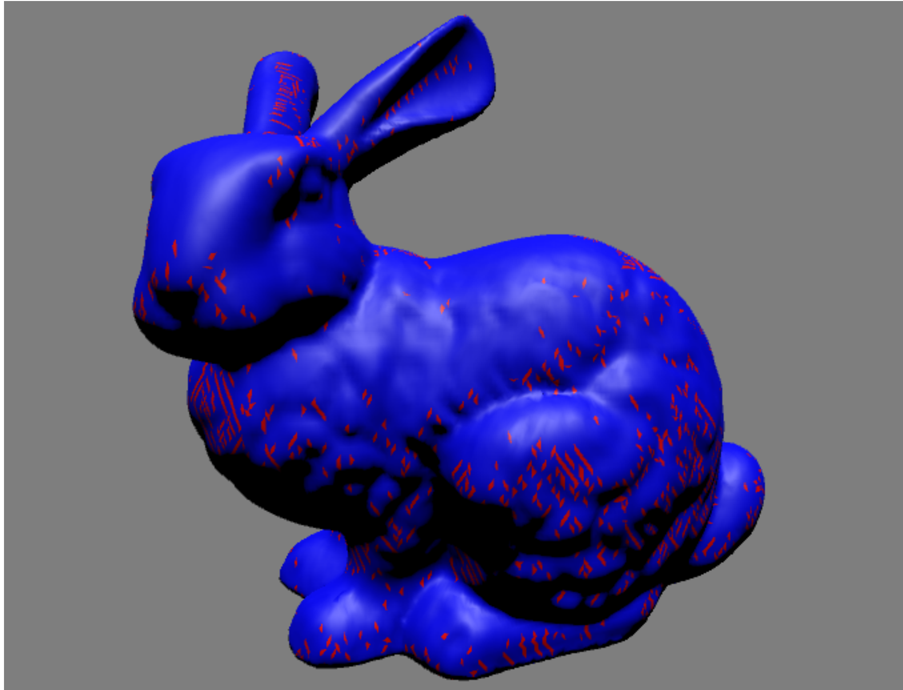

Figure 7.2: High-poly bunny mesh with separated polygons highlighted

Normals validation Some of the meshes used for experiments have incorrect normal directions. These cases result in incorrect visuals, as the object essentially looks as if lit from the inside out (if double-sided face rendering is enabled), or as an inversion of itself (if it is disabled).

In order to combat this issue, the application performs a normals check: for a small number of randomly selected normals, a ray is fired in its direction. The number of hits is then counted. An even number of hits (including zero) implies correct normals facing, while an odd number implies inverted facing. The amount of odd and even results is then totaled, and if the number of odd results exceeds the number of even results, the normals of the entire mesh are inverted. This check can however only be performed on a closed mesh (as with an open mesh, the rays could enter or exit through the holes in the mesh). For open meshes, the application relies on disabling backface culling for model rendering.

At this point, the mesh is ready to be displayed. It remains in this state until the application needs to display it. At that point the mesh data is inserted into a Unity Mesh object, which is then provided to a MeshFilter component attached to a GameObject that exists within the virtual scene.

7.5 Screen system

In order to represent the experiment, a system of screen definitions is used. A screen in the application is a distinct scene of an experiment, such as a personal data entry screen or a tutorial screen. Each screen has an entry point and exit point. When a screen switch occurs (usually when the user finishes the task the screen represents) the following order of operation happens:

- The current screen finalizes any data storage it may be responsible for.
- The current screen runs any screen-specific cleanup code.
- The generic cleanup code is executed for the current screen.
- The next screen runs its screen-specific initialization code.
- The generic screen initialization code is executed.
- The application switches to the next screen.
- The screen after the next screen starts preloading any model data it may initially require.

There are two special cases. In the case the current screen is the second-last or the last one, the preloading step is skipped. In the case the current screen is the last one, the initialization of the next screen is also skipped and the experiment finalization code is executed. Finally, in order to simplify this process and eliminate edge case errors, a dummy screen is inserted at the beginning of each experiment. This screen simply proceeds to the next screen on the first frame its displayed on.

7.6 Individual screen implementations

Since the individual screens implement most of the applications functionality, they each contain a sizable portion of unique code. In order to further distinguish their classes from similarly-named base C# classes, they contain the keyword MCO in their name (standing for MeshComparisonOculus, from when the application only supported the Oculus Rift).

7.6.1 BaseMCOScreen

The abstract class is inherited from by all other screens. It defines the following abstract or virtual methods and their common functionality:

jumpedTo This method is called when the screen is jumped to (usually from the preceding screen) and contains setup code.

jumpedFrom This method is called when the screen is jumped from and contains cleanup and finalization code.

frameEnd This method is called once every logic frame and contains update code.

frameEndfixed This frame is called once every 1/60th of a second exactly, and contains update code that relies on precise timing.

setAsFinished Sets this screen as finished. On the next frame, the screen system will jump from it to the next one.

preload This method is called during a jump to the preceding screen. It starts the preloading of any models required by this screen.

getShortLabel This method is used to fetch the short label displayed on top of the screen dashboard.

getText This method is used to fetch the longer descriptive text that some screens use.

Additionally, as part of the abandoned prop system, it defines the following methods:

serialiseProps Serialises props into their Unity-independent versions, as Unity game objects cannot be serialised.

deserialiseProps Deserialises props from their Unity-independent versions into Unity game objects.

showProps Enables all props attached to this screen.

addProp Adds a new prop to this screen.

7.6.2 MCOCommandlineScreen

Intended primarily for debugging purposes, this screen displays the commandline arguments the application was launched with, using the GetCommandLineArgs method of the Environment class from the System namespace.

7.6.3 MCOCustomScreen

The simplest of the screens, this screen implements no base functionality whatsoever. Its intended primarily for displaying text to the user. It was also meant to be used with the unfinished prop system.

7.6.4 MCODummyStartScreen

Unlike other screens, this screen is not supported in the XML definition file, but rather automatically added to the start of an experiment during the loading process. It is a simple solution to some edge case issues (such as the lack of a model preload step for the first screen in an experiment). It automatically sets itself as finished when jumped to, resulting in a single frame duration.

7.6.5 MCOModelDisplayScreen

Used as a single model display. The model is displayed using the central model gameObject of the comparison experiment triplet.

7.6.6 MCOPersonalInfoScreen

Used to collect subject personal information. Since the only data required at this point is the user's sex, this boils down to a simple two-button binary choice. If further data is required in the future, this class is the one that will be modified.

7.6.7 MCORecordingPlaybackScreen

Plays back an experiment recording file. In order to allow quick seeking, all the models required by the experiment recording are loaded immediately. During playback the process is simple: during every fixed frame, read the next frame's data and update the position and rotation of the experiment models as well as the headset and controller stand-ins, then check if any of the models have changed and replace them if they have. When seeking, the file reader skips to the desired percentage and then keeps reading until it finds the magic constant "|F|", at which point it proceeds at normal.

7.6.8 MCOTutorialScreen

Most of the tutorial functionality is implemented in the TutorialStages singleton. This screen therefore simply calls its setupPath() method when jumped to.

7.6.9 MCOCustomScreenEditorScreen

While this screen can be inserted as a part of an experiment definition, this is not its intended usage. Rather, this screen allows the saving, loading and creating of other experiment definition files. It provides an editing interface capable of changing most of the defining variables of screens contained in an experiment, as well the addition of new screens and removal of existing ones.

7.6.10 BaseExperimentScreen

This abstract class implements the common functionality between the two supported experiment types and is inherited from by their respective screens. In its frameEnd override, it checks if all models from the current duplet or triplet are displayed, and if not and they are done preloading, displays them. It also handles the counting of tasks performed by the user and storage of available models. It defines the following abstract method:

generateChoiceSet This method generates a set of model data paths from the given set to perform an experiment on (either a duplet or a triplet, in either case containing the original model).

Additionally, it implements the following methods.

addDataSource Adds a new model data source.

deleteDataSource Removes an existing data source.

setupModelStructure Generates the model structure required to run this experiment.

setupNextChoice Called after the user performs a step in the experiment and a new set of models is required. It calls the generateChoiceSet method, then starts preload threads for the model data paths generated by it. It also handles the final step by setting the screen as finished instead.

7.6.11 MCOComparingExperimentScreen

This screen represents the Binary-forced choice experiment method. Its implementation of the generateChoiceSet method generates model triplets using a simple random generation algorithm. Depending on the flags set in the experiment definition file, the algorithm might exclude triplets that contain the original model in one of the copy slots or triplets that contain the same copy twice.

7.6.12 MCORatingExperimentScreen

This screen represents the Anchored scoring experiment method. Its implementation of the generateChoiceSet method generates model duplets using a similar random generation algorithm as the screen above.

7.7 Worldspace UI implementation

Unlike applications displayed on a regular screen, virtual reality applications cannot draw their user interface directly on top of the world in predefined locations relative to the screen. An alternative solution is therefore required. Usually and in the case of this application, this is handled by drawing the user interface on objects existing in the virtual reality world (or in "worldspace"). Luckily, the Unity user interface system supports this approach.

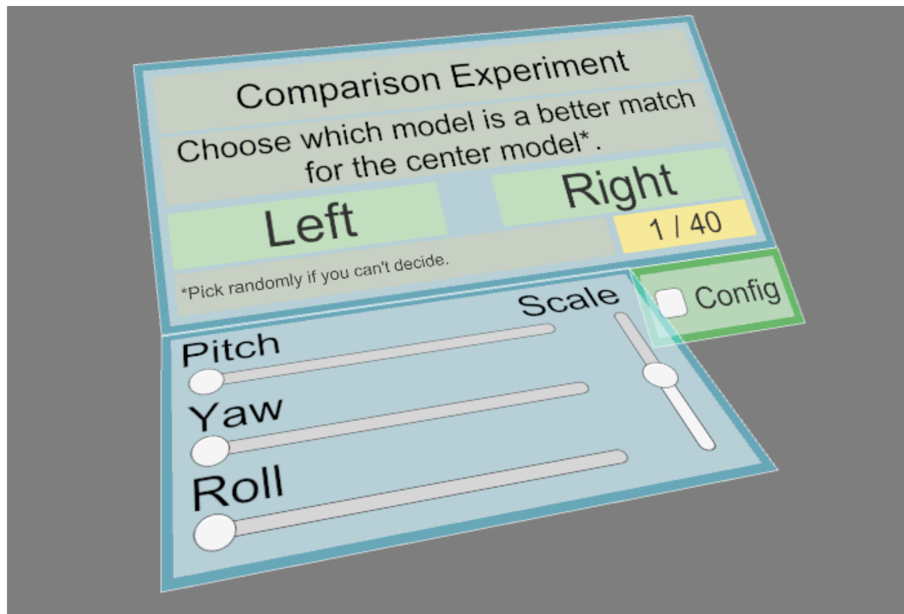

Figure 7.3: User interface example

7.7.1 Dashboards

The primary user interface of the application is the dashboard, a panel located directly in front and below the user that contains all information required for the specific screen currently active. Each dashboard class has a similarly named singleton game object that it is attached to. Only one dashboard can be active at a given time and all screens are directly associated with a specific kind of dashboard.

BaseDashboard This abstract class implements the common functionality of all dashboards, which all inherit from it. It has the virtual methods open and close, which when overridden implement any additional setup and removal code necessary by the inheriting dashboards. It also implements the methods setTopText, which sets the text on the title bar common to all dashboards, and the method setModelRotatorVisibility, which enables or disables the model rotation panel described further on.

Simple inheriting dashboards The dashboards SimpleDashboard, CommandlineDashboard, PersonalInfoDashboard, PlaybackDashboad implement the relatively simple functionality required by their associated screens, ranging from simple comfirmation buttons to a slider controlling the current position in a recording.

EditorDashboard More complicated than the other dashboard, this dashboard offers three panels that provide experiment definition editing functionality. The left panel offers buttons for saving, loading and creating experiment definition files. The central panel lists the screens in the currently loaded experiment definition file and allows for selecting specific screens to edit, as well as buttons that allow removal of individual screens. The right panel contains a list of screens, each with a corresponding button that adds a new screen of that type to the experiment definition being edited after the currently selected screen.

A fourth, context sensitive panel is displayed for selective screens, allowing for editing their screen-specific variables. For example, for a binary forced choice experiment screen, this panel displays a slider that determines the number of the tasks in the experiment, whenever the experiment allows for trick questions and if the tasks can contain the original model as one of the comparison options, followed by a listing of models used for the experiment and buttons that allow the addition of new ones using a file browser panel.

CommonExperimentDashboard Implements the common functionality between the two experiment-related dashboards, which is: displaying the progress of the task and a spinning loading icon when preloading is still in progress for the currently active model set.

MCOComparisonExperimentDashboard Has two buttons, one for the left model and one for the right model, and some clarification text.

MCORatingExperimentDashboard Has a rating slide with one button per number on the scale defined in the experiment definition file.

7.7.2 Model manipulation panel

The model manipulation is only displayed for screens that have models to manipulate. It has four sliders, one for each dimensional axis and one for scaling the model.

7.7.3 Configuration UI

Under the dashboard a configuration checkmark can be found. When enabled, a separate user interface opens to the right of the dashboard allowing configuration of the dashboards scale, position, and height. It is separate from the dashboard since in order to provide useful feedback, the changes are applied immediately, which would result in a feedback loop if it were attached to the dashboard.

7.8 Model display objects

In order to render the model meshes in worldspace, they must be attached to a gameObject with the MeshFilter and MeshRenderer components. In order to simplify their removal and (in case of meshes split into submeshes) generation, these components are actually attached to objects generated on demand that are then anchored to a parent object. These anchor objects always exist in the world and also contain the supporting components necessary to enable manipulating them. Primary amongst these is the GripRotatableModel component, which handles motion controller interaction and controls the rendering of interaction hints.

In order to eliminate differences in perception caused by different scales and rotations, all models share the same scale, with their rotation also being the same, only offset as to form a partial ring around the user's starting position.

Additional objects attached to the anchor object are a platform on the ground that represents the objects location even when there is no object displayed, a label that shows the loaded model file that is only displayed during debugging, and crucially, an object specific light source. This light source only affects the specific model that it is attached to (and is in fact the only light source that model is affected by) in order to provide identical lighting conditions for all the models.

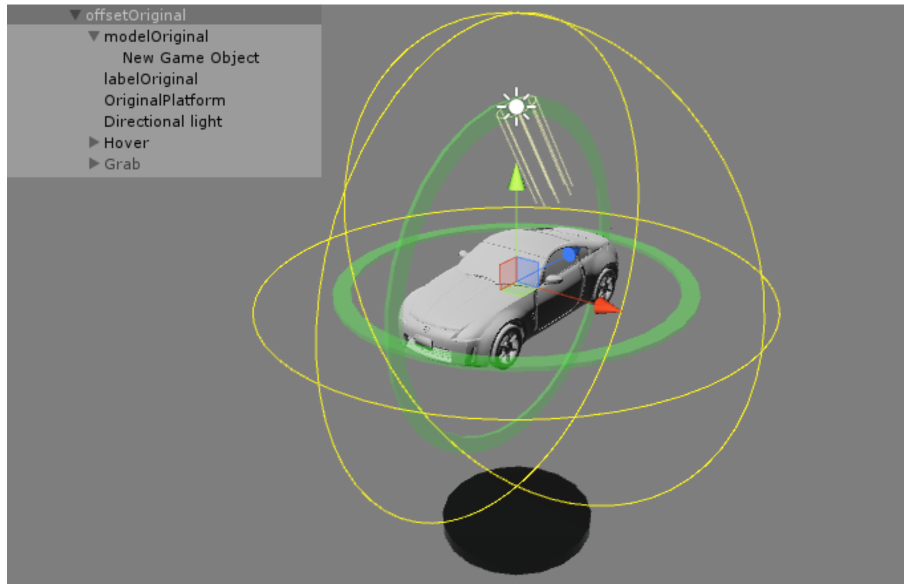

Figure 7.4: Model object hierarchy

7.9 Experiment file format

During application runtime, the experiment is represented by C# objects. The application is capable of both exporting and importing an experiment definition to XML files using C#'s System.Xml.Serialization namespace. The XML format is derived directly from its equivalent C# classes.

The XML file defines the experiment's parameters along with various utility data and all the screens contained withing the experiment. Figure 7.5 is a minimalistic example file. A file that contains all supported options and screens can be found in Attachment A.

7.9.1 Model file paths

In order to save model paths in at least partially environment-independent manner, they are stored as a class that inherits from the abstract class BaseMCODatapathReference (which class is used depends on what kind of path is being stored: a path to the original model, a path to one of its copies, or a path to a directory containing multiple model files) instead of a raw string. When an instance of this class is created from a raw string file path, it attempts to first generalize this path to a relative one.

```xml
<?xml version="1.0" encoding="utf-8"?>
<Experiment>
  <Screens>
    <TutorialScreen/>
    <PersonalInfoScreen/>
    <ComparingExperimentScreen>
      <TaskCount>40</TaskCount>
      <Randomize>true</Randomize>
      <IncludeOriginal>false</IncludeOriginal>
      <AllowTrickQuestions>true</AllowTrickQuestions>
      <Models>
        <ModelGroup>
          <Paths>
            <Directory>ExpData/bunny</Directory>
            <OriginalModel>
            ExpData/bunny/original.dat
            </OriginalModel>
          </Paths>
        </ModelGroup>
      </Models>
    </ComparingExperimentScreen>
    <CustomScreen>
      <Title>Thank you!</Title>
      <Text>You are done! Click to send results.</Text>
    </CustomScreen>
  </Screens>
  <Language>En</Language>
  <ReportEmail>targetmail@fictional.com</ReportEmail>
</Experiment>
```

Figure 7.5: Example experiment definition file

7.9.2 Experiment log

The format used by the experiment logs is very simple, with one entry per line.

Personal data choice Chosen ["male"|"female"]

Binary forced choice Rating [better model name] > [worse model name]

Anchored scoring rating Rating for [rated model name] is [rating value]

7.9.3 User movement recording

In order to record the experiment, the singleton Recorder takes a snapshot of the world state every fixed frame, specifically the position and orientation of the model display objects and the user's headset and controllers, as well as the models currently displayed by the model display objects. It also keeps a Dictionary of model files used. It stores the position and rotation data into a RecorderFrame struct which is then compressed and written to a temporary file. The format of this struct is as thus:

Positions and rotations are encoded into 48 bits (16 bits per coordinate).

- magic constant string "|F|"
- Headset position/rotation
- Left controller position/rotation
- Right controller position/rotation
- Model shared scale encoded as a single byte
- Model shared rotation
- Five bytes encoding the currently active models

Once the experiment is finished a new file is created at the location specified in the experiment definition file. The models used are written into this file first, followed by a copy of the temporary recording file, which is then deleted. Finally, the recording file is compressed along with the experiment log.

7.9.4 Result e-mail

To simplify data gathering and aid in development, the application supports sending experiment results using e-mail using the System.Net.Mail namespace. The application only attempts this if the experiment definition file contains a target e-mail address. As this requires a SMTP mail host, there are also fields in the experiment definition file for all necessary information: the host's address, port, e-mail account login and password. If these fields are omitted, the application defaults to a public G-mail account with a known password.

While this is an unsafe and poorly scaling solution, it is sufficient for development and small-scale purposes. In case it is necessary, the code is easily replaceable.

7.10 Abandoned features

The application also contains two features that, while functional, have been abandoned in favor of alternate solutions during development.

7.10.1 Editor

The application is capable of presenting an almost completely featured experiment definition editor. The only feature missing is the ability to enter text. While the editor was perfectly usable, ultimately it proved to be a slower and more difficult way of creating experiment definitions than simply editing them directly with a text editor. The editor is however still available and functional.

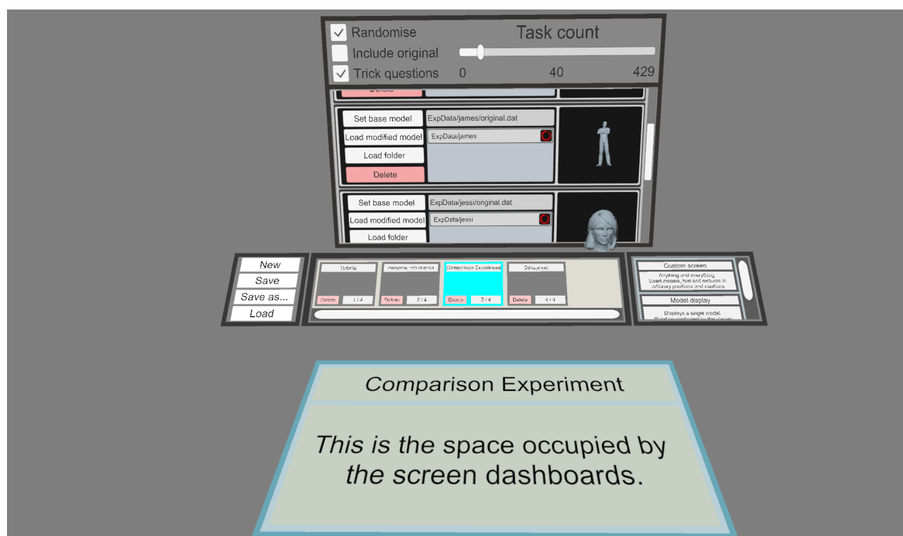

Figure 7.6: Experiment editor

7.10.2 Prop system

As a subsystem of the screen system, the application at one point supported arbitrarily-positioned screen-specific props. Four kinds of props were supported:

ImageProp This prop would have displayed the specified image.

TextProp This prop would have displayed the specified text.

ModelProp This prop would have displayed the specified model.

ButtonProp This prop would have enabled the user to select an arbitrarily-defined option that would then be logged to the result file.

In order to manipulate the props in the editor, they were suspended over a platform (which was hidden at runtime) that, when rightclicked on, displayed various manipulation handles.

While the prop system remains functional, there is no longer any way to add one short of manually entering the data into the experiment definition file directly. While this is possible, the definition is somewhat cumbersome and it is difficult to properly position the props without a reference point.

8 Pilot study

In order to validate the functionality of the application as well as to improve its usability, a pilot study was performed. A pilot study can be defined as a small-scale preliminary study conducted to determine possible issues and improvements before conducting a full-scale study. The advantage of a pilot study is that it can be conducted without significant investment of time and resources, with the results being used to improve the project before further use.

8.1 Study conduct

The study was performed on West Bohemia University premises on the 6th of April, 2017. There were three participants. Interaction with the test subjects related to the application was kept to a minimum in order to gather authentic data.

8.2 Form

The form for the study was created using Google Forms. It contained the following questions:

Performance - Frame drops Rated from 1 (best) to 5 (worst).

Performance - Model loading time Rated from 1 (best) to 5 (worst).

Performance - Overall framerate Rated from 1 (best) to 5 (worst).

Tutorial - Quality Rated from 1 (too short) to 5 (too long), with 3 being just right.

Tutorial - Topic coverage Rated from 1 (best) to 5 (worst).

Tutorial - Topic suggestion Text entry.

Model manipulation - Quality Rated from 1 (best) to 5 (worst).

Model manipulation - Method Rated from 1 (best) to 5 (worst).

Model manipulation - Suggestions Text entry.

User interface - Readability Rated from 1 (best) to 5 (worst).

User interface - Informativeness Rated from 1 (best) to 5 (worst).

Experiment - Attention span Rated from 1 (best) to 5 (worst).

Experiment - Nausea Rated from 1 (best) to 5 (worst).

Experiment - Clarity Rated from 1 (best) to 5 (worst).

Other suggestions Text entry.

8.3 Results

Results were as follows:

Question	1	2	3	4	5
1	2	1	0	0	0
2	3	0	0	0	0
3	1	2	0	0	0
4	0	2	1	0	0
5	2	1	0	0	0
7	0	2	1	0	0
8	1	2	0	0	0
10	2	1	0	0	0
11	2	1	0	0	0
12	1	1	1	0	0
13	2	1	0	0	0
14	2	1	0	0	0

Table 8.1: Pilot study results.

8.3.1 Changes enacted

The following changes were enacted based on the pilot study results as well as observations made during it:

Manipulation panel enlargement It was suggested by one of the participants to increase the size of the model manipulation panel. The panel was resized, with a corresponding increase of precision in model manipulation.

Tutorial clarification A slight issue with the tutorial was observed during the study. The last step that clarifies the experiment purpose was improved to further differentiate between the presented models and to highlight the correct choice.

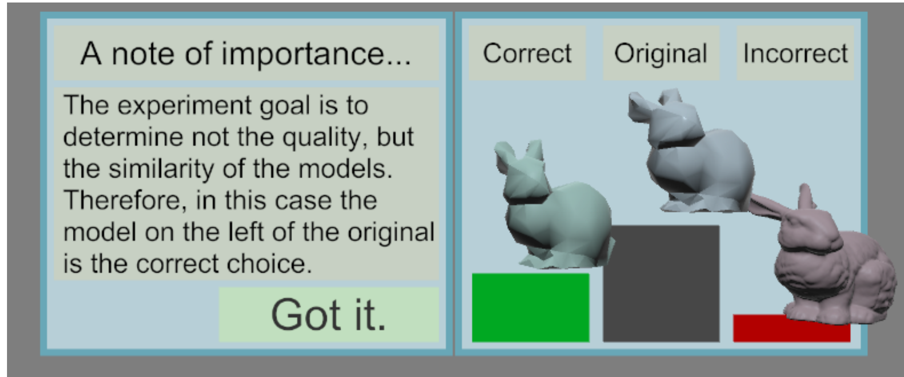

Figure 8.1: Clarified tutorial section

Manipulation code improvements Slight improvements were made to the grip model rotation code to fix some observed issues.

Grammar fixes A few grammar errors in the tutorial were corrected.

9 Application usage

This section describes user usage of the application.

9.1 Launch options

In order to facilitate user friendliness, the application can be launched in several ways by drag-n-dropping different filetypes onto it:

No file Launching the application directly will start the application using the experiment found in the file default.xml in the application's root directory.

.xml file Launching the application with a valid experiment definition file will run that experiment

.dat or .obj file A short experiment definition is generated and run, containing a commandline arguments screen and a model display screen keyed to the model supplied.

.rec file A short experiment definition is generated and run, containing a recording playback screen keyed to the recording supplied.

In order to ease debugging, the application saves a copy of any experiment it runs into the file latestRun.xml in the application's root folder upon launch. This copy will also contain empty XML tags for any option that was missing in the original file, since it is generated via serialization of the runtime experiment object as opposed to directly copying the input file.

9.2 Typical experiment example

The following text examines the typical user experience in detail.

9.2.1 Tutorial

First, the application runs each user through a short tutorial:

UI interaction This screen explains how to interact with the user interface, using either the motion controls or mouse pointer system depending on used virtual reality device. This section comes first primarily because it is necessary to proceed through the rest of the tutorial and experiment.

Manipulation - Rotation This section of the tutorial explains how to rotate displayed models, presenting both the control panel option and the motion controller grab option (or only the control panel if motion controllers are unavailable). For this section, the scaling controls of the manipulation panel are disabled.

Manipulation - Scaling This section explains how to change the size of the model using the control panel. For this section, the rotation controls of the manipulation panel are disabled.

Explanation of experiment goal Finally, the goal of the experiment is explained using example models in order to increase clarity.

9.2.2 Personal data collection

At this point, personal information about the user is collected. In the interest of simplicity and anonymity, this is limited to the minimal amount of information required: the subject's sex. This choice is made using a simple two-option panel.

9.2.3 Additional explanations

Any other supplemental information may be presented here using a collection of custom screens, such as expanded details on the experiment, model examples or listings. Usually the user will simply click a confirmation button in each of these screens.

9.2.4 Experiment

The experiment is run. The user is presented with randomized model triplets of a preset count and chooses between them (in the case of a binary forced choice experiment) or randomized model duplets of a preset count and rates their similarity on a predefined scale (in the case of an anchored scoring experiment). The user makes a choice using either the two-button panel or rating slide presented to them.

9.2.5 Finalization

After the last triplet or duplet, the user is thanked for his contribution. The application then saves the resulting experiment log and recording and closes. If a target email was specified in the experiment definition file, the application also copies the resulting files into a compressed archive and emails

it along with a timestamp of the experiment. If the attempt to send the email fails (for example because the computer's internet connection drops or because the email account data entered was incorrect), the application will crash after a timeout period. The experiment will however finalize properly and the compressed experiment package can still be found in the game directory and sent by other means.

10 Conclusion

The application performs well without any issues, significant or otherwise, with all required functionality implemented along with most of the optional functionality. The user interface, while not perfect, is comfortably usable with no immediately obvious improvements presenting themselves. Performance-wise, on all tested machines (which measured from state-of-the-art to barely meeting the standard virtual reality-capable device requirements) the application maintains a steady framerate with only occasional dips caused by model loading (which themselves only occur when the user rushes through multiple model sets in rapid succession).

Only a small number of non-developer testing was performed, so the possibility of missed bugs still exists. However, confidence remains high that any crippling issues have been eradicated. Any remaining issues should be of an annoying nature at worst.

An interesting facet of development was the rapidly changing nature of the field. During the two years invested into the work, several new virtual reality platforms were made available, and the software API layer was changed several times as a result, both from the developer side and from the hardware vendor side. Even Unity has changed its own virtual reality integration from an external plugin, to an official plugin, to a integrated part of the engine itself.

10.1 Future work

Some further iteration on the user interface system might prove beneficial. While the current system performs fine, there have been some voiced complaints regarding the initially unintuitive results of grabbing the models.

An undocumented but nonetheless present feature of the original MeshTest software was the ability to move the light source. A similar implementation would be trivial once the application would no longer require to maintain parity with the official feature set of MeshTest.

At present, the application supports two types of experiment. Other types could be implemented in order to broaden the possible experiments that could be run using the software.

Dynamic mesh support could be implemented in order to facilitate dynamic mesh algorithm data collection. This would probably require the addition of an animation control user interface, possibly containing a play

button and a slider similar to the recording playback screen. Given the amount of raw data contained in a polygonal-mesh defined animation, a more robust loading system might also be necessary.

Bibliography

Abrash, Michael (2014). *What VR Could, Should, and Almost Certainly Will Be within Two Years - Steam Dev Days 2014*. URL: http://media.steampowered.com/apps/abrashblog/Abrash%20Dev%20Days%202014.pdf (visited on 10/05/2017).

Bian, Zhe, Shi-Min Hu and Ralph R Martin (2009). "Evaluation for small visual difference between conforming meshes on strain field". In: *Journal of Computer Science and Technology* 24.1, pp. 65–75.

Bolin, Mark R and Gary W Meyer (1998). "A perceptually based adaptive sampling algorithm". In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, pp. 299–309.

Cignoni, Paolo, Claudio Montani and Roberto Scopigno (1998). "A comparison of mesh simplification algorithms". In: *Computers & Graphics* 22.1, pp. 37–54.

Daly, Scott J (1992). "Visible differences predictor: an algorithm for the assessment of image fidelity". In: *SPIE/IS&T 1992 Symposium on Electronic Imaging: Science and Technology*. International Society for Optics and Photonics, pp. 2–15.

Dumont, Reynald, Fabio Pellacini and James A Ferwerda (2003). "Perceptually-driven decision theory for interactive realistic rendering". In: *ACM Transactions on Graphics (TOG)* 22.2, pp. 152–181.

Ebrahimi, Touradj (2009). "Quality of multimedia experience: past, present and future". In: *MM" 09: Proceedings of the seventeen ACM international conference on Multimedia*. MMSPL-CONF-2010-003. ACM, pp. 3–4.

Fairchild, Mark D (2013). *Color appearance models*. John Wiley & Sons.

Ferwerda, James A, Ganesh Ramanarayanan et al. (2008). "Visual equivalence: an object-based approach to image quality". In: *Color and Imaging Conference*. Vol. 2008. 1. Society for Imaging Science and Technology, pp. 347–354.

Ferwerda, James A, Peter Shirley et al. (1997). "A model of visual masking for computer graphics". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., pp. 143–152.

Haas, John (2014). "A History of the Unity Game Engine". PhD thesis. WORCESTER POLYTECHNIC INSTITUTE.

Hoffman, David M et al. (2008). “Vergence–accommodation conflicts hinder visual performance and cause visual fatigue”. In: *Journal of vision* 8.3, pp. 33–33.

Howlett, Sarah, John Hamill and Carol O’Sullivan (2004). “An experimental approach to predicting saliency for simplified polygonal models”. In: *Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization*. ACM, pp. 57–64.

Huang, F., K. Chen and G. Wetzstein (2015). “The Light Field Stereoscope: Immersive Computer Graphics via Factored Near-Eye Light Field Displays with Focus Cues”. In: *ACM Trans. Graph. (SIGGRAPH)* 4 (34).

Karni, Zachi and Craig Gotsman (2000). “Spectral compression of mesh geometry”. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., pp. 279–286.

Kim, Sun-Jeong, Soo-Kyun Kim and Chang-Hun Kim (2002). “Discrete differential error metric for surface simplification”. In: *Computer Graphics and Applications, 2002. Proceedings. 10th Pacific Conference on*. IEEE, pp. 276–283.

Křivánek, Jaroslav, James A Ferwerda and Kavita Bala (2010). “Effects of global illumination approximations on material appearance”. In: *ACM Transactions on Graphics (TOG)*. Vol. 29. 4. ACM, p. 112.

Lanier, Jaron (1992). “Virtual reality: The promise of the future.” In: *Interactive Learning International* 8.4, pp. 275–79.

Larabi, Mohamed-chaker, Vincent Brodbeck and Christine Fernandez (2006). “A novel approach for constructing an achromatic contrast sensitivity function by matching”. In: *Image Processing, 2006 IEEE International Conference on*. IEEE, pp. 441–444.

Lavoué, Guillaume (2011). “A multiscale metric for 3D mesh visual quality assessment”. In: *Computer Graphics Forum*. Vol. 30. 5. Wiley Online Library, pp. 1427–1437.

Lavoue, Guillaume, Florence Denis and Florent Dupont (2007). “Subdivision surface watermarking”. In: *Computers & Graphics* 31.3, pp. 480–492.

Lee, Chang Ha, Amitabh Varshney and David W Jacobs (2005). “Mesh saliency”. In: *ACM transactions on graphics (TOG)*. Vol. 24. 3. ACM, pp. 659–666.

Lee, Haeyoung, Pierre Alliez and Mathieu Desbrun (2002). “Angle-Analyzer: A Triangle-Quad Mesh Codec”. In: *Computer Graphics Forum*. Vol. 21. 3. Wiley Online Library, pp. 383–392.

Lubin, Jeffrey (1995). “A visual discrimination model for imaging system design and evaluation”. In: *Vision models for target detection and recognition* 2, pp. 245–357.

Luebke, David P (2001). “A developer’s survey of polygonal simplification algorithms”. In: *IEEE Computer Graphics and Applications* 21.3, pp. 24–35.

Luebke, David and Benjamin Hallen (2001). “Perceptually driven simplification for interactive rendering”. In: *Rendering Techniques 2001*. Springer, pp. 223–234.

Mannos, James and David Sakrison (1974). “The effects of a visual fidelity criterion of the encoding of images”. In: *IEEE transactions on Information Theory* 20.4, pp. 525–536.

Movshon, J Anthony and Lynne Kiorpes (1988). “Analysis of the development of spatial contrast sensitivity in monkey and human infants”. In: *JOSA A* 5.12, pp. 2166–2172.

Myszkowski, Karol (1998). “The visible differences predictor: Applications to global illumination problems”. In: *Rendering Techniques’ 98*. Springer, pp. 223–236.

Pappas, Thrasyvoulos N, Robert J Safranek and Junqing Chen (2000). “Perceptual criteria for image quality evaluation”. In: *Handbook of image and video processing*, pp. 669–684.

Peng, Jingliang, Chang-Su Kim and C-C Jay Kuo (2005). “Technologies for 3D mesh compression: A survey”. In: *Journal of Visual Communication and Image Representation* 16.6, pp. 688–733.

Qu, Lijun and Gary W Meyer (2008). “Perceptually guided polygon reduction”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.5, pp. 1015–1029.

Ramasubramanian, Mahesh, Sumanta N Pattanaik and Donald P Greenberg (1999). “A perceptually based physical error metric for realistic image synthesis”. In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., pp. 73–82.

Reddy, Martin (1997). “Perceptually modulated level of detail for virtual environments”. In:

Rogowitz, Bernice E and Holly E Rushmeier (2001). “Are image quality metrics adequate to evaluate the quality of geometric objects?” In: *Photonics West 2001-Electronic Imaging*. International Society for Optics and Photonics, pp. 340–348.

Sorkine, Olga, Daniel Cohen-Or and Sivan Toledo (2003). "High-Pass Quantization for Mesh Encoding." In: *Symposium on Geometry Processing*. Vol. 42.

Váša, Libor and Jan Rus (2012). "Dihedral angle mesh error: a fast perception correlated distortion measure for fixed connectivity triangle meshes". In: *Computer Graphics Forum*. Vol. 31. 5. Wiley Online Library, pp. 1715–1724.

Wang, Zhou et al. (2004). "Image quality assessment: from error visibility to structural similarity". In: *IEEE transactions on image processing* 13.4, pp. 600–612.

Williams, Nathaniel et al. (2003). "Perceptually guided simplification of lit, textured meshes". In: *Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM, pp. 113–121.