Evaluation of an Appearance-Preserving Mesh Simplification Scheme for Configura AB

Rasmus Hedin

Supervisor : Harald Nautsch
Examiner : Ingemar Ragnemalm

LIU LINKÖPING
UNIVERSITY

Abstract

Abstract.tex

Acknowledgements

`acknowledgements.tex`

Contents

List of Figures

1 Introduction

For many years the field of computer graphics has been an important part in many industries, but especially in the entertainment industry (for instance video games and motion pictures). These industries generate a lot of money, and are quickly growing in size. A recent survey by *Kroon and Nilsson* [9] from *Dataspelsbranschen* have shown that the video games industry in Sweden generated €1325 M in revenue in 2016, a steep increase from the €392 M in 2012. Also, most movies nowadays use to some extent 3-D computer graphics in scenes where the cost would be to large to reproduce in reality, be too risky for actors, or simply be impossible.

The rendering of meshes (a collection of polygons describing a surface) is one of the main activities in computer graphics (usually, a collection of meshes; a so called scene description). In many cases, these meshes are very detailed, and require a large amount of polygons to fully describe a surface. This is problematic, since the rendering time of a scene depends on the number of polygons it has. Therefore, it is important to reduce the number of polygons in a mesh as much as possible. This is especially true in video games, where the scene needs to be rendered in real-time. However, if the number of polygons are reduced too much, it will degrade the visual quality of a mesh, giving a progressively flatter surface than intended and removing small surface details. This destroys the intended *geometrical appearance* of the mesh.

1.1 Motivation

While the geometrical appearance of a mesh is important, it is not the only factor which gives the final appearance of a mesh when rendering. According to *Cohen et al.* [1], both the surface curvature and color are equally as important contributors. *Textured appearance* will be used as the common name for these since surface properties are usually specified with a texture map.

In computer graphics, the process to reduce the number of polygons in a mesh based on some metric is called a *mesh simplification algorithm*, as seen in *Talton's survey* [16] in the field. Historically, these have been mostly concerned with minimizing the geometrical deviation of a mesh when applying it. Somewhat recently, methods for minimizing the texture deviation when simplifying a mesh have also appeared. They attempt to reduce the texture deviation and stretching caused when removing polygons from a mesh, as described in *Hoppe et al.* [6].

By simultaneously taking into account the geometrical and texture deviation, one can preserve the *visual appearance* of a mesh when simplifying it. If polygons can be removed without affecting this appearance significantly, the rendering time can be reduced for "free".

1.2 Aim

The aim of this thesis is to first perform a literature study of mesh simplification algorithms that preserve the visual appearance of a mesh. A suitable solution will then integrated as a preprocessing step in *Configura's* (see Section 1.5) graphics pipeline. This will enable Configura to generate better *Level of Detail* (LoD) meshes for speeding up their rendering time. Currently, Configura only takes into account the geometrical deviation when simplifying, with no regard for the textures (e.g. diffuse or normal) on top of the mesh.

1.3 Research Questions

1. What alternative *mesh simplification schemes* exist that *preserves the appearance* of a mesh?

2. Which of these alternatives would be appropriate to integrate into Configura's software?

1.4 Delimitations

Since the thesis is done on a time limit a comparison of multiple mesh simplification schemes taking the appearance into account is not feasible. Therefore, one mesh simplification algorithm will be chosen to be implemented and evaluated. The choice will be based on a study of algorithms that can be found in the literature.

1.5 Background

This thesis was requested by Configura AB, a company in Linköping which provides space planning software. Their main product, *CET designer*, lets companies plan, create and render 3-D office spaces (among other things). These scenes can have a large amount of polygons that need to be rendered in real-time for customers to evaluate their creations in CET designer.

To allow larger scenes to be rendered with higher frame-rates (e.g. needed when exploring environments in *Virtual Reality* (VR), to prevent motion sickness), it would be beneficial to reduce the amount of polygons as much as possible. The meshes in these scenes usually have textures applied to them, and it is therefore important to keep the quality as high as possible.

While Configura already has a mesh simplification in their pipeline, it only accounts for surface simplifications, and does not take into account the texture appearance that might be degraded when applying mesh simplification. Hence, the given task was to integrate a new mesh simplification scheme that takes into account texture quality when simplifying a mesh.

2 Theory

Since several mesh simplification algorithms are being considered, Section 2.1 gives a brief overview of the most notable schemes found in previous work. A more detailed explanation of the algorithms can be found in Sections 2.2 to 2.4.

Afterwards, in Section 2.7, the different metrics that can be used to measure the appearance preservation after a simplification has been done is discussed. This will later be used to evaluate the solution empirically by giving a concrete metric for the amount of appearance deviation.

Finally, in Section 2.8, the methods and common practices for measuring performance of an algorithm are discussed. Based on existing industry practices, we show how to measure the computation time and memory usage of the algorithms. Since these measurements can be noisy, statistical methods will need to be used to truthfully answer our research questions.

2.1 Related Work

Some different approaches have been presented in the literature to solve the problem of simplifying a mesh. Early solutions focused on the geometrical error which is enough in many cases. But in the case of a mesh with appearance attributes this may give a poor result. Other solutions have been presented that takes the attributes into account to give a better appearance of the mesh. This section will give a brief overview of the simplification algorithms that can be found in the literature.

According to *David Luebke's survey* [12] on the subject, mesh simplification is a technique which transforms a geometrically complex mesh (with a lot of polygons) to a simpler representation by removing unimportant geometrical details (reducing the number of polygons). It does this by assuming that some meshes are small, distant, or have areas which are unimportant to the visual quality of the rendered image. For example, if the camera in a scene always faces towards a certain direction, then removing details from the backside of a mesh won't affect the final rendered result, since they will never be seen by the camera anyway. Reducing the number of polygons allows meshes to use less storage space and need less computation time.

There are many mesh simplification algorithms, as can be seen in *David Luebke's survey* [12], each presenting a new approach with their own strengths and weaknesses. The first scheme is due to *Schroeder et al.* [15] in 1992, called *mesh decimation*. It was meant to be used to simplify meshes produced by the marching cubes algorithm, which usually gives unneces-

sary details. It works by making multiple passes through the meshes' vertices, and deleting vertices that do not destroy local topology and are within a given distance threshold when re-triangulated.

Besides *decimation-based methods*, such as the aforementioned mesh decimation scheme, there exists another class of simplifiers based on *vertex-merging mechanisms*. According to *Luebke* [12], these work by iteratively collapsing a pair of vertices (v_j, v_k) into a single vertex v_i. This will also remove any polygons which were suspended by (v_j, v_k). The first collapse scheme is due to *Hoppe et al.* [7], which shows an *edge collapse* of $e_{ji} = (v_j, v_i) \rightarrow v_i$ as in Fig. 2.1 (a). There exist other schemes, such as *pair contraction* in Fig. 2.1 (c) where vertices within a distance t are allowed to be merged. These don't tend to preserve the local topology of the original mesh, and instead focus on a more aggressive simplification.

(a)   (b)   (c)
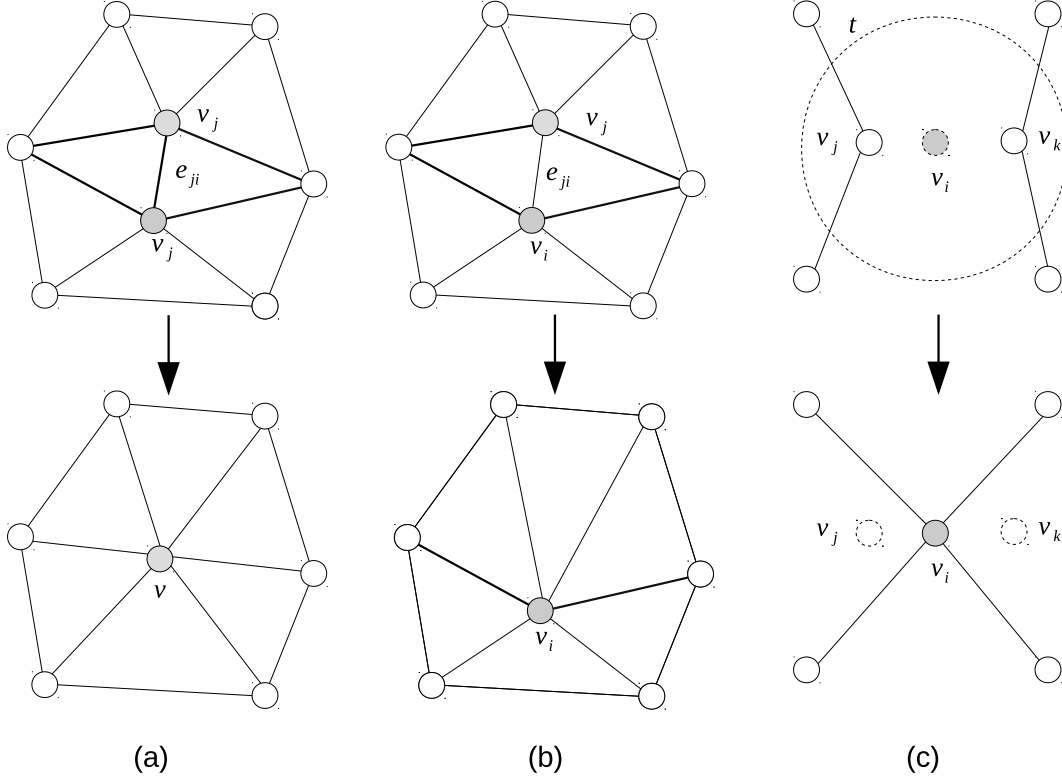
Figure 2.1: (a) edge collapse, (b) vertex removal, and (c) pair contraction

Quadric Error Metrics (QEM), due to *Garland and Heckbert* [3] performs edge collapses to provide a provably optimal simplification. In each iteration, an edge is collapsed $(v_j, v_i) \rightarrow \mathbf{v}$ and $\mathbf{v}$ is re-positoned at the position which gives the lowest possible geometrical error. *Hoppe* [6] also perform edge collapses but he tries to minimize an energy function. The edge with the lowest estimated energy is chosen for the collapse.

2.1.1 Preserving appearance

Focusing on minimizing the geometrical error during simplification works well for many meshes. But in the case of a mesh with appearance attributes such as color, normals, and texture coordinates the result may be poor. A common way to solve this is to use a metric which does not only take the geometry, but also the appearance attributes into account.

Cohen et al. [1] defines a new *texture deviation metric* which takes three attributes into account: Surface position, surface curvature, and surface color. These attributes are sampled from the input surface and the simplification is done with edge collapses and vertex removals.

Sander et al. [13] uses the texture deviation metric together with a *texture stretch metric* to better balance frequency content in every direction all over the surface.

An extended more general version of the QEM is presented by *Garland and Heckbert* [2] where the metric can be used for points in n-dimensional space. Thus, when the color is considered each vertex would be represented by a 6-dimensional vector. Another version of QEM by *Hoppe* [5] base the attribute error on geometric correspondance in 3D rather than using points in n-dimensional space.

Image-driven simplification defined by *Lindstrom and Turk* [11] captures images from different angles of the mesh. The distance between images of the mesh before and after an edge collapse are used to guide the simplification.

2.2 Appearance-Preserving Simplification

In order to preserve the appearance of a model when it is simplified, *Cohen et al.* [1] defines a *texture deviation metric*. This metric takes three attributes into account: Surface position, surface curvature, and surface color. To properly sample these attributes from the input surface, the surface position is decoupled from the color and normals stored in texture and normal maps, respectively. The metric guarantees that the maps will not shift more than a user-specified number of pixels on the screen. This user-specified number is defined as ϵ.

Approximation of the surface position is done offline with simplification operations such as edge collapsing and vertex removals. At run-time, the color and normals are sampled in pixel-coordinates with mip-mapping techniques. Mip-maps are a pre-calculated sequence of images with progressively lower resolution. Here the decoupled representation is useful since the texture deviation metric can be used to bound how much the mapped attributes value's positions deviate from the positions of the original mesh. This guarantees that the sampling and mapping to screen-space of the attributes is done in an appropriate way.

Before any simplification can be made, a parametrization of the surface is required in order to store the color and normals in maps. If the input mesh does not have a parametrization, it is created and stored per-vertex in texture and normal maps. Next, the surface and maps are fed into the simplification algorithm which decides which simplification operations to use and in what order. The deviation caused by each operation is measured with the texture deviation metric. A *progressive mesh* (PM) with error bounds for each operation is returned by the algorithm, which can then be used to create a set of LOD with error bounds. Using the error bounds, the tessellation of the model can be adjusted to meet the user-specified error ϵ.

2.3 Quadric-Based Error Metric

Mesh simplification with *Quadric Error Metrics* (QEM), due to *Garland and Heckbert* [3], is based on the vertex merging paradigm. It provides a provably optimal simplification in each iteration, by collapsing the edge $(v_j, v_i) \rightarrow v_i$ and re-positioning it at $\mathbf{v}$, which gives it the lowest possible geometrical error. By assigning a matrix $\mathbf{Q}_i$ for each vertex v_i, one can find the error $\Delta(\mathbf{v})$ introduced by moving v_i to $\mathbf{v}$. $\Delta(\mathbf{v})$ is the sum of squared distances from $\mathbf{v}$ to the planes $\mathbf{f}_k$ in v_i's neighborhood N_i (all polygons around v_i as in Fig. 2.2).

Since $\Delta(\mathbf{v})$ is quadratic, finding a $\mathbf{v}$ which gives a minimal error is a linear problem. The best position $\bar{\mathbf{v}}_i$ for v_i after a collapse $(v_j, v_i) \rightarrow v_i$ is a solution to Eq. (2.1).

$$(\mathbf{Q}_j + \mathbf{Q}_i)\bar{\mathbf{v}}_i = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^\top. \tag{2.1}$$
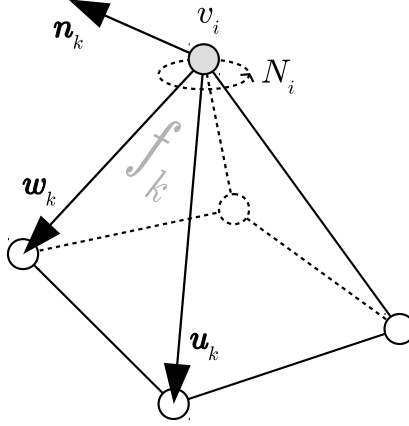
Figure 2.2: depiction of one of the planes f_k in the neighborhood N_i of the vertex v_i. It has a normal $\mathbf{n}_k$; found by the $\mathbf{w}_k \times \mathbf{u}_k$ of its edges.

$$\Delta(\mathbf{v}) = \mathbf{v}^\top \mathbf{Q}_i \mathbf{v} \ , \quad \mathbf{Q}_i = \sum_{f_k \in N_i} \mathbf{f_k f_k}^\top \ . \tag{2.2}$$

By storing the $\Delta(\bar{\mathbf{v}}_i)$ for every valid collapse $(v_j, v_i) \to v_i$ in a min-heap, the least cost collapse on the top of the heap can be done in each iteration, removing a vertex in each step. This is repeated until either a user-given vertex count $|\mathcal{V}|$ is reached or until some error threshold ϵ.

The results by *Garland and Heckbert* [3] show that QEM can reduce the simplification error by up to 50 % in comparison to a naïve scheme where $\bar{\mathbf{v}}_i = (v_i + v_j) \div 2$ and $\Delta(\mathbf{v}) = \|\mathbf{v} - v_i\|$, as can be seen in Fig. 2.3. They also argue that QEM gives higher-quality simplifications than *vertex clustering* and that it is faster than *progressive meshing* (which we also present later).

Figure 2.3: simplification using a naïve (top image) and a quadric error metric (bottom image) at different levels of detail at some vertex count (left to right: 50 %, 35 % and 17 % of original).

2.3.1 Preserving appearance

A general version of QEM was later presented by Garland and Heckbert [2] where vertices can be placed in n-dimensional space. This makes it possible to, for example, include the color of

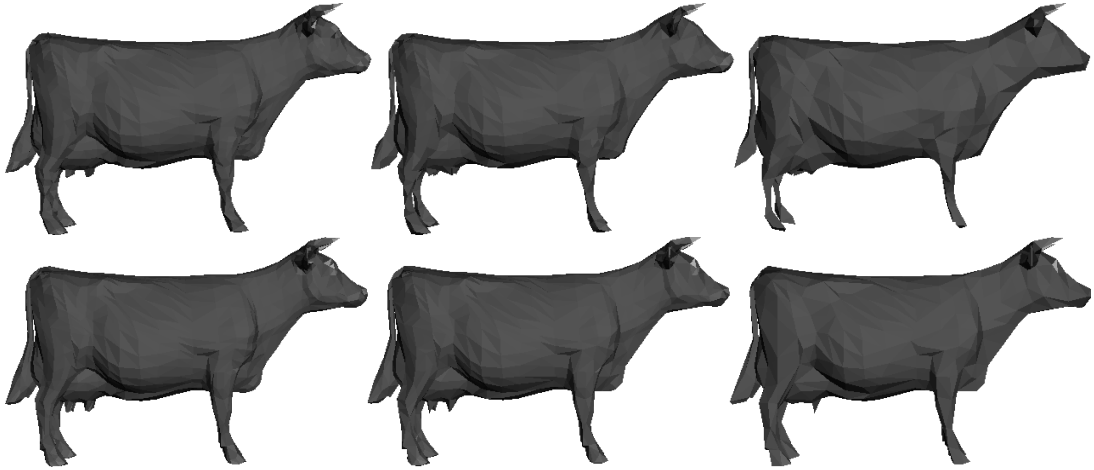the surface in the computation. Each vertex is treated as a vector $v \subset \mathbb{R}^n$. Thus, when the color is considered each vertex will be represented by a 6-dimensional vector $v = [x, y, z, r, g, b]^\top$. The first three values is the spatial coordinates and the last three values will be the color. The same thing can be done with for example texture coordinates where each vertex would be represented by a 5-dimensional vector $v = [x, y, z, s, t]$ where s, t is the 2D texture coordinates.

The original version of QEM used a 4x4 homogeneous matrix Q_i for the computations. A more convenient notation is used in the general version. A face in the original model defines a plane which satisfies the equation $n^\top v + d = 0$ where n is the face normal and d is a scalar. The squared distance of a vertex to a plane is given by

$$D^2 = (\mathbf{n}^\top \mathbf{v} + d)^2 = \mathbf{v}^\top (\mathbf{n}\mathbf{n}^\top)\mathbf{v} + 2d\mathbf{n}^\top \mathbf{v} + d^2 \tag{2.3}$$

D^2 can now be represented as the quadric Q

$$Q = (\mathbf{A}, \mathbf{b}, c) = (\mathbf{n}\mathbf{n}^\top, d\mathbf{n}, d^2) \tag{2.4}$$

$$Q(\mathbf{v}) = \mathbf{v}^\top \mathbf{A}\mathbf{v} + 2\mathbf{b}\mathbf{v} + c \tag{2.5}$$

where $Q(\mathbf{v})$ is the sum of squared distances. This representation performs matrix operations on 3x3 matrices instead of 4x4 as in the previous notation. This increases performance when, for example, performing matrix inversions.

The overhead of the higher dimensional quadrics is not extreme according to Garland and Heckbert. However, when using this with colors, normals, and texture coordinates some caution is needed. Colors may need to be clamped and normals needs to be normalized to unit length.

Garland and Heckbert have assumed that the properties vary continiously over the whole surface. However, if we want to for example apply a texture to a cylinder there will always be a seam where the ends of the texture meets. All vertices along this seam needs to be duplicated since they would require two different texture coordinates. The authors suggested having boundary constraints that would maintain the seam. However, in the case where a mesh have a corresponding texture atlas where each face have a specific part of the texture this might not work. The solution suggested by the authors is to allow multiple quadrics for each vertex, but, this is not yet implemented.

Hugues Hoppe [5] also uses QEM for meshes with appearance attributes. Instead of calculating the distances to hyperplanes as Garland and Heckbert, Hoppe base the attribute error on geometric correspondence in 3D. A point p is projected onto a face in $\mathbb{R}^3$ rather than a plane in a higher dimension and then both the geometric and attribute error is computed. According to the author this gives a better result compared to Garland and Heckberts general QEM.

2.4 Progressive Meshes

Hugues Hoppe [6] introduced the *Progressive Mesh* (PM) representation as a scheme for storing and transmitting arbitrary polygon meshes. An abitrary mesh $\hat{M}$ in PM form is defined by a sequence of meshes $M^0, M^1, ..., M^n$ with increasing accuracy of the original mesh $\hat{M} = M^n$. Only the most coarse mesh M^0 is stored together with records of *vertex splits* that is used to refine M^0 into the more detailed meshes. A vertex split will transform M^i the more detailed mesh M^{i+1} and an edge collapse transforms M^i to the coarser mesh M^{i-1}.

To construct a PM the edge collapses of the original mesh needs to be determined. Multiple possible algorithims for chosing those edge collapses is presented by Hoppe [6], some with high speed but low accuracy and some more accurate but with lower speed. A fast but maybe not so accurate strategy is to choose the edge collapses at random but with some conditions. Another more accurate scheme is to use heursitics. According to Hoppe, the construction of a

PM is supposed to be a preprocess, Therefore, the author chose an algorithm that take some time but is more accurate.

Hoppe based the simplification on the previous work *Mesh Optimizaton* [7] where the goal is to find a mesh that fits a sets X of points $x_i \subset \mathbb{R}^3$ with a small number of vertices. This problem is defined as a minimization of the energy function.

$$E(M) = E_{dist}(M) + E_{spring}(M) + E_{scalar}(M) + E_{disc}(M)n \qquad (2.6)$$

Distance energy $E_{dist}(M)$ measures the sum of the squared distances from the points to the mesh. *Spring energy* E_{spring} is used to regularize the optimization problem. *Scalar energy* E_{scalar} measures the accuracy of the scalar attributes of the mesh. The last term, E_{disc} measures the geometric accuracy of the discontinuity curves (e.g creases). All legal edge collapses is placed in a priority queue where the edge collapse with lowest ΔE (estimated energy) is on the top of the queue. After a transformation is performed, the energy of the neighboring edges is updated.

2.4.1 Preserving appearance

Given an arbitrary mesh, *Sander et. al* [13] presents a method to construct a PM where a texture parametrization is shared between all meshes in the PM sequence. In order to create a texture mapping for a simplified mesh, the original mesh's attributes, e.g normals, is sampled. This method was developed with two goals taken into consideration:

- Minimize *texture stretch*: When a mesh is simplified the texture may be stretched in some areas which decrease the quality of the appearance. Since the texture parametrization determines the sampling density, a balanced parametrization is preferred over one that samples with different density in different areas. The balanced parametrization is obtained by minimizing the largest texture stretch over all points in the domain. No point in the domain will therefore be too stretched and thus making no point undersampled.

- Minimize *texture deviation*: Conventional methods use geometric error for the mesh simplification. According to the authors this is not appropriate when a mesh is textured. The stricter texture deviation error metric, where the geometric error is measured according to the parametrization, is more appropriate. This is the metric by *Cohen et al.* [1] explained in Section 2.2. By plotting a graph of the texture deviation vs the number of faces, the goal is to minimize the height of this graph.

Cohen et al. [1] stored an error bound for each vertex in a PM. *Sander et al.* [13] instead tries to find an atlas parametrization that minimizes both texture deviation and texture stretch for all meshes in the PM.

2.5 Mesh parametrerization

In order to apply, for example, a texture to a mesh each vertex is given a texture coordinate. This problem of finding a mapping between a surface (mesh)and a parameter domain (texture map) is called parameterization. Given a triangular mesh Hormann et al. [8] refers to the mapping problem as mesh parameterization. According to Hormann et al. there exists a one-to-one mapping between two surfaces with similar topology. Thus, a surface that is homeomorphic to a disk can be mapped onto a plane, i.e. a texture. If a mesh is not homeomorphic to a disk it has to be split into parts which are homeomorphic to a disk. These parts can then be put onto the same plane and only one texture is needed.

2.6 Multiple attributes for a vertex

A vertex of mesh often have a property associated with it such as texture coordinates, color, and normal. A common way of texturing a model is to use a texture atlas where each triangle of the mesh is assigned a specific part of the texture. Since a texture is in a 2-dimensional domain a mesh parameterization needs to be performed. In many cases the mesh needs to be cut somewhere which will create seams. The vertices along this seam would require two sets of texture coordinates and this will create a discontinuity. In Fig. 2.4 a cylinder is unwrapped to 2D which creates a seam. All vertices along the seam will have two sets of texture coordinates (s, t), one with $s = 0$ and a second with $s = 1$.
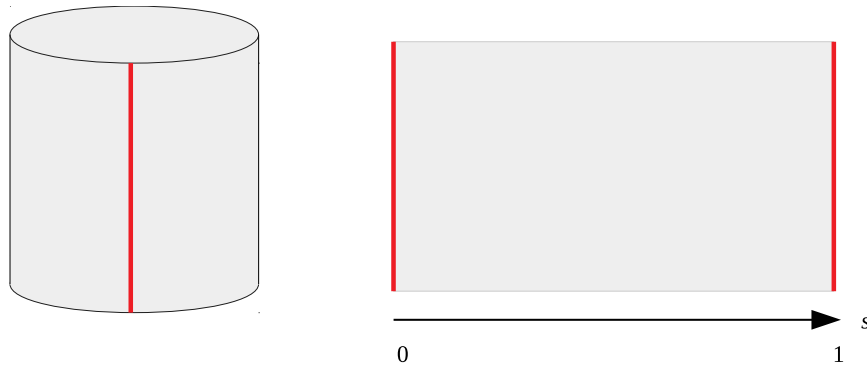

Figure 2.4: Unwrapping a cylinder. Vertices along seam (red line) require two texture coordinates

Hugues Hoppe [4] presents a mesh representation where each face adjacent to a vertex can have different appearance attributes. The attribute values is associated with the corners of the faces instead of the vertices. A *corner* is defined by Hoppe as a tuple $< vertex, face >$. The attribute values at a corner defines the value that should be used for face f at vertex v. Corners with the same attributes and adjacent vertex are stored in a *wedge*. The wedge has one or more corners and each vertex will be divided into one or more wedges. This representation is useful for simplification of meshes with discontinuities in their attribute fields. Discontinuities could result in a simplified mesh where adjacent faces have been separated which introduces new holes in the mesh.
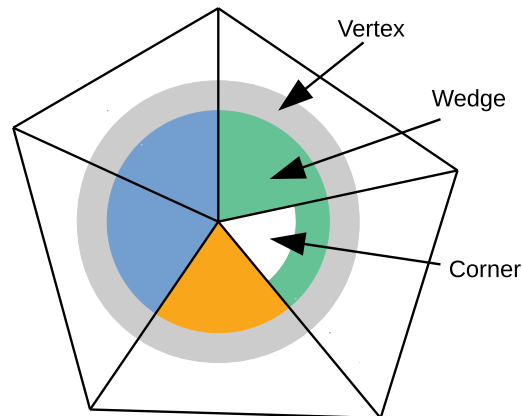

Figure 2.5: Vertex with wedges

2.7 Metrics for Appearance Preservation

Previously in Section 2.2 and Section 2.4.1, the metrics texture deviation and texture stretch have been defined. But to measure more exactly how much the visual appearance of a simplified mesh deviate from the original mesh another metric would be better. *Lindstrom and Turk* [11] defines *image-driven simplification* which captures images from different angles of the mesh. The difference between the images of the original and simplified mesh are computed in order to measure how well the appearance is preserved. This metric is more general and can be applied to all simplification algorithms since it only compares the original mesh to the simplified mesh.

The *image metric* is defined as a function taking two images and gives the distance between them. To measure the distance the authors use root mean square of the luminance values of two images Y^0 and Y^1 with dimensions $m \times n$ pixels. It is defined as:

$$d_{RMS}(Y^0, Y^1) = \sqrt{\frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij}^0 - y_{ij}^1)^2} \tag{2.7}$$

To evaluate the quality of the simplified mesh the authors capture images from 24 different camera positions. The positions are defined as the vertices of a rhombicuboctahedron which can be seen in Fig. 2.6. Two sets of l images $Y^0 = Y_h^0$ and $Y^1 = Y_h^1$ with dimensions $m \times n$ is rendered and the RMS is then computed as:

$$d_{RMS}(Y^0, Y^1) = \sqrt{\frac{1}{lmn} \sum_{h=1}^{l} \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{hij}^0 - y_{hij}^1)^2} \tag{2.8}$$
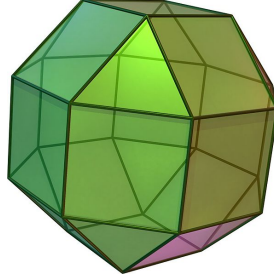

Figure 2.6: Rhombicuboctahedron with 24 vertices which is used as the camera positions. (Rhombicuboctahedron by Hellisp / CC BY 3.0)

2.8 Measuring Algorithmic Performance

According to *David Lilja* [10, p. 4], there are three fundamental techniques that can be used when confronted with a performance-analysis problem: *measurement*, *simulation* or *modeling*. While the book concentrates on evaluating computer performance, these techniques can also be applied when evaluating different algorithms. Measurement would be to actually execute the implemented algorithm and simultaneously gather interesting statistics (e.g. how long it took to finish and how much memory was needed), and use this to compare the algorithms. While modeling would be to analytically derive an abstract model for the algorithm (e.g. the Big $\mathcal{O}$ worst-case running time and memory), and see which of them has a lower complexity.

Since not all of the algorithms in Section 2.1 have an analytical model derived by the authors, and also because the algorithms are to be evaluated in a real system, only the problems inherent to the measurement approach will be considered. One of the problems with doing measurements of a real system (a program running on a computer in this case), according to *David Lilja* [10, p. 43], is that they introduce noise. This noise needs to be modeled to be able

10

to reach a correct conclusion, such as determining if algorithm A is faster than algorithm B. One way of doing this, according to *David Lilja* [10, p. 48], is to find the confidence interval of the measured value, by assuming the source's error is distributed according to some statistical distribution (like the Gaussian or the Student t-distribution). The confidence interval $[a, b]$ when assuming the source's error is t-distributed, can be found as shown below. Where n tests are taken (giving $n - 1$ degrees of freedom), with a significance level of α (usually 5 %).

$$a = \bar{x} - t_k \frac{s}{\sqrt{n}}, \quad b = \bar{x} + t_k \frac{s}{\sqrt{n}}, \quad t_k = t_{1-\alpha/2, n-1} \tag{2.9}$$

One common mistake, according to *Schenker et al.* [14], when using confidence intervals to determine if e.g. an implemented algorithm A is faster than B, is the use of the overlapping method to reach conclusions. If two confidence intervals *do not* overlap, then the result is provably significant (that is, algorithm A is either faster or slower than B). However, the converse is not true, if two intervals *do* overlap, then no conclusions can be reached since the result could be either significant or not significant.

3 Method

TODO: Overview of method chapter.

3.1 Implementation

3.2 Evaluation

4 Results

TODO: Overview of results chapter.

5 Discussion

TODO: Overview of discussion chapter

6 Conclusion

TODO: Overview of conclusion chapter.

Bibliography

[1] Jonathan Cohen, Marc Olano, and Dinesh Manocha. "Appearance-preserving simplification". In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM. 1998, pp. 115–122.

[2] Michael Garland and Paul S Heckbert. "Simplifying surfaces with color and texture using quadric error metrics". In: *Visualization'98. Proceedings*. IEEE. 1998, pp. 263–269.

[3] Michael Garland and Paul S Heckbert. "Surface simplification using quadric error metrics". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1997, pp. 209–216.

[4] Hugues Hoppe. "Efficient implementation of progressive meshes". In: *Computers & Graphics* 22.1 (1998), pp. 27–36.

[5] Hugues Hoppe. "New Quadric Metric for Simplifiying Meshes with Appearance Attributes". In: *Proceedings of the Conference on Visualization '99: Celebrating Ten Years*. VIS '99. San Francisco, California, USA: IEEE Computer Society Press, 1999, pp. 59–66. ISBN: 0-7803-5897-X. URL: http://dl.acm.org/citation.cfm?id=319351.319357.

[6] Hugues Hoppe. "Progressive meshes". In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM. 1996, pp. 99–108.

[7] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. "Mesh optimization". In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. ACM. 1993, pp. 19–26.

[8] Kai Hormann, Bruno Lévy, and Alla Sheffer. "Mesh parameterization: Theory and practice". In: (2007).

[9] Jacob Kroon and Anna Nilsson. "Game developer index 2017". In: *Based on 2017 Annual Reports. Dataspelbranchen* (2017).

[10] David J Lilja. *Measuring computer performance: a practitioner's guide*. Cambridge University Press, 2005.

[11] Peter Lindstrom and Greg Turk. "Image-driven simplification". In: *ACM Transactions on Graphics (ToG)* 19.3 (2000), pp. 204–241.

[12] David P Luebke. "A developer's survey of polygonal simplification algorithms". In: *IEEE Computer Graphics and Applications* 21.3 (2001), pp. 24–35.

[13] Pedro V Sander, John Snyder, Steven J Gortler, and Hugues Hoppe. "Texture mapping progressive meshes". In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, pp. 409–416.

[14] Nathaniel Schenker and Jane F Gentleman. "On judging the significance of differences by examining the overlap between confidence intervals". In: *The American Statistician* 55.3 (2001), pp. 182–186.

[15] William J Schroeder, Jonathan A Zarge, and William E Lorensen. "Decimation of triangle meshes". In: *ACM Siggraph Computer Graphics*. Vol. 26. 2. ACM. 1992, pp. 65–70.

[16] Jerry O Talton. "A short survey of mesh simplification algorithms". In: *University of Illinois at Urbana-Champaign* (2004).