Mesh Simplification

BACHELORARBEIT 1

Student Christian Mayr, 0910601022
Betreuer Dr Simon Ginzinger

Salzburg, 23. Jänner 2012

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, und keine anderen als die angegeben Quellen und Hilfsmittel benutzt habe. Weiters versichere ich hiermit, dass ich die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission weder im In- noch im Ausland vorgelegt und auch nicht veröffentlicht.

Datum Unterschrift

Kurzfassung

Vor- und Zuname:	Christian MAYR
Institution:	FH Salzburg
Studiengang:	Bachelor MultiMediaTechnology
Titel der Bachelorarbeit:	Mesh Simplification
Begutachter:	Dr Simon Ginzinger

In Computerspielen müssen dem Detailreichtum von 3D Modellen Grenzen gesetzt werden, um die Anforderungen an die Hardware im moderaten Rahmen zu halten. Wird dem Benutzer oder der Benutzerin die Möglichkeit eröffnet, selbst erstellte 3D Modelle in das Spiel zu integrieren, muss die Anzahl der Polygone kontrolliert und eventuell reduziert werden, um ein flüssiges Spielerlebnis gewährleisten zu können. Vor allem in 3D Online-Spielen ist dieses Verhalten wichtig, weil die 3D Modelle vom Server heruntergeladen und von Clients mit verschiedensten Hardwarekonfigurationen gerendert werden müssen. Unnötige Komplexität kann hier zu Performanceeinbrüchen und zu großem Datentransfer führen.

Als Lösung bieten sich Algorithmen an, die den Anwendern oder den Anwenderinnen ermöglichen, ihre 3D Modelle selbstständig bis zu einem gewünschten Detailgrad vereinfachen zu lassen. Diese Algorithmen verfolgen verschiedene Ansätze und sind ein seit vielen Jahren gut erforschtes Gebiet.

Hier vergleichen wir verschiedene Lösungsansätze um den Geeignetsten für diesen Anwendungsfall zu identifizieren. Im Rahmen dieses Vergleiches kamen wir zu dem Schluss, dass sich Algorithmen, die auf Vertex Pair Contraction basieren, als eine der Geeignetsten für den vorhin genannten Anwendungsfall erweisen. Diese Arbeit wird diese Entscheidung begründen, indem sie relevante Algorithmen theoretisch und praktisch miteinander vergleicht und deren Vor- und Nachteile in Bezug auf den definierten Anwendungsfall abwägt.

Schlagwörter: Mesh Simplification Algorithms, Vertex Decimation, Edge Collapse, Vertex Pair Contraction, Vertex Clustering, Metro, Mesh Simplification Viewer

Abstract

3D models, which are used in computer games, need to be restricted within their level of detail to keep the hardware specs at a moderate level. If the user can upload models to be used within the game engine it is necessary to control and possibly reduce the number of polygons for providing a gaming experience at playable frame rates. Particularly within the context of 3D online games such behavior is important, because various clients need to render and download these meshes from the game's server. Too much complexity may lead to weak performance and increased data transfer.

Algorithms, which provide the possibility to autonomously reduce 3D models to an adjustable degree of detail, are a desirable solution. These mesh simplification algorithms pursue different approaches and have been a well researched topic for the last decades.

We compare different solutions to find the most suitable for our use case. Within this comparison we come to the conclusion that algorithms based on vertex pair contraction are one of the most suitable mesh simplification algorithms for the use case described in the last paragraphs. This thesis will justify this result by comparing relevant algorithms both theoretically and practically and measuring their strengths and weaknesses in context to the predefined use case.

Keywords: *Mesh Simplification Algorithms, Vertex Decimation, Edge Collapse, Vertex Pair Contraction, Vertex Clustering, Metro, Mesh Simplification Viewer*

Contents

List of Abbreviations

GPU Graphics Processing Unit
PLY Polygon File Format
RAM Random Access Memory
SIGGRAPH Special Interest Group on Graphics and Interactive Techniques

1 Introduction

1.1 Motivation

Reducing the number of polygons in a model is useful to improve storage, transmission, computation and display (Heckbert and Garland 1997, 1). Within the scope of a virtual world architecture this feature is crucial for improving performance.

In online communities like Twinity[1] or Second Life[2] everybody can express himself by altering his avatar and creating user generated content, which causes lot of incalculable momentum. The possibility to upload self created 3D models leads to a major problem: In Twinity there is currently no optimization for the amount of triangles used within these meshes, which may lead to overly complex models placed in user generated areas (cf. Metaversum GmbH 2011). The complexity of these models will be defined as the number of vertices, edges and faces within a single mesh.

Therefore the users in online gaming should be able to enable mesh simplification on their models before uploading to avoid the following scenarios (Ferreira 2011, 7):

- Overly complex content may lead to poor performance on some clients with limited computational resources.
- Very detailed meshes require more bandwidth when they are being downloaded on the fly. Users with limited bandwidth may be handicapped when the mesh data can't be downloaded fast enough for an acceptable user experience.

Algorithms handling mesh simplification have been a well researched topic over the last decades. Since Schroeder, Zarge and Lorensen presented a vertex decimation algorithm at SIGGRAPH '92 (Schroeder, Zarge, and Lorensen 1992) a lot of research to this area has been proposed. Generally they vary widely in approach, efficiency, quality and generality. Also many algorithms are restricted to or only perform well on manifold[3] surfaces (Talton 2004).

Research on these major algorithms needs to be taken to find suitable candidates. To achieve best possible results, a comparison in certain terms relevant for the portrayed use case is essential.

1.2 Requirements for the Use Case

The mesh simplification must not occur in real time, but needs a wide support for different mesh types while delivering good approximation to the original mesh. The less restrictions included within an algorithm the better it will be qualified for the use case. Preserving the original topology of the mesh is not required and also not desired since it introduces additional constraints in the reduction process (Franc 2002, 5).

In the end, following points, which will be explained in the next paragraphs, must be taken in account:

- The algorithm must be efficient,
- the algorithm must be automatic and
- the algorithm must be general (Ferreira 2011, 8).

1. `http://www.twinity.com`
2. `http://secondlife.com`
3. Manifold surfaces will be further defined in chapter 1.2.

Efficiency is the tradeoff between speed and a good optical result. A good optical result is defined as a reduced mesh with a tolerable loss in quality where the most important mesh details appear to be unaltered in the game. It is also important that the algorithm doesn't invest too much time for simplification since it will be executed every time users upload custom meshes to the system.

Automatism means here that the algorithm must not require much parameters or options for configuration to work properly. Thus the algorithm should also be useable for novice users without any special knowledge and also improve usability in continuous exertion. A desirable result is the need of only one parameter controlling the compression rate.

Generality defines that the algorithm should work on a wide variety of mesh structures. This is a very important point, since the algorithm will be confronted with user generated content. For instance it can't be guaranteed that the mesh is manifold or non–intersecting[4].

Garland defines a polygonal surface as manifold "if every edge has exactly two incident faces (expect edges on the boundary which must have exactly one), and the neighborhood of every vertex consits of a closed loop of faces (or a single fan of faces on the boundary)" (Garland 1999b, 9). Check out figure 1 for a few examples of manifold and non-manifold faces.

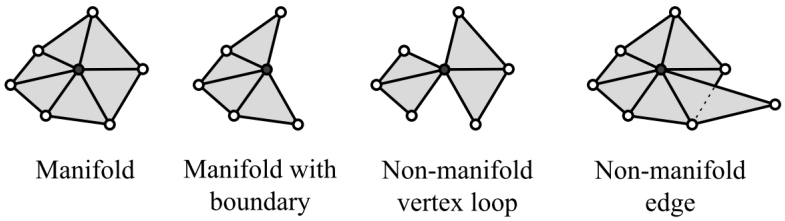


Figure 1: Manifold and non–manifold neighborhoods of a given vertex (in black) (Garland 1999b, 9).

The algorithm shouldn't have to assume manifold or non–intersecting setups, because it should work flexible on different mesh topologies.

1.3 Research Question and Scientific Methods

Because of the vast amount of research and solutions for mesh simplification we concentrate only on well–proven approaches that fit the mentioned requirements best. Fitting candidates are vertex decimation, iterative edge collapse, vertex pair contraction and vertex clustering. Strengths and weaknesses of each technique will be elaborated and the remaining candidates will be compared in a practical implementation with various setups in mesh type and compression rate to gather reproducible and comparable values. Additionally the latest scientific approaches in this field will be considered for an alternative solution.

In my research I came to the conclusion that vertex pair contraction using quadric error metrics introduced by Garland and Heckbert (Garland and Heckbert 1997) provides an excellent solution for my problem. This thesis confirms why quadric error metrics are my instrument of choice by answering the following research question:

Why is vertex pair contraction using quadric error metrics one of the most suitable mesh simplification algorithms for automatic compression of user generated meshes?

4. When a face is folded over another face of the same mesh, the mesh is intersecting.

2 Mesh Simplification Algorithms

Simplification is nowadays a fix practice within the field computer graphics. For instance the quantization of color to 24 bit or the clamping of precisions of coordinates or normals within a mesh are useful and long–used approximations, but are indeed simplifications (Luebke et al. 2002, 8). To avoid misunderstanding simplification in this paper aims only on reduction of polygonal meshes regarding their complexity.

Proposals of some procedures focusing on mesh simplification have been published much more than 20 years ago (Franc 2002).

Mesh simplification strategies can be grouped into two categories: Local and global strategies. Local strategies are by far the most common and Talton defines them as following:

> "Typically, they define some mesh operation S that, when applied to a mesh M, acts on a small collection of its elements and produces a new mesh $\overline{M}$ with fewer elements. By repeated application of S, a mesh may be simplified arbitrarily. In order to determine the mesh elements to which S should be applied on a given iteration, S may be associated with an error function (or cost function) that measures the amount of error the operation will introduce into the approximation. By computing the error associated with every possible application of S at a particular iteration, the algorithm can apply the one with minimal cost. This type of heuristic is quite reasonable for simplification problems, and in practice these methods work well." (Talton 2004)

All algorithms introduced in this chapter except for vertex clustering are following this approach and are part of the group of local strategies. The essential difference between these local algorithms is to choose which edge to collapse and how to collapse (Yuan 1999).

Global strategies, on the other hand, are applied to the input mesh as whole (Talton 2004).

Other common methodologies are refinement and decimation. Garland defines them as following:

> "A refinement algorithm is an iterative algorithm which begins with an initial coarse approximation and adds elements at each step. Essentially the opposite of refinement, a decimation algorithm begins with the original surface and iteratively removes elements at each step. Both refinement and decimation share a very important characteristic: they seek to derive an approximation through a transformation of some initial surface." (Garland 1999a, 5)

For our use case we only focus on decimation. A reconstruction of the original mesh is not desirable.

2.1 Vertex Decimation

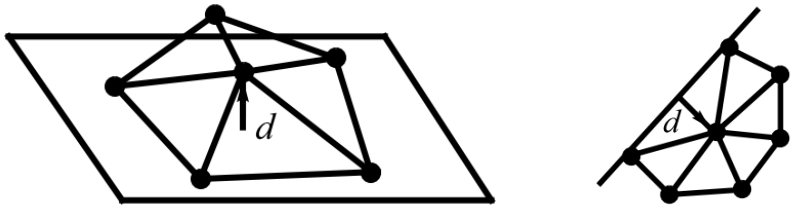
Vertex Decimation was firstly introduced by Schroeder, Zarge and Lorensen at SIGGRAPH '92. They define its basic work as following:

> "Multiple passes are made over all vertices in the mesh. During a pass, each vertex is a candidate for removal and, if it meets the specified decimation criteria, the vertex and all triangles that use the vertex are deleted. The resulting hole in the mesh is patched by forming a local triangulation. The vertex removal process repeats, with possible adjustment of the decimation criteria, until some termination condition is met." (Schroeder, Zarge, and Lorensen 1992)

The termination condition in our use case is the compression rate, which defines the allowed number of remaining vertices in the resulting mesh.

In the beginning the algorithm must decide whether a vertex is a candidate for deletion or not. Therefore all vertices must be classified within five groups that define whether a vertex can be removed and how it's best performed. These groups are divided into simple, complex, boundary, interior edge and corner vertices and will be explained in the next paragraphs.

When a vertex is surrounded by a complete cycle of triangles and each edge using it is exactly part of two triangles then the vertex is simple. They are deleted by using the distance to plane criterion. If the vertex is within a specified distance to the average plane (figure 2a) it may be deleted (Schroeder, Zarge, and Lorensen 1992).



(a) Average Plane (b) Boundary

Figure 2: Average plane and boundary, which assist on deciding whether a vertex is deleted or not (Schroeder, Zarge, and Lorensen 1992)

Schroeder, Zarge and Lorensen describe a further classification of simple vertices as edge or corner vertices:

> "If the dihedral angle between two adjacent triangles is greater than a specified feature angle, then a feature edge exists. When a vertex is used by two feature edges, the vertex is an interior edge vertex. If one or three or more feature edges use the vertex, the vertex is classified a corner vertex." (Schroeder, Zarge, and Lorensen 1992)

Another type is a vertex on the boundary of a mesh (figure 2b), for instance within a semi–cycle of triangles, and is called boundary mesh. Both interior and boundary vertices are deleted by using the distance edge criterion. If the distance to the line defined by the two boundary or feature edges is less than a predefined value, it can be deleted (Schroeder, Zarge, and Lorensen 1992).

On the other hand, non–manifold vertices are defined as complex vertices and are no candidates for deletion (Schroeder, Zarge, and Lorensen 1992).

Figure 3: Vertex decimation distinguishes between these five groups of vertices (Schroeder, Zarge, and Lorensen 1992)

After vertex deletion the resulting hole must be retriangulated. Schroeder, Zarge and Lorensen choose a recursive loop splitting procedure:

> "Each loop to be triangulated is divided into two halves. The division is along a line (i.e., the split line) defined from two non–neighboring vertices in the loop. Each new

> loop is divided again, until only three vertices remain in each loop. A loop of three vertices forms a triangle, that may be added to the mesh, and terminates the recursion process." (Schroeder, Zarge, and Lorensen 1992)

They evaluate the loop split by using a split plane. When every point in a candidate loop is on one side the vertex is free for removal (Schroeder, Zarge, and Lorensen 1992).

Talton observes that vertex decimation algorithms "excel at eliminating extraneous geometry" (Talton 2004). Garland and Heckbert remark the "reasonable efficiency and quality" of vertex decimation (Garland and Heckbert 1997). Vertex decimation has also been improved over the years by the development of several more sophisticated algorithms, like for instance hierarchical triangulation developed by Soucy and Laurendau (Soucy and Laurendeau 1996). Garland remarks that the better results created by such algorithms lead to worse performance and more space consumption (Garland 1999a, 8).

The preservation of the original topology will be rated as a big flaw for our use case defined in chapter 1.2. Schroeder mentions that preserving the original topology is "a strong limiting factor in overall reduction capability, since objects with a large number of holes or other topological constraints cannot be effectively reduced" (Schroeder 1997). Also vertex decimation can't handle non-manifold surfaces. This is also a flaw, because allowing manifold surfaces allows the simplification algorithm to select the better choice on geometric fidelity since it is not limited (Garland 1999b, 9).

Summing up, efficiency and automatism is provided by vertex decimation and makes it a suitable candidate for our use case. On the other hand, the limitations caused by the preservation of the original topology and the restriction to manifold surfaces can't guarantee full support for all types of mesh structures and doesn't fulfill all requirements defined in generality in chapter 1.2.

2.2 Edge Collapse and Vertex Pair Contraction

Garland and Zhou remark that many of the most effective simplification algorithms are based on iterative edge contraction (Garland and Zhou 2005, 2). The edge collapse operator has firstly been introduced by Hoppe at SIGGRAPH '93 (Hoppe et al. 1993).

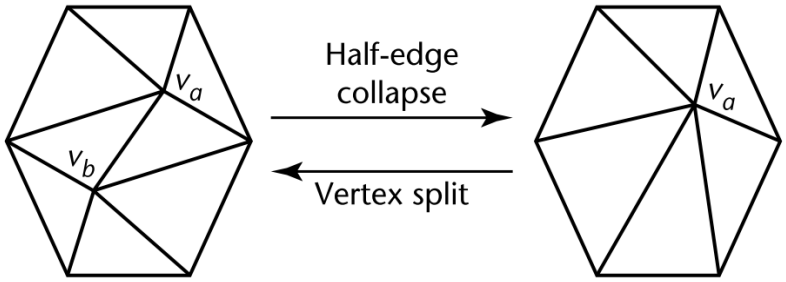
The edge collapse operator removes an edge, which will be replaced by a new vertex v_{new}. The new vertex can be one of the edge's end points (half–edge collapse, figure 4a) or a newly computed vertex (full–edge collapse, figure 4b) (Luebke et al. 2002, 21).

Edge contraction can close holes, but not join unconnected regions. Vertex pair contraction, which is a generalization of edge collapse (Garland and Heckbert 1997; Lindstrom and Turk 2000, 3), supports this type of aggregation. An algorithm based on vertex pair contraction has the benefit, that it is less sensitive to the mesh connectivity of the original model (Garland and Heckbert 1997). This might change the overall mesh topology, but for our use case and for instance in applications with focus on rendering the prevention of the original topology is less important than preservation of the overall shape.

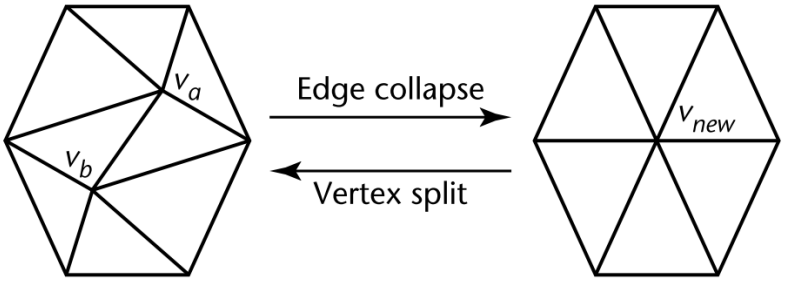
Stan Melax developed a fairly good working algorithm based on edge contraction, which has been used in BioWare's Omen 3D engine (Melax 1998). This algorithm will be practically evaluated in chapter 3.4 of this thesis, but not fully explained at this place, since the main workflow is analogous to vertex pair contraction.

Briefly, Melax evaluates the cost of collapsing an edge by the length of the edge multiplied by a curvature term. He explains this term as following:

> "The curvature term for collapsing an edge *uv* is determined by comparing dot products of face normals in order to find the triangle adjacent to *u* that faces furthest away from the other triangles that are along *uv*." (Melax 1998)



(a) Half–edge Collapse



(b) Edge Collapse

Figure 4: Half–edge collapse: the resulting vertex is newly computed. Edge collapse: the new vertex is one of the edge's end points. (Luebke et al. 2002, 21–22)

The edges with the minimum cost will be selected for removal.

In vertex pair contraction two unconnected vertices v_a and v_b are collapsed (Luebke et al. 2002, 23). Luebke et al. define the basic course of actions as following:

> "Since these vertices do not share an edge, no triangles are removed by a vertex–pair collapse, but the triangles surrounding v_a and v_b are updated as if an imaginary edge connecting v_a and v_b underwent an edge collapse. For this reason, the vertex–pair collapse operator has also been referred to as a virtual–edge collapse. Collapsing unconnected vertices enables connection of unconnected components as well as closing of holes and tunnels." (Luebke et al. 2002, 23)

A well developed and proven algorithm based on vertex pair contraction is surface simplification using quadric error metrics by Garland and Heckbert. They are supporting non–manifold surfaces, since the possibility of the creation of non–manifold regions after joining two separate regions is pretty high and the effort in avoiding the creation of non–manifold regions without limiting the possibilities not worthwhile (Garland and Heckbert 1997).

First of all the algorithm needs to consider the vertices for removal. The selection of all vertices leads to bad performance (Luebke et al. 2002, 23), so Garland and Heckbert select a set of valid vertex pairs at initialization time by the definition of following parameters:

> "We will say that a pair (v_1, v_2) is a valid pair for contraction if either: (v_1, v_2) is an edge, or $|v_1 - v_2| < t$, where t is a threshold parameter." (Garland and Heckbert 1997)

The threshold parameter must be carefully set, since too high values will again lead to bad performance and connect widely separated portions of the model. When the threshold parameter is set

to zero, the algorithm will behave as a simple edge contraction algorithm (Garland and Heckbert 1997).

Then for each vertex pair their quadric error is evaluated and they are sorted by these errors, which are equivalent to the cost of a contraction, in a priority queue (Luebke et al. 2002, 133).

Quadric error metrics try to avoid the flaws that come with other metrics for error measurement (Garland 1999b, 42):

- Metrics based on distance measurements produce good results, but are expensive to evaluate.
- Metrics based on local approximations don't work with non–manifold surfaces.
- Simple measurements, like dihedral edge angles explained in chapter 2.1, do not provide enough information to come up with good results.

A vertex coordinate v and a quadric Q, which is a 4 x 4 symmetric matrix capturing information about a set of planes, will give the sum of the squared distances from the vertex to each plane by solving ($v^T Qv$). The quadrics at each vertex of the original model are initialized to represent the planes of all triangles that meet at that vertex. When two vertices are merged, their quatric of the new vertex is simply the sum of the two old metrices (Luebke et al. 2002, 133-134). The whole mathematical procedure is elucidated in Garland's and Heckbert's paper about error metrics. They summarize the main algorithm as following:

1. Compute the Q matrices for all the initial vertices.
2. Select all valid pairs.
3. Compute the optimal contraction target $\overrightarrow{v}$ for each valid pair (v_1, v_2). The error $v^{-T}(Q_1 + Q_2)\overrightarrow{v}$ of this target vertex becomes the cost of contracting that pair.
4. Place all the pairs in a heap keyed on cost with the minimum cost pair at the top.
5. Iteratively remove the pair (v_1, v_2) of least cost from the heap, contract this pair, and update the costs of all valid pairs involving v_1.

(Garland and Heckbert 1997)

Garland and Heckbert note that their algorithm "provides a mix of efficiency, quality, and generality not found in earlier algorithms" (Garland and Heckbert 1997). Since the requirements for our use case in chapter 1.2 aim exactly in that direction this algorithm is a promising candidate for evaluation.

But there are also a clear weaknesses in simplification stated, relying on the information gathered in the quatrics, where the detection of defunct faces after accumulation might not lead to perfect results (Garland 1999b).

As conclusion mesh simplification using quadric error metrics is not the most accurate and not the fastest algorithm for mesh simplification but the flaws of the others may lead to much more serious conflicts in the given use case. This hypothesis will be proven in the practical comparison in chapter 3.4.

2.3 Vertex Clustering

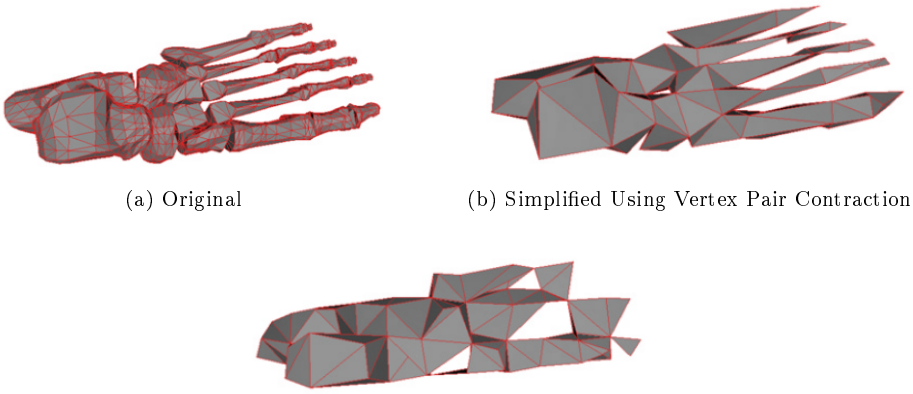
Vertex clustering is the only global simplification strategy that will be explained and evaluated in more detail. Talton observes that "global strategies are far from prevalent in the simplification literature" (Talton 2004). Reasons for that will be evaluated in the following paragraphs.

Vertex clustering has firstly been proposed by Rossignac and Borrel in 1992 (Rossignac and Borrel 1992). It works as following: At first, weight or importance to every vertex in the model is assigned.

Next, a 3D grid is overlaid and all vertices within a cell are collapsed to the single most important one. All triangles that became degenerate within this process are filtered out and deleted (Luebke et al. 2002, 122).

Garland observes that vertex clustering methods are very fast and work well on different types of meshes (Garland 1999a, 7). Even when speed is not the most important point when uploading and simplifying user generated meshes, the support for a wide diverseness in topology of different meshes is a big plus. But when analyzing the quality of the simplified meshes, big flaws are exposed: The process itself is hard to control and the results are not always desirable (Xin et al. 2011).

For better illustration Garland and Heckbert compared algorithms using vertex clustering with those based on edge collapse and vertex pair contraction by simplifying a detailed model of a human food. They come to the conclusion that vertex clustering provides the visually worst result (Garland and Heckbert 1997) as you can see in figure 5c. Allocated to our use case vertex clustering can't guarantee a good enough quality for a wide variety of different meshes. So lots of detail might get lost by the wrong size of the surrounding grid. Additionally vertex clustering doesn't provide enough possibilities to detect areas with retainable detail. So these areas might easily get lost in the simplification process which won't produce desirable results. Also Garland notes that "clustering methods tend to work well if the original model is highly over–sampled and the required degree of simplification is not too great" (Garland 1999a, 7). This observation is not compatible to user generated content, because highly over–sampled models will require a high degree of simplification to improve frame rates.



(a) Original

(b) Simplified Using Vertex Pair Contraction

(c) Simplified Using Vertex Clustering

Figure 5: Simplifying a human food using vertex pair contraction and vertex clustering, where vertex clustering provides the visually worst result (Garland and Heckbert 1997).

Because of these reasons vertex clustering won't be taken in account in the practical comparison at chapter 3.4.

2.4 Recent Approaches

This chapter will deliver an overview over techniques focusing on mesh simplification, which have been established over the last years. They all won't be chosen for the final comparison in chapter 3.4 because of reasons, that will be stated for each technique.

2.4.1 Mesh Simplification using the GPU

DeCoro and Tatarchuk established a technique that adopts vertex clustering to the GPU. With the introduction of the geometry shader to the GPU pipeline real time mesh simplification is amenable to the GPU. The research is based upon performing the vertex pair collapse operation simultaneously to each vertex in a cluster. For that DeCoro and Tatarchuk described a GPU–friendly data structure called the "probalistic octree" (DeCoro and Tatarchuk 2007).

One of the main improvements is the dramatic increase in speed compared with implementations on the CPU. In their test cases the GPU completes the simplification task at least 15 times faster than on the CPU (DeCoro and Tatarchuk 2007).

Executing mesh simplification algorithms on the GPU opens up new possibilities for real time simplification. The users, who upload user generated content, might benefit of fast generated previews of meshes in different simplification stages. But this technique assumes a setup that can't be guaranteed in our use case. Without geometry shaders the GPU can't handle the simplification operation. But for future approaches mesh simplification on the GPU will most likely become interesting for general usage.

2.4.2 Quadrilateral Mesh Simplification

Most simplification algorithms are based on meshes with triangular surfaces, but the interest in developing algorithms that operate on quad meshes has been growing over the last years (Daniels et al. 2008). Lai, Kobbelt and Hu remark that quad dominant meshes describe the structure of a mesh in a more natural way than their equivalent with triangular surfaces (Lai, Kobbelt, and Hu 2008).

Since the research in this area is rather new and not so well studied this thesis won't focus on this type of simplification. Also Daniels et al. mention that the structure of quadrilateral elements forces constraints on mesh connectivity (Daniels et al. 2008). In addition it must be considered that a triangular mesh must be converted into a quad mesh before any simplification can be executed. Because of these flaws quadrilateral mesh simplification is currently not suitable for our use case.

2.4.3 Out–of–Core Simplification

Lindstrom observes that with improvements in processor speed and memory capacity it is possible to create datasets in a size, which is no longer adoptable to early designed simplification algorithms. With this huge amount of data these algorithms require a lot of internal storage for computation, which leads to insufficient simplification speed. Because of that Lindstrom developed an out–of–core simplification algorithm based on uniform sampling via vertex clustering capable of simplifying very large models (Lindstrom 2000).

It is highly unlikely that such huge meshes need processing in our use case. Meshes, which are created for games don't follow the requirements for out–of-core–visualization, which copes with models, that are too large to fit into main memory (Lindstrom 2000). Therefore out–of–core simplification algorithms are not designed for our use case and won't be considered.

3 Comparison

After the theoretical comparison of various simplification techniques we come to the conclusion that algorithms based on edge collapse and vertex pair contraction, which have been evaluated in chapter 2.2, fit the requirements for efficiency, automatism and generality best. Based on the theoretical evaluation in chapter 2 and also on Ferreira's paper about mesh simplification in a

virtual world (Ferreira 2011, 9) algorithms based on Garland's and Heckbert's surface simplification using quadric error metrics (Garland and Heckbert 1997) should perform best.

For verifying these results a practical setup, which compares algorithms based on edge collapse and vertex pair contraction, is required. It should be able to simplify a wide variety of meshes within different compression rates and compare their similarity to the uncompressed meshes.

Jeffrey Somers implemented four different algorithms, which are based on surface simplification using error metrics and edge collapse (Somers 2002). We extended the functionality of his program to create the required setup.

3.1 Measuring Error in Simplified Surfaces

For measuring the similarity between the simplified and original meshes the small program Metro[5] will be used, because of the following reasons:

- Metro is a general tool, which allows the user the comparison of different simplification approaches on meshes to decide, which simplification method fits best to the target mesh (Cignoni, Rocchini, and Scopigno 1998).
- Metro is simple to use and requires no knowledge on the simplification approach adopted to build the reduced mesh (Cignoni, Rocchini, and Scopigno 1998).
- Metro's comparison technique is well proven and established since it has been integrated into MeshLab[6], which is "an open source, portable, and extensible system for the processing and editing of unstructured 3D triangular meshes" (Visual Computing Laboratory 2011).
- Metro natively supports the file format of the simplified meshes, which are in usage at our setup and exported into PLY format. The PLY format will be shortly explained in chapter 3.2.

M.E.S.H.[7] provides similar functionality, but there is no big difference besides a faster execution time (cf. Aspert, Santa-Cruz, and Ebrahimi 2002). The decision for Metro has been made because it can easily be executed via the command line and is better adjustable for generating various comparisons within a batch process.

Metro numerically compares two triangle meshes and evaluates their differences on the basis of the approximation error. The approximation error is calculated by taking a bunch of points on mesh A and compare them with the closest point on mesh B (Cignoni, Rocchini, and Scopigno 1998). But before being able to calculate this value the points must be sampled.

Metro supports three different sampling methods:

- Montecarlo sampling, which picks random samples in the interior of each face,
- subdivision sampling, which subdivides each face along the longest edge and chooses the sample in the center of each cell and
- similar triangles sampling, which subdivides each face in polygons similar to the face and sample it with the vertices of these polygons (Cignoni, Rocchini, and Scopigno 1998).

5. using version 0.4.07 from `http://sourceforge.net/projects/vcg/files`
6. `http://meshlab.sourceforge.net`
7. `http://mesh.berlios.de`

The value measuring similarity will be calculated via the Hausdorff distance. Guthe, Bordin and Klein define the Hausdorff distance "as the maximum of the distances of all points on both meshes to the other mesh" (Guthe, Borodin, and Klein 2005). In more detail, it calculates the maximum value of minimum distances of each point to the surface of the other mesh.

As a result Metro computes an error measured as an approximation of the surface-to-surface distance, which is evaluated by computing a point-to-surface distance for each sampled point. This value is a numerical evaluation of surface meshes "likeness" (Cignoni, Rocchini, and Scopigno 1998).

To be more precise Metro calculates the minimum normal distance of the points to the surface to evaluate the adjacent plane. With the points on the plane it evaluates the maximum distance to the planes on the other model to get the maximum error. This measurements are taken from the simplified to the normal mesh and vice versa and the maximum error value is selected.

The lower this calculated error value between a simplified and an uncompressed mesh, the better the optical quality of the simplified mesh.

3.2 Mesh Simplification Viewer

Jeffrey Somers implemented four different mesh simplification algorithms, where the user can remove triangles from the mesh and the results are displayed in real time. The program with the name Mesh Simplification Viewer and its source code can both be downloaded at Somers project page (cf. Somers 2002).

All the models used for comparison are imported in the PLY polygon file format[8] based on ASCII presentation to make the content of the file readable in text editors. It is a simple and flexible file format, which is very popular in the academic and research world. A ply file begins with a header and then displays the indices of the mesh's vertices in 32 bit integers. Then the faces are described by the predefined indices of the vertices (McHenry and Bajcsy 2008, 14).

Garland and Heckbert's simplification algorithm using quadric error metrics is implemented in two versions: "a version weighted by the area of the triangles in the mesh and a version where the triangle areas are not taken into account" (Somers 2002). Somers remarks the following differences between his implementation and the algorithm described in Garland and Heckbert's paper (Garland and Heckbert 1997):

> "Only existing edges are collapsed in this program. Also, no steps are taken to prevent mesh inversion after an edge collapse, which may result in the mesh folding over itself after a collapse." (Somers 2002)

This shouldn't lead to much difference in the results, since Metro computes the values based on distances and not on face directions as pointed out in chapter 3.1.

Somers also implemented Stan Melax's polygon reduction algorithm, which has been described in chapter 2.2. So the setup is ideal for a practical comparison of algorithms based on vertex pair contraction and edge contraction.

Additionally Somers implemented an algorithm he calls "shortest edge first", which he describes as following:

> "The shortest edges within the mesh are the first ones to be removed. This is not a particularly good algorithm. It's just here for comparison's sake." (Somers 2002)

This algorithm will also be compared with the other ones, but only because of the reason that it has already been implemented. There are no expectations that this algorithm will achieve better results than the other ones.

8. Check out `http://paulbourke.net/dataformats/ply` for a further definition of the PLY format

3.3 Implemented Extensions

Somers only allows the reduction of the imported mesh in a single predefined simplification step of five percent. Although it is possible to reduce the mesh multiple times by this fixed percentage a reduction rate that can be freely defined is essential to receive useable results. The performance of the implemented algorithms should be testable in relation to different simplification rates of the target mesh. Somers already defined a variable, which can be set in the code, but his attempt is a little erroneous.

The percentage of simplification is represented by the variable `NUM_PAGEUPDN_INTERVALS`, which was originally an integer–variable. Unfortunately this led to no simplification and wrong values on higher reduction rates, because `NUM_PAGEUPDN_INTERVALS` had always the same value on reduction rates beyond 50 %. This has been easily fixed by turning `NUM_PAGEUPDN_INTERVALS` to a float-variable.

Listing 1: main.cpp: setting up a reduction rate of 90 %

```
183 const int REDUCE_TRI_PERCENT = 90;
184 const float NUM_PAGEUPDN_INTERVALS = 100.0f / REDUCE_TRI_PERCENT;
```

To be able to compare the execution time of the algorithms measurements of the execution time have been added by retrieving the clock time before and after the simplification. After successful simplification this time will be displayed in the title window of the application.

Also the mesh class has been enhanced by the function `writePLY()` to save the simplified data to disk and be able to compare their similarities with Metro. Since the vertices in the data structure are not deleted and only set to inactive due to support for undoing simplification steps, all active vertices are collected, sorted and their positions are written to the file. In the end, the vertex indices for the faces are also collected and written to the file.

Listing 2: mesh.cpp: the writePLY() function

```
361 void Mesh::writePLY()
362 {
363     ofstream plyFile("reduced.ply");
364     plyFile << "ply" << std::endl;
365     plyFile << "format␣ascii␣1.0" << std::endl;
366 
367     std::vector<int> vertexIndizes;
368 
369     int faceCount = 0;
370     for(int i = 0; i < _plist.size(); ++i)
371         if(_plist[i].isActive())
372         {
373             vertexIndizes.push_back(_plist[i].getVert1Index());
374             vertexIndizes.push_back(_plist[i].getVert2Index());
375             vertexIndizes.push_back(_plist[i].getVert3Index());
376             faceCount++;
377         }
378 
379     std::sort(vertexIndizes.begin(), vertexIndizes.end());
380     // delete dual entries
381     vertexIndizes.erase(
382         std::unique(vertexIndizes.begin(), vertexIndizes.end()),
383         vertexIndizes.end());
384 
385     // write the header
```

```
386     plyFile << "element␣vertex␣" << vertexIndizes.size() << std::endl;
387     plyFile << "property␣float32␣x" << std::endl;
388     plyFile << "property␣float32␣y" << std::endl;
389     plyFile << "property␣float32␣z" << std::endl;
390
391     plyFile << "element␣face␣" << faceCount << std::endl;
392     plyFile << "property␣list␣uint8␣int32␣vertex_indices" << std::endl;
393     plyFile << "end_header" << std::endl;
394
395     for(int i = 0; i < vertexIndizes.size(); ++i)
396     {
397         Vec3 pos = _vlist[vertexIndizes[i]].getXYZ();
398         plyFile << pos.x << "␣" << pos.y << "␣" << pos.z << std::endl;
399     }
400
401     for(int i = 0; i < _plist.size(); ++i)
402         if(_plist[i].isActive())
403         {
404             int newIndex1 = -1;
405             int newIndex2 = -1;
406             int newIndex3 = -1;
407
408             for(int j = 0; j < vertexIndizes.size(); ++j)
409             {
410                 if(vertexIndizes[j] == _plist[i].getVert1Index())
411                     newIndex1 = j;
412                 else if(vertexIndizes[j] == _plist[i].getVert2Index())
413                     newIndex2 = j;
414                 else if(vertexIndizes[j] == _plist[i].getVert3Index())
415                     newIndex3 = j;
416
417                 if(newIndex1 != -1 && newIndex2 != -1 && newIndex3 != -1)
418                     break;
419             }
420
421             plyFile << "3␣" << newIndex1 << "␣"
422                 << newIndex2 << "␣" << newIndex3 << std::endl;
423         }
424
425     plyFile.close();
426 }
```

The final course of events during simplification looks like following:

Listing 3: main.cpp: simplification process including all enhancements

```
242 if (g_pProgMesh)
243 {
244     int size = g_pProgMesh->numEdgeCollapses() / NUM_PAGEUPDN_INTERVALS;
245     if (size == 0) size = 1;
246     bool ret = true;
247     double start = omp_get_wtime();
248     for (int i = 0; ret && i < size; ++i) {
249          ret = g_pProgMesh->collapseEdge();
250     }
```

```
251     if (!ret) MessageBeep(0);
252     double end = omp_get_wtime();
253     g_pWindow->displayWindowTitle(end - start);
254     InvalidateRect(g_pWindow->getHWnd(), NULL, TRUE);
255     g_pProgMesh->writePLY();
256 }
257 return 0;
```

3.4 Practical Results

All simplifications are computed using Jeffrey Somer's Mesh Simplification Viewer with our extensions compiled as release version in Visual Studio 2008 Professional. The used hardware environment is based on an Intel Core 2 Duo P8700 @ 2.53 GHz using 4 Gigabytes of RAM.

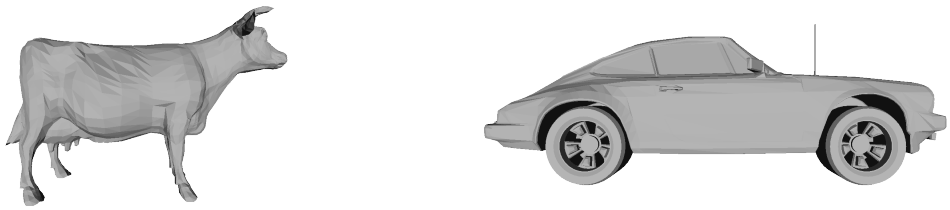
For receiving representative results models with different complexity are imported. All implemented algorithms are measured in execution time and similarity between compromised and original mesh. Different simplification degrees are executed on all models. Around 30 %, 60 % and 90 % of the original edges are removed. For an optical comparison the original and simplified models will be imported into MeshLab[9], where snapshots of the original and all simplified versions are exported.

Different models stored in PLY format are used from different sources, which will be parted in low– and high–poly meshes. Both the execution time and the Hausdorff distance computed by Metro are measured. Metro's default settings with similar triangles sampling will be used.

Also the amount of vertices and faces of the simplified model is documented. This is suitable to draw conclusions to the reduction of file size and saving of storage space. But since we don't expect magnificent differences between the algorithms these numbers are only documented for the sake of completeness.

3.4.1 Measuring Low–Poly Meshes

In Somers's source code are a few models included, from which the cow– and the Porsche–model (figure 6) are used for illustrating the performance on a relatively low number of vertices and faces.



(a) Cow (2903 vertices and 5804 faces) (b) Porsche (5247 vertices and 10474 faces)

Figure 6: The models used for our measurements on low–poly meshes

Models with a low number of vertices will most likely appear erroneous when too much detail is removed. Since the execution time of the simplification of low–poly meshes is negligible — in our use case it must not appear in real time — the quality of the algorithms will mostly depend on the remaining visual similarity between the original and the simplified mesh.

It's no surprise that the shortest edge first algorithm on table 1 delivers the worst results. At a reduction rate of 60 % a lot of detail at the cow's feet and face is lost in comparison to Melax's

9. using version 1.3.1 from `http://meshlab.sourceforge.net`

Cow (2903 vertices/5804 faces)

Reduction rate		Vertices	Faces	Distance	Time
30 %	Quadric (weighted)	2036	4070	0,011601	0,05 s
	Quadric	2036	4070	0,014130	0,04 s
	Melax	2034	4066	0,015793	0,04 s
	Shortest edge first	2039	4076	0,045357	0,05 s
60 %	Quadric (weighted)	1168	2334	0,034556	0,07 s
	Quadric	1168	2334	0,044961	0,06 s
	Melax	1164	2326	0,029562	0,06 s
	Shortest edge first	1173	2344	0,176740	0,07 s
90 %	Quadric (weighted)	301	598	0,127527	0,10 s
	Quadric	300	596	0,199953	0,06 s
	Melax	294	584	0,104653	0,06 s
	Shortest edge first	299	584	0,437767	0,09 s

Table 1: Low–poly measurements on the cow model

algorithm (figure 8), which surprisingly occurs to preserve the visual appearance best at higher compression rates.

When comparing the optical differences of Melax's algorithm with Garland and Heckbert's quadric based simplification at the highest compression rate we see that quadric based simplification keeps the structure of the cow's feet better, while losing more detail at the cow's horns (figure 9a). In the shortest edge first algorithm the results are no longer satisfying since the overall shape of the cow begins to disappear (figure 9c).

Porsche (5247 vertices/10474 faces)

Reduction rate		Vertices	Faces	Distance	Time
30 %	Quadric (weighted)	3678	7336	0,292323	0,04 s
	Quadric	3679	7338	0,287693	0,08 s
	Melax	3680	7340	0,286906	0,04 s
	Shortest edge first	3681	7342	0,051999	0,05 s
60 %	Quadric (weighted)	2109	4196	0,287693	0,07 s
	Quadric	2110	4200	0,287693	0,13 s
	Melax	2113	4206	0,282963	0,06 s
	Shortest edge first	2114	4204	1,386437	0,07 s
90 %	Quadric (weighted)	539	1050	1,409054	0,10 s
	Quadric	540	1054	1,381588	0,09 s
	Melax	539	1058	1,418255	0,07 s
	Shortest edge first	544	1066	1,386437	0,09 s

Table 2: Low–poly measurements on the Porsche model

When examining the results of table 2 the shortest edge first algorithm seems to deliver the best optical result at low compression rates. That's mainly because of an absorption at the car's front lid, which is not correctly triangulated by the other three algorithms (figure 10a). But when having a closer look the weakness of the shortest edge first algorithm is obvious when examining the car's antenna, which already starts disappearing at the compression rate of 30 % (figure 10b).

When compressing to 60 % the shortest edge first algorithm is the only one, which loses all detail at the car's antenna which leads to the visually worst result (figure 11b). At the highest compression rate all algorithms are unable to preserve the most important details of the car (figure 11c).

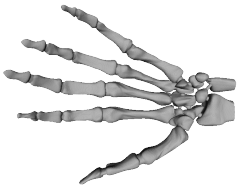
Summing up, when reducing models to low–poly meshes all algorithms are able to simplify the given input very fast. But when analyzing the visual appearance of the simplified meshes the

shortest edge first algorithm delivers in most use cases insufficient results, while algorithms based on vertex pair contraction and edge collapse are able to maintain important detail even at higher compression rates and therefore are both suitable algorithms for our initial use case.

3.4.2 Measuring High–Poly Meshes

The meshes from the Stanford University Computer Graphics Laboratory[10] and Robin Bing–Yu Chen's course page[11] have been used for illustrating the performance of the algorithms on an high number of vertices and faces (figure 7).

(a) Stanford Bunny (34834 vertices and 69451 faces)



(b) Skeleton Hand (327323 vertices and 654666 faces)

Figure 7: The models used for our measurements on high–poly meshes

Even at high compression rates the visual appearance of models with a high number of vertices should appear unaltered. When there's no optical difference after the simplification of high–poly content the quality of the algorithms will be dependent on the execution time.

Bunny (34834 vertices/69451 faces)

Reduction rate		Vertices	Faces	Distance	Time
30 %	Quadric (weighted)	24390	48567	0,15569	0,37 s
	Quadric	24385	48565	0,15569	0,26 s
	Melax	24433	48690	1,174789	0,25 s
	Shortest edge first	24386	48588	1,205414	0,23 s
60 %	Quadric (weighted)	13945	27691	0,261796	0,52 s
	Quadric	13936	27676	0,232709	0,44 s
	Melax	14030	27927	1,565916	0,40 s
	Shortest edge first	13938	27736	1,604202	0,50 s
90 %	Quadric (weighted)	3501	6816	0,925099	1,23 s
	Quadric	3488	6799	1,020669	0,64 s
	Melax	3573	7048	6,455544	1,01 s
	Shortest edge first	3490	6899	3,576613	0,75 s

Table 3: High–poly measurements on the bunny model

Compared to the other implementations at table 3 Garland and Heckbert's algorithm occurs to deliver superior conservation of the model's visual appearance even at high compression rates. Melax's and the shortest edge first algorithm both deliver much less precise results at all compression rates. When comparing the wireframe of the quadric based simplification with Melax's algorithm no big differences stand out at first sight. Only the overall shape of the bunny appears a little less distorted (figure 12a). The high differences in the Hausdorff distance may also come from minor differences between the faces which sum up to a high value because of the immense polygon count.

10. Stanford bunny, which is available on `http://graphics.stanford.edu/data/3Dscanrep`
11. Skeleton hand, which is available on `http://graphics.im.ntu.edu.tw/~robin/courses/cg03/model`

When comparing the bunny's appearance at the highest compression rate we recognize that vertex pair contraction delivers the best optical result (figure 13a). When turning the attention on the rabbit's ears we observe that the preservation of the shortest edge first algorithm is erroneous (figure 13b), while Garland and Heckbert's algorithm is still able to maintain the overall shape. Melax's algorithm is also able to maintain the overall shape, but is producing a lot of holes in the highest compression rate, which of course is not desirable. Though this might be a hint that not all face data is correctly restored after simplification (figure 13c).

Hand (327323 vertices/654666 faces)

		vertices	faces	distance	time
30 %	Quadric (weighted)	229129	457489	0,011598	3,25 s
	Quadric	229130	457453	0,017287	2,26 s
	Melax	229380	454880	0,001957	2,36 s
	Shortest edge first	228743	457119	0,005928	3,22 s
60 %	Quadric (weighted)	130939	261496	0,013832	6,34 s
	Quadric	130941	261437	0,016132	4,51 s
	Melax	131276	259446	0,002291	4,72 s
	Shortest edge first	130675	261304	0,014553	6,45 s
90 %	Quadric (weighted)	32751	65458	0,017274	9,28 s
	Quadric	32750	65420	0,018268	6,77 s
	Melax	33022	64614	0,006332	7,25 s
	Shortest edge first	32700	65385	0,028075	9,67 s

Table 4: High–poly measurements on the hand model

Models with so much detail as pictured at table 4 will never be simplified in our use case. Also when the simplified model still contains over 30000 vertices we won't identify the course of actions as simplification, but more as reduction, since the simplified models still have too much detail for being rendered within a game engine. These tests have solely been made to benchmark all algorithms under extreme circumstances to find out how long the removal of several thousand vertices takes. We come to the conclusion that all algorithms are still fast enough to achieve this within an acceptable time frame.

When analyzing the similarities of the meshes, Melax's algorithm performs best. But since there are still enough vertices left at a reduction rate of 90 % no clear optical differences beyond the different compression rates of the meshes attract attention. Even when only displaying the wireframe of the model it appears as a surface, because of the huge amount of vertices (figure 14).

As conclusion, all implemented algorithms are able to simplify huge amounts of vertices within reasonable time, where both vertex pair contraction and edge collapse are able to remain the quality level of the simplified mesh. The reduction of the skeleton hand was only a benchmarking test where all algorithms were able to simplify the model in reasonable time, so the results have lower priority than the results of the reduction of the bunny model.

4 Conclusion and Answer to the Research Question

We measured a variety of well researched simplification algorithms. Garland and Heckbert's algorithm based on vertex pair contraction and quadric error metrics delivered good results both in our theoretical and practical comparison.

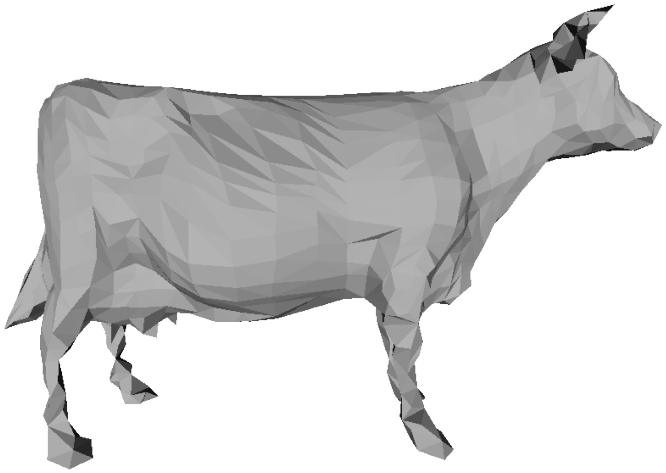
The theoretical comparison leads to the conclusion that within the most established and best researched mesh simplification algorithms those based on edge collapse and vertex pair contraction support the widest range of different mesh structures. Vertex decimation suffers from limitations caused by the prevention of the original mesh topology and the restriction to work only on manifold

surfaces, which also limits the options for simplification strategies. Vertex clustering on the other hand supports a wide variety of mesh structures and works in most cases faster than Garland and Heckbert's algorithm. But the results after the simplification process are not satisfying, especially if there are small areas with retainable detail.

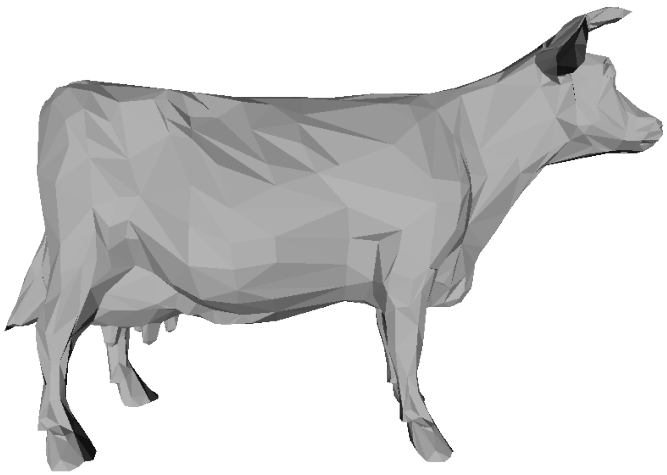
In practice, all algorithms measured performed very efficiently. Even when simplifying a huge model with more than 600000 faces to around a tenth of the original size it was able to perform this task in less than 10 seconds. The analysis of more practical data in relationship to our use case — which is automatic simplification of user generated content with much less vertices — was also satisfying. For instance, on our test machine the simplification of meshes with around 10000 faces was proceeding almost in real time. This implies that even low–end computers should be able to perform this task within reasonable time.

In Jeff Somers's Mesh Simplification Viewer all implemented algorithms are able to simplify meshes only by selecting the favored compression rate. There are no additional input parameters required, which makes all algorithms easily useable by users without technical background.

Summing up, vertex pair contraction using quadric error metrics is one of the most suitable mesh simplification algorithms for automatic compression of user generated meshes because it provides an acceptable tradeoff between efficiency, automatism and generality and meets the requirements defined for our use case. Also algorithms based on edge collapse are an alternative worth mentioning. In our practical tests Melax's algorithm based on edge collapse was able to compete with Garland and Heckbert's solution, but vertex pair contraction has the additional advantage that it supports the merge of unconnected regions which leads to more authentic results in some use cases.

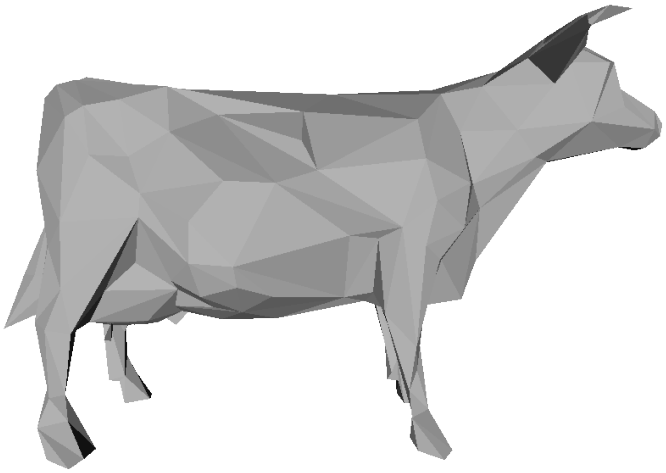


(a) Shortest Edge First Algorithm

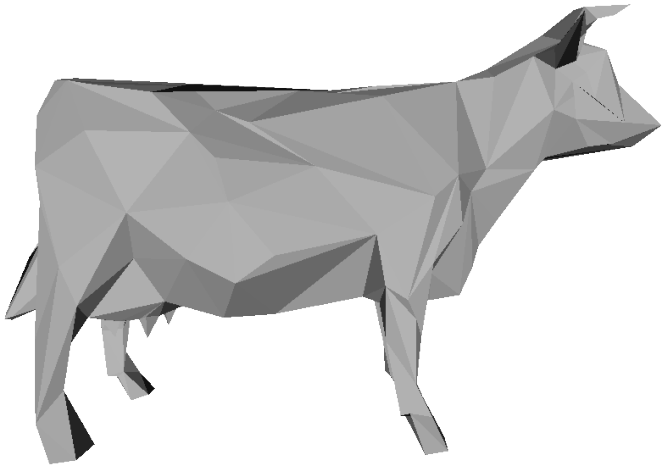


(b) Melax's Algorithm

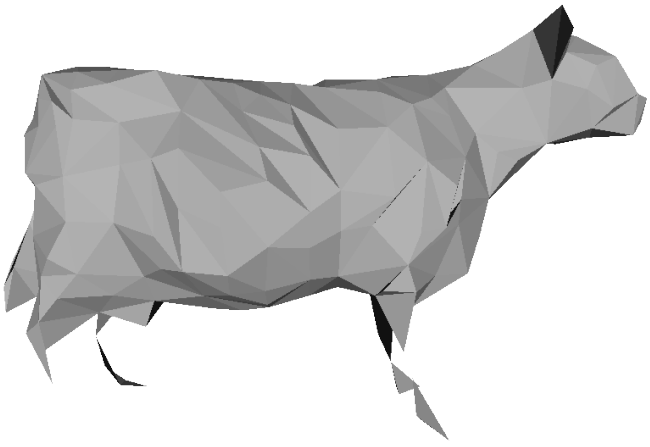
Figure 8: When reducing 60 % of the cow model with the shortest edge first algorithm a lot of detail at the cow's feet and face is lost in comparison to Melax's algorithm.



(a) Garland and Heckbert's Algorithm

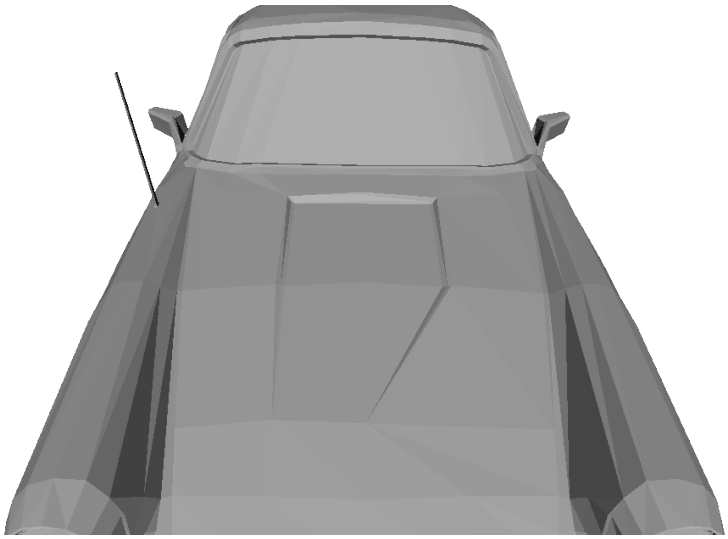


(b) Melax's Algorithm

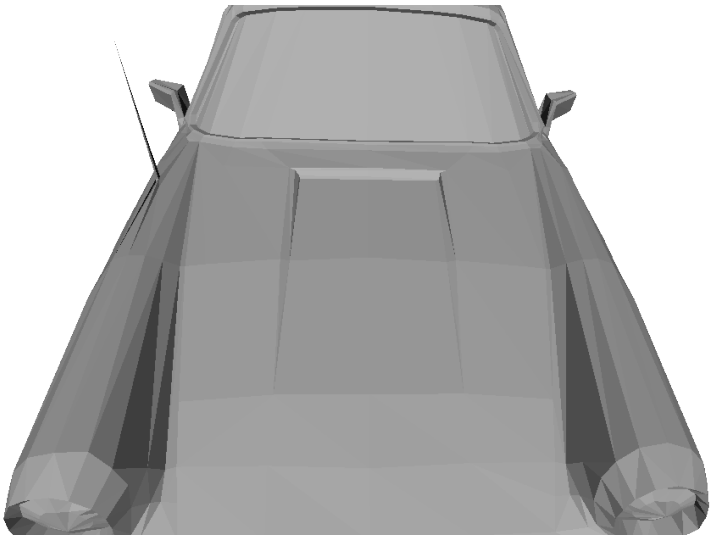


(c) Shortest Edge First Algorithm

Figure 9: When reducing 90 % of the cow model with the shortest edge first algorithm a lot of detail is lost compared to the other implementations.

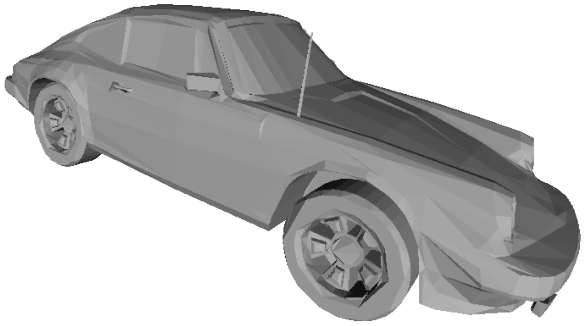


(a) Garland and Heckbert's Algorithm

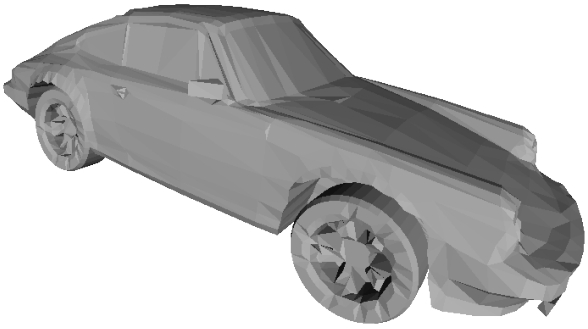


(b) Shortest Edge First Algorithm

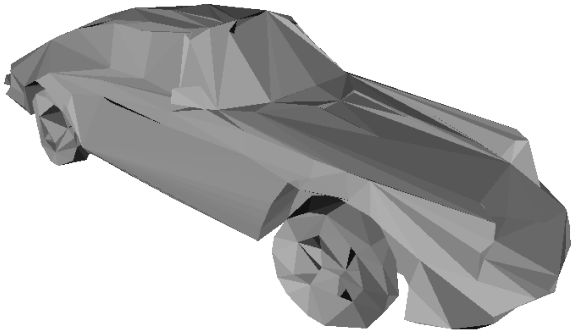
Figure 10: The shortest edge first algorithm is the only one which is able to triangulate the absorption at the car's front lid correctly at a compression rate of 30 %.



(a) Garland and Heckbert's Algorithm at a reduction rate of 60 %

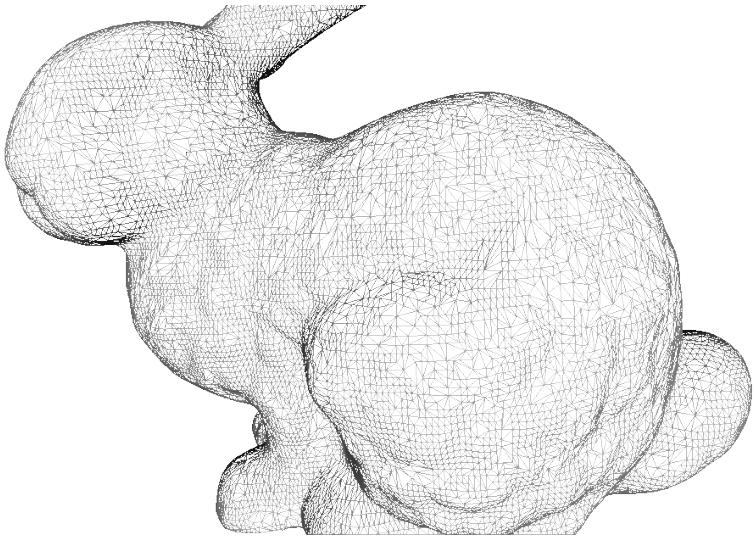


(b) Shortest Edge First Algorithm at a reduction rate of 60 %

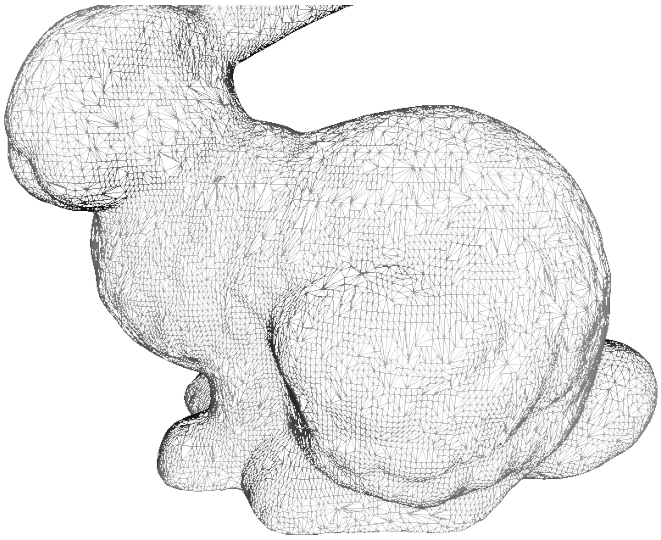


(c) Melax's Algorithm at a reduction rate of 90 %

Figure 11: At a compression rate of 60 % the shortest edge first algorithm loses all detail on the car's antenna, while at the highest compression rate all algorithms are unable to preserve the most important details.

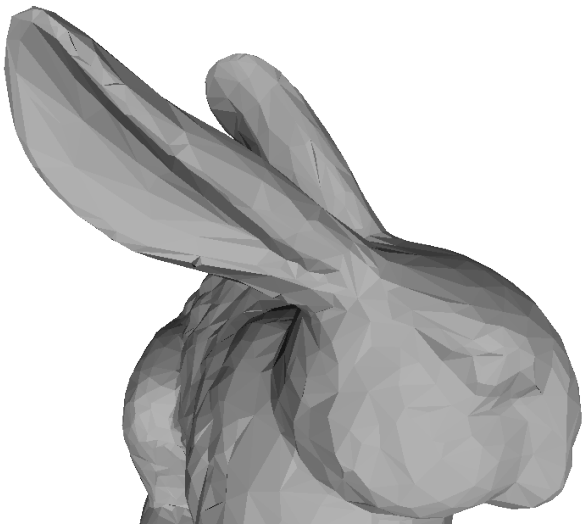


(a) Garland and Heckbert's Algorithm

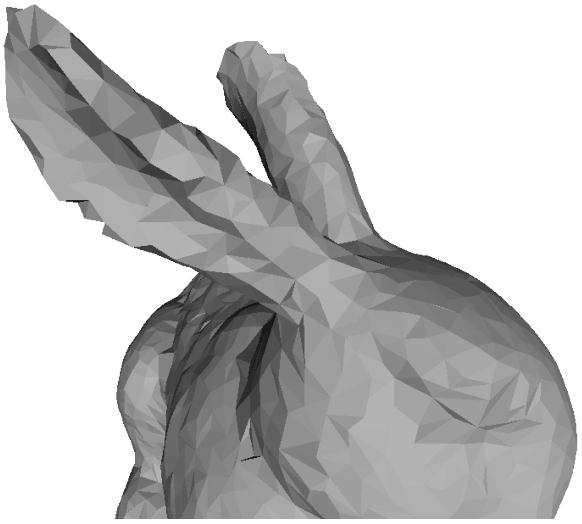


(b) Melax's Algorithm

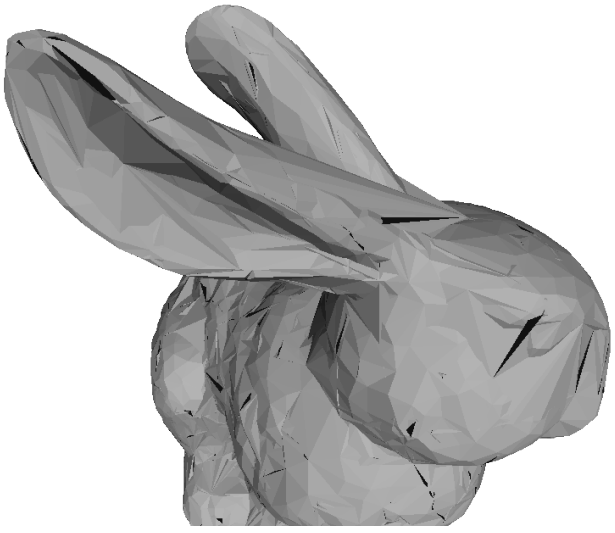
Figure 12: Because of the high detail of the original model no big differences stand out at a compression rate of 30 %.



(a) Garland and Heckbert's Algorithm

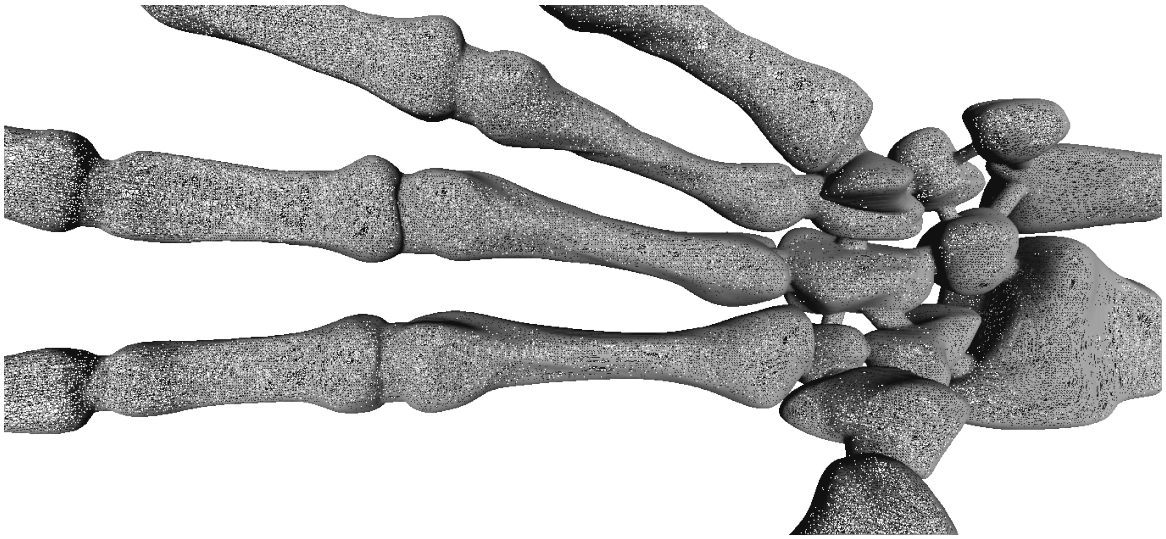


(b) Shortest Edge First Algorithm

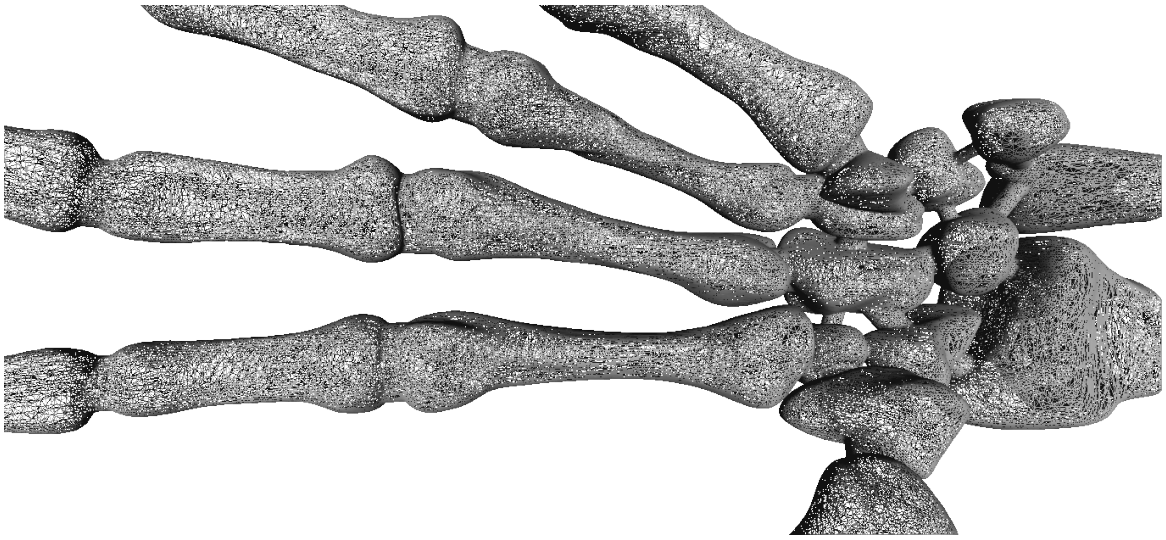


(c) Melax's Algorithm

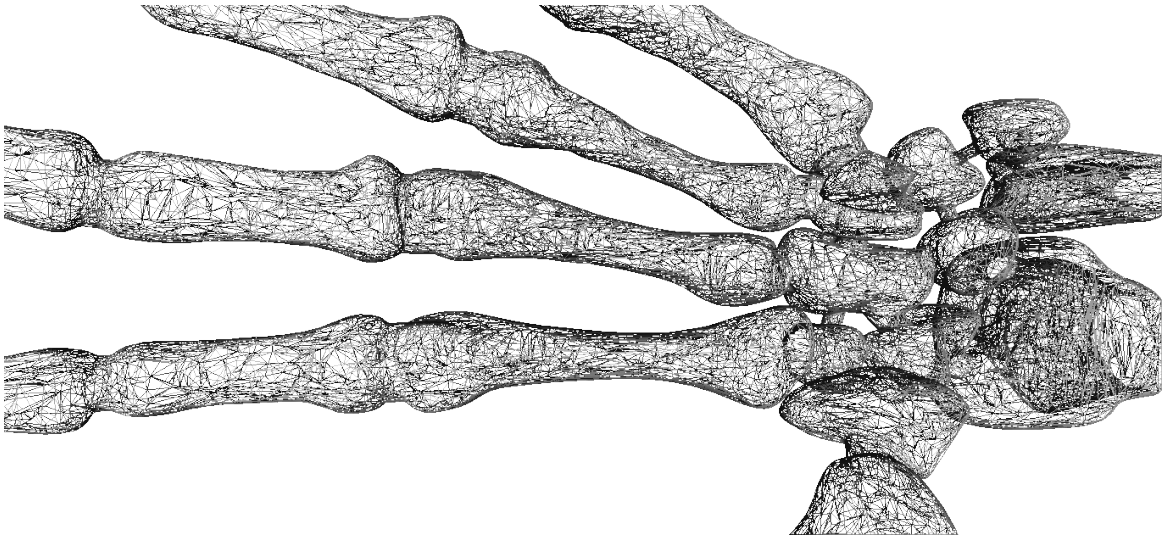
Figure 13: At a compression rate of 90 % vertex pair contraction delivers the best optical result while the shortest edge first algorithm loses detail at the rabbit's ears and Melax's algorithm creates an erroneous mesh.



(a) Melax's Algorithm at a reduction rate of 30 %



(b) Melax's Algorithm at a reduction rate of 60 %



(c) Melax's Algorithm at a reduction rate of 90 %

Figure 14: Because of the huge amount of vertices no visual differences are perceptible within the various reduction steps.

List of Figures

Listings

List of Tables

References

Aspert, N., D. Santa-Cruz, and T. Ebrahimi. 2002. MESH: Measuring Errors between Surfaces using the Hausdorff distance. (Lausanne), Proceedings of the IEEE International Conference on Multimedia and Expo 2002 (ICME), I-II:705–708.

Cignoni, P., C. Rocchini, and R. Scopigno. 1998. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum* 17 (2): 167–174. ISSN: 1467-8659, http://dx.doi.org/10.1111/1467-8659.00236.

Daniels, Joel, Cláudio T. Silva, Jason Shepherd, and Elaine Cohen. 2008. Quadrilateral Mesh Simplification. *ACM Trans. Graph.* (New York, NY, USA) 27 (5): 148:1–148:9. ISSN: 0730-0301, http://doi.acm.org/10.1145/1409060.1409101.

DeCoro, Christopher, and Natalya Tatarchuk. 2007. Real-time Mesh Simplification Using the GPU. (Seattle, Washington), I3D '07:161–166. http://doi.acm.org/10.1145/1230100.1230128.

Ferreira, Eddy. 2011. Texture Atlasing, Mesh Simplification, and Progressive Meshes for a Virtual World Architecture (May).

Franc, Martin. 2002. *Methods for Polygonal Mesh Simplification,* Jan.

Garland, M. 1999a. Multiresolution Modeling: Survey & Future Opportunities.

Garland, Michael. 1999b. Quadric-Based Polygonal Surface Simplification. AAI9950005. PhD diss.

Garland, Michael, and Paul S. Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. (New York, NY, USA), SIGGRAPH '97:209–216. http://dx.doi.org/10.1145/258734.258849.

Garland, Michael, and Yuan Zhou. 2005. Quadric-Based Simplification in Any Dimension. *ACM Trans. Graph.* (New York, NY, USA) 24 (2): 209–239. ISSN: 0730-0301, http://doi.acm.org/10.1145/1061347.1061350.

Guthe, Michael, Pavel Borodin, and Reinhard Klein. 2005. Fast and Accurate Hausdorff Distance Calculation between Meshes. *J. of WSCG* 13:41–48.

Heckbert, P. S., and M. Garland. 1997. Survey of Polygonal Surface Simplification Algorithms. (Pittsburgh, PA 15213).

Hoppe, Hugues, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1993. Mesh Optimization. (Anaheim, CA), SIGGRAPH '93:19–26. http://doi.acm.org/10.1145/166117.166119.

Lai, Yu-Kun, Leif Kobbelt, and Shi-Min Hu. 2008. An Incremental Approach to Feature Aligned Quad Dominant Remeshing. (Stony Brook, New York), SPM '08:137–145. http://doi.acm.org/10.1145/1364901.1364921.

Lindstrom, Peter. 2000. Out-of-Core Simplification of Large Polygonal Models. (New York, NY, USA), SIGGRAPH '00:259–262. http://dx.doi.org/10.1145/344779.344912.

Lindstrom, Peter, and Greg Turk. 2000. Image-Driven Simplification. *ACM Trans. Graph.* (New York, NY, USA) 19 (3): 204–241. ISSN: 0730-0301, http://doi.acm.org/10.1145/353981.353995.

Luebke, David, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, and Amitabh Varshney. 2002. *Level of Detail for 3D Graphics.* New York, NY, USA: Elsevier Science Inc. ISBN: 1558608389.

McHenry, K, and Peter Bajcsy. 2008. An Overview of 3D Data Content, File Formats and Viewers. *Technical Report ISDA08002:*21. http://isda.ncsa.illinois.edu/peter/publications/techreports/2008/NCSA-ISDA-2008-002.pdf.

Melax, S. 1998. A Simple, Fast and Effective Polygon Reduction Algorithm. *Game Developer Magazine,* no. 11 (Nov.): 44–49. http://www.melax.com/gdmag.pdf.

Metaversum GmbH. 2011. Twinity. [Online; accessed on 19.01.2012]. Oct. http://www.twinity.com.

Rossignac, Jarek, and Paul Borrel. 1992. *Multi-Resolution 3D Approximations for Rendering Complex Scenes.* Technical report. IBM Research Report RC 17697. Also appeared in *Modeling in Computer Graphics*, Springer, 1993. Yorktown Heights, NY 10598, Feb.

Schroeder, William J. 1997. A Topology Modifying Progressive Decimation Algorithm. (Phoenix, Arizona, United States). http://portal.acm.org/citation.cfm?id=266989.267059.

Schroeder, William J., Jonathan A. Zarge, and William E. Lorensen. 1992. Decimation of Triangle Meshes. (New York, NY, USA):65–70. http://dx.doi.org/10.1145/133994.134010.

Somers, Jeffrey. 2002. Mesh Simplification Viewer. [Online; accessed on 19.01.2012]. http://jsomers.com/vipm_demo/meshsimp.html.

Soucy, Marc, and Denis Laurendeau. 1996. Multiresolution Surface Modeling Based on Hierarchical Triangulation. *Comput. Vis. Image Underst.* (New York, NY, USA) 63 (1): 1–14. ISSN: 1077-3142, http://dl.acm.org/citation.cfm?id=229144.229146.

Talton, Jerry O. 2004. A Short Survey of Mesh Simplification Algorithms.

Visual Computing Laboratory. 2011. Meshlab. [Online; accessed on 19.01.2012]. http://meshlab.sourceforge.net/.

Xin, Shi-Qing, Shuang-Min Chen, Ying He, Guo-Jin Wang, Xianfeng Gu, and Hong Qin. 2011. Isotropic Mesh Simplification by Evolving the Geodesic Delaunay Triangulation. (Washington, DC, USA), ISVD '11:39–47. http://dx.doi.org/10.1109/ISVD.2011.14.

Yuan, Ping. 1999. *Benchmarking Mesh Simplification Algorithms in Real-time Rendering Applications.*