FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Weighted Mesh Simplification of 3D Triangular Surfaces

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computergraphik/Digitale Bildverarbeitung

eingereicht von

Norbert Ketterl
Matrikelnummer 0627793

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Univ.Prof. Dipl.-Ing. Dr. techn. Werner Purgathofer
Mitwirkung: Dipl.-Ing. Dr. techn. Robert F. Tobler

Wien, 13.04.2011

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

# Erklärung zur Verfassung der Arbeit

Norbert Ketterl
Rosinagasse 1-3/14
1150 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, September 14, 2011 _____

Ort, Datum Unterschrift

# Abstract

Representing huge triangular datasets in a real-time rendering environment is a challenge which receives continuous attention. There is a growing complexity of geometric meshes on the one hand and an increasing computational power of graphics hardware on the other hand. Although hardware acceleration is very powerful, simplification using software is much cheaper and more flexible. Additionally, for a large class of geometric models, simplification can be performed as a preprocessing step where there is no demand for real-time operations.

The underlying theory of this diploma thesis is based on mesh simplification using quadric error metrics. This algorithm was first published by Michael Garland and it was implemented into Aardvark, a sophisticated rendering framework. It contains a comprehensive set of libraries dealing with data structures in general and polygonal mesh data structures in particular.

The application takes a triangulated mesh as input and iteratively creates an even more simplified approximation by weighting and collapsing suitable mesh areas. Some surface details will be lost, but the overall shape of the model will be preserved. The presented code can handle models by preprocessing static, closed triangular meshes. The iterative computation stops when a user specified percentage of the original mesh size is reached.

A further improvement in a later research project will try to add vertex colors, normals and texture coordinates to the simplification method.

# Kurzfassung

Die Darstellung von außergewöhnlich großen Datensätzen in Echtzeitumgebungen stellt eine Herausforderung dar, der laufend Aufmerksamkeit entgegengebracht wird. Neben dem Anstieg in der Komplexität von geometrischen Netzen nimmt auch die Rechenleistung der Grafikhardware mehr und mehr zu. Obwohl Rechenleistung und die Berechnung durch Hardware sehr leistungsfähig ist, bietet sich durch die Vereinfachung mittels Software an, da sie eine kostengünstigere und flexiblere Lösung darstellt. Zudem kann für viele geometrische Objekte der Prozeß der Vereinfachung als Vorverarbeitungsschritt durchgeführt werden, was nicht in Echtzeit geschehen muß.

Das grundlegende Thema dieser Diplomarbeit basiert auf der Vereinfachung von Dreiecksnetzen mit der auf Quadriken basierenden Fehlermetrik. Dieser Algorithmus wurde erstmalig von Michael Garland veröffentlicht. Der Algorithmus wurde in Aardvark implementiert, ein durchdachtes Rendering-System, welches eine reichhaltige Anzahl von Softwarebibliotheken über Datenstrukturen im Allgemeinen und Datenstrukturen über polygonale Datenstrukturen im speziellen beinhaltet.

The Anwendung lädt ein trianguliertes Modell und erzeugt sukzessive eine vereinfachte Annährung an das Original durch die Gewichtung und das Zusammenfassen geeigneter Bereiche des Netzes. Einige Oberflächendetails gehen dabei verloren, jedoch bleibt die grundlegende Form des Objekts erhalten. Die vorliegende Anwendung behandelt statische und geschlossene Netze. Nachdem der gewünschte Grad der Vereinfachung erreicht ist, stoppt das Programm.

Weitere Verbesserungen in späteren Forschungsarbeiten könnten durch das Hinzufügen von Farbinformationen, Normalen und Texturkoordinaten erreicht werden.

# Dedication

I dedicate this thesis to my parents

**THEODOR & FRANZISKA KETTERL**

who offered me unconditional love and
support throughout my academic studies.

I also dedicate this thesis to my girlfriend

**BARBARA**

for her love and patience.
She was always there and encouraged me to finish my work.

# Acknowledgements

It is a pleasure for me to thank the many people who made this thesis possible.

First of all, I would like to express my sincere gratitude to Professor Dr. Robert F. Tobler, who gave me the opportunity to write my master thesis at the VRVis. He always took the time to give me support, advice and some important hints. Moreover, he provided me a space in the office and support whenever I needed.

My deepest and sincere thanks go to Michael Schwärzler for his kind writing advices and organizational assistance. He continuously encouraged me and provided a stimulating and entertaining environment.

I am grateful to thank Christian Luksch and Harald Steinlechner with their great programming skills. They helped me to add new functionalities and to optimize my source code.

I wish to thank Stefan Maierhofer and the administration staff of the VRVis for their immediate assistance according to organizational questions and problems.

Sebastian, Carsten, Josef and Oliver from my former studies in Regensburg assisted me in proofreading my thesis and gave me some useful hints and suggestions.

# Contents

# Overview

This thesis examines different techniques for geometrical mesh simplification algorithms, their options, their differences and their assets as well as their drawbacks.

The following Chapter 1 contains a general introduction on mesh simplification. The introduction deals with general questions concerning meshes, reasons for simplification and quality criteria.

Chapter 2 is an overview of preceding work in the field of mesh simplification. Different simplification techniques based on vertices, edges or triangles are discussed with respect to additional references.

Chapter 3 deals with the theory of triangular meshes and quadric error metrics. These techniques provide an insight into the theory of mesh simplification. Using a simple model as an example for generating a reduced model. The essential steps of simplification are pointed out.

Chapter 4 reveals ways and means for an implementation of the simplification algorithm in the Aardvark framework using C#. A major part of this chapter deals with challenges and ways to circumvent various pitfalls, that may occur during the implementation of the simplification algorithm.

Chapter 5 presents the results of the implemented simplification algorithm. The evaluation contains different input models and their subsequent approximations with various levels of detail.

Chapter 6 concludes this thesis by summarizing both the theoretical and practical work and takes a look at aspects that remain for future options and improvements.

Chapter 1

# Introduction

The three-dimensional representation of geometric data in computer graphics make wide use of complex spatial models for defining the underlying geometry, the logical space and the topological content. Generally, the complexity of a model is often measured by its number of polygons. The rendering speed is often determined by the displayed number of triangles per second. Especially in application areas, where interactive frame rates are of a big concern, it is very important to deal with highly detailed polygon models while maintaining fast rendering performance. The ever increasing processing speed of graphics hardware combined with hardware accelerated rendering algorithms on different platforms leads to more efficient processing of polygonal models, even on low-cost graphics hardware. Still, despite growing rendering speed, the complexity of hand-crafted or reconstructed models is often too high to achieve acceptable frame rates for interactive graphics rendering.
Therefore it is often necessary to substitute large and highly detailed models with simpler approximations as seen in Figure 1.1).

## 1.1 Why Mesh Simplification?

Mesh simplification (or mesh decimation) is the process of removing vertices and faces from a mesh in order to occupy less storage space in memory and being more efficient in rendering the mesh. Usually in computer graphics, there is an high demand on rendered images which has to be as realistic as possible. Therefore, very detailed scenes are an important source for high quality renderings. One way of getting this level of detail is using high resolution objects. Unfortunately, high resolution of spatial content will use a lot of memory and rendering is complex and uses a lot of hardware. This limitations makes them unusable for interactive rendering frame rates, e.g. architectural demos, visualization applications and games. It is simply impossible to render a few hundred million faces in real-time and being able to move around interactively. However, it might be possible to render one object consisting of 100k triangles and the rest of the scene with

Figure 1.1: Illustration of Simplification. Image taken from CGAL Manual [6]

another 100k. By using mesh simplification, several versions of different complexity levels of a model can automatically be generated.
There are two common kinds of simplified representations:

The first one is a view-dependent simplification in Figure 1.2 which is often named as *level-of-detail*. It is often used for terrains, where details closer to the viewer are displayed more detailed than objects which are farther away from the camera, e.g. there is one level if the camera is near to an object, one when the camera is a little bit further away and one when the object is almost impossible to determine from the point of view.

The main concept of *view-independent simplification* uses a three-dimensional object as input and automatically reduces the number of triangles along the surface without losing the representativeness of the object ((see Figure 1.2).

Nowadays, polygonal surface models are the most common representations used in three-dimensional computer graphics applications. Especially in real time rendering, triangular surfaces have been used for many years. A triangle consists of three corners or vertices which are connected by three sides or edges. Every edge is a line segment. A triangle is both the simplest polygonal form and the simplest spatial surface possible, as shown in Figure 3.7. The geometrical simplicity of the triangle led to a de facto standard in computer graphics environments over the last few years. Because of the simplicity

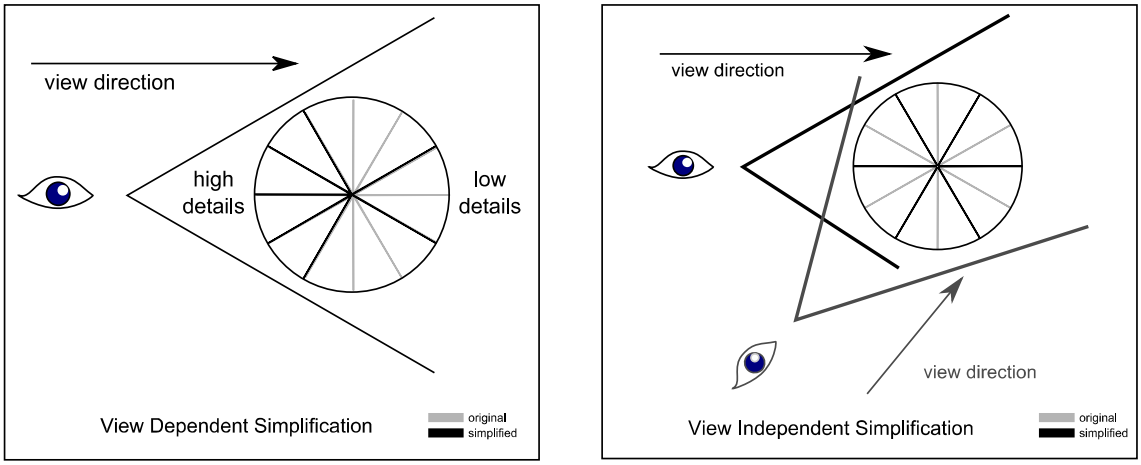


Figure 1.2: View-Dependent and View-Independent Simplification

of triangles, the rendering pipeline and hardware acceleration focuses on an enormous speedup for the representation of triangles. The rapid evolution in processing power of graphics hardware has contributed to the availability of hardware accelerated triangle rendering algorithms on many hardware platforms. Besides the research on triangle datasets much research happened in the field of simplification, e.g. level-of-detail, texture management, and the digital reconstruction of three-dimensional objects. More details of the theory on triangles and triangular meshes can be found in Chapter 3.

There are many different sources for the enormous quantity of getting digital datasets from:

- Medical imaging with computed tomography (CT)
- Products designed via computer-aided design (CAD)
- Hand-crafted models for simulation and animation
- Data acquisition from Laser Ranger Scanner (Section 1.2.2)
- Reconstruction from computer vision systems

The major challenge concerning mesh simplification is to find the best balance between the preservation of visual content and the optimal processing speed.

This Thesis presents different geometric techniques for the simplification of complex polygonal models in order to achieve high-quality approximated models at low processing costs and fewer user interactions.

## 1.2 Introduction to Simplification

There is a wide variety of mesh simplification algorithms available. So, how do we choose the one which fits the required tasks best? According to the article of David P. Luebke [27], there are three major questions regarding the problem of model simplification:

- *"Why do I need to simplify polygonal objects?"* 1.2.1
- *"What are my models like?"*
- *"What matters to me most?"* 1.2.3

### 1.2.1 Reasons for Simplification

Several reasons for the simplification of polygonal objects are:

Initially, the elimination of redundant geometry is a major part in models with flat regions. A huge amount of small and coplanar triangles can be merged into larger, polygonal regions. There is no geometric error introduced if all of the triangles lie in the same geometric plane.

The second reason for the use of simplification: The possibility of publishing and streaming 3D content with reduced storage costs in the Internet. It can be offered in different levels of complexity. Simplification leads to optimized upload and download speeds and therefore does drastically reduced storage requirements. The benefit of minimum storage requirements is often used for creating larger archives at lower storage costs for spatial models.

The third reason is for the performance gain while rendering simplified models. Polygonal simplification is often used to generate different levels of detail (LODs) for a scene. The objects closest to the viewer has the highest LOD while the objects furhter away from the viewer has a lower detail level. There are different metrics available, e.g. the detail importance of the object or the position and the speed of the moving camera. Applications ranging from video games to simulations use this approach for accelerated rendering and smoother animation and advanced interactivity. But there are also trade offs in using LODs. By switching between two adjacent detail levels, the adding or subtraction of details can be noticed as popping. The scenery looks quite odd because the discontinuity seems not very authentic to the viewer.

### 1.2.2 Data Acquisition

There are different ways of acquiring three-dimensional data like explained in Section 1.1. The current Section contains a few details on data acquisition from laser scanners used at the Stanford University. As an example for laser scanning technology, the sculptures and architecture of Michelangelo were scanned with the goal of combining multiple range and color images together. The external shape and detailed surface characteristics of this ancient sculptures should be accurately digitized. After about 10 years of hard work, there is now a full-resolution 3D model of Michelangelo´s 5-meter statue of David. The model contains about one billion polygons. Table A.1 in Section A presents an overview of the whole process for obtaining digital data of the statue mentioned above.

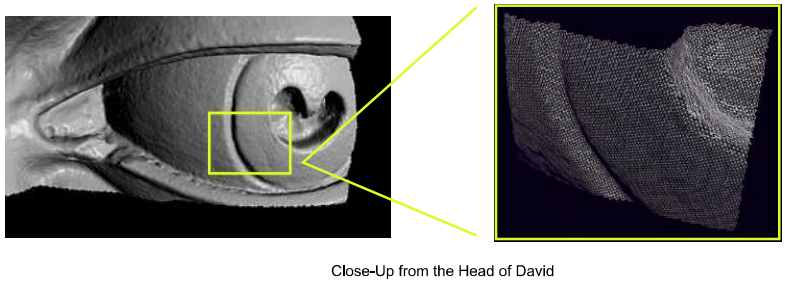


Figure 1.3: Eye Close-Up from Statue of David.
Image taken from Stanford University [26]

As an example of the fine-grained scan of the statue, Figure 1.3 shows a three-dimensional scan of the left eye of David from *The Digital Michelangelo Project* [26]. The left image shows the dataset at the full resolution at 0.29mm. The right image shows a close-up of the reconstructed polygonal surface. The highly detailed model of the eye consists of about 500,000 polygons. In fact, the resulting quality of the model is quite perfect, but it is way too complex for being processed and displayed on current rendering hardware. The qualitative complexity of a model is directly linked to its processing effort. But mostly there is no demand to show every detail of a model, like the hidden background of a scene. In addition, there is an increasing demand for interaction with three-dimensional models, like visualization in real time. Because of the limitations in processing speed while rendering uncompressed datasets, mesh simplification algorithms has been considered as a procreative solution.

### 1.2.3 Quality Criteria

The optimal quality of a simplified model is very hard to determine. On the one hand, the geometric accuracy is measurable. On the other hand, the perception of high visual fidelity is based on the subjective sense of each person. Perception is much harder to quantify than geometry. But some algorithms basically provide higher visual fidelity than others do. Chapter 2 contains a detailed overview of the algorithms which are currently available.

Another quality criteria deals with the preprocessing time. Simplification done via a batch process, handling extremely complex models with many parts and billions of polygons per part, can take from seconds over hours to days until completion. This can lead from a slight inconvenience to a severe handicap in usability of applications. Producing different resolutions of a model for a video game can be seen as just a small restriction at startup with high quality simplification at runtime. Applications using CAD with many altered parts must be viewed in the context of many other parts, are very inappropriate time-consuming simplification procedures.

## 1.3 Nomenclature

The following list shows a the used abbreviations and symbols for polygonal meshes an mesh simplification:

*Genus:*
: The largest number of non-intersecting closed curves that can be drawn on a surface without separating it.

*LOD:*
: Level of detail

*Local topology:*
: Connectivity between immediate neighbors of a face, edge or vertex.

*Manifold:*
: Locally Euclidean space with topological neighborhood information around every point.

*Polygon count:*
: Number of triangles in mesh structure as an simple indicator of mesh complexity.

*Topology:*

The connected structure of a polygonal mesh structure.

*Topology insensitive:*

Valid topology of original mesh not crucial.

*Topology modifying:*

Close up holes.

Connect unconnected regions

*Topology preserving:*

Leaves holes untouched.

Preserves the manifoldness of the mesh.

Does not connect unconnected regions.

*Topology tolerant:*

Leave local non manifold topology untouched.

*VIPM:*

View-Independent Progressive Mesh. Mesh representation with the same resolution and quality independent from the viewpoint and viewing.

*VDS:*

View Dependent Simplification Mesh representation with resolution and quality dependent from the viewpoint and depth-value.

Chapter 2

# State of the Art

The research field of surface simplification is getting more and more attention in the recent years. The simplification process of triangular models is difficult, because quality is hard to determine when using an algorithm. The best approximation of the original mesh is done when the polygon count is drastically reduced but the overall shape and the characterization of the object will be kept. The quality can only be determined with visual perception. Another issue of the algorithm is the computational complexity of the decimation costs, which is a very time consuming and costly task, both in preprocessing and at runtime.

## 2.1 Polygonal Graphics Representation

In computer graphics, a polygonal surface is the most common representation method nowadays. But this is not the only technique of displaying and organizing surfaces. There are various non-polygonal surface alternatives available with their own assets and drawbacks compared to polygonal models. The following points show some reasons for the large distribution and popularity of polygonal models:

- Algorithmic
- Convertible
- Flexible
- File format variety
- Platform Independent
- Hardware accelerated

Basically, the meshes and surfaces used in this thesis have to fulfill the following requirements and specifications: A polygonal mesh is a set of planar polygons in the

three-dimensional Euclidean space $\mathbb{R}^3$. This thesis focuses only on models consisting of triangular polygons. It is possible for any model to contain single vertices which do not belong to an edge and contains some edges which do not belong to a triangle. This non-manifoldness of a mesh has to be handled in a pre-processing phase. The appearance of isolated vertices or edges strengthens the demand for a restrictive implementation of the presented algorithm. The geometrical consistency of triangular meshes is crucial for gaining high fidelity both in original and simplified meshes. The most relevant algorithms available for reducing the complexity of meshes can be roughly divided into this five classes:

- Vertex Clustering
- Vertex Decimation
- Iterative Edge Contraction
- Re-Tiling
- Wavelets

The *Vertex Clustering* algorithm from Rossignac and Borrel [36] divides the whole mesh into a volumetric grid with equally-sized cells. All vertices sharing exactly the same cell are merged together into a single vertex. For achieving coarser models, the size of the clusters has to be increased iteratively. The advantage of this technique is the fast and simple implementation. The output quality is often very bad. More related work on this technique is shown in Section 2.5.1.

The *Vertex Decimation* technique from Schroeder et al. [38] is a very early work on mesh decimation. The algorithm iteratively selects a vertex from the mesh, removes all adjacent faces and closes the resulting hole by local re-triangulation. A closer look at this technique is defined in Section 2.5.2, where different algorithms, including the most recent one, are shown.

The *Iterative Edge Contraction* can be considered as an hybrid approach between the two former algorithms. The surface simplification algorithms using quadric error metric for quality reasons [15] is therefore a member of iterative edge contraction algorithms. The present work focuses on this technique. More details on the original algorithm can be found in Section 3.3.1.

*Re-Tiling*, first introduced by Turk [43], describes an automated way of producing models with multiple levels of detail from an original polygonal model. With re-tiling, an entirely new set of vertices are distributed over the existing surface of the model. Although it is well suited for curved surfaces, it does not perform well for sharp corners and edges. More details can be found in Section 2.5.5.

Using *Wavelets* in mesh compression was introduced by Gross et al. [17]. The extension of wavelets from regular 2D images to irregular 3D meshes involves primarily remeshing techniques. More information about this is presented in Section 2.5.6.

## 2.2 Non-polygonal Graphics Representation

In addition to polygonal surfaces, there exists a wide range of alternative surface representations in computer graphics. This chapter presents short overview of different types of currently available data structures in computer graphics. Although this work deals mainly with polygonal meshes, the current section contains a short overview about other simplification techniques.

### 2.2.1 Point Clouds

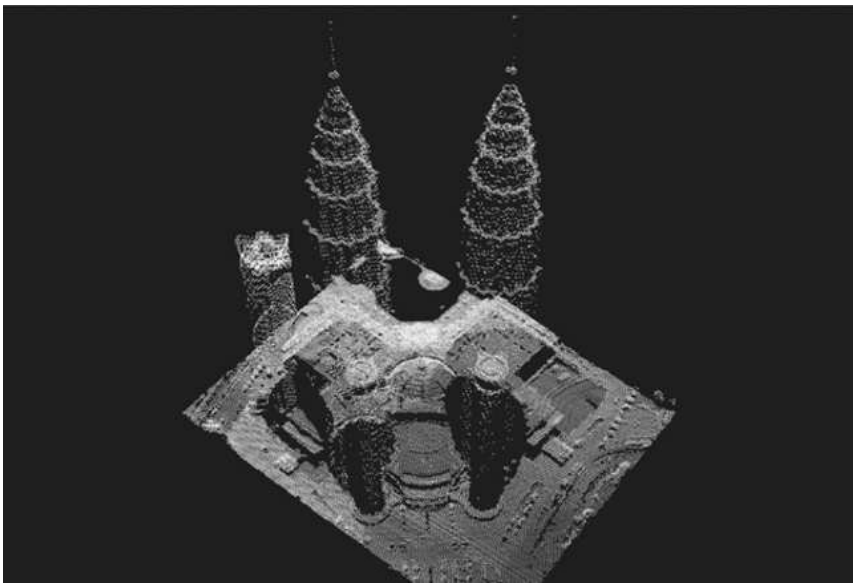

Figure 2.1: Point cloud of Petronas Towers. Image taken from asprs.org [34]

A point cloud dataset [31] [35], as seen in Figure 2.1, is a set of vertices in a three-dimensional coordinate system. Point clouds are mostly created by 3D laser range scanners, where thousands or even millions of discrete 3D points are collected using pho-

togrammetry. The scanner device grabs a large number of points automatically from the surface of the designated object. Besides the spatial coordinates (representing the model) more information (attribute data) like surface color can also be stored. Every point in the dataset can be rendered by representing it as a surfel[1]. A surfel describes the surface surrounding the designated point. The circular points are interpolating the surface between the individual points of the cloud by extending the point to a surfel disc (Figure 2.2). A typical issue is that point clouds taken from large ranges is that they are not uniform. As different datasets are often combined, there can appear some alignment errors. Apart from the position of a point and its color attributes, a surfel also stores the normal vector and the radius of the surface element. It can also be extended by other attributes like material or reflection. A fundamental difficulty of using point clouds is to create continous approxmiations of the underlying models, especially when reconstructing outdoor scenes.

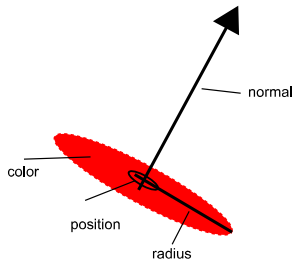


Figure 2.2: Scheme of a Surfel Disc

There are two different techniques mainly used for simplifying point-based models. These techniques are reducing the complexity of objects by representing them with less points than the original model. Point cloud simplification can be ueseful to reduce the complexity of geometric models early in the 3D content creation pipeline, as shown in Figure 2.3.

The main concepts for simplifying point clouds were introduced by Pauly, Gross and Kobbelt in the paper *Efficient Simplification of Point-Sampled Surfaces* [33]. The presented algorithms rely on point-based objects. The simplification processes are directly assigned to the point cloud dataset. The present techniques are similar to the mesh based approaches and modified for point-based data. The selected actions can be divided in these two areas:

- Clustering Simplification (Unsampled)
- Particle Simplification (Sampled)

[1] SURFace ELement

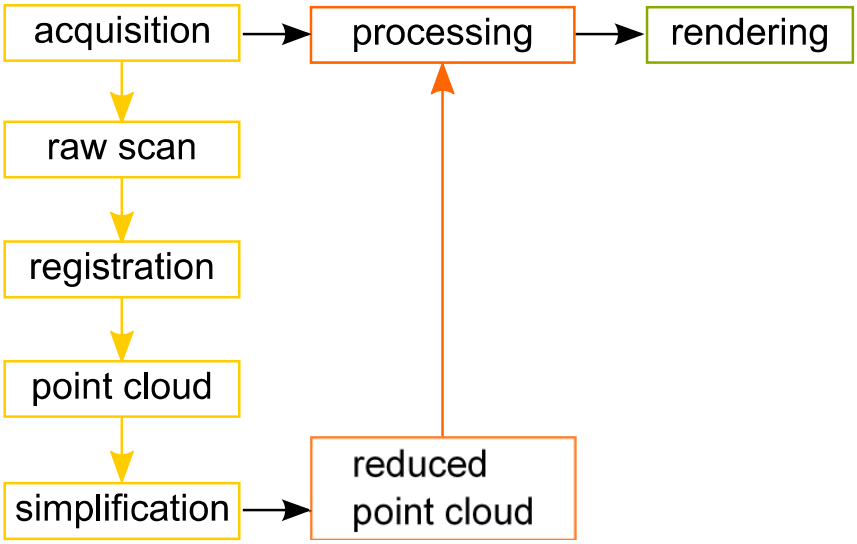


Figure 2.3: Point Cloud Rendering Pipeline

On the one hand, clustering techniques (in Figure 2.4) are chosen to merge several points into a given cell-structure depending on cell-sizes and thresholds. The incremental clustering starts with a random seed point and adds its nearest points successively to the cluster until the designated cluster reaches the maximum size. Hierarchical clustering uses a binary-space partition method, where each cluster of the point-cloud is split. If the current cluster extends an user-specified maximum of points, it collapses into a single cluster. With clustering techniques, no resampling of the overall point structure is going to be performed, which can lead to unpicturesque aliasing artifacts. It is simply a method for removing points from the clouds or merging a bunch of points into a single one.

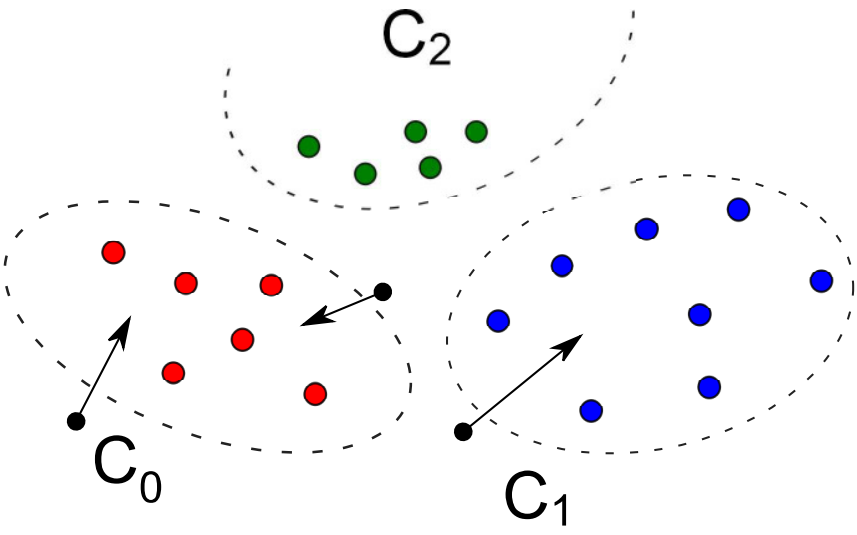


Figure 2.4: Point Cloud Clustering

On the other hand, the particle-based simplification introduced by Moenning and Dogson [32] uses either uniform sampling or adaptive sampling during the simplification process. The presented method is based on a Cartesian raster grid and starts with a small sub-area of the whole point cloud. In contrast to clustering, particle simulation randomly distributes a desired number of particles all over the surface of the underlying point cloud. The number of particles define the number of points for the resulting, and thereby simplified point cloud. The particles on the surface move according to the inter-particle repelling forces until an equilibrium state is reached.

### 2.2.2 Voxels

A Voxel [2] [23] [39] in three-dimensions is analogue to a pixel [3] in two dimensions. A pixel is an array divided into uniform cells, typically squares. Similarly to pixels, voxels divide the 3D space into uniform 3D cells, usually cubes like seen in Figure 2.5. Since voxels are aligned in a regular grid, the absolute location of a voxel is not explicitly stored in spatial coordinates. The selected cell is determined by its relative position to adjacent cells and the origin position of the data set. In contrast to polygonal models, efficient spatial referencing is possible due to the uniform grid structure. Elizabeth Beckman [3] mentions in her article *CT scanning the early years* the Nobel Prize winners in medicine, Sir Godfrey Hounsfield and Allan M. Cormack. They invented the computer assisted tomography and introduced the idea of voxels. Voxels are nowadays widely used in medical imaging.

An important feature of object simplification is the preservation of the original genus of an object. Talton [22] describes a sampling mechanism using a three-dimensional voxel grid. The volumetric representation of the polygonal model is sampled into a three-dimensional voxel grid in an initial processing stage. To each voxel containing polygonal geometry, a value of either 1 or 0 is assigned. Its value depends on whether or not the sample point of the object lies inside or outside the object. Within the next step, a low-pass filter known from signal processing is applied in order to resample the volume. This results in a new volumetric representation with lower resolution. According to the signal processing theory, high-frequency details (like sharp edges or squared-off corners) will be eliminated using a low-pass filter. Unfortunately, results from voxel-based simplification algorithms are not adequate if the algorithm should keep tiny but important details.

[2] VOlumetric piXEL
[3] PICture ELement

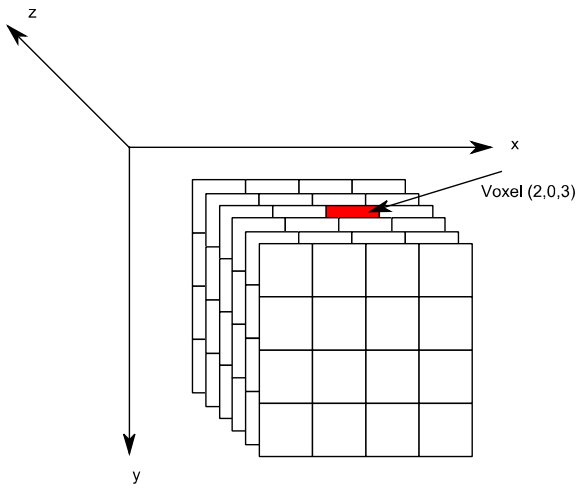


Figure 2.5: A 4x4x6 3D Voxel Grid Image Volume

### 2.2.3 Isosurfaces

Isosurfaces [1] [39] are an enhancement of voxel data sets. An isosurface, displayed in Figure 2.6, is the three-dimensional equivalent to an isocurve in 2D. Suppose a continuous set of value is assigned to each point of a surface. Then a contour line is a line of constant value extracted from that surface. In analogous fashion, an iso-surface is a surface which consists of constant values extracted from a three dimensional dataset with continuous values assigned to each point. Isosurfaces are frequently used in scientific visualization. In these, the datasets are much smaller than in medical applications. In fluid simulation environments the average size of datasets is approximately 100x100x100 voxels. Volume grids created by industrial computer tomographic devices range between 512x512 and 1024x1024 pixels with a scan depth of around 100 slices. An isosurface represents points of a constant value within a volume of space. The main usage of isocurves in computer graphic applications are used for data visualization methods in computational fluid dynamics. It helps engineers to study the flow around objects, such as the air-stream around a wing of airplanes. In medical imaging, isosurfaces are used to represent particular density functions in order to highlight important attributes which are fundamental for visualizing internal bones, organs and other structures.

### 2.2.4 Billboard Clouds

Another method often used in realtime rendering applications is the use of image-based models. The image-based approach is quite powerful in terms of reproducing real-world scenarios. In conjunction with traditional geometric models, image based modeling is an effective technique for load balancing in real-time systems. Image-driven simplification [10] deals with the quality of the simplified model when rendered. Most currently

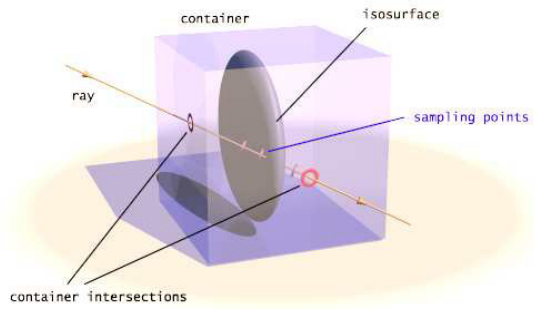


Figure 2.6: Creating an isosurface. Image taken from Christophers Page [21]

available simplification algorithms focus on geometrical error for simplification, whereas image based simplification focuses on an image-based metrics. With the billboard cloud technique [28] in Figure 2.7, complex models are simplified by creating texturized billboards of some of the faces of the original model. Texture and transparency maps are used for getting several levels of detail. With a given geometric error threshold, a billboard cloud is built. This can be useful, especially for a numerous distribution of objects like trees or or houses which have many small polygons. If a tree or a house is viewed from far away, the small polygons are mostly not noticed. But for a closer view, the details are important for modeling a realistic scenario. This kind of geometrical simplification is essential for an optimal distribution of the resources. There should be a good balance between quality and performance.

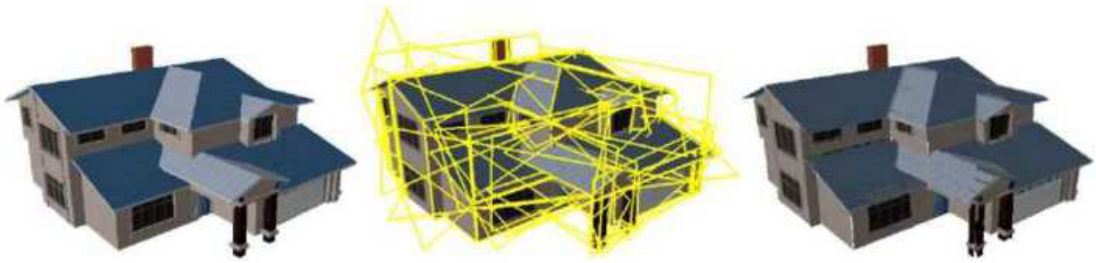

Figure 2.7: Billboard Cloud Example. Image taken from [11]

### 2.2.5 Parametric Spline Surfaces

The parametric spline surface [1], [12] is the most common alternative to polygonal models in computer graphics. It is a nonuniform bicubic spline surface which passes

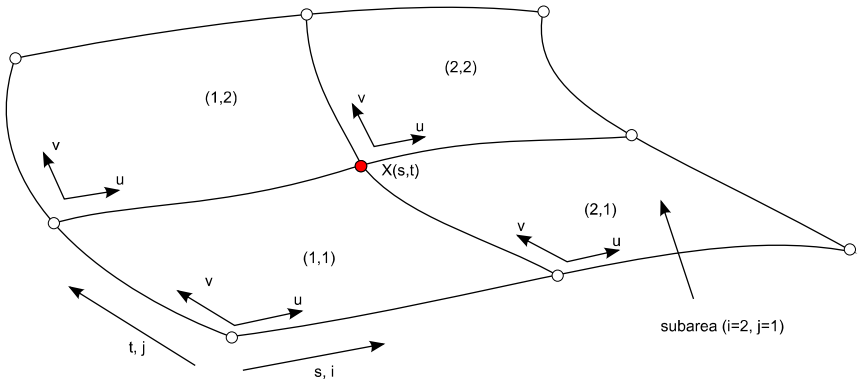


Figure 2.8: Parametric Spline Surace

through a grid of tangent vectors given for each point. They are used to design curved surfaces, such as those found in cars, boats and planes. The grid is curvilinear in the $uv$ space. This system is derived from a set of Cartesian coordinates by using a locally invertible one-to-one mapping at each point. It assigns a number pair $(u, v)$ to each number pair $(s, t)$. This means, a point given in Cartesian coordinates can be converted to its curvilinear coordinates and back. Like shown in Figure 2.8, a point $X(s, t)$ on the plane can be represent by $(s, t)$ coordinates and in Cartesian vector form like in Equation 2.1, where $u$ and $v$ are basis vectors.

$$X = su + ve \tag{2.1}$$

Polygons are represented by piecewise-linear patches. Spline surfaces are composed of piecewise-polynomial patches. Therefore, the polygonal representation is a specialization of the parametric spline representation. Through the use of high-order polynomials, smooth surfaces can be approximated much more accurately than with planar polygons. As with polygons, a spline surface discretized an object into a fixed number of patches. Figure 2.8 shows a parametric spline surface with cubic degree in both directions of its control points. The whole surface consists of several subareas where each of them are a bicubic tensor product face. The point $\mathbf{X}(s, t)$ in Figure 2.8 is defined in the equation 2.3 below:

$$\begin{aligned} \mathbf{X}(s,t) &= a_{i,j} + b_{i,j} \cdot u + c_{i,j} \cdot u^2 + d_{i,j} \cdot u^3 \\ &+ e_{i,j} \cdot v + f_{i,j} \cdot vu + g_{i,j} \cdot vu^2 + h_{i,j} \cdot vu^3 \\ &+ k_{i,j} \cdot v^2 + l_{i,j} \cdot v^2u + m_{i,j} \cdot v^2u^2 + n_{i,j} \cdot v^2u^3 \\ &+ p_{i,j} \cdot v^3 + q_{i,j} \cdot v^3u + r_{i,j} \cdot v^3u^2 + s_{i,j} \cdot v^3u^3 \end{aligned} \tag{2.2}$$

The indices $i, j$ in Equation 2.3 number the subsurface $(i, j)$ of the spline surface matrix. The parameters $a_{i,j} \cdots s_{i,j}$ are the coefficients of the cubic spline curves in both directions and $(u, v)$ are the the parameters of the subface $(i, j)$. With spline surfaces it is possible to define and create arbitrarily curved surfaces. The drawbacks of splines are the extensively high computation effort of their surface coefficients and lack of local curve control. Moving just a single control point affects the whole curve and all surface patches adjacent to it. Some of these drawbacks can be overcome with derived constructions such as *B-Spline surfaces* [1] and *Subdivision Surfaces* [37].

## 2.3 Hybrid Rendering Systems

Additionally, the previously discussed interpretation of graphical objects and rendering techniques, polygonal and non-polygonal, are not used purely standalone. For achieving interactive rendering rates, hybrid approaches are used in scientific research. Points and triangles have enormous differences in terms of model representation and rendering. Both have several advantages and disadvantages. If a single part of the model scene is converted to a screen projection. It covers multiple pixels (see left illustration in Figure 2.9). An algorithms using triangle rendering are more efficient than rendering a number of points to fill the pixel area.

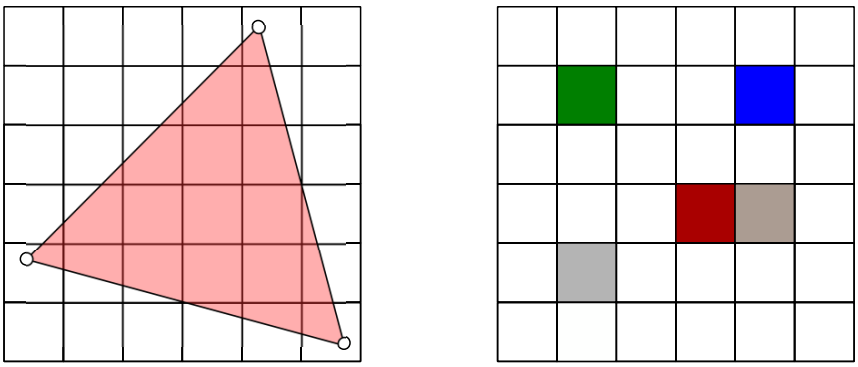

Figure 2.9: Triangle Rendering vs. Point Rendering

If the screen projection resides in sub-pixel sizes, point-based rendering has less overhead in contrast to triangle meshes and is more flexible, too.
Some work has been done for the introduction of hybrid rendering systems, which have the capability of rendering both points and triangles on demand. Chen and Nguyen present in their paper *POP: A hybrid point and polygon rendering system for large data* [8] an effective extension to pure point rendering systems. For rendering large mesh models, both points and triangle meshes are used rather than points only. The proposed technique builds a tree model structure of triangle representation at the leaf nodes and

point representation at the intermediate nodes. The mixture of triangles and points ensures the quality of triangle rendering on one hand and the speedup with point rendering on the other hand. The choice whether triangles or points are used depends on the point of view. The closer the eye is to the model, the higher percentage of triangles is rendered. Points are used as a secondary rendering primitive, mainly used as preview. The most detailed level is still rendered with triangles only, which is no speedup compared to rendering with triangles only.

Cohen et al. present in their paper about hybrid simplification [9] a technique for combining multi-resolution polygon rendering and point rendering, where two single data structures are being merged into a single hierarchical data structure.

## 2.4 Data Structures representing Meshes

In computer graphics, the data structures representing polygonal meshes are very important. The better organized the data structures are, the more they are capable of being rendered without major modifications. Within point cloud sets, the main informations are the spatial coordinates, the colors and the underlying grid for maintaining the modular grid of the points. There is no connection between the single points. In mesh data structures, the connectivity between vertices and triangles must be established. A pure geometrical representation of a triangle mesh contains piecewise linear surfaces aligned along their edges. The pure mesh geometry can be described as a tupel $M = (K, V)$, where K is the simplicial complex, which describes the face connectivities (adjacent vertices, edges and triangular faces), and $V = \{v_1, ...., v_m\}$ represents the set of vertex positions defining the geometrical shape of the mesh (in $R^3$). Rather than representing just geometry, mesh data structures often contain appearance attributes, too. According to Hugues Hoppe [19], these attributes are categorized into *discretized* and *scalar* types.

First of all, the discrete attributes belong to the faces of the mesh. Material identifiers or shading parameters for texture lookups are common discrete attribute types. Second of all, many scalar attributes are connected with a mesh, including vertex normals $(n_x, n_y, n_z)$, the diffuse color $(r, g, b)$ and also texture coordinates $(u, v)$. In a simple configuration, every single vertex is connected with just one set of scalar attributes. But since, because of both meshes and scalar attributes can have discontinuities across vertices, scalar attributes are not limited to vertices only within a mesh. Rather, the scalar attributes are being assigned to corners of a mesh. A corner is made up of a tuple $(v, f)$ of vertices and faces. Scalar attributes at a corner $(vertex, face)$ specify the shading characteristics of face $f$

at vertex $v$. For example, Figure 2.10 shows two geometrically identical meshes. The difference lies in the crease angle. The lower the crease angle the more creases the mesh has. A crease shows a discontinuity in a mesh. The right mesh in Figure 2.10 shows one vertex normal per four faces whereas the left image shows one normal per corner. More vertex normals result in a coarser appearance with more facets visible.

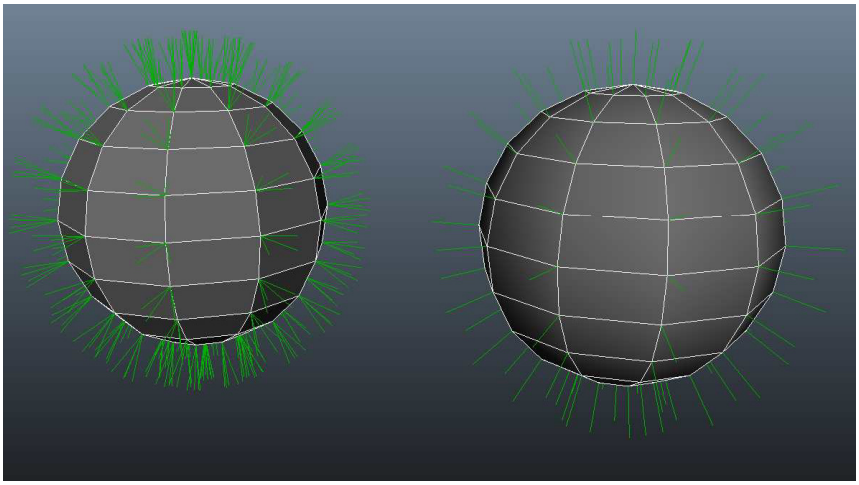

Figure 2.10: Crease Angle in Meshes

The geometrical description of a mesh is expressed as a tuple $M = (K, V, D, S)$, where $K$ is the connectivity of the mesh and $V$ specifies its geometry. Additionally, $D$ represents the set of discrete appearance attributes $d_f$, which are associated with the face $f = j, k, l \in K$ and $S$ is the set of scalar attributes $s_{v,f}$ of every corner $(v, f)$ of $K$.

The attributes $D$ and $S$ are very important for the visual appearance of the mesh. They shape up the coarseness or the smoothness of a mesh respectively. An edge $\{v_j, v_k\}$ of the mesh is called *sharp*, if it meets one of the following discontinuities:

- The edge is a boundary edge
- Two adjacent faces $f_l$ and $f_r$ have different discrete attributes (i.e. $d_{f_l} \neq d_{f_r}$)
- Adjacent corners have different scalar attribute (i.e. $s_(v_j, f_l) \neq s_(v_j, f_r)$)

Figure 2.11 shows a set of sharp edges of a three-dimensional model. The general view shows a set of *discontinuity curves* , e.g. the white curves in the left image. The pillow in the image at the right of Figure 2.11 shows the smooth edges dashed and the sharp edges in solid line style.

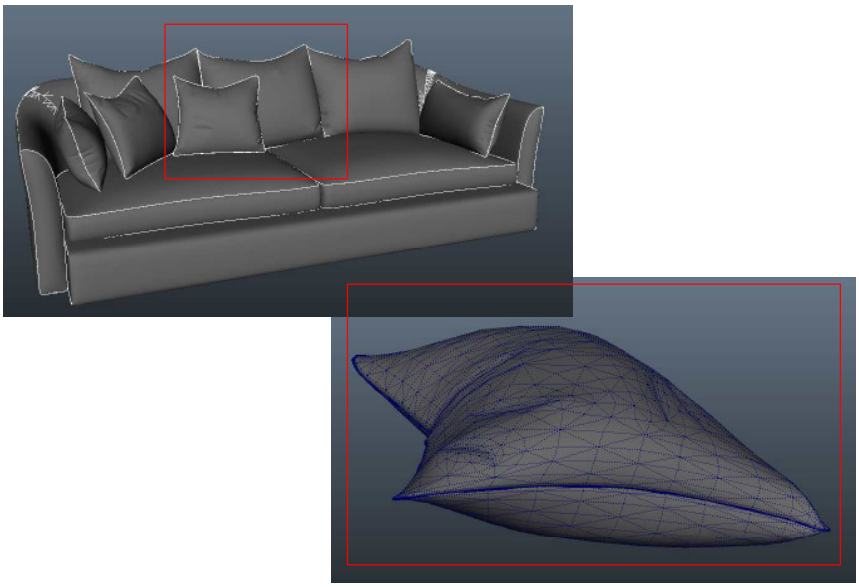

Figure 2.11: Sharp Edges

### 2.4.1 Vertex Based Data Structures

Vertex based mesh data structures consider triangles and vertices only. Edges and their relation between edges and triangles or between edges and vertices are not recorded because there is no need for this kind of information. Figure 2.12 shows the main parts of triangle vertex based data structures. There is a triangle which consists of three vertices. The edge connectivity is not explicitly stored. But due to the indexing of the triangle indices of the vertices, edges are drawn without storing vertex-edge or triangle-edge connections. The triangle vertex index arrays, which are used in the mesh data structures of Tobler and Maierhofer [42], use the numbering shown in Figure 2.12.

#### Indexed Face Set

Indexed face sets (IFS) are mainly part of the VRML 2.0 file format specification [44]. They specify a set of planar faces which are used in the coordinate system. The face set contains two main parts like seen in Figure 1:

- Coordinates
- Indexed Face Set (IFS)

The coordinate section is a node with three-dimensional vector data used in the field of vertex-based geometry like the indexed face set. The IFS represents a spatial shape (i.e. polygons) by indexing the vertices in the fields of the vertex coordinates. A index of "-1" indicates the end of one face and marks the start of the next one. The face set is indexed starting with $0$, so that the coordinate node contains $N + 1$ coordinates.

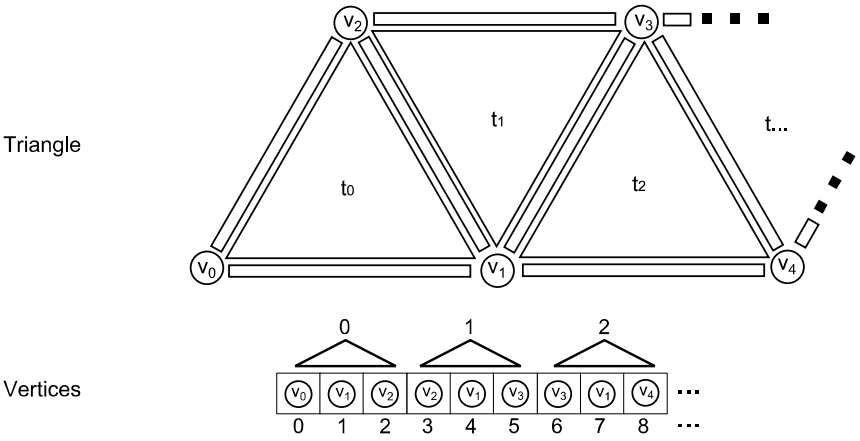


Figure 2.12: Indexed Face Set Numbering

## Kd-Tree

In computer science, a kd-tree (short for k-dimensional tree) is a space-partitioning binary tree data structure. The algorithm was introduced by Jon Louis Bentley in his paper *K-d trees for semidynamic point sets* [4]. It is a very useful tree structure for organizing points in a k-dimensional space in a simple way. The iterative pair contraction of Michael Garland [15] also focuses on the use of kd-trees. The dimension of the data is $k$ (a spatial tree is called a kd-tree of dimension 3 instead of 3d-tree). For a total of $n$ points, the space complexity is $\Omega(n)$ and the time complexity for the height is $\Omega(\log n)$ for balanced trees. The tree supports operations on range of nearest-neighbor queries. Every node consists of two child nodes, satellite informations and a key value. The key represents either a coordinate value or a pair of coordinates representing the splitting plane. The splitting plane is stored in every non-leaf-node and splits the space into two parts, known as subspaces. Every leaf node contains a k-dimensional point, which represents the points in space (for a kd-tree of dimension 3).

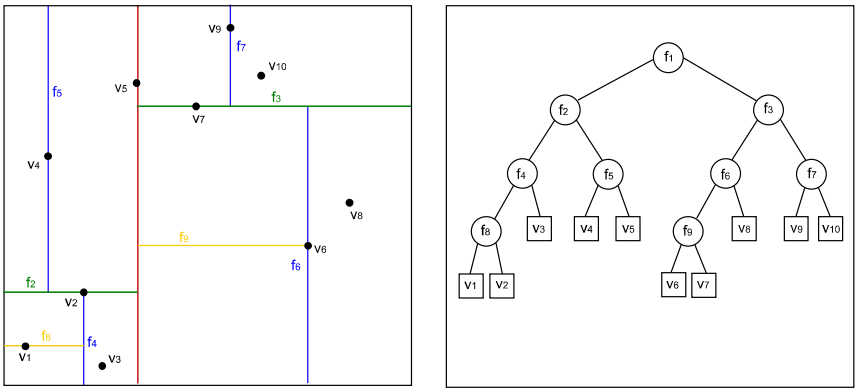


Figure 2.13: kd-tree of dimension 2

The two figures 2.13 show a kd-tree of dimension 2. The left image shows the split lines

around medians. To reach the optimal kd-tree for a fixed point set, the point set is divided into subspaces with an almost equal number of points in its left and right subtree.
The following three different searching schemes are common:

- Point search
- Region search
- Nearest neighbor search

The nearest neighbor (NN) search is used to find the nearest point in conjunction with a given input point. The localization of the point can be done very efficiently by using tree specific properties. By searching in a tree with large portions of search space can be eliminated quickly.

### 2.4.2 Edge Oriented Data Structures

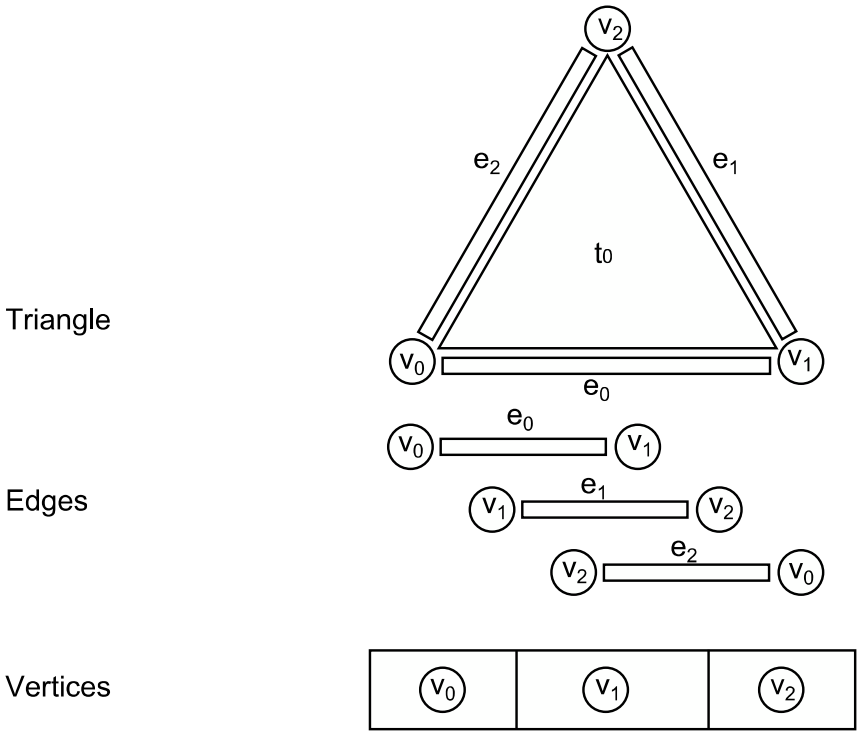


Figure 2.14: Triangle Edge Vertex Data Structure

There are several data structures for storing triangular mesh representations. For edge contraction algorithms, edge-based data structures are most efficient because nearly all information of vertices, edges and faces are stored in respect to the edges itself. Widely used data structures [10] are the *Winged-Edge Data Structure*, introduced by Baumgart

[2], the *Half-Edge Data Structure* desribed by Kettner [24], and the *Mesh Data Structure for Rendering and Subdivision* presented by Tobler and Maierhofer [42] invented at the VRVis [13].

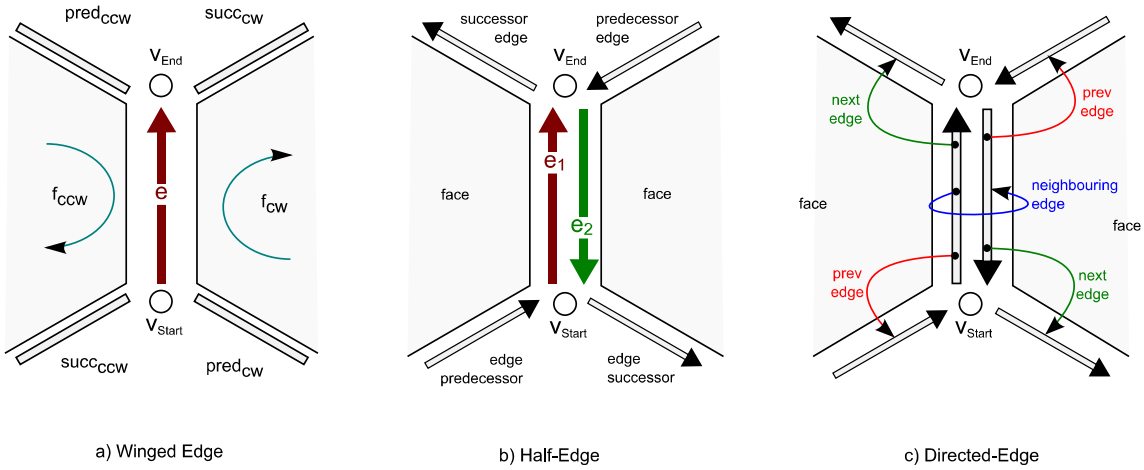


Figure 2.15: Edge-based Mesh Data Structures

The winged-edge data structure (Figure 2.15 a) is an all-round mesh data representation. In detail, it describes the geometry and topology of vertices, edges and faces, where exactly two faces share a common edge. Each entry in the edge table contains the following information: Edge name, start and end vertex, left and right face, and the successor and predecessor edges (by processing either the left or the right face). Clockwise ordering is used for traversing the adjacent edges. The left face in Figure 2.15 is denoted as counterclockwise face, because it is on the left side of edge $e$, but the ordering is clockwise. Hence, the predecessor of edge $e$ is $pred_{ccw}$ for the left face and $pred_{cw}$ for the right face. The direction of the selected edge $e$ goes from $v_{start}$ to $v_{end}$. If the direction of the selected edge is changed from $v_{end}$ to $v_{start}$, all entries except the first one in the Table 2.4.2 must be adjusted accordingly.

| Edge | Vertices | | Faces | | Left Traverse | | Right Traverse | |
|---|---|---|---|---|---|---|---|---|
| Name | Start | End | Left | Right | Pred | Succ | Pred | Succ |
| $e$ | $v_{start}$ | $v_{end}$ | $f_{ccw}$ | $f_{cw}$ | $pred_{ccw}$ | $succ_{ccw}$ | $pred_{cw}$ | $succ_{cw}$ |

Table 2.1: Ordering of a winged-edge data structure

The half-edge data structure (Figure 2.15 b) is an extension to the previously specified winged-edge structure. It is named half-edge because of the way the edges are stored within the mesh. Instead of storing the edges of the mesh, half of an edge is stored. The two half-edges are directed and make up an edge pair with opposite directions. Figure

2.15 b) shows a schematic half-edge representation of a triangular mesh, where $v_{start}$ and $v_{end}$ represent the vertices of the mesh and the red and green arrows show the half-edges. In contrast to the ordering of winged-edge, this ordering can be either strictly clockwise or counterclockwise just as long as the same convention is used in every face equally.
Each one of the two half-edges stores its index, a pointer to its incident face on the left, two indices of the previous and the next edges along the boundary of the incident face, and the index of its twin half-edge. This is oriented in reverse to the origin one.

The directed-edge data structure from Tobler and Maierhofer [42] is presented in Figure 2.15 c). This approach is a hybrid adaption of both the winged-edge and the half-edge data representation. This technique avoids the pointer data structures from the winged-edge principle. Additionally it uses the idea of the half-edge structure where an edge between two faces is interpreted as two directed edges, where each directed edge belongs to a circular group of corresponding, directed edges. There is also a link to its corresponding twin edge in the neighboring face.

### 2.4.3 Progressive Meshes

The simplification of meshes can be used for the construction of LOD representations. Therefore, a target value of triangles can be specified. By reaching the number of triangles, the algorithm will stop. While the exact number of triangles can not be reached, approximations of the desired triangle count can be achieved. Hugues Hoppe originally introduced the term *Progressive Meshes* in his paper [19]. The mesh data sets are represented in a sequence of different resolutions. Every single resolution of the mesh depends directly on the triangle count.
The construction of progressive mesh is divided into two subproblems. In the first part, called *Preprocessing*, the highly detailed original mesh $\hat{M}$ will be gradually simplified (Figure 2.16). Each step of the simplification procedure is recorded into a progressive mesh data structure. The contraction technique of edge collapse (details in Section 2.5.4) is used in the preprocessing part. With every single step of a sequence of edge collapses a transition $M_s \rightarrow M_{s-1}$ to a coarser mesh is added. The workload for this preprocessing step can be very time-consuming.

## 2.5 Survey on Mesh Simplification Algorithms

This section contains a short survey on the common triangle mesh simplification methods available nowadays. Each classified algorithm in this section is depicted with the underlying technique of the used simplification mechanism. The most important details are:

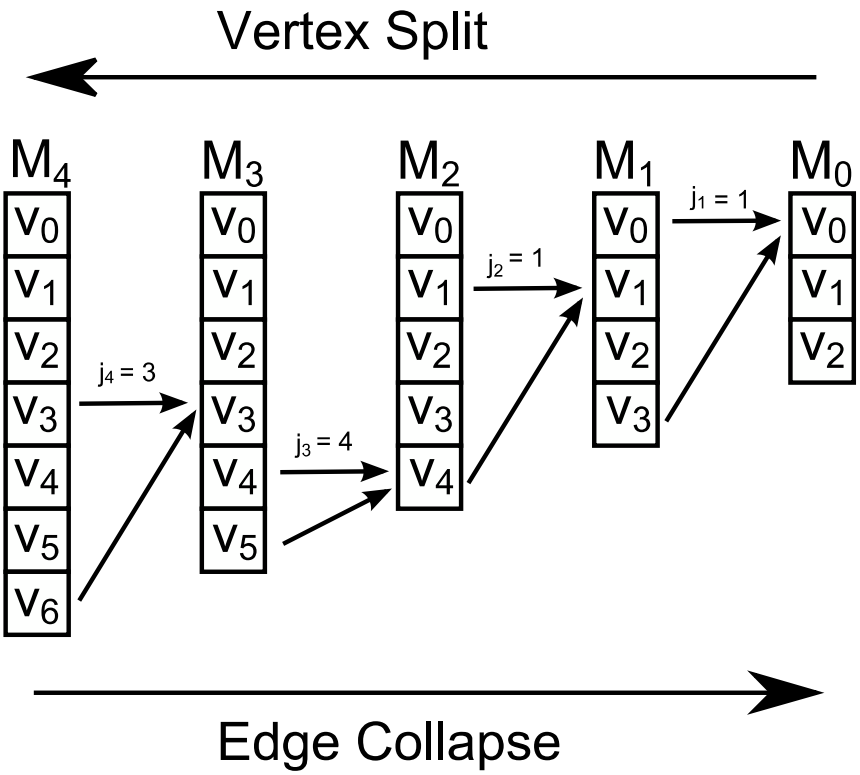


Figure 2.16: Progressive Mesh Creation

How topology is treated and which expected results will be achieved depending on the computational time. This section contains a brief catalog of the alternatives to quadric error metrics in polygonal mesh simplification.

### 2.5.1 Vertex Clustering

Vertex clustering is a very fast algorithm which works on various triangular models. With this technique, the number of vertices in a mesh is decreased. The algorithm starts with a bounding box containing the whole model. The bounding box is evenly divided into three-dimensional cells which form a Cartesian grid. The left image (Figure 2.17) shows a two-dimensional mesh with its vertices lying in different rectangular cells with the same size. The next step selects a vertex representative $v$ from the set of vertices $v_i$ in every cell. All vertices lying in the same cell are merged into a single vertex. The position of the new vertex $v$ will be determined via a heuristic, i.e. the edge length of the connected vertices, a mean value of all vertices, or a median value. A minimum of two points in the same cluster lead to the removal of triangles (Figure 2.17 shows the merge of three red vertices into one vertex and the removal of one triangle, whereas the merge of four green vertices into a single vertex removes two triangles in the cluster). The algorithm continues with the reconnection of the newly created vertices. To do this, the connections

of the new mesh depend on the connections of the original mesh. If there is a connection between two clusters in the original cluster, there will be a connection between the newly distributed vertices per cluster as well.

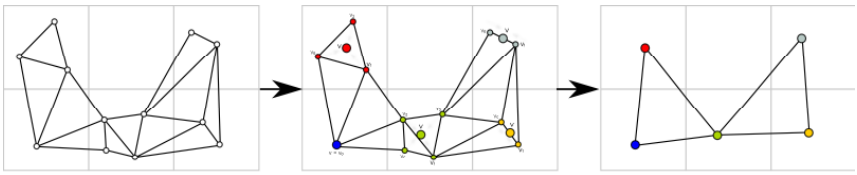

Figure 2.17: Vertex Clustering

### 2.5.2 Vertex Decimation

Apart from vertex clustering, vertex decimation is one of the most used techniques for triangular mesh simplification. The iterative simplification algorithm was originally introduced by Schroeder et al. [39]. In each step of the algorithm, a vertex (i.e. the center vertex in Figure 2.18) is selected for removal by its decimation criteria. The classification of a vertex decides whether and when it is removed in the iterative computation process. Simple vertices are weighted with an average plane, which is computed from the surrounding triangles. The normal vector of the average plane is an average normal from every triangle-normal of the triangle-fan connected to the chosen vertex. The normal distance from the vertex to the average plane is used as a decimation criterion. If the distance is less than an user-specified threshold the vertex will be removed and all adjacent triangles of the triangle fan are removed from the mesh. A threshold of zero will remove only planar regions. If the distance is larger than the threshold, the vertex will be kept.
The successful vertex removal leads to a polygonal hole in the mesh. The algorithm of Schroeder [39] describes a retriangulation of the resulting hole. The precise description of the triangulation algorithm is not discussed in this topic because it is beyond the focus of this Thesis.

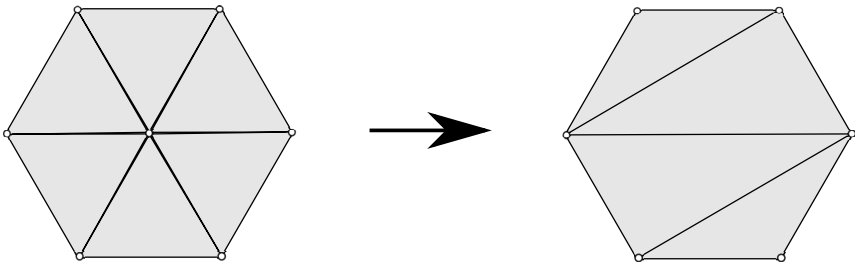

Figure 2.18: Vertex Decimation

### 2.5.3 Vertex-Pair Contraction

In contrast to edge contraction, pair contraction (Figure 2.19) is a more generalized approach. The original algorithm [15] enriches the edge contraction algorithm with the possibility to join unconnected vertices, or vertices, which do not share an edge. Thus, there is an possibility to alter the mesh topology during the simplification process. The pair contraction allows the contraction of two vertices. It is irrelevant if the vertices are topologically or just geometrically close to each other. This is a crucial enhancement especially for the support of non-manifold meshes. The situation for topologically adjacent vertices is equivalent to edge contraction in Figure 2.21. For geometrically close, but topologically unrelated vertices, unconnected regions will be merged. The pair contraction takes two vertices $v_0$ and $v_1$, where $(v_0, v_1)$ is not an edge. The distance between the two vertices must satisfy the condition, that the distance between $v_0$ and $v_1$ is lower than a user defined threshold $\tau$, denoted as:

$$\|v_0 - v_1\| < \tau$$

If $\tau = 0$, no pair contraction will be performed. To use the vertex pair contraction algorithm, the decimation threshold has to be chosen carefully. If the limit value is too high, unconnected mesh segments far away from each other will be joined by accident. This can lead to manifold meshes with a loss of the general structure of a model. In the worst case (see Figure 2.20), vertex pair contraction results in a non-manifold mesh with many holes in it and as well the general shape of the mesh is lost. The threshold of contracting unconnected vertices is almost unpredictable, but manual adjustments can avoid negative results.

The algorithms for edge contraction and vertex pair contraction require different ways of storing and accessing the data of a triangular mesh representation. The next Section 2.4.2 depicts a short survey of edge-based mesh data structures, whereas Section 2.4.1 contains an overview of vertex-based data models.

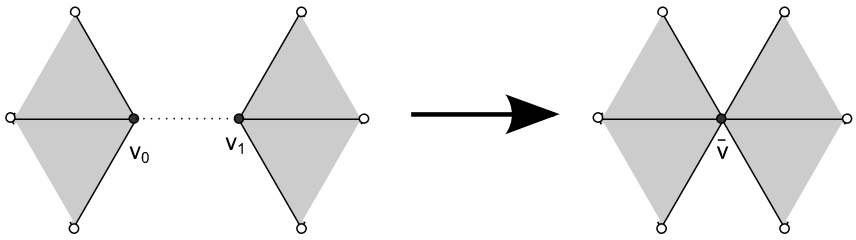


Figure 2.19: Pair Contraction Principle

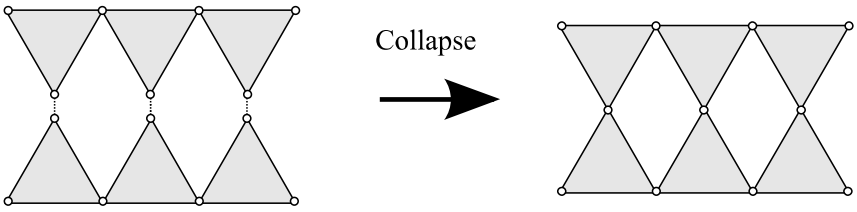


Figure 2.20: Pair Contraction - Worst case in 2D

### 2.5.4 Edge Collapse

The edge collapse algorithm is used to generate different mesh variants of different quality levels. The inverse operation of edge collapse is called vertex split (detailed information in Figure 2.16). This algorithm is based on a more generalized approach, called pair contraction. Depending on the used contraction metric, an initial mesh is taken and an edge is going to be selected. This edge will be collapsed into a single vertex. A pair of triangles sharing this edge will be degenerated and removed from the initial mesh.
The paper *A note on Edge Contraction* [5] contains a formal overview on the edge contraction algorithm based on graph theory. In Figure 2.21, the triangulated surface is going to be simplified by reducing the vertex count. Repeated edge contraction leads to a reduced set of triangles and vertices. The left part of Figure 2.21 shows the original mesh, where the edge $e_0$ (red) is going to be contracted. After the edge collapse is performed, the vertices $v_0$ and $v_1$ are merged into a new vertex $\overline{v}$, written as:

$$e_0 := (v_0, v_1) \rightarrow \overline{v}$$

The contraction of edge $e_0$ removes the edge as well as the two triangles $(v_0, v_1, v_2)$ and $(v_0, v_3, v_1)$. All incident edges of the vertices $v_0, v_1$ are being reconnected to $\overline{v}$, except the edges of the triangles $t_0$ and $t_1$.

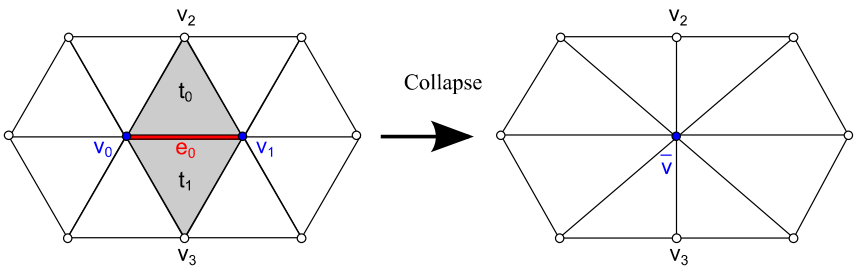


Figure 2.21: Contraction of edge $e_0$

### 2.5.5 Polygonal Re-Tiling

The polygonal re-tiling is similar to the particle decimation algorithm shown in Section 2.2.1. The re-tiling method work on polygonal surface representations but not on point clouds. The first standard re-tiling algorithm was developed by Hugues Hoppe [20]. The original algorithm was improved by Greg Turk [43] and published in a paper which describes an algorithm which re-tiles polygonal surfaces. This method is a quite optimal solution for smooth, curved surfaces. Organic models meet this precondition very well, whereas sharp edges are not handled with this algorithm. The first cycle of the algorithm distributes a user-defined number of vertices on the polygon mesh, like the particle decimation technique in Section 2.2.1. The next step deals with the repulsion forces between the vertices to get the vertices almost equally distributed. Vertices are only allowed to move in $uv$-direction on the surface. After the repelling process is finished, the remaining original vertices are decimated from the surface. The next step leads to the re-tiled model with the new vertices only.


Figure 2.22: Re-tiling polygonal meshes. Image taken from [25]

### 2.5.6 Wavelet Simplification

The wavelet compression technique was previously used in the area of 2D image compression. Both the the recent JPEG2000[4] standard and the MPEG4[5] still image coder uses wavelets for compressing images. The main difference between 2D images and spatial meshes using wavelet compression is that images are sampled on a regular grid whereas the vertices and their connections in a mesh are distributed irregularly.

[4] Joint Photographic Experts Group
[5] Moving Picture Experts Group

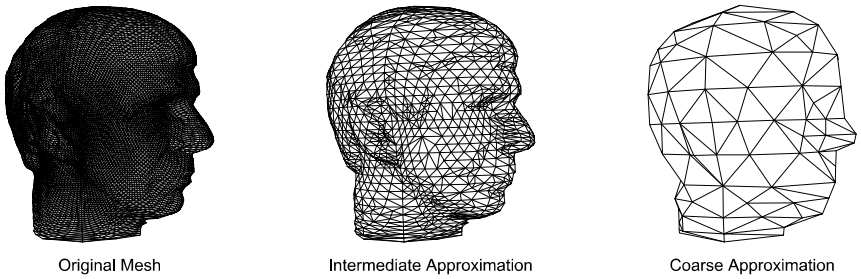


Figure 2.23: Subdivision Surface Wavelets. Image taken from [40]

## 2.6 Summary

This chapter explained an introduction on the topic of mesh simplification in computer graphics. Alternative mesh simplification algorithms and their benefits and trade offs were discussed. The terms necessary to understand both polygonal and non-polygonal mesh representations were defined, and a survey of the common simplification techniques was given. More information on the topics to create much coarser meshes with preserving important details in computer graphics can be found in the paper *A Survey of Polygonal Simplification Algorithms* of David Luebke [27].

Chapter 3

# Theory

The research field of computer graphics requires some mathematical background to understand. The theory of mesh simplification needs some introduction in geometry. Furthermore, it requires theories on mesh topologies, graph theory, and abstract data types. Because of the interconnectivity between geometry, graph and topological theory, the introductory section is not explicitly subdivided into each one of this theories separately.

## 3.1 Geometric Primer

This section discusses geometric primitives in general. It starts off with the introduction of geometrical simplex representations. A simplex in geometry is a generalized notion of an object in its arbitrary dimension. Especially, a $n$-simplex in math is a $n$-dimensional polytope which is a geometric object with planar sides. A $n$-simplex represents is the convex hull of its $n + 1$ vertices. For example, an edge is a 1-simplex with two (i.e. $n + 1$ vertices). A regular $n$-simplex is constructed from a regular $(n - 1)$-simplex by connecting a new vertex to all original vertices. Table 3.1 contains an overview of the simplices used in this work.

| Simplex | Name | vertices | edges | faces | cells | fig |
|---|---|---|---|---|---|---|
| 0-simplex | point | 1 | | | | 3.1 a) |
| 1-simplex | line segment | 2 | 1 | | | 3.1 b) |
| 2-simplex | triangle | 3 | 3 | 1 | | 3.1 c) |
| 3-simplex | tetraeder | 4 | 6 | 4 | 1 | 3.1 d) |

Table 3.1: Geometric Simplices

The next Sections show a short introduction into the geometric simplices used in the thesis. The simplices are used in conjunction with the theory of graphs to form a polygonal/triangular mesh data structure. Details on how the simplices are going to be connected can be found in Chapter 3.2.

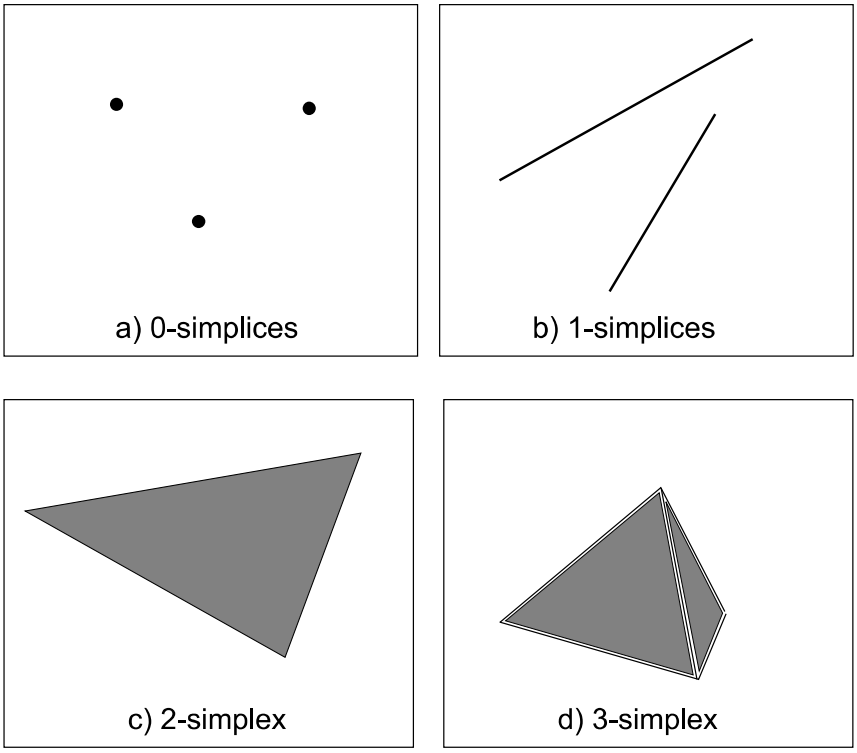


Figure 3.1: Simplices in Geometry

### 3.1.1 Vertex

A vertex (plural vertices) in geometry describes a special point representation of a mathematical object. It describes the intersections or corners of geometric models or shapes. Vertices are widely used in computer graphics. They define the corners of surfaces (i.e. triangles) or knots of mesh sets. Each such point is given as a vector. For example, a square has four corners (i.e. vertices). Vertices are most commonly used in polygons, polyhedra and graphs.

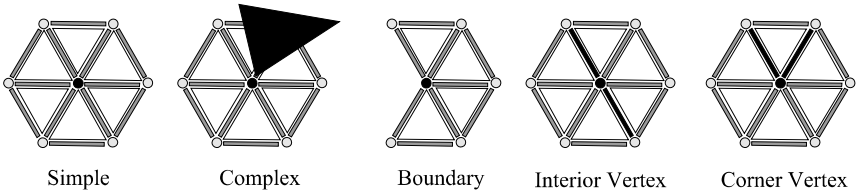


Figure 3.2: Vertex occurrences in meshes

### 3.1.2 Polygon

The definition of polygons is not easy to formulate, because a precise definition generally depends on the context it is used in. In computer graphics, a polygon is defined as a

geometric object consisting of a number of vertices and the same number of line segments, called edges. In another definition, a polygonal curve is a finite chain of line segments, called edges. Their endpoints are called vertices. A polygon is a closed figure with straight sides. Segments like arcs or bezier patches between vertices are not legal for polygonal objects. The vertex set is cyclically ordered in a plane, where no three adjacent vertices of the vertex set are collinear.

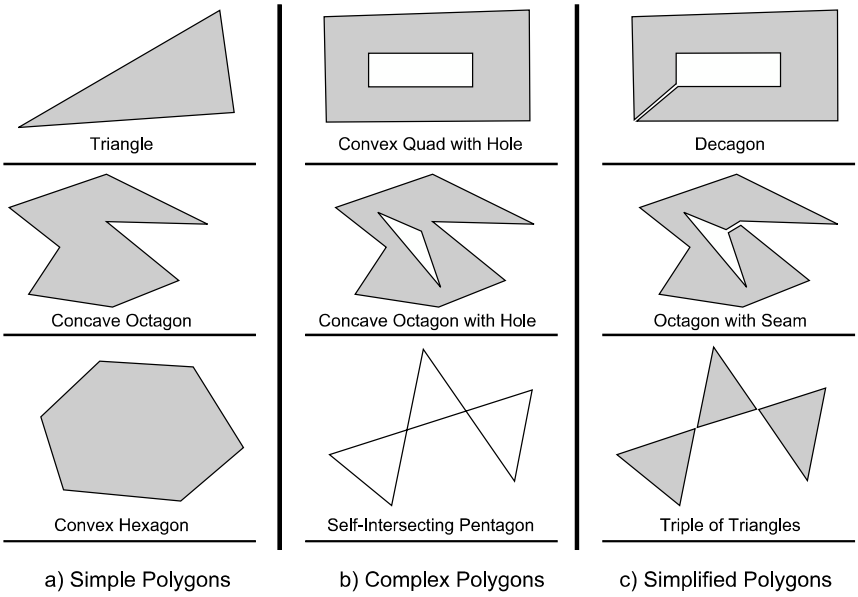


Figure 3.3: Simple - Complex - Simplified Polygons

### Convex & Concave

In geometry, polygons are divided into convex and concave polygons. A convex polygon which is shown in the left of Figure 3.4 is simple and meets the following two properties for convexity:

- Every internal angle is less than 180 degrees
- Every straight-line segment inside the polygon remains inside its boundaries

A simple polygon is strictly convex if each internal angle is less than 180 degrees. A polygon which is not convex is called concave. At least one interior angle in concave polygonal shapes will always have an angular sum that is greater than 180 degrees (details in the right shape of Figure 3.4). A concave polygon can be cut into a set of convex polygons. Chazelle & Dobkin [7] describe a polynomial-time algorithm for decomposing concave polygons into as few convex polygons as possible.

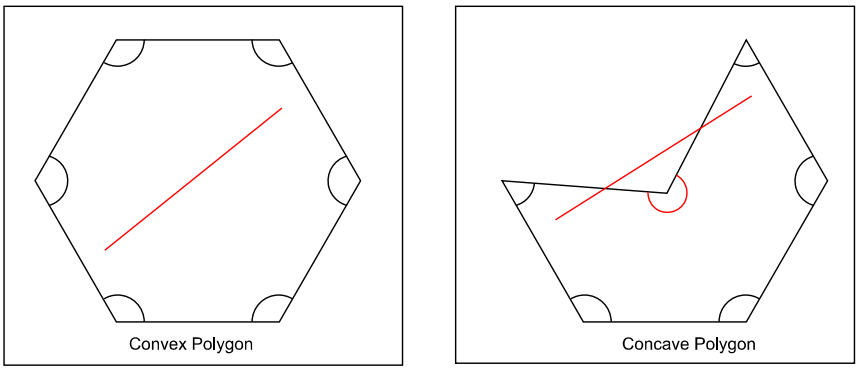


Figure 3.4: Convex - Concave Polygons

### Simple & Complex

Figure 3.3 a) shows some simple polygons. A simple polygon is a closed polygonal curve without self-intersection and without "holes" in it. A polygon divides the plane into three parts: The Interior area, the exterior area and the boundary itself. A complex polygon can contain holes (Figure 3.3 b). A simple polygon is defined by a circular enumeration of its vertices. The enumeration defines the front side and the back side of the polygon. In right-handed coordinate systems, the normal vector of a counter-clockwise oriented face points in direction of the viewer. Whereas clockwise orientation is used in the left-handed environment.
Complex polygons in computer graphics are neither convex nor concave. A complex polygon can be grouped in one of the the following:

- Self-Intersecting: Includes star polygons like the self-intersecting pentagon in Figure 3.3 b)
- Polygon with Holes: Has a boundary comprising discrete circuits in Figure 3.3 b)

It is possible to convert complex polygons into simple ones by adding pairs of edges. The resulting seam edges (Figure 3.3 c)) thus lead to new polygons.

## 3.1.3 Quadrilateral

In euclidean geometry, a quadrilateral (also called quadrangle) is a polygon with four sides and four corners. Quadrilaterals are a specialization of polygons. Figure 3.5 shows the three different occurrence types of quadrangles. The type of a quad can either be convex, concave or self-intersecting (e.g. bow-ties or butterflies).
A quadrilateral is called planar, if all four vertices of its corners lie in a plane. A quadrilateral is called a trapezoid, if two sides are parallel. A quad is a parallelogram if the

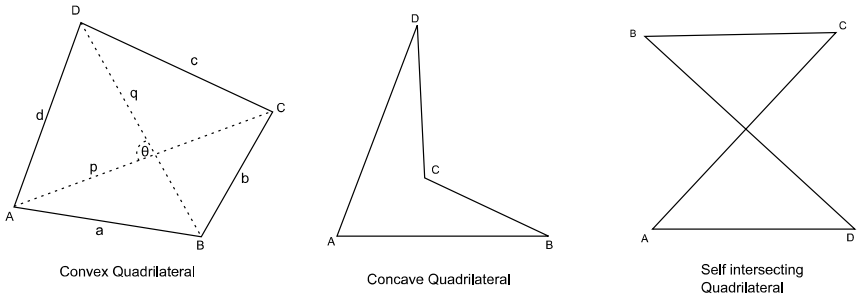


Figure 3.5: Different Quadrilaterals

opposite pair of sides are parallel. For a planar convex quad (left of Figure 3.5), let the lenghts of the sides be $a, b, c$, and $d$, the polygon diagonals $p$ and $q$. The diagonal $p$ is normal to the diagonal $q$, if and only if

$$a^2 + c^2 = b^2 + d^2 \tag{3.1}$$

The sum of the squares of side lengths denotes the equation 3.2, where $x$ is the length of the line joining the midpoints of the polygon diagonals.

$$a^2 + b^2 + c^2 + d^2 = p^2 + q^2 + 4x^2 \tag{3.2}$$

If all corner points are non-planar, the quadrilateral is called a skew quadrilateral (Figure 3.6) with facial discontinuities.

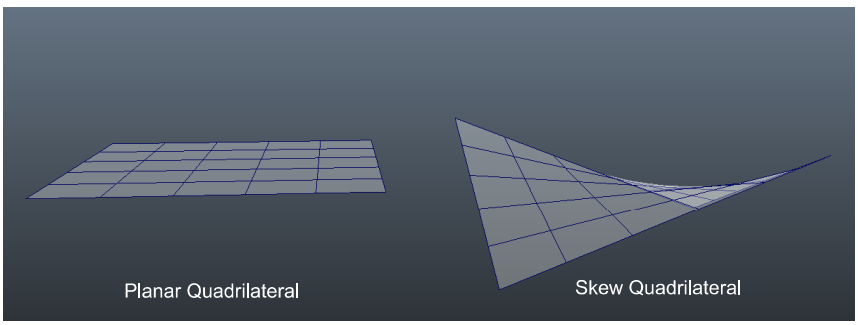


Figure 3.6: Planar - Skew Quadrilateral

### 3.1.4 Triangle

Triangles are of fundamental importance in geometry, modeling and graphics. A triangle is one of the basic surfaces (i.e. a three-sided polygon) in geometry. Surfaces of complex three-dimensional objects, such as characters or cars, are approximated with many triangles. The angles of a triangle in euclidean space always sums up to 180 degrees. So, the third angle of any triangle with two given angles can be computed. A group of connected

triangles is called triangle mesh. Meshes in general and triangle meshes in particular are the topic of Section 3.2.

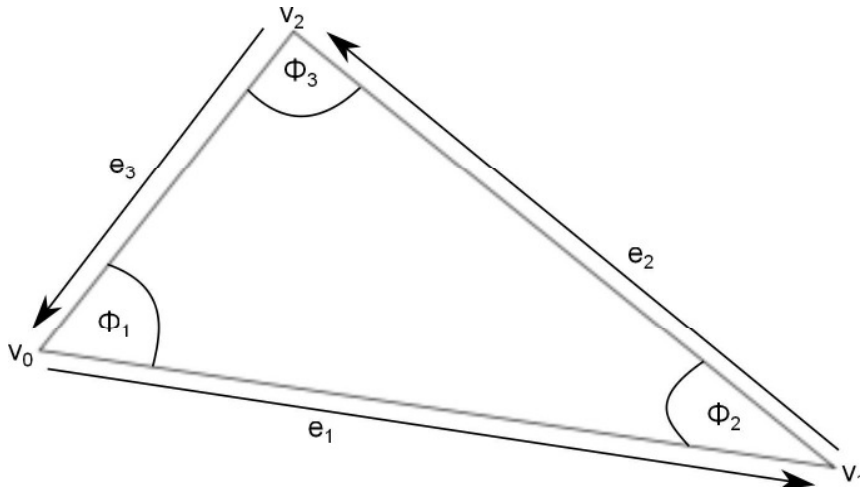


Figure 3.7: Simple Triangle with Labels

A triangle consists of three vertices as corner and three sides or line segments, called edges. Figure 3.7 shows a triangle consisting of the three vertices $\mathbf{v}_0, \mathbf{v}_1$ and $\mathbf{v}_2$ (Figure 3.7). The figure also shows the labeled interior angles, counter-clockwise edge vectors $\mathbf{e}_i$ and the side lengths $l_i$. In euclidean geometry, any combination of three non-collinear vertices results in a unique plane and determines a unique triangle. The plane equation of the triangle is important in many applications. The present thesis also takes up the plane equation which will be discussed in Section 3.1.4.

### Area of a Triangle

For determining the contraction candidates in Section 3.4, the triangle area is also taken into account. There are several techniques for computing the area of a triangle. The best-known method is the calculation of the area from base and altitude (i.e. the height). The area of a parallelogram (left illustration in Figure 3.8) is equal to the product of base and height. Because the triangle takes exactly one-half of this area, the area of a triangle is

$$A = \frac{1}{2}bh \tag{3.3}$$

In 3D the cross product is used to compute the triangle area. The magnitude of the cross product of two vectors $\mathbf{e}_0$ and $\mathbf{e}_1$ is equal to the occupied space of the parallelogram. The right image in Figure 3.8 shows the calculation of the triangle area. Given two edge vectors, the triangle area equation is given by:

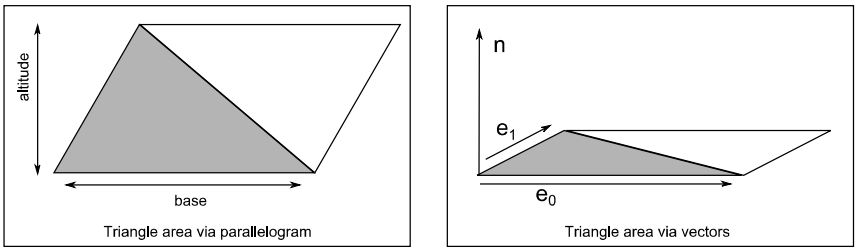


Figure 3.8: Triangle Area Calculation

$$A = \frac{\| \mathbf{e}_0 \times \mathbf{e}_1 \|}{2} \tag{3.4}$$

### Plane Equation

A plane divides the three-dimensional space into two regions, also known as half spaces. Geometric properties or error metrics are often calculated using planes. This work uses the plane equation for the calculation of the contraction candidates using quadric error metrics. Since the algorithm focuses on triangles, the plane equation is derived from triangles.
The standard equation of a plane in three-dimensional space is defined by following Equation 3.5:

$$Ax + By + Cz + D = 0 \tag{3.5}$$

Where $A$, $B$ and $C$ indicate a vector perpendicular to the plane (i.e. plane normal). $D$ is the shortest distance from a point $p_1 = (x_1, y_1, z_1)$ to the plane. If and only if the point $p_1$ lies in the plane, the distance $D = 0$.
An alternative 3D plane equation is the *Hesse Normal Form* in equation

$$n^\top v + d = 0 \tag{3.6}$$

Where $n = [a, b, c]^\top$ is the normal vector of the plane, $d$ is a distance constant and $a^2 + b^2 + c^2 = 1$ is the unit normal vector of the plane.

### Distance to Point

The distance from any point to a plane is an important tool to measure how curvy a set of triangles can be. The distance between a point $P = (x_1, y_1, z_1)$ to a plane is determined by the normal vector $\mathbf{N} = (A, B, C)$ and a fixed point $Q = (x_0, y_0, z_0)$ which lies on the plane.

Let $\mathbf{v}$ be the vector from $Q$ to $P$. The distance of $P$ to the plane is simply the length of the projection of $\mathbf{v}$ onto the unit normal vector $\mathbf{n}$. Since $\mathbf{n}$ is length one, the distance is simply the absolute value of $\mathbf{v} \cdot \mathbf{n}$. The distance is labeled as $d$ in Equation 3.9.

$$\begin{aligned} d &= |v \cdot n| && (3.7)\\ &= |(x_1 - x_0, y_1 - y_0, z_1 - z_0) \cdot \mathbf{n}| && (3.8)\\ &= \frac{A(x_1 - x_0) + B(y_1 - y_0) + C(z_1 - z_0)|}{\sqrt{A^2 + B^2 + C^2}} && (3.9) \end{aligned}$$

## 3.2 Triangle Meshes

By definition, a triangular mesh $M = (V, F)$ is a pair, which contains of a list of vertices $V$ and a list of triangular faces $F$. Both the vertex list $V = (\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_m)$ and the faces list $F = (f_1, f_2, ...f_n)$ are an ordered set of index numbers. Each index of every list is accessible with an unique integer. Each vertex (in Figure 3.10) is defined as a column vector in the Euclidean space $\mathbb{R}^3$.

$$\mathbf{v}_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \tag{3.10}$$

Each triangular face (in Item 3.11) in the model consists of three integers representing the three indices of the corners $(\mathbf{v}_j, \mathbf{v}_k, \mathbf{v}_l)$ of $f_i$.

$$f_i = \begin{pmatrix} j & k & l \end{pmatrix} \tag{3.11}$$

To describe a triangle mesh, two distinct types of information are required;

- Geometry Information: the spatial information of all the vertex coordinates
- Topological Information: the connectivity of vertices to form edges and faces

The previous Section 3.1 showed an overview on the geometric or locational information of triangle meshes.

### 3.2.1 Mesh Topology

Topological information describes the connectivity of a mesh. In computer graphics, *topology* means the connectivity of vertices to an edge or to one or more triangles. A good mesh representation has to be space efficient. In addition, if the topology of the

mesh is going to be changed (e.g. in a 3D modeling packages) where polygons have to be split, merged, removed or inserted, there must be efficient access to the following kinds of information:

- All vertices around vertex (independent set)
- All edges of a face (strictly three for triangles)
- All vertices of a face (strictly three for triangles)
- All faces around a vertex (open fan / closed fan)
- All edges around a vertex (open fan / closed fan)
- Faces of an edge on both sides (boundary / non-boundary)
- Vertices of an edge (both vertices)
- Find faces with given vertices (triangle fan)

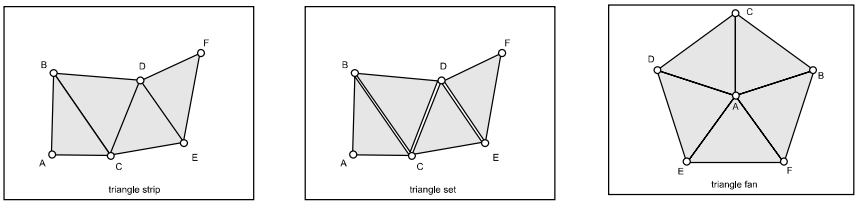


Figure 3.9: Different Mesh Topologies

There is a difference between local and global topology. The left image in Figure 3.9 shows a set of triangles. The local topology of every vertex is a simplicial complex triangle. The global topology of a triangle mesh is either a triangle strip or a triangle fan, which will be discussed in the next passage.

### Triangle Strip

In the left image of Figure 3.9, a triangle strip is shown. Triangle meshes consist of many connected triangles which form a smooth surface. One advantage of representing the connectivity between triangles is the use of triangle strips. Through the use of triangle strips the code efficiency increases. The first triangle is defined with three vertices. Each adjacent new triangle can be defined by only one additional vertex. Two vertices of the previous triangle are shared. The use of triangle strips decreases the size of data which has to be stored. The number of vertices to be stored is reduced from $3N$ to $N + 2$, where $N$ is the number of triangles to be drawn. The triangle sequence in the figure can be simply stored as $ABCDEF$.

#### Triangle Set

A triangle set is in contrast to the ordered triangle strip a bunch of triangles without global topology. The center graph in Figure 3.9 shows a triangle set similar to the triangle strip. The triangles are treated as four separate triangles: $ABC, CBD, CDE,$ and$EDF$. The number of vertices to be stored is $3N$. This leads to increased disk usage compared to the triangle strip.

#### Triangle Fan

A triangle fan (shown in the right of Figure 3.9) is similar to a triangle strip. In contrast to a triangle strip, each triangle in the fan share a single vertex. The calculation of the error metrics for vertices in Chapter 3.4.3 uses this technique. Therefore a fast mesh representation is required to start at a given vertex and cycle through the independent set of all adjacent triangles.

### 3.2.2 Mesh Properties

Every triangle mesh consists of one or more properties which can be divided into geometry and optional added to pure geometry.
The first part in Section 3.2.2 is about geometric conditions in triangle meshes, which must be met. A proper topology of vertices, edges and faces is mandatory for achieving good results. It is also very important how vertex normals and face normals are set to show or hide mesh details or get a more smoothed surface.
The second part in Section 3.2.2 is about properties beyond simple geometry. The additional use of normals, colors, and textures helps to maintain an high accuracy of the approximated model in contrast to the original object.

#### Mesh Conditions

Due to the simplicity and effectiveness of triangles in storage, the support for hardware acceleration and the ease of use in productive environments are the main reason for the wide range of supported applications. The polygons in a mesh are usually triangles. There is the possibility to mix triangles and quadrilateral primitives in a single mesh, but this document only focuses on pure triangular meshes. As stated in [37], a triangle mesh must satisfy the following conditions:

a) Each vertex must be shared by at least one edge
(no isolated vertices allowed)

b) Each edge must be shared by at least one face
(no isolated edges or poly-lines allowed)

c) At the intersection of two faces, the vertex or edge of intersection must be part of the mesh. (an edge of one face may not be in the interior of another face)

Figure 3.10 shows a collection of meshes, which do not satisfy the conditions above.

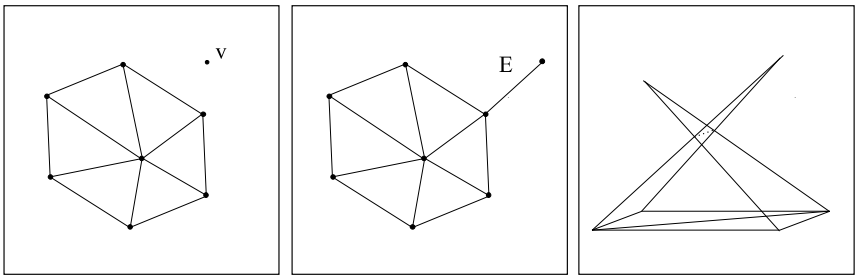


Figure 3.10: Illegal Mesh Conditions

### Mesh Attributes

Besides simple geometry, surfaces can have several additional properties, such as normals, colors, or texture coordinates [37]. These data must also be considered at the simplification process. Each vertex in addition to its position in space has some values associated which describe extended properties. Along with the geometric position, these values will be interpolated over every triangle of the object. These properties have to be continuously and cannot be restricted to a discrete set.

- In case of using color attributes in Goraud-shading, each vertex has a corresponding color value $[rgb]^T$ where $0 \leq r, g, b \leq 1$.
- By using normals, each vertex has one or more normals assigned, where each normal has the value $[a, b, c]^T$
- Texture coordinates are also correlated to vertices. They are defined in an two-dimensional space with the value $[s, t]^T$

## 3.2.3 Simplicial Complex

Let $\mathcal{K}$ be a finite set of simplices in $\mathbb{R}^m$. If these simplices are nicely fitted to each other, $\mathcal{K}$ is called a *simplicial complex*. The term *nicely fitted* means in detail:

- Any arbitrary face of a simplex of $\mathcal{K}$ belongs to $\mathcal{K}$

- The intersection of any two simplices $\sigma_1, \sigma_2 \in \mathcal{K}$ is either empty or a face of $\sigma_1$and$\sigma_2$.
- An empty set of simplices in $\mathcal{K}$ is an abstract simplicial complex, i.e. a complex without associated geometry.

The construction of simplicial complexes by connecting points, line segments and triangles in a topological space can lead to manifold or non-manifold meshes. The proposed algorithm works on both manifold and non-manifold surfaces.

### Manifold

Surfaces in geometric sense can either be manifold or non-manifold. Michael Garland [14] depicts in his thesis that manifold surfaces are unable to distinguish from a planar surface. In other words, a *closed manifold* is a surface where each point has a neighbor. The resulting surface is topologically equivalent to a disk or a half-disk.

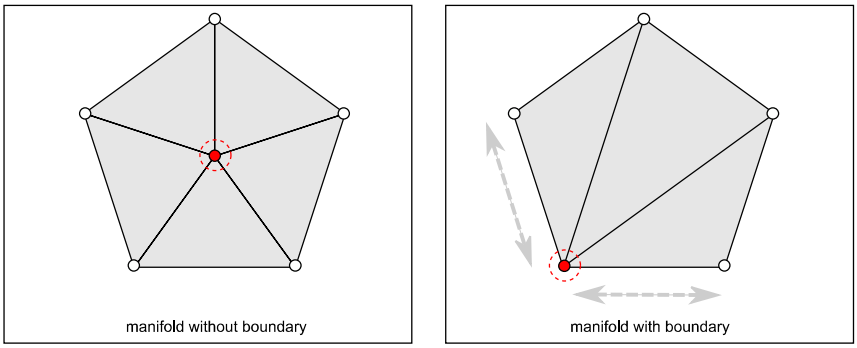


Figure 3.11: Manifold surfaces

A polygonal surface is a *closed manifold* (Figure 3.11) if every edge (connected with the red vertex) has exactly two incident faces. Also the surrounding faces of the red vertex have to be a closed loop of faces (i.e. a triangle fan). The right graphics in Figure 3.2.3 shows a manifold surface with boundary. For the closed part of the fan, the same rules as for closed manifolds are valid. Whereas at the boundary, every edge has exactly one adjacent face.

### Non-Manifold

Non-manifold meshes are meshes with irregularities in the topology. Figure 3.12 show two different cases of non-manifold topology. The left image shows a non-manifold vertex which is a vertex that, if removed, separates the boundary in two different parts. The non-manifoldness can be solved by splitting the single vertex in two different vertices with the same coordinates. After that every triangle has its own boundary.

The right image in Figure 3.12 shows a very simple example of a non-manifold edge. The edge is sharing three faces instead of sharing just two faces like in the manifold case. This case can also be solved by splitting the single edge in two different edges. Then each of the two edges suffice the properties which are necessary to meet the manifold conditions.

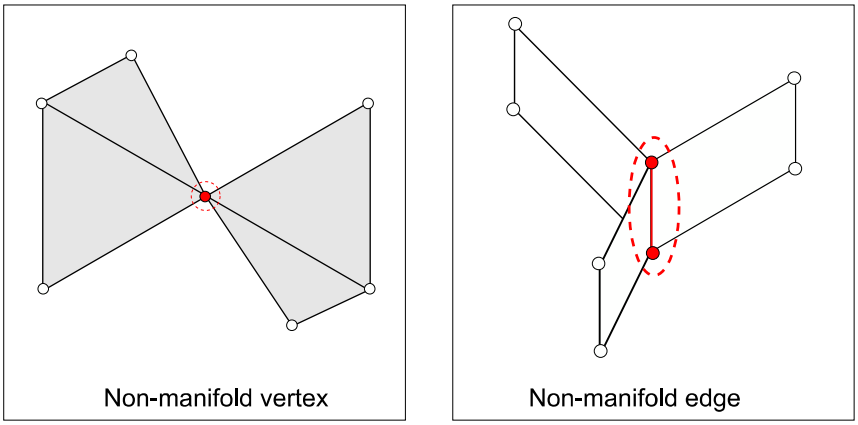


Figure 3.12: Non-manifold surfaces

## 3.3 Simplification Algorithm by Michael Garland

The simplification of polygonal models is the key part of this work. This thesis refers primarily to the PhD thesis "Surface Simplification using Quadric Error Metrics" from Michael Garland [14]. The basic algorithm takes a initial mesh $(M_{in})$ as input and computes a simplified model $(M_{out})$ as output. The input mesh has to be triangulated for this algorithm. Any other meshes, like polygonal meshes or quadrilateral meshes, must be converted to a triangular one. In the resulting mesh $(M_{out})$, the topology must not be preserved strictly. The presented concept is able to join unconnected regions and can close topological holes in meshes on demand.

### 3.3.1 Original Algorithm

The original algorithm focuses on the simplification of polygonal models, where the polygons are treated as triangles only. If the mesh is just available with polygonal or quadrilateral primitives, it has to be triangulated in a preprocessing step. The quality of the mesh topology is fundamental for the reliability of the decimation algorithm. For example, if the corners of two faces meet in one point, it should be defined as sharing just one vertex instead of representing two vertices coincident in space.
The original algorithm [15] is based on iterative vertex pair contraction. At computation time of the algorithm, the geometric error of each vertex of the current model is approximated and updated. The geometric error is represented as a quadric error metric. The

edge contraction algorithm is considered as a specialization of the vertex pair algorithm. The algorithm presented here is able to merge unconnected regions of a model using a threshold because object topology is not much to care for, except for medical imaging, where the preservation of topology is crucial. Previous simplification algorithms assumed, to use manifold surfaces explicitly. This surface algorithm can deal with both manifold and non-manifold surfaces. In case of joining unconnected regions, the algorithm will always produce non-manifold surfaces. More details on manifoldness are given in Section 3.2.3.

#### Algorithm Overview

1. Compute quadric error $Q$ for all triangles
2. Compute quadric error $Q$ for every vertex with surrounding faces
3. Choose valid pairs/edges
4. Compute optimal costs and targets
5. Sort pairs in heap where root is the minimal cost
6. Repeat until desired approximation is reached
   (a) Remove least cost pair from heap
   (b) Contract this pair
   (c) Update costs of all incident faces

### 3.3.2 Enhanced Algorithm

In contrast to the original algorithm in Section 3.3.1, Garland introduced an enhanced algorithm in *Simplifying surfaces with color and texture using quadric error metrics* [16]. In contrast to the base algorithm the new algorithm also takes normals, colors and texture coordinates into account. The following table shows an overview of the memory occupied by using additional information per vertex:

| model type | vertex | **A** | # coefficients |
|---|---|---|---|
| geometry only | $[x\ y\ z]^T$ | $3 \times 3$ | $\binom{5}{2} = 10$ |
| geometry + 2D texture | $[x\ y\ z\ s\ t]^T$ | $5 \times 5$ | $\binom{7}{2} = 21$ |
| geometry + color | $[x\ y\ z\ r\ g\ b]^T$ | $6 \times 6$ | $\binom{8}{2} = 28$ |
| geometry + normals | $[x\ y\ z\ a\ b\ c]^T$ | $6 \times 6$ | $\binom{8}{2} = 28$ |

## 3.4 Contraction Metrics

The contraction candidates for the edge contraction algorithm are determined in several ways. There are different techniques available for selecting edges going to be contracted which differ in speed, memory, the expected and real output quality, and the complexity of the used algorithm:

- Random Selection
- Edge Length
- Quadric error metrics

The edge selection of the first three methods is rather simple. The random selection presented in Section 3.4.1 demands no cost function at all. The determination of the cost per edge length is a simple local criterion only. The weighting of the triangle area is similar to the previous one. Weighting the edge contraction with quadric error metrics needs more calculation because the surrounding faces are taken into account. The following Sections depict the differences, assets and drawbacks of every single edge metrics technique. Both the resulting quality and the processing time taken will be discussed in Chapter 5.

### 3.4.1 Random Selection

The edge selection by chance is a very simple algorithm based on random edge selection which performs very fast but the desired qualitative result is undefined. This technique is used for runtime comparisons only because neither global nor local mesh characteristics are taken into account. The algorithm uses a source of pseudo-random numbers. The behavior and the resulting output can vary every time the algorithm is run. Therefore no correct analysis for a correct running time can be given. But approximative results are sufficient in this work. The pseudo-code for this operation is shown in Figure 3.4.1:

```
1: procedure RANDOM SELECTION(Input Mesh)
2:     while target face count not reached do
3:         edges[] ← Random(0, edgeCount)
4:         for all edge in edges[] do
5:             Select Valid Edges
6:         end for
7:         Contract Edges
8:         Rebuild Mesh
```

9: **end while**
10: **end procedure**

### 3.4.2 Edge Length

Stan Melax presents in his Article *A Simple, Fast and Effective Polygon Reduction Algorithm* [30] a formula for computing the edge cost metric by taking the edge length and the angle between its adjacent faces into account (Figure 3.13). According to Equation 3.12 the symbol $e$ means the actual edge; $f_0$ and $f_1$ are its adjacent neighbor faces.

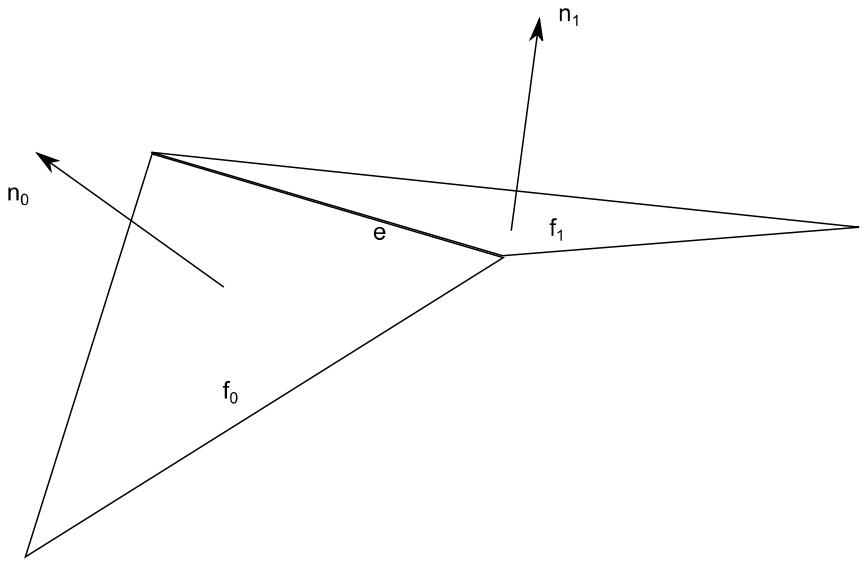


Figure 3.13: Principle of the edge-length metric

Under the terms of the Equation 3.12, longer edges will have higher costs than shorter ones. A larger discontinuity at the edge between its adjacent faces means larger difference in the face normals, resulting in smaller dot products. If there is a right angle between the faces, the fraction becomes zero and the cost becomes infinite. The lower the cost of an edge the more likely it will be contracted first. The pseudo-algorithm 3.4.2 shows the algorithm in detail:

$$cost(e) = \|e\| \frac{1}{|f_0 \cdot f_1|} \tag{3.12}$$

1: **procedure** EDGE LENGTH METRIC(Input Mesh)
2: **while** target face count not reached **do**
3: **for all** edge in edges[] **do**
4: $cost[edge] \leftarrow Length(edge)$
5: **end for**
6: Create ascending permutation of Costs

```
7:          for all cost in costs[] do
8:              Select Valid Edges
9:          end for
10:         Contract Edges
11:         Rebuild Mesh
12:     end while
13: end procedure
```

### 3.4.3 Quadric Error Metric

The quadric error metrics algorithm is fast and produces good quality. The previous algorithms are quite a bit faster, but there is always a trade-off between runtime efficiency and output quality. As the expected result has to be a suitable approximation of the original shape, the algorithm presented here is the best choice. Because of the knowledge that each triangle represents a plane we use the plane equation to calculate the squared distances from a point to the plane. Each vertex $v_i$ also has a set of $P_i$ of faces incident to it and each face (i.e. triangle) defines a plane $P$.
The derivations for quadrics are:

- Plane $P = [A, B, C, D]^T$
- Vertex $v = [x, y, z]^T$

The coefficients $A, B, C$ and $D$ of $\mathbf{K}_p$ are the coefficients of the plane equation in Section 3.1.4. The variables $A, B, C$ are given by its normal whereas the variable $D$ can be calculated by inserting a point $p = [x, y, z]^T$ into the plane equation to solve d: $d = -(Ax + By + Cz)$. The error of every vertex $v_i$ in the triangle mesh is defined as the squared distance sum of $v$ to its related planes $(v)$. The error metric for each vertex can be described as in Equation 3.13. The fundamental error quadric for every triangle is defined in Equation 3.14.

$$\Delta(v) = \sum_{p \in planes(v)} p^T v^2 = \sum_{p \in planes(v)} v^T (pp^T) v = v^T ( \sum_{planes(v)} K_p) v \tag{3.13}$$

where $p$ is the plane which is constructed by $v$ and its related triangle set and $K_p$ is the fundamental error quadric of $p$. The following section shows the calculation of the error quadric $Q$ using the fundamental quadric $K_p$.

$$K_p = pp^T = \begin{bmatrix} A^2 & AB & AC & AD \\ AB & B^2 & BC & BD \\ AC & BC & C^2 & CD \\ AD & BD & CD & D^2 \end{bmatrix} \tag{3.14}$$

### Calculating the Quadric Q

After the fundamental quadric $K_p$ has been calculated for every face, all incident faces for every vertex $v_i$ must be found. For each triangle $f_i$ it is necessary to calculate the so called *fundamental error quadric* $\mathbf{K}_p$ *of p* [15] which is used to calculate the squared distance between any point in space and the plane of the underlying face. With

$$Q = \sum_{p \in planes(v_i)} \mathbf{K}_p \tag{3.15}$$

the quadric error of $v$ evaluates to

$$\Delta(v) = v^T \mathbf{Q} v \tag{3.16}$$

It is important to mention that the error of any inital vertex equals to zero ($\Delta(v) = 0$), since every initial vertex is the intersection point of its related triangles (see Paragraph 3.2.1 for details). Before the contraction costs can be calculated it is important to know what sort of contraction should take place:

- $(v_i, v_j)$ is an edge (Edge contraction)
- $||v_i - v_j|| < \tau$, a threshold (Vertex Pair contraction)

  $\tau = 0$ using edge contraction only

  $\tau > 0$ can connect disjoint regions

The performance of each contraction technique (or the mixture of both) depends on the underlying data structure. The edge contraction technique needs an edge-based data structure only where the edges and adjacent faces/vertices can be easily located. The vertex pair contraction needs an underlying kd-tree to retrieve the nearest neighbor of a vertex. If the distance from the current vertex to the nearest neighbor is lower than the given threshold, the vertex pair is being decimated into a single vertex. This work only focuses on edge contraction. Therefore, the next step determines the calculation of the contraction cost of edges only.

### Contraction Cost

For just using edge collapses, unconnected regions are not taken into account. The notation $(v_i, v_j) \rightarrow \bar{v}$ is used for edge contraction. The vertices $v_i$ and $v_j$ are a valid contraction pair if they are connected by an edge. Unconnected vertices are not valid. The edge contraction quadric is calculated by Equation 3.17.

$$Q_{\bar{v}} = Q_i + Q_j = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ q_{41} & q_{42} & q_{43} & q_{44} \end{bmatrix} \tag{3.17}$$

$$\Delta(\bar{v}) = \bar{v}^T (Q_i + Q_j)\bar{v} \tag{3.18}$$

The edge cost is determined by Equation 3.18. The calculated costs for each edge is put in a sorted heap with least cost first. The next step involves the placement for the new vertex.

## 3.4.4 Vertex Placement

There are at least two different approaches for finding a new place for a contracted vertex. The most common techniques are the *optimal placement*, and the *subset placement* shown in Figure 3.14. The subset placement can be used if the optimal placement fails or as a standalone solution for a faster algorithm. The vertices are moved to a new position $\bar{v}$. All incident edges of $v_j$ are reconnected to the vertex $v_i$ and the vertex $v_j$ will be deleted. By knowing that the error is equal to Equation 3.16, finding $\bar{v}$ is an optimization problem by minimizing $\Delta(\bar{v})$. This linear problem is solved by finding the vector $\bar{v}$ which satisfies:

$$\min_{(\bar{v}} \ \bar{v}^T \, Q_{\bar{v})} \, \bar{v} : \frac{\partial \Delta}{\partial x} = \frac{\partial \Delta}{\partial y} = \frac{\partial \Delta}{\Sigma z} = 0 \tag{3.19}$$

The optimal location for the new vertex position $\bar{v}$ can be calculated as in Equation 3.20. The Figure 3.14 shows the principle of the underlying equation.

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \bar{v} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \Rrightarrow \bar{v} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.20}$$

But what happens if the quadric $Q_{\bar{v}}$ is not invertible? In other words, there may be infinitely many minimal points. In this case there is a fall-back strategy called *subset placement*. This approach can also be divided into two parts. The new vertex position

can either be one of the source vertices ($\bar{v} = v_i$) and ($\bar{v} = v_j$) or it can be placed right in between ($\bar{v} = \frac{v_i+v_j}{2}$) like shown in Figure 3.14.

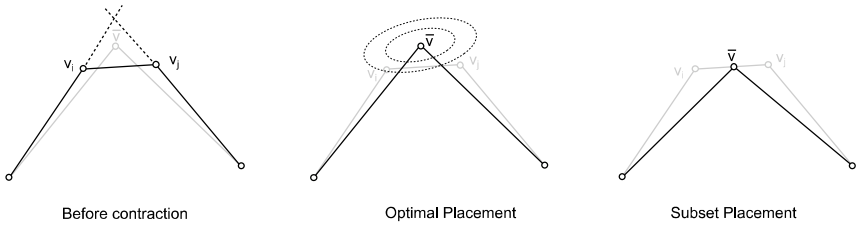


Figure 3.14: Optimal Vertex Placement

## 3.5 Representation Enhancements

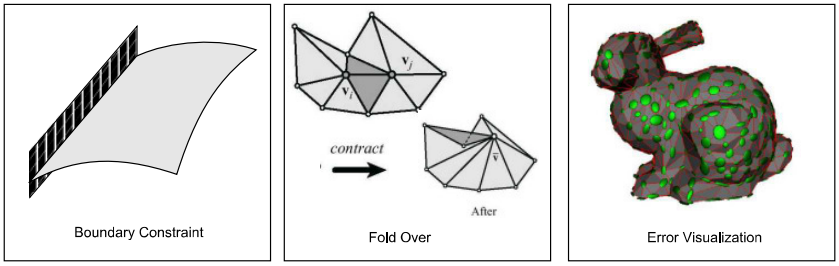


Figure 3.15: Algorithmic Enhancements

### 3.5.1 Boundary Constraints

Boundary constraints [15] are added for models with holes or models with boundaries such as terrain height fields. It is often necessary to preserve boundary curves and discrete color discontinuities while simplifying their shape. Each edge is labeled as either a normal edge or a discontinuity edge. For each face surrounding a particular discontinuity edge, a perpendicular plane, running through the edge (Figure 3.15), will be generated. These constrain planes are then converted into quadrics and weighted by a large penalty factor. It is also possible to prevent an edge from the contraction process by excluding the designated edges from the heap of contraction costs.

### 3.5.2 Triangle Fold-Over

Pair contraction does not necessarily preserve the orientation of the faces in the area of the contraction as seen in Figure 3.15. For instance, it is possible that contracting an edge and causes some neighboring faces to fold over on each other. Considering a

possible contraction, the normal of each neighboring face is compared before and after the contraction. If the normal flips, the designated contraction candidate can be either heavily penalized or disallowed and removed from the pipeline.

### 3.5.3 Error Visualization

The right image in Figure 3.15 show what quadrics are really doing. They characterize the error at a vertex. The level surfaces of these quadrics are almost always ellipsoids. In some cases, the ellipsoids are degenerate. The quadric matrix used for finding the optimal vertex positions (Equation 3.20) will be invertible as long as the level surfaces are non-degenerate ellipsoids. In this case, $\bar{v}$ will be at the center of the ellipsoid.

## 3.6 Summary

In summary, the algorithm from Michael Garland [15] offer an optimistic trade-off between quality and efficiency. Due to the many re-sorting steps after the edge contraction and updating the updating of the vertex costs, it is crucial to have an efficient data structure for edges, faces and vertices. The good estimation for contraction errors preserves higher-level surface characteristic, like the overall shape and tiny details.
There are a number of areas in which the current simplification methods can be improved. The current simplification method from Michael Garland assumes that the surface being simplified is rigid. There are many applications where surfaces are changing over time. For example, animation systems usually represent characters as a surface attached to artificial skeletons. As the skeletal joints bend, the surface is deformed. Current simplification methods must be extended to handle this more generalized class of models.

## Chapter 4

# Implementation

All implementation work has been done in Visual Studio 2010 using the C# programming language in its latest version 4.0. The advantage of a managed programming language focuses on a simplified code construction and code debugging. There is more space for optimizing and getting the algorithm work properly. Because C# is fully integrated with the .NET library, it provides access to a great repository of functionality, lots of predetermined classes and a good support. A disadvantage is less efficiency in execution performance, compared to C or C++. The gained productivity of coding allows further code optimization that virtually compensates the drawback concerning execution power.

The algorithms from this Thesis were implemented into the Aardvark rendering framework invented at the VRVis [13]. By using the rendering framework, there are the following advantages over using native OpenGL or DirectX:

- **Model Loading:** Aardvark provides the functionality of either implementing the loading of own file formats or loading indexed face sets from VRML files or STL files:
  - **STL (STandard Lithograhy)** [41] files are common for rapid prototyping. STL format describes only the surface geometry of a three-dimensional object without any representation of color, texture or other mesh attributes.
  - **VRML 2.0** (Virtual Reality Modeling Language) [44] files for extended mesh attributes. VRML is a text file format with vertices and indices forming polygons. The surface color, texture coordinates, textures, transparency and many other attributes can be loaded from a VRML file.
- **Math Helpers:** There are many math helpers, data structures and sorting algorithms available. Besides matrices, vectors and mesh connectivity for simple geometry, data structures for vertex attributes like normals, color and texture coordinates are available and ready-to-use.

- **PolyMesh Data Structure:** Aardvark contains a sophisticated API for polygonal mesh representation. Since this work focuses on triangle meshes only, any mesh loaded from file or created on-the-fly will be triangulated first.

## 4.1 User Interface

The user interface of the simplification application is shown in Figure 4.1 and is divided into three parts, which will be described in the next Sections:

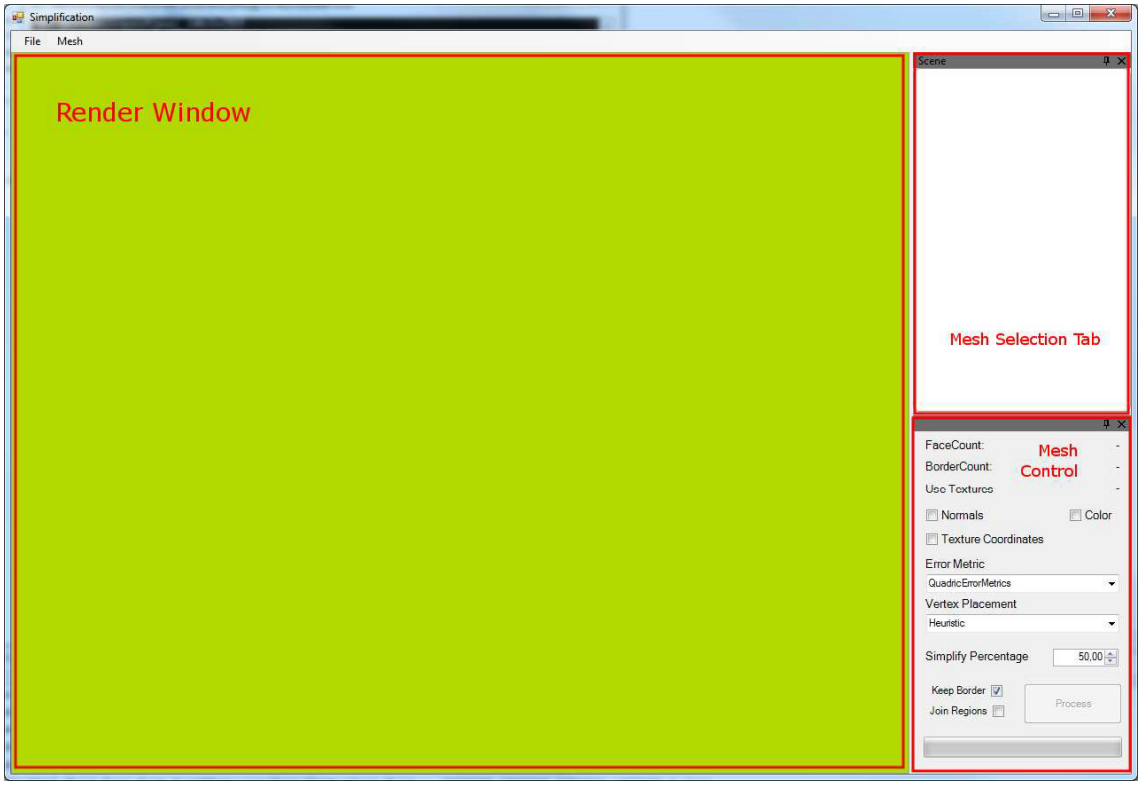


Figure 4.1: User Interface of Application

### 4.1.1 Render Window

The render window is the central element where the active models are currently displayed. The user interaction and navigation are similar to Autodesk Maya [29]. By holding the left mouse button (LMB), the object rotates, panning is possible by holding the middle mouse button (MMB) and moving the mouse. Moving the mouse wheel forward and backwards zooms in and out. Holding the right mouse button and moving the mouse back and forth has the same effect.

### 4.1.2 Mesh Selection Tab

After loading a mesh from the file system the model is added to the selection tab. Every simplified model is also added to this tab. Every model in the selection tab can be enabled or disabled for being rendered by checking the box in front of the name. A mesh which needs to be simplified is selected by highlighting its name. After that, additional information are shown in the control tab and the simplification process can be started.

### 4.1.3 Mesh Control Tab

The mesh control tab (see Figure 4.2 for a detailed close-up) offers a user interface to alter the options for mesh simplification. For any loaded model which is highlighted in the selection tab, the following details are shown:

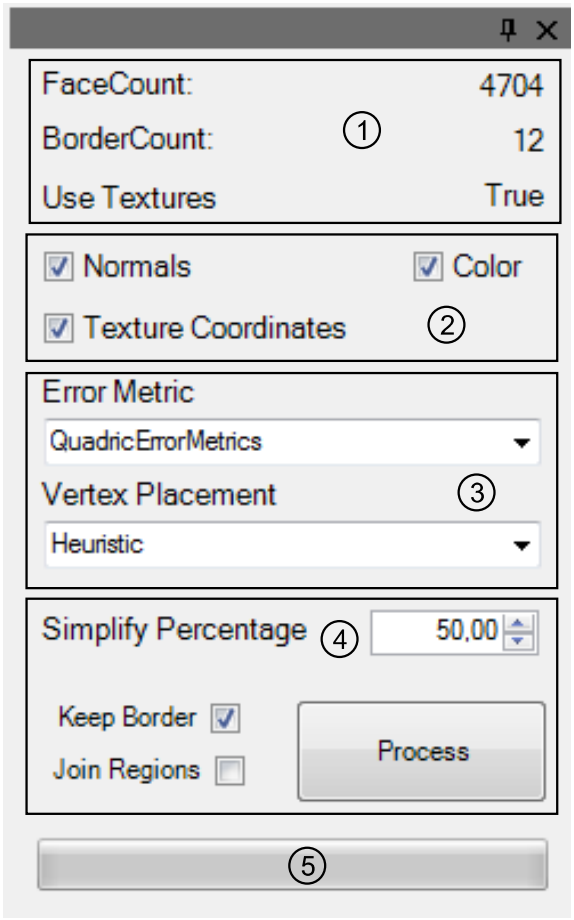


Figure 4.2: Mesh Control Close-up View

1. **Mesh Statistics**

   - **Face Count:** The face count of the model is an important indicator for its level of quality. The next Chapter 5 show the results of different levels of mesh simplification.
   - **Border Count:** Normally, closed models does not have borders. This information is useful for terrains or data types with holes. While the hole gets bigger by not preserving the border, the diameter of the hole is hardly modified otherwise.
   - **Use Textures:** If there are texture attributes specified, they can be taken into account while simplifying the model. The necessary option can be specified in the mesh attribute section below.

2. **Mesh Attributes**

   - **Normals:** Even if the normals are not included in the models, they can be generated in the framework and they are necessary for creating the triangle plane equation as described in Section 3.1.4. With the enhanced algorithm, introduced in Section 3.3.2, normals can be included in mesh simplification.
   - **Color:** This option is available only if either face colors (for flat shading [1]) or vertex colors (Gouraud shading or better [1]) are specified in the model. Otherwise the option is set to inactive.
   - **Texture Coordinates:** If textures and texture coordinates are assigned to the loaded model in the selection tab, this option is available for aligning the texture coordinates to the new vertex positions, too.

3. **Simplification Technique**

   - **Error Metric:** There are different error metrics selectable (see Figure 4.3) which differ in processing speed and output quality. A representation of the results is shown in Chapter 5.
   - **Vertex Placement:** If the selected error metric is equal to *Quadric Error Metrics*, the vertex placement can also be specified
   - **Simplify Percentage:** The face count of the mesh is going to be decreased by this percentage factor. The value can be selected between 1.0 % and 100.0% where no action is taken. There are several processing steps possible, where already simplified objects can be processed again.

4. **Mesh Topology**

- **Keep Border:** The borders of meshes with holes or open meshes are preserved if this option is selected.

5. **Progress Bar:** With the progress bar, the user can follow the progress of the mesh simplification. Every mesh simplifying iteration is tracked and the bar is updated accordingly.

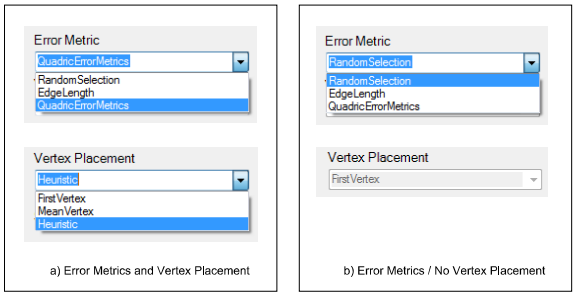


Figure 4.3: a) Quadric Error Metric - b) Random/Edge Length

## 4.2 Poly Mesh Data Structure

The polygonal mesh data structure is the key for the mesh simplification algorithm. It offers a rich API and an intelligent topological behavior to cycle efficient between faces, vertices and edges. Because this work only focuses on triangles only, this section do not contain further information on how faces of different types are handled. Therefore, the loaded mesh must be triangulated first:

```
PolyMesh triangleMesh = srcMesh.TriangulatedCopy().WithTopology();
```

After getting the triangle mesh from the source mesh, there is access to the properties of the poly mesh API as seen in Figure 4.4:

## 4.3 Create Quadrics

The quadric setup is an iterative process. The method

```
public PolyMesh ProcessMesh() {...}
```

in Listing C.1 shows the iterative process from the initial model to the simplified one. At first, the face quadric in Listing C.2 is created. At second, the quadrics for every vertex are going to be calculated (see Listing C.3). At third, every edge of the mesh is cycled. If the edge is a boundary and should be kept, it will be excluded from the simplification process. The simplification process ends if the target number of faces is reached. After that, the mesh topology is rebuilt and the new mesh will be returned.

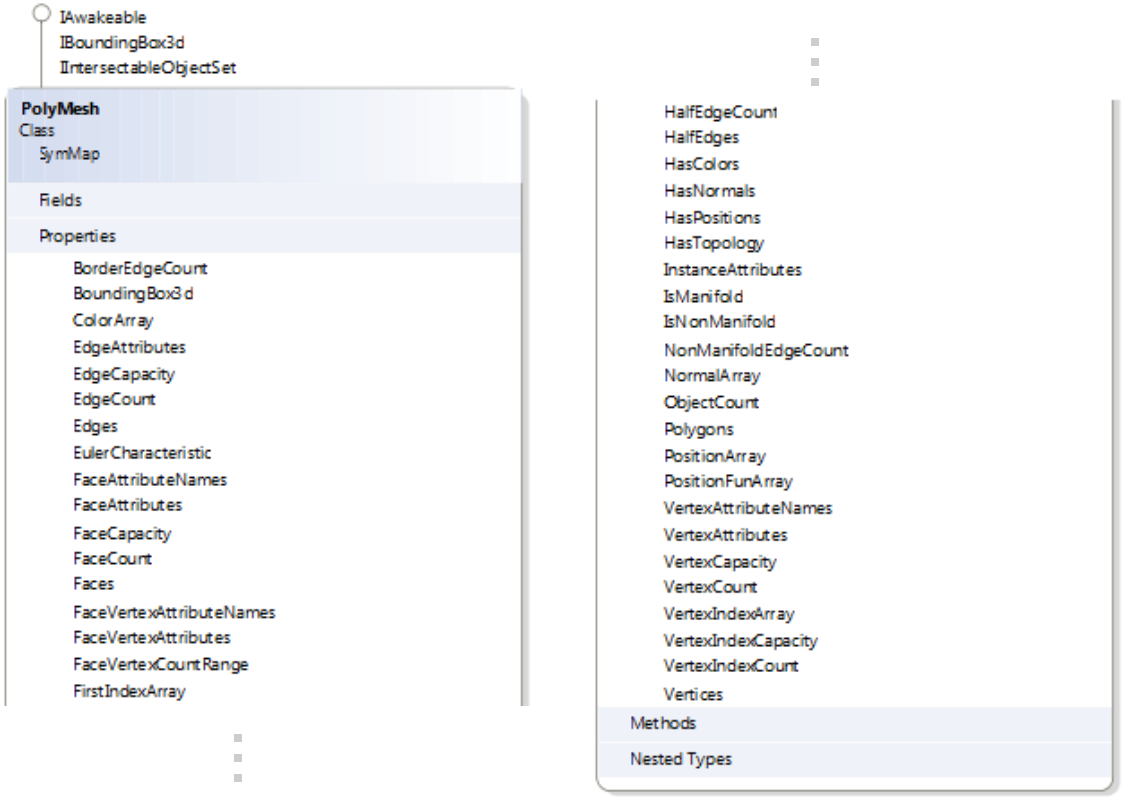


Figure 4.4: PolyMesh API

## 4.4 Summary

This chapter showed the user interface and the available settings for the mesh simplification process. Furthermore some source code show the creation of the face quadrics and vertex quadrics and the construction and sorting of the edge metrics as a basis for iterative edge contraction.

## Chapter 5

# Results

The following Sections presents the qualitative achievements and the performance results by simplifying the source models at different levels of detail. All computations and tests have been made using a computer system equipped with the following hardware components in addition with the listed software:

- **Hardware**
  - Core2Quad Q6600 2.4 GHz
  - 8GB RAM
  - nVidia GeForce 9800 GX2
  - HDD Samsung HD753LJ
- **Software**
  - MS Windows 7 64-bit
  - MS Visual Studio 2010
  - DirectX 10.0
  - Autodesk Maya 2011
  - Deep Exploration™5.5 Trial

The computations of the metrics and the overall simplification procedures are processed iteratively without parallel optimizations in the Aardvark framework. The resulting approximated models were rendered in Maya without further optimizations of the geometry or normals. The overlays in the Figures 5.6, 5.7 and 5.8 were created using Deep Exploration [18].

## 5.1 Source Data

Two different source objects were taken for the survey on mesh simplification. The first is a feline shown in Figure 5.1 which is a closed, static triangulated mesh with a total of 99.732 faces without borders. The characteristics of the mesh are some quite planar parts on the back or the wings of the model as well as some sharp boundaries around the wings or at the tail. The welsh dragon in Figure 5.2 in contrast consists mainly on fine granulated detailed sections. The triangulated model is highly detailed with a total of 2.2 million faces.

Figure 5.1: Feline with 99.732 Faces

The following Section 5.2 shows some simplified approximations of the original models. Depending on the selected contraction metrics from Section 3.4 the simplification process is either preserving or destructing the quality of the mesh.

## 5.2 Quality Comparison

The following Sections show some simplified qualitative results using either *Random Edge Selection* 5.2.1, *Edge Length Metrics* 5.2.2 or the main technique of this work, *Quadric Error Metrics* in Section 5.2.3.

Figure 5.2: Welsh Dragon with 2.210.673 Faces

### 5.2.1 Random Edge Selection

The random iteration process is quite fast but very destructive for the overall shape. Figure 5.3 shows the approximation of the source model in Figure 5.1 with 25 percent of its original face count. Especially the fine details at the head and the wings are gone. The whole model has irregularities and looks jagged. Because of the random approach, the local topology is not taken into account. This results in increasing non-manifoldness and thereby to more defective meshes than using quadric error metrics.

### 5.2.2 Edge Length Metrics

In contrast to the random selection, the present technique (refer to Section 3.4.2 for details) offers a simple quality metric by taking the edge, its length, and the angle between the normals of the adjacent faces into account. The result in Figure 5.4 offers a face count reduction to 25% of its original size. The quality is much better than with random edge contraction. The resulting mesh looks smoother and less jagged. But there are some irregularities (noticeable by black spots) at the neck and the lower extremities. The loss of detail can be seen best at the wings.

Figure 5.3: Feline 25% using Random Edge Selection

### 5.2.3 Quadric Error Metrics

The main topic of this work deals with the mesh simplification using *quadric error metrics*. Therefore, more results are shown and will be compared to the original mesh. Image 5.5 shows a simplification of 25% of the original size. Compare the results to the original in Figure 5.1. The simplified version of the model is hardly distinguishable from the original mesh. The random approach and the approach based on edge length show the huge output differences. Due to the fact that the local topology is taken into account, the shape is better preserved and no irregularities can be determined.

### QEM Levels vs. Original Mesh

This Section shows the levels of simplification (25%, 5% and 1% of the original face count) as overlays compared to the original mesh using the feline as an example. Figure 5.6 shows the reduced mesh in the left image. The right image shows the original mesh in gray color. The red patches in the right image show, where the edge contractions has been made. The reduced spots are almost evenly distributed. Mainly overhead details has been removed and tiny details has been kept very well (e.g. on the shoulder, ears, eye).

Figure 5.4: Feline 25% using Edge Length Metric

Figure 5.7 shows a more advanced level of simplification at only 5 percent of the original face count. The patches in the right image are getting bigger and some fine-grained parts appear less detailed. The overall design is kept very well with fewer faces compared to the 25% approach with random selection or even edge length metric.

The most advanced simplification can be seen in Figure 5.8. Whereas the overall shape is relatively good preserved, some details are already missing. The tail is quite jagged, the details of the eye, the ear and the pads are more squared than smooth.

The next Section 5.3 shows an overview of the runtime of the different quality levels and with different edge contraction techniques.

## 5.3 Performance Comparison

The last Section gave an overview of the resulting quality of the different edge contraction metrics. The disadvantage of better quality is a longer processing time. Figure 5.9 shows an overview of the time the feline model is going to be simplified with the quadric error metrics. Each simplification value starts from the original mesh and outputs a model with

Figure 5.5: Feline 25% using Quadric Error Metrics

the stated face count.
Table 5.10 shows the processing time of the welsh dragon, a much bigger model, to the same percentage of the original model as in the Figure before.
The Diagram 5.11 shows a runtime comparison from the simplification procedure of the feline and the welsh dragon in the foregoing tables. The y-Axis is logarithmically scaled to have a good overview of the elapsed time.
There are five different edge metrics in total (quadric error metrics uses different vertex placement options) for the overview of the average runtime shown in Figure 5.12 with the feline as source model. The determined testing times were averaged from five different measurements of each edge metric technique.

## 5.4 Summary

In this chapter the results of two different input meshes using different simplification settings were shown. Different levels of quality were reached and has been compared in contrast to the necessary processing time to get coarser detailed models.

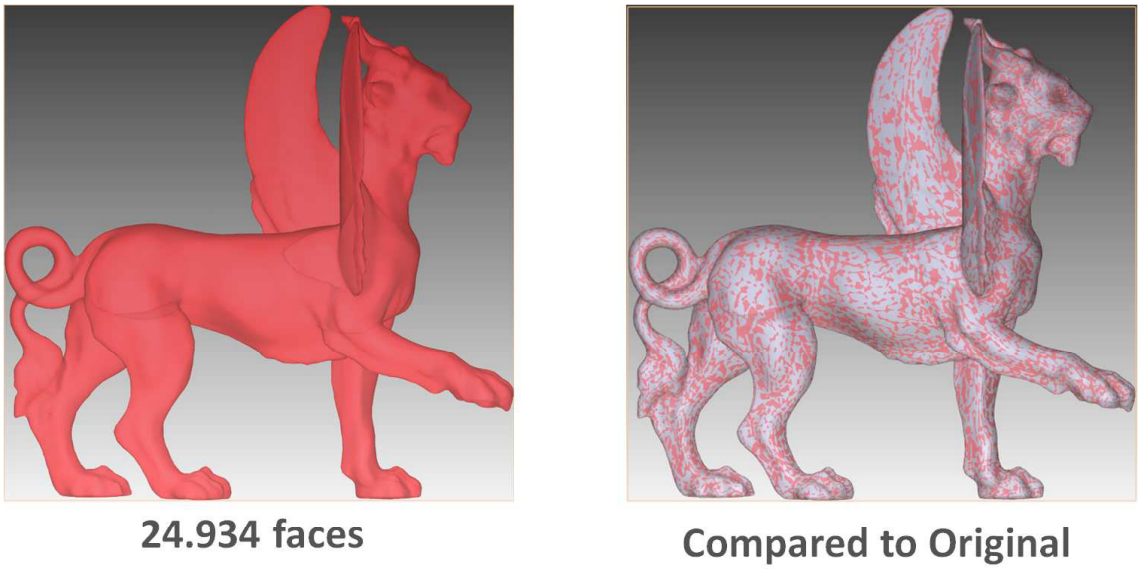


Figure 5.6: Simplified to 25% - Original Size

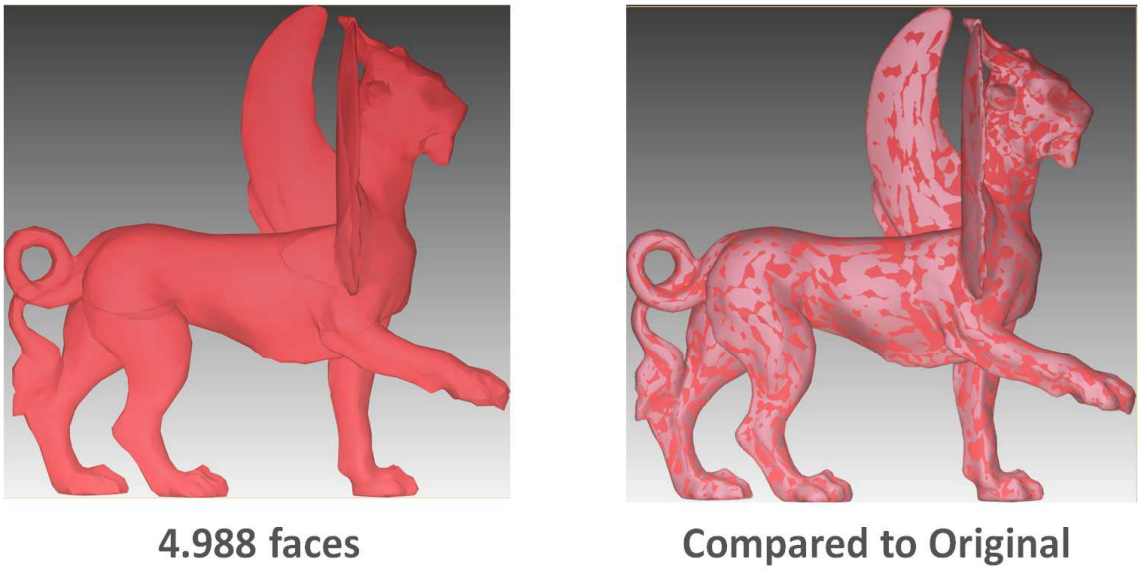


Figure 5.7: Simplified to 5% - Original Size

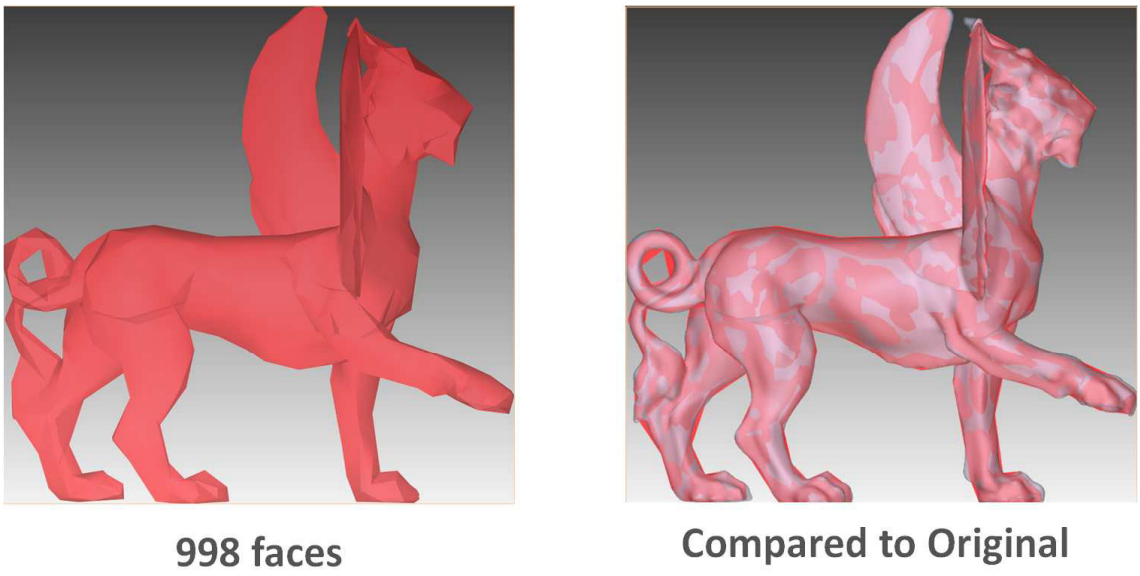


Figure 5.8: Simplified to 1% - Original Size

| % of original | Resulting Faces | Time in s |
|---|---|---|
| 75 | 74.800 | 4,55 |
| 50 | 49.868 | 9,26 |
| 25 | 24.934 | 12,34 |
| 10 | 9.974 | 14,33 |
| 5 | 4.988 | 15,05 |
| 2 | 1.996 | 15,44 |
| 1 | 998 | 15,63 |

Figure 5.9: Feline - QEM Simplification

| % of original | Resulting Faces | Time in s |
|---|---|---|
| 75 | 1.658.006 | 82,00 |
| 50 | 1.105.338 | 166,26 |
| 25 | 552.670 | 231,40 |
| 10 | 221.080 | 268,35 |
| 5 | 110.540 | 281,29 |
| 2 | 44.214 | 282,26 |
| 1 | 22.108 | 285,68 |

Figure 5.10: Welsh Dragon - QEM Simplification

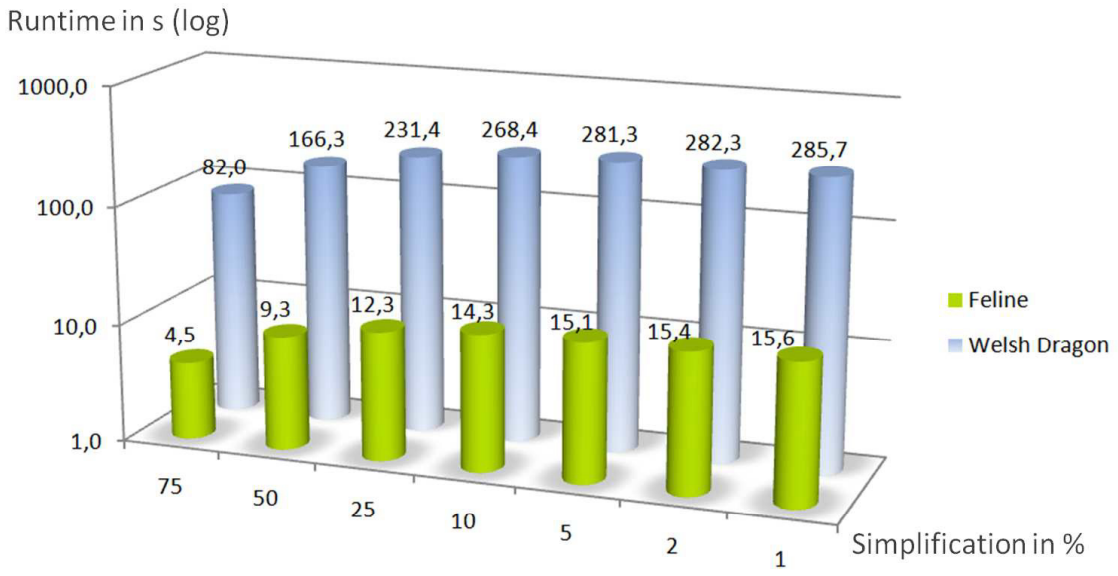


Figure 5.11: Feline & Welsh Dragon Simplification Graph

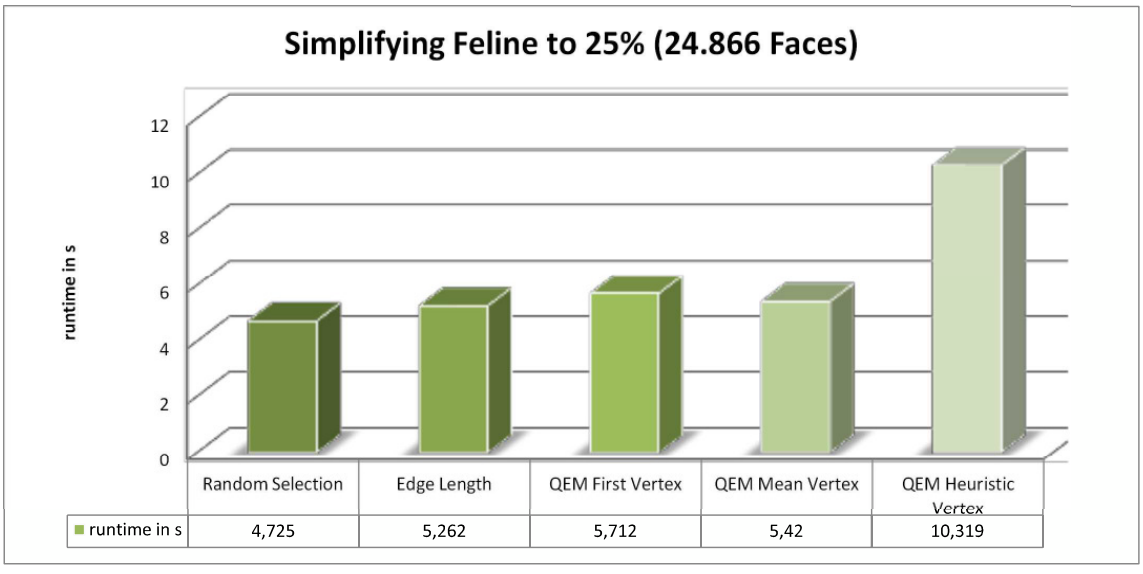


Figure 5.12: Feline Simplification Comparison Graph

Chapter 6

# Conclusion

## 6.1 Summary

This thesis gave an overview of the importance and reasons for mesh simplification. Chapter 1 introduced the reader to the major questions why simplified meshes are useful in computer graphics, which kind of models are taken, and which qualitative information should be kept and which information overhead can be removed. There are many techniques, concerning simplification, available. Chapter 2 showed the state of the art in simplifying polygonal or non-polygonal graphics representations. Chapter 3 started with a geometric primer of vertices, edges and triangles and their interconnectivity in a mesh. The underlying algorithm of this work, *Surface simplification using quadric error metrics* [15], was presented next. Three different edge contraction metrics were presented in total. The first algorithm selects edges at random. The second one selects the edge being contracted by weighting its length. The underlying algorithm of the third available uses the surrounding faces and vertices to safely contract edges by its surrounding surfaces. The results in case of quality and performance of the different simplification techniques has been shown in Chapter 5. The simplification using quadric error metrics produced a good shape preservation at reasonable processing performance and is preferable among the other available techniques.

## 6.2 Future Work

At the moment, the implemented algorithm simplifies geometry only. For future projects, face attributes or vertex attributes can be taken into account. Depending on the mesh data structure, per face attributes can be normals or colors. Vertex attributes can also be colors, normals or texture coordinates. If the vertex moves to a new position and the texture coordinate stays the same, the overlying texture will be stretched and unevenly distorted. Using parallel processing can increase the process time of mesh simplification. The mesh can be divided in several parts to calculate the face quadrics or the vertex quadrics in concurrent threads because of there independences of each other.

## Appendix A

# David Scan Statistics

| | |
|---|---|
| 480 individually aimed scans | For each scan, the gantry was moved to a new position. In its maximum configuration, our gantry weighs 1800 pounds. It can be rolled, with difficulty, by two strong people. |
| 2 billion polygons | Each part of the statue was covered by at least two scans. Difficult sections were covered 5 or more times, for example when trying to reach the bottoms of cracks. |
| 7,000 color images | Each image is 1520 x 1144 pixels. For each position, one image was taken with our calibrated spotlight and one without. Subtracting the two eliminates ambient lighting. |
| 32 gigabytes | Data was stored on hard drives. Each morning, the night's scans were copied to a second hard drive, which was walked back to our laboratory and backed up on tape. |
| 1,080 man-hours of scanning | This does not include 1,000 hours of editing, aligning, and combining the scans, and 1,500 or so hours pumping up our software to handle the increasingly large datasets. |
| 22 people | 2 professors, 1 research associate, 1 technical staff, 6 PhD students, and 12 undergraduates. Of these 22, 15 were full-time and 7 were part-time. |

Table A.1: Statistics for the Scan of David. Table taken from the Digital Michelangelo Project [26]

## Appendix B

# Indexed Face Set from VRML97

**Specification 1** VRML97. Code taken from [44].

```
Coordinate {
   exposedField MFVec3f point  []   # (-∞,∞)
}

IndexedFaceSet {
  eventIn       MFInt32 set_colorIndex
  eventIn       MFInt32 set_coordIndex
  eventIn       MFInt32 set_normalIndex
  eventIn       MFInt32 set_texCoordIndex
  exposedField  SFNode  color             NULL
  exposedField  SFNode  coord             NULL
  exposedField  SFNode  normal            NULL
  exposedField  SFNode  texCoord          NULL
  field         SFBool  ccw               TRUE
  field         MFInt32 colorIndex        []        # [-1,∞)
  field         SFBool  colorPerVertex    TRUE
  field         SFBool  convex            TRUE
  field         MFInt32 coordIndex        []        # [-1,∞)
  field         SFFloat creaseAngle       0         # [0,∞)
  field         MFInt32 normalIndex       []        # [-1,∞)
  field         SFBool  normalPerVertex   TRUE
  field         SFBool  solid             TRUE
  field         MFInt32 texCoordIndex     []        # [-1,∞)
}
```

## Appendix C

# Source code

### C.1 Process Mesh

```
public PolyMesh ProcessMesh()
{
  while (!m_settings.ReachedTargetFaceCount)
  {
    CreateFaceQuadrics();
    CreateVertexQuadrics();
    CreateEdgeMetrics();

    SelectValidEdges();
    ////////////////////////// END 1st iteration

    RebuildMesh();
    InitMembers(m_inputMesh, m_settings);
  }

  m_settings.ReachedTargetFaceCount = false;

  return m_inputMesh;
}
```

### C.2 Create Face Quadrics

```
void CreateFaceQuadrics()
{
  var fia = m_inputMesh.FirstIndexArray;
  var via = m_inputMesh.VertexIndexArray;

  for (int fi = 0; fi < m_inputMesh.FaceCount; fi++)
  {
    var fvi = fia[fi];  // 0
    var fve = fia[fi + 1]; // 0 + offset == 3
```

```
    // Three Vertices per Face
    var offset = fve - fvi;

    if (offset != 3)
    {
      throw new IndexOutOfRangeException("Triangle Offset != 3");
    }

    var poly = m_inputMesh.GetPolygon(fi);
    var plane = poly.ToPlane3d();

    m_faceQuadrics[fi].CreateQuadric(plane);
  }
}
```

## C.3 Create Vertex Quadrics

```
void CreateVertexQuadrics()
{
  var vc = m_inputMesh.VertexCount;

  foreach (var v in m_inputMesh.Vertices)
  {
    foreach (var f in v.Faces)
    {
        m_vertexQuadrics[v.Index] += m_faceQuadrics[f.Ref.Index];
    }
  }
}
```

## C.4 Create Edge Metric

[label=lst:edgeMetric]

```
void CreateEdgeMetrics()
{
  var pts = m_inputMesh.PositionArray;
  var ec = m_inputMesh.EdgeCount;

  foreach (var e in m_inputMesh.Edges)
  {
    var ei = e.Index;
    var v0 = e.FromVertex;
    var v1 = e.ToVertex; ;
```

```
    var edgeIsBorder = e.IsBorder || e.OppositeIsBorder;

    if (edgeIsBorder && m_settings.KeepBorder)
    {
        LockVertices(new[] { v0, v1 });
    }
    else
    {
        m_edgeMetrics[ei].Create(m_vertexQuadrics[v0.Index],
                                 m_vertexQuadrics[v1.Index],
                                 v0, v1, m_settings.VertexPlacement);

        m_edgeCost[ei] = m_edgeMetrics[ei].Cost;
    }
  }

  m_costPermutation = m_edgeCost.CreatePermutationQuickSortAscending();
}
```

# List of Figures

# List of Tables

# Bibliography


[1] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008. 23, 24, 26, 64

[2] Bruce G. Baumgart. Winged edge polyhedron representation. Technical report, Stanford University, Stanford, CA, USA, 1972. 32

[3] Elizabeth C. Beckmann. CT scanning the early days. *The British journal of radiology*, 79(937):5–8, January 2006. 22

[4] Jon Louis Bentley. K-d trees for semidynamic point sets. In *Proceedings of the sixth annual symposium on Computational geometry*, SCG '90, pages 187–197, New York, NY, USA, 1990. ACM. 30

[5] H. L. Bodlaender and T. Wolle. A note on the complexity of network reliability problems. Technical Report UU-CS-2004-001, Institute of Information and Sciences, Utrecht, The Netherlands, 2004. 37

[6] Fernando Cacciola. Cgal.org manual, chapter 38. http://www.cgal.org/Manual/3.3/doc_html/cgal_manual/Surface_mesh_simplification/Chapter_main.html, 2007 (accessed March 8th, 2011). 11, 82

[7] Bernard Chazelle and David P. Dobkin. Optimal convex decompositions. In *Computational Geometry*, pages 63–133, 1985. 42

[8] Baoquan Chen and Minh Xuan Nguyen. Pop: A hybrid point and polygon rendering system for large data. *Visualization Conference, IEEE*, 0:null, 2001. 26

[9] Jonathan D. Cohen, Daniel G. Aliaga, and Weiqiand Zhang. Hybrid simplification: Combining multi-resolution polygon and point rendering. In *IEEE Visualization*, 2001. 27

[10] Leila De Floriani, Leif Kobbelt, and Enrico Puppo. *A survey on data structures for level-of-detail models*, pages 49–74. Series in Mathematics and Visualization. Springer Verlag, Berlin, 2005. 23, 31

[11] Xavier Decoret, Fredo Durand, and Francois X. Sillion. Billboard clouds. In *Proceedings of the nineteenth annual symposium on Computational geometry*, SCG '03, pages 376–376, New York, NY, USA, 2003. ACM. 24, 82

[12] Fletcher Dunn and Ian Parberry. *3D Math Primer for Graphics and Game*. Wordware Publishing Inc., Plano, Texas, US, 2002. 24

[13] Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH. Vrvis. http://www.vrvis.at, 2010 (accessed September 29th, 2010). 32, 61

[14] Michael Garland. *Quadric-based polygonal surface simplification*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1999. Chair-Heckbert, Paul. 51, 52

[15] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 18, 30, 36, 52, 57, 59, 60, 76

[16] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 263–269, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press. 53

[17] Markus H. Gross, Oliver G. Staadt, and Roger Gatti. Efficient triangular surface approximation using wavelets and quadtree data structures. *IEEE Transaction on Visualization and Computer Graphics*, 2(2):130–143, 1996. 19

[18] Right Hemisphere. Deep exploration cad edition. http://www.righthemisphere.com/products/dexp/, 2011 (accessed April 13th, 2011). 67

[19] Hugues Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22(1):27–36, 1998. 27, 33

[20] Hugues Hoppe, Tony Derose, Tom Duchamp, John Mcdonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992. 38

[21] Christoph Hormann. Isosurface object speedup patch. http://www.imagico.de/fast_iso/patch.html, 2010 (accessed December 6th, 2010). 24, 82

[22] Jerry O. Talton III. A short survey of mesh simplification algorithms, 2004. 22

[23] Bin Shyan Jong, Juin-Ling Tseng, and Wen Hao Yang. An efficient and low-error mesh simplification method based on torsion detection. *The Visual Computer*, 22(1):56–67, 2006. 22

[24] Lutz Kettner. Halfedge data structures. In CGAL Editorial Board, editor, *CGAL-3.2 User and Reference Manual*. CGAL, 2006. 32

[25] David Körner. Computergraphische simulation dynamischer bruchbildung in starr-plastischen gemengen. Diplomarbeit, Technische Universität Dresden, 2008. 38, 82

[26] Stanford University Computer Graphics Laboratory. Stanford digital michelangelo project. http://graphics.stanford.edu/projects/mich/, 2010 (accessed February 10th, 2010). 14, 77

[27] David Luebke. A survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, 21:24–35, 1997. 13, 39

[28] Christian Luksch. Implementation of an improved billboard cloud algorithm. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, July 2009. 24

[29] Autodesk Maya. Autodesk, inc. http://usa.autodesk.com/maya/, 2011 (accessed April 6th, 2011). 62

[30] Stan Melax. A simple, fast and effective polygon reduction algorithm. *Game Developer Magazine*, pages 44–49, 2008. 55

[31] Niloy J. Mitra, Natasha Gelfand, Helmut Pottmann, and Leonidas Guibas. Registration of point cloud data from a geometric optimization perspective. In *Symposium on Geometry Processing*, pages 23–31, 2004. 19

[32] Carsten Moenning and Neil A. Dodg. Intrinsic point cloud simplification. In *Proceedings of the 14th GraphiCon*, 2004. 22

[33] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 163–170, Washington, DC, USA, 2002. IEEE Computer Society. 20

[34] Roland L. Plaster. The imaging & geospatial information society. http://www.asprs.org/publications/pers/2001journal/november/highlight2.html, 2001 (accessed April 3rd, 2011). 19, 82

[35] Fabio Remondino. From point cloud to surface: The modeling and visualization problem. In *Proceedings of the International Workshop on Visualization and Animation of Reality-based 3D Models*, volume XXXIV-5/W10, 2003. 19

[36] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximation for rendering complex scenes. In *Geometric Modeling in Computer Graphics*, pages 455–465. Springer Verlag, 1993. 18

[37] Philip J. Schneider and David Eberly. *Geometric Tools for Computer Graphics*. Elsevier Science Inc., New York, NY, USA, 2002. 26, 49, 50

[38] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, New York, NY, USA, 1992. ACM. 18

[39] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, New York, NY, USA, 1992. ACM. 22, 23, 35

[40] E. J. Stollnitz, A. D. DeRose, and D. H. Salesin. Wavelets for computer graphics: a primer.1. *IEEE Computer Graphics and Applications*, 15(3):76–84, May 1995. 39, 82

[41] 3D Systems. Stereo lithography. http://www.3dsystems.com, 2011 (accessed April 6th, 2011). 61

[42] Robert F. Tobler and Stefan Maierhofer. A mesh data structure for rendering and subdivision. In *Proceedings of WSCG (International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision)*, pages 157–162, 2006. 29, 32, 33

[43] Greg Turk. Re-tiling polygonal surfaces. In *Computer Graphics*, pages 55–64, 1992. 19, 38

[44] web3D Consortium. Vrml97 functional specification. http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/, 2011 (accessed February 14th, 2011). 29, 61, 78