A COMPARATIVE STUDY ON POLYGONAL MESH SIMPLIFICATION ALGORITHMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MURAT YİRCİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2008

Approval of the thesis:

POLYGONAL MESH SIMPLIFICATION AND REGULAR REMESHING

submitted by **MURAT YİRCİ** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İsmet Erkmen _____
Head of Department, **Electrical and Electronics Engineering**

Assist. Prof. Dr. İlkay Ulusoy _____
Supervisor, **Electrical and Electronics Engineering Dept., METU**

Examining Committee Members

Prof. Dr. Yasemin Yardımcı Çetin _____
Informatics Institute, METU

Assist. Prof. Dr. İlkay Ulusoy _____
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Veysi İşler _____
Computer Engineering Dept., METU

Assist. Prof. Dr. Çağatay Candan _____
Electrical and Electronics Engineering Dept., METU

Assist. Prof. Dr. Cüneyt F. Bazlamaçcı _____
Electrical and Electronics Engineering Dept., METU

Date: 01.09.2008

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Murat Yirci

Signature :

ABSTRACT

A COMPARATIVE STUDY ON POLYGONAL MESH SIMPLIFICATION ALGORITHMS

Yirci, Murat

M.S., Department of Electrical and Electronics Engineering

Supervisor: Assist. Prof. Dr. İlkay Ulusoy

September 2008, 115 pages

Polygonal meshes are a common way of representing 3D surface models in many different areas of computer graphics and geometry processing. However, these models are becoming more and more complex which increases the cost of processing these models. In order to reduce this cost, mesh simplification algorithms are developed. Another important property of a polygonal mesh model is that whether it is regular or not. Regular meshes have many advantages over the irregular ones in terms of memory requirements, efficient processing, rendering etc. In this thesis work, both mesh simplification and regular remeshing algorithms are studied. Moreover, some of the popular mesh libraries are compared with respect to their approaches and performance to the mesh simplification. In addition, mesh models with disk topology are remeshed and converted to regular ones.

Keywords: Polygonal Mesh Simplification, Regular Remeshing, 3D Models, Mesh Libraries

ÖZ

POLİGONAL SADELEŞTİRME ALGORİTMALARININ KARŞILAŞTIRILMASI

Yirci, Murat
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Yrd. Doç. Dr. İlkay Ulusoy

Eylül 2008, 115 sayfa

Poligonal yüzey modelleme üç boyutlu nesnelerin modellenmesinde sıkça kullanılan bir yöntemdir. Poligonal yüzey modelleri bilgisayar grafiği, bilgisayarla görme, tıbbi görüntüleme vb. alanlarda yaygın olarak kullanılmaktadır. Ancak, bu modellerin karmaşıklık düzeylerinin gün geçtikçe artması bu modellerin işlenmesinde zorluklar çıkarabilmektedir. Poligonal sadeleştirme algoritmaları kullanılarak bu modellerin karmaşıklık düzeyleri azaltılabilir. Poligonal yüzey modellerin bir diğer önemli özelliği de modelin yapısının düzgün olup olmadığıdır. Düzgün modellerin düzgün olmayan modellere göre hafızada daha az yer kaplama, daha kolay ve etkili işleme gibi birçok avantajı vardır. Bu tez çalışmasında poligonal yüzey modellerin sadeleştirilmesinde ve yeniden düzgün olarak modellenmesinde kullanılan algoritmalar incelenmiştir. Yapılan incelemelere ek olarak poligonal modellerin işlenmesinde yaygın olarak kullanılan kütüphaneler poligonal sadeleştirme açısından karşılaştırılmıştır. Son olarak da dairesel topolojiye sahip olan poligonal yüzey modellerin yeniden modellenmesi uygulama olarak geliştirilmiştir.

Anahtar Sözcükler: Poligonal Yüzey Sadeleştirme, Yeniden Düzgün Modelleme, 3B Modeller, Poligonal Yüzey Modelleme Kütüphaneleri

To My Parents

ACKNOWLEDGMENTS

I would like to thank Assist. Prof. Dr. İlkay Ulusoy for her valuable supervision, and patience. Her support on this thesis work increased my motivation to the top level and her guidance encouraged me to complete this thesis.

I wish to thank my mother and father separately for their invaluable, affectionate support during my whole life. In addition, I would also like to thank to my sisters, and all my friends for giving me encouragement and patience during this thesis.

I want to thank my colleagues and managers at ASELSAN for their continuous support on this research.

Finally, I have very special thanks for my dear friend Çiğdem Alkan, for her precious support and patience during this thesis.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Polygonal meshes are a common way of representing 3D surface models in many different areas of computer graphics and geometry processing [3]. Due to their popularity, polygonal meshes are widely supported by commercial graphics hardware and software. Polygonal models are generated in a number of different ways in different application domains. For instance, in computer vision laser range scanners are used to capture the geometry of the real world objects [68], whereas in scientific visualization iso-surfaces are extracted from volume data [32]. Moreover, in remote sensing height maps are constructed using the satellite photographs and in computer graphics and CAD systems modeling tools are utilized. Unfortunately, surface meshes obtained by using these methods rarely well suit the needs. Mostly, they are too complex or the quality is poor in some sense which require post-processing. Polygonal mesh simplification and remeshing algorithms are among the most important mesh processing algorithms.

With the evolution of modern technology devices, more and more complex surface mesh models have been generated. As the complexity of these models increases, the visual approximation to the real world objects gets better but there is a trade-off between the cost of processing these models and better visual approximation. In fact, for real-time interactive applications simpler models are acceptable most of the time without violating the expectations. As a result, complex models are needed to be reduced where the mesh simplification algorithms come in. Correspondingly, mesh simplification can be defined as the process of reducing the number of faces and vertices of a given input mesh while maintaining a faithful approximation to the original mesh.

Regarding the importance of mesh simplification algorithms they have been studied a lot for years and since then a number of survey papers were published on the subject [3, 6, 31, 37, 38, 39, 40]. Based on these researches, mesh simplification algorithms can be classified into two main groups according to their approaches to the subject. Iterative algorithms [41, 42, 43, 44, 45, 48] can be placed into the first category in which a mesh element (vertex, face or edge) is removed once at a time. On the other hand, the second category of algorithms can be named as single pass algorithms which process the whole input mesh in a single step [50, 51].

There are two key issues for iterative mesh simplification algorithms. The first one is the applied simplification operator which determines how to remove a mesh element and the second is the error calculation. Vertex removal scheme is first proposed by Schroeder at el. [41], but later, edge collapse and its variations become dominant operators such that today nearly all simplification algorithms use them. The second issue is the error calculation method that is used to determine which mesh element to remove at each step. There are a number of error calculation methods some of which only consider local changes such as distance to average plane [41], maximum supporting plane distance [45], and quadric error metrics [44], whereas Hausdorf distance [48] and simplification envelops [49] are considering the global changes. Generally, global error measurements give better results but when quality and timing requirements are considered, quadric error metric is much more promising than the others.

Single pass algorithms process the input mesh as a whole and generate the simplified mesh without an iterative process. Vertex clustering algorithms [50, 51] are the most famous single pass algorithms, where the bounding box of the models are divided into small regions and all the vertices in a region are clustered to a new position depending on some error calculation.

Since it is a very active research area, lots of mesh simplification algorithms are proposed some of which are easy to implement whereas some may not be as easy as the others. They are so commonly used algorithms such that some of them are embedded into the well known publicly available open source mesh libraries [24, 25, 26]. Thus, developers working on different research areas can utilize these algorithms rather than implementing their own, which may be a very difficult task for a non-specialist. However, there is not any work done on comparison of mesh simplification algorithms that are implemented in these libraries. In this thesis, our aim is to study and compare the common mesh libraries in terms of their approach and performance to the mesh simplification problem. By this way, we try to fulfill this absence in the literature and supply a guidance for developers who are seeking for publicly available mesh libraries in order to implement polygonal mesh simplification.

We used the metro tool [47] for measuring the geometric distance (both maximum and average) between the simplified and original models. There are also other tools [57, 58, 59] that can be used for the same job but metro is the first and widely accepted one among them. Two models with different properties are selected for comparison and all algorithms are executed with nine different reduction rates.

Besides the mesh simplification, remeshing of surface mesh models is also another key component for geometry processing and computer graphics. Remeshing can simply be defined as improving the quality of a mesh subject to some criteria. Different applications, of course, have different requirements. For example, numerical simulation applications such as finite element methods require well-shaped (triangles with aspect ratio close to one) and regular triangles. In fact, regular meshes show uniform structures which are preferable by many applications. Moreover, regular meshes have many advantages over the irregular ones. They can be stored in two dimensional arrays which simplifies the efficient processing of corresponding models. On the other hand, irregular meshes need

some other expensive complex data structures. In addition, it has been shown that regular meshes are more suitable for rendering [61, 62, 63].

Before the work of Gu et al. [61], automatic regular remeshing of arbitrary triangular meshes were not possible. He parameterized the input meshes on to a completely regular structure called geometry images and then reconstruct the original mesh by taking regular sampling points on the parameterization domain. In order to perform the parameterization step, the input meshes are cut and opened if their topology are not equivalent to disks. However, in our implementation, since we aimed the regular remeshing of human face models which are topologically equivalent to disks, the first part of the Gu's algorithm is omitted in our application.

In this thesis work, mesh simplification and regular remeshing algorithms are studied. In Chapter 1, an introduction to the subject of mesh simplification and regular remeshing is presented. Then, in Chapter 2, some general properties of polygonal meshes are described prior to further analysis of mesh simplification and regular remeshing. In Chapter 3, analysis and categorization of mesh simplification algorithms are performed. Moreover, mesh simplification algorithms that are embedded in mesh libraries are discussed in Chapter 4 which is followed by the comparison of these algorithms in Chapter 5. In Chapter 6, our solution to regular remeshing is explained and finally in Chapter 7, conclusions and future works are considered.

CHAPTER 2

POLYGONAL MESHES

2.1. Definitions

Before going into the details of mesh simplification and regular remeshing algorithms some of the general terms about the meshes should be defined. Also, it should be noted that in this thesis work only orientable, 2-manifold triangular meshes are considered.

2.1.1. Polygonal Meshes vs. Triangular Meshes

A polygonal mesh is a piecewise linear surface approximation of a real world object in computer graphics. It is composed of vertices, edges and faces. The faces of the polygonal meshes are convex polygons. Triangular meshes (Figure-1) are a subset of polygonal meshes where all of the faces are triangles. In this thesis work, only triangular meshes will be considered, since every polygon can be triangulated [1], which means every polygonal mesh can be converted to a triangular mesh.

More formally, Hoppe [2] defined a triangular mesh as follows: A triangular mesh M consists of a geometric and a topological (connectivity) component, which can be represented by a pair (K, P). K is a simplicial complex with a set of vertices $V = \{v_1, ..., v_m\}$ and composed of subsets of its vertices. These subsets are known as simplices and three types of them are more meaningful for triangle meshes. These are 0-simplices $\{i\} \in K$ (vertices of a mesh), 1-simplices $\{i, j\} \in K$ (edges of a mesh) and 2-simplices $\{i, j, k\} \in K$ (faces of a mesh). For a definition of simplicial complexes and k-simplices see Appendix-A. $P = \{p_1, ..., p_m\}$ is a set of 3D position vectors and defines the geometry or the shape of the triangular mesh.

The geometric embedding of a triangular mesh into R^3 is specified by associating a 3D position p_i to each vertex of the simplicial complex K such that $p_i := p(v_i) = (x(v_i) \quad y(v_i) \quad z(v_i)) \in R^3$ [5].

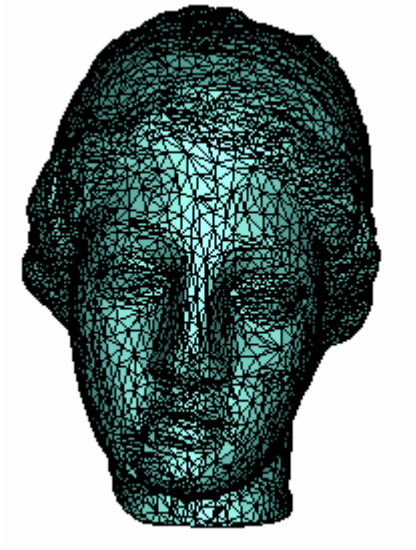

Figure 1. Egea: an example of a triangular mesh [33].

2.1.2. Manifold vs. Non-manifold Meshes

Manifold models are desirable for most of the mesh processing algorithms [6]. Many mesh simplification algorithms only accepts manifold input meshes. Manifold meshes are also required for finite element analysis and radiosity applications.

A surface is 2-manifold if each point of the surface is locally homeomorphic (topologically equivalent) to a disk or a half-disk(at the boundaries). For the case of a triangular mesh, the definition given above can be interpreted as follows: A triangular mesh is 2-manifold if it does not contain a non-manifold edge or a non-

manifold vertex nor self-intersecting. A non-manifold edge (Figure-2a) is incident to more than two faces and a non-manifold vertex (Figure-2b) is shared by a number of unconnected sets of triangles.

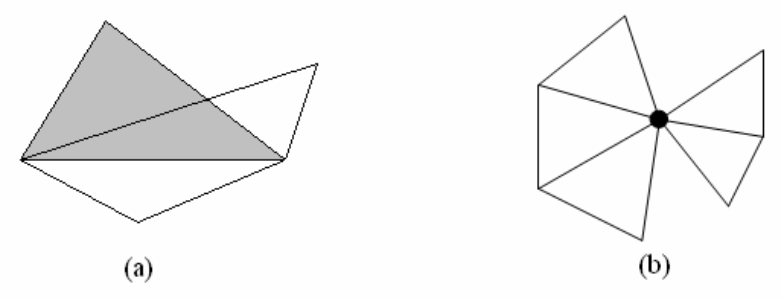

Figure 2. A non-manifold edge (a), and a non-manifold vertex (b)

2.1.3. Orientable vs. Non-orientable Meshes

A connected 2-manifold mesh is orientable if it is two-sided, which means that one can distinguish the inner and outer sides of the mesh. On the contrary, if we can not distinguish the inner and the outer sides of a mesh (one sided), then it is non-orientable. Spherical and cylindrical meshes are examples for orientable surfaces and möbius (Figure-3) and klein bottle are examples for non-orientable surfaces.

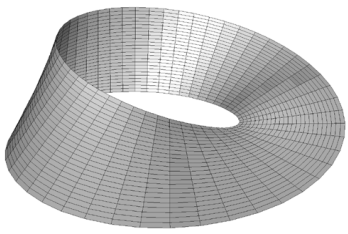

Figure 3. Möbius: a non-orientable mesh

2.1.4. Genus and Euler Formula

Genus is a topologically invariant property of a surface and it is defined as the largest number of non-intersecting simple closed curves that can be drawn on the surface without separating it [7]. Roughly, it is the number of holes in the surface mesh. For example, sphere (Figure-4a) and cube are genus-0, torus (Figure-4b) is genus-1 and double torus (Figure-4c) is genus-2.

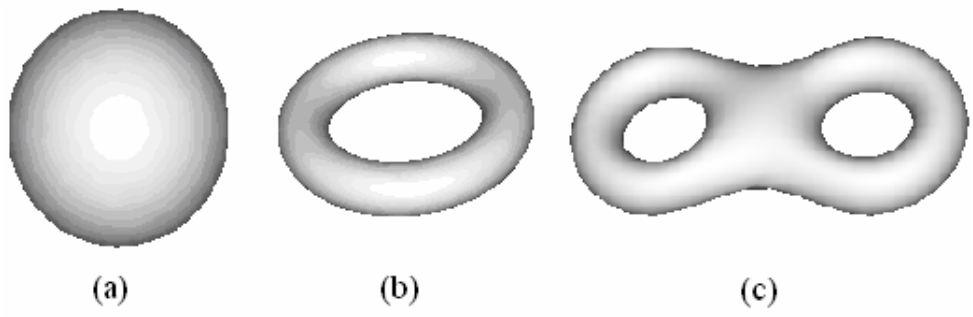

Figure 4. Genus-0 (a), Genus-1 (b), and Genus-2 (b) meshes

Euler [8] discovered the relation between faces, edges and vertices of a polyhedron and he stated that $V - E + F = 2$, where V, E, and F are the numbers of vertices, edges and faces respectively. This formula can be generalized to $V - E + F = 2(1 - G)$ for surfaces with genus $G \neq 0$.

2.1.4. Irregular, Semi-regular and Regular Meshes

The structure of a mesh can be irregular, semi-regular and regular. A vertex in a triangular mesh is called regular if it has 6 neighboring vertices for interior vertices or 4 for boundary vertices. These numbers are 4 and 3 for quadrangular meshes, respectively.

In a regular mesh (Figure-5c) all the vertices are regular and they have a lot of advantages over the irregular and semi-regular meshes due to their regular connectivity (Chapter 6). Semi-regular meshes (Figure-5b) are obtained by regular subdivision of a coarse initial mesh. The subdivided mesh contains both regular and irregular vertices but regular ones are generally much more than the irregular ones. Irregular meshes (Figure-5a), on the other hand, do not show any kind of regularities.

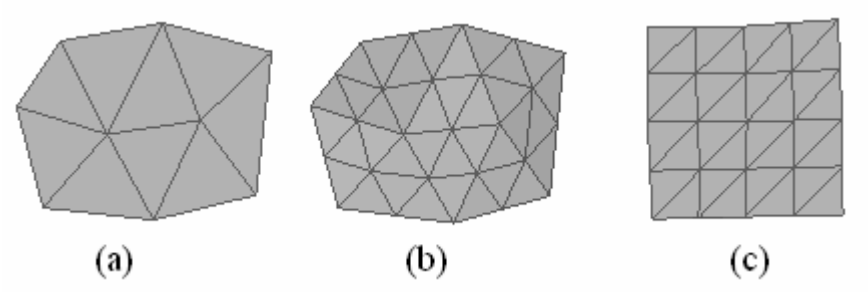

Figure 5. Irregular (a), Semi-regular (b), and Regular (c) meshes

2.2. Mesh Representations and Data Structures

In order to get use of meshes, we need to consider how to store meshes and how to process them. The answer of the former question is the file formats and the latter one is the mesh data structures.

2.2.1. Mesh File Formats

As mentioned earlier (section 2.1.1) all meshes have two common features, one of which is the geometry (defines the three dimensional position vectors of the vertices) and the other is the topology (defines the connectivity between the vertices, i.e. how the vertices are connected). When storing a mesh, what is

actually stored is the geometry and the connectivity of the mesh. Moreover, some meshes might have additional information, such as vertex normals, face normals, vertex colors and texture coordinates.

There are over five hundred mesh file formats [9]. This number is big because nearly each program, institute or library generates its own mesh file format, so it is very difficult to describe all of them here. In fact, one can easily generate his own file format as long as he can decide how to describe the mesh features. The most popular file formats are Wavefronts Obj file format [10], OFF (Object File Format) [11], STL (Stereolithography) [12], VRML(Virtual Reality Modeling Language) [13], and PLY (Polygon File Format)[14].

Most of the mesh file formats have both ASCII and binary versions. The ASCII version is more readable whereas the binary format is more compact. In Appendix-B a simple tetrahedron is represented in OBJ, OFF, STL ASCII and STL Binary formats as an example.

2.2.2. Mesh Data Structures

Meshes are stored in files (section 2.2.1), but if we want to process them we need to parse and load these files into our program, i.e. load the meshes to data structures. By processing a mesh it is meant that editing (changing the geometry and/or topology) and/or rendering (drawing on a window) that mesh.

Before choosing a data structure, one has to consider the topological and algorithmic requirements of the intended application [3]. Topological requirements are related with the kinds of meshes that will be represented by the data structure and the following questions should be answered: do we need to represent meshes with boundaries or closed meshes, manifold or non-manifold meshes, triangular or arbitrary polygonal meshes? Are the meshes regular, irregular or semi-regular? On the other hand, algorithmic requirements are seeking for the purpose of the user.

Do we need to modify the mesh geometry or topology or just render the mesh? Answers of these questions determine the choice of underlying data structure.

Most of the mesh processing algorithms require some sort of queries which we call them adjacency queries. Adjacency queries enable local and global traversal of meshes and some examples of them are given below:

- Which faces use this vertex?
- Which edges use this vertex?
- Which faces are adjacent to this face?
- Which edges border this face?
- Which vertices belong to this face?

A mesh data structure should be capable of answering these questions in an efficient way. There are many different data structures developed for representing polygonal meshes. It is not possible to describe all of them here, but most important ones from primitive to complex are described in the following sections.

2.2.2.1. Triangle Soup Data Structure

The simplest and first come in mind data structure is the triangle soup. A triangle soup data structure can be constructed as follows:

```
// struct to store a vertex
struct Vertex
{
    double x, y, z;
};

// struct to store a face
struct Face
{
    Vertex v1, v2, v3;
};
```

```
// geometry and connectivity
Face faceArray[NUM_FACES];
```

Triangle soup representation is not memory efficient since shared vertices are replicated. Also, connectivity information is not explicit, so adjacency queries have a high processing cost and take non-constant time.

2.2.2.2. Shared Vertex Data Structure

An improvement to the triangle soup data structure is the shared vertex representation. In this data structure, the memory drawback of triangle soup is overcome by storing vertex positions separately and giving references to the corresponding vertices from faces. These references may be pointers to vertices or integer indexes to vertex position arrays. One possible implementation of shared vertex data structure is given as follows:

```
// struct to store a vertex
struct Vertex
{
    double x, y, z;
};

// struct to store a face
struct Face
{
    // indexes to vertex array
    int v1, v2, v3;
};

// geometry of the mesh
Vertex vertexArray[NUM_VERTICES];

// connectivity of the mesh
Face   faceArray[NUM_FACES];
```

In the shared vertex data structure the connectivity information is still implicit, so adjacency queries take non-constant time and they have a high processing cost. However, updating the geometric information of a mesh is easier with respect to triangle soup.

2.2.2.3. Winged-edge Data Structure

Triangle soup and shared vertex are face based data structures, i.e., the connectivity information is given in terms of faces which results in an implicit connectivity. In order to have an explicit connectivity information, edge-based data structures were developed. Baumgart developed the first edge-based data structure, the winged-edge[15]. Winged-edge data structure is capable of only representing 2-manifold orientable meshes.

In the winged-edge data structure all edges are directed and clockwise ordering is used for traversing a face when looking from outside of the mesh. Edges are at the center of the data structure and they are associated with eight references (Figure-6). These references are: two vertices (start and end), two faces (left and right) and four edges (left predecessor and successor, right predecessor and successor). Left successor and predecessor edges are used for traversing the left face and right successor and predecessor ones are used when traversing the right face. These four edges form the wing of the selected edge (bold one in Figure-6), that is why this data structure is called as winged-edge. For vertices and faces it is enough to store a reference to one of their adjacent edge.

In the winged-edge data structure edge orientations are not globally consistent which means an edge is traversed in opposite directions when its left and right faces are traversed [16]. As a result, a case distinction (the calculation of orientation of an edge with respect to the traversed face) is necessary when traversing.

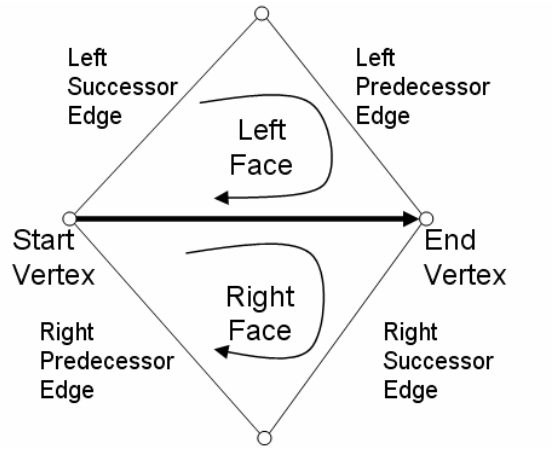

Figure 6. Winged-edge Data Structure

A sample implementation of a vertex, face and edge structures are given below. A 2-manifold, orientable mesh can be represented by these structures by organizing them in a linked list or dynamic array.

```
// struct to store a winged-edge
struct Wedge
{
    Vertex *vs, *ve;
    Face   *lf, *rf;
    Wedge  *ls, *lp, *rs, *rp;
};

// struct to store a vertex
struct Vertex
{
    double x, y, z;
    Wedge* edge
};

// struct to store a face
struct Face
{
    Wedge* edge
};
```

2.2.2.4. Half-edge Data Structure

The half-edge data structure is designed to overcome the orientation problem in the winged-edge data structure. Remember that, calculation of an edge's orientation with respect to the traversed face is necessary for winged-edge data structure which is inefficient.

In the half-edge data structure each edge is split in two half-edges, in opposite orientations such that all half-edges are oriented consistently in counter clockwise order around each face [17], [18]. For each half-edge, a reference is stored for the pair half-edge, for the adjacent face, for the next half-edge and for the end-vertex (Figure-7). Moreover for each vertex a reference is kept for an outgoing half-edge and similarly for each face a reference for an adjacent half-edge is kept.

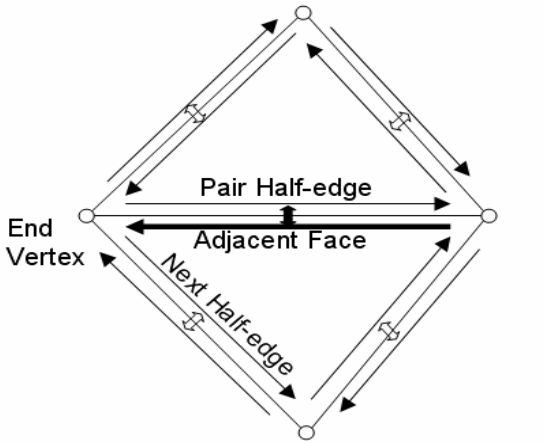

Figure 7. Half-edge Data Structure

The structure is capable of representing arbitrary 2-manifold orientable polygonal meshes and a sample implementation for the vertex, face and half-edge is given in the next page. A more complex implementation can be found in [19].

```
// struct to store a half-edge
struct Hedge
{
    Hedge *next, *pair;
    Face* face;
    Vertex *vert;
};

// struct to store a vertex
struct Vertex
{
    double x, y, z;
    Hedge* edge;
};

// struct to store a face
struct Face
{
    Hedge* edge;
};
```

Note that, the implementation given above realized the references in terms of pointers, but they may be realized in different ways also, for example indexes into data arrays can be used as well [3].

2.2.2.5. Other Data Structures and a Simple Comparison

Quad-edge data structure developed by Gubias and Stolfi in 1985 [20] is a variant of winged-edge data structure. Besides, orientable and non-orientable 2-manifold meshes it can represent both the mesh and it's duality at the same time. Directed edges data structure is based on the half-edges but it can only represent triangular meshes which may be preferred due to its efficient use of memory [3].

Up to now, some data structures are mentioned and most of them are used for representing 2-manifold meshes. In order to summarize their properties, a very simple comparison is done just for a summary in Table-1.

Table 1. A Simple Comparison of Mesh Data Structures

Data Structures	Represent Manifold/Non-Manifold	Represent Orientable/Non-Orientable	Adjacency Queries
Triangle Soup	Both	Both	Inefficient
Shared Vertex	Both	Both	Inefficient
Winged-edge	Manifold	Orientable	Efficient
Half-edge	Manifold	Orientable	Efficient
Quad-edge	Manifold	Both	Efficient
Directed-edge	Manifold	Orientable	Efficient

For data structures that are capable of representing non-manifold surface meshes see [22] and [23]. Among them, Radial-edge data structure is a popular one [21].

2.3. Mesh Libraries

Implementing memory and time efficient, robust, easy to use and generic mesh data structures is not an easy task. As the computer graphics, computational geometry and other application domains have been using these data structures frequently, the community developed some open source and freely available mesh libraries. The most popular ones are: Visualization Toolkit (VTK) [24], Computational Geometry Algorithms Library (CGAL) [25], and OpenMesh [26]. These libraries do not only implement data structures but also they are equipped with a number of standard operations and algorithms such as mesh simplification, subdivision, delaunay triangulation, etc.

2.3.1. VTK – Visualization Toolkit

VTK is developed in C++, in an object oriented manner by the Kitware Company [27]. It is open source and freely available for non-commercial use. Besides the C++ class library VTK also includes interface layers for Tcl/Tk, Python and Java. The latest version of the library 5.0.4 is released in January, 2008 and it supports a

wide variety of visualization, modeling and imaging algorithms such as mesh smoothing, contouring, simplification, cutting, delaunay triangulation, etc. [26], [28]. The library has two types of object models: graphics model and visualization model. The graphics model is used to visualize (rendering) data and the visualization model is responsible for implementing the algorithms [28].

VTK does not have a standard data structure such as winged-edge or half-edge. Instead, it has its own data structure based on data arrays. A very similar data structure is described in [41] and it is called as space-efficient vertex-triangle ring structure. In this data structure faces and vertices are kept in data arrays similar to shared vertex data structure but different than shared vertex, vertices have also indices to reach the faces which improves the efficiency [30]. VTK has also other data structures for different kinds of input data other than the irregular meshes.

2.3.2. CGAL – Computational Geometry Algorithms Library

The CGAL project started in 1996 as a consortium of seven sites in Europe and Israel [25]. The current project partners are Max Planck Institute for Computer Science (Germany), INRIA Sophia-Antipolis (France), Tel-Aviv University (Israel), Geometry Factory (France), ETH Zurich (Switzerland), Utrecht University (The Netherlands), Free University Berlin (Germany), Forth (Greece), and SciSoft (Argentina) and they released the latest version of CGAL 3.3.1 in August 2007.

CGAL is open source and freely available for non-commercial use. It is implemented in C++ and it offers many data structures and algorithms for the computer graphics, computational geometry, scientific visualization, computer aided design and some other research areas. A complete feature list can be found in [29]. Similar to VTK, CGAL has also many data structures for different types of input data. However, only the way that CGAL handles the irregular polygonal meshes is important for the purpose of our work, which is the half-edge data structure.

2.3.3. OpenMesh

OpenMesh is a generic and efficient data structure for representing polygonal meshes [26]. It is developed in C++ at the Computer Graphics Group, RWTH Aachen and based on the half-edge data structure. OpenMesh's latest stable release 1.1.0 was made available in 2007 by the group. Like the other two libraries, OpenMesh is also open source and freely available and it provides some set of standard geometry processing algorithms such as smoothing and simplification. However, it has less number of algorithms with respect to other two library.

CHAPTER 3

POLYGONAL MESH SIMPLIFICATION ALGORITHMS

Advances in technology and growing expectation for realism in many application domain result in more complex polygonal meshes than we actually want [6, 31]. Especially, with the evolution of modern geometry acquisition devices such as laser scanners polygonal meshes with millions of vertices can easily be generated. The complexity of such polygonal meshes sometimes causes problems in rendering, processing and transferring of these models. Mesh simplification algorithms try to eliminate the redundant portions of the meshes so that models can be rendered and processed for a lower cost without a significant loss in the models' visual content.

Mesh simplification algorithms are applied in many application areas, such as computer vision (range data is captured from range scanners), scientific visualization (iso-surfaces are extracted from volume data), remote sensing (terrain data is acquired from satellite photographs), geometric design and computer graphics (model representation and levels of detail) and finite element analysis (structural analysis of bridges, to simulate air flow around airplanes and to simulate electromagnetic fields) [31]. In all these areas developers use simplification mainly for three reasons [6]. These are:

1. **Eliminating the redundant geometry:** Mesh simplification algorithms are mostly used for eliminating the redundant geometry since redundancy of models results in loss of computation time. In scientific visualization, the output of the marching cubes algorithm [32] generates redundant triangles especially in the planar regions. Also before a finite element analysis the model under analysis may be subdivided first and than the redundant geometry can be removed later.

2. **Reducing the model size:** Meshes with a high number of vertices and faces allocate a lot of disk and memory space. For example, in the Aim@Shape repository [33] there are some meshes above 150 mega bytes. In order to reduce the size of such meshes, mesh simplification algorithms can be applied as well as mesh compression algorithms. Generally, simplification algorithms are lossy, i.e. the process can not be reversed, but on the other hand there are also algorithms which compress the models without any loss of data [34]. By using simplification or compression algorithms, disk and memory requirement of a model can be reduced and network transmission speed can be increased.

3. **Improving the run-time performance:** For interactive applications such as video games, flight simulators and computer aided design real time performance is very important. For such applications, mesh models can be simplified to multiple levels of detail so that each object in the scene can be displayed on the screen as a function of their relative distance to the viewer. When the object is closer to the viewer, it is rendered with a complex mesh. On the contrary, when it is farther a simplified version of the same mesh is used. This technique is called multiresolution modeling or levels of detail and utilizes the fact that distant objects do not require more resolution than the closer objects [35, 36].

3.1. Classification of Mesh Simplification Algorithms

Many different approaches and algorithms are proposed for mesh simplification in recent years which makes the classification of these algorithms very difficult. In fact, there are a number of papers classifying these algorithms in their own way. Luebke [6, 40] categorized mesh simplification algorithms with respect to the mechanism, view-dependence, error metric and topology. Erikson [39] also used a similar but less complex classification than Luebke. On the other hand, Heckbert

and Garland [31] preferred to classify them according to the types of the input meshes: height fields and parametric surfaces, manifold surfaces and non-manifold surfaces. Moreover, Pauly [3] and Talton [38] used the fact that some algorithms are iterative and some are single step. Pauly also mentioned one more different approach as resampling algorithms. Cignoni et al. [37] listed a number of different approaches and then gave a taxonomy of the simplification algorithms based on the characterization of the input/output data domain, simplification goal, the strategy adopted to drive/evaluate mesh approximation and whether the simplification process follows an incremental approach or not.

In this work, the top level classification of mesh simplification algorithms mostly depends on the work by [3, 38], i.e. simplification approaches are grouped into two categories: local approaches that iteratively simplify the input mesh by using a local topological operator and global approaches that are applied to the input mesh as a whole. Moreover, some parts of the other ideas from [31, 37, 39, 40] are placed in this main level classification. By this way, we believe that, the algorithms can be understood more easily.

3.1.1. Iterative Mesh Simplification Algorithms

Iterative mesh simplification algorithms remove one mesh element at a time. First, the cost of removing each element is calculated and depending on this calculation a priority queue is constructed. Then, the mesh element with the lowest error is deleted which results in a hole and a change in the geometry and connectivity of the mesh. After re-triangulating the resultant hole, the cost function is re-evaluated only for the nearby mesh elements that are affected by the removal and depending on this calculation the priority queue is reorganized. A very simple pseudo code for a general iterative mesh simplification algorithm is given in the next page.

```
FOR each element

     Calculate the cost of removal of the current
     element

END FOR

WHILE stop condition is not met

     Select the element with the lowest cost

     Remove the selected element

     Re-triangulate the resultant hole

     FOR each element affected from the removal

          Calculate the cost of removal of the
          current element

     END FOR

 END WHILE
```

Here the term “element” is used as a vertex, an edge, a face or a patch (a group of a face) of a mesh. The type of the element depends on the algorithm itself and it is closely related with the simplification mechanism or local operator used by the algorithm. In addition, calculating the error of removal of an element is also algorithm dependent. As a result, most of the iterative algorithms have a common framework and they are distinguished by the local simplification operator they use and how they calculate the error.

3.1.1.1. Local Mesh Simplification Operators

There are several choices for the basic removal operation but most popular ones are vertex decimation and edge collapsing. Vertex decimation was first proposed by Schroeder [41] but later edge collapsing became more popular such that today

nearly all iterative algorithms use some sort of edge collapsing. There are also other types of local operators such as triangle decimation and patch decimation but these are rarely used by algorithms since the major design goal is to keep the local operation as simple as possible [3]. Moreover, a triangle decimation can be thought as two edge collapse operations.

Vertex Decimation

Vertex decimation operates on a single vertex by deleting that vertex and re-triangulating the resulting hole. Deleting a vertex with valence n (the number of neighboring vertices), leaves a n-sided hole. This hole can be triangulated more than one way and each triangulation will contain n-2 triangles [1]. As a result, at the end of deleting a vertex and its adjacent triangles from the mesh and re-triangulating the created hole, the mesh is simplified by one vertex, two faces and three edges (Figure-8). Observe that Euler formula is invariable under this operation.

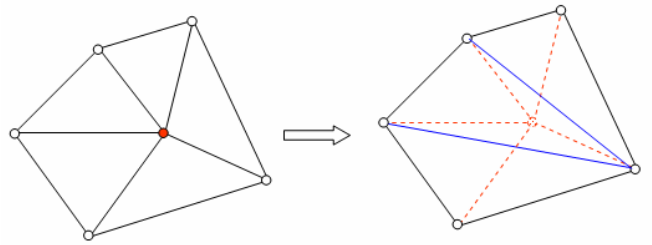

Figure 8. Vertex decimation and a sample choice of re-triangulation

Edge Collapse

Edge collapse operation was first proposed by Hoppe [2] and then it became the most common local simplification operator. There are three variants of the edge collapse operator, full-edge collapse operator, half-edge collapse operator and

vertex-pair collapse operator. Full-edge collapse operator simply abbreviated as edge collapse operator, takes two adjacent vertices and collapses the edge between them. The vertices at the ends of the collapsed edge are moved to a some new position (Figure-9).

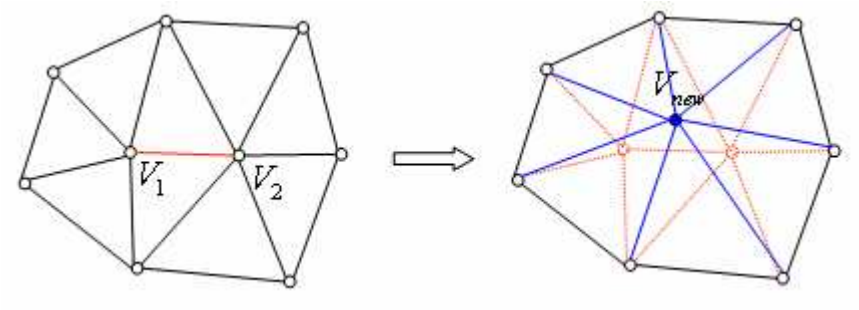

Figure 9. Full-edge collapse operation

Half-edge collapse operator [42] is a special case of the edge collapse operator and it collapses an edge to one of its end vertices (Figure-10), i.e. one of the end vertices is moved to the position of the other. The difference from edge collapse operation is that in half-edge collapse there is no freedom of selecting the position of the vertex to which the edge collapses.

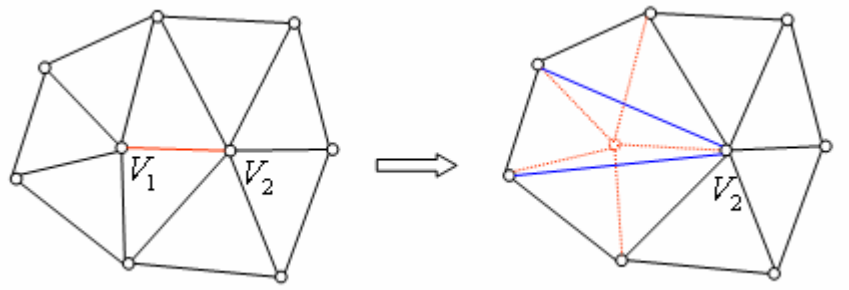

Figure 10. Half-edge collapse operation

Observe that half-edge collapse is also a subset of vertex decimation operator [35]. Remember that in vertex decimation at some stage a hole is generated. This hole can be triangulated in several ways and one of these triangulations is the same as a half-edge collapse.

The last variant of the edge collapse operation is the vertex-pair collapse [43, 44] where two vertices are collapsed in to a new vertex even if they are not connected by an edge (Figure 11). This operation reduces the number of vertices by one but keep the number of triangles and edges constant. Thus, Euler formula becomes inconsistent with a mesh if a vertex-pair collapse is applied. On the contrary, edge collapse and half-edge collapse operations reduce one vertex, two faces and three edges, and keep the mesh consistent with the Euler formula.

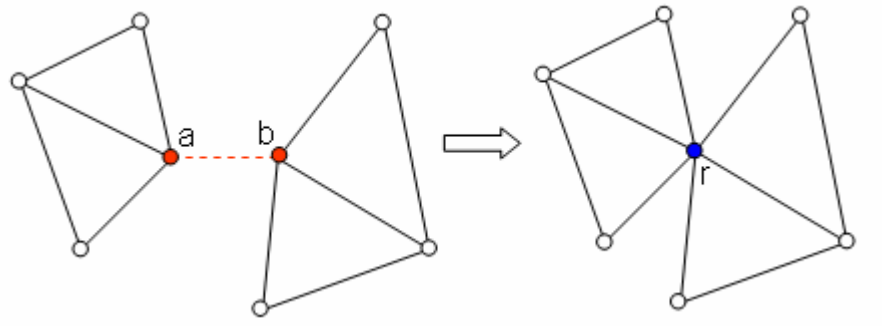

Figure 11. Vertex-pair collapse operation

Half-edge and edge collapse operations preserves the topology of a mesh but rarely an edge collapse operation may generate non-manifold topology. However, this behavior of edge collapse can be controlled and it can be prevented from generating non-manifold topology [35]. Vertex-pair collapse operation can connect unconnected portions of a mesh and changes the topology. Also, a genus can easily be closed by vertex-pair collapsing.

Comparison of Local Mesh Simplification Operators

The edge collapse and the vertex-pair collapse operators has much more freedom in determining the new vertex position but this requires solving a continuous optimization problem. On the other hand, half-edge operator does not produce new vertex positions and keeps a subset of the vertices of the original mesh which makes it memory and time (in terms of computation) efficient. Vertex decimation would be as efficient as half-edge collapse if the retriangulation step were handled more easier. Choosing the way of triangulation among the several ways is a discrete optimization problem.

Edge collapse, half-edge collapse and vertex decimation operators preserve the topology of the input mesh. Topology preserving algorithms preserve the manifold connectivity and do not close the holes of the mesh. Preserving the topology is crucial for some application domains such as medical imaging and finite element analysis. On the contrary, real-time rendering systems rarely need to preserve the topology because they need drastic simplifications that topology preserving algorithms cannot satisfy.

Edge collapse and its variants are easier to implement but since vertex-pair collapse operation generates non-manifold topology, a data structure that can handle non-manifold topology should be used and that may make vertex-pair collapse a bit hard to implement. Due to the re-triangulation step in 3D, the implementation of vertex decimation is not an easy task either.

3.1.1.2. Error Measurement Methods

Error measurements are used for selecting the candidate mesh element to apply a simplification operator such as vertex decimation or edge collapsing. These measures can be calculated locally or globally, i.e. the effect of removing an element is considered by looking at the local changes or global changes.

Local Error Measurement Methods

Distance to Average Plane

The simplest error metric is based on the distance measure from a vertex (a potential vertex for removal) to an average plane (Figure-12) [41]. An average plane is constructed using the adjacent triangle normals $\hat{n}_i$, centers $\vec{x}_i$ and areas A_i (Equation 1 and 2).

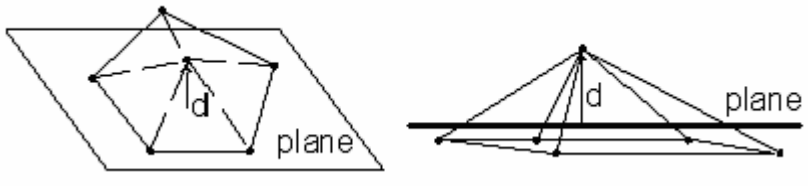

Figure 12. Distance to average plane

The normal $\vec{N}$ of the average plane is calculated by the average of the adjacent triangle normals weighted by the triangle areas.

$$\vec{N} = \frac{\hat{n}_1 A_1}{A_T} + \frac{\hat{n}_2 A_2}{A_T} + ... + \frac{\hat{n}_i A_i}{A_T} + ... + \frac{\hat{n}_m A_m}{A_T} = \frac{\sum \hat{n}_i A_i}{\sum A_i} \quad \text{and} \quad \hat{N} = \frac{\vec{N}}{\left|\vec{N}\right|} \qquad (1)$$

where $A_T = A_1 + A_2 + ... + A_i + ... + A_m$ and m is the total number of surrounding triangles. A point $\vec{\mathrm{X}}$ of the average plane is also calculated by the average of the adjacent triangle centroids ($\vec{x}_i$) weighted by the triangle areas.

$$\vec{\mathrm{X}} = \frac{\vec{x}_1 A_1}{A_T} + \frac{\vec{x}_2 A_2}{A_T} + ... + \frac{\vec{x}_i A_i}{A_T} + ... + \frac{\vec{x}_m A_m}{A_T} = \frac{\sum \vec{x}_i A_i}{\sum A_i} \qquad (2)$$

Then the distance (d) between the vertex and the average plane is calculated as

$$d = \left| \hat{N} \bullet (\vec{v} - \vec{\mathrm{X}}) \right| \tag{3}$$

where $\vec{v}$ is the position vector of the vertex. The error is proportional with the distance, i.e. error is low if distance is low and error is high if distance is high.

Distance to average plane error metric is mostly used for eliminating the redundant geometry in the planar regions and keeping the sharp features of meshes. This is why the distance is proportional with the error metric. This error metric is used by [41] with a vertex decimation operator. A vertex is selected for removal with the lowest error or with the lowest distance d to the average plane.

Maximum Supporting Plane Distance

Another local error metric first used by Ronfard and Rossignac [45] is the maximum squared distance between a simplified vertex and supporting planes associated with that vertex. Ronfard and Rossignac utilizes this error metric with an edge collapse simplification operator.

Each triangular face adjacent to a vertex defines a plane and we call each of these planes as a supporting plane for that vertex. Initially, in the original mesh, all vertices are associated with its own supporting planes. For example, the set of supporting planes $Sp(V_1)$ and $Sp(V_2)$ for vertices V_1 and V_2 in Figure-13 are as follows:

$$Sp(V_1) = \{t_1, t_2, t_3, t_4, t_5\}, \text{ and } Sp(V_2) = \{t_2, t_3, t_6, t_7, t_8, t_9\}$$

When an edge, let's say $|V_1V_2|$ is collapsed, a simplified vertex, V_{new} is generated and the set of supporting planes for this new vertex is calculated as the union of supporting planes from the two edge vertices (Equation 4). This set of planes grows larger and larger as the new edges are collapsed.

$$Sp(V_{new}) = Sp(V_1) \bigcup Sp(V_2) = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9\} \tag{4}$$

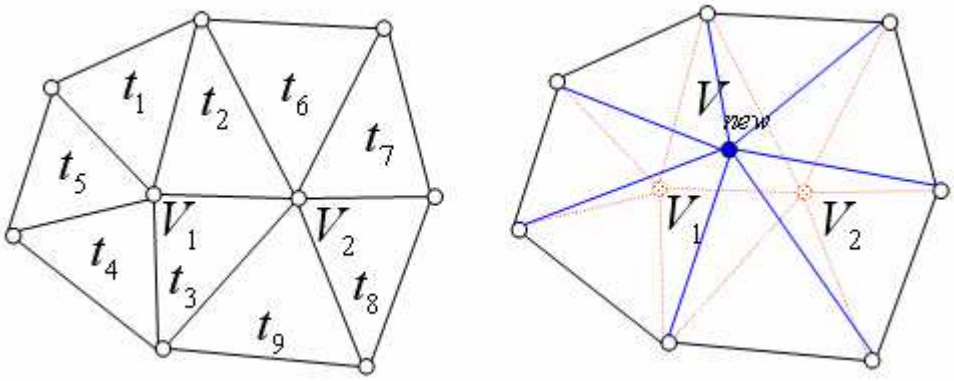

Figure 13. The set of supporting planes for V_1 and V_2

The plane equation for a supporting plane t_k can be written as:

$$a_k x + b_k y + c_k z + d_k = 0 \tag{5}$$

where $\hat{n}_k = (a_k, b_k, c_k)$ is the unit normal of t_k and d_k is just a constant. Using (5), a squared distance function $D_k^2(v)$ which calculates the squared distance between the given plane and a vertex $v = (x, y, z)$, can be defined as:

$$D_k^2(v) = (a_k x + b_k y + c_k z + d_k)^2 \tag{6}$$

Then the error for a potential edge collapse operation is determined by choosing the maximum of the squared distances between the supporting planes and the candidate new vertex:

$$E = \max D_k{}^2(V_{new}) \tag{7}$$

where E is the error and k is a positive integer such that $t_k \in Sp(V_{new})$.

Quadric Error Metric

Similar to Ronfard and Rossignac [45], Garland also associated a set of planes with every vertex of a model [44, 46] but he replaced the maximum supporting plane distance error metric in [45] with the sum of squared supporting plane distances. Garland expressed this error metric in a quadric form and named it as quadric error metric. Quadric error metric is more compact and easy to calculate such that it is a quite popular error metric since then.

The standard representation of a plane in (5) can be written as:

$$n^T v + d = 0 \tag{8}$$

where $n = [a \quad b \quad c]^T$ is the unit normal and d is a scalar constant. Given a plane in the form of (8) and a vertex $v = [x \quad y \quad z]^T$ equation (6) can be rewritten as:

$$D^2(v) = (n^T v + d)^2 \tag{9}$$

For a vertex v with an associated set of supporting planes, Garland [46] defined the error at this vertex as

$$E = \sum_i D^2{}_i(v) = \sum_i (n_i{}^T v + d_i)^2 \tag{10}$$

He also put the equation (9) in a new form:

$$\begin{aligned} D^2(v) &= (n^T v + d)^2 \\ &= (v^T n + d)(n^T v + d) \\ &= v^T n n^T v + 2 d n^T v + d^2 \\ &= v^T (n n^T) v + 2 (dn)^T v + d^2 \end{aligned} \tag{11}$$

where nn^T is a 3x3 matrix (outer product matrix). Thus, a fundamental quadric $Q = (nn^T, dn, d^2)$ can be defined for a given plane in the form of (8) such that

$$Q(v) = v^T (nn^T) v + 2 (dn)^T v + d^2 \tag{12}$$

Garland used the term quadric because the iso-surfaces $Q(v) = \varepsilon$ are quadric surfaces [46]. Furthermore, evaluating the quadric (12) for a plane is completely equivalent to evaluating the squared distance to that plane. As a result, the error equation (10) can be rewritten as

$$E = \sum_i D^2{}_i(v) = \sum_i Q_i(v) \tag{13}$$

In order to make the calculation of error is more easier, the quadric Q defined in (12) can be thought as a homogeneous matrix [44]:

$$Q = \begin{bmatrix} nn^T & dn \\ dn^T & d^2 \end{bmatrix} = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \tag{14}$$

Once the quadric matrix Q in (14) is calculated for a plane, $Q(v)$ in (12) can be easily calculated by:

$$Q(v) = \tilde{v}^T Q \tilde{v} \tag{15}$$

where $\tilde{v} = [x \quad y \quad z \quad 1]^T$ is the vertex position in homogenous coordinates.

This error metric can be used with all variations of edge-collapse simplification operator and a pair of vertices (V_1 and V_2) can be chosen for collapsing which will result in the lowest error. The quadric for the simplified vertex (V_{new}) is calculated simply by adding the two quadrics:

$$Q_{new} = Q_1 + Q_2 \tag{16}$$

and the error for the new vertex is:

$$E_{new} = \tilde{v}_{new}^T Q_{new} \tilde{v}_{new} \tag{17}$$

By this representation of quadric error metric, the explicit calculation of squared distances in [45] is replaced by just adding two 4x4 matrices which results in a very fast algorithm relatively.

Global Error Measurement Methods

Hausdorf Distance

Hausdorf distance is a way of calculating the distance between two point sets and since surfaces can be expressed as a set of continuous points, it also applies to surfaces. However, since a surface is composed of infinitely many points,

calculating the Hausdorf distance is very expensive for surfaces. In order to overcome this problem, surfaces can be sampled in different ways to measure Hausdorf distances [47].

Hausdorf distance is used by [48] to control the approximation error between the original and simplified mesh models. At each step of the algorithm a vertex is removed from the mesh whose removal makes the lowest contribute to the overall Hausdorf distance without exceeding a predefined Hausdorf distance.

Hausdorf distance between two point sets, A and B can be defined as follows [35]:

$$H(A,B) = \max(h(A,B), h(B,A)) \tag{18}$$

where $h(A,B)$ denotes the one-sided Hausdorf distance and it is defined as:

$$h(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\| \tag{19}$$

Observe that when calculating the one-sided Hausdorf distance $h(A,B)$, for each point in A the closest point in B is found and the maximum of these is taken as one-sided Hausdorf distance. One-sided Hausdorf distance is not symmetric, thus $h(A,B)$ is not necessarily equal to $h(B,A)$.

Simplification Envelops

Simplification envelops [49] are two non self-intersecting offset surfaces constructed on each side of the original surface using a user specified offset (Figure-14). The surface is then simplified in the volume that is generated by two simplification envelops, thus the error is bounded globally. The amount of simplification is controlled by the offset used for constructing the simplification envelops. Moreover, this offset value can be changed in different regions of the

original mesh, so that the user can have a control over which regions of an object should be approximated more and which ones should be less.

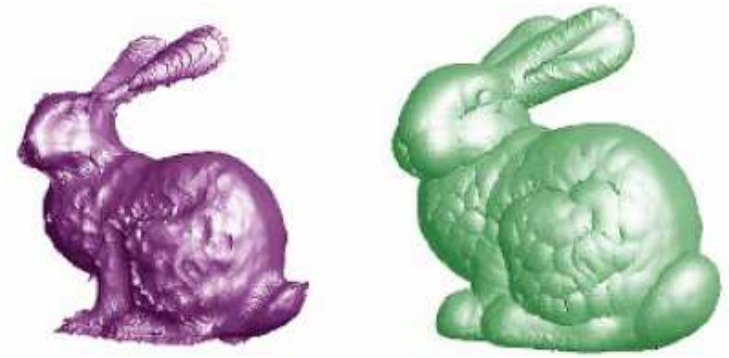

Figure 14. Inner and outer simplification envelops for a bunny model [49]

3.1.2. Single Pass Mesh Simplification Algorithms

Single pass algorithms process the input mesh as a whole and generate the simplified mesh without an iterative process. Generally, iterative algorithms give better results but single pass algorithms are much faster than the iterative algorithms. Vertex clustering algorithms are the dominant algorithms among the single pass algorithms.

3.1.2.1. Vertex Clustering

Vertex clustering simplification algorithm was first proposed by Rossignac and Borrel [50] in 1993 and since then several variations of the algorithm have been proposed. Simply, the original algorithm works as follows: first a weight is assigned to each vertex of the input mesh which shows the importance of that vertex. Then, the bounding space around the input mesh is partitioned into 3D cells according to a given resolution. An analogous of partitioning in 2D is shown in

Figure-15. Vertices inside each cell is clustered to the most important vertex within that cell and after removing the degenerate faces produced in the clustering step, the simplified mesh is obtained.

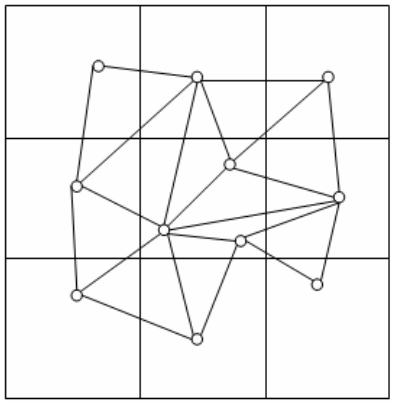

Figure 15. Partitioning a bounding box of a mesh model into cells in 2D

Rossignac and Borrel assigned higher importance to vertices attached to large faces (edges of large faces are longer than the others) and to vertices in regions of high curvature (the maximum angle between all pairs of attached edges to a vertex is small in regions of high curvature). Clearly, the level of the simplification can be determined by the resolution of the bounding grid. A coarse grid will simplify the input mesh drastically whereas a fine grid will perform minimal reduction.

One of the key issues of the vertex clustering algorithms is that where to cluster the vertices within each cell should be decided. In the original algorithm, all vertices in each cell is clustered to the most important vertex. The importance of a vertex can be determined in many ways. Rossignac and Borrel gave high importance to the vertices near the large faces and near the high local curvature. Giving high importance to the boundary vertices may also be beneficial for keeping the boundary of the input mesh as constant. Another approach is that, instead of clustering all vertices within a cell to one of the vertices, a new place is chosen for

clustering. Simple ideas such as taking the center of each cell, the average or median of vertices rarely results in good results [3]. Lindstrom [51] realized the fact that merging n vertices within a cell is equivalent to performing any sequence of (n-1) vertex-pair contractions until a single vertex remains within the cell. He then used a quadric error metric to determine the clustered vertex position. This quadric error metric is developed by Lindstrom and Turk [55] for controlling the volume and area changes of the original mesh.

Vertex clustering algorithms are generally easy to implement and they are very fast with respect to the iterative algorithms [3]. Neither they require manifold topology for input meshes nor they preserve the topology.

CHAPTER 4

POLYGONAL MESH SIMPLIFICATION ALGORITHMS EMBEDDED IN MESH LIBRARIES

Since polygonal meshes are very popular in computer graphics and geometry processing they are used widely which obligates the community to develop mesh processing libraries. Although there are a number of these libraries the most popular and common ones are: Visualization Toolkit (VTK) [24], Computational Geometry Algorithms Library (CGAL) [25], and OpenMesh [26]. These three libraries supply interfaces to implement some outstanding mesh simplification (both iterative and single pass) algorithms.

4.1. Polygonal Mesh Simplification with VTK

There are three different mesh simplification algorithms that come with the currently latest version (5.0.4) of Visualization Toolkit (VTK) library [24, 28]. Previously, the library was supporting four different algorithms but one of them vtkDecimate was discarded in the latest version. The remaining supported algorithms are: vtkDecimatePro, vtkQuadricDecimation and vtkQuadricClustering.

4.1.1. vtkDecimatePro

vtkDecimatePro algorithm is similar to the algorithm originally described in [41] with four major differences. First, this algorithm does not necessarily preserve the topology of the input mesh. Second, if the parameters of the algorithm are set accordingly, the algorithm guarantees to reach the user specified reduction rate. Third, edge collapse simplification operator is used instead of vertex decimation and finally, progressive mesh representations are used internally as described by Hoppe [52], but the algorithm only gives the final simplified mesh as output.

vtkDecimatePro accepts only triangular meshes but it can be applied to other polygonal meshes once they are triangulated.

The first step of vtkDecimatePro is the classification of input mesh vertices which characterizes the local geometry and topology of vertices. Each vertex is assigned one of five possible classifications: simple, complex, boundary, interior or corner vertex [41] (Figure-16).

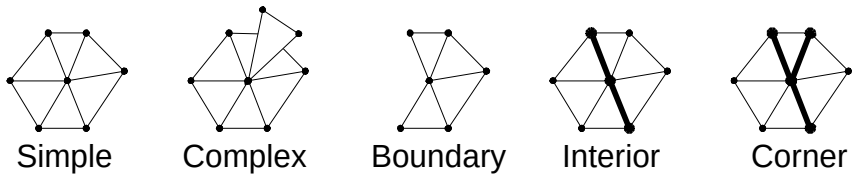

Figure 16. Vertex categories in vtkDecimatePro

A simple vertex is surrounded by a complete cycle of triangles, each of which shares a single edge with the vertex. Simple vertices can be further classified as interior or corner according to the geometry of the surrounding triangles. If the dihedral angle (the angle between two planes) between two adjacent triangles is greater than a user specified feature angle then a feature edge exists (Figure-17). If a simple vertex is used by feature edges then the vertex is either an interior edge vertex or a corner vertex. For an interior edge vertex the number of feature edges is exactly two, whereas for a corner vertex the number of feature edges is either one, three or more. Boundary vertices differ in simple vertices that the set of triangles does not form a complete cycle. Simple, interior, corner and boundary vertices are all manifold vertices and non-manifold vertices are called as complex vertices.

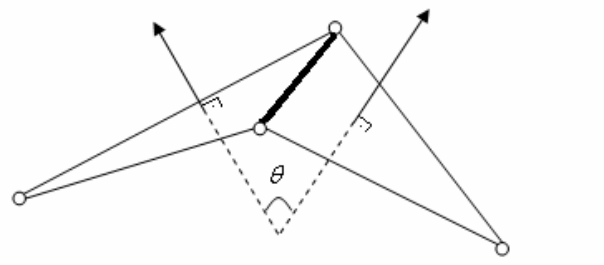

Figure 17. Dihedral angle and a feature edge

After classifying the vertices, they are inserted into a priority queue. The priority is based on the distance to average plane error metric. This error metric is used for simple and corner vertices and for boundary and interior vertices a variance of this metric, distance to edge is used (Figure-18). In this case, the distance to the line defined by the two vertices creating the boundary or feature edge is measured. While inserting the vertices in to the priority queue, vertices that can not be deleted are skipped. Mostly, these vertices are the non-manifold complex vertices and the sharp interior edge or corner vertices. Then each vertex in the priority queue is deleted one by one until the priority queue is empty. Next, if the desired reduction rate is not reached and the algorithm is allowed to modify the topology, the algorithm splits the mesh into separate pieces along sharp edges or at non-manifold vertices. Then the algorithm starts to evaluate these newly generated vertices as described previously until the desired reduction rate is achieved.

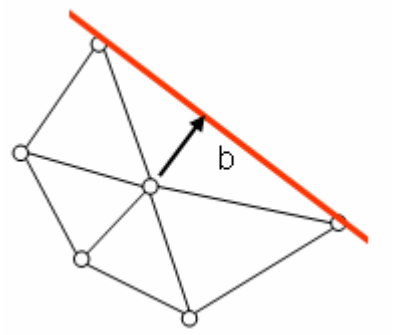

Figure 18. Distance to line error metric: a variance of the distance to average plane metric

vtkDecimatePro is a highly controllable algorithm such that the user can determine nearly all of the parameters of the algorithm. There are thirteen (13) different parameters that can be set, but six of them are more important than the others. The first parameter TargetReduction is the desired reduction rate whereas four of the remaining are PreserveTopology, Splitting, BoundaryVertexDeletion and MaximumError are used to control the topology modification. In order to modify the topology PreserveTopology should be disabled and Splitting and BoundaryVertexDeletion should be enabled. Moreover, MaximumError should be set to its maximum value which is VTK_DOUBLE_MAX. The remaining relatively important parameter is the AccumulateError. As the vertices are deleted the error may be accumulated similar to [44, 45], if AccumulateError parameter is enabled.

4.1.2. vtkQuadricDecimation

Similar to vtkDecimatePro, vtkQuadricDecimation is also an iterative mesh simplification algorithm. vtkQuadricDecimation used the quadric error metric as described by Garland [44, 46]. Later, Garland and Heckbert generalizes this approach to deal with appearance attributes such as vertex normals and texture coordinates [53]. Moreover, Hoppe improved this work of Garland and Heckbert in [54] and obtained better results. The algorithm vtkQuadricDecimation is based on the works of both Garland and Hoppe and uses the edge collapse simplification operator.

The algorithm operates similar to a classical iterative mesh simplification algorithm using an edge collapse operator and quadric error metric as a cost function. It stops execution until the desired level of reduction is reached or topological constraints prevent further reduction. Although it is written in the documentation of the algorithm that topological constraints may prevent further reduction, most of the time the user specified reduction rate is achieved and the

topology of the input mesh is not preserved (non-manifold vertices and edges may be generated and holes may be closed).

There are two main input parameters supplied by the user. These are TargetReduction and AttributeErrorMetric. TargetReduction is the desired reduction rate and AttributeErrorMetric is used to control the quadric error metric. If the user sets the AttributeErrorMetric parameter of the algorithm, then the mesh attributes are taken into account in the quadric error metric. The user can also select which attributes to consider and their corresponding weights in the error metric.

4.1.3. vtkQuadricClustering

vtkQuadricClustering is a vertex clustering algorithm and it is mostly based on the paper [51] by Lindstrom in which the clustered vertex position is determined by a quadric error metric. Remember that vtkDecimatePro and vtkQuadricDecimation only accept triangular meshes. On the other hand, vtkQuadricClustering accepts all types of meshes but it gives better results if the faces of the input mesh are convex polygons.

vtkQuadricClustering starts with the decomposition of the bounding box of the input mesh into three dimensional cells. The resolution of the decomposition is either selected by the user or the algorithm determines the resolution by itself. After dividing the bounding box into cells, the algorithm determines which vertices are in which cells. If two or three vertices of a triangle fall in the same cell, the triangle is discarded. The other remaining triangles (all three vertices of these triangles fall in a different cell) are used to calculate the quadric of the cells. A cell quadric is defined as the sum of the quadrics of triangles those have a vertex inside that cell. Lindstrom defined the quadric of a triangle $t = (x_1, x_2, x_3)$ as follows [51]:

$$Q = \begin{pmatrix} A & -b \\ -b^T & c \end{pmatrix} = nn^T \tag{20}$$

$$n = \begin{pmatrix} x_1 \times x_2 + x_2 \times x_3 + x_3 \times x_1 \\ -[x_1, x_2, x_3] \end{pmatrix} \tag{21}$$

where x_i is a position vector and n is a 4-vector made up of area weighted triangle normal and the scalar triple product of its tree vertices. This quadric is constructed in a manner that it can be used to measure volume changes after an edge collapse. Remember that clustering all vertices within a cell to a vertex can be considered as a successive sequential vertex pair collapses [51].

After all triangles are processed and the quadric of every cell is calculated, the representative vertex for each cell is computed using the quadric for that cell. This vertex position is either chosen among the vertices within each cell or a new vertex position is calculated. No matter which option is selected by the user, the vertex position is calculated in an optimal way such that the resultant vertex minimizes the volume distortion.

vtkQuadricClustering can also be used for out-of-core simplification. That is, very huge meshes that can not be loaded into the main memory at once can be divided into multiple pieces and each piece is simplified individually. Then, the simplified pieces combined together and the simplified mesh is obtained.

There are two main parameters for the algorithm: AutoAdjustNumberOfDivisions and UseInputPoints. If the user wants to determine the resolution of the decomposition step of the algorithm, she can disable the auto adjustment property of the algorithm and indicate the number of divisions along the three axes. Otherwise, the number of divisions are determined by the algorithm logically. The other parameter, UseInputPoints is used to select the behavioral of the algorithm

when determining the representative vertex. If the parameter is enabled the position for the representative vertex is chosen among the vertices within each cell. Otherwise a new position is calculated.

4.2. Polygonal Mesh Simplification with CGAL

CGAL (version 3.3.1) mesh simplification package can only be used to simplify oriented manifold triangular meshes using edge collapse simplification operator. The package has a generic iterative mesh simplification algorithm such that the user can select two types of different strategies: Lindstrom-Turk and Edge-Length Midpoint. The strategies differ from each other in cost function and vertex placement.

Besides the strategy the user also has to determine one more mandatory parameter. This parameter is the stop predicate that stops the algorithm execution when a desired level of reduction is achieved as long as topology constraints are not violated (topology is preserved). Stop predicate can be selected as a ratio of edges between the simplified and the original meshes and also a direct number of desired edges may be given. One more choice about the execution of the algorithm is that whether to use enriched polyhedron or not. Enriched polyhedrons have an extra id field for both half-edges and vertices to store edge indexes. This selection does not change the quality of the simplified mesh but it speeds up the algorithm in most cases if input mesh is not very simple.

4.2.1. Lindstrom – Turk Strategy

The first algorithm in the library is named as Lindstrom-Turk strategy which is the implementation of the paper by Lindstrom and Turk [55, 56]. Lindstrom and Turk approach the problem of finding a vertex (replacement vertex) position for an edge collapse as a continuous optimization problem such that the chosen replacement vertex minimizes a cost function. Hoppe [2] also used optimization for mesh

simplification but his approach is related with the whole process of simplification. Hoppe's energy function is composed of three components. The first component is used to minimize the squared distances between the vertices of the original and simplified meshes while the second one penalizes high number of vertices and the last one is the regularizing term helping to find a minimum.

Lindstrom and Turk computed the replacement vertex position as the solution to a system of three linearly-independent linear equality constraints. First three constraints are determined by considering the volume and area preservation. Then using these constraints some other constraints are obtained by minimizing a quadric objective function [29]. Thus, more than three constraints are determined and among them three linearly-independent constraints are chosen with respect to an importance. If three linearly-independent constraints can not be found, then the edge will be discarded from collapsing. When three compatible constraints are found in the form of $a_i^T v = b_i$, then the replacement vertex position v is computed as the solution to linear system:

$$Av = b, v = A^{-1}b \tag{22}$$

Observe that each constraint is a plane equation and a_i is the normal vector whereas b_i is a scalar. Combining the tree constraints a 3x3 matrix A and 3-vector b is generated. After finding the replacement vertex position, cost of collapsing the investigated edge is calculated as the weighted sum of optimization terms related with the area and volume preservation.

4.2.2. Edge-length Midpoint Strategy

Edge-Length Midpoint Strategy assigns higher importance to relatively long edges than the others and the replacement vertex position is chosen as the midpoint of the edge being collapsed. Since there is no optimization, and replacement vertex is

calculated very easily, this strategy runs very fast, but on the other hand it may result in a poor quality.

4.3. Polygonal Mesh Simplification with OpenMesh

OpenMesh (version 1.9.5) mesh decimation framework has an iterative mesh simplification algorithm which can be modified by the user. The algorithm uses half-edge collapse operator for simplification and vertex to be removed is determined by decimating modules. There are five different decimating modules that can be selected and some of them can operate in two different modes: binary mode and non-binary mode. In the binary mode the decimating module returns LEGAL_COLLAPSE or ILLEGAL_COLLAPSE for a potential half-edge collapse with respect a user specified criterion. On the other hand, in the non-binary mode a floating point priority is calculated which is used to feed a priority queue. Mesh decimation framework allows to use more than one decimating module at the same time with some restrictions. The framework allows only one binary module. Every further module must be a binary module. Moreover, during the decimation binary modules are evaluated first, if a potential half-edge collapse passes the test of binary module then it is inserted to a priority queue whose priority is calculated by the non-binary module.

Besides the triangular and manifold meshes OpenMesh mesh decimation framework also accepts non-manifold and non-triangular meshes. However, the latter case requires a time consuming pre-processing. Faces that contain non-manifold vertices and edges are considered as isolated faces and non-triangular faces are triangulated.

4.3.1. Quadric Module

Quadric decimating module can be used in both binary and non-binary modes. In the non-binary mode, quadric module computes half-edge collapse priority based on quadric error [44]. However, in the binary mode, the module allows the collapse if the calculated quadric error is not bigger than the user specified maximum quadric error.

4.3.2. Roundness Module

Roundness decimating module can be used in both binary and non-binary modes. In the non-binary mode the module returns a normalized roundness value associated with a potential half-edge collapse. Roundness is calculated for a each face that would be created after a half-edge collapse and minimum of them is returned.

Roundness of a triangle ABC (Figure-19) is computed by dividing the radius of the circumcirle by the length of the shortest edge.

$$Roundness = \frac{r}{\min(a,b,c)} = \frac{abc}{4A(ABC)\min(a,b,c)} \tag{23}$$

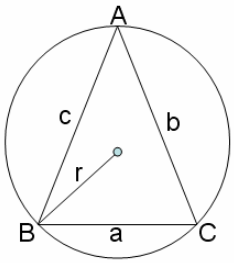

Figure 19. Circumcircle of a triangle $\triangle ABC$

A roundness value can be calculated for every triangle and among them the minimum is achieved by the equilateral triangles. For all equilateral triangles roundness is $1/\sqrt{3}$ and this value can be used to normalize the roundness.

$$Roundness_{Normalized} = \frac{1/\sqrt{3}}{Roundness} \tag{24}$$

Thus, for an equilateral triangle normalized roundness is equal to 1 and every other triangles have a normalized roundness smaller than 1. Considering these facts, normalize roundness can be used to measure the similarity of an arbitrary triangle with an equilateral one. If a triangle is close to an equilateral triangle, then its' normalized roundness is close to 1, otherwise it is close to 0. Roundness decimating module tries to keep the well shaped triangles which are close to equilateral triangles. In the binary mode the normalized roundness returned for a potential half-edge collapse is tested against a user supplied minimum roundness. If the calculated normalized roundness is less than the minimum roundness than LEGAL_COLLAPSE is returned, otherwise ILLEGAL_COLLAPSE is returned.

4.3.3. Normal Flipping Module

Normal Flipping decimating module is only used in binary mode. The module returns LEGAL_COLLAPSE or ILLEGAL_COLLAPSE depending on the angular deviation between the face normals of the original faces and normals of the faces after the half-edge collapse. If each deviation is below a given threshold, then half-edge collapse is allowed, otherwise it is not allowed.

4.3.4. Independent Sets Module

Independent Sets decimating module can only be used in binary mode and after a half-edge collapse operation is done, it locks the neighboring vertices around the remaining vertex to prevent further half-edge collapses. Thus, a half edge collapse is not allowed if it is related with a locked vertex.

4.3.5. Progressive Mesh

Progressive mesh decimating module does not contribute to the decimation but it collects information during the simplification process. Then, this information is written to a file so that the user can generate progressive mesh representation [52]. Using the generated file, original model can be entirely reconstructed.

CHAPTER 5

COMPARISON OF POLYGONAL MESH SIMPLIFICATION ALGORITHMS ON A LIBRARY BASIS

5.1. Comparison Strategy

Three commonly used mesh libraries VTK, CGAL and OpenMesh have a number of different polygonal mesh simplification algorithms. Nearly, all of these algorithms can be extensively modified which makes the comparison very difficult. Thus, although some of these algorithms can be parameterized in many ways, only the main lines of these algorithms are included in the comparison.

VTK has three simplification packages (section 4.1). Two variations of vtkDecimatePro and vtkQuadricDecimation are taken into account and together with the vtkQuadricClustering, five different simplification methods are included in the comparison from VTK. In the first variation of vtkDecimatePro, accumulate error is not used whereas in the second one it is enabled and variations of vtkQuadricDecimation differ from each other whether mesh attributes (vertex normals) are taken into account or not.

CGAL implements a mesh simplification framework where the user can select among two different strategies: Lindstrom-Turk Strategy and Edge-Length Midpoint Strategy. Both of these algorithms are included in the comparison.

OpenMesh also has a mesh simplification framework where algorithms are implemented as decimating modules. There are five different modules but only two of them can be used alone (modules that can operate in the non-binary mode). Quadric module is included in the comparison alone but the other less promising non-binary module Roundness is used with the binary Normal Flipping decimating

module. Normal Flipping module is started with a 5 degree of flipping angle and if the desired reduction rate can not be achieved, the flipping angle is increased by 5 degrees more until the desired level of reduction is achieved. The other remaining Progressive Mesh and Independent Sets modules are discarded. Progressive Mesh module is not a real decimating module since it only collects information about the simplification and Independent Sets module is not thought as useful as the others because locking the neighboring of a decimated vertex prevents simplification more than desired.

All of the algorithms are mainly run on two different triangular surface mesh models. The first model is the Stanford Bunny model that has been used for a long time for testing nearly all kinds of mesh processing algorithms and it is freely available in the Stanford 3D scanning repository [70]. Bunny model has 34834 vertices, 69451 faces and 104288 edges. The second model is the Bumpy Torus model that is available in the Aim@Shape model repository [33] and is composed of 16815 vertices, 33630 faces and 50445 edges. Each algorithm is executed in nine different reduction rates and the obtained results are compared with the metro tool. Metro tool calculates the deviance between the original and simplified models in terms of maximum and mean geometric errors. The calculated errors are normalized by dividing the obtained results with the maximum of the corresponding errors.

While choosing the models for comparison, it is intended to select models with different properties from each other. By this way, the algorithms can be tested with different kinds of input mesh models. The first selected model, Stanford Bunny is twice as complex as the Bumpy Torus model. Moreover, Stanford Bunny is not a closed model and it has five boundaries with genus-0. On the other hand, Bumpy Torus is a closed mesh with genus-1 and the high curvature regions of the Bumpy Torus model is much more than the Stanford Bunny. Despite all these differences they have also some common properties such that both meshes are manifold, orientable and irregular.

The selected reduction rates are started from “10%” and continues with “20%”, “30%”, “40%”, “50%”, “60%”, “70%”, “80%”, and “90%”. All rates are calculated in terms of face numbers of the models. Moreover, all of the algorithms are executed on a core duo 2.4 GHz, 3GB RAM, Windows XP system.

5.2. Comparison of the Mesh Libraries with respect to their Approaches to Polygonal Mesh Simplification

All of the simplification algorithms implemented in the concerned mesh libraries are a kind of iterative mesh simplification algorithm except vtkQuadricClustering which makes it unique. Also, it is the only algorithm that can be used in out-of core simplification. However, in this algorithm the user can not specify a desired reduction rate although the resolution of the simplified mesh may be controlled by choosing the appropriate parameters.

Iterative algorithms are more or less different from each other in how they decide which vertex to be removed from the mesh. In all these algorithms a reduction rate is specified but whether to achieve this rate or not is algorithm dependent. Generally, if an algorithm does not preserve the topology, the desired reduction rate is achieved all the time. These and some other properties of the algorithms have already been discussed in the previous sections but it may be helpful to summarize the behaviors of these algorithms in tables (Table-2, 3 and 4).

CGAL and OpenMesh have a mesh simplification framework where the choice of desired algorithm is made by changing some parameters. On the other hand, in VTK each algorithm is implemented individually. OpenMesh differs from other libraries that it supplies five modules in two categories: binary and non-binary modules. Any of these modules can be grouped into a set (at least one non-binary module is required) and then used for a simplification. One of the supplied modules in OpenMesh is the Progressive Mesh module which is not a true

decimating module. It only collects information about the history of the simplification so that the original mesh can be reconstructed from this information [52]. vtkDecimatePro also utilizes the progressive meshes but it does not give this information to the user.

Table 2. Polygonal Mesh Simplification with CGAL

CGAL		
Algorithms	Lindstrom-Turk Strategy	Edge-Length Midpoint Strategy
Input Mesh Type	Triangular	Triangular
	Manifold	Manifold
Preserve Topology	Yes	Yes
Simplification Method	Edge Collapse	Edge Collapse
Error Metric	Shape Preserving Opt.	Edge-length

Table 3. Polygonal Mesh Simplification with VTK

VTK			
Algorithms	vtkDecimatePro	vtkQuadricDe-cimation	vtkQuadricClus-tering
Input Mesh Type	Triangular	Triangular	Polygonal
	Manifold & Non-Manifold	Manifold & Non-Manifold	Manifold & Non-Manifold
Preserve Topology	Depends on the choice	Not necessarily	Not necessarily
Simplification Method	Edge Collapse	Edge Collapse	Vertex Clustering
Error Metric	Distance to plane and edge	Quadric error metric (with attributes optionally)	Shape preserving quadric error metric

Table 4. Polygonal Mesh Simplification with OpenMesh

OpenMesh			
Algorithms	Quadric Module	Roundness Module	Normal Flipping Module
Input Mesh Type	Polygonal	Polygonal	Polygonal
	Manifold & Non-Manifold	Manifold & Non-Manifold	Manifold & Non-Manifold
Preserve Topology	Yes	Yes	-
Simplification Method	Half-edge Collapse	Half-edge collapse	Half-edge collapse
Error Metric	Quadric error metric	Roundness	Normal flipping

5.3. Comparison of the Algorithms with respect to their Performance and Execution Time

5.3.1. Metro Tool

As stated before, the main goal of mesh simplification is to produce a surface approximation while keeping the model's visual content as similar as possible to the original. In order to measure the geometric difference between two meshes (describing the same surface at different levels of detail) two most common error metrics from function approximation is utilized. The first metric L_∞ norm which is also known as Hausdorf distance is used to measure the maximum deviation between two models. On the other hand the second error metric L_2 norm provides a measure of average deviation.

There are a few tools [47, 57, 58, 59] developed for measuring the geometric approximation error between two meshes. Among them the most popular and commonly used one is the metro tool [47]. Metro is a freely available tool and

besides the numerical results it also visualizes the error by rendering the higher resolution mesh with a color for each vertex which is proportional to the error (Appendix-F). Metro begins with sampling the first input mesh by taking sampling points on the mesh surface and then calculates the distances between these sampling points and the second mesh. Then the roles change and the second mesh is sampled and distances are measured accordingly. Based on these distance calculations metro returns the mean and maximum errors between the two meshes.

In this work, metro (version 4.06) is selected for comparing the simplification results by considering its popularity and acceptance from the community. All the parameters of the tool is left as default.

5.3.2. Results for the Stanford Bunny Model

The first model chosen for comparison is the Stanford Bunny model which is a very common model for testing different kinds of geometry processing algorithms including polygonal mesh simplification (Figure-20). It is freely available in the Stanford 3D Scanning Repository [70] and composed of 34834 vertices, 69451 faces and 104288 edges.

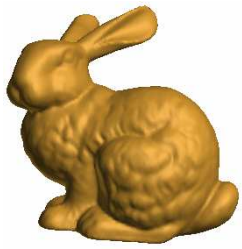

Figure 20. Stanford Bunny model

Obtained results are visualized by drawing the related graphics. In all graphics, the x-axes are representing the number of faces in the model. In addition, markers are also used to indicate the reduction rates. For instance, the left most markers show the 90% reduction rate whereas the right most markers are to used indicate the 10% reduction rate. The y-axes on the other hand show the maximum or mean geometric error. The error values are normalized by dividing each error value by the corresponding maximum error values so that the visualization of the results are better. The actual error values for Stanford Bunny can be found in Appendix-D and some sample figures from the simplification can be seen in Appendix-C.

According to the obtained results for the maximum geometric error vtkQuadricDecimation algorithm shows the best performance in all percentages (Figure-21, 22).

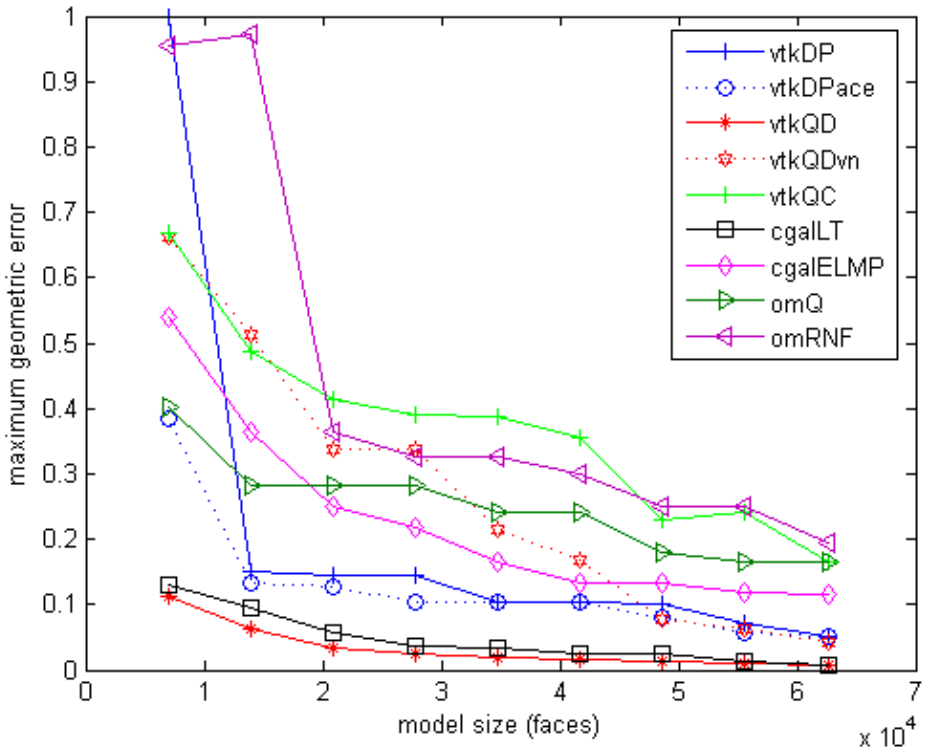

Figure 21. Comparison of all mesh simplification algorithms with respect to the normalized maximum geometric error for the Stanford Bunny model

The second best performing algorithm on the Stanford Bunny model is the CGAL Lindstrom-Turk strategy which is very close to vtkQuadricDecimation. Moreover, vtkDecimatePro with error accumulation takes the third place although vtkQuadricDecimation with vertex normals is slightly better when the reduction rates are 10% and 30%. However, vtkDecimatePro with error accumulation is better for all remaining percentages.

Once determining the first three best performing algorithms in terms of maximum geometric error, it is not very obvious to determine the other rankings. For example, in the remaining algorithms, for the 10%, 20% and 30% reduction rates vtkQuadricDecimation with vertex normals is better. However, its performance decreases very fast as the reduction rates increase. A similar behavior is observed for the vtkDecimatePro. For the 90% reduction rate vtkDecimatePro becomes the worst performing algorithm although it is very close to the third place algorithm (vtkDecimatePro with error accumulation) in all remaining percentages.

OpenMesh Quadric decimation module does not perform well enough even though it utilizes one of the best error measurement methods. On the contrary, CGAL Edge-length Midpoint performs beyond the expectation when considering its simple error metric. On average, the two algorithm OpenMesh Roundness with Normal Flipping and vtkQuadricClustering give the worst results for the maximum geometric error (Figure-21).

If each library is analyzed individually, for CGAL mesh simplification algorithms, Lindstrom-Turk strategy is better than Edge-length Midpoint strategy in all percentages (Figure-23). Similarly, for OpenMesh library (Figure-25) the Quadric decimating module and for VTK, vtkQuadricDecimation are the better ones in their own libraries.

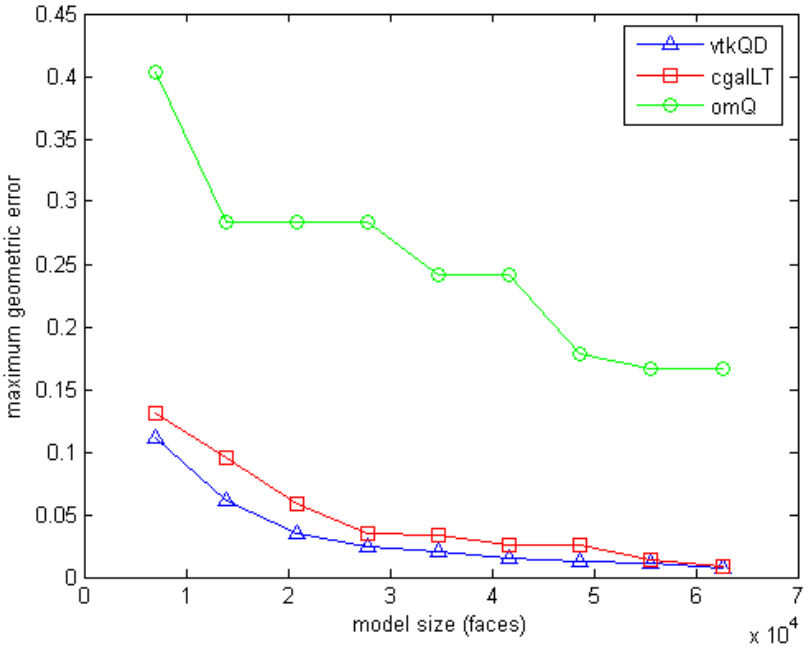

Figure 22. Comparison of the best mesh simplification algorithms from each library with respect to the normalized maximum geometric error for the Stanford Bunny model

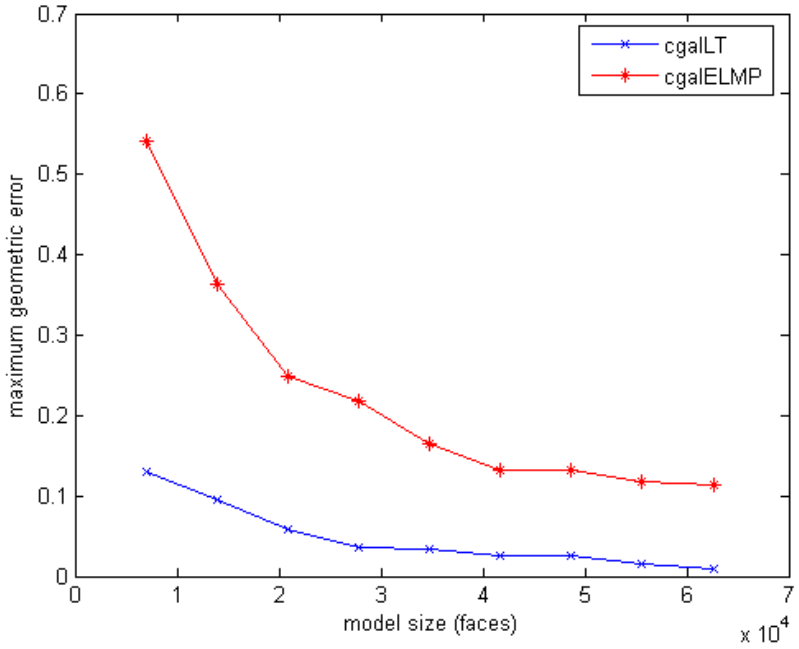

Figure 23. Comparison of CGAL mesh simplification algorithms with respect to the normalized maximum geometric error for the Stanford Bunny model

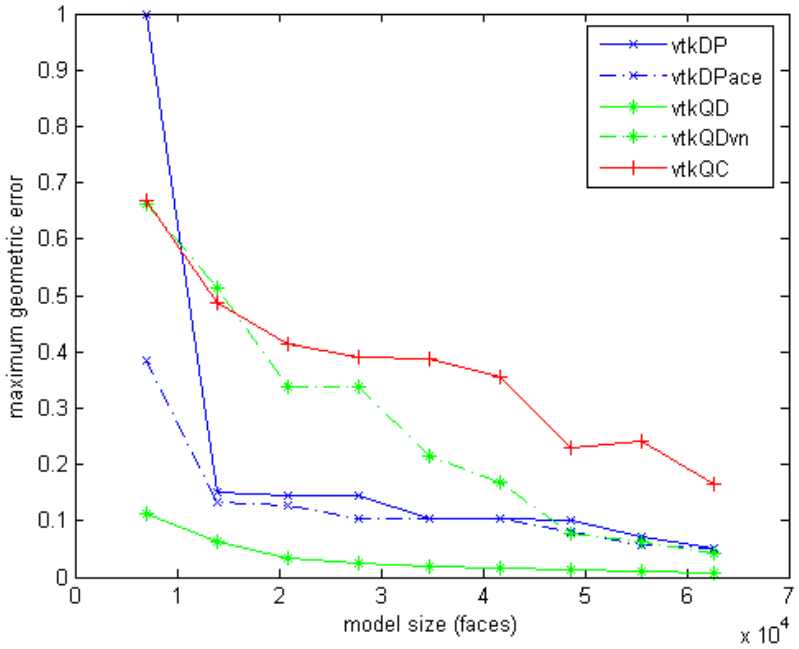

Figure 24. Comparison of VTK mesh simplification algorithms with respect to the normalized maximum geometric error for the Stanford Bunny model

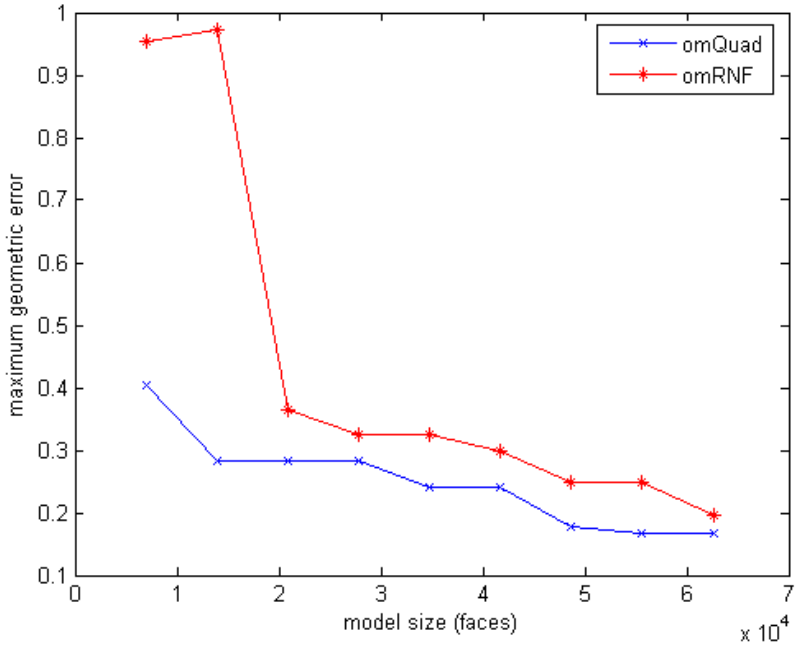

Figure 25. Comparison of OpenMesh mesh simplification algorithms with respect to the normalized maximum geometric error for the Stanford Bunny model

As the mean geometric errors are compared, more smooth graphs are obtained with respect to the maximum geometric error graphs. Within each library, the best place does not change and vtkQuadricDecimation, CGAL Lindstrom-Turk and OpenMesh Quadric module again take the first places in their own libraries (Figure-26, 27, 28). Surprisingly, vtkQuadricClustering gets higher in the ranking such that it takes the second place after the vtkQuadricDecimation in VTK library by considering the fact that other three algorithms in VTK perform better only in the 10% reduction rate. On the contrary, the other worst algorithm of the maximum geometric error, vtkQuadricDecimation with vertex normals does not make such an improvement and it gets the last place in the overall ranking.

When the over all ranking is considered for the mean geometric error, there is a change in the first place and CGAL Lindstrom-Turk algorithm takes the first place from the vtkQuadricDecimation (the best performer with respect to maximum geometric error) (Figure-29). Moreover, OpenMesh Quadric module takes a step forward and becomes the third best performer which is unexpectedly in the lower rankings for the maximum geometric error.

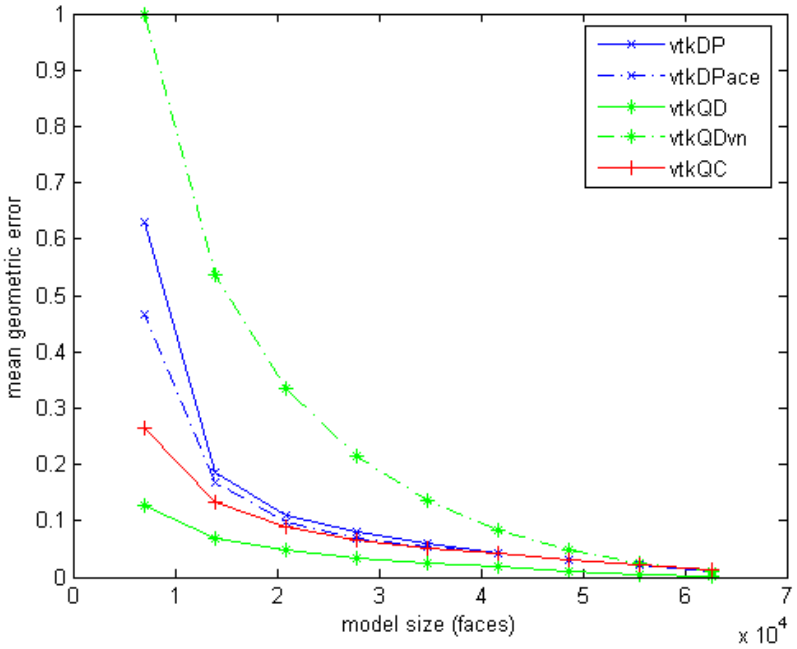

Figure 26. Comparison of VTK mesh simplification algorithms with respect to the normalized mean geometric error for the Stanford Bunny model

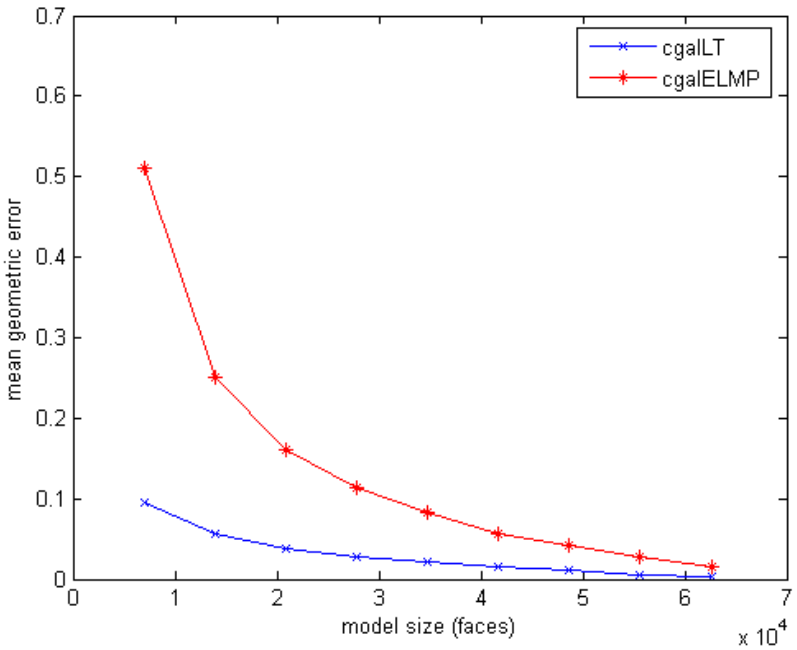

Figure 27. Comparison of CGAL mesh simplification algorithms with respect to the normalized mean geometric error for the Stanford Bunny model

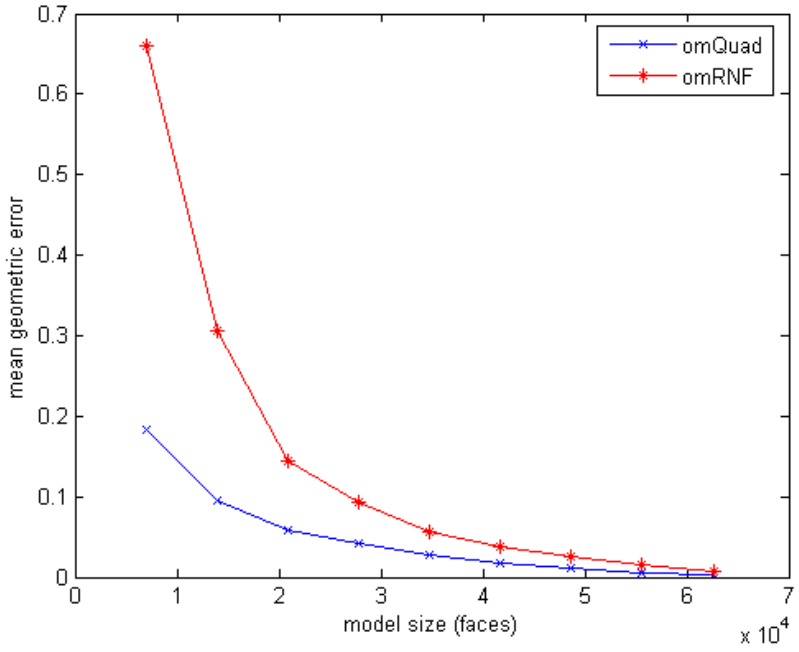

Figure 28. Comparison of OpenMesh mesh simplification algorithms with respect to the normalized mean geometric error for the Stanford Bunny model

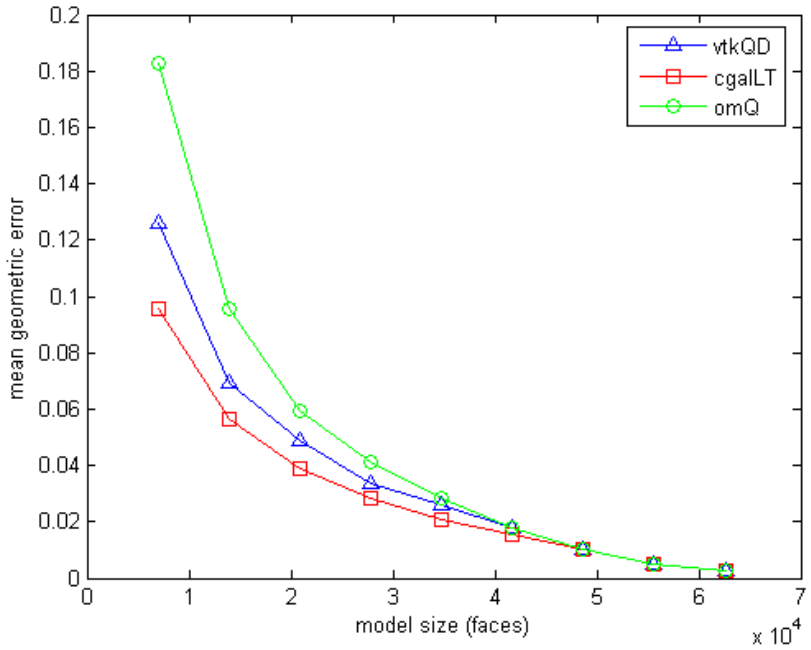

Figure 29. Comparison of the best mesh simplification algorithms from each library with respect to the normalized mean geometric error for the Stanford Bunny model

When the timing performance of the algorithms are considered, VTK library algorithms show an outstanding performance with respect to other libraries. CGAL library algorithms need the most amount of time, especially Lindstrom-Turk strategy is the worst algorithm in terms of timing requirements (Figure-30). This is mostly because of the error metric used, since a number of optimization problems are solved for determining the vertex position. However, It is expected that CGAL Edge-length Midpoint algorithm would be faster when considering its simple error measurement method. This may be caused by the inefficient implementation of the data structure in CGAL. OpenMesh algorithms are performing better than the CGALs' although they use the same data structure, but all the same, they can not come nearer to the VTK algorithms. In Figure-31, the timing performances of VTK algorithms can be seen more clearly and all of them are differing from each other with seconds. Observe that the timing performance of vtkQuadricClustering is inversely proportional with the reduction rate as it is expected.

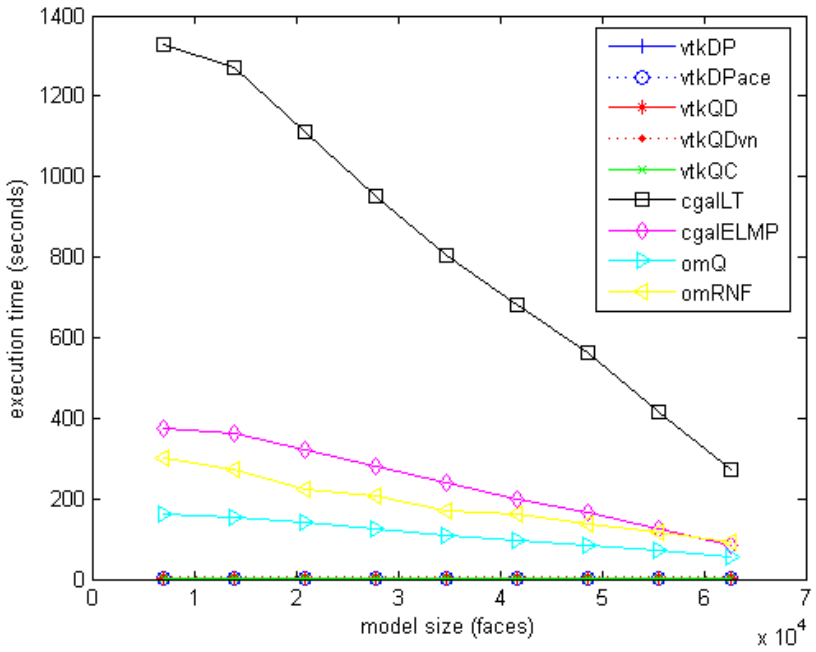

Figure 30. Comparison of timing performances of mesh simplification algorithms for the Stanford Bunny model

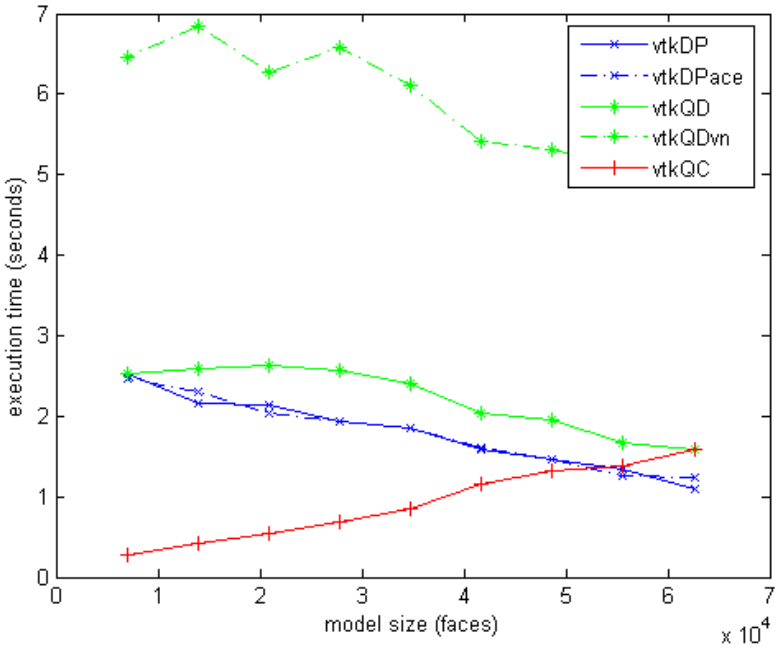

Figure 31. Comparison of timing performances of mesh simplification algorithms from VTK library for the Stanford Bunny model

5.3.3. Results for the Bumpy Torus Model

The second model selected for comparison is the Bumpy Torus model (Figure-32) [33] which is composed of 16815 vertices, 33630 faces and 50545 edges. Observe that, the high curvature nature of Bumpy Torus model makes it extraordinary.

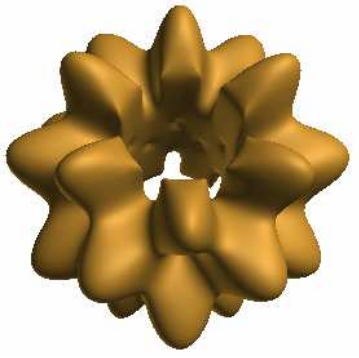

Figure 32. Bumpy Torus Model

As in the analysis of Stanford Bunny model, error values are also normalized in a similar way and the actual error values calculated for the Bumpy Torus model can be found in Appendix-E. Moreover, x, y axes of the graphics and the markers are used in the same way as explained in the previous section.

When maximum geometric error is considered for the Bumpy Torus model, vtkQuadricDecimation with vertex normals gives the best performance in all reduction rates except the 90% (Figure-33). At this reduction rate vtkQuadricDecimation is slightly better, but according to our consideration, this does not prevent vtkQuadricDecimation with vertex normals algorithm from taking the first place. This is probably because of the high curvature nature of the Bumpy Torus model where considering the vertex normals makes the difference. The second best performing algorithm is the vtkQuadricDecimation except the 70% reduction rate. At 70% reduction rate both CGAL Lindstrom-Turk and OpenMesh Quadric modules are better than the vtkQuadricDecimation (Figure 33-34). Moreover, It is not very easy to determine the third best performing algorithm since CGAL Lindstrom-Turk and OpenMesh Quadric module performs better with respect to each other in some percentages. For example, CGAL Lindstrom-Turk gives better results at 30%, 60%, 70%, 80% and 90% whereas OpenMesh Quadric module is better at 10%, 20% 40% and 50% (Figure-33). If we select the bottom two algorithms these can be the vtkDecimatePro and CGAL Edge-length Midpoint.

The best algorithms from CGAL and OpenMesh do not differ from the Bunny model (Figure-35, 36). Indeed, only the performance of vtkQuadricDecimation with vertex normals increased significantly and it becomes the best performer in terms of maximum geometric error both in the overall ranking and in VTK. In addition, the performance of OpenMesh Quadric module also increases with respect to the Bunny Model since it competes with the CGAL Lindstrom-Turk for the third place in the overall ranking. Remember that CGAL Lindstrom Turk takes the second place for the Stanford Bunny model in the ranking with respect to the maximum geometric error.

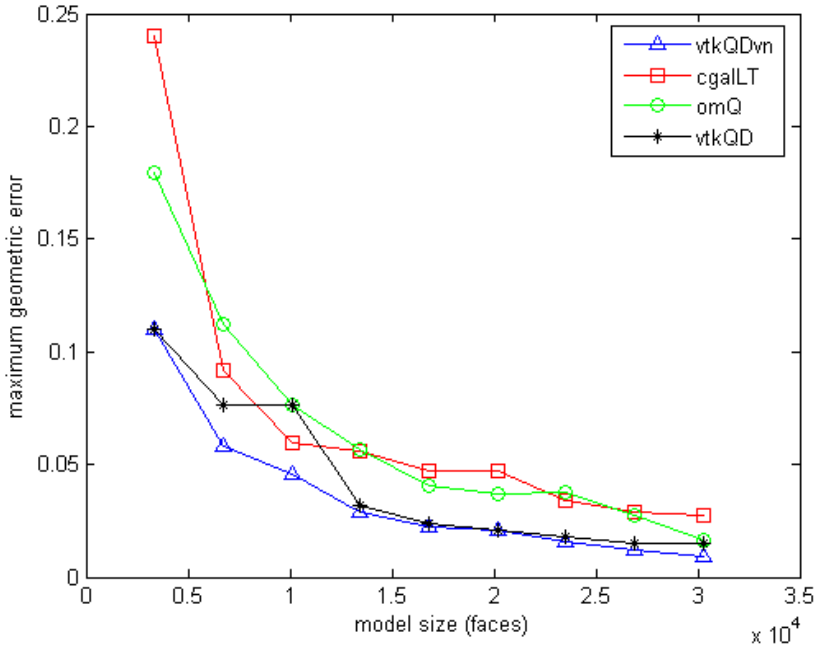

Figure 33. Comparison of best mesh simplification algorithms from each library and vtkQuadricDecimation with respect to the normalized maximum geometric error for the Bumpy Torus model

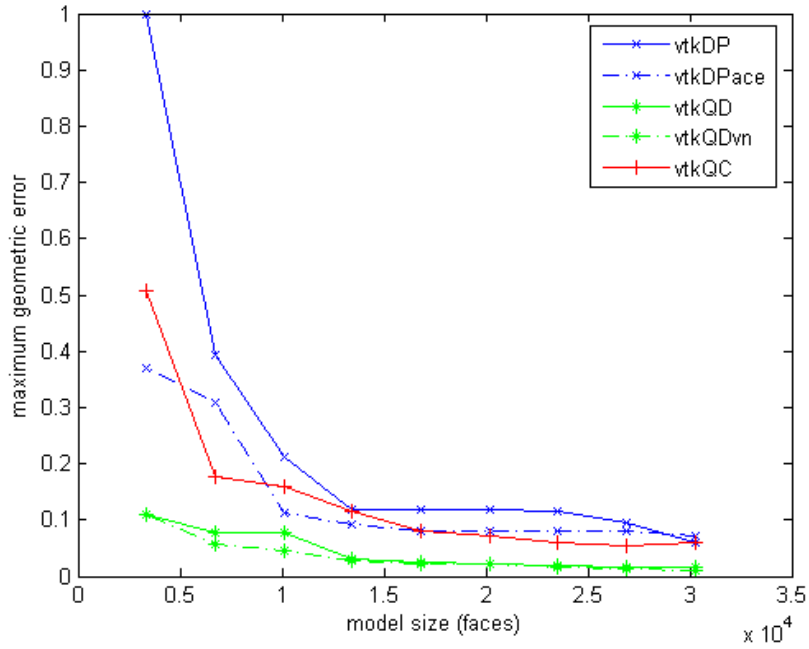

Figure 34. Comparison of VTK mesh simplification algorithms with respect to the normalized maximum geometric error for the Bumpy Torus model

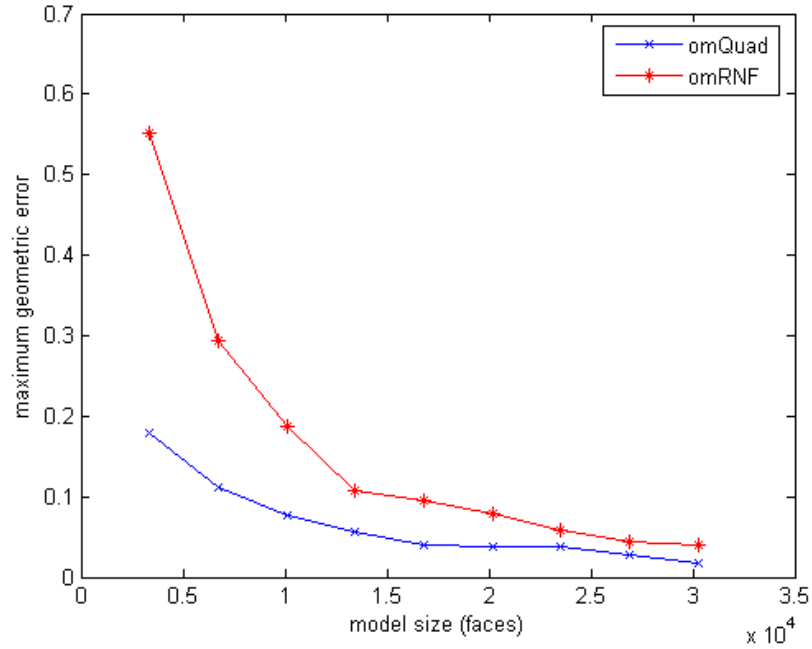

Figure 35. Comparison of OpenMesh mesh simplification algorithms with respect to the normalized maximum geometric error for the Bumpy Torus model

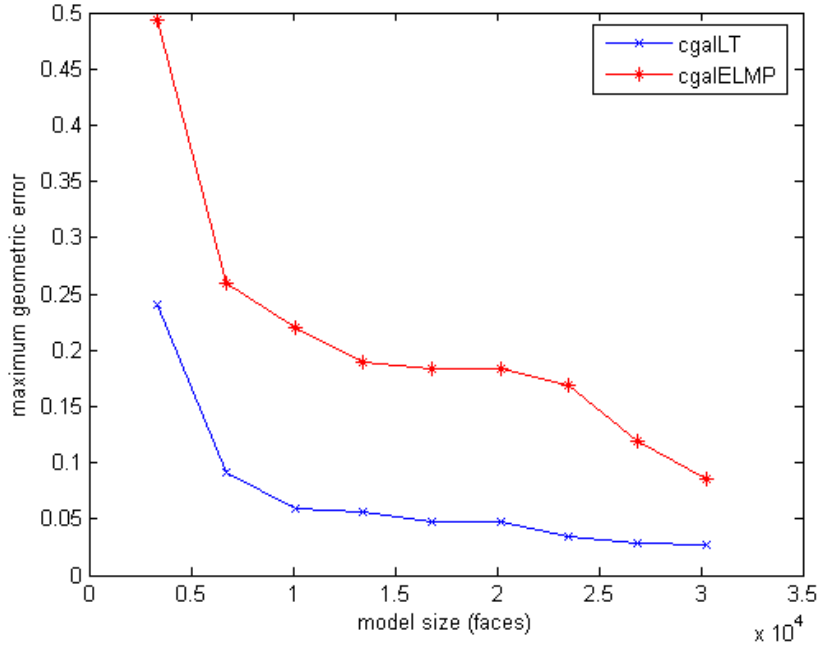

Figure 36. Comparison of CGAL mesh simplification algorithms with respect to the normalized maximum geometric error for the Bumpy Torus model

CGAL Lindstrom-Turk algorithm one more time becomes the best algorithm in terms of mean geometric error after the Bunny Model (Figure-37) except the 10% reduction rate. At this rate, vtkQuadricDecimation, vtkQuadricDecimation with vertex normals and OpenMesh Quadric module are better than Lindstrom-Turk. The second place also does not change and vtkQuadricDecimation gets the second rank. However, this time vtkQuadricDecimation with vertex normals is almost as good as the vtkQuadricDecimation such that their error values are almost equivalent and it gets the third place in the over all ranking.

The graphs for the comparison of the algorithms with respect to the mean geometric error within their own library are given in Figure-38, 39 and 40.

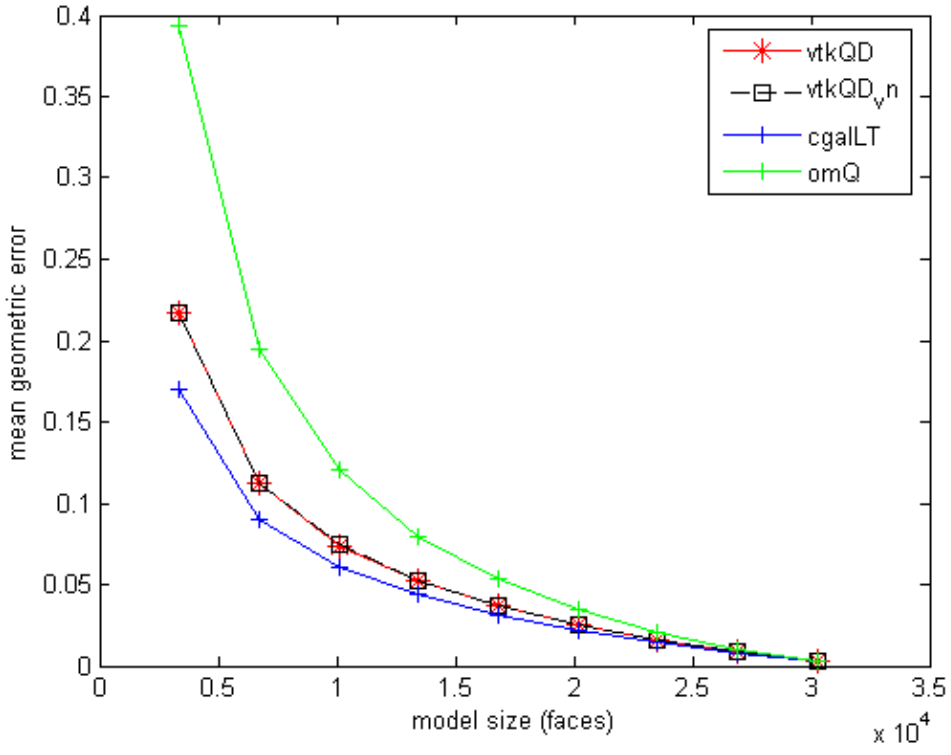

Figure 37. Comparison of best mesh simplification algorithms from each library and vtkQuadricDecimation with vertex normals with respect to the normalized mean geometric error for the Bumpy Torus model

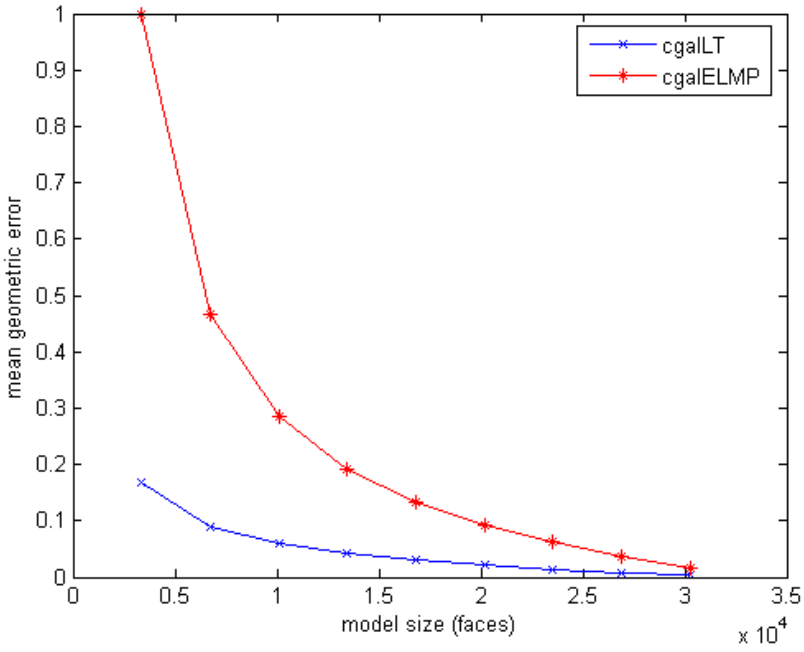

Figure 38. Comparison of CGAL mesh simplification algorithms with respect to the normalized mean geometric error for the Bumpy Torus model

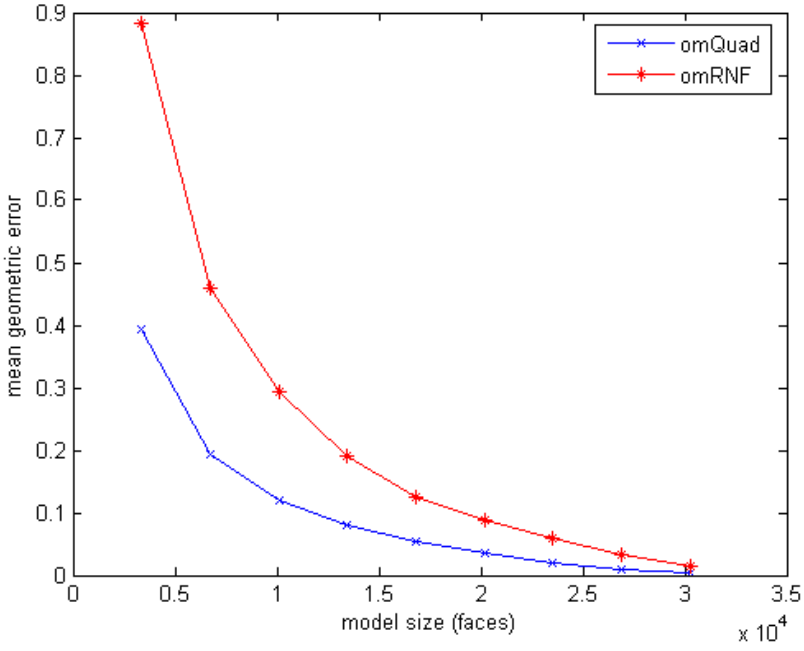

Figure 39. Comparison of OpenMesh mesh simplification algorithms with respect to the normalized mean geometric error for the Bumpy Torus model

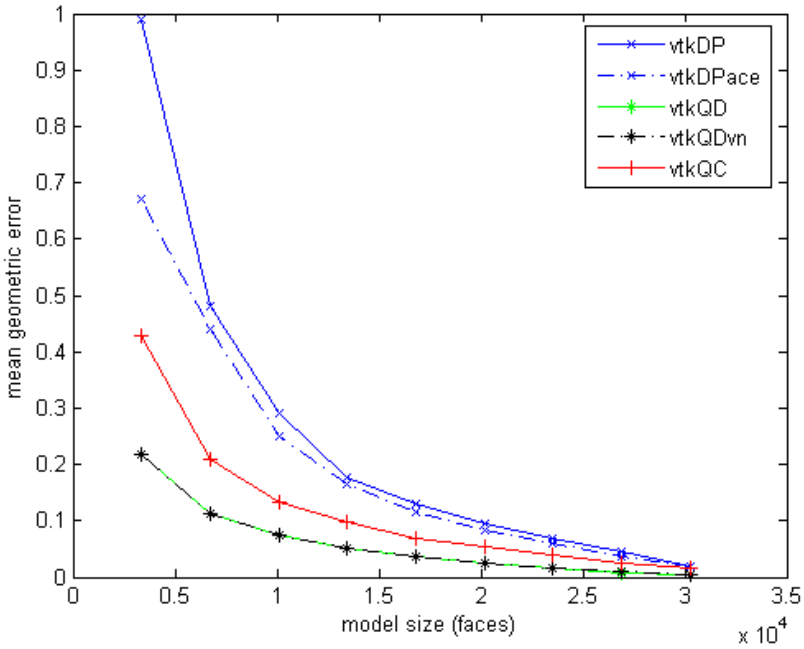

Figure 40. Comparison of VTK mesh simplification algorithms with respect to the normalized mean geometric error for the Bumpy Torus model

All of the algorithms show a similar timing performance when the results are compared with the those obtained for Stanford Bunny model. Again, VTK algorithms are outstanding and they are far more better than the CGAL and OpenMesh algorithms. Similarly, CGAL algorithms require the most amount of time for their execution (Figure-41). In order to analyze VTK algorithms separately, Figure-42 is provided. Observe that, some unexpected peaks are occurred in the given graphics. This may be because of the non-uniform behaviors of the used operating system, since the measured times are very small.

In Appendix-C some figures of the simplified Bumpy Torus model can be seen besides the Stanford Bunny model.

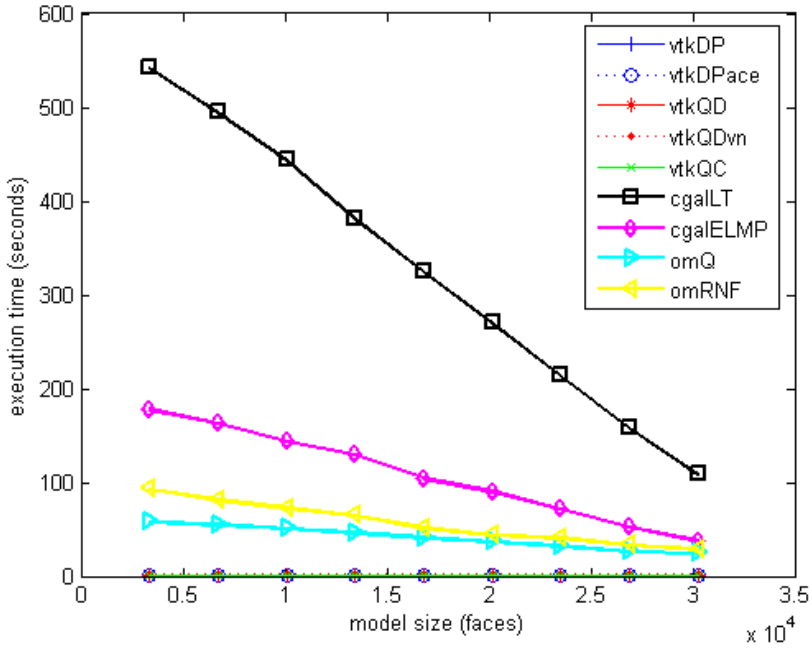

Figure 41. Comparison of timing performances of mesh simplification algorithms for the Bumpy Torus model

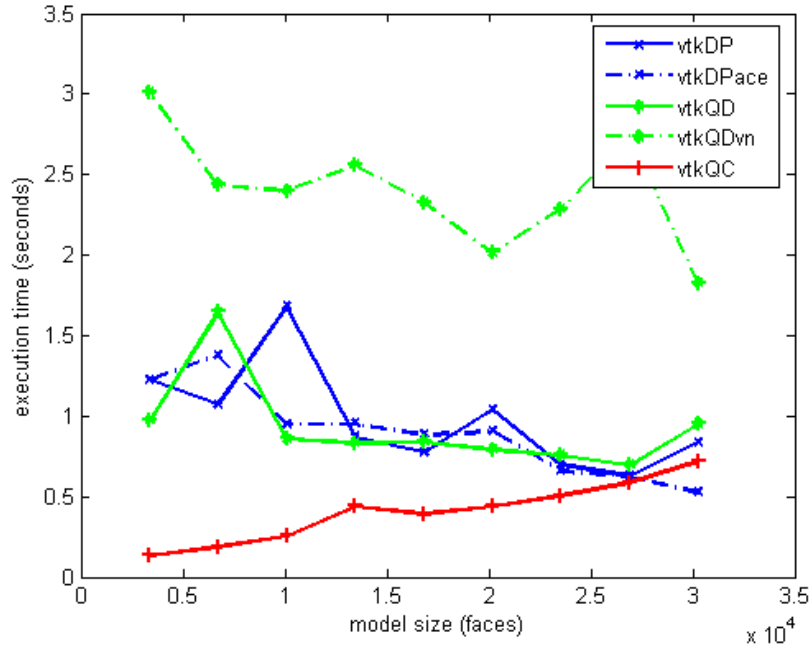

Figure 42. Comparison of timing performances of mesh simplification algorithms from VTK library for the Bumpy Torus model

5.3.4. Summary of the Results and Recommendations

We have compared nine different mesh simplification algorithms from three different mesh libraries over two different triangular mesh models. The evaluation is mainly based on the maximum and mean geometric errors between the simplified and original models. Moreover, the execution times of the algorithms are also considered. As described previously, all algorithms are executed for nine different reduction rates from 10% to 90%. Obtained results have already been discussed in the previous sections (sections 5.3.2, 5.3.3) but in order to summarize them in a tabular format the following tables may be helpful.

Table 5. Best Three Performing Algorithms for Stanford Bunny Model

Maximum Geometric Error	Mean Geometric Error
1. vtkQuadricDecimation	1. CGAL Lindstrom-Turk
2. CGAL Lindstrom-Turk	2. vtkQuadricDecimation
3. vtkDecimatePro with error accumulation	3. OpenMesh Quadric Module

Table 6. Best Three Performing Algorithms for Bumpy Torus Model

Maximum Geometric Error	Mean Geometric Error
1. vtkQuadricDecimation with vertex normals	1. CGAL Lindstrom-Turk
2. vtkQuadricDecimation	2. vtkQuadricDecimation
3. CGAL Lindstrom-Turk / OpenMesh Quadric Module	3. vtkQuadricDecimation with vertex normals

If a developer wants to implement a mesh simplification algorithm without a significant effort, we will recommend utilizing one of the freely available open source mesh libraries. However, our evaluation reveals that choosing the best library for mesh simplification mostly depends on the needs of the developer. In section 5.2, we describe the main properties of the proposed mesh libraries, where we try to answer the questions of the following type: what kinds of input meshes are supported and does the algorithm preserve the topology ? Later, based on the results of carried experiments, we see that the performance of the mesh simplification algorithms change with respect to the applied input mesh. Moreover, different algorithms are better for the same model when maximum and mean geometric errors are considered.

Regarding the issues listed in the above paragraph and obtained results so far, if a mesh simplification algorithm is required that is keeping the average deviation between the simplified and original meshes at minimum, our recommendation will be the CGAL Lindstrom-Turk algorithm. On the other hand, for minimizing the maximum geometric error a good choice will be the vtkQuadricDecimation. Furthermore, if the input mesh is highly curved, vtkQuadricDecimation with vertex normals may be applied as well.

Although CGAL Lindstrom-Turk algorithm is one of the best algorithms among the proposed ones, it needs the longest time for execution. Especially, for very large meshes, e.g. for a mesh with a million of faces, the execution time may reach to two to four hours. As a result, vtkQuadricDecimation algorithm can be offered to anyone due to its high quality results and speed.

CHAPTER 6

REGULAR REMESHING

Remeshing is the process of improving the quality of a mesh while approximating the original mesh acceptably. It is used in many geometric modeling algorithms such as shape editing, animation, morphing and numerical simulation [3]. Obviously, different applications have different quality criteria and requirements. Alliez and his co-workers [60] classified the remeshing techniques into five categories based on their goal: structured, compatible, high quality, feature and error-driven remeshing. In this work, structured remeshing also known as regular remeshing is studied for triangular meshes with disk topology.

Remember that in a triangular mesh a vertex is called regular if its valence (number of neighboring vertices) is equal to 6 for an interior vertex or 4 for a boundary vertex. In addition, we call a mesh regular if all vertices of the mesh are regular. Regular meshes offer certain advantages over irregular ones. First, complex data structures are not needed due to the fact that in a regular mesh the connectivity is implicit which means only the geometric information is loaded into the memory. This property of regular meshes improve the efficiency and facilitate the implementation of many algorithms such as mesh compression and morphing. Second, regular meshes has been shown to be useful for rendering (vertex caching), texture and other modulation mapping (normal, transparency mapping) and levels of detail generation [61, 62, 63].

Gu [61] developed a method for regular remeshing of arbitrary triangular meshes and named his method as geometry images. He begins by parameterizing the input mesh on to a 2D square which requires the input mesh to be topologically equivalent to a disk. If this is not the case, the input mesh can be cut and opened (Figure-43). Then, the geometry of the input mesh is captured as a simple n x n

array of [x y z] values by just taking sampling points on the parameterized domain and calculating the 3D equivalents of these points. The x, y, and z coordinates of points on the surface are then used to construct an image (geometry image) by assigning them to the red, green and blue components of corresponding pixels. By this way, a 3D mesh is represented with a 2D image (Figure-44).

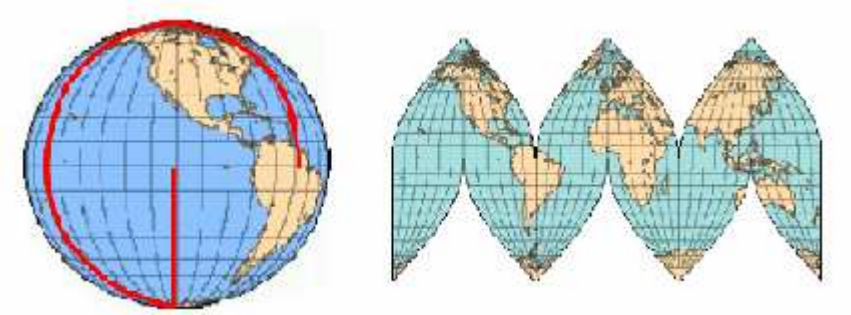

Figure 43. Obtaining a mesh with a disk topology by cutting
(http://research.microsoft.com/~hoppe/)

Once a geometry image is constructed, signal processing techniques then can be used on these images. For example, geometry images can be encoded using traditional image compression algorithms such as wavelet based coders. Moreover, they may be transmitted to the graphics pipeline in a compressed form just like texture images [61]. And a regular mesh can be reconstructed from a geometry image by interpreting each pixel as a vertex, and connecting vertices which correspond to neighboring pixels into a regular grid of triangles or quadrangles (Figure-45).

The cut algorithm described in the geometry images paper is a two step algorithm. In the first step, a topologically sufficient cut is found and an initial parameterization is created. In the second step, using the information from the first parameterization, the cut path is improved and a reparameterization is performed. The second step is iterated over and over until the cut path can not be improved

anymore. In the iterative part of the algorithm shape-preserving parameterization of Floater [64] is used whereas for the final parameterization geometric-stretch minimizing parameterization [65] is preferred. Floater parameterization is better for determining the extremal points (points with high curvature) but it introduces more distortion (angular and areal) than geometric-stretch minimizing parameterization. Once extremal points are identified the cut path is improved such that it passes over the extremal points so that the resultant distortion will be minimized.

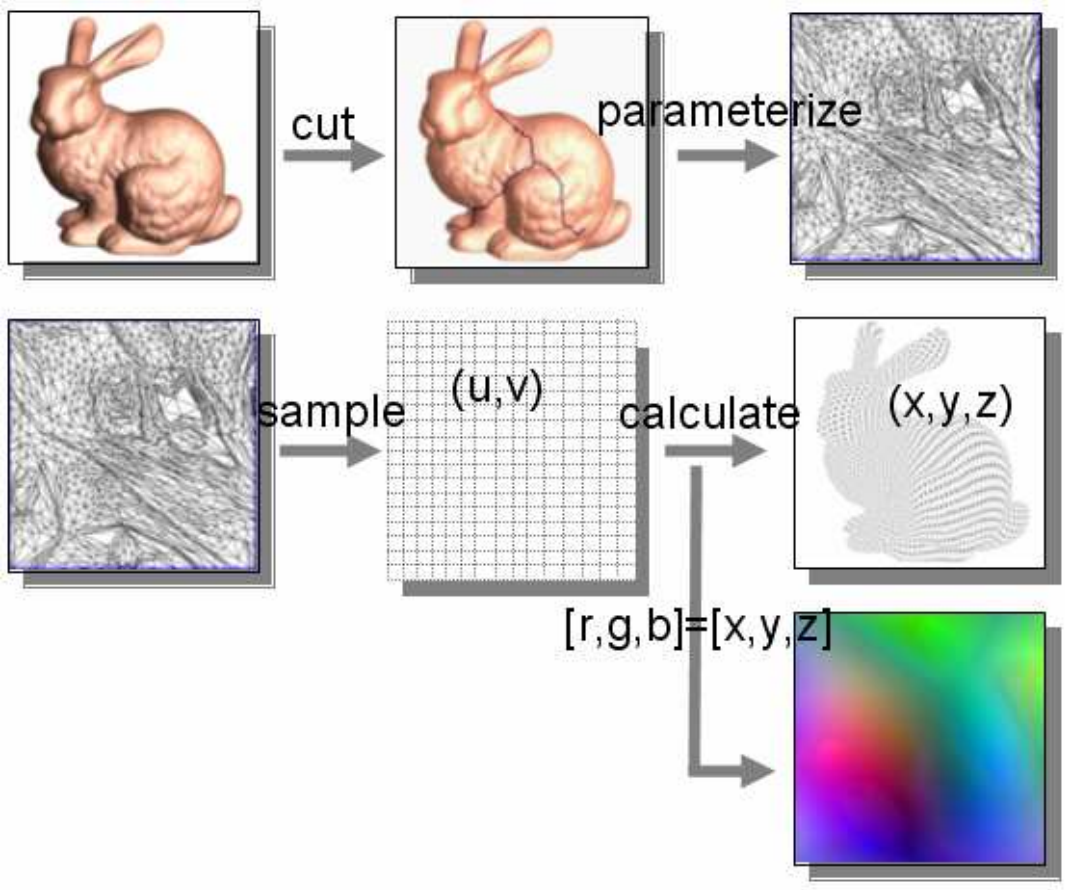

Figure 44. Constructing geometry images
(Figure is reproduced and modified from [61])

The quality of the reconstructed mesh from a geometry image mostly depends on the parameterization step. Parameterization is the process of finding a one-to-one (bijective) and piecewise linear mapping from a 3D image onto a 2D domain. The map is piecewise linear, associating each triangle of the original mesh with a triangle in the parameterization domain (Figure-46). The other important goal of mesh parameterization is to obtain bijective (invertible) maps where each point on the domain corresponds to exactly one point of the mesh. Both of the mesh parameterization algorithms [64, 65] used in the construction of geometry images are bijective and piecewise linear.

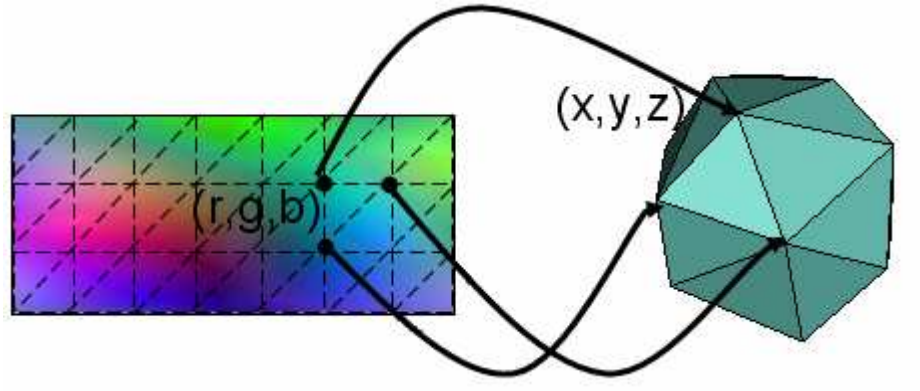

Figure 45. Reconstructing the regular mesh from a geometry image

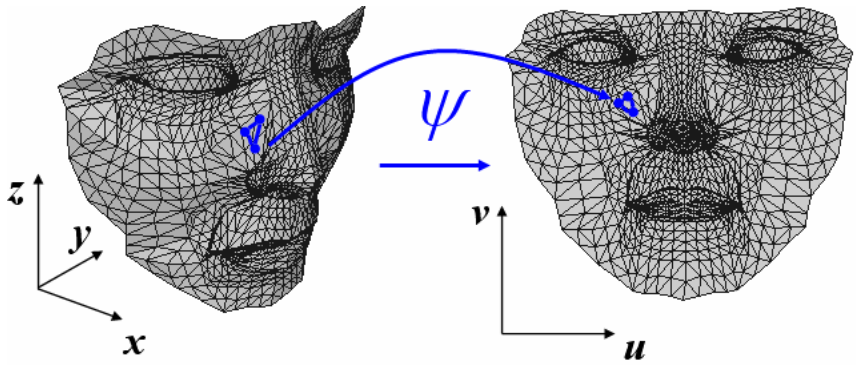

Figure 46. A piecewise and bijective parameterization

Generally, all mesh parameterization methods try to obtain an isometric mapping which preserves the lengths (both area and angles), i.e. the length of any arc on the original mesh is equal to the corresponding arc in the parameterization domain. However, almost all the time isometric mappings can not be obtained for an arbitrary mesh. Thus, mesh parameterization algorithms either concentrates on minimizing the angle (conformal mappings) or area (authalic mappings) or both (distance minimizing mappings) of them [66]. Shape preserving parameterization of Floater mainly tries to minimize the angular distortion whereas geometric stretch parameterization minimizes distances.

6.1. Implementation

The motivation behind this work is the regular remeshing of human face models which are generally genus-0 and topologically equivalent to disks. Thus, the complex cutting algorithm in the geometry images is omitted. Moreover, regular models are directly constructed after the parameterization step without generating the geometry images and finally it is allowed to sample the geometry in any resolution in both directions. The steps for the implemented algorithm is given in below:

```
Parameterize the input mesh onto a 2D square

Sample the parameterized mesh (nxm)

For each sampled point

     Determine in which triangle the current
     sampled point is

     Calculate the barycentric coordinates in
     the parameter domain

     Use the barycentric coordinates to calculate
     the corresponding point (x, y, z) in 3D

End For
```

```
Construct the regular mesh by triangulating
or quadrangulating the calculated 3D points
```

For the parameterization step of the algorithm, mesh parameterization package of CGAL library [25] is used. The package has 5 different parameterization methods and among them the Floater-Mean Value Coordinates Parameterization [67] is chosen since it is more promising than the others. Mean value coordinates parameterization is very similar to [64]. In fact, Floater approximates his shape-preserving parameterization by using mean value coordinates at some step of the algorithm which gives very similar results while making the computational part of the shape-preserving algorithm more easier [67].

Once the parameterization is obtained (for each triangle/vertex in the 3D mesh a corresponding triangle/vertex is found in 2D), the next step is to sample the geometry in the parameterization domain. Since the parameterization is piecewise linear and bijective, we can compute the corresponding 3D points to the sampled points by using a linear interpolation. For the interpolation, barycentric coordinates in triangles are used by which a point's position can be determined uniquely with respect to a triangle. Moreover, barycentric coordinates are also used for triangle inclusion test that is used to determine if a given point is inside a triangle or not.

Consider three points P_1, P_2, and P_3 in a plane. If w_1, w_2, and w_3 are scalars such that

$$w_1 + w_2 + w_3 = 1 \tag{25}$$

then the point

$$P = w_1 P_1 + w_2 P_2 + w_3 P_3 \tag{26}$$

is a point on the plane of the triangle $\triangle P_1P_2P_3$ and we say that $[w_1, w_2, w_3]$ are the barycentric coordinates of P with respect to P_1, P_2, and P_3. Furthermore, the position of the point P can be determined as follows (Figure 45):

- The point P is inside the triangle $\triangle P_1P_2P_3$ if $0 \le w_1, w_2, w_3 \le 1$.
- The point P is outside the triangle $\triangle P_1P_2P_3$ if any of w's is less than 0 or greater than 1.
- The point P is on the edge of the triangle if one of the w's is equal to 0.
- The point P is on the vertex of the triangle if two of the w's is equal to 0 and the remaining one is equal to 1.

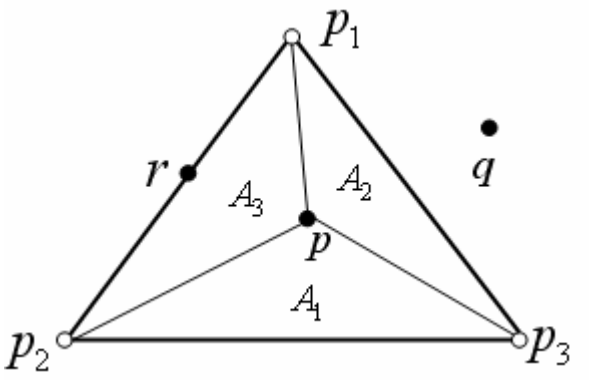

Figure 47. Barycentric coordinates: p is inside the triangle ($0 \le w_1, w_2, w_3 \le 1$), q is outside the triangle ($w_2 < 0$) and r is on the edge $|P_1P_2|$ ($w_3 = 0$)

Equations (25) and (26) can be used to construct a linear system:

$$\begin{pmatrix} P_1 & P_2 & P_3 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} P \\ 1 \end{pmatrix} \tag{27}$$

and using the Cramer's rule the linear system can be solved such that

$$w_1 = A_1/A, \quad w_2 = A_2/A, \quad w_3 = A_3/A \tag{28}$$

where A, A_1, A_2 and A_3 is the signed area of the triangles $\triangle P_1P_2P_3$, $\triangle PP_2P_3$, $\triangle P_1PP_3$ and $\triangle P_1P_2P$ respectively (Figure 47). The signed areas are defined as follows:

$$A = \begin{vmatrix} P_1 & P_2 & P_3 \\ 1 & 1 & 1 \end{vmatrix}, \; A_1 = \begin{vmatrix} P & P_2 & P_3 \\ 1 & 1 & 1 \end{vmatrix}, \; A_2 = \begin{vmatrix} P_1 & P & P_3 \\ 1 & 1 & 1 \end{vmatrix}, \; A_3 = \begin{vmatrix} P_1 & P_2 & P \\ 1 & 1 & 1 \end{vmatrix} \tag{29}$$

Once the barycentric coordinates of a sampled point are calculated, i.e., w_1, w_2, and w_3 are found, the 3D equivalent of the point is calculated by using these coordinates and equation (26). After all points in 3D are found, it is very easy to triangulate or quadrangulate the points since they have a regular structure.

6.2. Results

The implemented regular remeshing algorithm is tested with four different face models with different resolutions. The first model, Nefertiti (Figure-48), is also the simplest one, composed of 299 vertices, 562 faces and 860 edges. Nefertiti model comes with the CGAL library [25] for testing algorithms on it.

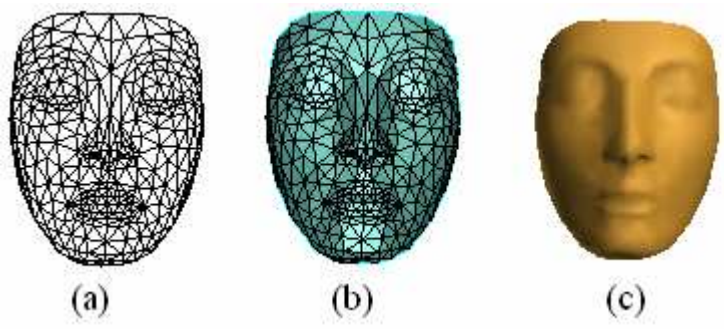

Figure 48. Nefertiti model: wireframe (a), shaded-1 (b) and shaded-2 (c)

As it is described in section 6.1, we begin by parameterizing the input mesh onto a 2D unit square. The parameterization of the Nefertiti model onto a 2D unit square is given in Figure-49. Observe that, in some of regions of the parameterization domain the density of the vertices are relatively high. These regions correspond to the high curvature regions of the original model. The model is then sampled in the parameterization domain and corresponding 3D points are calculated by a linear interpolation (Figure-50). Remember that barycentric coordinates are used for the linear interpolation and also for the triangle inclusion test. The sampling resolution can be selected as desired but it gives better results when the two dimensions are selected equally. After getting the regular point cloud, the connectivity of these points can be constructed in three different ways. In the first way, faces of the regular mesh can be generated as quadrangles whereas the other two methods result in triangular meshes.

In Figure-51, a sample quadrangulation and one of the two possible triangulations are shown for (33x33) resolution and a higher resolution (65x65) is shown in Figure-52. As the resolution of the sampling increases, more complex but regular mesh models can be obtained.

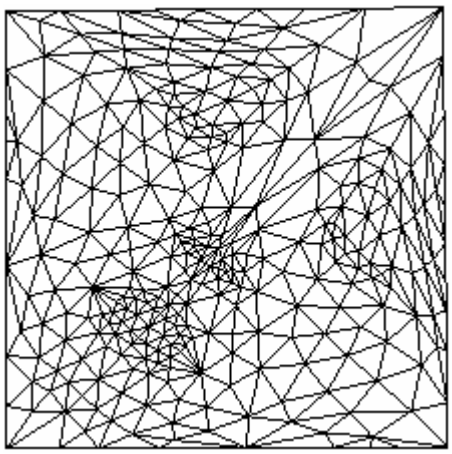

Figure 49. Nefertiti model parameterized onto a 2D unit square

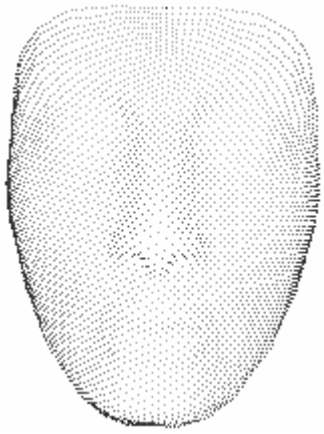

Figure 50. Point cloud, obtained by sampling (65x65) the parameterization domain

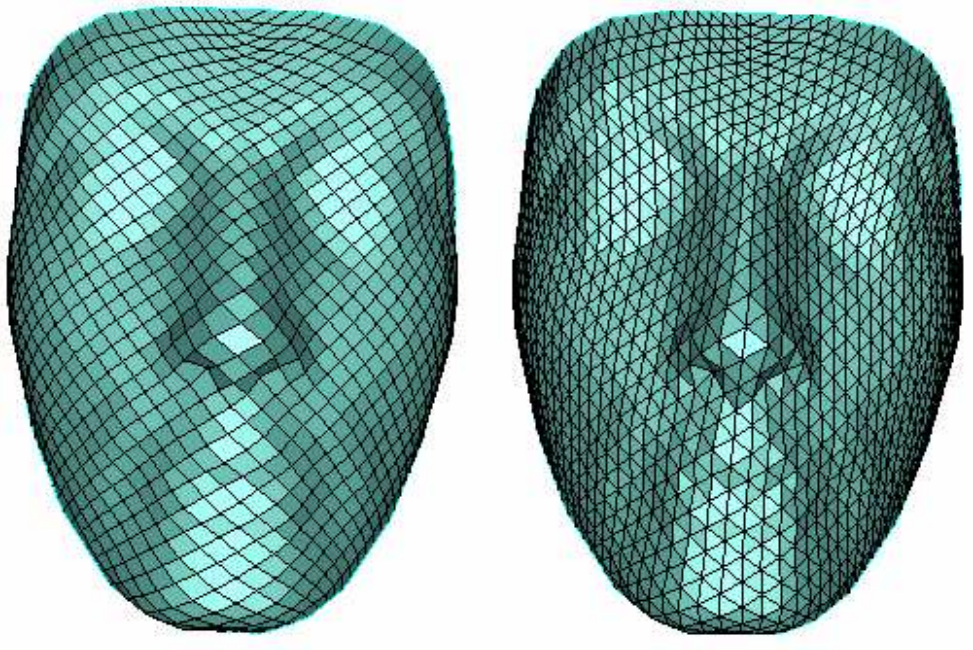

Figure-51. A sample quadrangulation and triangulation of the Nefertiti model (Resolution: 33x33)

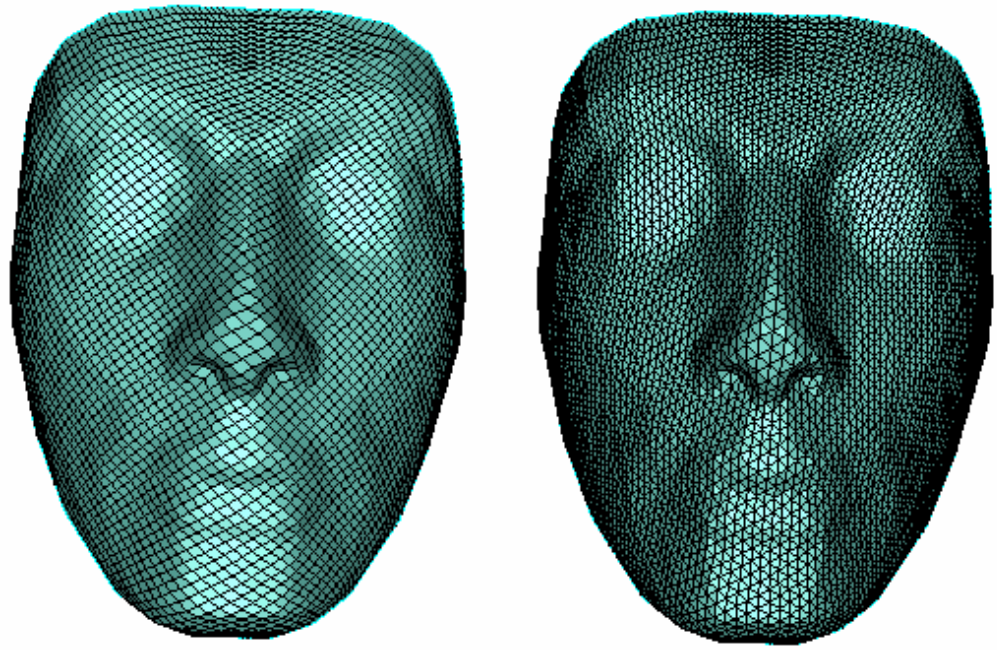

Figure 52. A sample quadrangulation and triangulation of the Nefertiti model (Resolution: 65x65)

The second model that regular remeshing applied is the Egea model obtained from the Aim@Shape Repository [33] (Figure-53). This model is nearly ten times complex than the Nefertiti model and composed of 3042 vertices, 5898 faces and 8939 edges. The same procedures with the Nefertiti model are applied and the obtained results are shown in Figure-54 (parameterization domain), 55 (point cloud obtained) and 56 (65x65 triangulation and 129x129 quadrangulation).

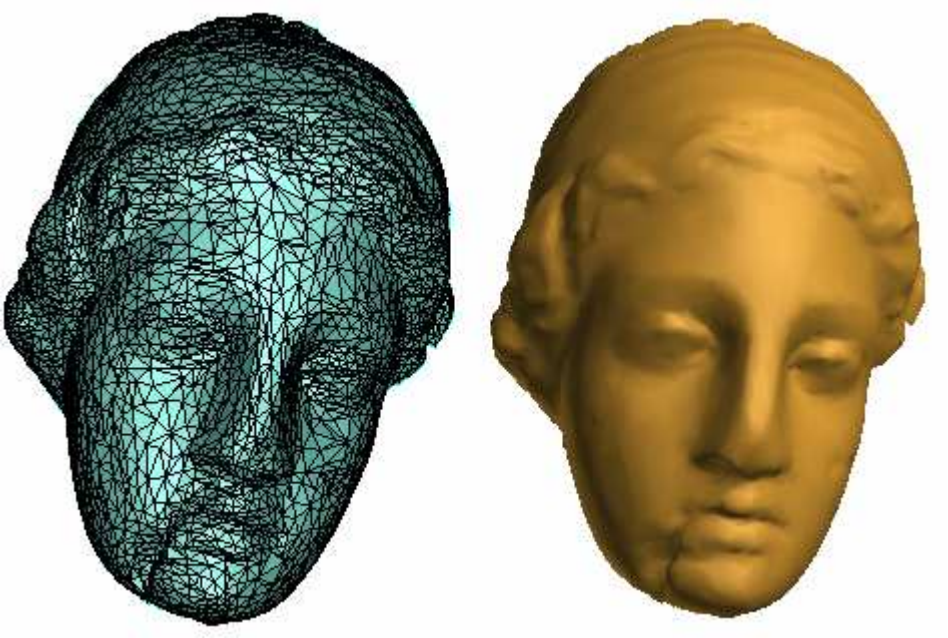

Figure 53. Egea model, shaded with two different shaders

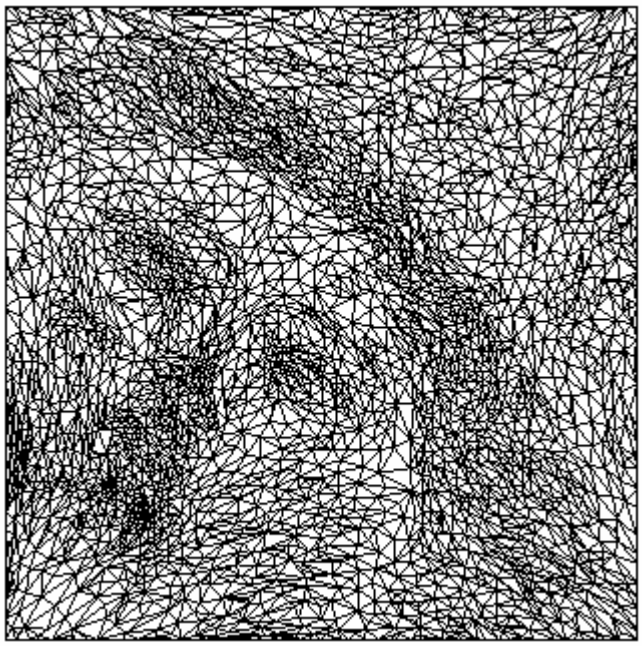

Figure 54. Parameterization of the Egea model onto a 2D unit square

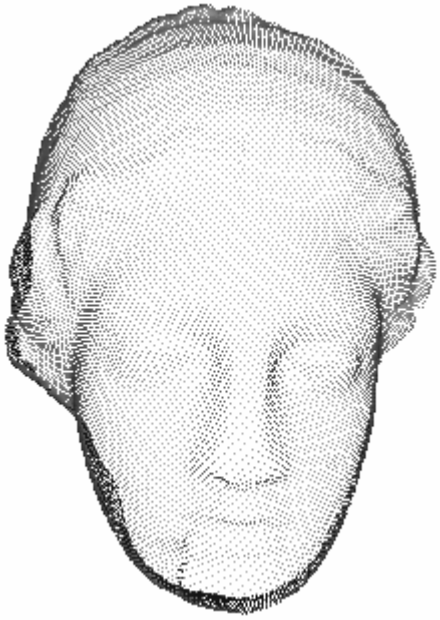

Figure 55. Point cloud, obtained by sampling (129x129) the parameterization domain of the Egea model

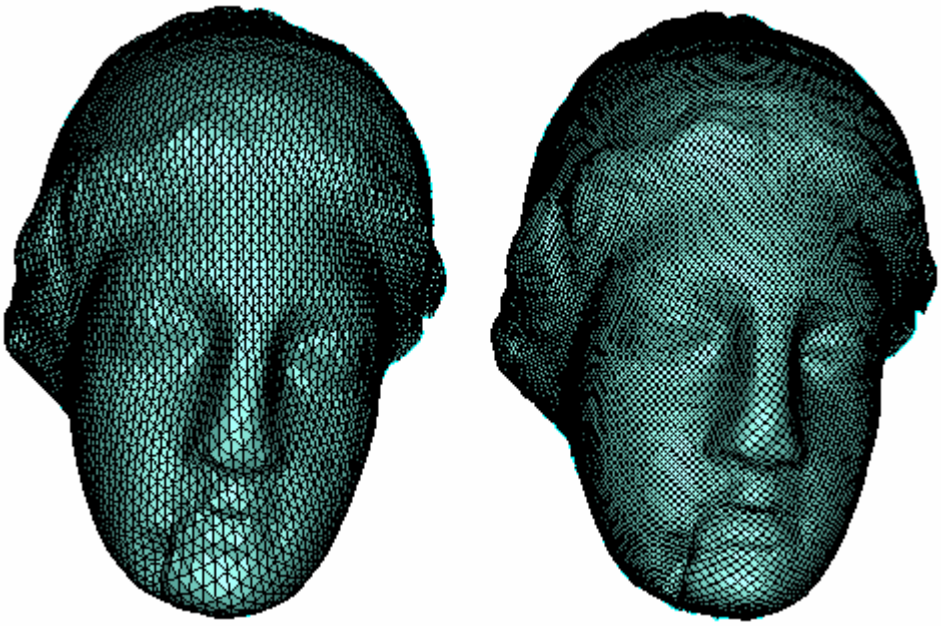

Figure 56. A sample triangulation (65x65) and quadrangulation (129x129) of the Egea model

The third model is a male face got from [68] (Figure-57). This model has 5321 vertices, 10457 faces and 15777 edges. After applying the previously described steps, similar results are obtained (Figure-58, 59). Note that for this model the nose of the regularly remeshed models (Figure-59) are not good as the other models. This is because of the high curvature in the nose. A highly curved region of the input mesh results in a relatively denser region in the parameterization domain, which hinders the adequate sampling to capture the geometry. However, this distortion may be negligible in most of the cases. In fact, if the obtained regular models are rendered, the distortion may not be even realized (Figure-60). Nevertheless, if the generated distortion is not acceptable, then this problem can be solved in two ways. First, the resolution of the sampling step can be increased. By this way, the distortion of the highly curved parts will be reduced but which also increases the complexity of the obtained mesh. Second, the parameterization method of the applied algorithm can be changed with a better one [61].

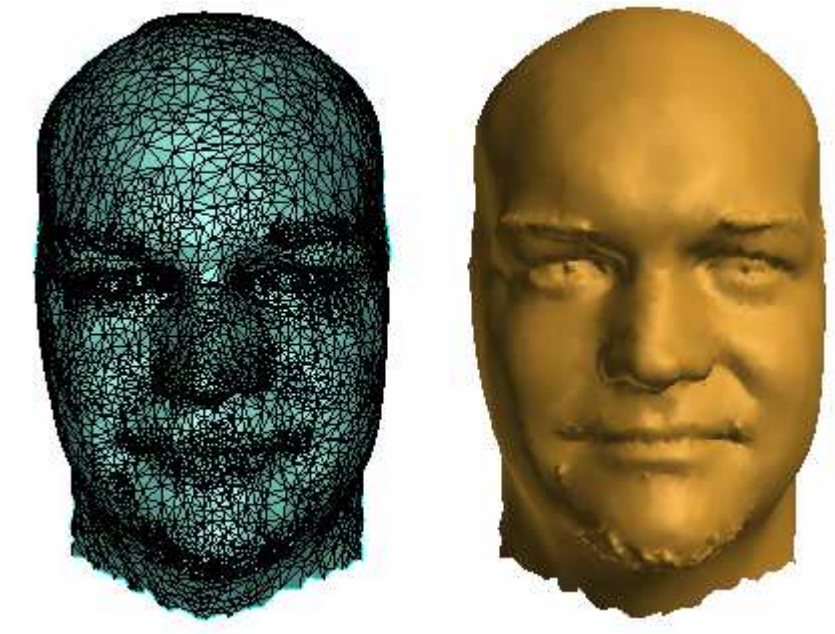

Figure 57. Male face model, shaded with two different shaders

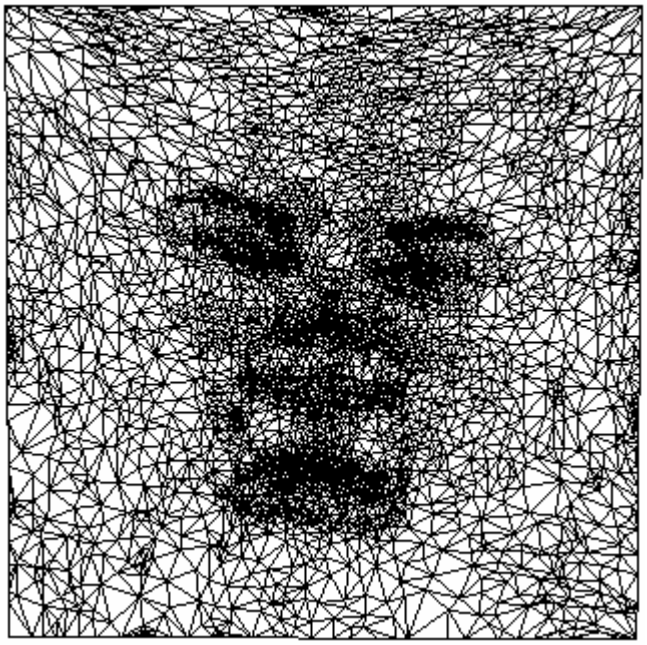

Figure 58. Parameterization of the Male face model onto a 2D unit square

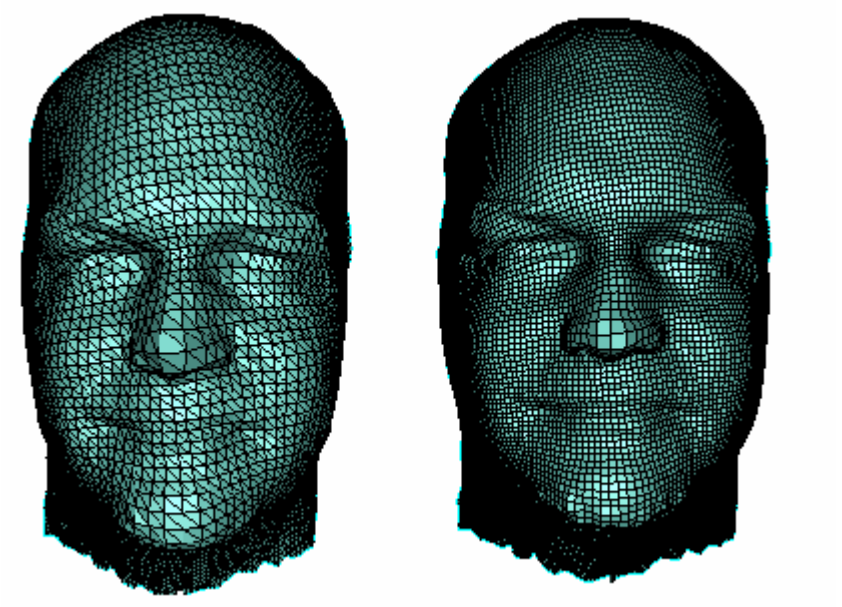

Figure 59. A sample triangulation (65x65) and quadrangulation (129x129) of the Male face model

Figure 60. Shading the regular Male face model (65x65)

The last model used for the regular remeshing is again a male face and got from [33] (Figure-61). This model is called as face-YH and it is the most complex model that is worked with such that it has 10199 vertices, 20000 faces and 30198 edges. The obtained results are illustrated in Figure-62 and 63.

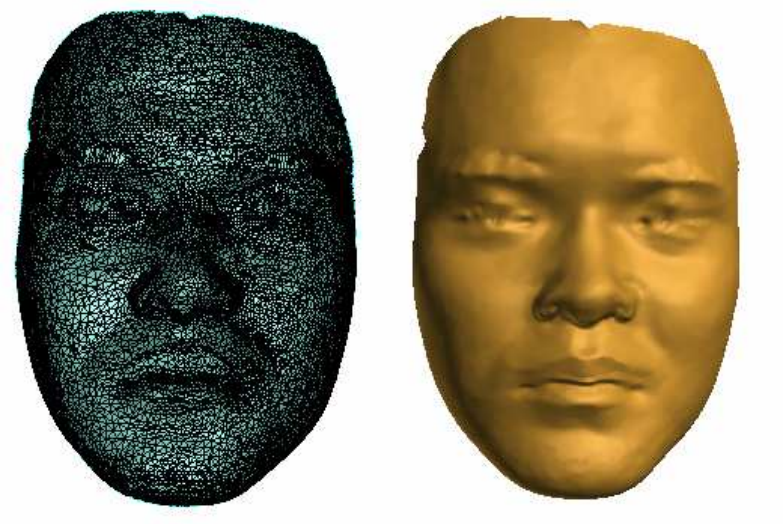

Figure 61. face-YH model, shaded with two different shaders

Figure 62. Parameterization of the face-YH model onto a 2D unit square

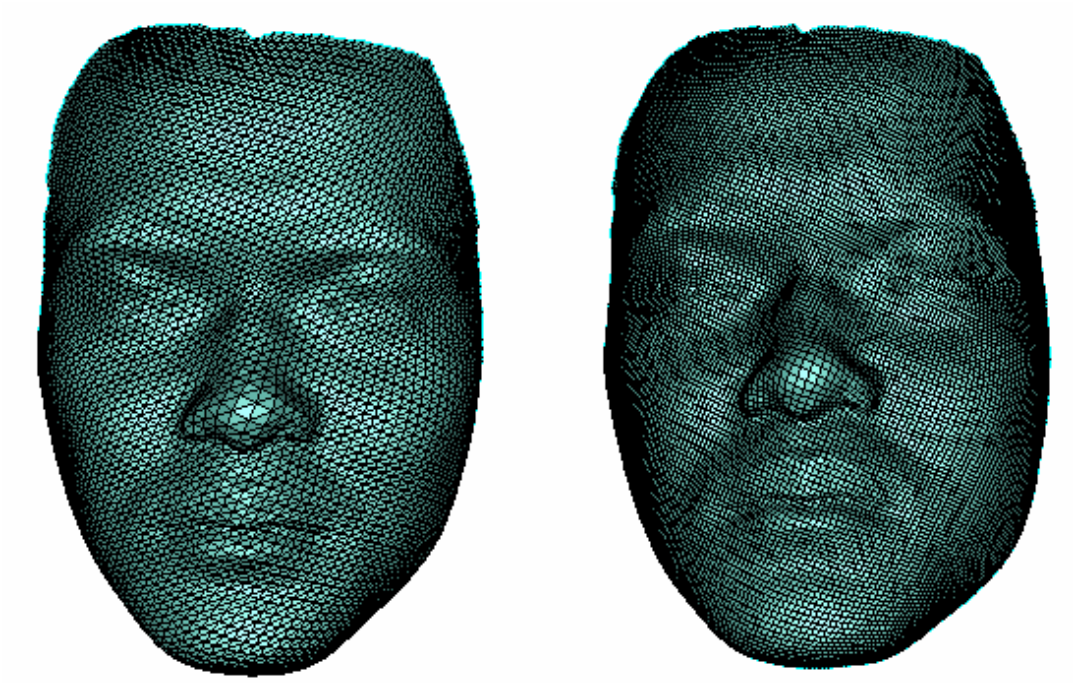

Figure 63. A sample triangulation (65x65) and quadrangulation (129x129) of the face-YH model

Observe that if the input mesh is sampled such that the produced regular mesh has fewer vertices, then the regular remeshing method can also be thought as a simplification algorithm. For instance, for the face-YH model one of the sampling rates is chosen as 65x65 (Figure-61). This resolution rate produces a regular mesh with 4225 vertices, 8192 faces and 12416 edges which exactly corresponds to a 59% reduced of the original mesh. Furthermore, regular remeshing algorithm can be compared with the other mesh simplification algorithms that are discussed in Chapters IV and V. Remember that one of the most efficient algorithms is the vtkQuadricDecimation algorithm both in terms of maximum and geometric errors and timing requirements. When vtkQuadricDecimation and regular remeshing algorithms are compared by using the metro tool, we see that vtkQuadricDecimation performs better (Table-7).

Table 7. Comparison of Regular Remeshing Algorithm with the vtkQuadricDecimation at a 59% Reduction Rate

Algorithm	Max. Geometric Error	Mean Geometric Error
Regular Remeshing	2.647200	0.056288
vtkQuadricDecimation	0.185284	0.011697

In addition, if the visual outputs of the metro tool is analyzed, it can be seen that regular remeshing algorithm approximates the original model much more better in the smooth regions of the original model (Figure-62). The distortions on the relatively high curvature regions increases the geometric errors. On the other hand, vtkQuadricDecimation algorithm does not decimate the highly curved regions and distribute the error along the whole mesh (Figure-63) .

Figure 64. Error distribution for the face-YH model when simplified with regular remeshing algorithm

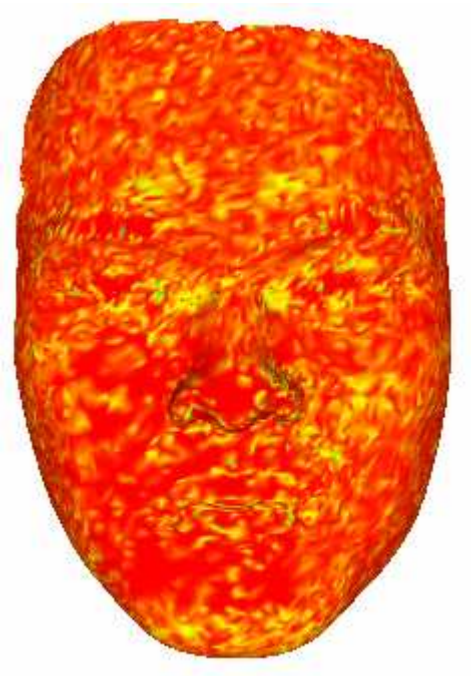

Figure 65. Error distribution for the face-YH model when simplified with vtkQuadricDecimation

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this thesis work, two important topics of mesh processing are studied: polygonal mesh simplification and regular remeshing of irregular polygonal meshes. The first topic, polygonal mesh simplification is a very active subject and there are a lot of different algorithms proposed, which makes the classification of these algorithms very challenging. Among the possible classifications we adopted one of them and by utilizing the other works, a classification of ours is offered which may be thought as the combination of the previous works.

Due to popularity of the subject, polygonal mesh simplification algorithms are implemented in a number of different mesh libraries. In order to provide a guide for commonly used mesh libraries in terms of mesh simplification, we compared the simplification algorithms from these libraries. Based on this evaluation, we concluded that there is no certain best algorithm since different algorithms may operate well on different input meshes. However, some of the algorithms are generally performing better than the others in terms of geometric error and execution time. Among them, vtkQuadricDecimation algorithm can be a good choice when considering its speed and results.

In addition to this work, similar works can be repeated for the commercial mesh simplification packages and as well as for some other free programs. By this way, the performance of the mesh libraries by means of mesh simplification can be determined relatively. Moreover, since these algorithms are open source, they may be modified to get better algorithms.

The second topic discussed in this work is the regular remeshing of the irregular polygonal meshes. The motivation behind this conversion is the advantages of the

regular meshes for both processing and rendering. There exists only one method for such a conversion and in this work, an implementation based on this algorithm is done and it is tested with the human face models. Moreover, regular remeshing algorithm is also considered as a mesh simplification algorithm for special cases and a comparison of the algorithm is performed with the dedicated mesh simplification algorithms.

Our implementation of regular remeshing can only be capable of handling input meshes that are topologically equivalent to disks. This work can be expanded to handle an arbitrary irregular polygonal mesh. Moreover, since the quality of the obtained regular meshes mostly depend on the parameterization step of the algorithm, better parameterization methods can be utilized.

REFERENCES

[1] O'Rourke, Joseph. Computational Geometry in C, Second Ed., Cambridge University Press, 1998, pg. 12-13.

[2] Hoppe Hughes. Mesh Optimization, In Proceedings of ACM SIGGRAPH 93, pg. 19-26, 1993.

[3] Botsch Mario and Pauly Mark. Geometric Modeling Based on Polygonal Meshes, ACM SIGGRAPH 2006 Courses.

[4] William S. Massey. A Basic Course in Algebraic Topology, Springer-Verlag, 1991

[5] Hoffmann M. Christoph. Geometric and Solid Modeling, Morgan Kaufmann Pub. 1989, pg. 49-53.

[6] Luebke P. David. A Developer's Survey of Polygonal Simplification Algorithms, IEEE Computer Graphics and Applications, 2001.

[7] Weisstein, Eric W. "Genus." From MathWorld--A Wolfram Web Resource. Retrieved August 2008 from http://mathworld.wolfram.com/Genus.html

[8] Hoffmann M. Christoph. Geometric and Solid Modeling, Morgan Kaufmann Pub. 1989, pg. 39-42.

[9] 3D Object Converter. Retrieved August 2008 from http://web.axelero.hu/karpo/

[10] Wavefront's Object File Format (OBJ File Format). Retrieved August 2008 from http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/

[11] Object File Format (OFF File Format). Retrieved August 2008 from http://ozviz.wasp.uwa.edu.au/~pbourke/dataformats/off/

[12] Stereolithography File Format (STL File Format). Retrieved August 2008 from http://mech.fsv.cvut.cz/~dr/papers/Lisbon04/node2.html

[13] Virtual Reality Modeling Language (VRML). Retrieved August 2008 from http://mkrus.free.fr/CG/vrml.html

[14] Polygon File Format (PLY File Format). Retrieved August 2008 from http://local.wasp.uwa.edu.au/~pbourke/dataformats/ply/

[15] Bruce G. Baumgart. A Polyhedron Representation for Computer Vision, National Computer Conference, Anaheim, CA, 1975, pg. 589-596.

[16] Bischoff Stephan and Kobbelt Leif. Teaching Meshes, Subdivision and Multiresolution Techniques, Computer Aided Design, Volume 36, Issue 14, December 2004, pg. 1483-1500.

[17] Weiler Kevin. Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments. IEEE Computer Graphics and Applications, 5(1):21-40, January 1985.

[18] Kettner Lutz. Using Generic Programming for Designing a Data Structure for Polyhedral Surfaces, 14th Annual ACM Symposium on Computational Geometry, 1998.

[19] A Sample Half-edge Data Structure Implementation. Retrieved August 2008 from http://www.holmes3d.net/graphics/dcel/

[20] Gubias Leonidas and Stolfi Jorge. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams, ACM Transaction on Graphics, 4(3):74-123, July 1985.

[21] Weiler Kevin. The Radial-edge Data Structure: A Topological Representation for Non-manifold Geometric Boundary Modeling, Geometric Modeling for CAD, 1988.

[22] Leila De Floriani and Annie Hui. Data Structures for Simplicial Complexes: An Analysis and a Comparison, Eurographics Symposium on Geometry Processing, 2005.

[23] Leila De Floriani and Annie Hui. A Scalable Data Structure for Three-Dimensional Non-manifold Objects, Symposium on Geometry Processing, pg. 72-82, 2003.

[24] Visualization Tookit Library (VTK). Retrieved August 2008 from http://www.vtk.org/

[25] Computational Geomerty Algorithms Library (CGAL). Retrieved August 2008 from thttp://www.cgal.org/

[26] OpenMesh Library. Retrieved August 2008 from http://www.openmesh.org/

[27] Kitware, Inc. Retrieved August 2008 from http://www.kitware.com/

[28] The Visualization Toolkit User's Guide, 2004.

[29] CGAL User and Reference Manual, Release 3.3.1

[30] Schroeder Will, Martin Ken and Lorensen Bill. The Visualization Toolkit An Object-Oriented Approach To 3D Graphics, 3rd Edition.

[31] Heckbert Paul S. and Garland Michael. Survey of Simplification Algorithms, In Proceedings of ACM SIGGRAPH 97.

[32] Lorenson W. and Cline H. Marching Cubes: A High Resolution 3D Surface Reconstruction Algorithm, In Proceediings of ACM SIGGRAPH 87.

[33] Aim@Shape Repository. Retrieved August 2008 from http://shapes.aim-at-shape.net/

[34] Gotsman C., Gumhold S. and Kobbelt L.. Simplification and Compression of 3D Meshes, In "Tutorials on Multiresolution in Geometric Modeling", A. Iske, E. Quak, M.S. Floater (Eds.), Springer, 2002.

[35] Luebke D., Reddy M., Cohen Jonathan D., Varshney A., Watson B. and Huebner R. Level of Detail for 3D Graphics, Morgan Kaufmann, 2003.

[36] Heckbert Paul S. and Garland Michael. Multiresolution Modeling for Fast Rendering, In Proceedings of Graphics Interface, 1994.

[37] Cignoni P., Montani C., Scopigno R. A Comparison of Mesh Simplification Algorithms, 1997.

[38] Talton Jerry O. A Short Survey of Mesh Simplification Algorithms, University of Illionis at Urbana-Champaign, Technical Report, 2004.

[39] Erikson Carl. Polygonal Simplification: An Overview, UNC-Champel Hill Computer Science Technical Report TR96-016, 1996.

[40] Luebke D. A Survey of Polygonal Simplification Algorithms, University of North Carolina, Technical Report TR97-045.

[41] Schroeder William J., Zarge Jonathan A. and Lorensen William E. Decimation of Triangle Meshes, In proceedings of ACM SIGGRAPH 92, pg. 65-70, 1992.

[42] Kobbelt L., Campagna S., and Seidel H. Peter. A General Framework for Mesh Decimation, In proceedings of Graphics Interface 98, pg. 43-50, 1998.

[43] Schroeder W. A Topology modifying progressive decimation algorithm. In IEEE Visualization 97 Conference Proceedings, pg. 205-212, 1997.

[44] Garland M. and Heckbert Paul S. Surface Simplification Using Quadric Error Metrics. In Proceedings of ACM SIGGRAPH 97, pg. 209-216, 1997.

[45] Ronfard R. and Rossignac J. Full-Range Approximations of Triangulated Polyhedra, In Proceedings of Eurographics, Vol. 15, C-67, 1996.

[46] Garland M. Quadric-based Polygonal Surface Simplification, Ph.D. Thesis, Carnegie Mellon University.

[47] Cignoni P., Rochini C., and Scopigno R. Metro: Measuring Error on Simplified Surfaces, Technical Report B4-01-96, I.E.I.-C.N.R., Pisa, Italy, 1996.

[48] Klein R., Liebich G., and Straßer W. Mesh Reduction with Error Control, In Proceedings of Visualization 96, pg. 311-318, 1996.

[49] Cohen J., Varshney A., Manocha D., Turk G., Weber H., Agarwal P., Brooks F. and Wright W. Simplification Envelops, In Proceedings of ACM SIGGRAPH 96, 1996.

[50] Rossignac J. and Borrel P. Multi-resolution 3D Approximations for Rendering Complex Scenes, Geometric Modeling in Computer Graphics: Methods and Applications, Springer-Verlag, Berlin, New York, pg. 455-465, 1993.

[51] Lindstrom P. Out-of-core Simplification of Large Polygonal Models, In Proceedings of ACM SIGGRAPH 00, pg. 259-262, 2000.

[52] Hoppe H., Progressive Meshes, In Proceedings of ACM SIGGRAPH 96, pg. 99-108 , 1996.

[53] Garland M. and Heckbert Paul S. Simplifying Surfaces with Color and Texture using quadric error metrics. In Proceedings of Visualization 98, IEEE Computer Soc. Press, Oct. 1998, pg. 263-269, 1998.

[54] Hoppe H. New Quadric Metric for Simplifying Meshes with Appearance Attributes. In Visualization 99, IEEE, pg. 59-66, 1999.

[55] Lindstrom P. and Turk G. Fast and Memory Efficient Polygonal Simplification, In Proceedings of Visualization 98, IEEE Computer Soc. Press, Oct. 1998, pg. 279-286.

[56] Lindstrom P. and Turk G. Evaluation of Memoryless Simplification, IEEE Transactions on Visualization and Computer Graphics, 5(2): 98-115, 1999.

[57] Roy M., Foufou S. and Truchetet F. Mesh Comparison using Attribute Deviation Metric, In International Journey of Image and Graphics (IJIG), 4 (1), pages 127-140, January 2004.

[58] Silva S., Maderia J. and Santos B. Sousa. POLYMECO: A Polygonal Mesh Comparison Tool, Ninth International Conference on Information Visualization 2005, pg. 842-847.

[59] Aspert N., Santa-Cruz D. and Ebrahmi T. MESH: Measuring Error Between Surfaces Using the Hausdorf Distance. In Proceedings of the IEEE International Conference on Multimedia and Expo 2002, vol. I, pg. 705-708.

[60] Alliez P., Ucelli G., Gotsman C. and Attene M. Recent Advances in Remeshing of Surfaces, state-of-the-art report, 2005.

[61] Gu X., Gortler S. J. and Hoppe H. Geometry Images. In Proceedings of ACM SIGGRAPH 2002, pg. 355-361.

[62] Praun E. and Hoppe H. Spherical Parameterization and Remeshing, In Proceedings of ACM SIGGRAPH 2003, pg. 340-349.

[63] Losasso F., Hoppe H., Schaefer S. and Warren J. Smooth Geometry Images, Eurographics Symposium on Geometry Processing 2003, pg. 138-145.

[64] Floater M. Parameterization and Smooth Approximation of Surface Triangulations, CAGD 14, 3 (1997), pg. 231-250.

[65] Sander P., Gortler S., Snyder J. and Hoppe H. Signal Specialized Parameterization, Microsoft Research MSR-TR-2002-27, January 2002.

[66] Sheffer A., Praun E., and Rose K. Mesh Parameterization Methods and Their Applications, Foundations and Trends in Computer Graphics and Vision 2(2):105-171, 2006.

[67] Floater M. Mean Value Coordinates, CAGD, 20(1), 19-27, 2003.

[68] Cyberware Inc. Retrieved August 2008 from http://www.cyberware.com/

[69] Shewchuk J.R. What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In proceedings of 11th International Meshing Roundtable, 2002.

[70] The Stanford 3D Scanning Repository. Retrieved August 2008 from http://www-graphics.stanford.edu/data/3Dscanrep/

APPENDIX A

K-SIMPLEX AND SIMPLICIAL COMPLEXES

Hoffman [5] defines a k-simplex as follows: a k-simplex is the convex combination of k + 1 linearly independent points. The dimension of the k-simplex is k. It is obvious that a 0-simplex is a point, 1-simplex is a line segment and 2-simplex is a triangle. For example, consider a three distinct point p_1, p_2 and p_3. The convex combination spanned by p_1, p_2 and p_3 is the set

$$\begin{aligned}\langle p_1, p_2, p_3 \rangle &= \left\{ \left(\lambda p_1 + (1-\lambda) p_2 \right) \mu + (1-\mu) p_3 \mid 0 \le \lambda, \mu \le 1 \right\} \\ &= \left\{ \mu q + (1-\mu) p_3 \mid q \in \langle p_1, p_2 \rangle, 0 \le \mu \le 1 \right\}\end{aligned}$$

Geometrically, if p_1, p_2 and p_3 are collinear than then $\langle p_1, p_2, p_3 \rangle$ is a triangle with vertices p_1, p_2 and p_3.

The boundary of a k-simplex S consists of all (k-d)-simplices contained in S, where k > 0. Every simplex in the boundary of S is a face of S. For instance, if S is a 2-simplex (triangle), then the boundary of S is composed of 0-simplices (vertices) and 1-simplices (edges) which are also a face of S. A k-simplex contains exactly $\binom{k+1}{d+1}$ d-simplices as faces.

Hoffmann [5] also defines simplicial complexes as a set of finite simplices, satisfying the following conditions:

- Let K be a simplicial complex and S is a simplex in K. Then, any face of a S is also in K.

- The intersection of any two simplices $S_1, S_2 \in K$ is either empty or is a simplex in K.

Observe that in the second condition that, $S_1 \cap S_2$ is a face of both S_1 and S_2. The dimension of a simplicial complex is defined as the maximum dimension of the simplices in it. Figure-66 and 67 show a sets of simplices one of which forms a simplicial complex whereas the other does not.

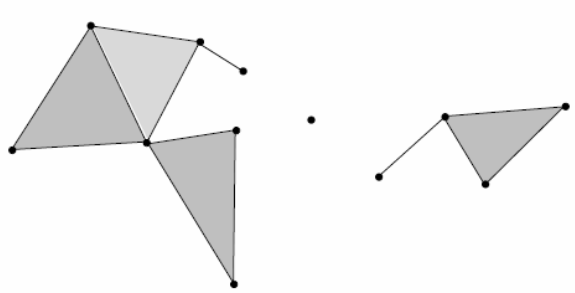

Figure 66. A simplex forming a simplicial complex

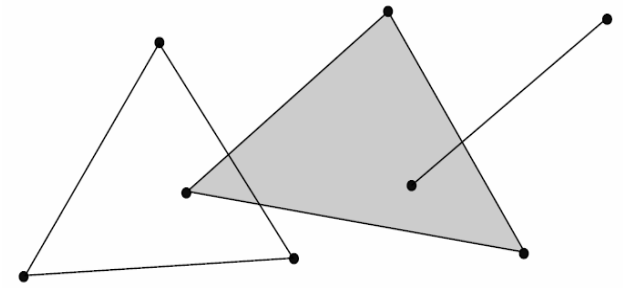

Figure 67. A simplex not forming a simplicial complex

APPENDIX B

MESH FILE FORMAT EXAMPLES

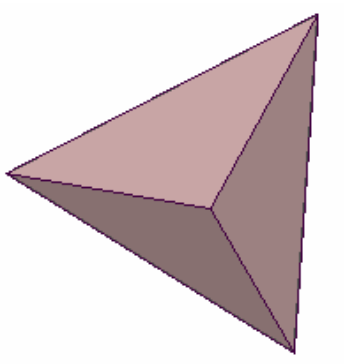

Figure 68. A 3D tetrahedron

The 3D tetra hedron shown in Figure-68 can be represented in different file formats as follows:

- .OBJ file format

```
v 1.00 1.00 1.00
v 2.00 1.00 1.00
v 1.00 2.00 1.00
v 1.00 1.00 2.00
f 1 3 2
f 1 4 3
f 1 2 4
f 2 3 4
```

- . OFF file format

```
OFF
5 4 6
0 0 0
1 1 2
1 2 1
2 1 1
1 1 1
```

```
3 3 2 1
3 4 3 1
3 4 1 2
3 4 2 3
```

- STL ASCII format

```
solid tetrahedron
  facet normal 0.000000 0.000000 -1.000000
    outer loop
      vertex 1.000000 1.000000 1.000000
      vertex 1.000000 2.000000 1.000000
      vertex 2.000000 1.000000 1.000000
    endloop
  endfacet
  facet normal -1.000000 0.000000 0.000000
    outer loop
      vertex 1.000000 1.000000 1.000000
      vertex 1.000000 1.000000 2.000000
      vertex 1.000000 2.000000 1.000000
    endloop
  endfacet
  facet normal 0.000000 -1.000000 0.000000
    outer loop
      vertex 1.000000 1.000000 1.000000
      vertex 2.000000 1.000000 1.000000
      vertex 1.000000 1.000000 2.000000
    endloop
  endfacet
  facet normal 0.577350 0.577350 0.577350
    outer loop
      vertex 2.000000 1.000000 1.000000
      vertex 1.000000 2.000000 1.000000
      vertex 1.000000 1.000000 2.000000
    endloop
  endfacet
    endsolid tetrahedron
```

- STL Binary format

```
56 43 47 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 04 00 00
00 00 00 00 00 00 00 00 00 00 00 00 80 BF
```

APPENDIX C

SAMPLE FIGURES FROM POLYGONAL MESH SIMPLIFICATION

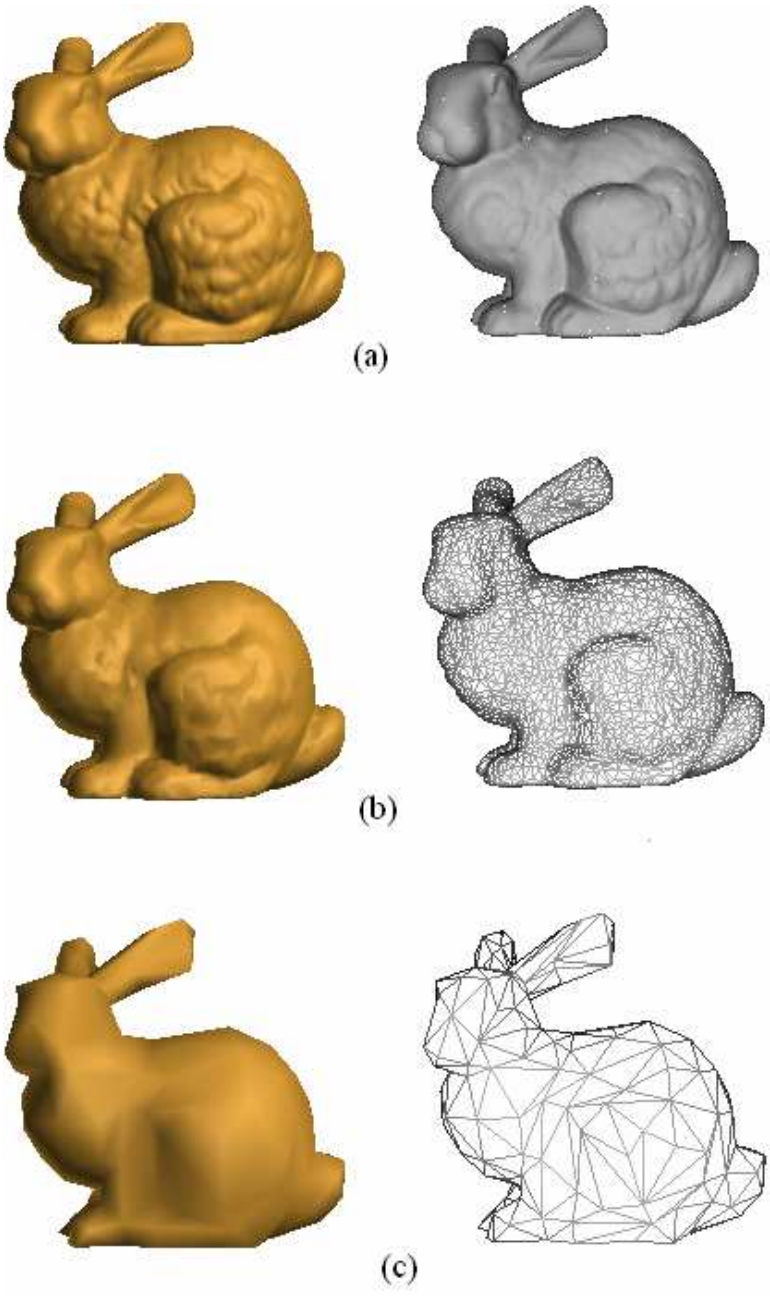

Figure 69. Stanford Bunny Model: shaded and wireframe models original model (a), 90% reduced model (b), 99% reduced model (c) CGAL Lindstrom – Turk algorithm is used for simplifications

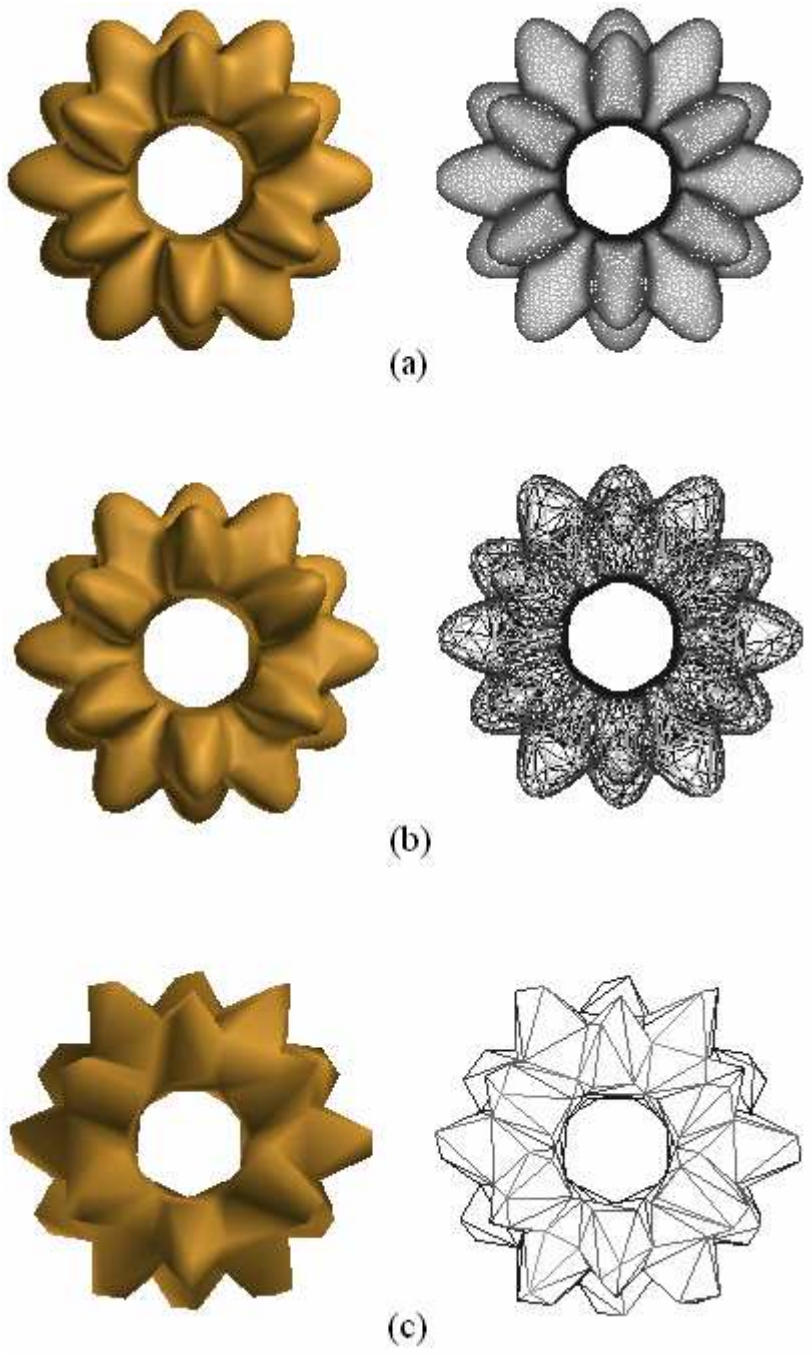

Figure 70. Bumpy Torus Model: shaded and wireframe models original model (a), 90% reduced model (b), 99% reduced model (c) vtkQuadricDecimation algorithm is used for simplifications

APPENDIX D

ACTUAL ERROR VALUES FOR THE STANFORD BUNNY MODEL

Table 8. Actual Geometric Errors for Stanford Bunny Model

Alg.	Reduc. Rate	Maximumum Error	Mean Error
vtkDecimatePro	10	0.000248	0.000004
	20	0.000333	0.000008
	30	0.000469	0.000012
	40	0.000484	0.000017
	50	0.000484	0.000023
	60	0.000684	0.000031
	70	0.000687	0.000042
	80	0.000711	0.000072
	90	0.004713	0.000245
vtkDecimatePro with error acc.	10	0.000248	0.000004
	20	0.000268	0.000008
	30	0.000383	0.000012
	40	0.000493	0.000016
	50	0.000493	0.000021
	60	0.000493	0.000027
	70	0.000602	0.000038
	80	0.000627	0.000065
	90	0.001812	0.000181
vtkQuadricDecimation	10	0.000037	0.000001
	20	0.000050	0.000002
	30	0.000062	0.000004
	40	0.000070	0.000007
	50	0.000094	0.000010
	60	0.000115	0.000013
	70	0.000163	0.000019
	80	0.000290	0.000027
	90	0.000524	0.000049
vtkQuadricDecimation with vertex normals	10	0.000193	0.000004
	20	0.000292	0.000010
	30	0.000366	0.000019
	40	0.000790	0.000032
	50	0.001016	0.000053
	60	0.001585	0.000083
	70	0.001585	0.000130
	80	0.002424	0.000208
	90	0.003115	0.000388

(Table 8. continued)

Alg.	Reduc. Rate	Maximumum Error	Mean Error
vtkQuadricClustering	10	0.000775	0.000005
	20	0.001134	0.000008
	30	0.001087	0.000012
	40	0.001674	0.000016
	50	0.001823	0.000020
	60	0.001845	0.000026
	70	0.001950	0.000035
	80	0.002298	0.000052
	90	0.003145	0.000103
CGAL Edge-length Midpoint	10	0.000540	0.000006
	20	0.000556	0.000011
	30	0.000622	0.000016
	40	0.000622	0.000022
	50	0.000772	0.000032
	60	0.001026	0.000044
	70	0.001176	0.000062
	80	0.001712	0.000097
	90	0.002549	0.000198
CGAL Lindstrom-Turk	10	0.000040	0.000001
	20	0.000068	0.000002
	30	0.000120	0.000004
	40	0.000120	0.000006
	50	0.000159	0.000008
	60	0.000166	0.000011
	70	0.000275	0.000015
	80	0.000450	0.000022
	90	0.000614	0.000037
OpenMesh Quadric Module	10	0.000783	0.000001
	20	0.000783	0.000002
	30	0.000842	0.000004
	40	0.001136	0.000007
	50	0.001136	0.000011
	60	0.001336	0.000016
	70	0.001336	0.000023
	80	0.001336	0.000037
	90	0.001901	0.000071
OpenMesh Roundness with Normal Flipping	10	0.000921	0.000003
	20	0.001175	0.000006
	30	0.001175	0.000010
	40	0.001412	0.000015
	50	0.001537	0.000022
	60	0.001537	0.000036
	70	0.001716	0.000056
	80	0.004585	0.000119
	90	0.004496	0.000256

APPENDIX E

ACTUAL ERROR VALUES FOR THE BUMPY TORUS MODEL

Table 9. Actual Geometric Errors for Bumpy Torus Model

Alg.	Reduc. Rate	Maximumum Error	Mean Error
vtkDecimatePro	10	0.080670	0.001731
	20	0.127255	0.004055
	30	0.155406	0.006285
	40	0.161423	0.008665
	50	0.161400	0.011603
	60	0.161400	0.015993
	70	0.285699	0.026186
	80	0.531355	0.043176
	90	1.355316	0.089026
vtkDecimatePro with error acc.	10	0.096902	0.001583
	20	0.108504	0.003384
	30	0.108504	0.005329
	40	0.108504	0.007515
	50	0.109406	0.010333
	60	0.126558	0.014858
	70	0.151020	0.022565
	80	0.417685	0.039623
	90	0.500582	0.060295
vtkQuadricDecimation	10	0.019928	0.000287
	20	0.019928	0.000770
	30	0.024366	0.001418
	40	0.028185	0.002266
	50	0.032186	0.003320
	60	0.043139	0.004687
	70	0.103235	0.006665
	80	0.103235	0.010078
	90	0.149019	0.019524
vtkQuadricDecimation with vertex normals	10	0.012065	0.000296
	20	0.016717	0.000790
	30	0.021020	0.001453
	40	0.028134	0.002306
	50	0.029778	0.003343
	60	0.039433	0.004702
	70	0.061458	0.006692
	80	0.078635	0.010111
	90	0.149095	0.019508

(Table 9. continued)

Alg.	Reduc. Rate	Maximumum Error	Mean Error
vtkQuadricClustering	10	0.079655	0.001359
	20	0.073820	0.002322
	30	0.079422	0.003536
	40	0.098697	0.004827
	50	0.110754	0.006307
	60	0.155166	0.008693
	70	0.216831	0.011999
	80	0.238422	0.018843
	90	0.687902	0.038458
CGAL Edge-length Midpoint	10	0.115418	0.001511
	20	0.161750	0.003337
	30	0.228665	0.005679
	40	0.248385	0.008413
	50	0.248385	0.011910
	60	0.256983	0.017129
	70	0.298007	0.025719
	80	0.352154	0.041994
	90	0.668535	0.089926
CGAL Lindstrom-Turk	10	0.036667	0.000317
	20	0.039144	0.000732
	30	0.045944	0.001299
	40	0.064101	0.001983
	50	0.064101	0.002838
	60	0.075849	0.003929
	70	0.080270	0.005422
	80	0.124495	0.008072
	90	0.325451	0.015243
OpenMesh Quadric Module	10	0.022744	0.000313
	20	0.036708	0.000933
	30	0.050539	0.001866
	40	0.050235	0.003151
	50	0.055351	0.004842
	60	0.076547	0.007182
	70	0.103148	0.010831
	80	0.151889	0.017472
	90	0.243354	0.035349
OpenMesh Roundness with Normal Flipping	10	0.052768	0.001228
	20	0.060556	0.002899
	30	0.078205	0.005211
	40	0.107309	0.007921
	50	0.128119	0.011282
	60	0.146445	0.017084
	70	0.253227	0.026413
	80	0.398908	0.041322
	90	0.746391	0.079289

APPENDIX F

A SAMPLE OUTPUT OF THE METRO TOOL

Metro tool has both numerical and visual output (Figure - X). Numerical output consists of the input mesh information and forward (distance from the first model to the second) and backward (distance from the second model to the first one) distances. A sample numerical output of the metro tool is given below:

```
        Metro V.4.06
    http://vcg.isti.cnr.it
  release date: Oct  3 2005
------------------------------

read mesh `sb_original.off'
read mesh `sb_cgalLT_0.2.off'

Mesh info:
        M1: 'sb_original.off'
                vertices   34834
                faces      69451
                area       0.1143
                bbox (-0.0947  0.0330 -0.0619)-( 0.0610  0.1873  0.0588)
                bbox diagonal 0.250246

        M2: 'sb_cgalLT_0.2.off'
                vertices   27881
                faces      55545
                area       0.1143
                bbox (-0.0947  0.0330 -0.0619)-( 0.0610  0.1873  0.0588)
                bbox diagonal 0.250265
```

Forward distance (M1 -> M2):

target # samples : 694510
target # samples/area : 6078459.558555
Vertex sampling
Edge sampling
Similar Triangles face sampling

distances:

max : 0.000046 (0.000171 wrt bounding box diagonal)
mean : 0.000002
RMS : 0.000004

# vertex samples	34834
# edge samples	368517
# area samples	256323
# total samples	659674
# samples per area unit:	5773569.467438

Backward distance (M2 -> M1):

target # samples : 694510
target # samples/area : 6077768.630400
Vertex sampling
Edge sampling
Similar Triangles face sampling

distances:

max : 0.000068 (0.000255 wrt bounding box diagonal)
mean : 0.000002
RMS : 0.000004

# vertex samples	27881
# edge samples	335675
# area samples	303072
# total samples	666628
# samples per area unit:	5833768.767255

Hausdorff distance : 0.000068 (0.000255 wrt bounding box diagonal)

Computation time : 6407 ms

samples/second : 207008.262370

Besides the numerical results metro can also give visual output (Figure-71, 72) by coloring the vertices of the original model with respect to the calculated error values at those vertices. The red regions can be considered as relatively low error regions whereas as the color deviates to yellow, green and blue then the corresponding error increases respectively.

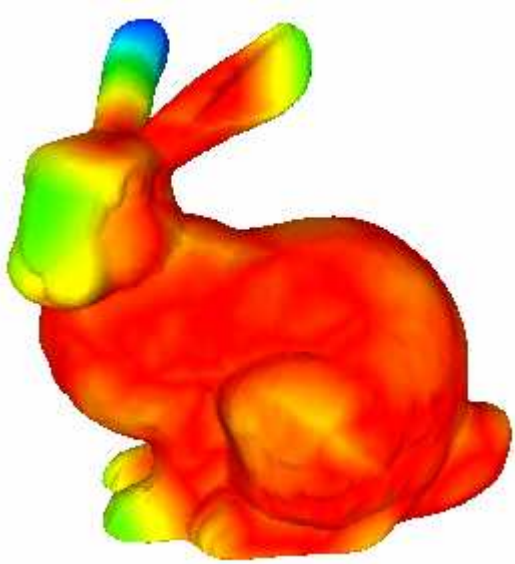

Figure 71. A sample visual output of the metro tool for the Stanford Bunny model

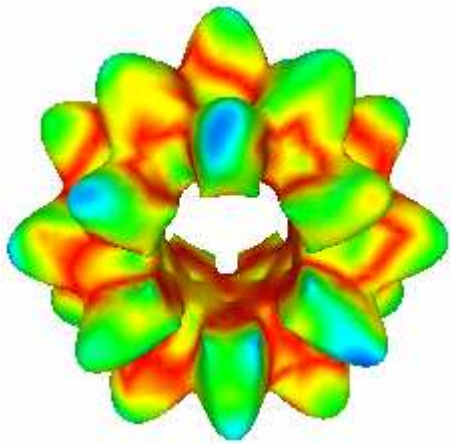

Figure 72. A sample visual output of the metro tool for the Bumpy Torus model