Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Methods for Polygonal Mesh Simplification

Ing. Martin Franc

disertační práce
k získání akademického titulu
v oboru Informatika a výpočetní technika

Školitel: Prof. Ing. Václav Skala, CSc.
Katedra: Katedra informatiky a výpočetní techniky

Plzeň 2007

Abstract

A common task in computer graphics is to visualise models of real-world objects. The visualisation of large and complex models is required more and more frequently. This is followed by number of operations which must be done before the own visualisation, whether it be an analysis of input data (e.g. searching for an iso-surface) or a model simplification. Surfaces of such models are usually represented by triangular meshes and often contain thousands or millions of triangles. Since a fast interaction with these models is desired, such as real time visualization, we need either to improve our graphics hardware or to simplify somehow the complexity of the mesh. In spite of huge progress made in graphics hardware field in last years, we still need to increase a performance using optimal algorithms and programming techniques. One of the techniques that enhance the power is parallel computation.

In this work we present an original efficient and stable algorithm for triangle mesh simplification also in parallel environment. We use a method based on our original super independent set of vertices to avoid critical sections. The advantage of this algorithm is its high speed even in sequential run and reasonable quality of resulting approximation.

Since more than just geometrical error matters in estimation of simplification algorithms, we present a new algorithm for triangular mesh simplification with respect to the similarity of appearance of the original model and resulting approximation. We introduce an approach how to estimate a new vertex position during an edge collapse algorithm based on supposed surface curvature.

Abstrakt

Jedním z hlavních úkolů počítačové grafiky je vizualizace modelů objektů z reálného světa. Stále častěji je požadováno zobrazování rozsáhlých modelů. To jde ruku v ruce s počtem a složitostí operací, které musí být provedeny ještě před samotným zobrazením, ať už se jedná o transformaci vstupních dat (např. hledání iso-plochy), nebo třeba zjednodušení daného modelu. Výsledné povrchy těchto modelů jsou obvykle reprezentovány trojúhelníkovou sítí obsahující často tisíce až miliony trojúhelníků. Jelikož požadujeme rychlou interakci s těmito modely (např. zobrazování v reálném čase), je třeba buď zdokonalovat grafický hardware, nebo automatizovaně zjednodušovat jejich narůstající složitost. Navzdory obrovskému vývoji na poli grafických karet a akcelerátorů, stále potřebujeme zvyšovat výkon využíváním optimálních algoritmů a programovacích technik. Jednou z takových technik může být například paralelizace.

V této práci je prezentován původní algoritmus pro redukci trojúhelníkových sítí, který je efektivní, stabilní a pracuje i v paralelním prostředí. Jedná se o postup založený na tzv. super-nezávislé množině vrcholů, díky které zamezíme kritickým sekcím v paralelním kódu. Předností tohoto algoritmu je vysoká rychlost i při sekvenčním běhu a přiměřená kvalita výsledné aproximace.

Jelikož geometrická chyba nemusí být jediným kritériem při posuzování výsledků simplifikačního algoritmu, představujeme novou metodu, která je založena na zachování podobnosti vzhledu mezi původním modelem a výslednou aproximací. Představujeme zde nový přístup jak určit novou pozici vrcholu při kontrakci hran, založený na předpokládané křivosti povrchu v daném místě.

Acknowledgement

I would like to thank my supervisor, prof. Václav Skala, for his support and guidance during my PhD study.
Many special thanks go to the other members of our computer graphics group at University of West Bohemia in Pilsen.
Not least, I would like to thank my family and my wife for their patience and encouragement throughout my studies.

This work has been supported by the following projects:

- MSM235200005 and LC06008 Ministry of Education CR
- 3DTV FP6-2003-IST-2 project Network of Excellence, No:511568
- Computer Graphics and Visualization in Parallel and Distributed Environment, MSMT CR - VS 97 155

Contents

Chapter 1

Introduction

In all areas, which employ complex models, there is a trade off between the accuracy with which the surface is modelled and the time needed to process it. To achieve acceptable running times, we must often substitute simpler approximations of the original model. A model, which captures very fine surface detail, may in fact be desirable when creating archival datasets; it helps ensure that applications that later process the model have sufficient accurate data. However, many applications will require far less detail than is present in the full dataset. Therefore, techniques for the simplification of large and highly detailed polygonal meshes have been developed.

This work is focused on the automatic simplification of highly detailed polygonal models to get an approximation containing fewer polygons making its surface (see Figure 1-1).

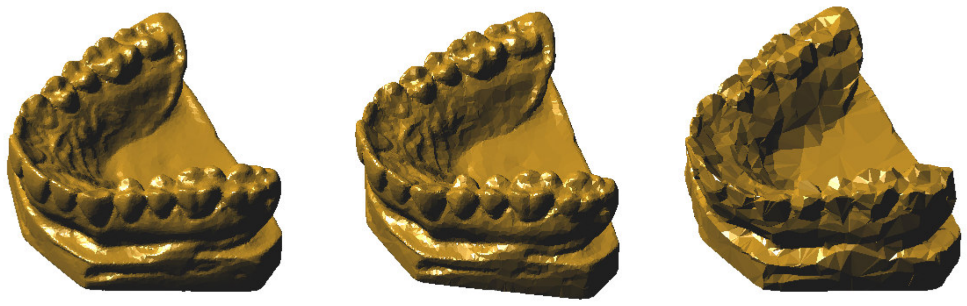


Figure 1-1: Automatic simplification of polygonal model (courtesy Cyberware). From left: original, 50% fewer polygons, 90% fewer polygons.

We will describe the simplification algorithm which has been developed to meet criteria such as high speed, large input capability and good approximation quality. In addition a super-independent set is introduced as a special vertices selection for removal during simplification process. Using such a set we can benefit from a parallel implementation without critical sections.

1.1 Motivation

One of the main tasks in computer graphics is a visualization of scientific data. Due to the wide technological advances in the field of computer graphics during the last few years, there has been an expansion of applications dealing with models of real world objects. Advances in technology have provided vast databases of polygonal (typically triangular) surface models, but these models are often very complex. With growing demands on quality, the complexity of the computations we have to handle models having hundreds thousands or perhaps even millions of triangles (for example well known model of Michelangelo's David [30] contains $2*10^9$ triangles, see Figure 1-2). The sources of such models are usually:

- Laser range scanners, computer vision systems, and medical imaging devices, which can produce models of real world objects.
- CAD systems, which commonly produce complex and high detailed models.
- Surface reconstruction or iso-surface extraction methods such as Marching Cubes algorithm [34] that produce models with a very high density of polygonal meshes displaying almost regular arrangement of vertices.

Figure 1-2: The digital Michelangelo scan of David (taken from [30]).

Since a fast interaction with the models is desired, such as real time visualization, we need either to improve our graphics hardware or to simplify somehow the complexity of the mesh. In recent years, many of mesh simplification algorithms has been developed. The aim of such techniques is to reduce the complexity of the model whilst preserving its important details.

Although, the first methods were focused on terrain or height field simplification, techniques for simplification of general 3D polygonal surfaces have been proposed relatively recently. At Siggraph '92 Shroeder et al. [46] presented algorithm called triangle decimation based on local vertex deletion followed by re-triangulation. Since that time other notable algorithms have been presented including methods that are guaranteed accurate within global error bounds [23] or within a simplification envelope [6].

Many algorithms are designed to preserve the original topology of the mesh. While this may be important for many applications (e.g. analysis or computational geometry), preserving topology introduces constraints into the reduction process. Mesh reduction is typically used to improve rendering speed or to minimize data size or compression requirements. In such applications topology-preserving reduction schemes could be incapable of achieving desired reduction levels. Removing topological constraint can create large gains in reduction factors.

Our goal has been to produce a simple and fast simplification algorithm that produces high quality results.

1.2 Simplification in Computer Graphics

At the beginning it is important to mention, that there are several forms of simplification in computer graphics. Some processes are so routine nowadays that we tend to disregard them, e.g. the quantization of colour to 24 bits or the use of three-color channels (red, green, and blue) to represent the spectral response of the virtual scene. Even storing a polygonal mesh may involve simplification, since we usually use limited precision of the coordinates (typically to 32 bits). Also the quantization of colour and geometric attributes, especially in interactive graphics, is indeed a simplification. This also implies a lot of benefits, for example use of fixed-point arithmetic, fewer bits of precision for vertex coordinates, etc. In general the line between simplification and losing compression techniques, which include quantization in its various forms, can be blurry. However, we will define a simplification throughout this work as processes that reduce the complexity of polygonal meshes, not their precision or storage size.

The thought of polygonal model simplification is not new. Proposals of some procedures are over 25 years old. The need of models with several levels of detail (LOD) arouse in military flight simulators and later in computer games

industry in general. In order to manage the level of detail of an object, we need to represent it as a multiresolution model – a surface representation that supports the reconstruction of various approximations, which can accommodate a wide range of viewing contexts.
We briefly describe three main frameworks of simplification algorithms:
discrete, continuous, and view-dependent LOD [36].

A **discrete LOD** is used in most 3D graphics applications today and we are mainly focused on it in this work as well. The main philosophy is to create multiple versions of every object, each at a different level of detail, during an offline pre-process. At run-time the appropriate level of detail, or LOD, is chosen to represent the object. Since long-distance objects use coarser LODs, the total number of polygons is reduced and rendering speed increased. Because individual approximations are computed offline during pre-processing, the simplification process cannot predict from what direction the object will be viewed. The simplification therefore typically reduces detail uniformly across the object, and for this reason we sometimes refer to discrete LOD as isotropic or view-independent LOD. This approach has many advantages. Since the simplification algorithm runs offline, it can take as long as it needs and run-time program simply choose which approximation to use. Furthermore, modern graphics hardware leads itself to the multiple model versions created by static level of detail. Individual approximations can be compiled during pre-processing to an optimal rendering format and use features such as triangle strips, display lists, and vertex arrays. These will be obviously rendered much faster than just an unordered list of polygons.

In **continuous LOD** (also called a progressive LOD [20]), rather than creating individual approximations during the pre-processing stage, we create a data structure encoding a continuous spectrum of detail. The desired level of detail is then extracted from this structure at run-time. A major advantage of this approach is better granularity: since the level of detail for each object is specified exactly rather than selected from a few pre-created options, no more polygons than necessary are used. This frees up more polygons for rendering other objects, which in turn use only as many polygons as needed for the desired level of detail, freeing up more polygons for other objects, and so on. Better granularity thus leads to better use of resources and higher overall fidelity for a given polygon count. Continuous LOD also supports streaming of polygonal models, in which a simple base model is followed by a stream of refinements to be integrated dynamically.

View-dependent LOD extends continuous LOD, using view-dependent simplification criteria to dynamically select the most appropriate level of detail for the current view. Nearby portions, or the silhouette regions of the object are shown at higher resolution than distant portions, or interior regions respectively. This leads to better fidelity for a given polygon count (see Figure 1-3).

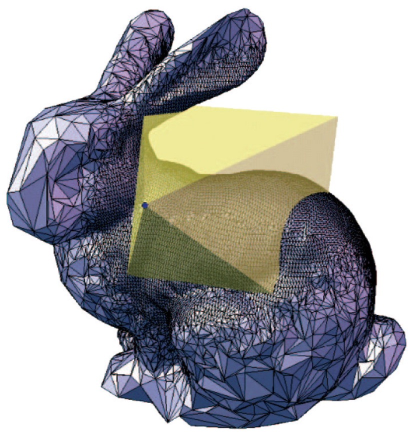


Figure 1-3: Example of view-dependent simplification (taken from [38]).

These methods are fairly necessary in visualisation of large objects, such as terrains or scientific data, which are represented by extremely large data sets and can not be adequately simplified in any other way.

It this work we will focus on a process of making discrete approximations driven by rules of budget-based simplification. However, algorithms presented here can be modified and used in continuous or view-dependent simplifications.

1.3 Contributions

The primary contributions of this work as described in this dissertation are:

- **Super-independent set of vertices**. We have defined a new criterion how to choose vertices as candidates for removal during simplification process. It is based on independent set of vertices [21] with more strict constraints to vertices neighbourhood. The use of super-independent set of vertices leads to ability to have a parallel code without critical sections in simplification process.
- **Surface Simplification Algorithm**. By combining several approaches of mesh simplification and principles of super-independent set of vertices,

we have developed a fast parallel algorithm capable to produce high-quality approximations of polygonal surfaces. This algorithm can simplify both manifold and non-manifold models. It's robust, very fast and accurate while preserving a mesh topology. Since the algorithm keeps the subset of original vertices, in addition to producing single approximations, it can be also used to generate multiresolution representations such as progressive meshes and vertex hierarchies for view-dependent refinement.

- **Edge classification and introduction of new vertex position**. Finally, we have introduced an original approach of edge evaluation and classification, which results in a new simplification algorithm. This algorithm mainly preserves the visual appearance by detecting and keeping important features of the original model such as sharp edges or high detail regions during even drastic simplification. While we suppose that original surface tends to be curved according to its vertex normals, a new vertex position is determined to lay on such supposed surface using near least-square curvatures.

1.4 Used terms

To streamline all the discussion and explanation, we present here a list of terms, abbreviations and symbols used later in this thesis.

By convention, all vectors in this text are assumed to be column vectors and are set in lowercase bold type. Therefore, $\mathbf{u}^T\mathbf{v} = \mathbf{u}.\mathbf{v}$ denotes the inner product of two column vectors **u** and **v**. However, in more complicated equations transposition notation will be used for better readability. Matrices are set in uppercase bold type, thus $\mathbf{A} = \mathbf{u}\mathbf{v}^T$ denotes the outer product matrix $a_{ij} = u_i v_j$. Also instead of vertex coordinates in space V=[x,y,z], we will use its radius vector (vector from origin to given coordinates $\mathbf{v}$ = (x-0,y-0,z-0) and will be typed in lowercase bold as a common vector.

Other terms are used as follows:

polygon

- usually triangular polygon since all general polygons can be transformed into a set of triangles

triangulation, polygonisation

- a triangular/polygonal mesh

mesh, polygonal mesh, triangular mesh (surface)

- input/output polygonal mesh as described in 2.1

reduction, simplification, decimation

- iterative process of decreasing number of vertices/edges/triangles in a mesh

LOD, multiresolution

- level of detail (one or more approximations of specific detail)

sharp edge

- an edge in triangulation whose adjacent triangles contains less angle than some specific threshold

In equations and formal text following symbols and their meaning are used:

N	number of vertices
i,j,k	iteration symbols
M	polygonal mesh
$\mathbf{n}$	norm vector
$\mathbf{x}$	vector

For a set of vertices will be used notation $V = (\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3,\ldots,\mathbf{v}_n)$. Analogously, for a set of polygons (triangles, faces) is used $F = (\mathbf{f}_1,\mathbf{f}_2,\mathbf{f}_3,\ldots,\mathbf{f}_n)$. Polygonal model is then referenced as a pair of $M=(V,F)$.

As a preface of next chapters it must be explained that for readability we have decided to use loose informal language when discussing topology, and in particular have avoided the use of simplicial complex notation. For example, we use expression such as "the set of triangles surrounding edge e" instead of the comprehensive notation $\lceil \lceil \lfloor e \rfloor \rceil \rceil$. However, for completeness, these formal definitions are mentioned in Chapter 2.

1.5 Overview of Material

This work is structured as follows. In Chapter 2 theoretical background and related work is presented. We define terms such as polygonal mesh and mesh topology. Also some base conditions and simplification operators are discussed and frequently used data structures are mentioned. Chapter 3 provides information about error estimation. Several metrics are described for different simplification approaches and METRO tool is presented. Having established the background information, we walk through state-of-the-art algorithms divided into lucid categories in Chapter 4. After methods survey Chapter 5 introduces our original algorithm based on some conclusions made in previous sections. Besides original simplification technique also a parallel processing is considered and super-independent set of vertices is presented. In Chapter 6 we show experimental results and discuss shortcomings of presented approach. Chapter 7 introduces a heuristic for mesh simplification with respect to a model appearance. Finally Chapter 8 concludes the work, recalls the main contributions and briefly suggests a possible future work.

Chapter 2

Background & Related Work

This chapter provides an overview of background material used throughout the rest of this dissertation. We describe mesh simplification as an optimization process under given conditions. Before we review some simplification approaches developed by others, we need to define a polygonal (triangular) surface as well as terms like a topology, manifold, or a non-manifold respectively.

After describing the range of mesh simplification operators available, we conclude with a discussion of the data structure and present some commonly used approaches to store the data introduced in last few years.

2.1 Surface Representation

The aim of polygonal surface simplification is to provide a mechanism for controlling the complexity of polygonal surface models, but these are not the only available surface representation. Various alternatives exist, and they each provide certain benefits and drawbacks as compared with polygonal models. However, none of these alternatives provide a solution which would obviate the need for simplification. In fact, they suffer from some of the same problems addressed by polygonal surface simplification.

The most important reason to focus on polygonal models is purely pragmatic: polygonal models are both flexible and ubiquitous. They are supported by the vast majority of rendering and modelling packages, and polygonal surface data is widely available. Hardware acceleration of polygon rendering is also becoming much more widely available; affordable yet reasonably powerful accelerator cards are now available in consumer-level computers. Currently, no other single type of model enjoys the same level of support. In fact, it is common practice in various situations to convert other model types into polygonal surfaces prior to processing.

In the most general sense, a polygonal surface model is simply a set of planar polygons in the three-dimensional Euclidean space $\mathbf{R}^3$. Without loss of generality, we can assume that the model consists entirely of triangular faces, since any non-triangular polygons may be triangulated in a pre-processing phase. The example of triangular surface model can be seen on Figure 2-1.

To streamline the discussion, we will assume that the models do not contain isolated vertices and edges, thus all vertices and edges are part of any triangle. Although the underlying algorithm stays the same, to handle such vertices and edges the implementation becomes more complicated. We also do not alter the mesh connectivity - if the corners of two triangles coincident in the space, then these triangles do not need to share a common vertex. Given these assumptions, we use a definition according to [11]. A polygonal surface model $M = (V,F)$ is a pair containing list of vertices V and a list of triangles F. The vertex list $V = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_r)$ is an ordered sequence, each vertex may be identified by a unique integer i. The face list $F = (f_1, f_2, \ldots, f_n)$ is also ordered, assigning a unique integer to each face. Every vertex $\mathbf{v}_i = [x_i\ y_i\ z_i]^T$ is a column vector in the Euclidean space $\mathbf{R}^3$. Each triangle $f_i = (j, k, l)$ is an ordered list of three indices identifying the corners $(\mathbf{v}_j, \mathbf{v}_k, \mathbf{v}_l)$ of f_i.

By design, this definition of a polygonal model corresponds to a form of simplicial complex. For our purposes here, a simplex σ is either a vertex (or 0-simplex), a line segment (1-simplex), or a triangle (2-simplex). In general, a k-simplex σ^k is the smallest closed convex set[1] defined by k+1 linearly independent points $\sigma^k = a_0a_1\ldots a_k$ which are called its vertices. We can express any point p within this set as a convex combination of the vertices $p = \sum_i t_i a_i$ where $\sum_i t_i = 1$ and $t_i \in [0,1]$. Any simplex defined by a subset of the points $a_0a_1\ldots a_k$ is a subsimplex of the simplex σ^k. A two-dimensional simplicial complex K is a collection of vertices, edges, and triangles satisfying the conditions:

1. If $\sigma_i, \sigma_j \in K$, then they are either disjoint or intersect only at a common subsimplex. Specifically, two edges can only intersect at a common vertex, and two faces can only intersect at a shared edge or vertex.
2. If $\sigma_i \in K$, then all of its subsimplices are in K. For instance, if a triangle f is in K, then its vertices and edges must also be in K.

[1] In other words, the convex hull

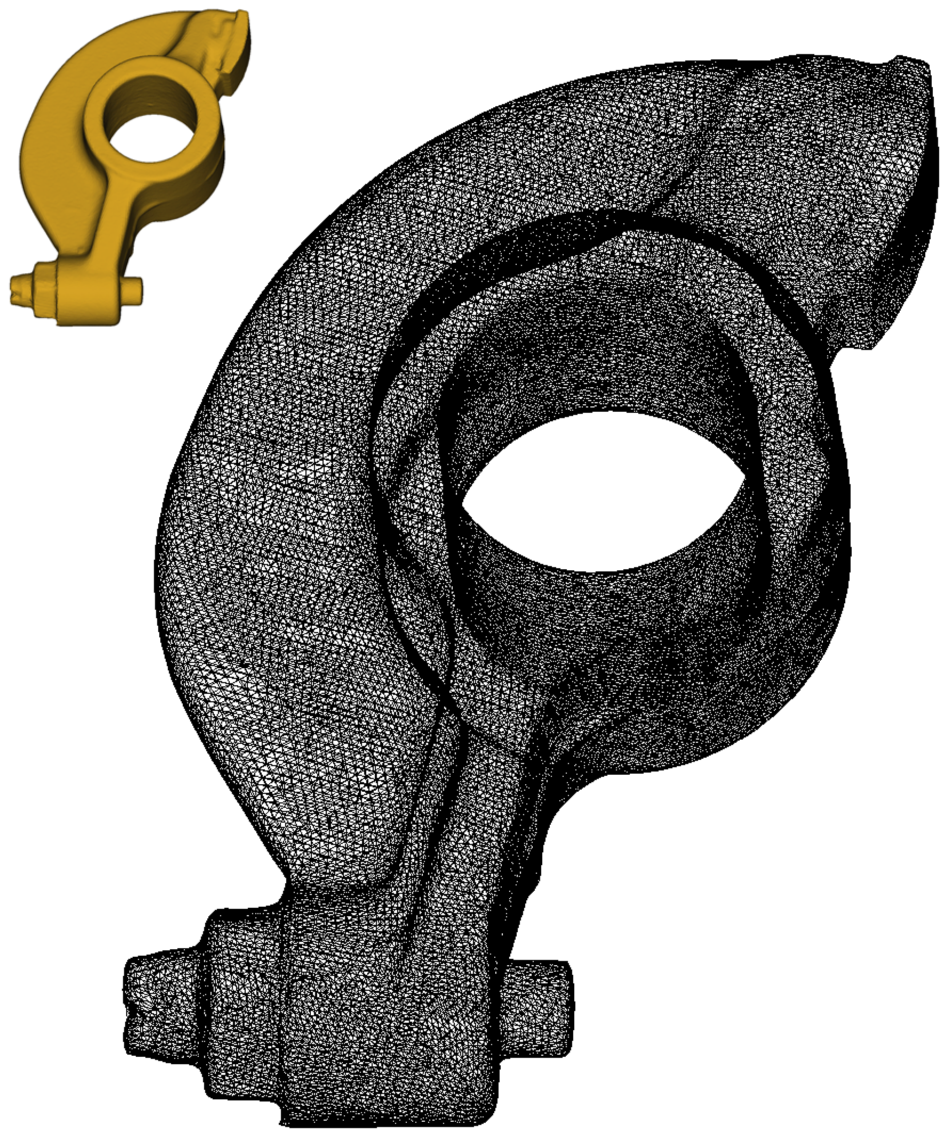


Figure 2-1: The example of polygonal surface model (courtesy Cyberware). This original model contains 40117 vertices and 80354 triangles.

The surface defined by this complex is the union of the point sets defined by its constituent simplices. Our definition of a polygonal model is slightly different and it is only explicitly a collection of vertices and faces. The only allowable edges are those which are implied by the intersection of neighbouring faces.

Manifold and Non-manifold Surfaces

Surfaces, in the mathematical sense, are often assumed to be manifolds. A manifold is a topological space that is locally Euclidean (i.e. around every point, there is a neighbourhood that is topologically the same as the open unit ball in $\mathbf{R}^n$). In other words, a manifold surface is, everywhere, locally homomorphic (that is, of comparable structure) to a two-dimensional disk (manifold surfaces with boundary are everywhere homomorphic to a disk or a half-disk).

For example, a disk may be fully applied to any portion of the torus, but the disk does not fully apply to all points of a pot (see Figure 2-2). In particular, the disk is truncated along the upper boundary of the pot bowl.

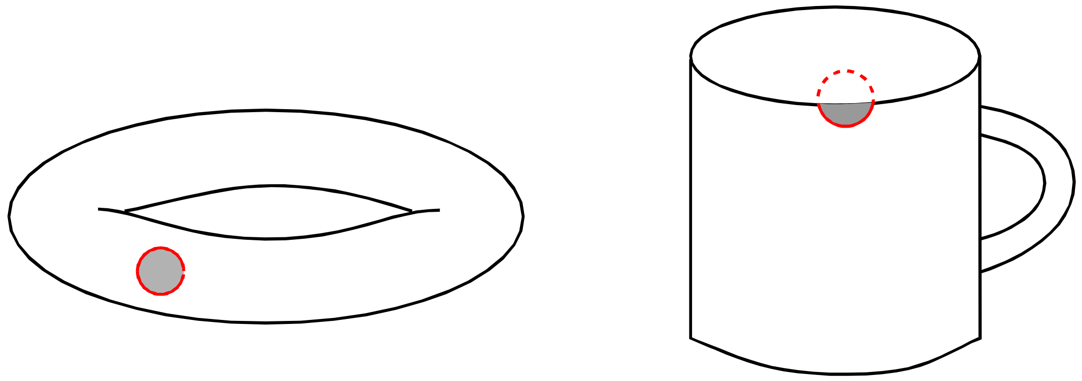


Figure 2-2: Manifold and manifold-with-boundary surfaces.

Any tessellation (e.g. triangulation) of a manifold surface will produce edges that are of degree two, which means that all edges are shared by exactly two faces. Tessellations of non-manifold surfaces produce edges of degree 1, 2, 3, or more. A polygonal surface is a manifold (with boundary) if every edge has exactly two incident faces (except edges on the boundary which must have exactly one), and the neighbourhood of every vertex consists of a closed loop of faces (or a single fan of faces on the boundary). Figure 2-3 illustrates four kinds of vertex neighbourhoods in a polygonal model.

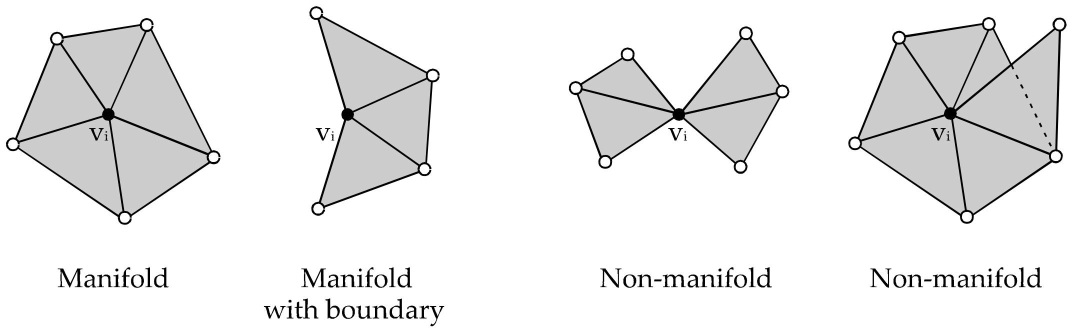


Figure 2-3: Neighbourhoods of a given vertex $\mathbf{v}_i$.

Many surfaces encountered in practice tend to be manifolds, and many surface-based algorithms require manifold input. It is possible to apply such algorithms to non-manifold surfaces by cutting the surface into manifold components and subsequently stitching them back together. However, it can be advantageous for simplification algorithms to explicitly allow non-manifold surfaces. Not only does this broaden the class of permissible input models, but it provides more flexibility during simplification. Many simplification algorithms proceed by repeatedly making local simplifications to the model. These local transformations can easily result in non-manifold regions. Consider the example shown in Figure 2-4. The same local simplification, namely edge contraction, is applied in two different ways.

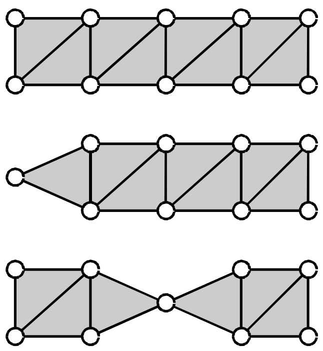


Figure 2-4: Two approximations of the same surface, both constructed by contracting a single edge (top – original, middle – manifold, bottom – non-manifold).

Depending on the choice of edge, contraction may result in either a manifold or non-manifold result. By allowing non-manifold surfaces, we allow the simplification algorithm to select the better choice based on criteria such as geometric fidelity rather than artificially limiting it to only apply operations which produce manifold surfaces.

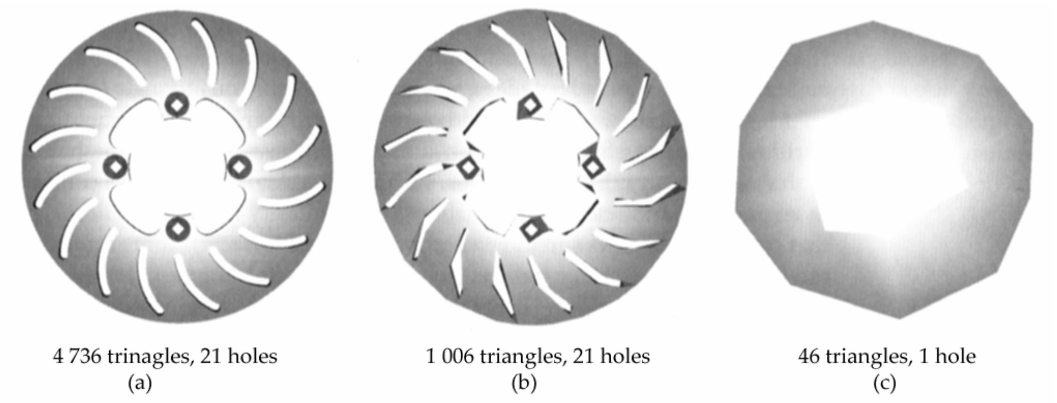


Figure 2-5: Preserving genus limits drastic simplification (taken from [36]).

This issue is of particular relevance in algorithms which seek to simplify the topology of the model. Imagine a model of a metal plate with many small holes drilled in it. The common contraction-based approach for removing a hole from this model would begin by collapsing one end of the hold into a single point, resulting in a non-manifold vertex neighbourhood. While it is possible to explicitly cut and re-stitch the surface during simplification, this can add substantial complexity to the algorithm. Figure 2-5 shows an example of two approximations with and without preserving object topology. The topology problem is more discussed in section 2.2.

2.2 Topology

A very important factor in advising particular algorithms is their relationship and behaviour regarding mesh topology. Topology studies the properties of a geometric object that remains unchanged by deformations such as bending, stretching, or squeezing but not breaking. In this concept a sphere is topologically equivalent, or homomorphic, to a cube because, without breaking them, each can be deformed into the other as if they were made of modelling clay. A sphere is not equivalent to a doughnut, because the former would have to be broken to put a hole in it. Such a definition of topology leads to the following mathematical joke [41]:

Q: What is a topologist? A: Someone who cannot distinguish between a doughnut and a coffee cup.

In this work a term topology refers to the structure of the connected polygonal mesh.

An important topological property of a surface is its Euler-Poincaré characteristic, a number which can be calculated from any polyhedral decomposition of the surface. If V is the number of points (vertices) in the decomposition, E is the number of line segments (edges), and F is the number of regions (faces), then the characteristic is given by $\chi=V-E+F$ and is the same for all possible polyhedral decomposition of the given surface. For a sphere, $\chi=2$, and the formula is identical with Euler's formula for the vertices, edges, and faces of a spherical polyhedron, to which the sphere is topologically equivalent. For a torus, $\chi=0$. The Euler-Poincaré characteristic for an orientable surface is $\chi=2-2g$, where g is called the genus of the surface.

Genus is a topologically invariant property of a surface defined as the largest number of nonintersecting simple closed curves that can be drawn on the surface without separating it [53]. Roughly speaking, it is the number of holes in a surface. For example, a sphere and a cube have a genus of zero, while a doughnut and a coffee cup have a genus of one.

Any orientable closed surface is topologically equivalent to a sphere with g handles attached to it; e.g., the torus, having $\chi=0$, is of genus 1 and is equivalent to a sphere with one handle, and a double torus (two-hole doughnut), equivalent to a sphere with two handles, is of genus 2 and has $\chi=-2$. For a nonorientable surface, $\chi=2-q$, where q is the number of cross-caps that must be added to a sphere to make it equivalent to the surface. (A cross-cap is a cap with a twist like a Möbius strip in it, see).

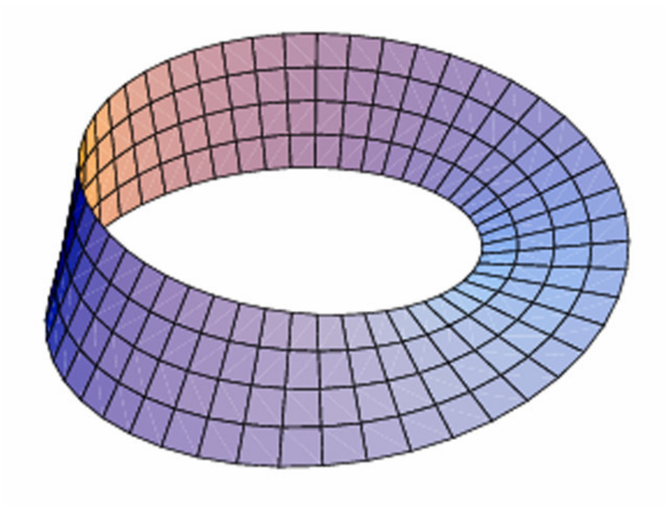


Figure 2-6: Möbius strip (taken from Wikipedia.org).

Closely related to the Euler-Poincaré characteristic is the connectivity number of a surface, which is equal to the largest number of closed cuts (or cuts connecting points on boundaries or on previous cuts) that can be made on the surface without separating it into two or more parts. The connectivity number is equal to $3-\chi$ for a closed surface and to $2-\chi$ for a surface with boundaries (e.g., a disk). A surface with a connectivity number of 1, 2, or 3 is said to be simply connected, doubly connected, or triply connected, respectively, and similarly for more complex surfaces; a sphere is simply connected, while a torus is triply connected. Thus, any surface can be classified by its boundary curves (if any), its orientability, and its Euler-Poincaré characteristic or connectivity number; and any surface is topologically equivalent to a sphere with an appropriate number of handles, cross-caps, or holes.

A surface is a simple example of a topological space, the basic entity studied in topology. The local topology of a face, edge, or vertex refers to the connectivity of that feature's immediate neighbourhood. As already said the mesh forms a 2-manifold if the local topology is everywhere equivalent to a disc, that is, if the neighbourhood of every feature consists of a connected ring of polygons forming a single surface. In a triangulated mesh displaying manifold topology, every edge is shared by exactly two triangles[2], and every triangle shares an edge with exactly three neighbouring triangles.

In simplification we recognize two approaches based on their relation to the topology: topology-preserving and topology-modifying algorithms.

Topology-preserving algorithms preserve manifold connectivity at every step. Such algorithms do not close holes in the mesh or join previously unconnected areas, and therefore preserve the overall genus. Since no holes are appearing or disappearing during simplification, the visual fidelity of the simplified object tends to be relatively good. This constraint limits the simplification possible, however, since objects of high genus cannot be simplified below a certain number of polygons without closing holes in the model (see Figure 2-7). Algorithms that preserve topology also require the initial mesh to be manifold. They either ignore non-manifold regions or faced to them simply fail.

There are plenty of areas where topology-preserving algorithms only are acceptable. For example study of tolerances in mechanical CAD requires that the topology of the models is not simplified. Similarly, in medical imaging the data collected from computer-aided tomography (CT) or magnetic resonance imaging (MRI) scans often have important topological structures that are better left in the data, rather than simplified away.

Topology-modifying algorithms have no limitations preserving a manifold topology and therefore can close up holes in the model and join separate elements such as vertices, triangles or even whole objects during simplification process, permitting drastic simplification beyond the scope of topology-

[2] A 2D manifold with boundary permits boundary edges, which belong to only one triangle.

preserving schemes. Most topology-modifying algorithms do not require valid topology in the initial mesh, which greatly increases their utility in real-world CAD applications.

Since topological simplification refers the capability of gradually simplifying any given mesh to a simplest surface with decreasing genus, it is often used for interactive visualization applications. In Figure 2-7 we can see several topology-preserving levels of detail of the brake rotor created by the simplification envelopes approach [6]. The closest rotor has 4700 triangles and the farthest rotor has about 1000 triangles. Most of the triangles in the farthest rotor are used for representing the 21 holes in the rotor even though barely one hole is visible. For this example, if the topology for the farthest rotor were simplified to a single hole, it will permit a much more aggressive geometric simplification without sacrificing visual realism (see Figure 2-5). Topology simplifications of sub-pixel holes may also help reduce aliasing artefacts, effectively removing details that will be undersampled.

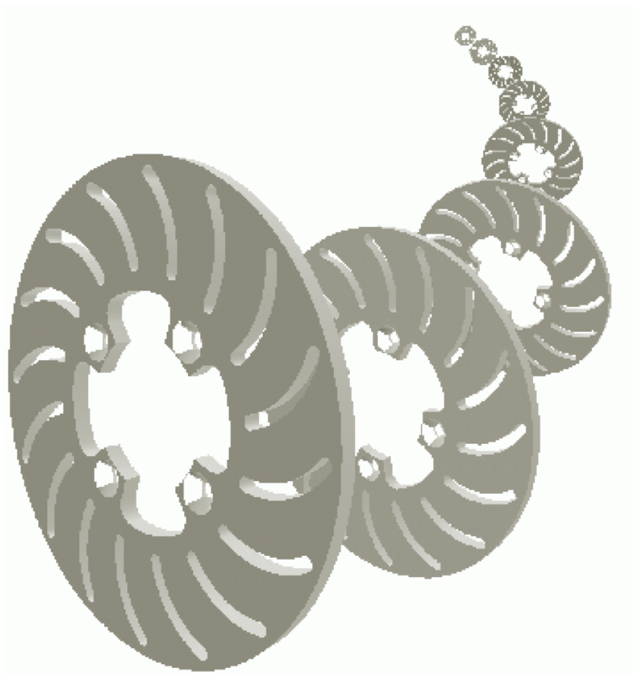


Figure 2-7: A level-of-detail hierarchy for the rotor from a brake assembly (taken from [6]).

In general, algorithms preserving topology are the most suitable when visual fidelity is crucial, or with an application such as finite element analysis, in which surface topology can affect results. Preserving topology also simplifies some applications, such as multiresolution surface editing, which require a correspondence between high- and low-detail representations of an object. When drastic simplification is required, topology-modifying algorithms do the best work. On the other hand this drastic simplification often comes at the price of poor visual fidelity.

New algorithms presented in this work have been developed to preserve the original mesh as much as possible, thus they are strictly topology-preserving.

2.3 Simplification Conditions

We can also look at the simplification process from the view of desired result. In some application it is required to have a specific number of triangles at the output rather than the exact value of approximation error reached. Different algorithms thus can be divided into two sets providing either budget-based or fidelity-based simplification [36].

Budget-based simplification

As already said, in budget-based simplification the user specifies the maximum number of resulting triangles, and the algorithm attempts to minimize the error E without exceeding given constraint. Since these algorithms generates a fixed number of triangles (given by the user at the beginning of simplification process), it is appropriate for time-critical applications where a desired frame rate dictates the per-frame triangle budget. Thus, this approach is often used for applications where interactivity is paramount. Since the error E is not controllable by the end user, this approach does not guarantee visual fidelity. This implies that solving the budget-based simplification problem optimally is difficult.

Fidelity-based simplification

In fidelity-based simplification user provides a fidelity constraint that the simplified mesh must satisfy with respect to the original input mesh. The simplification algorithm then generates a simplified mesh, attempting to minimize the number of triangles while respecting the fidelity constraint. The constraint is usually specified as some measure of the difference between the simplified mesh and the input mesh, denoted by the simplification error E. This error can be measured many ways; Chapter 3 discusses various error metrics in detail. Solving this minimization problem optimally is suspected to be NP-hard. Fidelity-based simplifications are typically best suited for applications in which visual fidelity is more important than interactivity.

The algorithms introduced in Chapter 5 and Chapter 7 fall into the category of budget-based simplification. As presented before, the main requirements motivated this work were capability of processing large datasets in critical time demands. Comparisons across different budget-based algorithms are often based on empirical observations on a few data sets that have become de facto benchmarks in the field. However, there are also measurement standards such as METRO which can be used as objective comparator (see Chapter 3).

2.4 Operation

This section provides a brief overview of various mesh simplification algorithms. We describe mesh simplification as an optimization process to be achieved by the application of local and global mesh simplification operators. Local operators simplify the geometry and connectivity in a local region of the mesh, reducing the number of polygons, while global operators operate over much larger regions and help simplify the mesh topology.

Local simplification operator

In this section we discuss the various low-level local operators that have been used for simplification of meshes. Each of these operators reduces the complexity of a mesh by some small amount.

First and probably most common is a collapse operator. Depending on a context used we recognize edge collapse, vertex pair collapse, triangle collapse and cell collapse operator.

Edge collapse

This operator collapses an edge ($\mathbf{v}_i$, $\mathbf{v}_j$) to a single vertex $\mathbf{v}_n$. This causes the removal of the edge ($\mathbf{v}_i$, $\mathbf{v}_j$) as well as the triangles sharing that edge. There is also an inverse operator called a vertex split, which adds the edge and the triangles adjacent to it. Thus, the edge collapse operator simplifies a mesh and the vertex split operator adds detail to the mesh.

There are two kinds of the edge collapse operator: half-edge collapse and full-edge collapse. In the half-edge collapse (see Figure 2-8), the vertex to which the edge collapses to is one of its end points. In the more general full-edge collapse or just edge collapse the resulting vertex $\mathbf{v}_n$ has a newly computed position.

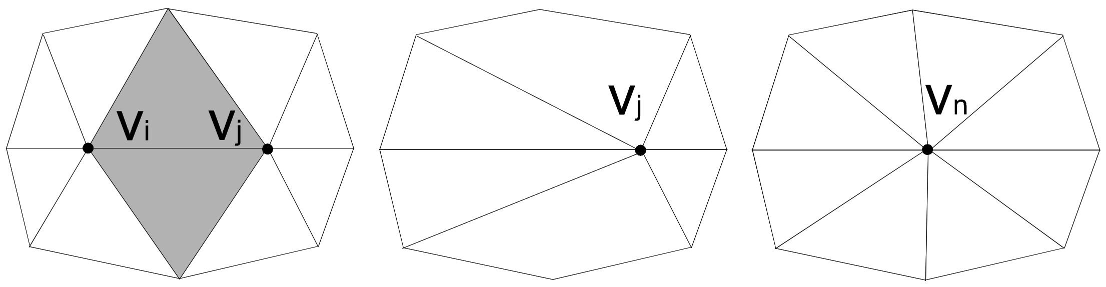


Figure 2-8: Edge collapse operator; original mesh (left), half-edge collapse (middle), full-edge collapse (right).

Although the edge collapse operator is simple to implement in some circumstances it would cause a mesh foldover or a topological inconsistency.

Mesh foldovers are an undesirable side effect of a special case of edge collapses. In Figure 2-9 you can see an example of half-edge collapse which will cause a triangle foldover. This can be detected by measuring the change in the normals of the corresponding triangles before and after an edge collapse: a mesh foldover is characterized by a large change in the angle of the normal, usually greater than 90°. Another solution of this phenomenon is described in section 5.6. Mesh foldovers result in visual artefacts, such as shading discontinuities.

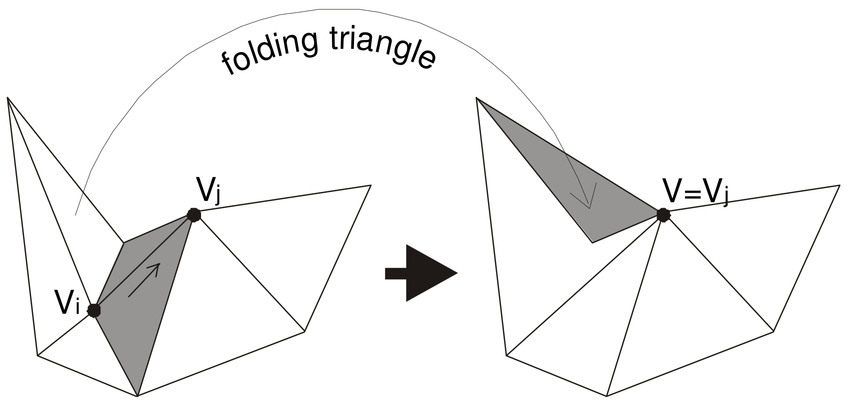


Figure 2-9: An edge contraction, which causes the mesh to fold over on itself.

If the neighbourhoods of two vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ share more than two vertices, the collapse of the edge ($\mathbf{v}_i$, $\mathbf{v}_j$) will create a nonmanifold areas in a mesh, where none has been before (see Figure 2-10). Non-manifold edges have one, three, or more adjacent triangles; since many algorithms rely on manifold connectivity, introducing such edges can create problems later in the simplification process.

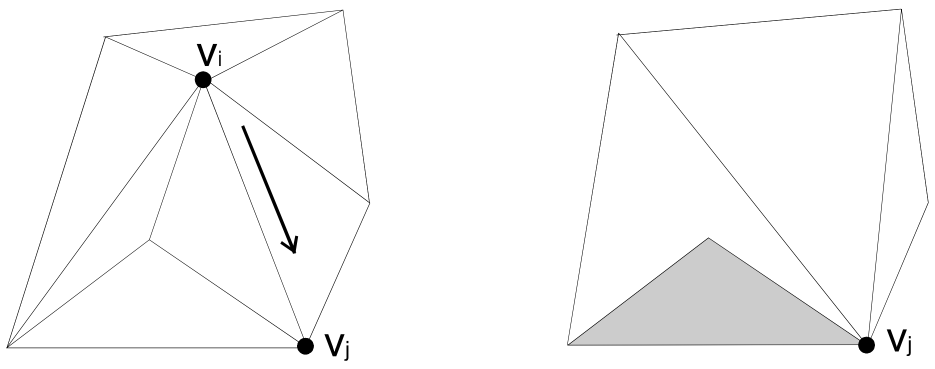


Figure 2-10: Edge collapse leading to nonmanifold mesh.

Vertex-pair collapse

A vertex-pair collapse is a special case of collapse where two unconnected vertices are joined. Since these vertices do not share an edge, no edges or triangles are removed. However, triangles surrounding collapsed vertices are updated as if the edge was present. For this reason, the vertex-pair collapse operator has also been referred to as a virtual-edge collapse. Collapsing unconnected vertices enables connection of unconnected components as well as closing of holes and tunnels, thus topology is changed. The virtual-edge collapse algorithms use some heuristic to limit the candidate virtual edges to a small number – usually virtual edge is constructed between vertices within some small distance δ.

Triangle collapse

A triangle collapse operator simplifies a mesh by collapsing a triangle $(\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k)$ to a single vertex $\mathbf{v}_n$. The vertex $\mathbf{v}_n$ can be either one of original vertices or a newly computed vertex. A triangle collapse is equivalent to two edge collapses. In general this operation requires less memory than an equivalent edge collapses, but is less fine-grained than an edge collapse.

Cell collapse

The cell collapse operator simplifies the input mesh by collapsing all the vertices in a certain volume, or cell to a single vertex. In [43] the vertices of the mesh are placed in a regular grid. All the vertices that fall in the same grid cell are then unified into a single vertex. All triangles of the original mesh that have two or three of their vertices in a single cell are either simplified to a single edge or a single vertex. Note that such a simplification does not preserve the topology and that the level of simplification depends on the resolution of the grid.

Besides collapse operators there are also removal operators used in simplification algorithms. One of the first ever used was vertex removal [46], followed by polygon merging and general region replacement.

Vertex removal
The vertex removal operator removes a vertex, along with its adjacent edges and triangles, and triangulates the resulting hole. Triangulation of the hole can be accomplished in several ways. One of these triangulations can be the same as a half-edge collapse. In this respect the vertex removal operator may be considered to be a generalization of the half-edge collapse operator. On the other hand, in general, the triangulation in 3D is non trivial optimization problem.

Polygon merging
In polygon merging nearly coplanar and adjacent polygons are merged into larger polygons, which are then triangulated. Polygon merging is more general than vertex removal since it can combine polygons (not just triangles). Since several vertices can be removed at once, it may even result in merged polygons with holes. Polygon merging as a simplification operator has been used in different applications under different names, such as superfaces or face clustering (see Figure 2-11).

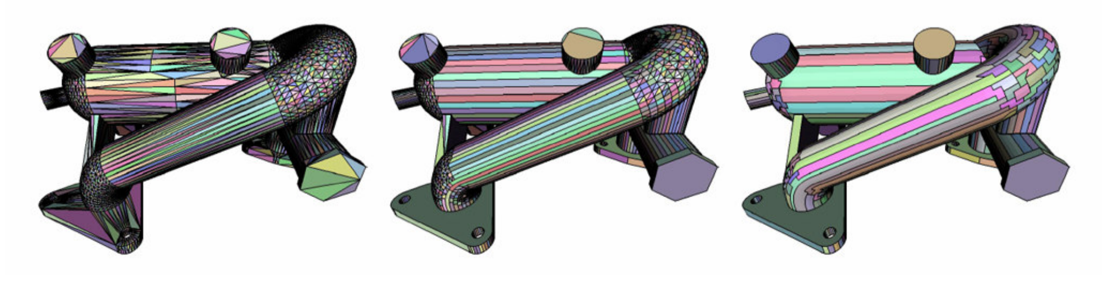


Figure 2-11: Face cluster partitions produced by iterative pair-wise merging (taken from [12]); original 11,036 clusters (left), 6000 clusters (middle), 1000 clusters (right).

If we go a step further in generalization we will get the general geometric replacement operator, which is the most general of the mesh simplification operators. It proceeds by replacing a subset of adjacent triangles by another set of (simplified) triangles, while their boundaries are the same. In addition to edge collapses and vertex removals, it can also encode an edge flip, where the common edge between two triangles is replaced by another edge that joins the two other opposite vertices. This general operator can be used to replace geometry of one primitive type with geometry of another primitive type.

The algorithms presented later in Chapter 5 will mainly gain from edge collapse operator combined with vertex removal. It is interesting to note that there is a subset of the possible edge collapse operations that are equivalent to a subset of the possible vertex removal operations. This common subset is the set of half-edge collapses, which share some of the properties of each of the other two operation types as will be shown.

Global simplification operators

The global simplification operators modify the topology of the mesh in a controlled fashion. They tend to be more complex than the local simplification operators, which only consider a small portion of the model. Before these operators can be applied, first the input model has to be voxelized. In the volumetric domain the topology modifiers are applied and the model is converted back into a triangular mesh using an iso-surface extraction method. The topology simplifying operators used in the middle stage are low-pass filtering [14] and morphological operations of dilation and erosion [37].

Low-Pass Filtering

Ten years ago first algorithms were introduced to simplify the topology of an input model in the volumetric domain. The input model is first converted to a volumetric data set. The straightforward approach is to build a spatial grid over the model and estimate a voxel density in each cell as a value of inclusion of the object in a cell. This enables data sets derived from polygonal meshes to be treated in the same manner as native volumetric data sets such as CT or MRI scans.

After that a low-pass filter to each of the grid values of the volumetric buffer is applied. Low-pass filtering eliminates fine details in the volumetric model, including topological features such as small holes and tunnels. This is followed by iso-surface reconstruction using a method such *marching cubes* algorithm [34]. Figure 2-12 shows the results of volume-domain topology simplification.

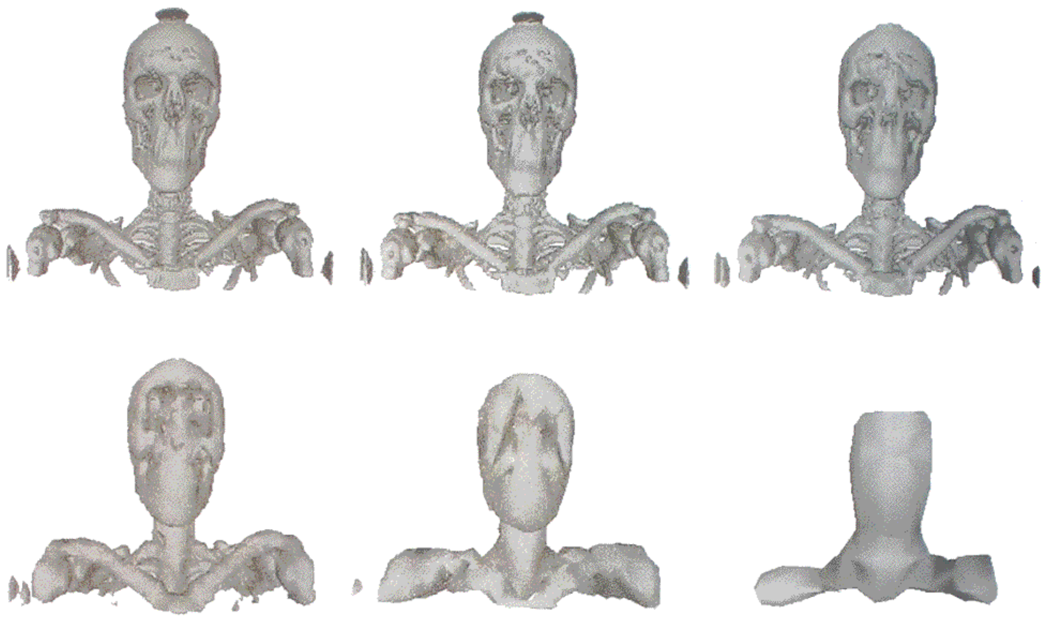


Figure 2-12: Medical iso-surface using volume-domain topology simplification (taken from [14]).

Morphological Operators

The basic idea behind using morphological operators is the voxelization of the input model into a volumetric data set using a parity-count scheme for a single component polygonal model, or a ray stabbing method for polygonal models with multiple intersecting components. In volumetric domain a distance field is built and erosion and dilatation are applied on it (see Figure 2-13). The distance field contains a distance of each inside/outside voxel to the nearest outside/inside voxel. In the dilatation operator with threshold T, every outside voxel with a distance value less than T is reclassified as being inside. Thus this process enlarges the object, fills the holes and connects unconnected components that lie within the distance T. The erosion operator is the complement of the dilation operator and ends up shrinking the object. A dilation operator followed by an erosion operator will result in a topologically simplified object with the same dimensions as the original.

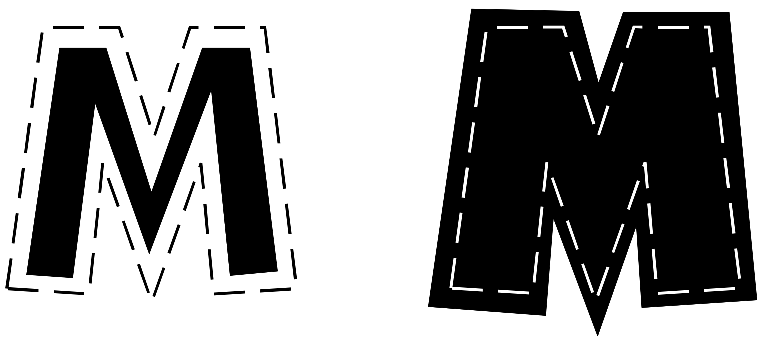


Figure 2-13: Erosion and dilatation in 2D.

After such simplification process in volumetric domain, the iso-surface is extracted using marching cubes algorithm. Resulting surface is then simplified using any above mentioned approach for geometry simplification. The advantage of this technique is that the dilatation and erosion operators are very precise and can be used to control very finely the level of topological simplification of the object.

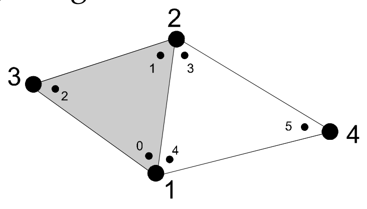
2.5 Data Structure

Just before we introduce several state-of-the-art algorithms a note about data structure should be mentioned here. To construct a polygonal mesh it is needed to have a data structure consisting of at least two pieces of information: the geometry, or coordinates, of vertices, and the definition of each triangle in terms of its three vertices. Since ordered lists of triangles surrounding a vertex are frequently required, it is desirable to maintain a list of the triangles adjacent to each vertex.

Although data structures such as a radial or a winged edge can represent this information, many implementations use a space-efficient vertex-triangle hierarchical ring structure [46]. This data structure contains hierarchical pointers from the triangles down to the vertices, and pointers from the vertices back up to the triangles sharing the vertex. These pointers essentially form a ring relationship. There are usually three lists: a list of vertex coordinates a list of triangle definitions and another list of lists of triangles containing each vertex. Edges are not explicitly defined but they can be defined as ordered vertex pairs in the triangle definition.

Another original approach is so called corner-table proposed by [42]. It is a simple data structure consisting of nothing more than two arrays of integers (the **V** and **O** tables). Similar to other structures, vertices are identified using positive integers and their location is stored in an array called **G** for "geometry". **V** and **O** have 3 times as many entries as there are triangles and hold the integer references to vertices and to opposite corners. In real implementation the object oriented approach is suggested to use. A corner *c* is the association of a triangle *c.t* with one of its bounding vertices *c.v*. The entries in **V** and **O** are consecutive for the 3 corners (*c.p==c.n.n, c, c.n*) of each triangle. Thus, *c.t* returns the integer division of *c* by 3 and the corner-triangle relation needs not be stored explicitly. For example, when *c* is 4, *c.t* is 1 and thus *c* is a corner of the second triangle. Figure 2-14 illustrates the example.

c	**V**	**O**
triangle 0 corner 0	1	7
triangle 0 corner 1	2	8
triangle 0 corner 2	3	5
triangle 1 corner 3	2	9
triangle 1 corner 4	1	6
triangle 1 corner 5	4	2



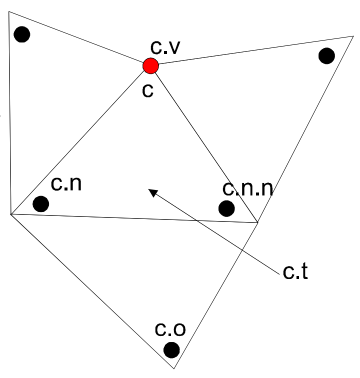


Figure 2-14: Vertices and corners in corner-table scheme.

The main advantage of such data structure is quite fast evaluation of triangle neighbours and it is extensible for edges as well. However it is not very practical for traversing the mesh or search for vertex neighbours. It also has some limitations like the need of oriented triangles or enclosed surfaces.

In our algorithms we use a modified ring structure. Figure 2-15 shows how information is stored.

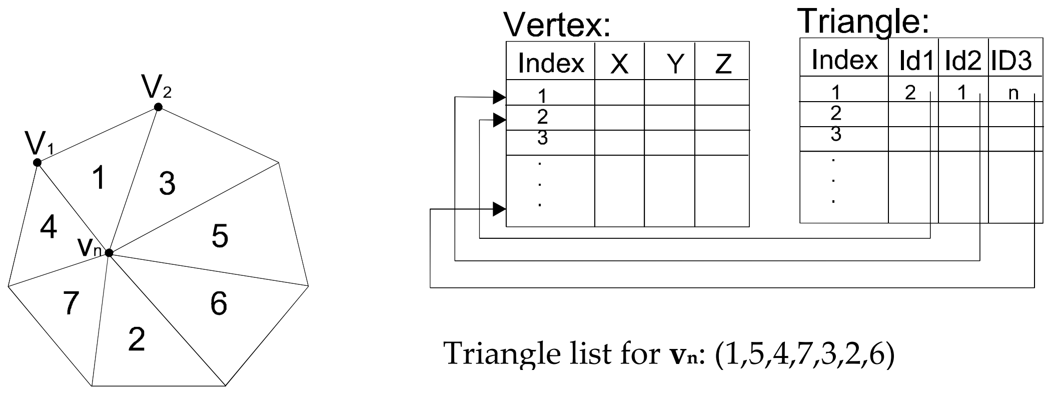


Figure 2-15: Scheme of data structure used.

We have list of vertex coordinates, where each vertex has its own unique id. The list of triangles consists again from triangle id and three indices of its corner vertices. In addition to vertex coordinates in vertex list we store there also a list of neighbouring triangles, which facilitates vertex importance evaluation and whole simplification process.

Chapter 3

Error Estimation

So far we have studied approaches to mesh simplification but we did not mention a crucial question – the measurement of the output quality. The way we measure error both during and after the simplification process can have a dramatic impact on the visual appeal and usefulness of resulting models. However, there are plenty of techniques describing how to compute the approximation quality, and each follows the specific property of the desired surface. In some applications it is important to preserve a volume or area, in others, for example, the geometric distance between the original and resulting model. Error measurement often involves complex geometrical constructs and error minimization may rely on solving nontrivial algebraic problems.

In this chapter we present some key elements of measuring simplification quality at a high level. Most simplification error metrics incorporate some form of object-space geometric error measure. Moreover, some algorithms also incorporate a measure of attribute errors such as colour, normal, and texture coordinate attributes. These geometric and attribute errors may be combined in a number of ways during the simplification process and later in the run-time rendering system. Several of the published algorithms will be referred to see the range of possible approaches to the problem.

3.1 Fidelity Metrics

As already mentioned, many simplification algorithms work on base of nested optimizations. The outer optimization process finds the best available simplification operation to perform, while the inner optimization makes each operation work as well as possible. The best available operation is usually defined with respect to some simplification error metric. The better the metric is the better choices are made during simplification process. Since optimizing an error metric in the inner optimization process improves the quality of performed outer operation, a consistent, quantitative error metric is useful for both of these optimization problems.

There are also situations when we want to know the exact quality of final approximation, separately metric used to guide the simplification process. For example if we want to compare results of several different algorithms, or just measure the final error of budget-based approaches. In such cases a general tool like METRO is reasonable (see section 3.7).

3.2 Similarity of Appearance

Although, most of simplification methods rely on purely geometrical basis, the similarity of appearance is often the crucial requirement and needs to be properly defined.

An appearance of model M regarding observer conditions ξ *(viewpoint)* is determined by raster image I_ξ, produced by a renderer. Assuming that and having given conditionsξ, we can consider models M_1 and M_2 as identical if their images I_1^ξ and I_2^ξ are alike.

All common error measures quantify the error present in input image making use of residue, that is, the input image subtracted from the original. In such way the mean square error averages the square of pixel differences:

$$mse^\xi = \left\| I_1^\xi - I_2^\xi \right\| = \frac{1}{K \cdot L} \sum_{x=1}^{K} \sum_{y=1}^{L} \left\| I_1^\xi(x,y) - I_2^\xi(x,y) \right\|^2 \tag{3.1}$$

where *K* and *L* defines a size of image raster and $\left\| I_1^\xi(x,y) - I_2^\xi(x,y) \right\|$ is Euclidean distance (see section 3.3) between two RGB values (represented as they were vectors in 3D space) at I_1(x,y) a I_2(x,*y)*.

Taking the square root is one way of reducing the range of values. Using equation (3.2), large pixel errors have a greater contribution in the error.

$$rmse = \sqrt{\frac{1}{K \cdot L} \sum_{x=1}^{K} \sum_{y=1}^{L} \left\| I_1(x,y) - I_2(x,y) \right\|^2} \tag{3.2}$$

The better M_2 approximates M_1 for given ξ, the smaller the values *mse* or *rmse* is. Having such formula we can measure a similarity of two models using integration of $\left\| I_1^\xi - I_2^\xi \right\|$ over all ξ. We assume the finite number of viewpointsξ.

The main advantage of this approach is the fact that we measure the real fidelity of resulting image, which is what we desire. On the other hand such measure is closely coupled with the viewpoints ξ, which must be identical for original and resulting model. Also, the more viewpoint we have the precise error measurement we get, which makes the comparison computationally very expensive. However, this approach is used by several simplification algorithms which are discussed in section 4.10

3.3 Geometric Error

Simplification of a polygonal mesh reduces the number of vertices and therefore changes the shape of the surface as a result. Measuring and minimizing a 3D geometric error as we perform the simplification allows us to preserve the original shape as best we can.

Euclidean geometry defines a measure of the distance between two points. For two points $\mathbf{p}_1$= (x_1,y_1,z_1) and $\mathbf{p}_2$= (x_2,y_2,z_2), the distance d between them is

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{3.3}$$

However, finding the distance between two surfaces is more involved than presented in previous section. We can think of each surface as an infinite set of infinitesimal points. Then conceptually, finding the distance between the two surfaces involves matching up pairs of points, computing their distances, and tabulating all the results. In practice, we may exploit the polygonal nature of our surfaces to allow us to use small, finite sets of points, or we can compute conservative bounds or estimates on these distances rather than taking many point-wise measurements.

The **Hausdorff distance** is a well-known concept from topology, used in image processing, surface modelling, and a variety of other application areas. The Hausdorff distance is defined on point sets, but because a surface may be described as a form of a continuous point set, it applies to surfaces as well. The Hausdorff distance is the maximum of minimum distances between points in the two point sets (M_1 and M_2). In other words, for every point in set M_1, the closest point in set M_2 is found, and vice versa. Following formula expresses the definition above:

$$H(M_1, M_2) = \max(h(M_1, M_2), h(M_2, M_1)) \tag{3.4}$$

where

$$h(M_1, M_2) = \max_{v \in M_1} \min_{w \in M_2} \|v - w\| \tag{3.5}$$

The function *h(M1, M2)* is called one-sided Hausdorff distance and finds for each point in *M_1* the closest point in *M_2* and takes the maximum. This function is not symmetric. Every point in M_1 is paired with a single point in M_2, but there may be unpaired (and multiply paired) points in *M_2*. Thus, *$h(M_1,M_2) \neq h(M_2,M_1)$*. The two-sided Hausdorff distance (or just Hausdorff distance) is constructed to be symmetric by considering both of the one-sided Hausdorff distances and reporting the maximum. This is illustrated in Figure 3.1, where *$d_{12}=h(M_1,M_2)$ and $d_{21}=h(M_2,M_1)$*. The two-sided Hausdorff distance is then:

$$H(M_1, M_2) = \max(\|d_{12}\|, \|d_{21}\|) = \|d_{12}\| \tag{3.6}$$

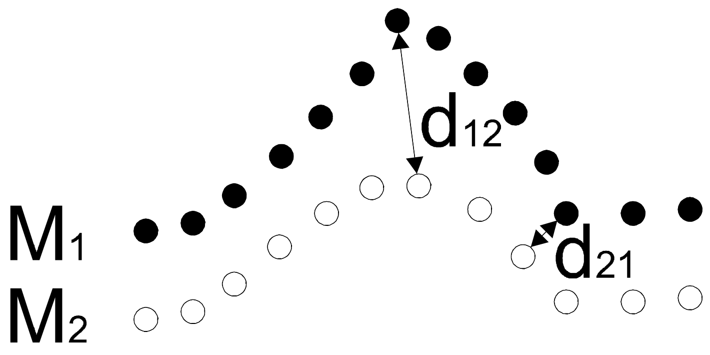


Figure 3-1: The Hausdorff distance between two surfaces M_1 and M_2.

Although Hausdorff distance is the tightest possible bound on the maximum distance between two surfaces, it has some shortcomings for polygon simplification. The problem is that it does not provide a correspondence mapping between the surfaces, which make it difficult to carry attribute values from the original surface to the simplified surfaces in a continuous fashion.

As an alternative to the Hausdorff distance a continuous bijection (one-to-one and onto mapping) between the two surfaces is considered, and the distance with respect to this mapping is measured [36].

Given such a continuous mapping $F : A \to B$, we define mapping distance

$$D(F) = \max_{a \in A} \| a - F(a) \| \tag{3.7}$$

Thus *D* is the distance between corresponding points in *A* and *B*, where the correspondence is defined by the mapping function *F*. If this mapping is accomplished via correspondences in a 2D parametric domain, such as a texture map, we call this a parametric distance, which can be expressed as

$$D(F) = \max_{x \in P} \left\| F_{i-1}^{-1}(x) - F_i^{-1}(x) \right\| \tag{3.8}$$

where **x** is a point in the 2D parametric domain and each of the F^{-1} functions maps this 2D point onto a 3D mesh, either before or after a particular simplification operation.

Because there can be a lot of such mappings and therefore many possible mapping distances, we simply search for minimum possible mapping distance as shows equation (3.9).

$$D_{\min} = \min_{F \in S} D(F) \tag{3.9}$$

Here *S* is the set of all such continuous mapping functions. Note that although D_{min} and its associated mapping function may be impossible to explicitly compute, any continuous mapping function *F* provides an upper bound on D_{min} as well as on the Hausdorff distance. If our simplification goal is to provide a guaranteed bound on the maximum error, any such function will accomplish this. However, if the bound is very loose, we might use much more polygons than necessary to provide a specified quality. Similarly, if our goal is to optimize the quality for a fixed polygon budget we would like the bound to be as tight as possible to ensure the best real quality in our rendered scene.

As specified above, both the Hausdorff distance (often presented as E_{max}) and the general mapping distance are evaluated as s the error compute the final distance between two surfaces as the maximum of all the point-wise distances. Although it is a valid choice, there are other reasonable choices as well. In some cases we are interested more in average or mean square distance rather than taking the maximum of the point-wise.

The average error E_{avg} is defined as follows:

$$E_{avg}(M_1, M_2) = \frac{1}{w_1}\int_{v\in M_1} d_v^2(M_2) + \frac{1}{w_2}\int_{v\in M_2} d_v^2(M_1) \tag{3.10}$$

where w_1, w_2 are the surface areas of M_1, M_2. In practice we approximate this exact metric using discrete set of points X_1, X_2. These sets should contain at least all the vertices of their models M_1, M_2.

$$E_{avg}(M_1, M_2) = \frac{1}{k_1 + k_2}\left(\sum_{v\in X_1} d_v^2(M_2) + \sum_{v\in X_2} d_v^2(M_1)\right) \tag{3.11}$$

where M_1 and M_2 are original and reduced model and k_1 and k_2 are numbers of vertices on each model. The distance $d_v(M)$ is defined as $\min_{w\in P(M)}\|v-w\|$.

The maximum error is often referred to a guaranteed error bound. For some applications such as medical and scientific visualizations is quite desirable to know that error is never higher than some specified tolerance.

The average error, on the other hand, can be an indication of the error across the entire surface as opposed to a few particularly bad locations. The maximum error can be even ten times larger than the average error for a particular model. This observation implies that algorithms minimizing the maximum error may ignore large increases in the average error. Similarly, heuristics that focus on minimalization the average error may introduce several regions with extremely high maximum error values.

3.4 Attribute Error

Today's polygonal models consist not only of vertex coordinates but also other attributes such as normals, colours or even texture coordinates. These attributes may be specified on faces or vertices. This allows vertices to have multiple attribute values, describing attribute discontinuities across adjacent faces when desired. For example, vertices on sharp edges can have two normals.

The earlier algorithms took exclusively geometric input and had o support for these attributes. Properties such as normals were computed as a cross product of the triangle edges if flat or smooth shading was desired.

Some present algorithms carry the attribute values and also to measure the attribute error incurred by the simplification operation. This allows to actually reduce the attribute error for a given simplification operation as part of the inner optimization by carefully choosing the attribute values for the newly created or modified vertices.

The natural space for **normals** is on the Gaussian sphere (a unit sphere with centre at the origin), on which each point represents a normal vector. The proper measure for distance between two normal vectors is an angular distance:

$$d_n = \arccos\left(\left(n_{1x}, n_{1y}, n_{1z}\right) \cdot \left(n_{2x}, n_{2y}, n_{2z}\right)\right) \tag{3.12}$$

Normals optimization us often used to prevent foldovers, comparing normal of triangle before and after proposed operation. If the angle between normals is greater than some given threshold, the operation may be disallowed. It is also possible to minimize the normal error by choosing the best normal vector at newly created vertices. A common approach is to optimize the normals as if they were in a standard Euclidean space, considering them as standard 3D points. However, optimizing normals in a Euclidean normal space generally requires renormalizing the resulting normal, which projects it back onto the Gaussian sphere.

The **colours** in computer graphics are usually stored as a triple of [R,G,B] in the range [0,1] for each value. The most straightforward way to measure colour error is to treat the RGB space as a Euclidean space and compute the RGB distance between corresponding points as

$$d_c = \sqrt{\left(r_1 - r_2\right)^2 + \left(g_1 - g_2\right)^2 + \left(b_1 - b_2\right)^2} \tag{3.13}$$

The RGB values are treated as three independent values and are optimized separately. A frequently ignored problem with this approach is that this RGB space is perceptually nonlinear. That means that equal distances between different portions of RGB appear to the human eye as different distances. The solution is to evaluate the error in some more perceptually linear space, such as CIE-L*u*v*. Another problem to solve is the case when a colour value of resulting exceeds [0,1] range. The colour component is usually set to one of the border values.

Texture coordinates are represented as *(u,v)* coordinate pairs that define a mapping of vertices to points in a 2D texture space. Analogously to the colour space, the texture space uses values in the range [0,1]. Thus, similar issues to the colour optimization have to be solved. However, unlike the colour space, the *u* and *v* values should not be considered as totally independent. The texture coordinates are intended to describe a bijection between the polygonal surface and the texture space, thus the simplification process should avoid creating folds in the texture space. Such folds would cause that the same texture space location is mapped to multiple locations on the surface.

Another kind of attributes which are not included in input models, but are of particular importance are **volume** and surface **area** of the model. Many algorithms preserve these attributes as the main optimization principle and the overall error bound is determined by the amount of volume or surface area changed. Although, such metrics can improve a lot resulting approximation, the computation cost is also noticeable. Therefore these optimizations are used mostly as a part of the inner optimization process, while in the outer another metrics is used.

3.5 Combining Errors

As presented above, a simplification algorithm can measure not only the geometry error, but can consider other attribute errors during the process of evaluation of a potential simplification operation. This is also the case of our algorithm. The geometry error is used in outer optimization process (namely vertex-to-plane distance, see section 3.6), however in inner process of vertex removal the impact on normal vectors and area of resulting mesh guides the simplification.

The majority of simplification algorithms are iterative simplifying processes. Triangles that are simplified in further steps can already be a result of some previous simplification operation. This leads to an important consideration whether to measure and optimize the total error (with respect to initial state) or just increment some error value while triangles are modified during the simplification (error value then reflect only the difference of the error before and after some operation is applied).

In many cases, the incremental error is more efficient to measure and optimize. However, the total error is the more useful measure as part of the simplification output.

3.6 Measurement Methods

Regarding previous section we can divide algorithms into categories according to the method they use to measure the error. Those categories are vertex-to-vertex, vertex-to-plane, vertex-to-surface, surface-to-surface approaches. Finally, we discuss the use of image-space metrics as opposed to object- and attribute-space metrics. The algorithms referenced in this section will be described more in detail in Chapter 4.

Vertex-to-vertex distance

The most straightforward approach to measure the approximation error is based on measuring distances between the original vertices and vertices of simplified model. The quality of such results is determined by the choice of correspondences of vertices. It is easy to imagine that few vertices on the same position in space can define different surface and therefore this method will be useful only on some particular cases. For example vertex merging operations such as edge collapse or cell collapse where clustering of two or more vertices gives exact mapping between original and resulting meshes (see Figure 3-2).

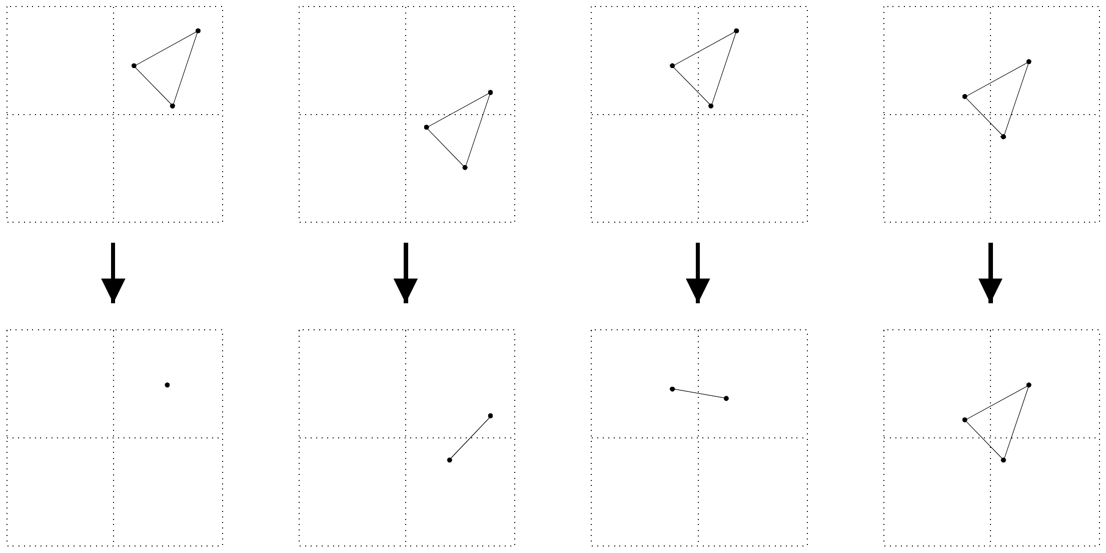


Figure 3-2: Several examples of vertex mapping while clustering one triangle.

In algorithms from grid-based vertex clustering family the maximal error bound is given by the diagonal of one cell. The floating-cell approach proposed by [35] does not use a regular grid for clustering. Instead of choosing vertices according to the cells, the cells are chosen to surround so called *representative* vertices (see Chapter 4 for more details). These cells are cubes or spheres of radius corresponding to vertex grade. During simplification all vertices in given cell are merged to chosen vertex and therefore the error bound is determined by the cell radius (and is halved compared to regular grid approach while supporting similar number of resulting vertices).

Vertex-to-plane distance

Even more computationally efficient than distance between two points is the distance between a point and a plane. Given a plane with unit normal **n**, and signed distance from the origin *D*, the shortest distance from point **p** = (*x,y,z*) to the plane is

$$d = n \cdot p + D = n_x x + n_y y + n_z z + D \tag{3.14}$$

Since the models are composed of planar polygons rather than infinite planes, the vertex-to-plane distance methods do not really provide a bound on the maximum or average distance between models. Measuring the error of a simplified mesh thus requires another metric or a tool such as Metro (see section 3.7). However, simplification methods based on vertex-to-plane distance are fast and moderately accurate, tending to produce approximations with low error for a given polygon count.

These methods usually work on a following basis. For each simplified vertex the maximum distance between the vertex and supporting planes is measured. Each adjacent triangle defines one supporting plane. When two vertices are merged, the set of supporting planes of resulting vertex will contain supporting planes of both vertices. The error metric is defined as

$$E_v = \max_{p \in planes(v)} (p \cdot v)^2 \tag{3.15}$$

where **v** = (*x,y,z,1*) and **p** = (n_x,n_y,n_z,*D*). This measure can either overestimate or underestimate the maximum distance because the vertex distance can be different for the plane and the actual polygon.

Probably most famous mesh simplification method [10] uses *error quadrics*. It replaces the maximum of squared vertex-to-plane distances, shown above, with the sum of squared distances as follows:

$$E_v = \sum_{p \in planes(v)} (p \cdot v)^2 = \sum_p (v \cdot p)(p \cdot v) = v \left[\sum_p p \cdot p \right] v = v \sum_p Q_p v = v Q_v v \tag{3.16}$$

Q_p is called error quadric and it is 4x4 symmetric matrix. It represents a contribution to E_v by plane *p*. The contribution of all planes surrounding vertex v is computed as a sum of all Q_ps. Thus when an edge is collapsed, the resulting quadric will be the sum of quadric of original vertices. As obvious, neither storage nor computation requirements grow during simplification process.

Vertex-to-surface distance

In vertex-to-surface distance metric the vertices of the original model are mapped to their closest points on the polygons of the simplified surface. This approach is entirely appropriate for models created from a set of points (for example from 3D scanner) which are then triangulated. This leads to the thought which is also discussed in Chapter 7, namely that the vertices are only true data to be preserved rather than the input surface. Vertex-to-surface approaches are much slower than vertex-to-plane approaches because suitable mappings must be found for the many input vertices.

Hoppe's progressive mesh algorithm [20] is based on edge contraction works in following steps:

1. Choose an initial value for the position of the new vertex **v** (e.g., an edge vertex or midpoint).
2. For each original vertex, map it to the closest point on one of the simplified triangles.
3. Optimize the position of **v** to minimize the sum of squared distances between the pairs of mapped points (solve a sparse, linear least-squares problem).
4. Iterate steps 2 and 3 until the error converges.

Since vertex mapping process is a part of inner loop, the whole computation is very expensive. Also in some cases, this system will never converge, because the optimal vertex position may be at infinity.

Surface-to-surface distance

A surface-surface distance metric can provide the strongest guaranteed bounds on the error of a simplified surface. By definition, such a metric considers all points on both the original and simplified surface to determine the error at a given stage of the simplification process. These methods generally choose to minimize the maximum error, perhaps because finding a guaranteed maximum bound on the simplification error is the whole point of using such a rigorous approach. Applications for which such bounds may be especially useful include medicine and scientific visualization. Algorithms based on surface-to-surface distance metrics are for example simplification envelopes [6], plane mapping [1] or tolerance volumes [13].

Image metric

Different algorithms use different measures of the geometric error. However, often the crucial measure of fidelity is not geometric but perceptual. As already briefly mentioned in section 3.2, there exist algorithms that their simplification process guide by comparing images of original and reduced model when rendered. One of such algorithms is [33], which measures the error for simplification operations by rendering multiple images of the object using a sphere of virtual cameras. Each camera renders an image of the original model and of the simplified model, and a root-mean-squared error of pixel luminance values is computed between the two sets of pixels from all the cameras – see equation (3.1).

This approach naturally incorporates visual errors due to a number of sources, such as motion of the silhouette, and deviation of colour, normal, and texture coordinate attributes. It even accounts for factors such as the content of the texture maps and the shading modes used to render the model (such as flat or Gouraud shading).

However, image-based simplification algorithm is significantly slower than the slowest geometric algorithms, since rendering multiple images for every edge collapse is an intrinsically expensive way to measure error.

3.7 METRO

Having presented several error estimation approaches there is a question how to compare particular simplification algorithms. Since the criteria to drive the simplification process are highly differentiated and many simplification approaches do not return measures of the approximation error introduced while simplifying the mesh, a common tool to measure the approximation error has been developed. Such a tool is called Metro [5].

Metro numerically compares two triangle meshes M_1 and M_2, which describe the same surface at different levels of detail. It requires no knowledge of the simplification approach adopted to build the reduced mesh.

It evaluates the difference between the two meshes on the basis of the approximation error measure defined in the section 3.3. It adopts an approximate approach based on surface sampling and the computation of point-to-surface distances. The surface of the first (pivot) mesh is sampled, and for each elementary surface parcel it computes the distance to the not-pivot mesh. The program outputs the maximum and average distance from the first model to the second model (SGI version even provides graphical output, see

Figure 3-3). A two-sided distance function may also be computed by swapping the order of the models and running the algorithm again. The tool has been used in several research papers to report on the relative merits of various simplification algorithms and it is also referred in results sections of presented algorithms in later sections of this work.

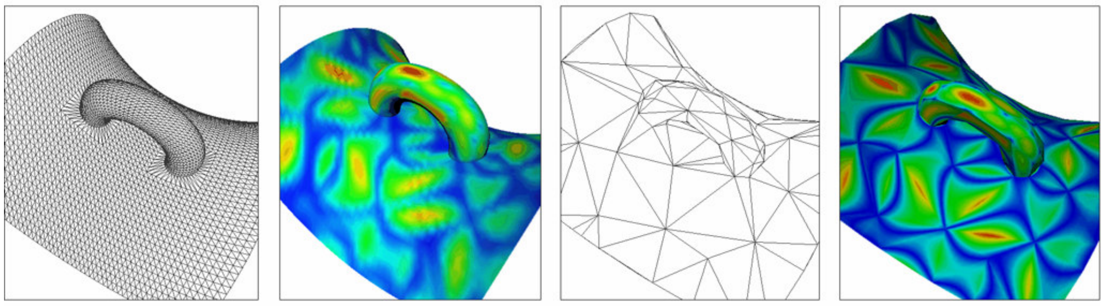


Figure 3-3: Graphical output of Metro (taken from [5]).

We have examined the motivation to measure approximation error, the key elements common to many simplification error metrics, and several particular metrics as well. These metrics has been classified into vertex-to-vertex, vertex-to-plane, vertex-to-surface, and surface-to-surface distance measures. Each class provides different characteristics in terms of speed, quality, robustness, and ease of implementation. At the end Metro has been mentioned as a standard tool for approximation error evaluation and comparison.

Chapter 4

Algorithms survey

A common application of simplification is reducing the complexity of very densely over-sampled models. Those models are often uniformly tessellated, thus a triangle density is the same in both flat and highly curved regions. It is usually preferable that local triangle density adapts to local curvature. In other words, we tend to obtain an approximation with a smaller amount of triangles in flat areas in opposite to curved parts of the surface.

Successful algorithms for simplifying curves and height fields were developed twenty years ago, but the work on more general surface simplification is much more recent [26]. Note that, since height fields are a special case of general surfaces, optionally approximating a surface is NP-hard. The traditional approach to multiresolution surface models has been manual preparation. A human designer must construct various levels of detail by hand. The general goal of the work done on surface simplification has been to automate this task.

Just before we present various simplification algorithms we present high level frameworks upon which the methods are constructed. Knowing these approaches helps to understand some fundamental concepts of particular methods. Since new algorithms presented in this work are related to vertex decimation and edge contraction methods, these two approaches are discussed more in detail in following survey.

4.1 High level frameworks

The typical simplification algorithm uses a nested optimization process: an outer optimization makes a sequence of discrete choices for which operations to perform and in what order, and an inner optimization makes choices in processing the operator, such as which way to fill the hole during a vertex removal or where to place the new vertex during an edge collapse.

We will discuss a few of the high-level queuing algorithms for choosing the ordering of the simplification operations as well as the choice of simplification operators.

Nonoptimizing

The simplest algorithms essentially apply all possible simplification operations in an arbitrary order. This is desirable in methods based on clustering, where any operation may be performed completely independently of any other. Note that such algorithms are nonoptimizing in the sense that they do not concern itself with the outer optimization problem. There is still some possibility to make optimizations during the actual application of the selected (inner) operation.

Greedy

Greedy algorithms solve the outer optimization problem according to some cost function. This function usually defines a resulting error of proposed simplification operation. At the beginning the cost of all operations is evaluated and they are sorted according to their values. The minimum cost operation is applied to the current mesh and removed from the priority queue. Since the operation may affect the cost of other operations in the neighbouring mesh, the cost of neighbouring operations is updated. The most of the time in real implementation is consumed by the evaluation of operation cost while updating the neighbours. Thus, choosing operations with fewer neighbours can significantly improve the performance. For example, vertex removal and half-edge collapse operations affect fewer triangles than the edge collapse, and thus they change the costs of fewer neighbouring operations.

Lazy

Lazy queuing algorithms attempt to reduce the number of calls to cost function in comparison to the greedy approach. The main idea is that it is not necessary to update the cost of operation in affected area every time, since it can be updated several times before it is actually used. Instead the dirty flag is used to mark the cost no longer accurate assuming that new value will be not to far off. On every operation taken from priority queue the dirty flag is checked. If it is false the operation is processed, otherwise an actual cost and reinserting it into the queue is applied.

Estimating
Another method for reducing the number of cost computations performed simply replaces the expensive computations with cheaper (faster) estimates. The priority ordering of the operations is determined entirely by estimated costs, whereas the accurate cost computation is performed only once per operation when applied. This method will work well if the ordering generated by the estimated costs is similar to the ordering that would be generated by the accurate costs.

Independent
The independent algorithms perform a maximum set of independent operations, or operations whose mutual neighbourhoods do not overlap, chosen in order of the cost function at the time. Each pass of iterative process creates one level of the simplification hierarchy. Within a pass, only operations affecting independent mesh neighbourhoods are applied, with the remaining operations placed on a list *L* for processing in a future pass. This approach leads to parallel processing and its extension is described in Chapter 5.

Interleaved simplification operators
In this approach geometry simplification alternates with topology simplification. This is motivated by the observation that each simplification stage allows the mesh to be simplified more aggressively than if only one kind of simplification had been applied. For instance, closing small holes by topology simplifications allows higher geometry simplification. The simplest scheme for both simplifications is the nonoptimizing approach. However, any of the previously mentioned approaches can be used.

We showed that simplification process can be characterized according to the optimization algorithm, simplification operator, and error metric that it uses. The particular choice depends on the constraints imposed by the target application, ease of coding, and the nature of the input data sets. In next section of this chapter we will present some state-of-the-art algorithms divided info categories already used in previous text.

4.2 Volume methods

Some work has been done on volumetric approaches to multiresolution modelling. Generally speaking, if the models in question are acquired as volumes and will be rendered as volumes, volume simplification is a good approach. However, if the simplified volumes must be converted into a polygonal form before rendering, volume methods become significantly less attractive. The simplification algorithm [37] for polygonal meshes has 4 stages:

- convert the input polygon model into a volumetric model
- application of morphological operators in the volumetric domain to simplify the topology
- iso-surface extraction to obtain a polygonal representation of the simplified surface
- topology-preserving triangle count reduction to decimate the iso-surface.

For mesh voxelization they use a parity count algorithm, which is simply the 3D extension to the parity count method of determining whether a point is interior to a polygon in 2D. Thus, a voxel V is classified by counting the number of times that the ray with its origin at the centre of V intersects polygons of the model. An odd number of intersections means that V is interior to the model, and an even number means it is outside.

The morphological operators are well suited to simplify the topology of objects because they present a clean and efficient way to remove small features, close holes and join disconnected components of a model. The first step in using morphological operators is the calculation of a distance map. Given a binary volume that is classified into feature and non-feature voxels, a distance map associates with each voxel the distance to the nearest feature voxel. Feature voxels are those that are inside the object and non-feature voxels are those that lie outside the object. Feature voxels have a distance map value of zero. The two atomic morphological operators are erosion and dilation. They take as input the volume, the distance map, and an erosion/dilation distance. For dilation, [37] look through the distance map, and any non-feature voxel that has distance less than or equal to the threshold is turned into a feature voxel. Erosion is the complement of dilation. In this case, the volume is negated (i.e. a feature voxel becomes non-feature and vice versa), the distance map is calculated and dilation performed. After this, the volume is negated again to obtain the final result. While useful by themselves, erosion and dilation are usually used in conjunction with each other. The reason is that if they are used in isolation, then they increase (dilation) or decrease (erosion) the bounds of the volume. When erosion is performed followed by dilation, it is called an opening. This is due the fact that this operation will widen holes, eliminate small features and disconnect parts of the model that are connected by thin structures. The complement of this operation is a closing, which is a dilation followed by an erosion. This will close holes and connect previously disconnected parts of the model.

To create a manifold polygonal model, [37] extracts an iso-surface from the volumetric representation of the model using the standard *marching cubes* algorithm [34].

The new iso-surface usually has simpler topology than the input model. In addition, because the Marching Cubes algorithm considers cubes in isolation, it frequently over tessellates the surface. Therefore, the number of triangles of this iso-surface can be drastically reduced without degrading the quality of the model. To achieve this end, Garland and Heckbert's polygon-based simplification method based on quadric error measurement [10] is used. This method is based on a generalized form of the edge collapse operation called vertex pair contraction, and will be described later in this chapter.

4.3 Simplification envelopes

Simplification envelopes [6] are something of a meta-method. The main feature of this algorithm is to use no error measure but only a geometric construction to control the simplification. Simplification envelopes are two surfaces constructed on each side of the original surface using a user specified offset and making sure these surfaces do not self-intersect. The space between the two surfaces is then used to build a new surface. Therefore, the only constraint is that the new polygons should not intersect with any one of the surfaces. This naturally preserves the original model topology, and guarantees a global error. In order to construct the envelopes, the original model must be an oriented manifold.

The amount of simplification is controlled by the offset used for constructing the surfaces. The case where envelope surfaces are most likely to self-intersect is along the sharp edges of the original mesh, where there will not be much room to build one of the surfaces. The surfaces which self-intersect must then be moved closer to the original mesh until the condition is verified. Thus, near sharp edges, the spaces between the two surfaces will be smaller and less simplification will be permitted. Conversely, in planar areas, the distance will be maximal, and therefore, maximal simplification will be allowed.

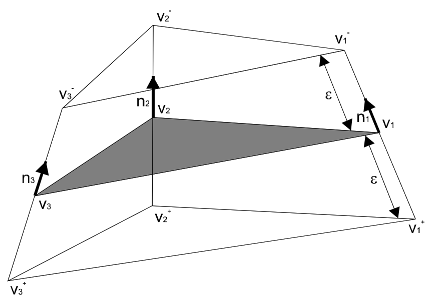


Figure 4-1: Building inner and outer envelopes for a triangle (taken from [6]).

At the beginning the envelopes have to be constructed (see Figure 4-1). The algorithm is as follows:

1. Offset the outer surface along vertex normals by a fraction of the desired final offset.
2. If, for any vertex, the surface self-intersects, cancel the move for that vertex.
3. Repeat 1 and 2 until either no further increment can be made without intersection, or the offset as reached the desired value.
4. Repeat 1 to 3 for the inner surface.
5. Repeat 1 to 3 for the border tubes. These are built along the borders of non-closed objects to allow for simplification there also.

The algorithm then goes on to generate the simplified mesh. For each vertex of the initial mesh:

1. Remove the vertex and the adjacent faces.
2. If possible, iteratively fill the hole by triangulation using as big faces as possible and ensuring that they do not intersect with the offset surfaces. Otherwise cancel the removal and process the next vertex.

This algorithm is appealing because it does not use any measure of the error. The envelopes are the only control over the simplification. However, this approach is computationally expensive, especially during the envelope construction phase.

4.4 Wavelet surfaces

Wavelet methods provide a fairly clean mathematical framework for the decomposition of a surface into a base shape plus a sequence of successively finer surface details. Approximations can be generated by discarding the least significant details. Wavelet decompositions are generally unable to resolve creases on the surface unless they fall along edges in the base mesh [20], [18].

Essentially, this requires that the surface be reconstructed using a wavelet representation. This is as usual difficult. Eck et al. [7] developed a method for constructing wavelet representations of arbitrary manifold surfaces. This is not actually a simplification algorithm. It is a pre-processor for another algorithm, which produces a multiresolution representation of a mesh, which is a compact geomorphic containing a simple base mesh and a series of wavelet coefficients that are used to introduce details in the mesh. From this representation, a new mesh can be retrieved at any required level of detail. However, it suffers from some serious drawbacks. Before the wavelet representation can be built, the surface must be remeshed so that it has subdivision connectivity. This process alone introduces error into the highest level of detail. In addition, the topology of the model must remain fixed at all levels of detail.

As already said, this method requires the input mesh to be constructed by **recursive subdivision** i.e. where each triangle is subdivided using a 4-to-1 split operator until the desired amount of detail is reached. Such a mesh is encoded into a multiresolution representation. The algorithm below describes how to convert any mesh so that it has the property of recursive subdivision.

It is an adaptive subdivision algorithm, which preserves topology but does identify any characteristic features in the mesh. Approximation error is measured using the distance to the original mesh. Harmonic maps are used at several steps to parameterize a 3D mesh into a planar triangulation. The algorithm has four main steps.

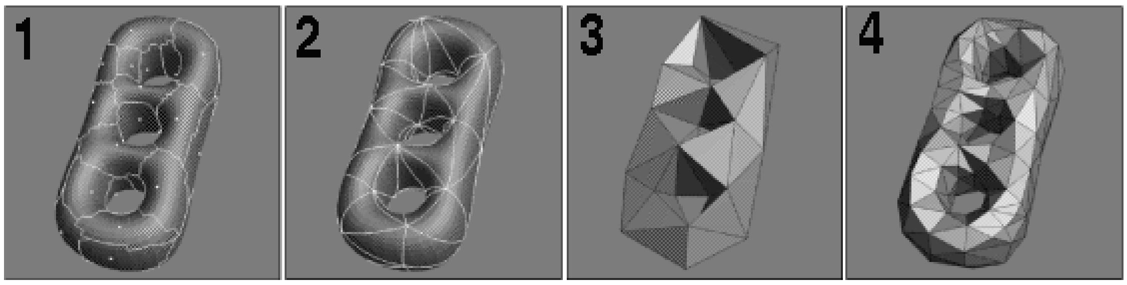


Figure 4-2: Four steps in the MRA algorithm (taken from [17]).

- Partitioning. A Voronoi-like diagram is constructed on the original mesh (Figure 4-2/1) using a multi-seed path finding algorithm in the dual graph of the mesh (where the nodes are the faces of the mesh and the arcs represent adjacency and are weighed using the distance between the centres of adjacent faces). This diagram is then triangulated using a Delaunay-like method and the harmonic maps to straighten the edges (Figure 4-2/2).

- Parameterization. The result is a base mesh (Figure 4-2/3) that is parameterized using a harmonic map. The parameterization is forced to be continuous across the faces so that the number of wavelet coefficients is minimal.
- Re-sampling. The base mesh is now re-sampled using the 4-to-1 split operator until the mesh is at a certain distance from the original mesh (Figure 4-2/4). Each step is parameterized as in previous step.
- Multiresolution Analysis. The resulting succession of meshes is passed to the multiresolution analysis algorithm to be encoded using wavelets.

Like other subdivision-based schemes, wavelet methods cannot easily construct approximations with a topology different from the original surface. The wavelet representation is unable to adequately preserve sharp corners and other discontinuities on the surface. Wavelet methods cannot change the topology and are capable of reducing smooth manifolds only. They produce a wide range of simplification and details can be added in specific parts of the mesh. But it is also computationally very expensive. Furthermore, extracting a valid mesh from wavelet-based representation is also expensive.

4.5 Vertex Clustering

Vertex clustering methods spatially partition the vertex set into a set of clusters and unify all vertices within the same cluster [43], [35]. They are generally very fast and work on arbitrary collections of triangles. Unfortunately, they can often produce relatively poor quality approximations.

The simplest clustering method is the uniform vertex clustering shown in Figure 3. The vertex set is partitioned by subdividing a bounding box on a regular grid, and the new representative vertex for each cell is computed using cheap heuristics (i.e. the average of vertices coordinates). This process can be implemented quite efficiently. The algorithm also tends to make substantial alternations to the topology of the original model.

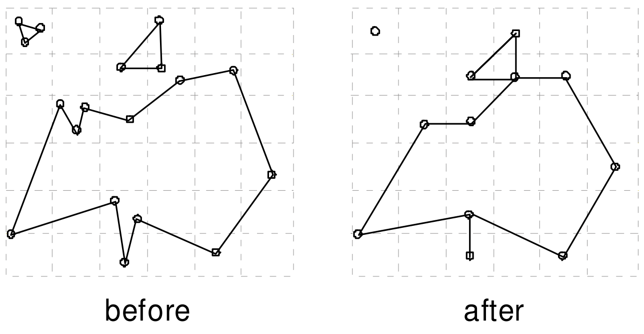


Figure 4-3: Uniform clustering in two dimensions.

Note that, the results of this algorithm can be quite sensitive to the actual placement of the grid cells. It is also incapable of simplifying features larger than cell size. A planar rectangle consisting of many triangles all larger than the cell size will not be simplified at all, even though it can be approximated using two triangles without error. The most natural scheme is to use an adaptive partitioning scheme such as octrees. Centring cells on important vertices, can also improve approximations.

Clustering methods tend to work well if the original model is highly over-sampled and the required degree of simplification is not too great. They also tend to perform better when the surface triangles are smaller than the cell size. Since no vertex moves further than the diameter of its cell, clustering algorithms provide guaranteed bounds on the Hausdorff approximation error sampled at the vertices of the original model and its approximation. However, to achieve substantial simplification, the required cell size increases quite rapidly, making the error bound rather weak. In particular, at more aggressive simplification levels, the quality of the resulting approximations can quickly degrade. Vertex clustering methods are fast and general. On the other hand the simplification process itself is hard to control and gives poor quality approximations.

4.6 Region Merging

A handful of simplification algorithms operate by merging surface regions together [16]. In general, these algorithms usually partition the surface into disjoint connected regions based on planarity assumption. At the beginning a planarity threshold has to be set. For each triangle its neighbourhood is checked and if it is sufficiently planar, the triangles are merged together. After such merges the boundary of each region is simplified, and the resulting region (loop) is triangulated. These algorithms are restricted to manifold surfaces, and do not alter the topology of the model. Region merging techniques produce good quality results, and they provide bounds on the approximation error. However, the implementation of such algorithms is more complicated in comparison to others without any superior approximations.

In some ways different, but in principle the same approach is proposed by Garland et al. [12]. Their method is derived from the edge contraction method. They create a dual graph of the surface, and instead of edge collapse in the original surface they perform face clustering in the dual representation using dual quadric metrics [10], [12], [15], [19], see Figure 4-4.

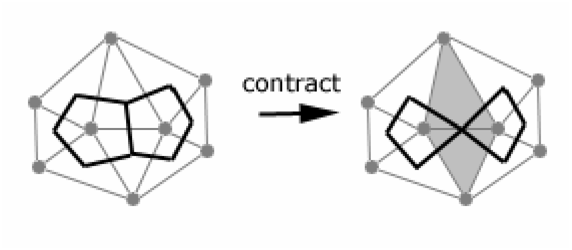


Figure 4-4: Edge contraction in the dual graph. The two faces of the surface corresponding to the endpoints of the dual edge are merged to form a single face cluster.

At the end the clusters of triangles are re-triangulated. In opposite to region merging algorithms, the dual quadric metrics seek to minimize the average deviation without any guaranteed bounds on the maximum. The quadric metric will be described later in detail. As [44] shows, clustering methods seem to be a promising approach for the simplification of massive meshes.

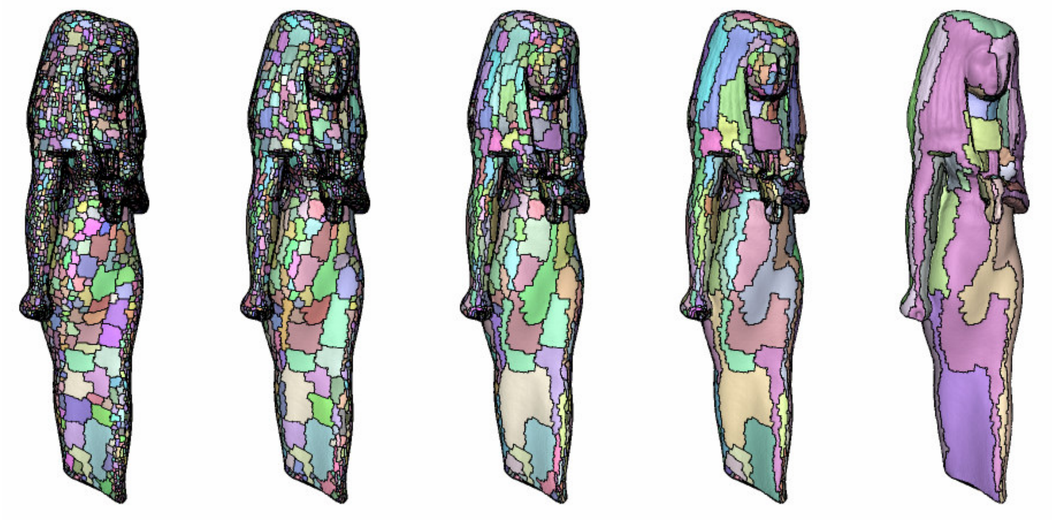


Figure 4-5: Face clusters computed for Isis statue, composed of 375,736 triangles. Cluster shape adapts to surface shape. From left 10,000 clusters, 5000 clusters, 2500 clusters, 1000 clusters and 100 clusters.

4.7 Vertex decimation

Decimation methods are algorithms that start with a polygonization (typically triangulation) and successively simplify it until the desired level of approximation is achieved. The advantage of decimation methods is that they can be generalized to volumes [39].

One of the more widely used algorithms is vertex decimation, an iterative simplification algorithm originally proposed by Schroeder et al. [46]. In each step of the decimation process, a vertex is selected for removal, all the faces adjacent to that vertex are removed from the model and the resulting hole is re-triangulated. Since the re-triangulation usually requires a projection of the local surfaces onto a plane, these algorithms are generally limited to manifold surfaces. The fundamental operation of vertex deletion is also incapable of simplifying the topology of the model. Schroeder [45] was able to lift these restrictions by incorporating cutting and stitching operations into the simplification process.

1. The three steps of the algorithm are:
2. Characterize the local vertex geometry and topology.
3. Evaluate the decimation criteria.
4. Triangulate the resulting hole.

Now let us have a look at the algorithm in more detail. All the vertices in a mesh are initially evaluated according to their importance in the mesh. There are 5 fundamental vertex types recognized by their topology, see Figure 5-2. Only three types of vertices are usually used in the simplification process; simple, boundary and interior edge vertices.

The vertex importance is calculated according to the vertex type. For a simple vertex the importance is equal to its distance from the average plane computed from the positions of surrounded vertices, see Figure 6a. In case of a boundary or interior edge respectively, vertex importance is equal to its distance from the bisector traversing through other two vertices on the border or important edges respectively (see Figure 4-6b).

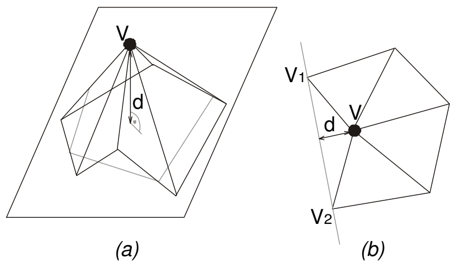


Figure 4-6: The distance between a vertex and the average plane (a) and between a vertex and the bisector (b).

An average plane is constructed using the triangle normals **n**$_i$, centres **x**$_i$ and areas A_i.

$$N = \frac{\sum n_i A_i}{\sum A_i} \quad n = \frac{N}{|N|} \quad x = \frac{\sum x_i A_i}{A_i} \tag{4.1}$$

where the summation is over all triangles in the loop. The distance of the vertex **v** to the plane is then $d = |\mathbf{n}\,(\mathbf{v}\text{-}\mathbf{x})|$. If the vertex is within the specified distance to the average plane it may be deleted. Otherwise it is retained.

Once the vertex is eliminated, the hole arising has to be triangulated. The new triangulation must be regular – no crossing triangles or triangles with vertices on the line. If such triangulation does not exist, the vertex must be preserved. It is also recommended that the new triangulation of the hole should be a good approximation of the original surface and the triangles would not be too thin and long. Schroeder et al. propose such a triangulation algorithm where a splitting plane iteratively divides the hole in two parts, until each hole has the shape of a triangle.

The original vertex decimation algorithm used a fairly conservative estimate of approximation error. When a vertex is being removed, its distance to a new surface is computed and this value is distributed then to neighbouring vertices. The vertex is deleted if its importance and the accumulated error value are under some threshold. More recent methods [24], [3] use more accurate error metrics, like the localized Hausdorff error. Klein et al. [24], [25] use one-sided Hausdorff distance computed before vertex removal. If the error value is sufficiently low, the vertex is removed, otherwise it is preserved. The algorithm of Schroeder et al. is reasonably efficient both in time and space, but it seems to have some difficulty preserving smooth surfaces [11]. The triangulation method has also been improved. Ciamplalini et al [3] use 2D triangulations: an ear cutting solution and minimum angle modification of the previous one. To be able to apply standard 2D triangulation algorithms, they must project a triangulated area onto a plane. They use 14 planes projecting the border of the hole on each of them until they find a ″valid″ projection plane (where the projection has no intersecting edges). Klein et al. [23], [25] have tested several triangulation methods and found that the optimal triangulation method depends on the model being reduced. For example Delaunay triangulation turned out to be unsuitable for models such as a coke can. Lee et al. [29] recently described an algorithm, which establishes smooth parameterizations for irregular connectivity, 2-manifold triangular meshes of arbitrary topology. By applying a vertex decimation algorithm they simplify the original mesh and use piecewise linear approximations of conformal mapping to incrementally build a parameterization of the original mesh over a low face count base domain. The resulting parameterizations are of high quality and their utility is demonstrated in an adaptive, subdivision-connectivity remeshing algorithm that has guaranteed error bounds.

Vertex decimation methods [39] produce good quality results, preserve mesh topology and are generally applicable to manifolds only.

4.8 Edge Contraction

The other class of decimation techniques is based on the iterative contraction of vertex pairs (edges) [10], [44], [31], [20], [19]. These algorithms have become very popular in recent years. An iterative edge contraction (or edge collapse) takes the two endpoints of the target edge, moves them to the same position and links all the incident edges to one of the vertices, deletes the other vertex, and removes any faces that have degenerated into lines or points, see Figure 4-7. Note that the fundamental operation of contraction does not require the immediate neighbourhood to be manifold. Thus contraction-based algorithms can more conveniently deal with non-manifold surfaces than vertex decimation algorithms.

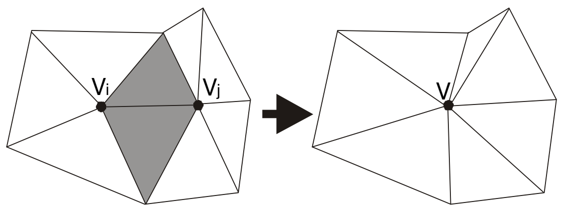


Figure 4-7: Edge ($\mathbf{v}_i$, $\mathbf{v}_j$) is contracted. The dark triangles become degenerate and are removed.

Typically, this removes two triangular faces per edge contraction. These algorithms work by iteratively contracting edges of the model. The primary difference lies in how the particular edge to be contracted is chosen and how a new vertex position is set.

The methods consist of repeatedly selecting the edge with minimum cost, collapsing this edge, and then re-evaluating the cost of edges affected by this edge collapse. The first step in the simplification process is to assign costs to all edges in the mesh, which are maintained in a priority queue. For each iteration, the edge with the lowest cost is selected and tested for candidacy. An edge is rejected as a candidate if no solution exists for its replacement vertex. There are usually some topological constrains to preserve the genus and to avoid introducing non-manifold simplexes. If the edge is not a valid candidate, its cost is set to infinity, and the edge is moved to the back of the queue. Given a valid edge, the edge collapse is performed, followed by a re-evaluation of edge costs for all nearby edges affected by the collapse. Once the costs for edges have been updated, the next iteration begins, and the process is repeated until a desired number of simplexes remain.

The general edge collapse method involves two major steps: choosing a measure that specifies the cost of collapsing an edge, and choosing the position for the new vertex that replaces the edge. Many approaches to vertex placement have been proposed, such as picking one of the vertices of the edge, using the midpoint of the edge, or choosing a position that minimizes the distance between the mesh before and after the edge collapse. This problem can be viewed as an optimization problem.

General pair contractions, where vertices need not be connected by an edge, have been proposed to provide a means of merging separate topological components during simplification. This may implicitly alter the topology of the surface (e.g., by closing holes). Contracting a non-edge pair will remove one vertex and join previously unconnected regions of the surface. In general, pair contraction requires the algorithm to support non-manifold surfaces, because when two separate components are joined together, a non-manifold region will almost certainly be created.

To perform the contraction, we must choose the target position for the new vertex. The simplest strategy is to use one of the original vertices, or the midpoint of the edge being contracted. However, the better approximation is usually required, and a new vertex is allowed to float freely in space in order to minimize some error metric. This will generally result in higher quality approximations, but the storage requirements for multiresolution representation will be higher.

The most important task is to find a way in which the cost of edges is evaluated for the contraction. The cost of the contraction is meant to reflect the amount of error introduced into the approximation by the contraction by the pair in question. Hoppe's algorithm [20] is based on minimization of an energy function. This function has four terms:

$$E(M) = E_{dist}(M) + E_{spring}(M) + E_{scalar}(M) + E_{disc}(M) \tag{4.2}$$

The first one E_{dist} ensures that the simplified mesh remains close to the original mesh. This geometric error term is very much like E_{avg} (see equation (3.11)). The second (E_{spring}) corresponds to placing on each edge of the mesh a spring of rest length zero and favours triangles with better proportions. The third term E_{scalar} discourages the simplification of colour and texture discontinuities. Finally, the last term E_{disc} discourages the simplification of topology and normal discontinuities. The algorithm maintains a set of sample points on the original surface, and the distances between these points and the corresponding closest points on the approximation determine the geometric error.

The basic steps of the algorithm are very similar to the general scheme of edge collapse algorithm. Anyway, we show it here for illustration:

1. Sort the edges using the least cost of simplification. This cost is measured using the variation of the energy function.
2. Apply the edge collapse operator for the edge at the head of the list and record the corresponding vertex split in the progressive mesh structure (including colour, texture and normal information).
3. The position of the new vertex is chosen among the two initial vertices and the centre of the edge, depending on which one is the closest to the original mesh.
4. Re-compute the cost for the edges that have been affected by the operator and reorder the list.
5. If the list is empty or the cost of the next simplification exceeds a certain bound, the algorithm terminates and returns the final progressive mesh.
6. Otherwise, jump to step 2.

This algorithm produces some of the highest quality results among currently available methods. The mesh optimisation algorithm [17] performs explicit search rather than simple greedy contraction. It exhibits even longer running times, but may produce the highest quality results.

The quadric error metric developed by Garland and Heckbert [10], [15] also defines error in terms of distances to sets of planes. However, it uses a much more efficient implicit representation of these sets. Each vertex is assigned a single symmetric 4x4 matrix, which can measure the sum of squared distances of a point to all the planes in the set. Under suitable conditions, the eigenvectors and eigenvalues of a quadric accumulated over a smooth surface region are determined by the principal directions and principal curvatures of the surface. While the quadric metric sacrifices some precision in assessing the approximation error, the resulting algorithm can produce quality approximations very rapidly.

Their simplification algorithm is based on the iterative contraction of vertex pairs; a generalization of the iterative edge contraction. A pair contraction moves vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ to the new position $\mathbf{v}$, connects all their incident edges to $\mathbf{v}_1$, and deletes the vertex $\Delta(\mathbf{v})$. The effect of contraction is small and highly localized. If $(\mathbf{v}_1,\mathbf{v}_2)$ is an edge, then 1 or more faces will be removed. Otherwise, two previously separate sections of the model will be joined at v. The primary benefit, which is gained by utilizing general vertex pair contractions, is the ability of the algorithm to join previously unconnected regions together. A potential side benefit is that it makes the algorithm less sensitive to the mesh connectivity of the original model.

At the initialization time, the set of valid pairs is chosen, and only these pairs are considered during the course of the algorithm. The pair $(\mathbf{v}_1,\mathbf{v}_2)$ is a valid pair for contraction if either $(\mathbf{v}_1,\mathbf{v}_2)$ is an edge, or $||\mathbf{v}_1\text{-}\mathbf{v}_2|| < t$, where t is a threshold parameter. Using a threshold $t = 0$ gives a simple edge contraction algorithm. To define the edge cost of contraction a symmetric 4x4 matrix **Q** is associated with each vertex, and the error at vertex v is defined as a quadric form $\Delta(v) = \mathbf{v}^T\mathbf{Q}\mathbf{v}$. For a given contraction of vertices **v1** and **v2** a new matrix **Q′**, which approximates the error at **v′**, is derived such as $\mathbf{Q}' = \mathbf{Q1} + \mathbf{Q2}$. A new position of vertex v is set according to the need to minimize $\Delta(\mathbf{v})$. This is equivalent to solving:

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} v' = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.3}$$

for **v′**. Assuming that the matrix is invertible, we can write

$$v' = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.4}$$

If the matrix is not invertible, the optimal vertex position is found along the segment $\mathbf{v}_1\mathbf{v}_2$. If this also fails, **v′** is chosen from amongst the endpoints and the midpoint.

The algorithm can be quickly summarized as follows:

- Compute the **Q** matrices for all the vertices.
- Select all valid pairs.

- Compute the optimal contraction target **v**′ for each valid pair ($\mathbf{v}_1$,$\mathbf{v}_2$). The error $\mathbf{v}'^T(\mathbf{Q}_1+\mathbf{Q}_2)\mathbf{v}'$ of this target vertex becomes the cost of contracting that pair.
- Place all the pairs in a heap keyed on cost with the minimum cost pair at the top.
- Iteratively remove the pair ($\mathbf{v}_1$,$\mathbf{v}_2$) of least cost from the heap, contract this pair, and update the costs of all valid pairs involving $\mathbf{v}_1$.

The initial matrices **Q** are computed using the matrix $\mathbf{K}_p$, where:

$$Q = \sum_{p \in planes(v)} K_p \tag{4.5}$$

$$K_p = pp^T = \begin{bmatrix} a^2 & ba & ca & da \\ ab & b^2 & cb & db \\ ac & bc & c^2 & dc \\ ad & bd & cd & d^2 \end{bmatrix} \tag{4.6}$$

where **p** = [a b c d]^T represents the plane for the triangles adjacent to vertex **v** defined by the equation ax + by + cz + d = 0 where $a^2 + b^2 + c^2 = 1$.

The "memoryless" algorithm developed by Lindstrom and Turk [31] is interesting in that, unlike most algorithms, it makes possible decisions based purely on the current approximation alone. No information about the original shape is retained. They use linear constraints, based primarily on the conservation of volume, in order to select an edge for contraction and the position at which the remaining vertex will be located.

In choosing the vertex position **v** from an edge collapse, [31] attempt to minimize the change of several geometric properties such as volume and area. The idea of preserving a volume is based on choosing a vertex position, which will minimize the volume of tetrahedrons constructed from the vertices of all triangles, affected by edge collapse, and the new vertex. Thus if we have a triangle $\mathbf{t}$=($\mathbf{v}_e$,$\mathbf{v}_1$,$\mathbf{v}_2$), where v_e is a vertex on collapsing edge, we try to minimize the volume of tetrahedral $\mathbf{t}_h$=($\mathbf{v}$,$\mathbf{v}_e$,$\mathbf{v}_1$,$\mathbf{v}_2$). The formula is as follows:

$$\sum_i V = ((v, v_0^{t_i}, v_1^{t_i}, v_2^{t_i})) = \sum_i \frac{1}{6} \begin{vmatrix} v_x & v_{0x}^{t_i} & v_{1x}^{t_i} & v_{2x}^{t_i} \\ v_y & v_{0y}^{t_i} & v_{1y}^{t_i} & v_{2y}^{t_i} \\ v_z & v_{0z}^{t_i} & v_{1z}^{t_i} & v_{2z}^{t_i} \\ 1 & 1 & 1 & 1 \end{vmatrix} = 0 \tag{4.7}$$

To further constraints the vertex position, [31] also attempts to minimize the unsigned volume of each individual tetrahedron, which is a measure of the local surface error for each corresponding triangle. To minimize these errors, the minimum of

$$f_v(e,v) = \sum_i V((v, v_0^{t_i}, v_1^{t_i}, v_2^{t_i}))^2 \tag{4.8}$$

is searched for.

By defining the edge cost in terms of the objective function that is minimized above, the vertex position is optimal with respect to the incurred cost of collapsing the edge. That is, the cost is a weighted sum of the terms minimized in the volume optimization. As a measure of triangle shape quality, Lindstrom and Turk have chosen the following expression:

$$f_s(e,v) = \sum_i L((v, v_i))^2 \tag{4.9}$$

which is the sum of squared lengths of the edges incident upon v.

The reported results suggest that it can generate good quality results, and that it is fairly efficient, particularly in memory consumption.

One of the major benefits of iterative decimation is the hierarchical structure that it induces on the surface. This quite naturally leads to useful multiresolution surface representation, as described in following section.

4.9 Multiresolution Models

The simplest method for creating multiresolution surface models is to generate a set of increasingly simple approximations. A renderer can then select, which model will be rendered, accordingly to the level of detail needed. In other words it would be using a series of discrete levels of detail, from which our multiresolution model would consist. The reason why the discrete multiresolution approach is so popular is its simplicity. However it can potentially cause significant visual artefacts. Since the number of polygons in two models can differ significantly, their appearances in the output image can be different as well. This can lead to "popping" artefacts while the level of detail is changing. Despite this limitation, discrete multiresolution models can be quite useful in many applications. Support for discrete levels of detail has been included in a number of commercial rendering systems, including RenderMan, Open Invertor, or IRIS Performer. Discrete levels of detail have also been used for accelerating the computation of radiosity solutions.

On the other hand there are many applications where discrete multiresolution models are not sufficient. Imagine a large surface, such as terrain, being viewed from the viewpoint positioned just above the surface, looking out towards the horizon. An approximation with a constant level of detail would be either too dense in the distance or too sparse near the viewpoint. In such a case we would like the level of detail to be view dependent. Thus, we need a multiresolution representation that continuously adapts the surface at run time based on viewing conditions. Since this kind of representation is required by many applications today, we dedicate a particular attention to this area.

The direct by-product of iterative contraction is an incremental representation, the so called simplification stream [11]. During the process of simplification, we get a sequence of models

$$M^0 \xrightarrow{\phi^1} M^1 \xrightarrow{\phi^2} M^2 \xrightarrow{\phi^3} \ldots\ldots \xrightarrow{\phi^k} M^k \qquad (4.10)$$

where each step $M^{i-1} \to M^i$ corresponds to the application of a single contraction ϕ^i. Thus each intermediate approximation *Mi* can be expressed as the result of applying some of the total sequence of contractions onto the original mesh M_0. Since we store the entire original model M_0 plus the contraction sequence, the resulting representation is necessarily larger than the original model. If we assume that our original model is very large and our desired approximation is quite small, certainly a common case, we are faced with more significant problem. Fortunately, a closely related representation can solve this problem.

The progressive mesh (PM) structure, originally introduced by Hoppe [20], [18], provides the same functionality as a simplification stream. However, it has two important advantages. First, the resulting representation can actually be smaller than the original model. Second, reconstruction time is proportional to the desired approximation size. A PM is, in essence, a reversed simplification stream. It exploits the fact that the contraction operator is invertible. For each edge contraction we can define a corresponding inverse called a vertex split. Thus we begin with the final approximation (in Hoppe's notation base mesh) and produce a sequence of models applying a sequence of vertex split operations and terminating at the original model. Each item in vertex split sequence must encode the vertex being split, positions for the two resulting vertices, and which triangles to introduce into the mesh. Hoppe [20] also demonstrated that PM is an effective technique for compressing the input geometry.

It does not need to be only an edge contraction approach that can generate multiresolution models. There are plenty of techniques based on vertex decimation as well. Klein et al. [23], [25] suggest starting from a base mesh, which is refined by adding vertices in the reverse order of their removal. The main difference to the progressive meshes algorithms is that aside from a new vertex and two anchor vertices also the triangulation of the accompanying fragment has to be transmitted. The unique solution brings Ciampaliny et al. [3], which can rebuild an approximation of certain error. Let us consider a set of all the triangles that were generated during the whole decimation process, including the triangle of the original mesh. Each facet from the set is characterized by two time stamps: its creation and its elimination. An intermediate mesh is associated by definition with each time stamp, and therefore we can associate with each time stamp the global approximation error held by the mesh. Given the birth and death time stamps, each facet is therefore tagged with two errors, called the birth and death errors. The extraction of a representation at a given precision is therefore straightforward: surface is composed of all of the facets such that their life interval contains the error threshold searched for.

4.10 Post-processing techniques

Mesh Smoothing

Since the majority of simplification algorithms is primarily focused on the amount of triangles and the error of the approximation, the need for other requirements such as the quality of triangulation or the smoothness of the surfaces are often neglected. However, it is often desired to use the approximations in further scientific processing and not only for visualization. It is common to find that the quality of approximations is improved in some post-processing. In recent years many techniques for mesh smoothing have been developed especially for the needs of CAD/CAM engineering design and analysis [52]. Since most existing algorithms based on fairness norm optimisation are prohibitively expensive for very large surfaces, Taubin's [48], [49] geometric signal processing on polygonal meshes with linear time and space complexity seems to be quite promising. The algorithm is based on an analogy with signal processing. The smooth surface can be seen as a signal without high frequencies. Taubin converts the problem of surface smoothing to the problem of low-pass filtering. Most smoothing algorithms move vertices of the polygonal mesh without changing the connectivity of the faces. Basically the internal vertices are iteratively moved to the barycentre of their neighbouring vertices. Taubin's special filter function avoids mesh shrinkage, and also the triangles of the mesh are forced to be equilateral.

His approach to extend Fourier analysis to signals defined on polyhedral surfaces of arbitrary topology is based on the observation that the classical Fourier transform of signal can be seen as the decomposition of the signal into a linear combination of the eigenvectors of the Laplacian operator. To extend Fourier analysis to surfaces of arbitrary topology it is only necessary to define a new operator that takes the place of Laplacian. A discrete surface signal is a function $x = (x_1,\ldots,x_n)^t$ defined on the vertices of a polyhedral surface. The discrete Laplacian of a discrete surface signal is defined by weighted averages over its neighbourhoods

$$\Delta x_i = \sum_{j \in i^*} w_{ij}(x_j - x_i) \tag{4.11}$$

where the weights w_{ij} are positive numbers that add up to one. The weights can be chosen in many different ways taking into consideration the neighbourhood structures. One particularly simple choice is to set w_{ij} equal to the inverse of the number of neighbours $1/i^*$ of vertex $\mathbf{v}_i$. If $\mathbf{W} = (w_{ij})$ is the matrix of weights, with

$w_{ij} = 0$ when j is not a neighbour of *i*, the matrix K is defined as **K** = **I** – **W**. Now low-pass filtering – the approximate projection onto the subspace of low frequencies – can be formulated in exactly the same way as for n-periodic signals, as the multiplication of a function *f*(**K**) of the matrix **K** by the original signal

$$x' = f(K)x \tag{4.12}$$

and this process can be iterated *N* times

$$x^N = f(K)^N x \tag{4.13}$$

The function of one variable *f*(*k*) is the transfer function of the filter. For example in the case of Gaussian smoothing the transfer function is $f(k) = 1 - \lambda k$. To avoid the shrinkage effect know in Gaussian smoothing, Taubin proposes following function

$$f(k) = (1 - \lambda k)(1 - \mu k) \tag{4.14}$$

where $0 < \lambda$, and μ is a new negative scale factor such that $\mu < -\lambda$. That is, after the Gaussian smoothing step with positive scale factor for all vertices is performed – the shrinking step –, we then perform another similar step for all the vertices, but with negative scale factor μ instead of λ:

$$\begin{aligned} \forall i:\ x_i' &= x_i + \lambda x_i \\ \forall i:\ x_i' &= x_i + \mu x_i \end{aligned} \tag{4.15}$$

Since $f(0) = 1$ and $\mu + \lambda < 0$, there is a positive value of *k*, the pass-band frequency k_{pb}, such that $f(k_{pb}) = 1$. The value of k_{pb} is

$$k_{pb} = \frac{1}{\lambda} + \frac{1}{\mu} > 0 \tag{4.16}$$

Values from 0.01 to 0.1 of k_{pb} produce good results. Since we want to minimize *N*, the number of iterations, λ is chosen to be as large as possible, while keeping $|f(k)|<1$ for $k_{pb} < k \leq 2$. The exact values of μ and λ are computed using the expressions above.

Image-Driven Mesh Optimization

This method presented by [32] does provide optimization on already simplified model comparing visual appearance of the original and resulting model. The algorithm begins with two input meshes, the original detailed mesh and a simplified version of this mesh that has the desired number of vertices. It is unimportant what method is used to create the simplified mesh. Given a number of viewpoints, the algorithm renders images of both the original and the simplified meshes for each viewpoint and an edge in the simplified mesh is selected for improvement (see Figure 4-8).

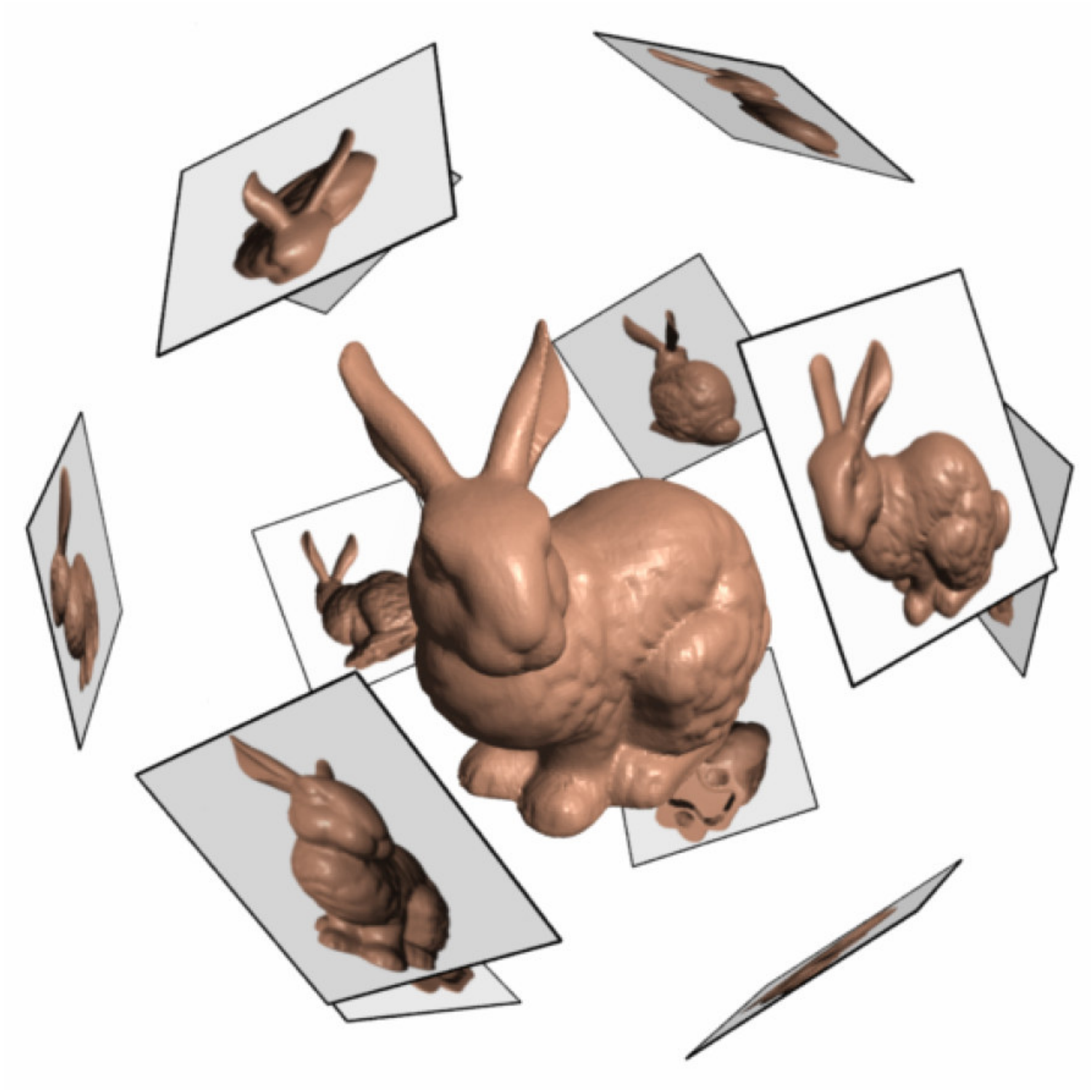


Figure 4-8: Twelve uniformly distributed views of a model. The viewpoints correspond to the vertices of a regular icosahedron (taken from [32]).

The algorithm then attempts a number of changes to the mesh at and around this edge to create a mesh whose rendered images are closer to those of the original mesh. Possible changes to the mesh include moving two or more vertices, edge swapping, or even a vertex teleport (moving a vertex between entirely different portions of the mesh).

The measure of similarity is based on the work by [33] in which an image metric is used to order a set of edge collapses. Although their method uses geometry-based heuristics for positioning the vertices, here is used the image metric directly to determine the best vertex positions and what changes to make to the connectivity.

This mesh optimization method takes into account not just the geometry of a model but also properties such as textures and surface normals. This approach fixes problems in a simplified mesh that simplification methods are insensitive to, such as cracks between surface parts and object interpenetration.

4.11 Conclusion

Recent research in the field of surface simplification has produced several effective techniques for constructing approximations and multiresolutional representations. Some commercial packages have included several simplification facilities. The algorithms available offer several possible trade offs between quality and efficiency. We have very high quality, but very slow algorithms such as mesh optimisation [17]. On the other hand there are very fast, but low quality vertex clustering algorithms [35]. Somewhere between these extremes we have a number of algorithms, such as the quadric error metric [10] or vertex decimation [46], which provide various compromises between the speed and quality of the approximation. There are a number of areas in which current simplification methods could be improved.

A new algorithm capable of producing approximations, which are provably close to optimal would be quite useful, or an algorithm, which could preserve higher-level surface characteristic, such as symmetry.

All current simplification methods assume that the surface being simplified is rigid. There are many applications where surfaces are changing over time. For example, animation systems usually represent characters as surface attached to articulated skeletons. As the skeletal joints bend, the surface is deformed. Current simplification methods must be extended to handle this more generalized class of models.

Many simplification methods are based on the same framework: greedy application of simplification operators. For example, greedy decimation can limit the quality of the final result. Since it only iteratively does what appears to be the best local operation to perform, a bad decision at some point can lead to results that are far from optimal. Alternative frameworks are possible. For instance techniques using a simulated annealing-like process [17], or methods based on the principle of signal processing on polygonal meshes [51].

New methods for measuring approximation error are also needed. Similarity of appearance is the ultimate goal for rendering applications. It would be helpful to have appearance-based metrics for comparing the visual similarity of two models. Even if there are primarily concerned with preserving the shape of an object, it is also unclear whether metrics like E_{max} and E_{avg} adequately reflect the similarity of an approximation whose topology has been simplified.

Chapter 5

Simplification Algorithm

In the previous chapter, we have reviewed various algorithms which have been developed for automatic simplification of polygonal surface models. Now, we will present the simplification algorithm which we have developed. It is founded on two fundamental components: vertex decimation and half-edge collapse algorithm. In this chapter, we will focus on the description of the algorithm itself. We will examine algorithm's performance and its genesis in Chapter 6 and 7.

5.1 Design goals

The core of the algorithm lies on vertex decimation. We assume to have digitally measured data, where vertices represent the exact values and edges and triangles just supply the mesh connectivity. In many cases the actual presence of edges in the mesh is usually given either by some higher knowledge or as a random result of polygonization algorithm. On Figure 5-1 you can see an example of four measured vertices (with boundary) - these values are exact but there is no additional information, how the edge e_1 should be placed in resulting triangulation.

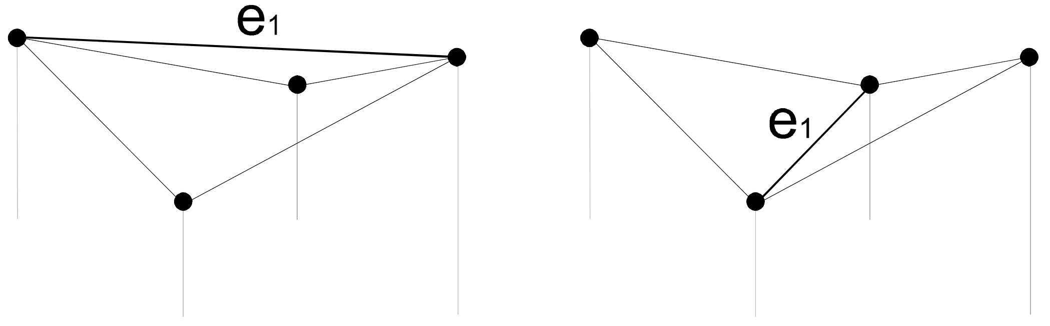


Figure 5-1: Two possibilities of placing edge e_1 to triangulate the area among four vertices.

Since surface energy tends to be minimal, we can suppose such edge position that will minimize the resulting area. However, in cases such as sharp-edges the situation can be different. This implies our goal not to introduce new vertices in simplified mesh but keep just a subset of original vertices in resulting approximation. Having these assumptions, we need a method to evaluate vertex importance considering neighbouring vertices only. Such evaluation provides a method proposed originally by W. Schroeder [46].

5.2 Importance evaluation

The evaluation scheme depends on vertex topology in input mesh. Therefore vertex type must be classified at first. We recognize 5 vertex types, see Figure 5-2.

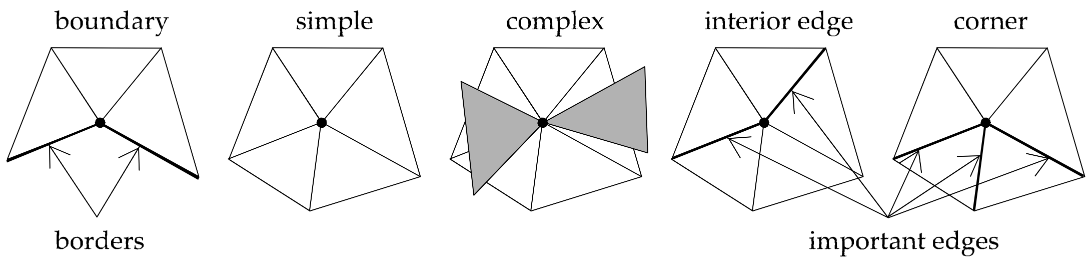


Figure 5-2: Vertex classification.

Simple vertex is a vertex with a neighbourhood topologically equivalent to a disk, where no sharp edge is adjacent to the vertex. A boundary vertex lies on the border of the mesh, where by the border is meant an edge adjacent to one triangle only. Interior-edge vertex is vertex similar to simple vertex, except it is adjacent to two sharp-edges. Corner vertex is adjacent to one, three or more sharp-edges, and a complex vertex is a vertex that belongs to non-manifold area.

It is obvious that the original algorithm can deal with non-manifold meshes too, but just in a manner, that non-manifold parts are left untouched. In our modification, we can process non-manifold meshes as well. Vertices of particular interest are only simple, border and interior-edge vertices. For each of them a different evaluation process is used.

The importance of simple vertex **v** is given by the distance of the vertex from the average plane given by neighbouring vertices (see Figure 5-3a). A neighbouring vertex is a vertex which shares an edge with the vertex in question, in other words, there exists an edge between two neighbouring vertices. The plane is called average since the neighbouring vertices generally do not lie on the plane and we have to specify it artificially using the average normal vector **n** and a point **x**. The normal vector **n** is computed as a normalized average of area weighted unitary normal vectors $\mathbf{n}_i$ of each surrounding triangle. The bigger the area P_i of a triangle is, the bigger is its influence on the resulting normal vector.

$$\mathbf{N} = \frac{\sum_m \mathbf{n}_i P_i}{\sum_m P_i}, \mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|}, \tag{5.1}$$

where m is the number of adjacent triangles to the vertex v.

Having the normal vector **n** of the average plane, we need to determine the point **x**, lying on the average plane. This point is defined in terms of the average of area weighted vectors $\mathbf{x}_i$ from the origin to the midpoint of each surrounding triangle

$$\mathbf{x} = \frac{\sum_m \mathbf{x}_i P_i}{\sum_m P_i} \tag{5.2}$$

The final distance d of the vertex v from the average plane is given by the following equation

$$d = |\mathbf{n}.(\mathbf{v} - \mathbf{x})|. \tag{5.3}$$

In case of boundary or interior edge vertices, the importance is evaluated as the distance of vertex v from a bisector given by its two neighbouring border vertices or vertices on opposite sides of interior edge respectively (see Figure 5-3 - b).

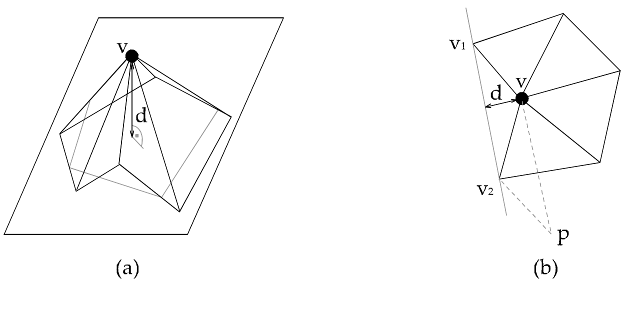


Figure 5-3: Distance d for vertex importance evaluation, (a) - simple vertex case, (b) - boundary vertex case.

To evaluate the distance of a point from a bisector, we can use an analogy with the relation of height and area in parallelogram (**v**,**v**1,**v**2,**p**). The area can be either computed as a product of the base length and the height, or as a size of vector product $|\mathbf{v}_1\mathbf{v}_2 \times \mathbf{v}_1\mathbf{v}|$. Thus the distance can be evaluated as

$$d = \frac{|\mathbf{v}_1(\mathbf{v} \times \mathbf{q})|}{|\mathbf{q}|}, \mathbf{q} = \mathbf{v}_1\mathbf{v}_2. \tag{5.4}$$

Complex and corner vertices have their importance set to some high value which is far below the threshold which allows vertices to be removed. When all the vertices are evaluated, a priority queue must be created, where the vertices are sorted according to their importance in increasing order.

As obvious, this process frames the outer optimization loop and is based on vertex collapse operator. In the inner loop an edge collapse operator is used. Choosing edge collapse had several reasons. The most important one is that after vertex removal a triangulation of resulting area has to be done. Although it is possible to use projection in 2D, such a triangulation is a non trivial problem from aside its computational cost, especially if we want to compare qualities of several possibilities. Therefore, once we take the least important vertex from the priority queue, we start evaluating its neighbourhood in terms of searching the best edge to collapse.

Since we want only the removal of the least important vertex, we use half-edge collapse approach, where no new vertex position is evaluated during simplification (chosen edge is collapsed to its endpoint different from removing vertex).

5.3 Local edge contraction

When vertex to be removed is chosen a local simplification operator is applied. All neighbouring edges are evaluated according to a change they cause to the surface when collapsed. To find such evaluation we simulate all possible contractions and estimate the quality of resulting mesh. This quality measure is defined as a minimal change to resulting mesh area while given maximal triangle normal deviation allowed. Such conditions are based on assumption that simplified area should be nearly planar. The edge with best evaluation is collapsed to its endpoint, opposite to the vertex being removed. The only exceptions are so called sharp edges; edges whose adjacent triangles have spatial angle smaller than given threshold (see Figure 5-4). In this case, the only solution is a contraction of this particular edge.

During half-edge collapse one vertex, one edge and two triangles are removed from the mesh. After the removal all affected vertices and triangles are updated (their importance values, normal vectors and adjacency information) and the global priority queue is re-sorted.

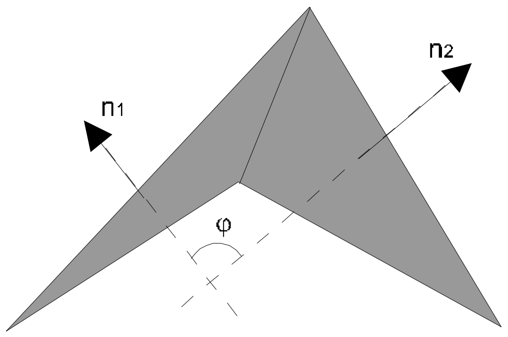


Figure 5-4: Spatial angle φ of two triangles is given by the angle between their normals.

Because the edge contraction can potentially introduce undesirable inconsistencies or degeneracies into the mesh, we must apply some consistency checks to a proposed operation. If one of the checks fails, we discard the contraction entirely.

The most common consistency check is related to the problem of mesh inversion. Consider the contraction shown in Figure 5-5. One popular approach to detecting this situation is to examine the normals of the facets adjoining vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ before and after contraction [4]. If a face's normal changes by more than some significant threshold, we can regard this face as being "flipped". A contraction is discarded if any of the local faces flip. To prevent fold–over we need to pick suitable threshold value.

We have chosen to use more careful check [10], which appears to perform more reliably in practice. For every face around $\mathbf{v}_i$, excluding the faces shared with $\mathbf{v}_j$, there is an edge opposite $\mathbf{v}_i$. If we place a plane perpendicular to the face through the edge, the vertex $\mathbf{v}_j$ must lie on the same side of the plane in all cases, see Figure 5-5.

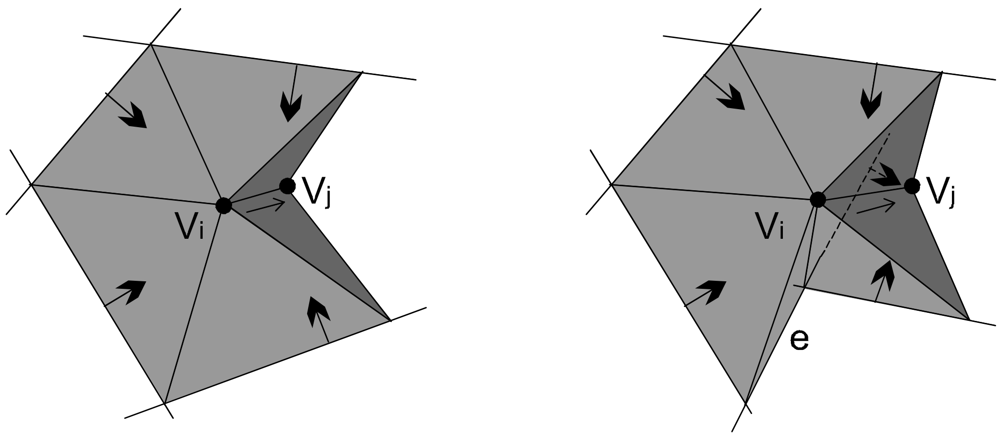


Figure 5-5: The new vertex ($\mathbf{v}_j$) must lie on the same half-plane of all oriented edges to prevent fold–over.

This technique can disclose real foldovers and no threshold values are needed to be set.

5.4 Parallel processing

Our primary aim was to implement an efficient triangle mesh simplification algorithm in parallel environment, to obtain faster results for very large data sets. We can see that algorithm described in previous two sections works in three steps:

- Vertex topology and importance evaluation. This part can be done in parallel, because all vertices are obviously independent to the others.
- Priority queue creation using Quick Sort algorithm to sort vertices according to their importance. Runs sequentially for some reasons described later.
- Decimation. This part of the algorithm could run effectively in parallel, but we have to achieve some important restrictions.

A basic idea, which has been used in theoretical work [22], is that decimation by deleting an independent set of vertices (no two of which are joined by an edge) can be run efficiently in parallel. The vertex removals are independent and they leave one hole per one deleted vertex, which can be retriangulated independently. This decreases the program complexity and running time significantly. Since deletion and retriangulation is related to the degree of vertices being removed (O (d^2) time in the worst case, where d is the vertex degree), [22] has advocated deleting low degree vertices, $d < 10$, and proved that this still allows large independent sets (>1/6 of vertices). However, this approach ignores the preservation of the model shape. Therefore we used a technique [21] when we assign an importance value to each vertex and then select an independent set to be deleted by greedily choosing vertices of low importance relative to their neighbours.

To construct an independent set from an assignment of importance values, it is natural to use a greedy strategy – go through the vertices in order of their importance and take a vertex if none of its neighbours have been taken. That means in independent set can be only vertices that do not share an edge with the each other.

5.5 Super-independent Set of Vertices

After several experiments we found, that the independent set described above is not fully adequate. The criterion of no sharing edge between two independent vertices is not sufficient for effective parallelisation. Using such an independent set of vertices leads to critical sections in parallel code. Since the computation is quite fast, critical sections rapidly decrease the efficiency of the algorithm.

Wherefore we used a *super* independent set, where every two triangles including two independent vertices can not share an edge, see Figure 5-6.

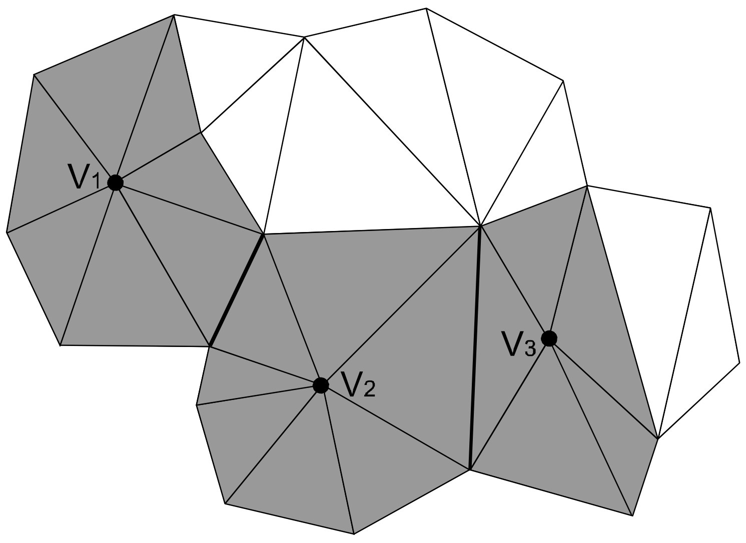


Figure 5-6: Vertices **v**$_1$, **v**$_2$, **v**$_3$ are independent to each other; vertices **v**$_1$ and **v**$_3$ are super independent.

If we remove one vertex in the independent set, the removal changes the properties of the vertex neighbours. That affects neighbourhood of other vertices in the set. In the *super* independent set are independent even vertex neighbours, so vertices are completely independent and the parallelisation can be done without critical sections in program code.

Due to data structures used we can create the independent set in O(*n*) time, where *n* is the number of vertices. Using the independent set of vertices, we can split third step of the algorithm (the decimation) in to two parts – making an independent set and own decimation (now easy to run parallel).

To decrease the system service overhead with managing threads, we split a set of computed vertices to number of parts equals to the number of free processors (used threads) roughly, and each thread computes one part.

Here is a detailed description of our parallel algorithm:

1. Divide the set of vertices into *n* parts, where n is equal to the number of free processors.
   - Get the number of processors.
   - Divide the set of vertices into *n* parts of the same number of vertices.
2. Run *n* threads to evaluate vertex importance according to its topology. Each thread makes a computation on its own set of vertices.
   - Determine a vertex topology.
   - For simple, boundary or interior edge vertices, compute their importance. The importance for any other type of vertex is set to any high value ("infinite").
3. After all threads finish their job, sort (Quick Sort algorithm) all the vertices (increasing) according to their importance.
4. Find an independent set of vertices. For each vertex do:
   - If the vertex is mark as unused in the independent set (initially all vertices are marked as unused), check its neighbours. If all the neighbours are unused too, put the vertex into the independent set and mark the vertex and all its neighbour vertices as used.
   - Used vertices and their neighbours are skipped.
5. Divide the independent set of vertices into *n* parts (n is the number of free processors).
6. Run *n* threads for decimation. Each thread makes the decimation on its own set of vertices.
   - For each eliminated vertex, find the shortest edge that includes it.
   - Test the consistency of the mesh if this edge is contracted (removed).
   - If the consistency test is OK, remove the vertex and retriangulate the arising hole, otherwise find another short edge and go to the previous point.
7. Repeat steps 1– 6 until the required degree of the mesh reduction is reached.

The algorithm is assembled from four main tasks. Vertex importance evaluation (which can run in parallel), building a priority queue according to vertex importance (runs sequentially for the moment), super-independent set creation (which has to be sequential) and own parallel simplification process. Figure 5-7 shows the time ratio for one and 8 processors used during simplification of Happy Buddha model.

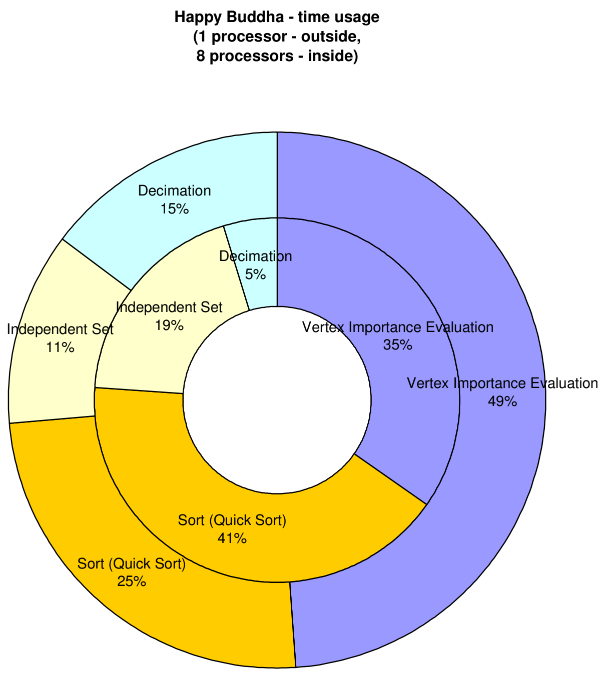


Figure 5-7: The time ratio of various parts of the algorithm for Happy Buddha model with one and eight processors used.

As obvious, the most time consuming parts are importance evaluation and sequential vertex sorting rapidly decreases the algorithm efficiency. Also the creation of super-independent set seems to be limiting as the maximal theoretical speedup of this approach is approximately 15. These results were subjects of further improvement as discussed in section 5.6 and also in Chapter 6.

5.6 Further Improvement

Using independent or super-independent sets of vertices, we need to sort vertices according to their importance and also to create the independent set itself. This appeared to be a critical part of the algorithm as described above.

Since we did not want to use sorting algorithms because of their time complexity, we used a special function to threshold vertices and let only few least important vertices to be considered as candidates for the reduction.

The initial idea was to divide data set (vertices) according to the number of free processors and run decimation as several independent parts. As we already mentioned, most of data are produced by 3D scanners or iso-surface extraction methods such as Marching Cubes. Considering the principle of both techniques as well as some possible rendering optimizations, we can suppose that such triangular mesh will be stored as a sequence of potential triangle strips. Thus neighbouring vertices may be stored next to each other in data file or memory. In other words, if we divide a data set into few groups, according to vertices index, there is a good probability that vertices in each group will be close to each other and will constitute a continual mesh. Those groups can be processed without critical sections except vertices on the boundary of each group. Such vertices are post-processed.

This approach brought surprisingly good results for the real object models in the sense of processing time and acceptable quality of approximation. On the other hand there were some problems controlling the simplification degree in sense of required budget (resulting number of triangles). Also vertices on the border of groups had to be handled in a special way. Another problem came up with artificially generated datasets or changed on purpose. Such models (e.g. Bunny, see Table 6-1) do not fit to the assumption about strips and the algorithm may be quite ineffective in this case.

Instead of such a blind vertex division, we searched for bucketing function which would “sort” vertices in O(*N*), where *N* is a number of vertices.

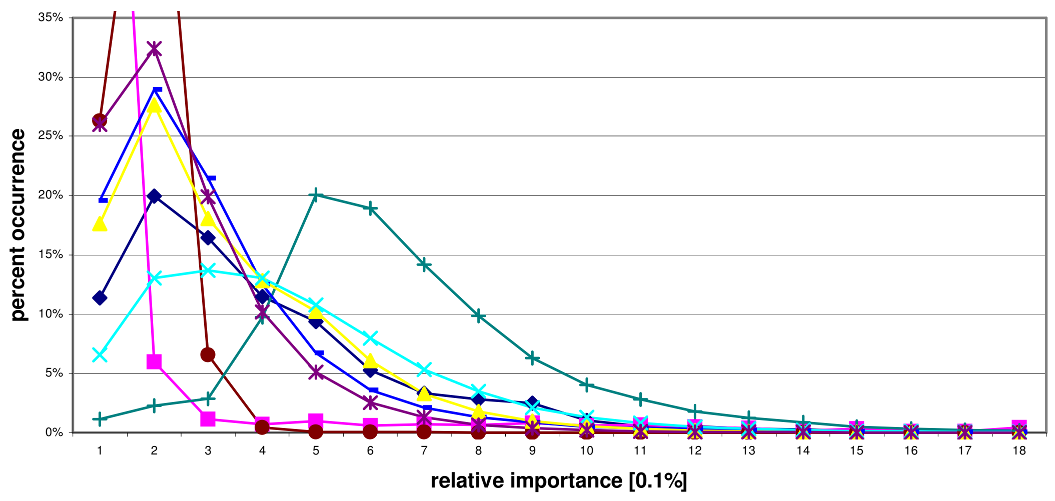


Figure 5-8: Vertex importance histogram.

A histogram of vertices importance for tested data sets is shown on Figure 5-8. It is obvious that over 90% of all vertices have their importance below 1% of maximum importance value. Therefore we proposed a simple bucketing function *y=f(x)* (5.5) shown on Figure 5-9.

$$y = \frac{x}{|x| + k} \tag{5.5}$$

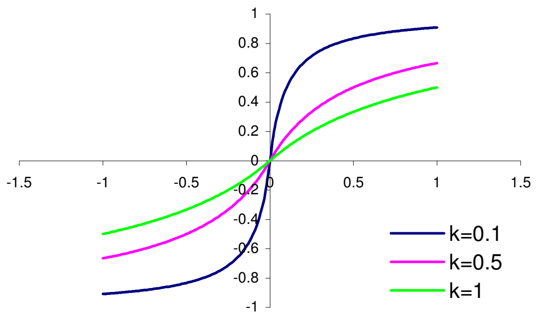


Figure 5-9**:** Graph of the function used.

This function is capable to map the interval $<0,\infty)$ to $<0,1)$ non-linearly. Because we are interested in only 1% of important vertices, the function (5.5) must be modified accordingly (see Figure 6) and scaling coefficient *C* must be introduced.

$$y = C\frac{x}{x+k} \qquad x \geq 0 \tag{5.6}$$

From equation (4) we can see that coefficients *C* and *k* must be determined somehow. In our approach we decided that we will have two parameters *a* and α that will be experimentally determined by large data sets processing (see Figure 5-10).

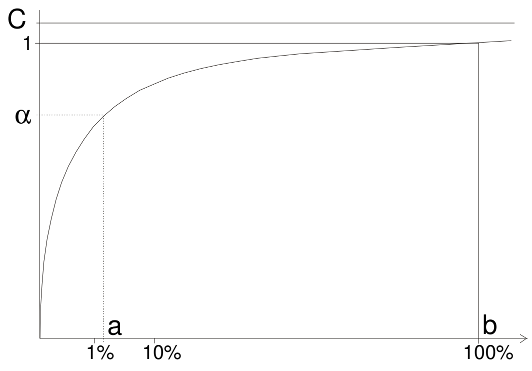


Figure 5-10**:** Definition of the hash function.

The coefficient *b* is equal to the maximal importance in the given data set and therefore *f(b)* must be equal 1. The coefficient *a* means the boundary for maximal importance of vertices to be considered for processing and α determines the slope of the curve, actually. Those conditions can be used for parameter *k* and *C* determination as follows:

$$1 = C\frac{b}{b+k}, \; b+k \neq 0 \tag{5.7}$$

$$\alpha = C\frac{a}{a+k}, \; a+k \neq 0 \tag{5.8}$$

Then

$$C = \frac{b+k}{b} \tag{5.9}$$

and

$$\alpha = \frac{b+k}{b}\frac{a}{a+k} \quad (5.10)$$

Solving that we get

$$k = \frac{ab(1-\alpha)}{\alpha b - a} \quad (5.11)$$

$$C = \frac{b+k}{b} = 1 + \frac{k}{b} \quad (5.12)$$

for $b = 1$

$$k = \frac{a(1-\alpha)}{\alpha - a}, \qquad C = 1 + k \quad (5.13)$$

According to our experiments on large data sets, we have found that the optimal values of coefficients are following: $a = 2$, $\alpha = 80$. It means that 2 percents of the least important vertices will be mapped onto 80% of the whole hash table. User has to set values of both parameters and. The parameter α has a direct influence to the cluster length in the hash data structure, where cluster length is equal to number of vertices in the same bucket.

Thanks to this function we can assign the vertices to clusters of the same reasonably small length. Vertices are then removed cluster by cluster from the least important clusters to the most important. The clusters are sorted while they are created and whole bucketing has O(*N*) time complexity.

Having described the specific details of the method, we can present our new algorithm now:

1. Evaluate the importance of all vertices
2. Make clusters according to the importance of the vertices
3. Remove vertex from the first cluster, if it is empty continue with the next one
4. Evaluate changed importance of neighbouring vertices and insert vertices in the proper cluster
5. Repeat steps 3 and 4 until desired reduction has been reached

All details about our simplification approach can be found in publications listed in Appendix A. In the following section we present our results and draw conclusions.

Chapter 6

Results and limitations

In this chapter we provide several experimental results and make a conclusion at the end. We present tables of running times, graphs of speed and approximation quality comparison and also pictures of rendered models. Because of some special kind of models used, we have to claim that no animal were harmed during following experiments.

6.1 Experimental Results

We have used several large data sets but we have drawn up our experimental results using only 7 different data sets, see Table 6-1.

model name	# triangles	# vertices
Teeth	58,328	29,166
Bunny	69,451	35,947
Horse	96,966	48,485
Bone	137,072	60,537
Hand	654,666	327,323
Dragon	871,414	437,645
Happy Buddha	1,087,716	543,652
Turbine blade	1,765,388	882,954

Table 6-1: Data sets used.

Table 6-2 shows time comparison achieved by reducing the models using 1 to 8 – processor computer (DELL Power Edge 8450 – 8xPentium III, cache 2MB, 550MHz, 2GB RAM, running on the Windows 2000).

Model name	Time [sec] obtained with different number of processors (threads) used							
	1	2	3	4	5	6	7	8
Horse	8.9	6.3	5.5	4.9	4.7	4.5	4.4	4.3
Bone	13.5	9.4	8.2	7.6	7.0	6.8	6.6	6.4
Hand	69.2	51.5	44.8	41.0	39.7	38.4	37.5	36.7
Dragon	93.6	69.5	61.5	57.6	54.3	52.6	51.3	50.6
Happy Buddha	118.3	89.1	78.1	73.2	69.3	67.8	65.9	64.5

Table 6-2: Obtained time (in seconds) for 90% reduction on 1 to 8 processors active.

We investigated the acceleration and the efficiency for different size of data sets according to the number of processors used.

6.2 Speedup comparison

Figure 6-1 shows a graph of the speedup comparison. The speedup *a* is computed from total times (sequential and parallel parts of the algorithm together) using expression (6.1).

$$a = \frac{time_1}{time_N} \tag{6.1}$$

where N=1..8 is a number of processors used and $time_N$ is the time obtained if N processors (threads) are used. Figure 6-2 presents the speed-up of particular parts of the simplification process. The vertex importance evaluation, vertices sorting, independent set of vertices creation and decimation.

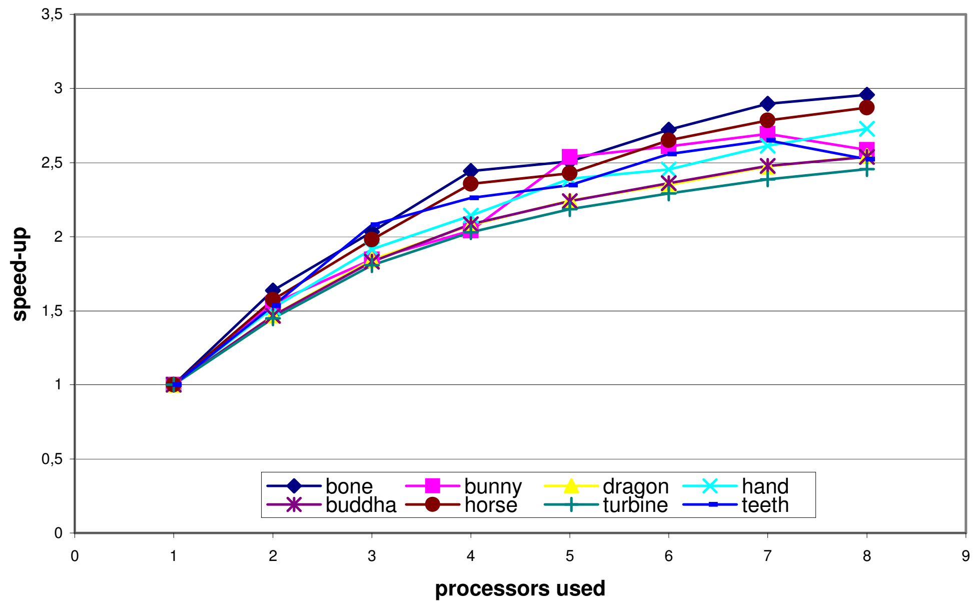


Figure 6-1: The speed-up of total computation (total time), parallel and sequential parts together; the acceleration is computed for several models of different amount of triangles.

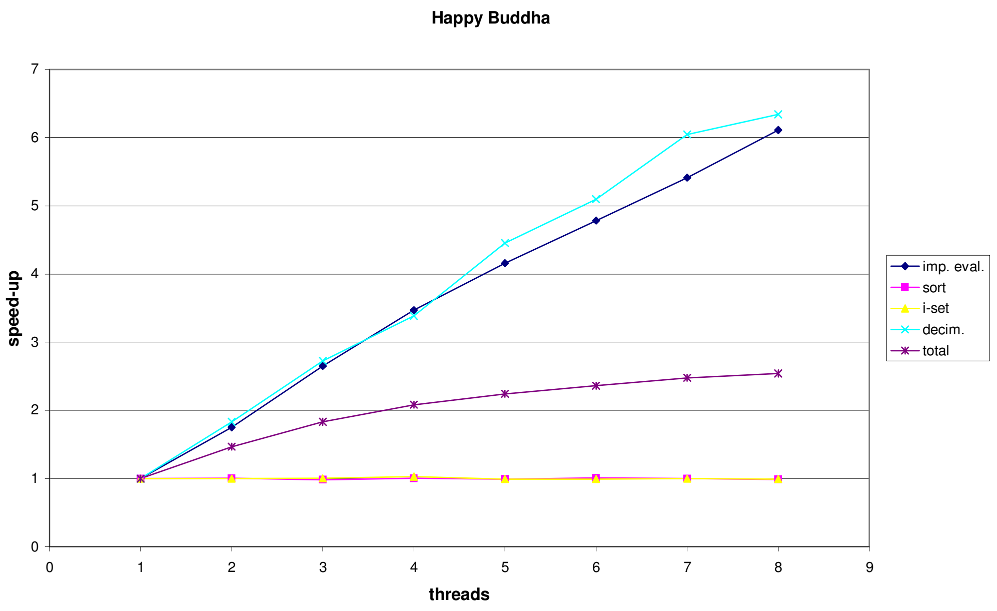


Figure 6-2: The speed-up of partial phases of the simplification algorithm.

6.3 Efficiency comparison

On Figure 6-3 we can see a comparison of the computational efficiency according to the size of data set and number of processors used. The efficiency *e* is defined as a speedup divided by the number of processors used:

$$e = \frac{time_1}{N * time_N} \qquad (6.2)$$

where *N*=1..8 is a number of processors used and $time_N$ is the computational time if *N* processors are used.

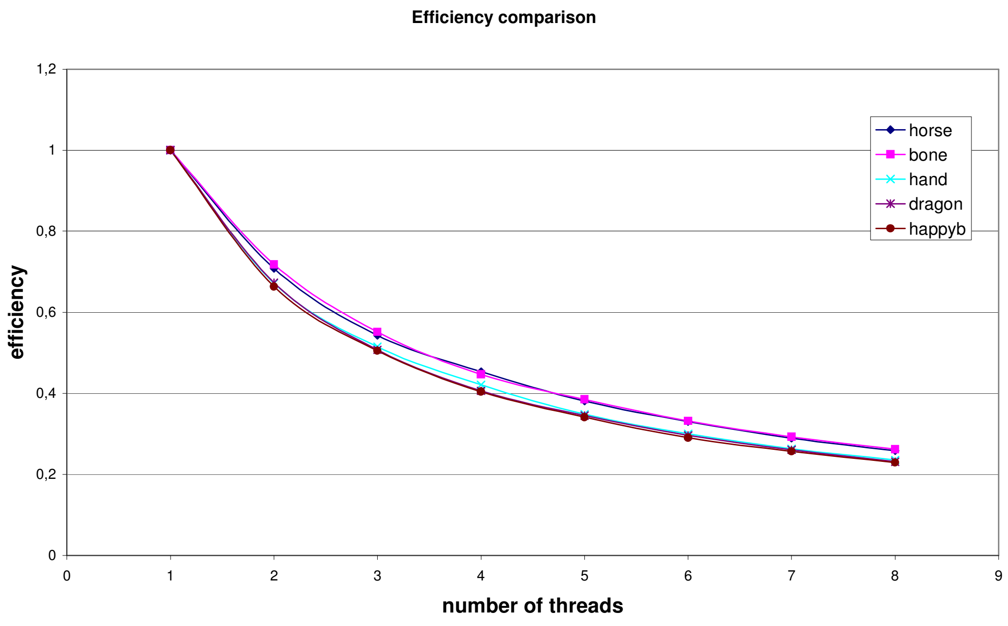


Figure 6-3: The efficiency of total computation; the efficiency is computed for several models of different amount of triangles.

6.4 Amdahl's law

The experiments proved that the method is stable according to the number of processors used and all the results meet the Amdahl's law (6.3) perfectly.

$$a_{teor} = \frac{1}{(1-p)+\frac{p}{N}} \qquad (6.3)$$

and therefore

$$p = \frac{N * (1 - a_{teor})}{a_{teor} * (N - 1)} \tag{6.4}$$

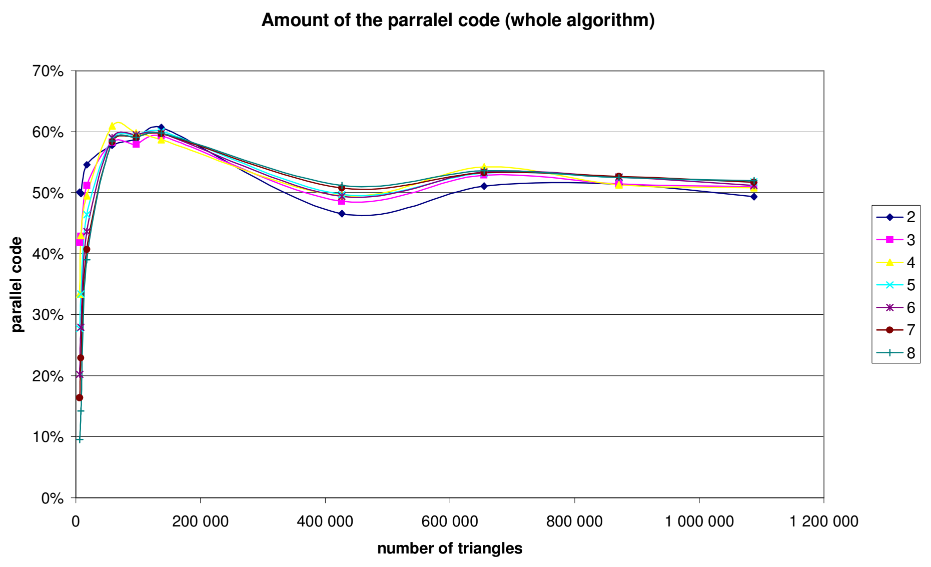
where p is potentially parallel code and N is the number of processors used.

The value of potentially parallel code is independent from the number of processors used, see Table 6-3 and for the large model Happy Buddha the value $p = 0.51$ was reached for the whole algorithm.

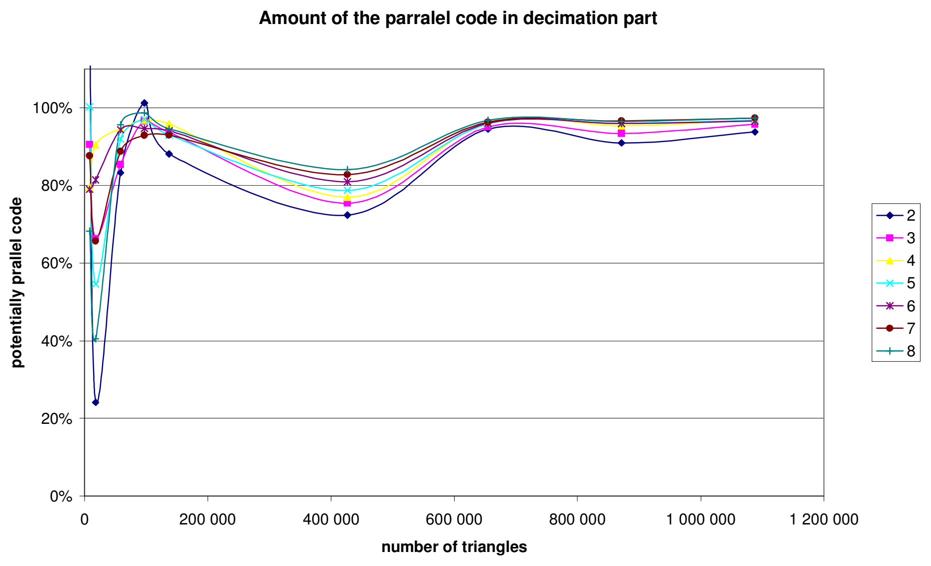
	Number of processors (threads) used							
	1	2	3	4	5	6	7	8
e	1	0.66	0.5	0.4	0.34	0.29	0.25	0.22
a	1	1.32	1.51	1.61	1.7	1.74	1.79	1.83
a_{teor}	1	1.32	1.51	1.61	1.7	1.74	1.79	1.83
p	X	0.49	0.51	0.50	0.51	0.51	0.51	0.52

Table 6-3: The experimental results and theoretical calculations according to Amdahl's law; computed for the Happy Buddha model.

Figure 6-4 shows the amount of potentially parallel code according to the number of triangles, for different number of processors used.



(a)



(b)

Figure 6-4: The amount of the parallel code for the whole algorithm (a) – the potentially parallel code according to Amdahl's law is approx. 51%, the amount of the parallel code for the decimation part (b).

Methods Comparison

Figure 6-5 shows the running time for 96% reduction for both mentioned approaches of vertices ordering. We can see that using the bucketing function we obtained the best running time in comparison to Quick Sort. It is necessary to point out that the methods using sorting algorithms were implemented in parallel while bucketing was sequential. The run time of "hash" function is faster than 8 processors running the method with sort algorithm.

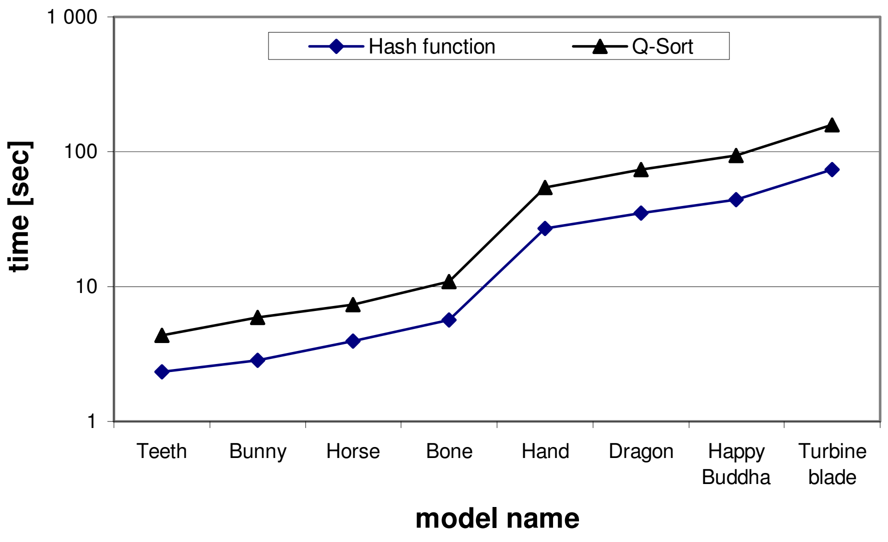


Figure 6-5: Achieved time comparison for two mentioned approaches. Presented Q-Sort time was achieved running on 8 processors, while bucketing function time is valid for sequential run on one processor.

Unfortunately the results are not comparable with other known algorithms, due to the different platforms. To make the results roughly comparable at least, we use the official benchmarks presented by SPEC as shown in Table 6-4, where η presents the superiority of the DELL computer against the SGI. Table 6-5 presents our results according to results obtained recently by other algorithms, taking the ratio η into the consideration.

benchmark test / machine	SGI R10000	DELL 410 Precision	η (DELL/SGI)
SPECfp95	8.77	13.1	1.49
SPECint95	10.1	17.6	1.74

Table 6-4: Benchmark test presented by Standard Performance Evaluation Corporation.

algorithm	time for a reduction from 69.451 to 1.000 triangles [sec]
Proposed algorithm	4,1 * η = 4,1* 1,49 = **6,11**
Garland [10]	10,40
Lindstrom & Turk [31]	2585,00
Hoppe [20]	500,00
JADE [3]	325,00

Table 6-5: Rough comparison of running-times of reduction of the Bunny model.

It is obvious that our algorithm is really fast. However, to be able to make a full comparison of these methods it is important to consider the approximation quality as well. Unfortunately, this kind of information is usually somehow hidden in majority of papers related to mesh simplification problematic.

Approximation quality

As presented in detail in Chapter 3, several error metrics can be used to measure the quality of approximation. The most frequently used approach is to compute a geometric error using E_{avg} metric (6.5, 6.6) derived from Hausdorff distance:

$$E_{avg}(M_1, M_2) = \frac{1}{k_1 + k_2}\left(\sum_{v \in X_1} d_v^2(M_2) + \sum_{v \in X_2} d_v^2(M_1)\right),\ d_v(M) = \min_{w \in P(M)} \|v - w\| \quad (6.5, 6.6)$$

where M_1 and M_2 are original and reduced model, k_1 and k_2 are numbers of vertices on each model, X_1 and X_2 are subsets of vertices in M_1, M_2.

As our method keeps the subset of original vertices, we use more simple formula (6.7):

$$E_{avg}(M_1, M_2) = \frac{1}{k_1} \sum_{v \in X_1} d_v^2(M_2), \quad X_1 \subset P(M_1) \qquad (6.7)$$

where k_1 is original number of vertices, $P(M_1)$ is a set of original vertices and $dv(M_2)$ is the distance between original and reduced set of vertices.

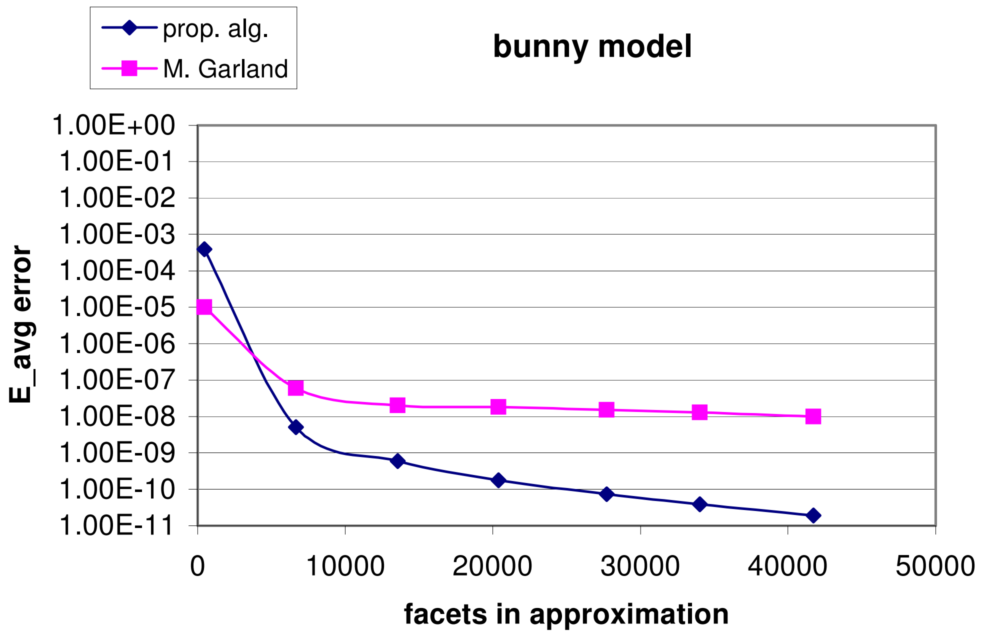


Figure 6-6: Approximation error comparison.

Figure 6-6 presents a comparison of error measurement of the proposed algorithm and M. Garland's method [10]. However, such error evaluation is not very accurate, because every two models can act in a completely different way.

If we compare the two above mentioned methods, we will find that the error values are almost the same. It is also hard to say which method gives the best results, because for each model we get different behaviour of the error. Examples of reduced models are presented on Figure 6-7, Figure 6-8 and Figure 6-9.

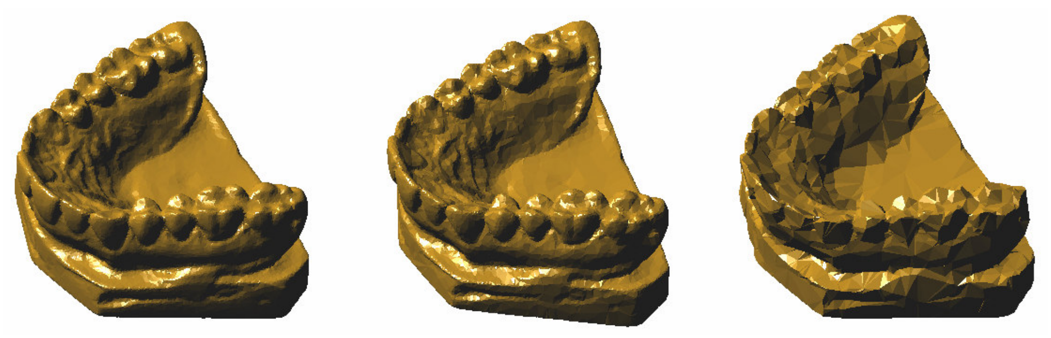


Figure 6-7: A teeth model (courtesy Cyberware) at different resolutions; the original model with 58,328 triangles on the left, reduced to approx. 29,000 triangles in the middle and 6,000 triangles approximation on the right.

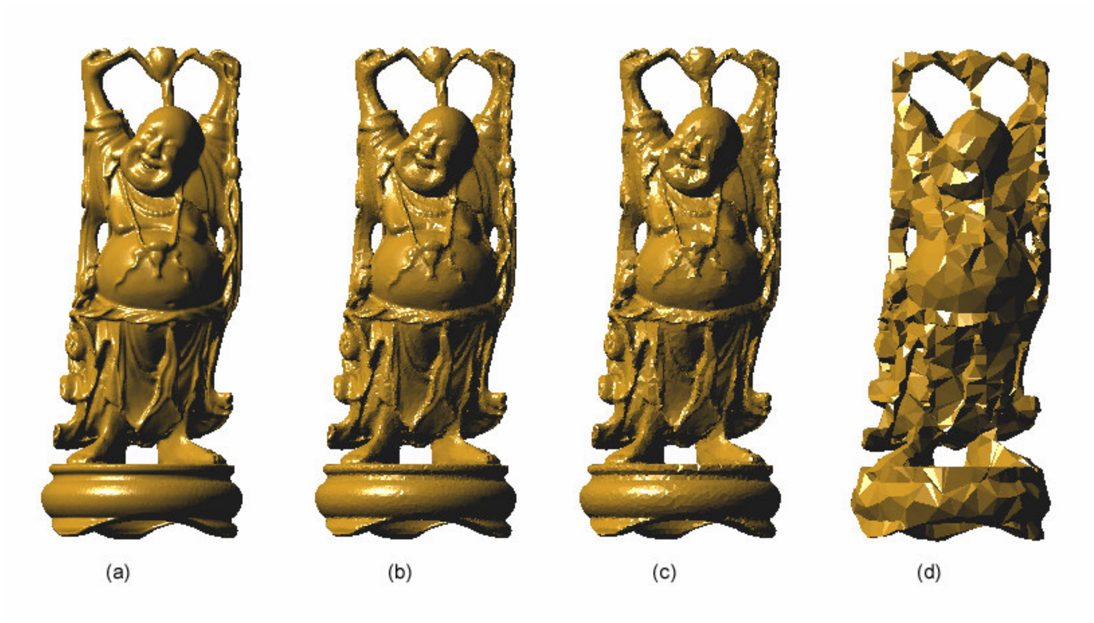


Figure 6-8: The Happy Buddha model (courtesy GaTech) at different resolutions; the original model with 1,087,716 triangles (a), reduced to 105,588 triangles (b), 52,586 triangles (c), 10,974 triangles (d).

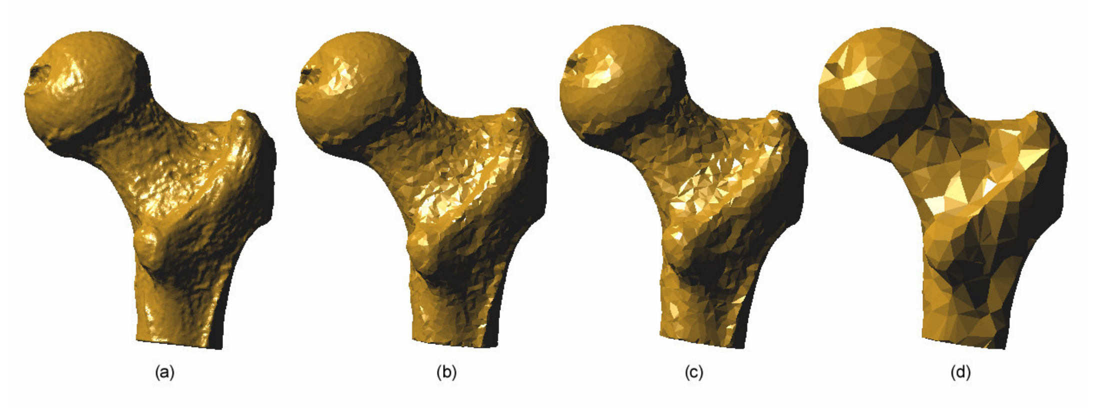


Figure 6-9: A bone model (courtesy Cyberware) at different resolutions; the original model with 137,072 triangles (a), reduced to13,706 triangles (b), 6,854 triangles (c), 1,248 triangles (d).

6.5 Conclusion

We have described our superior algorithm for the simplification of triangular meshes, which is capable of producing good quality approximations of polygonal models in uncontested running times.

The algorithm combines Schroeder's vertex decimation and edge contraction methods. In the outer optimization loop vertex importance is evaluated based on decimation criteria, while inner optimization loop uses edge collapse operator to find best fitting contraction, considering even some attribute values, and to provide retriangulation in a short time. We have also introduced a bucketing function, which can be used instead of expensive vertices sorting. Our algorithm has proved its high speed and simplicity. It can be run in parallel and is best suitable for fast preview providing satisfactory level of detail.

However, there are many problems connected with vertex collapse operator used in decimation in general. At first, it is necessary to explicitly define the sharp edges. Sharp edge is such an edge, where the angle between the two adjacent triangles is less than the specific threshold or the edge is required by the application. To set the threshold some experience of the user is required and such a threshold may be different for different models. The main disadvantage of vertex decimation methods is that they are not able to preserve a volume, since they produce shrinkage of the reduced model (assuming closed surfaces with a majority of convex vertices). Figure 6-10 illustrates the situation in 2D. The more the model is reduced, then the smaller is its volume, or area respectively. Unfortunately there is no way how to avoid this effect. Therefore methods based on edge collapse seem to be more promising from this point of view.

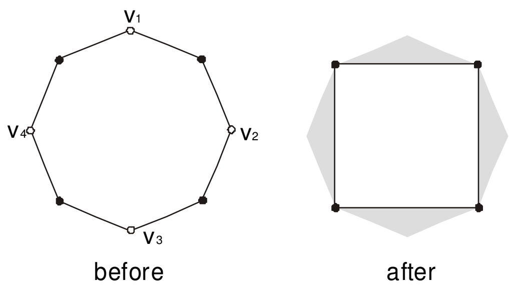


Figure 6-10: Shrinkage caused by the removal of vertices $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ and $\mathbf{v}_4$.

Moreover, edge contraction methods offer intuitive techniques for eliminating approximation error by optimal positioning of a new vertex after performing an edge collapse. This issue has been also adressed by Taubin, see [50] for details.

Chapter 7

Similarity of Appearance

At this chapter we will present another approach that looks at the simplification problem from a slightly different point of view. Although a geometrical distance can tell a lot about model similarity in space, for a similar visual appearance we would rather follow some other criteria.

The human visual system allocates different amounts of processing resources to different portions of the visual field which provides a trade-off between resources and time. On the one hand, attention can be shifted to a new location through a saccadic eye movement. On the other hand, the photoreceptor density that decreases between the fovea and the periphery induces no-uniform processing capability over the entire field. In fact, the conclusion is still more surprising: features will only be perceived if they success in attracting attention [8]. A great deal of biological vision research has addressed a problem of defining such features. Even thought many biological studies helped shape computational models, we focus on feature perception in computer vision.

The proposed approach is based on detection of the main features of original model and applying few heuristic rules tries to keep these features over whole simplification process.

7.1 Vertex estimation – feature detection

According to the notes in previous section, in the following, we consider as a feature either an extreme vertex (a peak) or a sharp edge (two or more extreme vertices). Thus features detection is naturally based on vertex evaluation. Since a lot of oversampled models that need to be simplified are produced by 3D scanners, one can argue that it is the input vertices that are the true data to be preserved rather than the input surface. Thus we do not study any properties of edges or triangles as they are defined in the model. We suppose that these elements (edges and triangles) are derived from original set of points anyway. Although we detect feature vertices, we of course mainly search for non-important vertices – vertices to be easily removed with a minimal harm on model's appearance. The best candidates for removal are the least important vertices being part of planar regions of the mesh.

We have studied 4 approaches how to evaluate vertex importance.

Average plane distance

First method was based on evaluating vertex property according to previously used distance from an average plane. The plane is given by vertices adjacent to evaluated vertex v. This is the same technique as described in section 5.2. The higher the distance is the more important is the vertex in a model. Vertices with high values are good candidates to be marked as feature vertices. Vertices with near-to-zero distance can be removed.

Gaussian curvature

Since we mostly search for planar regions, we also did several experiments using Gaussian and mean curvature estimation of the surface [47]. Because of our focus on flat areas and the fact that we search for vertex pair with the same evaluation, the Gaussian curvature only was sufficient.

$$K = \frac{2\pi - \sum_{i=1}^{m} \alpha_i}{\frac{1}{3} \sum_{i=1}^{m} A_i} \tag{7.1}$$

The curvature K is given by the equation (7.1), where i goes over all neighbours of evaluated vertex, α is the vertex angle in each of neighbouring triangle and A means the area of neighbouring triangles.

Note that we are searching for single vertex property only, thus we do not need to classify the mesh geometry exactly.

Volume estimation

Another widely used criterion in mesh simplification [31] is based on underlying condition to keep the volume of the original model. In this approach a vertex importance is related to the volume of the mesh below the vertex (part of the mesh given by adjacent triangles). For each vertex and its neighbourhood we introduce a new vertex $\mathbf{v}_v$, given as an average point of all vertices adjacent to the vertex in question $\mathbf{v}_0$.

$$\vec{v}_v = \frac{\sum_{i=1}^{m} \vec{v}_i}{m} \tag{7.2}$$

Having this *virtual* vertex **v**$_v$ we compute the a of volumes of tetrahedral **v**$_0$,**v**$_v$,**v**$_i$,**v**$_j$, where vv is the virtual vertex and **v**$_0$,**v**$_i$,**v**$_j$ are vertices of triangles in our triangulation. See Figure 7-1.

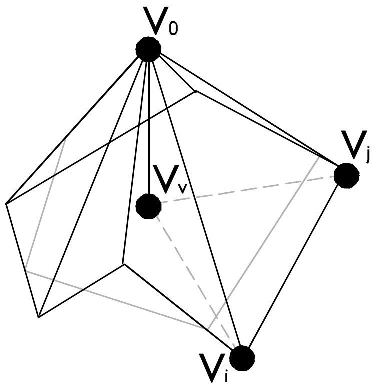


Figure 7-1: Vertex related volume estimation.

The resulting volume value *V* is weighted by the longest edge going out from vertex **v**$_0$ to somehow normalize the values over the whole mesh. The volume has been evaluated according to following formula (7.3, 7.4).

$$D_k = \begin{vmatrix} x_v & y_v & z_v \\ x_i & y_i & z_i \\ x_j & y_j & z_j \end{vmatrix}, \quad V = \frac{1}{6}\sum_{k=1}^{m} |D_k|, \tag{7.3, 7.4}$$

where D_k is the volume of one tetrahedron and *k* goes over all tetrahedrons related to vertex **v**$_0$ which is supposed to lay in the origin.

Average normal vector

The last and in some way straight forward method evaluates vertices by estimating a normal vector in vertex **v**$_0$. The normal vector n is computed as an average normal of all triangles adjacent to the vertex **v**$_0$, see Figure 7-2.

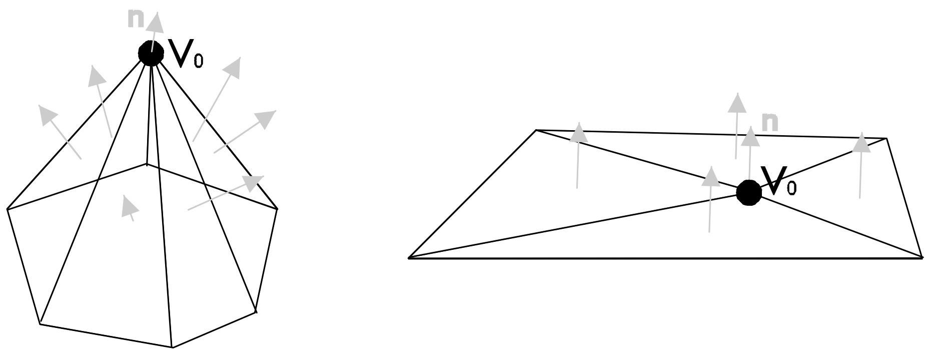


Figure 7-2: Average normal vector.

We used the easiest way of computation, which is not-weighted average, see equation (7.5).

$$N = \frac{\sum_{i=1}^{m} n_i}{m} \tag{7.5}$$

Note that the normals of triangles have unit length. The importance value is the inverse of the length of the normal. The more the normal length is closer to 1 the more flat area is around the vertex in question. Naturally, if all the neighbouring triangles have their normals in the same direction, the area of the triangle fan is flat and the length of resulting normal will be equal to 1. The more the vertex represents a peak in a mesh the less will be the resulting normal length, since each of partial normals points to different direction.

7.2 Final algorithm

Evaluation results

Studying the results of presented evaluation, we have decided to use the average normal for vertex importance estimation (feature detection). On Figure 7-3 you can see the example of cow model with 50% most important vertices highlighted according to all methods presented. Since all the pictures show exactly 50% most important vertices, it is obvious, that the average normal estimation (top left) gives the best results showing the main features of the model. As you can see, it is a kind of caricature, where the most important contours are highlighted (horns, ears, eyes, neck, and legs). The Gaussian curvature estimation (top right) also gives good results which could be even better with combination of mean curvature to detect sharp edges too instead of peak points only. However, the computation would be too time-consuming. Average distance evaluation (bottom left) tends to involve the sharp edges too.

On the other hand it misses the details kept by small triangles in areas such as eyes and also highlights which could be supposed not to be very important such as a belly. Probably the worst result gives the estimation of tetrahedrons volume, which was anyway more or less experimental.

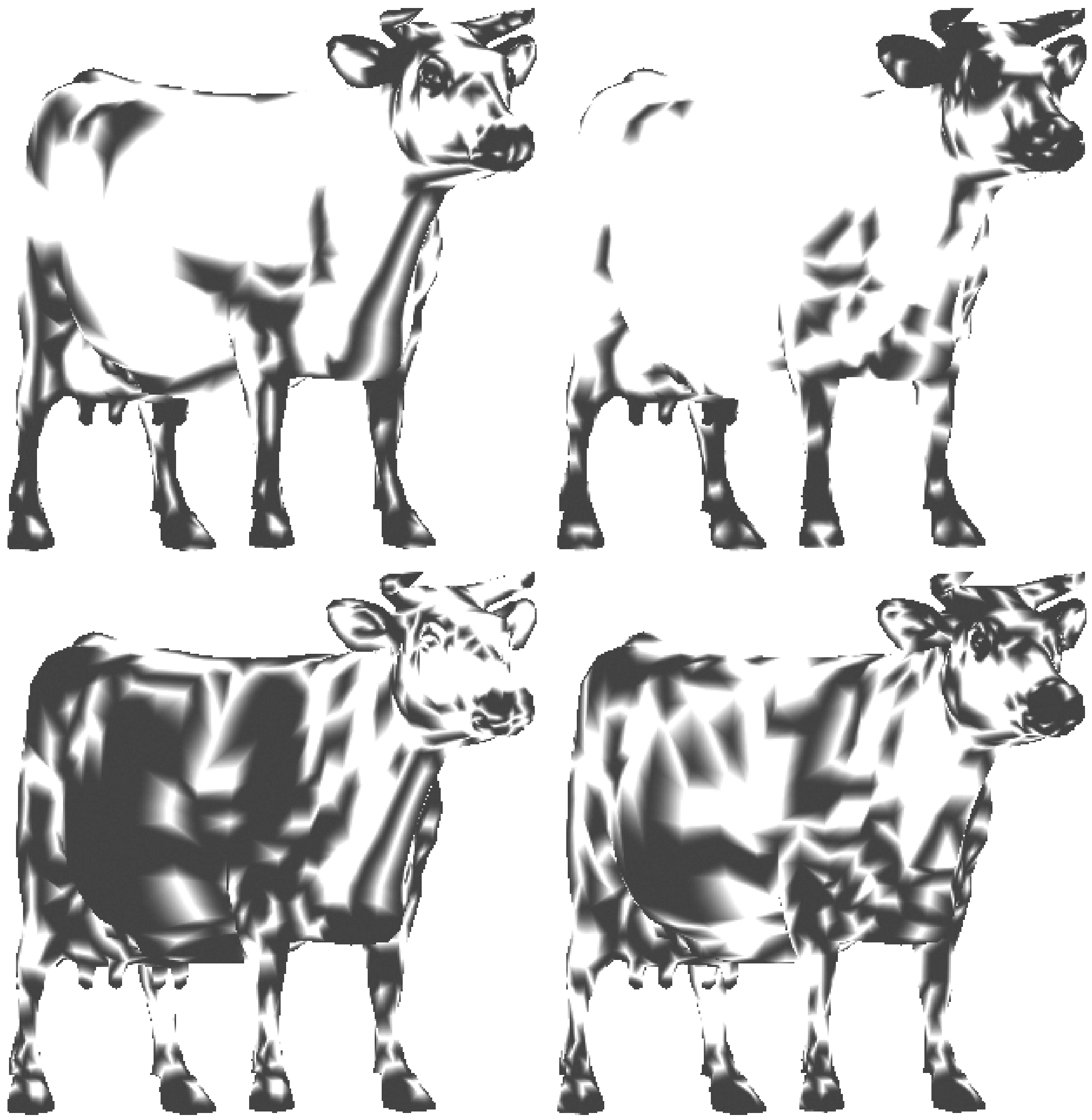


Figure 7-3: 50% important vertices of cow model according to 4 different evaluation used – average normal (top left), Gaussian curvature (top right), average plane distance (bottom left), tetrahedral volume (bottom right).

The graph on Figure 7-4 shows the rate of vertices and their importance. The picture says that from all the number of vertices (approx. 3000) there were about 2000 vertices with importance lower then 0.5. It's obvious, that all the importance evaluation methods act in the similar way and the majority of vertices have a low importance (tests have been performed on several models naturally). Again, the average normal vector evaluation gives the most wanted results - declaring the majority of vertices as non-important (the lowest line).

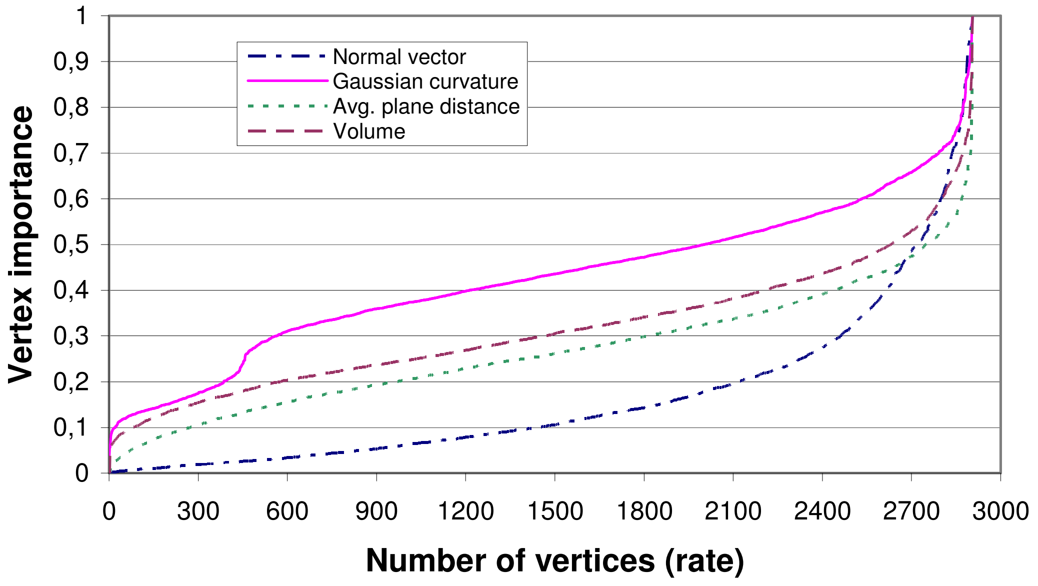


Figure 7-4: Vertex evaluation according to the method used in a model of a cow.

After several experiments with the average normal computation (not weighted, weighted by triangle area), we concluded to the evaluation, where each normal is weighted by the apical angle of given triangle. The precise formula can be seen on equation (7.6).

$$N = \frac{\sum_{i=1}^{m} \alpha_i n_i}{\sum_{i=1}^{m} \alpha_i} \tag{7.6}$$

where m is the number of neighbouring triangles and α_i is the apical angle of i-th triangle at the vertex (see Figure 7-5). Note that the resulting normal vector N is not normalized.

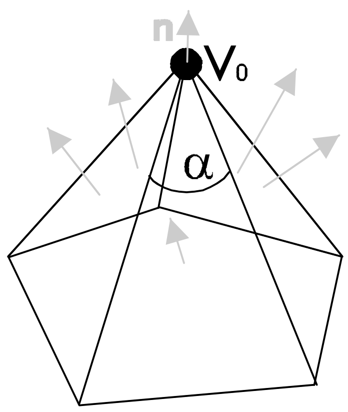


Figure 7-5: Normal vector weighted by apical angle α for each of adjacent triangle.

This approach produces best quality evaluation which is independent on tessellation at the vertex (see Figure 7-6). In case of non-weighted normal, the resulting normal vector will be different in example on the left (the other two will be the same and the normal will have a direction more to the front). If the normal vector would be weighted by triangle area, the first two examples will have the same normal, but the third one will be different (since the area is smaller). Only the apical angle weight will give us the same results for all three cases.

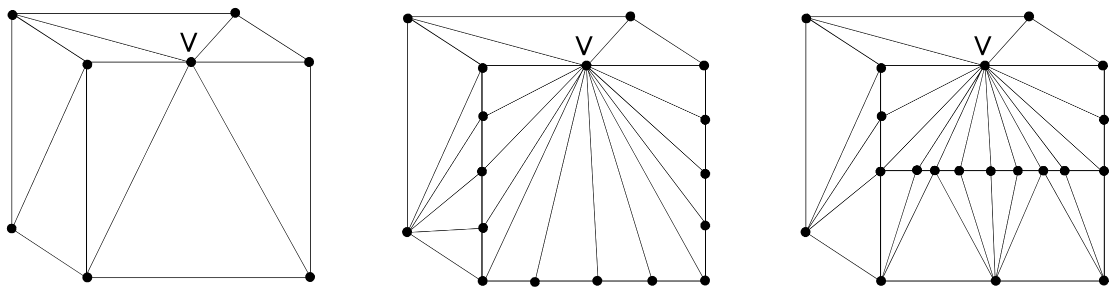


Figure 7-6: Three examples of vertex neighbourhood.

Having evaluated all the vertices we can sort them into priority queue according to their importance. The least important vertices are the best candidates for removal. Since vertex removal followed by re-triangulation is neither trivial in 3D nor natural when ignoring original edges, we perform and edge contraction instead. For a given vertex the adjacent edges are investigated and the best-fitting one is replaced by a new vertex. By this step we get a correct triangulation and are able to follow some other criteria on a quality of resulting mesh.

Best-fitting edge estimation

As already said, once we have a vertex candidate for the removal, we need to search for the best edge to perform the contraction. The chosen edge will be replaced by a new vertex of a specific position. Before we describe the edge estimation, we must first introduce an evaluation of new vertex position.

Since the aim of the method is to find a simplified approximation of the model with respect to the similarity of appearance, we do not subordinate the vertex position to any error estimation. We try to find such a vertex which would approximate the suppositional surface in between the two end-points of the edge, see Figure 7-7.

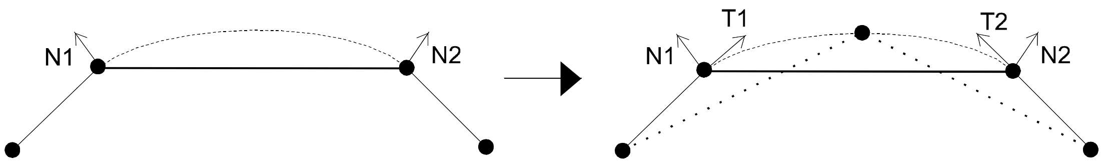


Figure 7-7: The new vertex should lie somewhere on the dashed line (in 2D).

Let's consider 2D case for clearer explanation. To determine a new vertex position we use a curve which approximates surface the way we show on Figure 7-7. At the beginning we have only two endpoints and non-normalized normals which determine vertices importance. We used a quadric curve with near least square acceleration, introduced originally to smooth the model contour by [2]. The nice thing about this curve is its invariance to tangent lengths. The tangents corresponding to a pair of normals at the vertices can be obtained by using the so-called Gram-Schmidt orthogonalization algorithm.

$$T1 = N2 - N1\,(N1 \cdot N2), \tag{7.7}$$

$$T2 = -N1 + N2(N1 \cdot N2). \tag{7.8}$$

Note, that the normals are assumed to be normalized. It should also be pointed out that when the angle between the normals, is zero or very close to zero, then we can not compute the tangent in this way. We use a linear interpolation on the edge instead. Let's have

$$P = P2 - P1 \tag{7.9}$$

and coefficients $\alpha, \beta, \alpha', \beta'$, where

$$\alpha = \frac{T_1T_2}{T_1T_1},\ \beta = \frac{P\,T_1}{T_1T_2}$$
$$\alpha' = \frac{T_1T_2}{T_2T_2},\ \beta' = \frac{P\,T_2}{T_1T_2} \qquad (7.10, 7.11, 7.12, 7.13)$$

we get a function

$$f(t) = \left(\frac{\alpha'\beta'T_2 - \alpha\beta T_1}{2}\right)t^2 + \left(P + \frac{\alpha\beta T_1 - \alpha'\beta'T_2}{2}\right)t + P_1 \qquad (7.14)$$

Thus the only question is how to choose the parameter t, which means, to where to put a new vertex on the curve. Since the curve itself is given by normals as well as the vertex importance, the natural way is to use a linear interpolation on the importance (length of the normal vectors). Thus, if the importance of vertices will be equal, the parameter t will be set to 0.5 (the new vertex will be placed in the middle of the curve).

Switching to 2.5D, we have a tool now how to place a new vertex somewhere "above" the contracted edge, instead of just somewhere in between the endpoints of the edge. In full 3D space we can also influence the vertex position by the properties of triangles adjacent to the edge, or their opposite corners respectively. We can use the quadric curve again and the only condition is that the area is not arrow-shaped, see Figure 7-8. In case of arrow-shaped quads we use only one curve constructed over the edge.

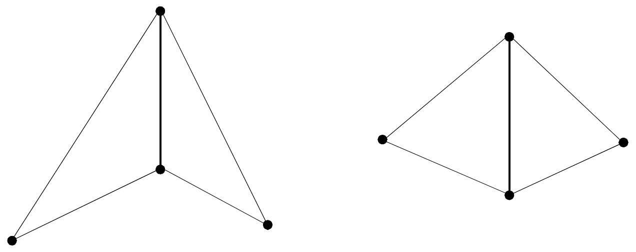


Figure 7-8: The arrow-shaped quad (left), the non-problematic case (right).

The parameter *t'* of the second curve (between opposite corners of two triangles adjacent to the edge) is again given by the importance of opposite corner vertices, but this time it is weighted by the inverse value of the distance of each vertex to the point estimated on the first curve. So, if the distance is equal, then only the importance matters. If the distance of one vertex is higher the less of its importance is considered. This condition forces to place the vertex in the position given by the importance of surrounding vertices and not the exact topology of the mesh. The final vertex $\mathbf{r}_v$ is placed just in the middle of the two vertices on both curves, see Figure 7-9.

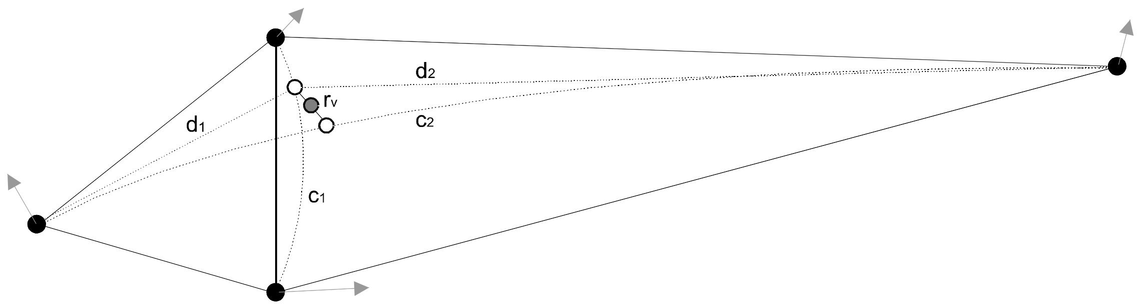


Figure 7-9: The construction of new vertex position.

Having described the new vertex position evaluation, we can get back to the procedure of choosing the best-fitting edge for the contraction. In our algorithm we take all the edges adjacent to given vertex (candidate for removal), and compute new vertex position for each edge. For every new vertex we examine affected mesh property like the difference between area of original and resulting triangles, or inconsistencies such as mesh folding or triangle degeneration caused. Since we primarily remove vertices on flat regions, we force the resulting mesh to be as flat as possible, thus minimal area of resulting triangles is prioritized. The edge with such a best evaluation is contracted for real. If there is no suitable edge to process, the removal is forbidden.

Upon a framework described above, we can present the proposed algorithm as follows

Init:
- go over all the vertices and compute their importance
- sort vertices according to the importance

Main loop:
- take the least important vertex
- for every adjacent edge
    - compute new vertex position for case of contraction

    - simulate the contraction and evaluate the quality of resulting mesh

- perform the contraction of best-fitting edge
- re-evaluate affected area
    - compute vertex importance
    - insert into a priority queue
- continue the main loop

The real implementation uses three *magic* parameters which are important for the resulting mesh and can be changed by user. The first parameter is an importance threshold. If vertex importance exceeds the value of the threshold, it is marked as extremely important one. These (feature) vertices are processed a bit different way than described above. The main difference is in estimation of parameter t, which is set either to 0 or 1 depending on which endpoint of an edge is extreme. If both of them are extreme, the more important vertex wins and its position is kept. This strategy leads to simplification which keeps the most important features represented by extreme vertices without any change from the original mesh. Such extreme vertices can be seen on Figure 7-10, for example at the neck. The lower is the threshold value the more features of the model will be kept and less simplification will be performed.

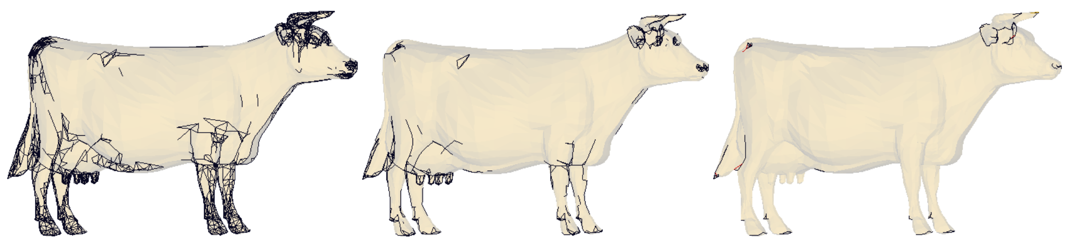


Figure 7-10: Model feature detection for (left) 50%, (middle) 75% and (righ) 90% simplification.

The second parameter is the maximal allowed angle between normals of triangles before and after edge contraction. This value helps to detect triangle folding and also controls the smoothness of the resulting mesh. The smaller is the angle the smoother is the resulting mesh – contractions producing not-wavy surfaces are preferred.

Third parameter is an angle between two adjacent triangles and helps to define so called flat edge. In general if the vertex selected for removal is extreme and one of its neighbourhood vertices is extreme as well, only the edge between these two vertices can be considered for removal. Applying this rule we can preserve sharp[3] edges. In this case a sharp edge is every edge that has extreme endpoints and is not a flat edge. In other words, we do not detect sharp edges studying the sharp angle between adjacent triangles. The algorithm marks the edge as sharp if both of its endpoints are extreme and the angle between adjacent triangles is less than the value given by our third magic parameter. If such angle is bigger (at most 180 degrees) the edge is marked as a flat edge and is prohibited from contraction, since it could dramatically change the shape represented by the mesh, see Figure 7-11.

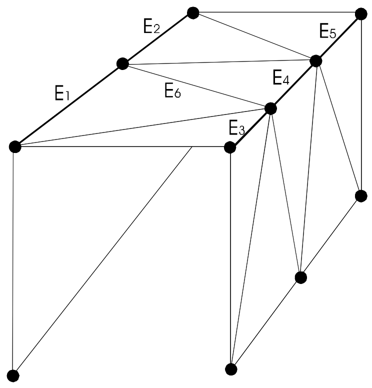


Figure 7-11: An example of sharp edge (E1-E5) and flat edge (E6).

In general two strategies can be used for simplification process - with or without memory of reduced vertex and affected area. The approach with memory initializes a counter of affected vertex during reduction and every time the vertex is marked as a candidate for removal the counter is decreased. The only vertices with a counter equal to zero can be considered for the reduction. Such a use of affected-vertices memory helps to distribute reduction process over whole surface and the resulting mesh has nicely shaped triangles. If the simplification runs without memory it can easily produce rapid-flat models, where flat regions are simplified in prior, see Figure 7-12.

[3] An edge, where the angle between its two adjacent triangles is lower than some specific value.

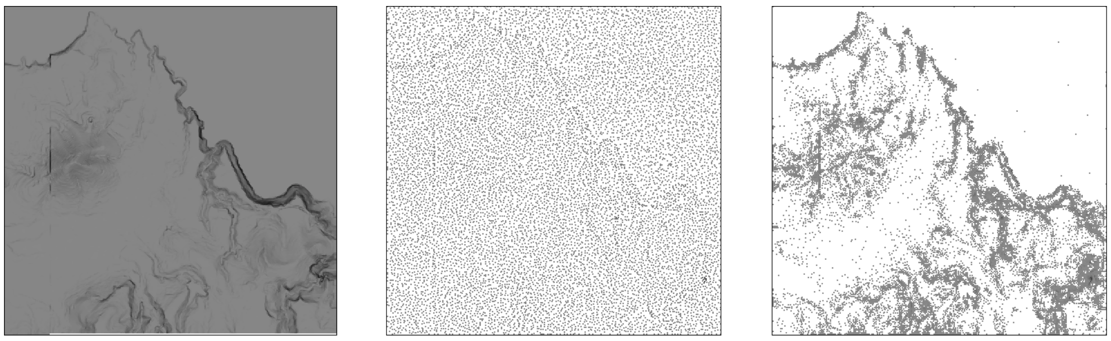


Figure 7-12: Points distribution (density of vertices from top-view) during simplification of a terrain model (left) with (middle) and without (right) vertex memory. The middle and left picture shows model after 80% simplification (20% of original data).

7.3 Results

The proposed method has been tested on several models, mostly from GaTech, Cyberware and Avalon depositories. Table 7-1 shows some fundamental information about models on which we will present obtained results. All the experiments were performed on Intergraph TDZ2000 400MHz Pentium II with 512MB RAM, running on WindowsXP.

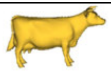
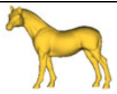
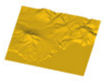
name	cow	fandisk	teeth	bunny	horse	bone	terrain	dragon
# vertices	2,905	6,475	29,166	35,947	48,485	60,537	65,829	437,645
# triangles	5,804	12,946	58,328	69,451	96,966	137,072	130,630	871,414
picture								

Table 7-1: Models used for presented results.

Table 7-2 shows the running times of 80% reduction. It is obvious that rapid-flat method (approach without vertex memory) is faster but the resulting mesh contains long and thin triangles. On the other hand the approach with vertex memory produces nicely shaped triangles but the running times are slightly worse.

name	cow	fandisk	teeth	bunny	horse	bone	terrain	dragon
mem	1.244	4.604	11.700	13.304	21.032	26.116	31.728	141.980
no mem	1.160	4.116	10.120	12.320	19.848	24.008	28.872	131.804

Table 7-2: Obtained timing [sec] for 80% reduction. Thresholds have been set to mark 15% vertices as extreme.

On Figure 7-13 you can see the resulting meshes of both methods for fandisk model. However, at the most drastical reduction (99% and more) the resulting meshes are similar for both, with and without memory, approaches.

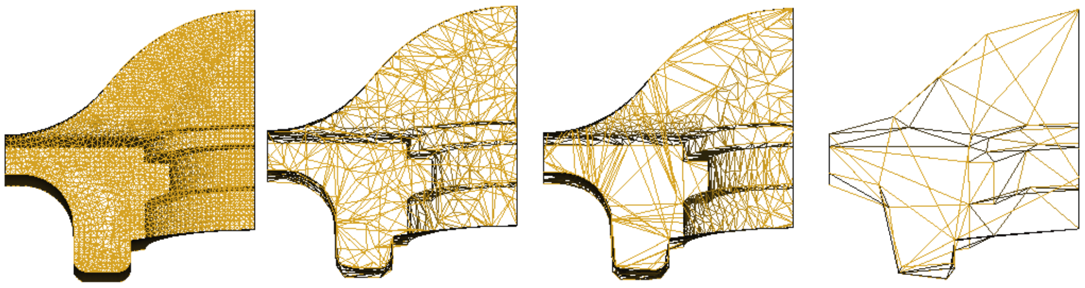
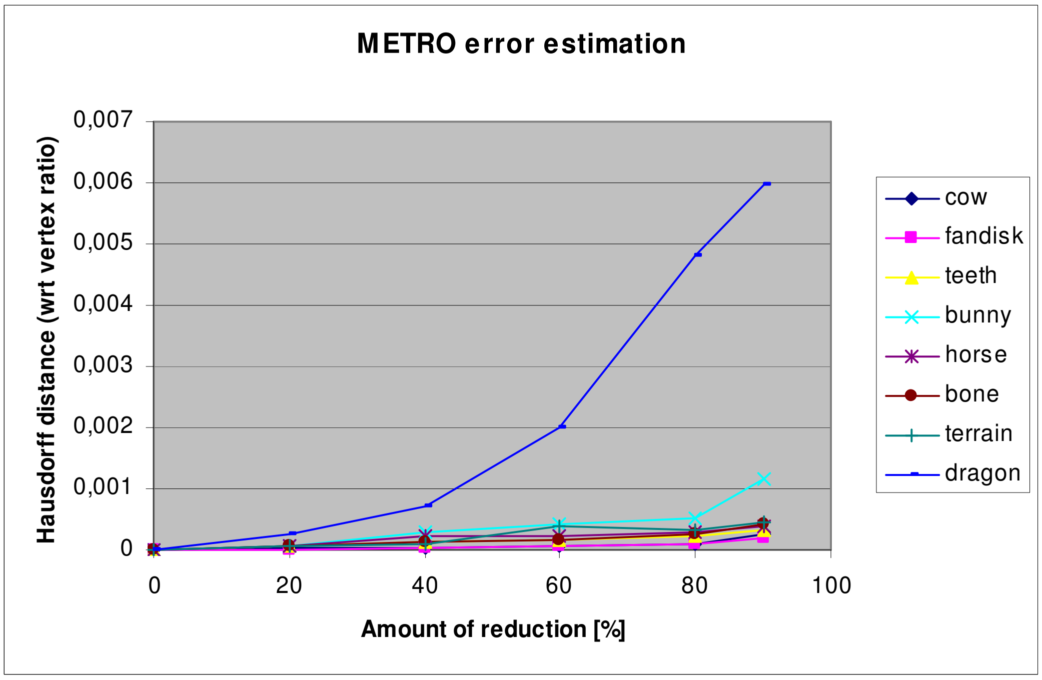


Figure 7-13: Example of reduced model. The original mesh (left), 90% reduction with and without vertex memory (two in the middle) and drastical 99% reduction (right).

On Figure 7-14 you can see graphs of error estimation for several models during simplification process. The models have been simplified from 0% up to 90%. The results are taken from METRO ver. 4.05 [5], using default values (vertex, edge and face sampling enabled, Monte Carlo sampling, 10times more samples than triangles in a mesh). To have all the values comparable, the METRO results were taken with respect to Dragon model, thus re-computed using following formula (7.15):

$$E_r = E_M \frac{V_c}{V_{\max}} \tag{7.15}$$

where E_M is the value evaluated by METRO, V_c is the number of vertices of current model in certain level of detail and V_{max} is the number of vertices of Dragon model, which is the maximum number of vertices for certain LOD.



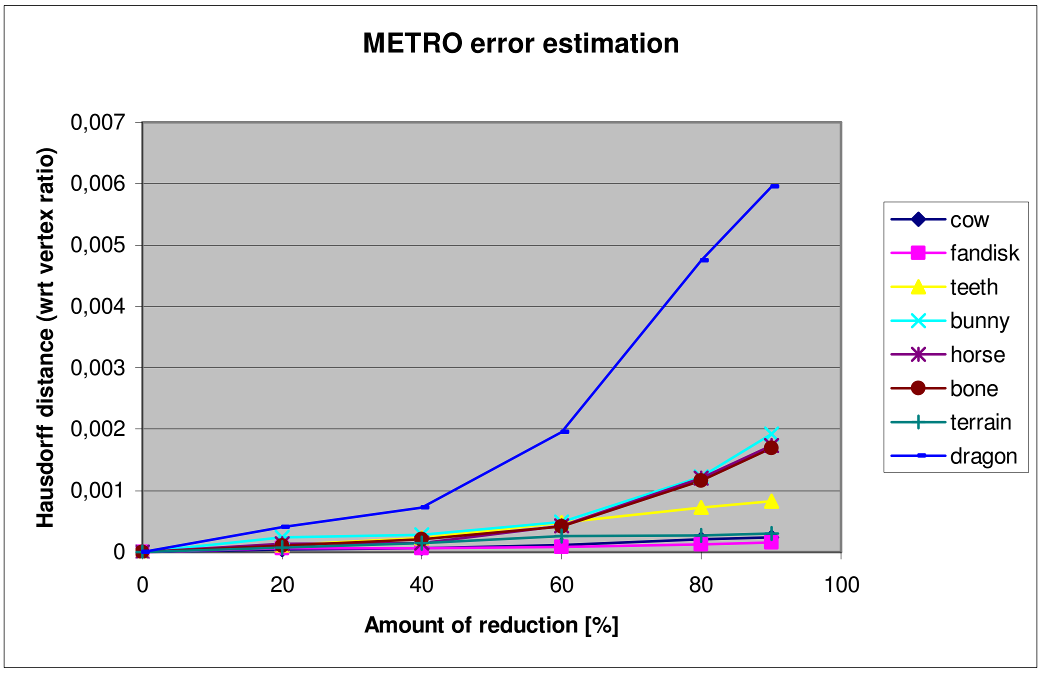


Figure 7-14: Approximation error for certain LOD for approaches with (upper) and without (lower) memory.

It is obvious that memoryless approach gives worse result in meaning of the Hausdorff distance. However, vertices distribution follows ones assumption that flat regions needs to be built from much less number of vertices than rugged surface. Here is noticeable difference between geometrical and perceptive evaluation of the approximation quality.

Also the oversampled models such as dragon, bunny and bone have error values higher than other datasets. Although, the values are higher than other simplification methods, it must be pointed out that METRO computes the error based on the Hausdorff distance which is not considered during a simplification in this case. The main goal of presented algorithm is to keep the similarity of appearance. However, the geometrical error is also important in mesh simplification to be able to compare the results with other methods. In Table 7-3 there are outputs of METRO in detail for cow, bunny and dragon models.

name	reduction	vertices	faces	area	bbox diag.	H-dist
dragon	0%	437645	871414	0.1452	0.266905	0.005964
	90%	41603	81808	0.1446	0.266801	
bunny	0%	35947	69451	0.1143	0.250246	0.022444
	90%	3824	5368	0.1125	0.249250	
Cow	0%	2905	5804	2.1802	1.271114	0.032040
	90%	391	776	2.0851	1.267350	

Table 7-3: METRO details for chosen models.

7.4 Conclusion

A new approach for triangular mesh simplification with respect to similarity of appearance was presented. This original method is based on vertex importance evaluation to select the least important vertex to be removed from the mesh. This evaluation uses vertex average normal vector which lowest values concern specific model features to be kept in approximations. Simplification itself is performed as an edge collapse where a new vertex position is evaluated with respect to supposed surface of the original object given by the endpoints of the edge, the normal vector at these points and opposite corners of adjacent triangles.

We showed that geometrical error does not have to be the only criterion of approximation quality and that a visual appearance can lead to opposite observation. This could be quite important in application such as computer games, 3DTV and other multimedia where mathematical precision is not a principal value. Conversely, preserving main visual features is more relevant.

Chapter 8

Conclusion

We have described a method for the automatic simplification of highly detailed polygonal surface models into faithful approximations containing fewer polygons. The empirical tests demonstrate that presented simplification methods are of very high speed while keeping a reasonable quality of the resulting approximations, see Figure 8-1.

We have also shown how to run simplification algorithm in parallel without a need of critical sections.

An original approach for triangular mesh simplification with respect to similarity of appearance was presented. Based on edge collapse operator this algorithm introduces the way how to compute an optimal position of resulting vertex after each edge contraction.

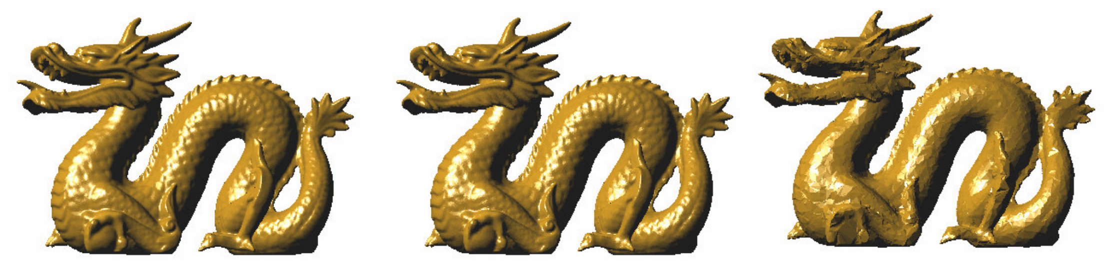


Figure 8-1: A dragon model (courtesy GaTech) at different resolutions; the original model with 871,414 triangles on the left, reduced to approx. 430,000 triangles in the middle and 87,000 triangles approximation on the right.

8.1 Summary of Contributions

To review, the primary contributions of my work as described in this dissertation are:

- **Super-independent set of vertices**. We have defined a new criterion how to choose vertices as candidates for removal during simplification process. It is based on independent set of vertices [21] with more strict constraints to vertices neighbourhood. The use of super-independent set of vertices leads to ability to have a parallel code without critical sections in simplification process.

- **Surface Simplification Algorithm**. By combining several approaches of mesh simplification and principles of super-independent set of vertices, we have developed a fast parallel algorithm capable to produce high-quality approximations of polygonal surfaces. This algorithm can simplify both manifold and non-manifold models. It's robust, very fast and accurate while preserving a mesh topology. Since the algorithm keeps the subset of original vertices, in addition to producing single approximations, it can be also used to generate multiresolution representations such as progressive meshes and vertex hierarchies for view-dependent refinement.
- **Edge classification and introduction of new vertex position**. Finally, we have introduced an original approach of edge evaluation and classification, which results in a new simplification algorithm. This algorithm mainly preserves the visual appearance by detecting and keeping important features of the original model such as sharp edges or high detail regions during even drastic simplification. While we suppose that original surface tends to be curved according to its vertex normals, a new vertex position is determined to lay on such supposed surface using near least-square curvatures.

8.2 Future Work

As we have a simplified model we are still able to compute back the curves on which approximately original vertices lied. This is very interesting and leads to an idea of mesh refinement. With some effort we should be able to refine simplified model even without the knowledge of simplification process or exact position of original vertices. A future work is to design such refinement method which would be capable of displaying a complex model just from a base mesh. Along a base mesh it will be necessary to provide some additional information as well, which will be the main subject of the further research.

Such method would provide an easy way of transferring simplified triangular meshes through the network and fine rendering on client's side using several mathematical operations.

Appendix A

Coloured pictures

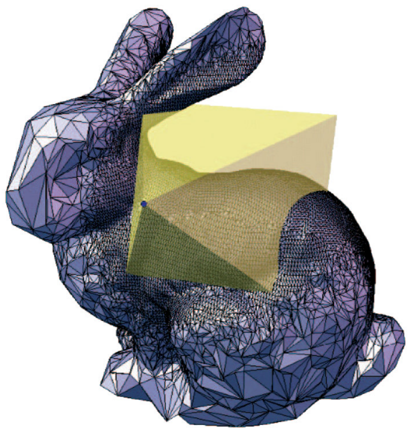


Figure 1-3, page 11.

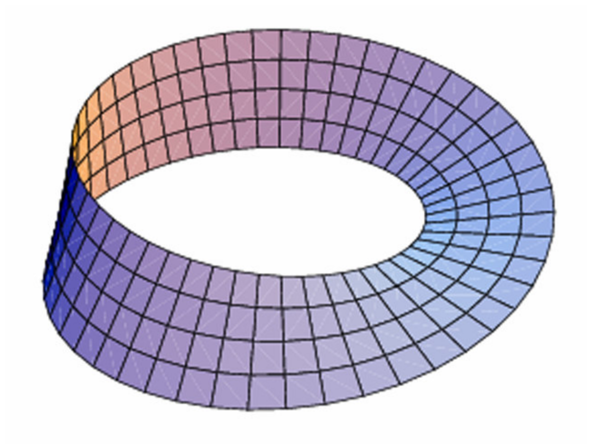


Figure 2-6, page 21.

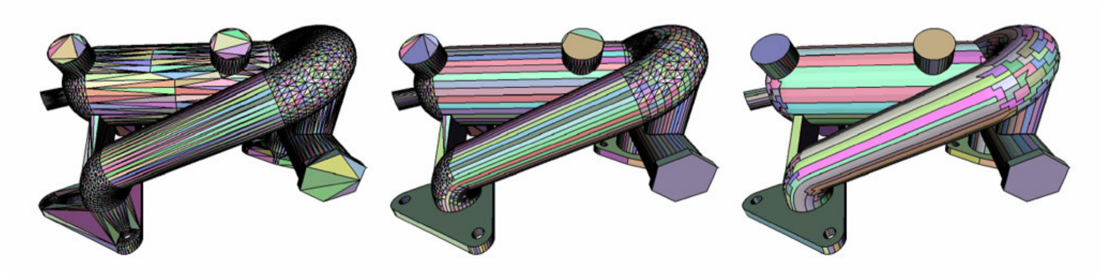


Figure 2-11, page 28.

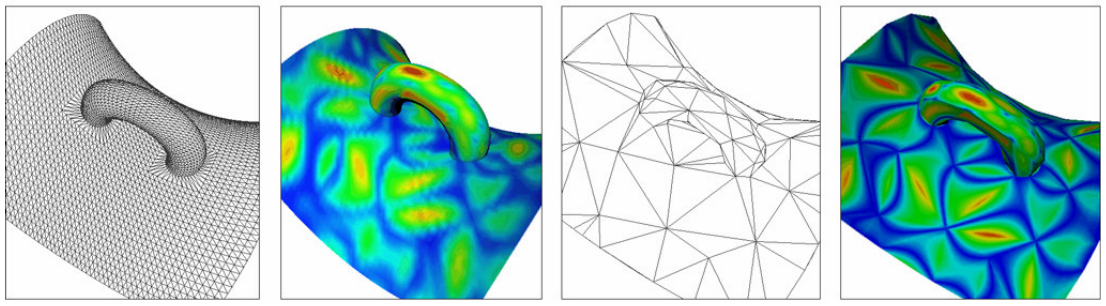


Figure 3-3, page 46.

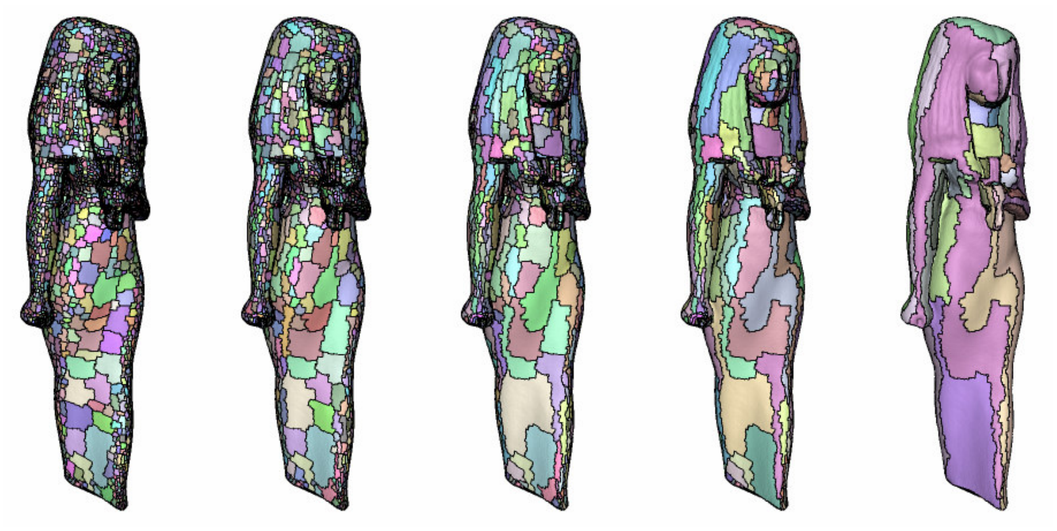


Figure 4-5, page 56.

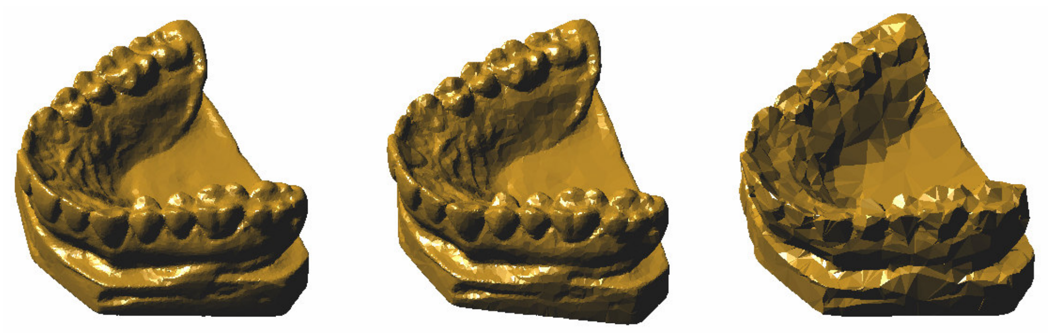


Figure 6-7, page 94.

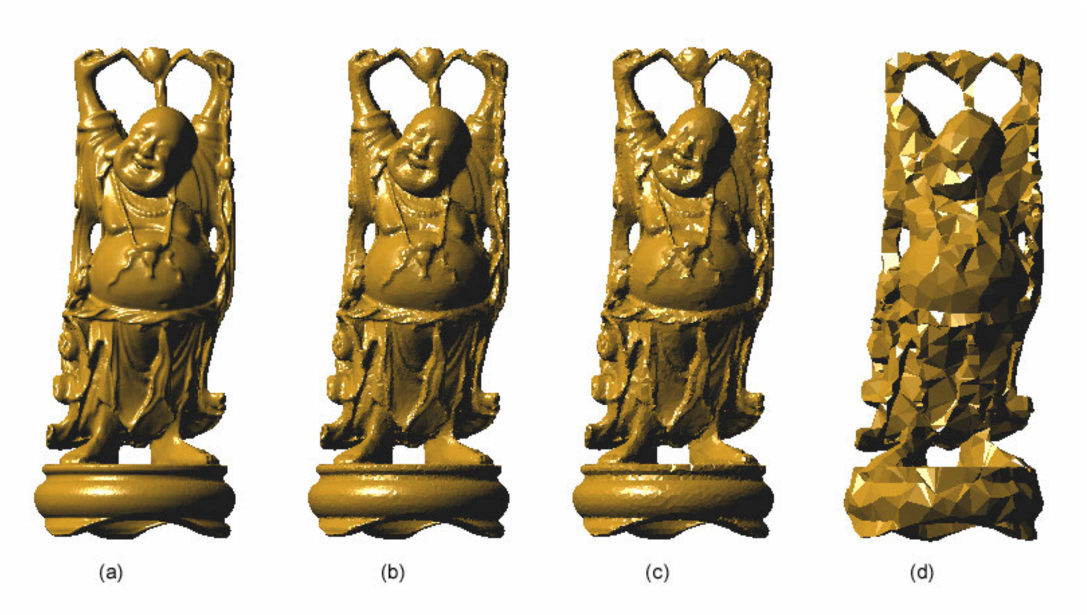


Figure 6-8, page 95.

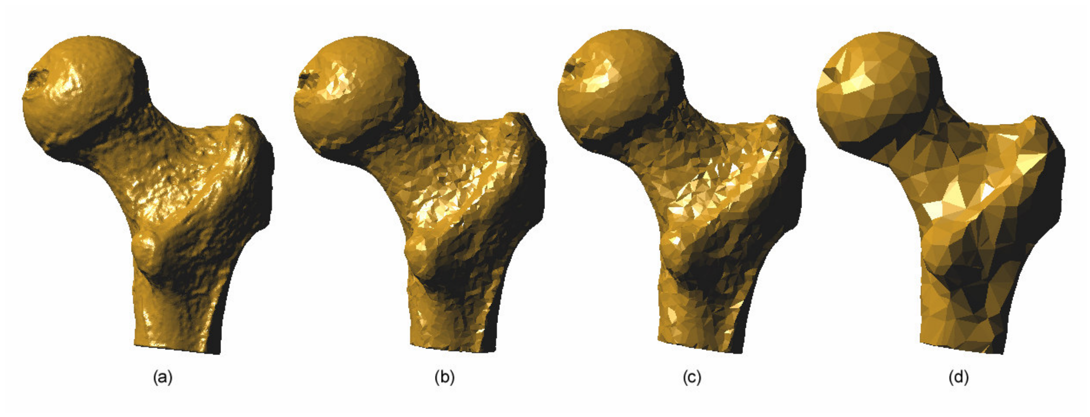


Figure 6-9, page 95.

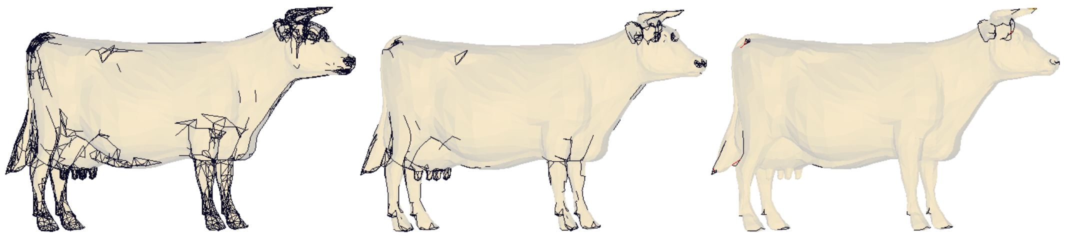


Figure 7-10, page 107.

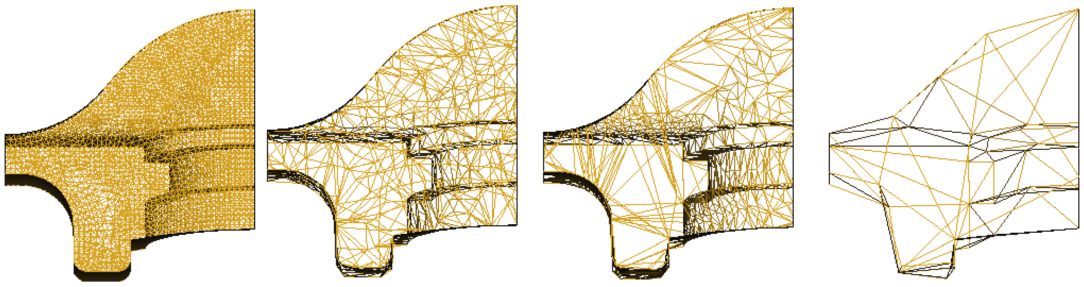


Figure 7-13, page 110.

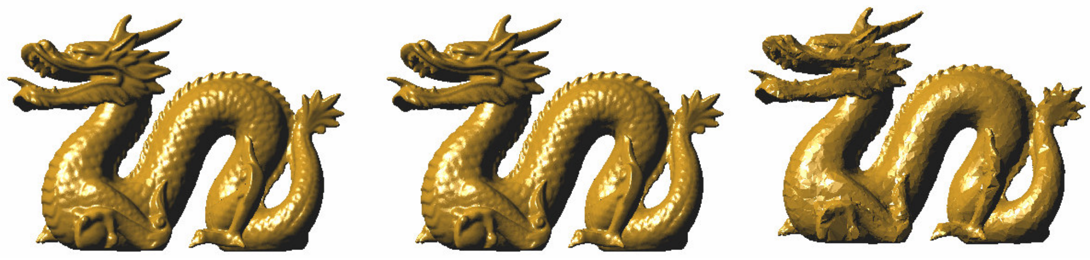


Figure 8-1, page 113.

Appendix B

List of Publications

[i] Franc M., Skala V.: Mesh simplification with respect to a model appearance. Proceedings Spring Conference on Computer Graphics 2006, ISBN 80-223-2175-3. Comenius University Bratislava, pp. 136-143, 2006.

[ii] Franc M., Skala V.: Fast Algorithm for Triangular Mesh Simplification Based on Vertex Decimation. Springer-Verlag Lecture Notes, CG&GM2002 Proceedings, Amsterdam, The Netherlands, April 2002.

[iii] Franc M., Skala V.: Parallel Triangular Mesh Decimation Without Sorting. In SCCG IEEE proceedings, ISBN 0-7695-1215-1, Los Alamitos, USA, pp.22-29, 2001.

[iv] Franc M., Skala V.: Parallel Triangular Mesh Decimation Without Sorting. In SCCG 2001 Conference Proceedings, Comenius University Bratislava, Slovakia, ISBN 80-223-1606-7, pp.69-75, 2001.

[v] Franc M., Skala V.: Triangular Mesh Decimation In Parallel Environment. EUROGRAPHICS Workshop on Computer Graphics and Visualization, Girona, Spain, pp.39-52, ISBN 84-8458-025-3.

[vi] Franc M., Skala V.: Parallel Triangular Mesh Reduction. In Proceedings of International Conference on Scientific Computing Algoritmy 2000, Slovakia, pp.357-367, ISBN 80-227-1391-0.

[vii] Franc M., Skala V.: Parallel Triangular Mesh Decimation. In Proceedings of International Conference SCCG'2000, Budmerice, Slovakia, pp.164-171, ISBN 80-223-1486-2.

[viii] Franc M.: Triangular Mesh Simplification Methods. MSc Thesis (in Czech), University of West Bohemia in Pilsen, May 2000 (supervisor: Skala V.).

[ix] Franc M.: Methods for Polygonal Mesh Simplification. State of the Art and Concept of Doctoral Thesis, Technical Report No. DCSE/TR-2002-01, University of West Bohemia, Plzen, Czech Republic, January 2002.

Appendix C

Stays and Lectures Abroad

Stays:

12.2.1999 – 28.5.1999	University of Girona, Spain
19.5.2001 – 27.5.2001	Univesity of Maribor, Slovenia
15.6.2001 – 28.6.2001	University of Ioannina, Greece

Lectures:

24.5.2001	Trinagular Mesh Decimation - University of Maribor, Slovenia
27.6.2001	Parallel Triangular Mesh Simplification – University of Ioannina, Greece

Conferences:

24.4.2000 – 25.4.2000	CESCG 2000, Budmerice, Slovakia
26.4.2000 – 29.4.2000	SCCG 2000, Budmerice, Slovakia
10.9.2000 – 15.9.2000	Algoritmy 2000 – Conference on Scientific Computing, Vysoke Tatry, Slovakia
28.9.2000 – 29.9.2000	3rd EUROGRAPHICS Workshop on Parallel Graphics & Visualization, Girona, Spain
25.4.2001 – 28.4.2001	SCCG 2001, Budmerice, Slovakia
21.4.2002 – 24.4.2002	CG&GM2002, Amsterdam, The Netherlands
21.4.2006 – 24.4.2006	SCCG 2006, Ciasta Papiernicka, Slovakia

Appendix D

Project work

Member of solving team of following projects:

- MSM235200005 and LC06008 Ministry of Education CR
- 3DTV FP6-2003-IST-2 project Network of Excellence, No:511568
- Computer Graphics and Visualization in Parallel and Distributed Environment, MSMT CR - VS 97 155

Appendix E

Citations

- Krivograd, S., Hren, G., Žalik, B., Jezernik, A., "Hiter algoritem za poenostavljanje in obnovitev trikotniških mrež za prenos rezultatov MKE preko svetovnega spleta (A fast triangular-mesh decimation-and-undecimation algorithm for transferring FEM results via the Web", Strojniški vestnik (Journal of Mechanical Engineering), L. 49, Št. 11, 2003, str. 524-537, ISSN 0039-2480.
- Jose Pablo Suárez Rivero and Ángel Plaza de la Hoz, Refinement and hierarchical coarsening schemes for triangulated surfaces, Proc. 11th Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2003), 2003
- Hradek,J., Skala,V.: Hash Function and Triangular mesh Reconstruction, Vol.29, No.6., pp.741-751, Computers&Geosciences, Pergamon Press, ISSN 0098-3004, 2003
- S. Krivograd, B. Zalik, F. Novak: Triangular mesh decimation and undecimation for engineering data modelling, Inf. MIDEM , Vol. 32, No. 3, 2002, pp. 219-223.
- S. Krivograd, B. Zalik, F. Novak: TriMeDeC tool for preparing visual teaching materials based on triangular networks, Computer Applications in Engineering Education , Vol. 10, 2002, pp. 144-154.
- M. Grabner, On-the-fly greedy mesh simplification for 2 1/2-D regular grid data acquisition systems, Vision with non-traditional sensors, Proc. of 26th Workshop of the Austrian Association for Pattern Recognition, F. Leberl and F. Fraundorfer (eds.), vol. 160, Austrian Computer Society, 2002

References

[1] Bajaj C., Schikore D.: Error-bounded Reduction of Triangle Meshes With Multivariate Data. SPIE vol. 2656, pp. 34-45, 1996.

[2] Barrera T., Hast A., Bengtsson E.: Surface Construction with Near Least Square Acceleration based on Vertex Normals on Triangular Meshes, Sigrad '02, pp. 43-48, 2002.

[3] Ciampalini A., Cignoni P., Montani C., Scopigno R.: Multiresolution decimation based on global error. Technical Report CNUCE: C96021, Istituto per l'Elaborazione dell'Informazione - Condsiglio Nazionale delle Richere, Pisa, ITALY, July 1996.

[4] Cignoni P., Costanza D., Montani C., Rocchini C., Scopigno R.: Simplification of tetrahedral meshes with error evaluation. In Proceedings of the IEEE Visualization.

[5] Cignoni P., Rocchini C., Scopigno R.: Metro: Measuring Error on Simplified Surfaces. Computer Graphics Forum, vol. 17, pp 167-174, 1998.

[6] Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Brooks, F., and Wright, W.: Simplification Envelopes. In Computer Graphics (SIGGRAPH'96 Proceedings).

[7] Eck M., DeRose T., Duchamp T., Hoppe H., Lounsbery M., Stuetzle W.: Multiresolution analysis of arbitrary meshes. In Robert Cook, editor, SIGGRAPH 95 Conference Proceedings, Annual Conference Series, pages 173--182. ACM SIGGRAPH, Addison Wesley, August 1995. Los Angeles, California, 06-11 August 1995.

[8] Fdez-Vidal, Xose R. and Garcia, J. A. and Fdez-Valdivia, J.: Using Models of Feature Perception in Distortion Measure Guidance. Pattern Recogntion Letters, Vol. 19, Issue 1, pp. 77-88, 1998.

[9] Franc M., Skala V.: Parallel Triangular Mesh Decimation. In Proceedings of International Conference SCCG'2000, Budmerice, Slovakia, pp.164-171, ISBN 80-223-1486-2.

[10] Garland M., Heckbert P.S.: Surface Simplification Using Quadratic Error Metrics. Computer Graphics (SIGGRAPH '97 Proceedings), pages 209-216, 1997.

[11] Garland M.: Multiresolution Modeling: Survey & Future Opportunities. Eurographics '99, State of the Art Report. 1999.

[12] Garland M., Willmott A., Heckbert P.S., Hierarchical Face Clustering on Polygonal Surfaces. ACM Symposium on Interactive 3D Graphics, March 2001.

[13] Gueziec A.: Locally Toleranced Surface Simplification. IEEE Transactions on Visualization and Computer Graphics, vol. 5(2) pp.168--189, 1999.

[14] He T., Hong L., Varshney A., Wang S.: Controlled Topology Simplification. IEEE Transactions on Visualization and Computer Graphics, vol. 2, pp. 171-184, 1996.

[15] Heckbert P.S., Garland M.: Optimal Triangulation and Quadric-based surface simplification. Journal of Computational Geometry: Theory and Applications, vol. 14 no. 1-3, pages 49-65, November 1999.

[16] Heckbert P.S., Garland M.: Survey of surface simplification algorithms. Technical report, Carnegie Mellon University - Dept. of Computer Science, 1997.

[17] Hoppe H., DeRose T., Duchamp T., McDonald J., Stuetzle W.: Mesh optimization. In SIGGRAPH 93 Conference Proceedings, pages 19-26, 1993.

[18] Hoppe H.: Efficient Implementation of Progressive Meshes. Computers & Graphics Vol. 22 No. 1, 27-36, 1998.

[19] Hoppe H.: New quadric metric for simplifying meshes with appearance attributes. In David Ebert, Markus Gross, and Bernd Hamann, editors, IEEE Visualization '99, pages 59--66. IEEE, October 1999. ISBN 0-7803-5897-X. Held in San Francisco, California.

[20] Hoppe H.: Progressive meshes. In Computer Graphics Proceedings, Annual Conference Series, 1996 (ACM SIGGRAPH '96 Proceedings) , pages 99-108, 1996.

[21] Junger B., Snoeyink J.: Selecting Independent Vertices For Terrain Simplification. In WSCG '98, Plzen, Czech Republic, February 1998.

[22] Kirkpatrick D.: Optimal Search in Planar Subdivisions. SIAM J. Comp., pages 12:28-35, 1993

[23] Klein R., Kraemer J.: Building multiresolution models for fast interactive visualization. Proceedings of SCCG '97 Spring Conference on Computer Graphics, Bratislava, June 5-8, 1997.

[24] Klein R., Liebich G., Straser W.: Mesh reduction with error control. Proceedings of Visualization ’96, 1996.

[25] Klein R.: Multiresolution Representation for Surface Meshes Based on the Vertex Decimation Method. Computers & Graphics, Vol. 22. No. 1, pp. 13-26, 1998.

[26] Krus, M., Bourdot, P., Guisnel, F., Thibault, G.: Levels of detail and polygonal simplification. ACM's Crossroads, 3.4., 1997.

[27] Langis C., Roth G., Dehne F.: Mesh Simplification in Parallel. In Proceedings of the 4th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2000), December 11-13, 2000. pp. 281-290. NRC 44161.

[28] Lee A.W.F., Moreton H., Hoppe H.: Displaced Subdivision Surfaces. Proceedings of SIGGRAPH 00 (2000).

[29] Lee A.W.F., Sweldens W., Schroder P., Cowsar L.: MAPS: Multiresolution Adaptive Parameterization of Surfaces. In SIGGRAPH '98 Proceedings. 1998.

[30] Levoy M.: The Digital Michelangelo Project. In Proceedings of the 2nd International Conference on 3D Digital Imaging and Modeling, October 1999.

[31] Lindstrom P., Turk G.: Fast and memory efficient polygonal simplification. IEEE Visualization 98 Conference Proceedings, 1998.

[32] Lindstrom P., Turk, G.: Image-Driven Mesh Optimization. Technical report GIT–GVU–00–16, Georgia Institute of Technology, June 2000.

[33] Lindstrom, P., Turk G.: Image-driven Simplification. Technical Report GIT–GVU–99–49, Georgia Institute of Technology, December 1999.

[34] Lorensen W., Cline H.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Computer Graphics (SIGGRAPH ’87 Proceedings), 21(4), pp. 163-169, July 1987.

[35] Low K., L., Tan T. S.: Model Simplification Using Vertex-clustering. Proceedings of the 1997 symposium on Interactive 3D graphics, Providence, Rhode Island, United States, pp. 75 - 83, 1997.

[36] Luebke D., Reddy M.,Cohen J.,Varshney A.,Watson B., Huebner R.: Level of Detail for 3D Graphics. Published by Morgan Kaufmann as part of their series in Computer Graphics and Geometric Modeling. ISBN 1-55860-838-9.

[37] Nooruddin F.S., Turk G.: Simplification and Repair of Polygonal Models Using Volumetric Techniques. GVU Technical Report GIT-GVU-99-37, Georgia Tech 1999.

[38] Pajarola R., DeCoro C.: Efficient implementation of Real-Time View-Dependent Multiresolution Meshing. IEEE Transactions on Visualization and Computer Graphics, vol. 10, no. 3, May/June 2004.

[39] Pawasauskas J.: Generalized Unstructured Decimation. Advanced Topics in Computer Graphics - CS563, March 18, 1997.

[40] Rendleman C.A., Beckner V.E., Lijewski M., Crutchfield W.Y., Bell J.B.: Parallelization of Structured, Hierarchical Adaptive Mesh Refinement Algorithms. Computing and Visualization in Science, April 1999.

[41] Renteln P. and Dundes A.: Foolproof: A Sampling of Mathematical Folk Humor. Notices Amer. Math. Soc. 52, 24-34, 2005.

[42] Rossignac J.: 3D Compression Made Simple: Edgebreaker with Zip&Wrap on a Corner-Table, Proceedings of the International Conference on Shape Modeling & Applications, pp. 278, May 07-11, 2001.

[43] Rossignac J., Borrel P.: Multi-resolution 3D approximations for rendering complex scenes. Modelling in computer graphics: Methods and Applications, pp. 455-465, 1993.

[44] Shaffer E., Garland M.: Efficient Adaptive Simplification of Massive Meshes. IEEE Visualization 2001.

[45] Schroeder W. J.: A Topology Modifying Progressive Decimation Algorithm. In IEEE Proceedings Visualization '97, pages 205-212.

[46] Schroeder W. J., Zarge J. A., Lorensen E.: Decimation of Triangle Meshes. Computer Graphics (SIGGRAPH '92 Proceedings), Vol. 26, No. 2, July 1992, pp. 65-70.

[47] Surazhsky T., Magid E., Soldea O., Elber G., Rivlin E.: A comparison of Gaussian and mean curvatures estimation methods on triangular meshes. IEEE International Conference on Robotics & Automation, 2003.

[48] Taubin G.: Geometric Signal Processing on Polygonal Meshes. Eurographics 2000 - State of the Art Report, August 2000.

[49] Taubin, G.: A Signal Processing Approach to Fair Surface Design. Computer Graphics, August 1995.

[50] Taubin G.: Curve and Surface Smoothing Without Shrinkage. Proceedings of the Fifth International Conference on Computer Vision, pp. 852-855, 1995.

[51] Taubin G., Zhang T., Golub G.: Optimal Surface Smoothing as Filter Design. Tech. Rep. 90237, IBM T.J. Watson Research, March 1996.

[52] Volpin O., Sheffer A., Bercovier M., Joskowicz L.: Mesh Simplification with Smooth Surface Reconstruction. Computer-Aided Design, vol. 30 no. 11, pp. 875-882, 1998.

[53] Weisstein, Eric W.: MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/.

[54] Wu J. H., Hu S. M., Sun J. G., Tai C. L.: An Effective Feature-Preserving Mesh Simplification Scheme Based on Face Constriction. Ninth Pacific Conference on Computer Graphics and Applications (PG'01), pp. 12-21, 2001.