ZÁPADOČESKÁ UNIVERZITA

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Methods for Polygonal Mesh Simplification

State of the Art and Concept of Doctoral Thesis

Martin Franc

Technical Report No. DCSE/TR-2002-01
January, 2002

Distribution: public

Technical Report No. DCSE/TR-2002-01
January, 2002

Methods for Polygonal Mesh Simplification

Martin Franc

Abstract

For twenty years, it has been clear that many datasets are excessively complex for applications such as real-time display, and that techniques for controlling the level of detail of models are crucial. Many applications in computer graphics and related fields can benefit from automatic simplification of complex polygonal surface models. Applications are often confronted with either very densely over-sampled surfaces or models too complex for the limited hardware capacity available. An effective algorithm that produces high-quality approximations of the original model is a valuable tool for managing data complexity.

This work begins with an overview of the most notable available algorithms for the automatic simplification of highly detailed polygonal models. Mesh decimation techniques are of particular interest because our previous work was focused on vertex decimation, and iterative edge contraction algorithms also seem to be a promising approach thanks to their ability to preserve volume, or other geometrical properties of the model. Our recent work in this field is also discussed and concludes the advantages and disadvantages of a vertex decimation algorithm. Finally a summary of the most significant directions in mesh simplification is presented, and the other possible future directions in this field are discussed.

This work was supported by the Ministry of Education of Czech Republic – project MSM 235200005 and Academy of Sciences of The Czech Republic – project A 2030801.

Abstract

For twenty years, it has been clear that many datasets are excessively complex for applications such as real-time display, and that techniques for controlling the level of detail of models are crucial. Many applications in computer graphics and related fields can benefit from automatic simplification of complex polygonal surface models. Applications are often confronted with either very densely over-sampled surfaces or models too complex for the limited hardware capacity available. An effective algorithm that produces high-quality approximations of the original model is a valuable tool for managing data complexity.

This work begins with an overview of the most notable available algorithms for the automatic simplification of highly detailed polygonal models. Mesh decimation techniques are of particular interest because our previous work was focused on vertex decimation, and iterative edge contraction algorithms also seem to be a promising approach thanks to their ability to preserve volume, or other geometrical properties of the model. Our recent work in this field is also discussed and concludes the advantages and disadvantages of a vertex decimation algorithm. Finally a summary of the most significant directions in mesh simplification is presented, and the other possible future directions in this field are discussed.

Contents

Chapter 1

Introduction

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 [23] contains $2*10^9$ triangles). The source of such models is 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 [37] that produce models with a very high density of polygonal meshes displaying almost regular arrangement of vertices.

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. The aim of such techniques is to reduce the complexity of the model whilst preserving its important details. Several effective techniques have been developed in recent years, and they provide valuable tools for tailoring large datasets to the needs of individual applications and for producing more economical surface models. Techniques for controlling the run-time level of surface detail are also very important in real-time rendering systems. For any given system, available hardware capacity – such as frame buffer fill rates, transformation and lighting throughput, and network bandwidth – is essentially fixed.

The thought of polygonal model simplification is not new. Proposals of some procedures are over 20 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. 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. [32] 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 a global error bounds [16] or within a simplification envelope [3].

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 are often incapable of achieving desired reduction levels. Removing topological constraint can create large gains in reduction factors and therefore, system responsiveness and interactivity.

Another important development in the field of polygon reduction is the progressive mesh introduced by Hoppe [14,12]. In Hoppe's scheme, the single topology preserving operation edge collapse (and its inverse edge split) is sufficient to transform the full resolution mesh into a simpler base mesh (and the base mesh back to the full resolution mesh). Progressive meshes offer many attractive properties, including the ability to store compactly and incrementally transmit and reconstruct geometry. Such capability is important for network based rendering and data transmission.

This work is structured as follows. In Chapter 2 we mention notable simplification algorithms and approaches. We focus on decimation techniques mainly. Advantages and disadvantages of the algorithms are pointed out, and the possible future directions are discussed. Chapter 3 contains a summary of our previous work – a technique based on vertex decimation. The achieved results, comparison and important properties of those algorithms are discussed in detail. Finally in Chapter 4 we briefly mention possible results from future research.

Chapter 2

State of the Art

Before we start describing various algorithms for simplification, we need to define a polygonal (triangular) surface as well as terms like a manifold, or a non-manifold respectively. Besides the number of triangles in the resulting approximation after a simplification process, the error value is the most significant information in algorithm evaluating. 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 is important to preserve a volume or area, in others for example the geometric distance between the original and resulting model. For efficient implementation, it is also necessary to choose a suitable data structure.

Polygonal Surface

We use a definition according to [7]. A polygonal model M is composed of a fixed set of vertices V = (v_1, v_2, ….., v_k) and a fixed set of faces F = (f_1, f_2, ….., f_n). It provides a single fixed resolution representation of an object. 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.

To streamline the discussion, we will assume that the models do not contain isolated vertices and edges. It must be also clear if the surface is manifold or non-manifold, since many algorithms can deal with manifold surfaces only. A manifold surface can be defined as a surface where every inner vertex of a surface has its neighbourhood topologically similar to a disc. Analogously, in non-manifold surfaces can appear edges shared by more than two triangles, or patches touching each other in one vertex.

Approximation Error

The goal of simplification is to get a surface approximation M’, which is as similar as possible to the original surface M. Therefore we need some means of qualifying the notion of similarity – the approximation error E(M,M’). When comparing general surfaces, there is no single distinguished direction along which to measure distances. Instead, distances between closest pairs of points are measured. The distance from a point **v** to the model M is defined as the distance to the closes point **w** on the model:

$$d_v(M) = \min_{w \in M} \|v - w\| \qquad (1)$$

where $\|.\|$ is the usual Euclidean vector length operator. Probably the most popular geometric error measure is the Hausdorff distance, which can be defined as

$$E_{\max}(M_1, M_2) = \max(\max_{v \in M_1} d_v(M_2), \max_{v \in M_2} d_v(M_1)) \qquad (2)$$

The Hausdorff error measures the maximum deviation between the two models. If $E_{\max}(M, M') < \varepsilon$, then we know that every point of the approximation is within ε of the original surface and that every point of the original is within ε of the approximation.

Analogically we can define E_{avg}, which measures the average squared distance between the two models as

$$E_{avg}(M_1, M_2) = \frac{1}{w_1} \int_{v \in M_1} d_v^2(M_2) + \frac{1}{w_2} \int_{v \in M_2} d_v^2(M_1) \qquad (3)$$

where w_1,w_2 are the surface areas of M_1, M_2. Note the symmetric construction of both E_{max} and E_{avg}. For many simplification algorithms it is not sufficient to simply consider every point on M_1 and find the closest corresponding point on M_2. We must do the same for every point on M_2. In practice, these error metrics are very expensive to compute exactly. Instead, we approximate these exact metrics 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. Some of the simplification algorithms even use such error metrics to directly guide the construction of approximations (for example the energy term used by Hope et al. [11,14]). The question of an error measurement is further discussed in Chapter 4.

Data Structure

The data structures usually consist 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 data structure can represent this information, many implementations use a space-efficient vertex-triangle hierarchical ring structure [32]. 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. Since edges are not explicitly defined, they can, however, be defined as ordered vertex pairs in the triangle definition.

Notable Algorithms and Approaches

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 [19]. 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.

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 [26] 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
- isosurface extraction to obtain a polygonal representation of the simplified surface
- topology-preserving triangle count reduction to decimate the isosurface.

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 center 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, [26] 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 an erosion is performed followed by a 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, [26] extracts an iso-surface from the volumetric representation of the model using the standard Marching Cubes algorithm [37].

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 [6] 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.

Simplification envelopes

Simplification envelopes [3] are something of a meta-method. The main specificity 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 guarantee 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.

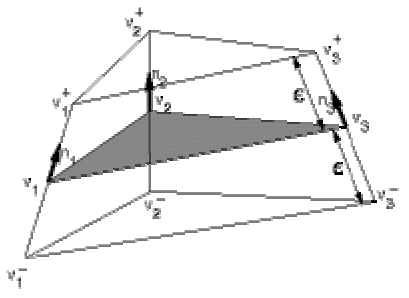


Figure 1: Building inner and outer envelopes for a triangle. Taken from [3].

At the beginning the envelopes have to be constructed (Figure 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:

- Remove the vertex and the adjacent faces.
- 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.

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 [14,12].

Essentially, this requires that the surface be reconstructed using a wavelet representation. This is as usual difficult. Eck et al. [4] developed a method for constructing wavelet representations of arbitrary manifold surfaces. This is not actually a simplification algorithm. It is a preprocessor 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.

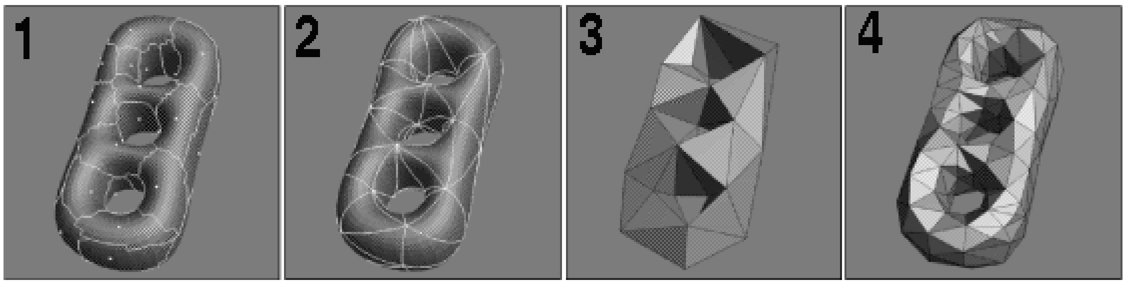


Figure 2: Four steps in the MRA algorithm. Taken from [4].

- Partitioning. A Voronoi-like diagram is constructed on the original mesh (Figure 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 centers of adjacent faces). This diagram is then triangulated using a Delaunay-like method and the harmonic maps to straighten the edges (Figure 2.2).
- Parameterization. The result is a base mesh (Figure 2.3) that is parameterized using a harmonic map. The parametrization 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 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.

Vertex Clustering

Vertex clustering methods spatially partition the vertex set into a set of clusters and unify all vertices within the same cluster [29,25]. 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.

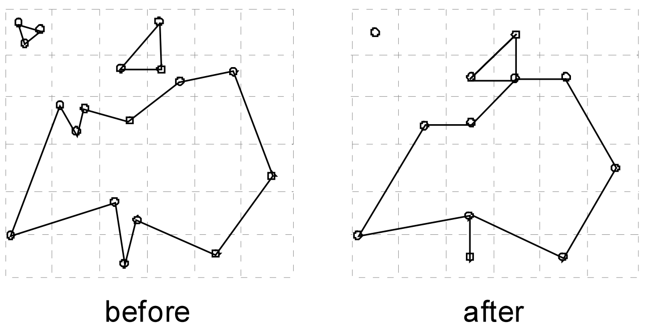


Figure 3: Uniform clustering in two dimensions.

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. Centering 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.

Region Merging – Face Clustering

A handful of simplification algorithms operate by merging surface regions together [10]. 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. [8]. 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 [6,8,9,13], see Figure 4.

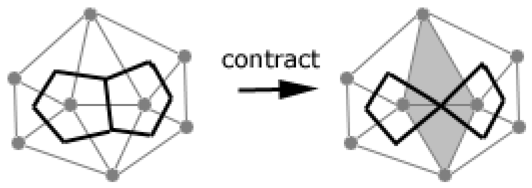


Figure 4: Edge contraction in the dual graph. The two faces of the surface corresponding to the endpoints of the dual edge are merged to form a single face cluster.

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 [30] shows, clustering methods seem to be a promising approach for the simplification of massive meshes.

Decimation Methods

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 [27].

One of the more widely used algorithms is vertex decimation, an iterative simplification algorithm originally proposed by Schroeder et al. [32]. 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 [31] was able to lift these restrictions by incorporating cutting and stitching operations into the simplification process.

The three steps of the algorithm are:

- Characterize the local vertex geometry and topology.
- Evaluate the decimation criteria.
- 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.

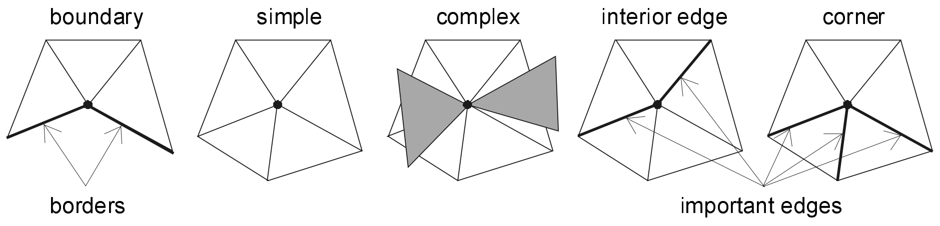


Figure 5: Vertex classification.

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*,* resp. interior edge*,* vertex importance is equal to its distance from the bisector traversing through other two vertices on the border, resp. important edges see Figure 6b.

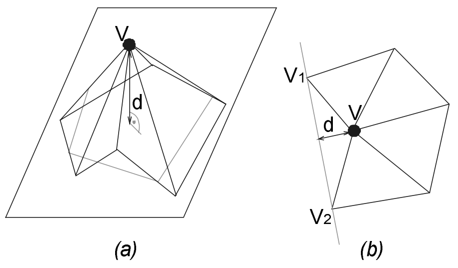


Figure 6: The distance between a vertex and the average plane (a) and between a vertex and the bisector (b).

An average plane is constructed using the triangle normals $\mathbf{n}_i$, centres $\mathbf{x}_i$ and areas A_i.

$$\vec{N} = \frac{\sum \vec{n}_i A_i}{\sum A_i} \quad \vec{n} = \frac{\vec{N}}{\left|\vec{N}\right|} \quad \vec{x} = \frac{\sum \vec{x}_i A_i}{A_i} \tag{4}$$

where the summation is over all triangles in the loop. The distance of the vertex $\mathbf{v}$ 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 doesn't 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 won't 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 is under some threshold. More recent methods [17,1] use more accurate error metrics, like the localized Hausdorff error. Klein et al. [17,18] 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 [7]. The triangulation method has also been improved. Ciamplalini et al [1] 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. [16,18] 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. [22] recently described an algorithm, which establishes smooth parametrizations 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 parametrization of the original mesh over a low face count base domain. The resulting parametrizations are of high quality and their utility is demonstrated in an adaptive, subdivision-connectivity remeshing algorithm that has guaranteed error bounds.

Vertex decimation methods [27] produce good quality results, preserve a mesh topology and are generally applicable to manifolds only.

The other class of decimation techniques is based on the iterative contraction of vertex pairs (edges) [6,30,24,14,13]. 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, 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 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.

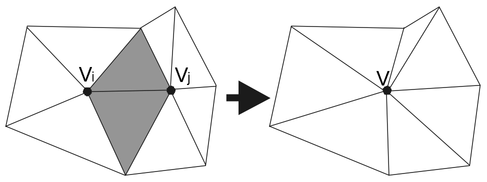


Figure 7: Edge (v_i, v_j) is contracted. The dark triangles become degenerate and are removed.

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 [14] is based on minimization of an energy function. This function has four terms:

$$E(M) = E_{dist}(M) + E_{spring}(M) + E_{scalar}(M) + E_{disc}(M) \qquad (5)$$

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}. 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 [11] 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 [6,9] 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 $\|\mathbf{v_1}-\mathbf{v_2}\| < t$, 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 $\mathbf{v_1}$ and $\mathbf{v_2}$ a new matrix **Q**', which approximates the error at **v**', is derived such as $\mathbf{Q}' = \mathbf{Q_1} + \mathbf{Q_2}$. A new position of vertex v is set according to the need to minimize $\Delta(\mathbf{v})$. This is equivalent to solving:

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} v' = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{6}$$

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

$$v' = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{7}$$

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:

$$Q = \sum_{p\in planes(v)} K_p \tag{8}$$

$$K_p = pp^T = \begin{bmatrix} a^2 & ba & ca & da \\ ab & b^2 & cb & db \\ ac & bc & c^2 & dc \\ ad & bd & cd & d^2 \end{bmatrix} \tag{9}$$

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 [24] 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, [24] 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 $t=(v_e,v_1,v_2)$, where v_e is a vertex on collapsing edge, we try to minimize the volume of tetrahedral $t_h=(v, v_e,v_1,v_2)$. The formula is as follows:

$$\sum_i V = ((v, v_0^{t_i}, v_1^{t_i}, v_2^{t_i})) = \sum_i \frac{1}{6} \begin{vmatrix} v_x & v_{0x}^{t_i} & v_{1x}^{t_i} & v_{2x}^{t_i} \\ v_y & v_{0y}^{t_i} & v_{1y}^{t_i} & v_{2y}^{t_i} \\ v_z & v_{0z}^{t_i} & v_{1z}^{t_i} & v_{2z}^{t_i} \\ 1 & 1 & 1 & 1 \end{vmatrix} = 0 \tag{10}$$

To further constraints the vertex position, [24] 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

$$f_v(e,v) = \sum_i V((v, v_0^{t_i}, v_1^{t_i}, v_2^{t_i}))^2 \tag{11}$$

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:

$$f_s(e,v) = \sum_i L((v, v_i))^2 \tag{12}$$

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.

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 consists. 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 equired 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 [7]. During the process of simplification, we get a sequence of models

$$M^0 \xrightarrow{\phi^1} M^1 \xrightarrow{\phi^2} M^2 \xrightarrow{\phi^3} \ldots\ldots \xrightarrow{\phi^k} M^k \tag{13}$$

where each step $M^{i-1} \rightarrow M^i$ corresponds to the application of a single contraction ϕ^i. Thus each intermediate approximation M^i 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 [14,12], 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 [14] 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. [16.18] suggest to start 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 have to be transmitted. The unique solution bring Ciampaliny et al. [1], 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.

Post-processing techniques – Mesh Smooting

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 [36]. Since most existing algorithms based on fairness norm optimisation are prohibitively expensive for very large surfaces, Taubin's [33,35] 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

$$\Delta x_i = \sum_{j \in i^*} w_{ij}(x_j - x_i), \tag{14}$$

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 vetex $\mathbf{v}_i$. If $W = (w_{ij})$ 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

$$x' = f(K)x \tag{15}$$

and this process can be iterated N times

$$x^N = f(K)^N x \tag{16}$$

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

$$f(k) = (1 - \lambda k)(1 - \mu k) \tag{17}$$

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:

$$\begin{aligned} \forall i:\ x_i' &= x_i + \lambda x_i \\ \forall i:\ x_i' &= x_i + \mu x_i \end{aligned} \tag{18}$$

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

$$k_{pb} = \frac{1}{\lambda} + \frac{1}{\mu} > 0 \tag{19}$$

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.

Since this approach uses only the analogy to the signal processing, the idea is offered to convert the polygonal mesh to a signal representation and to solve the whole simplification process in the signal space as considered in chapter 4.

Possible Future Directions

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 [11]. On the other hand there are very fast, but low quality vertex clustering algorithms [25]. Somewhere between these extremes we have a number of algorithms, such as the quadric error metric [6] or vertex decimation [32], 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 [11], or methods based on the principle of signal processing on polygonal meshes [34].

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 3

Previous Work and Conclusions

We have been studying approaches based on vertex decimation. As already mentioned vertex decimation is a very fast and easy to implement method of mesh simplification. Let us repeat the main characteristics of this technique and the approach we have initially chosen.

Vertex Decimation Improvement

Vertex decimation is an iteration process where all the vertices in a given mesh are evaluated and the least important one is chosen and removed. We have come out from Schroeder's method [32], where a vertex importance is given as a distance of the vertex and the average plane computed from neighbouring vertices (see Figure 6). In general the importance should tell us how much the shape of the mesh would be changed after the vertex removal. In other words, for example, a vertex which lies in a „flat" area of the mesh, should have equal importance or be close to zero. Analogically a vertex on a peak of the mesh will have as great importance as the peak is higher relatively to its neighbour vertices.

Another step after vertex evaluation is to find the least important one. Well, this would be quite an easy job if we could remove just one vertex in one iteration. However, there are usually several vertices of at least similar or even the same importance after one evaluation. Therefore we can remove some vertices concurrently. This approach obviously improves the efficiency and running time of the algorithm. On the other hand, instead of searching for the vertex with the least importance, we need to sort all the vertices according to their importance (then we first remove several vertices from the sorted list). Since the sorting algorithm gives us O(NlogN) time complexity (N is the number of vertices), we have to put our effort into finding any other ways of sirting vertices efficiently.

After many experiments and observations of vertex importance histograms on several models, we set a hash function. Thanks to this function we can bucket the vertices into 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.

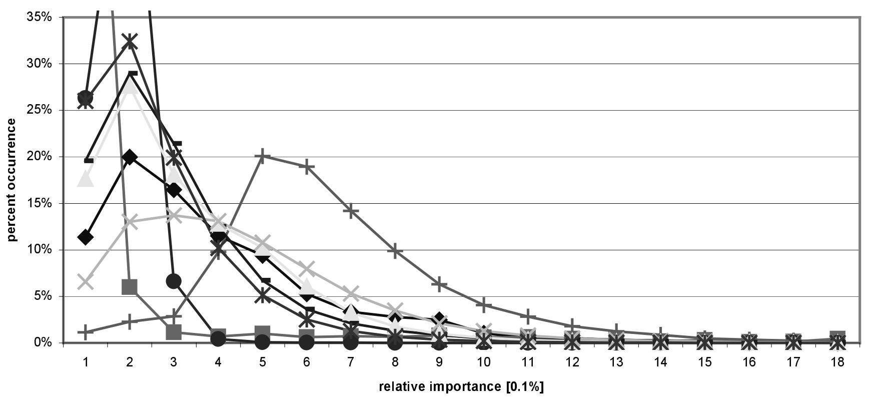
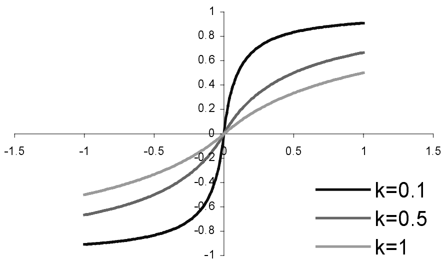


Figure 8: Vertex importance histogram.

A histogram of vertices importance for tested data sets is shown in Figure 8. It is obvious that over 80-90% of all vertices have the importance below 1% of maximum importance value. The hash function has been derived as follows. We used a simple hash function

$$y = \frac{x}{|x| + k} \tag{20}$$

shown in Figure 9 (left).



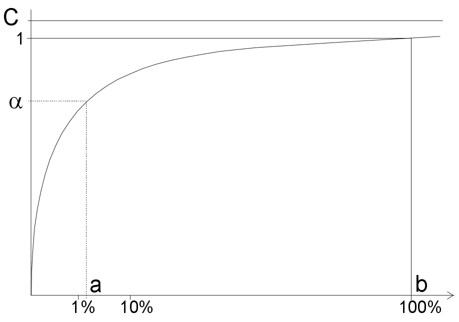


Figure 9: Graph of the function used (*left*), definition of the hash function (*right*)

This hash function enables us to map the interval $< 0,\infty\,)$ to $< 0,1\,)$ non-linearly. Because we need to map only 1% of important vertices the hash function (20) must be modified accordingly, see Figure 9 (right), and scaling coefficient *C* must be introduced.

$$y = C\frac{x}{x + k} \qquad x \geq 0 \tag{21}$$

From equation (5) 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*, see Figure 7 (right), that will be experimentally determined by large data sets processing.

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:

$$1 = C\frac{b}{b+k},\quad b+k \neq 0 \qquad \alpha = C\frac{a}{a+k},\quad a+k \neq 0$$

Then

$$C = \frac{b+k}{b} \qquad \text{and} \qquad \alpha = \frac{b+k}{b}\frac{a}{a+k}$$

Solving that we get:

$$k = \frac{ab(1-\alpha)}{\alpha b - a} \qquad C = \frac{b+k}{b} = 1 + \frac{k}{b}$$

for *b = 1*

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

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

Having created a set of vertices for removal, we can now delete vertex by vertex from this set. After each vertex removal we have to re-triangulate the emerging hole in the mesh. This step is probably the most important in whole decimation process. Here we choose between the quality of the final triangulation and the running time needed to process it. By the quality of the approximation we understand at this moment mainly the shape of the new triangles. Therefore many methods are tried to obtain a kind of Delaunay triangulation or any, in some way, optimal triangulation. In general it is not possible to use directly any two-dimensional algorithm to construct the triangulation, since the loop is usually non-planar. We have to at least project the hole onto a plane avoiding any edge of the hole crossing another, which is not always easy. The method proposed by Schroeder is to divide each hole into two halves. The division is along the split line defined from two non-neighbouring vertices in the loop (hole boundaries). Each new hole is divided again, until only three vertices remain in each loop. Such loops form triangles that may be added to the mesh. Because the loop is non-planar and star-shaped, the loop split is evaluated using a split plane orthogonal to the average plane given by vertices in the loop.

In our method we remove a vertex by collapsing one of the edges adjacent to the vertex. To choose the proper edge to collapse we evaluate all the edges according to a surface area after the edge collapse of each of the edges. Then we chose the edge, which will result the minimal area of the surface after the collapse. The main advantage of such an approach is the high speed of the algorithm and also the very easy way of the new triangulation. All we need is to check if none of the triangles folds over the mesh (see Figure 10) and such a test is again very easy.

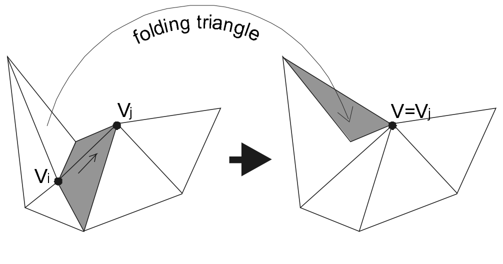


Figure 10: An edge contraction, which causes the mesh to fold over on itself.

Another advantage of this approach is that we can deal with non-manifold meshes. Having described the specific details of the method, we can present our new algorithm now:

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

To make the algorithm more efficient we proposed a special data structure. We used a table of triangles, where we store indices of all vertices for each triangle. In the table of vertices we store each vertex coordinates, a list of triangles sharing this vertex and the address of the cluster where the vertex currently belongs. We also have a vector of clusters where the indices of vertices are stored.

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.

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 1.

Model name	**No. of triangles**	**No. of 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 1: Data sets used.

Methods Comparison

Figure 11 shows the running time for 96% reduction for three different approaches of vertices ordering. We can see that using the hash function we obtained the best running time in comparison to the other methods. It is necessary to point out that the methods using thresholding or sorting algorithms were implemented in parallel. The run time of hash function is faster than 8 processors running the method with sort algorithm. It is because both the sort algorithm and the creation of the independent set of vertices were implemented sequentially. With the thresholding we removed the sort algorithm completely, but an independent set of vertices remained.

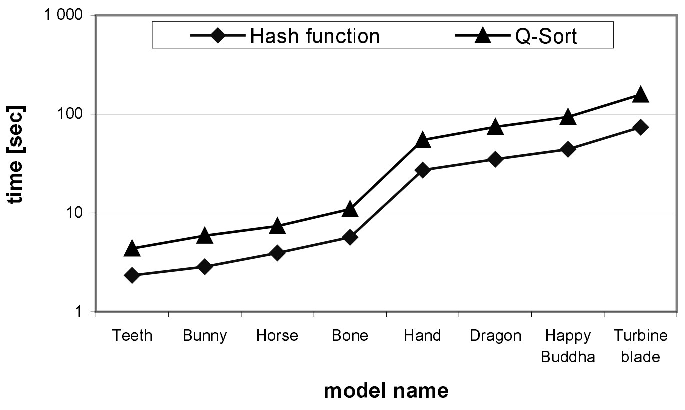


Figure 11: Achieved time comparison for three mentioned approaches.

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 2, where η presents the superiority of the DELL computer against the SGI. Table 3 presents our results according to results obtained recently 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 2: 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 [6]	10,4
Lindstrom & Turk [24]	2585
Hoppe [11]	500
JADE [1]	325

Table 3: Rough comparison of running-times of reduction of the Bunny model.

It is obvious that our algorithm is quite fast, however, we do not know the approximation quality reached by the other algorithms.

Approximation quality

The quality of an approximation can be measured by several approaches. Probably the most popular way is to compute a geometric error using E_{avg} metric (22,23) derived from Hausdorff distance:

$$E_{avg}(M_1,M_2)=\frac{1}{k_1+k_2}\left(\sum_{v\in X_1} d_v^2(M_2)+\sum_{v\in X_2} d_v^2(M_1)\right),\ d_v(M)=\min_{w\in P(M)}\|v-w\| \qquad (22,23)$$

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 (8):

$$E_{avg}(M_1,M_2)=\frac{1}{k_1}\sum_{v\in X_1} d_v^2(M_2),\ X_1\subset P(M_1) \qquad (24)$$

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

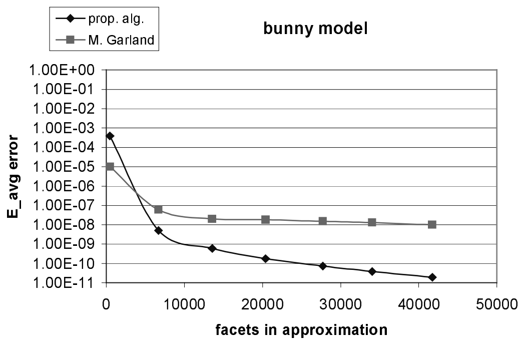


Figure 12: Approximation error comparison.

Figure 12 presents a comparison of error measurement of the proposed algorithm and M. Garland's method. 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 error. Examples of reduced models are presented on Figure 13,14, 15.

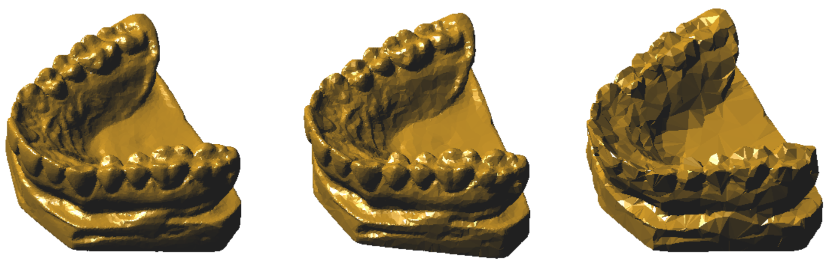


Figure 13: A teeth model (courtesy Cyberware) at different resolutions; the original model with 58,328 triangles on the left, reduced to approx. 29,000 triangles in the middle and 6,000 triangles approximation on the right

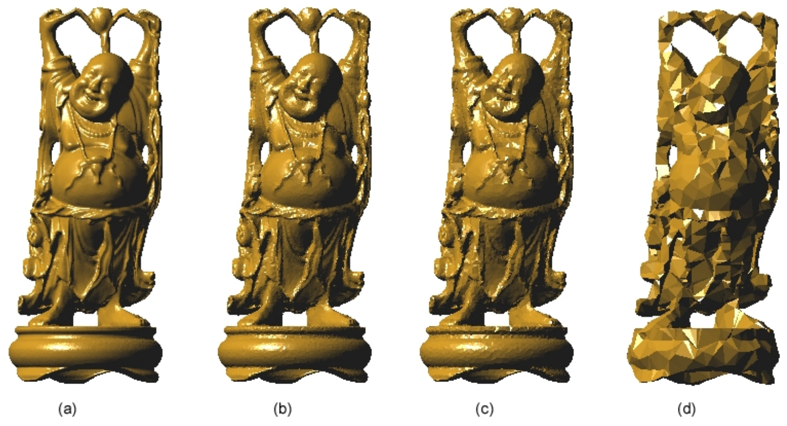


Figure 14: The Happyb model (courtesy GaTech) at different resolutions; the original model with 1,087,716 triangles (a), reduced to 105,588 triangles (b), 52,586 triangles (c), 10,974 triangles (d)

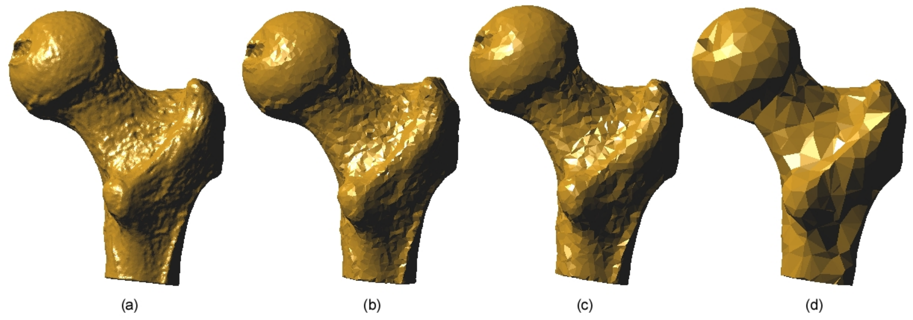


Figure 15: A bone model (courtesy Cyberware) at different resolutions; the original model with 137,072 triangles (a), reduced to13,706 triangles (b), 6,854 triangles (c), 1,248 triangles (d).

To improve the efficiency of our algorithm we also studied a problem of parallel implementation [20]. The basic idea 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 triangulated independently. This decreases the program complexity and run time significantly. Since deletion and triangulation is related to the degree of vertices being removed (in the worst case with O (d^2) time complexity, where d is the vertex degree). However, this approach ignores the preservation of the model shape. Therefore we use a technique where we assign an importance value to each vertex, then select an independent set to delete by greedily choosing vertices of low importance relative to their neighbours.

It is natural to use a greedy strategy to construct an independent set from an assignment of importance values. It means to go through all the vertices in order of their importance and take a vertex if none of its neighbours have been taken. It means that only those vertices that do not share an edge with the each other can be in the independent set. To avoid critical sections, which drastically decreased the performance, we introduced a super-independent set of vertices instead [5].

However, the results of the experiments showed that such straightforward parallelization does not lead to any significant speedups, and the cost of algorithm complexity is greater than the benefits.

Conclusion

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

The algorithm combines Schroeder's decimation (its vertex importance evaluation) with edge contraction to simplify object models in a short time. We have introduced a hash function, which we use instead of expensive vertices sorting. Our algorithm has proved its high speed and simplicity.

However, there are many problems connected with vertex decimation in general. The first thing is that it is necessary to explicitly define the sharp edges. Sharp edge is that edge, where the angle between the two adjacent triangles is less than the specific threshold. To set the threshold some experience of the user is necessary and such a threshold is different for each model. 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 16 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.

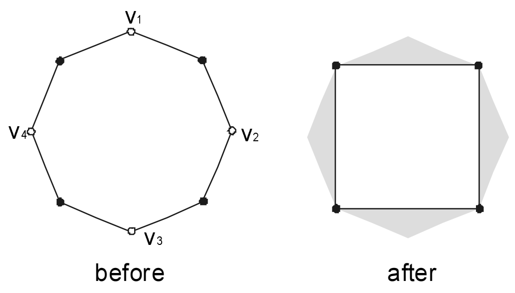


Figure 16: A shrinkage caused by the removal of vertices v_1, v_2, v_3 and v_4.

Moreover, edge contraction methods offer intuitive techniques for eliminating approximation error by optimal positioning of a new vertex after performing an edge collapse.

Chapter 4

Possible results from further research

Since almost every application uses multiresolution models generated offline, the speed of the simplification algorithms is no longer the main requirement. Algorithms based on the precision with which the approximation is modelled are needed instead. Applications like rapid prototyping, CAD/CAM and other industrial systems rely on the accuracy of the models. Consequently, the question arises how to measure the approximation error. The majority of algorithms evaluates the error using Hausdorff distance. However, this measure is either computationally prohibitively expensive in its original form, or quite inaccurate using a discretization. Generally, it should give us the information about the geometrical distance between the two models. However, it does not tell us anything about the volume or area differences. On the other hand, if we have two models of the same volume or area, there can be quite a difference between their appearance. In our future research we will search for a new error estimation that could give us a satisfactory answer concerning the similarity of the models. The base of the error evaluation will lie with the geometrical properties of the model, but the appearance should be also considered. New algorithms must also produce a high quality mesh by mean of resulting triangulation, which could tend to be somehow optimal.

The other question arises with the growing field of new applications requiring multiresolution models. For example liquid flow simulation systems produce a huge amount of data that should be visualized. Here we need new algorithms capable of creating a multiresolution model of data, which cannot be loaded into the memory at once in their original size, and also to handle a non-rigid material. This is even more obvious in animation systems, where some hierarchy must be put in the models being moved at the scene; virtual dancers for instance. We would like to find an algorithm and a structure successfully dealing with the mesh hierarchy.

It would be also interesting to perform experiments concerned with solving simplification problems in a different space. Some recent studies show that volumetric methods or wavelet functions give surprisingly good results, however, they are more complicated. Fourier transformation should be the next method to be explored, as it is already successfully used for mesh smoothing.

Having explained our motivation, we can say that our future research will be focused on three main goals as follows:

- To find an accurate metric to measure the approximation quality. Acording to this metric we will design a new simplification method which will minimize the error of the approximation.
- The new method will be capable of processing data larger than free memory available.
- The last aim is to avoid surface intersections during simplification process.

References

[1] 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.

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

[3] 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).

[4] 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.

[5] 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.

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

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

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

[9] 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.

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

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

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

[13] 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.

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

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

[16] 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.

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

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

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

[20] 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.

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

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

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

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

[25] Low K., Tan T.: Model simplification using vertex-clustering. 1997 Symposium on Interactive 3D Graphics. ACM SIGGRAPH, 1997.

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

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

[28] 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.

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

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

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

[32] 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.

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

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

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

[36] 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.

[37] 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.

Appendix A
Publications

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

[ii] 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.

[iii] 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.

[iv] 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.

[v] 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.

[vi] 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.

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

Appendix B

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