Feature-Based Mesh Simplification With Quadric Error Metric Using A Line Simplification Algorithm

by

Rafael Jose Falcon Lins

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2010

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled “Feature-Based Mesh Simplification With Quadric Error Metric Using A Line Simplification Algorithm” by Rafael Jose Falcon Lins in partial fulfillment of the requirements for the degree of Master of Computer Science.

Dated: August 26, 2010

Supervisors: _____

_____

Reader: _____

_____

DALHOUSIE UNIVERSITY

DATE: August 26, 2010

AUTHOR: Rafael Jose Falcon Lins

TITLE: Feature-Based Mesh Simplification With Quadric Error Metric Using a Line Simplification Algorithm

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: MCSc CONVOCATION: October YEAR: 2010

Permission is herewith granted to Dalhousie University to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

_____

Signature of Author

I dedicate this work to my wife, Amanda Lisboa Brandao, who has always supported and given me the incentive to finish it. I extend my dedication to my parents who through all means have made possible that this dream become reality.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Mesh simplification is an important task in Computer Graphics due to the ever increasing complexity of polygonal geometric models. Specifically in real-time rendering, there is a necessity that these models, which can be acquired through 3D scanning or through artistic conception, have to be simplified or optimized to be rendered on today's hardware while losing as little detail as possible. This thesis proposes a mesh simplification algorithm that works by identifying and simplifying features. Then it simplifies the remaining mesh with the simplified features frozen. The algorithm is called Quadric Error with Feature Curves (QEFC). Quadric Error with Feature Curves works as a tool that allows the user to interactively select a percentage of the most important points of the feature curves to be preserved along with the points determined by the Quadric Error Metric algorithm.

ACKNOWLEDGEMENTS

I owe my deepest gratitude to my two supervisors, Dirk Arnold and Stephen Brooks, for their guidance and patience. Without them this work would not have been possible.

I am indebted to many of my colleagues who supported me during the completion of this work.

Rafael Jose Falcon Lins

Chapter 1: Introduction

Computer Graphics studies the manipulation, representation and rendering of graphical objects using computers. It has had an impact on industries such as gaming, animation and film as well as fields such as medicine, defense and architecture. It relates to any type of method used to draw a graphical element. The visual element can be a 2D model, like an image, or a 3D object. The former could be generated using techniques such as pixel art or vector graphics. The latter, which is the subject of this work, can be generated using different rendering methods such as 3D projection or computing 2D images from 3D volumetric data sets.

Polygonal meshes are a common form of definition of 3D models for interactive graphics due to their mathematical simplicity [1]. This type of model has been used in areas ranging from scientific fields [2] to real-time applications like game development [3]. Surface meshes can be optimized or simplified. In other words, a polygon model can be rendered with a different number of polygons (i.e. resolution) depending on how far it is located from the viewer. However, with the advance of technology, storage and transmission capacity has increased significantly. The progress of technology has brought the need to use large data sets to represent surface models in scientific research and commercial applications. Millions of points can be required to represent a surface when Computer Aided Design (CAD) or 3D scanners are used to generate complex models. For this reason, one must use representations of complex surfaces at different levels of resolution. Given the memory and processing demands required by complex models or groups of models, there is a need to develop algorithms to reduce mesh resolution while maintaining a mesh's fundamental visual features as seen in Figure 1.1. Figure 1.2 shows that depending on the point of view and distance, it is visually difficult to assess differences between the simplified model and the original one. Therefore, it is beneficial to minimize the number of polygons in a model when viewed at a distance.

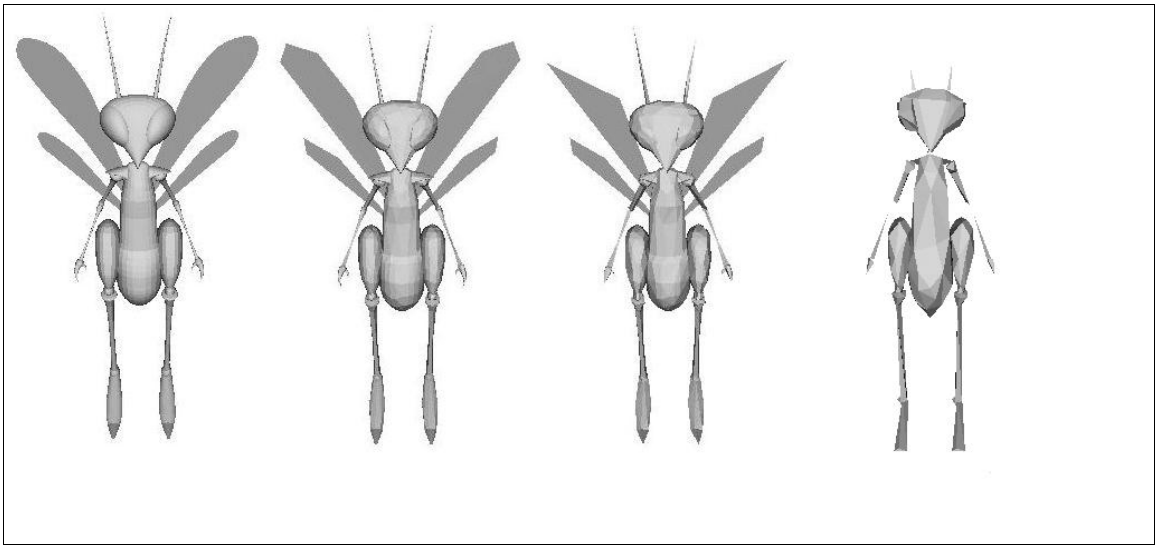


Figure 1.1 Mesh as different levels of simplification. From left to right: 25100, 2536, 1284 and 376 faces.

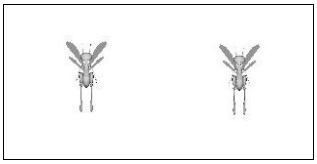


Figure 1.2 Mesh at different levels of simplification. It is difficult to discern that the models are different because they are far from the viewer, although the model to the right has 10 times fewer polygons. From left to right: 25100 and 2536 faces.

Mesh simplification is the field that deals with reducing model complexity in order to obtain a coarser model while maintaining its visual quality as much as possible. Several different approaches to mesh simplification exist which have strengths and weaknesses. Moreover, the combination of some of these algorithms can show improved results when

compared to using a stand-alone algorithm such as the methods proposed by Wu et al [5] and Kim et al. [6].

Since the fidelity of a model can be considered a subjective matter, there are different ways to assess the results of these techniques. They can be compared visually or through the use of tools such as Metro [4], which estimates the Hausdorff distance to the original mesh. Other factors maybe be taken into account as well when comparing simplification methods such as speed, memory requirements and flexibility. These characteristics will be discussed in the next chapters.

The integration of multiple techniques for simplification comes at a cost. It becomes difficult to find appropriate parameters [5, 6], when considering the variation in complexity, shape and attributes of different models. Given this obstacle, the idea is to provide a tool that lets the user interactively define the relative importance of the components of the integrated simplification method, specifically the importance of the feature and line simplification methods with respect to the base simplification algorithm.

This thesis presents a mesh simplification algorithm called Quadric Error with Feature Curves (QEFC) that works by identifying and simplifying features. Then it simplifies the remaining mesh using the Quadric Error Metric (QEM) method [9] with the simplified features frozen. The Feature Detection for Surface Meshes (FDSM) algorithm [7] works by identifying important sections of the model that are represented as curves. Those curves are simplified using a method utilized originally to reduce detail in cartography maps [8].

Chapter 2 starts with a review of basic 3D graphics concepts such as polygon and mesh topology. Then the literature on mesh simplification is covered including simplification iterators, error metrics, simplification methods and curvature detection.

Chapter 3 covers the QEFC algorithm. It describes the methods that are directly related to this work and the integration steps required to implement the QEFC method. There is also a discussion of the motives that led to using those algorithms.

Results obtained by applying the novel algorithm to several meshes are presented in Chapter 4. They are compared with an implementation of the QEM algorithm.

Finally, the last chapter concludes with suggestions for improving the QEFC algorithm and extending it to handle other attributes such as colors or normals.

CHAPTER 2: BACKGROUND

In the field of computer graphics there is a distinction between solid and surface modelling. The former is a technique that involves computing solid objects. This method is used in Computer-Aided Design (CAD), scientific visualization and in professional animation and prototyping of products [10]. The latter is a method that uses topology and geometry elements like vectors, edges and polygons to define a surface shell.

Early solid modelling techniques include Constructive Solid Geometry (CSG) or Boundary Representation (B-Rep) to define models [11]. CSG is a solid modelling method that works by defining a set of Boolean operations to manipulate objects. B-Rep consists of defining a surface that sets apart the interior and exterior of an object. Points are tested to be in or out the model using ray casting. This work will focus on surface modelling rather than solid modelling, since mesh simplification techniques are mostly used with the former.

Both solid and surface modelling can use implicit and parametric techniques to generate shapes. Implicit surfaces are two-dimensional geometric shapes that exist in three-dimensional space. Due to the nature of their representation, it is relatively easy to determine collisions or if a point is within, outside or on the surface. On the other hand, it is difficult to manipulate free form shapes using implicit surfaces. It works by defining a point in the space and evaluating the result of the function *F* represented, for example, by:

$$F(x, y, z) = x^2 + y^2 + z^2 - 1 \qquad (2.1)$$

that defines the surface of a sphere. For a point defined by *p(x, y, z)*, if *F* is greater than zero, the point is outside the shape, less than zero, the point is within it and if it is zero, the point is on the surface.

Parametric surfaces are defined in 3-dimensional space. The surface points are defined by a function with two parameters as in:

$$x = f(u) = r\cos(u) \tag{2.2}$$

$$y = g(u) = r\sin(u) \tag{2.3}$$

$$z = r \tag{2.4}$$

Intersections and point classification are harder to determine. However, unlike implicit surfaces, they are easier to manipulate and they facilitate the definition of free form shapes. One such parametric surface technique that uses control points to change the shape is Bezier patches. Non-polygonal parametric surfaces are more convenient to manipulate and more compact than polygon meshes, though rendering times tend to be higher than for polygons [12].

Implicit and parametric surface models are rendered using ray tracers or by converting the data to known geometric primitives like polygons. Most specialized hardware supports polygons as primitives. Consequently, in order to be rendered on hardware that processes geometric primitives, there is another layer of complexity in the implicit and parametric surface methods since the need to first be converted into polygon mesh representations.

Polygon models are flexible and they are considered the most used form of surface representation for model simplification methods [6] due their rendering simplicity. However, mesh-based models need many polygons to represent detailed shape and they are hard to test for intersection and difficult to change. It is important, however, to note that these representation methods can be used together. For example, polygon meshes can be coupled with techniques like parametric surfaces to generate smooth models.

Subdivision surfaces is an area of computer graphics that studies techniques that refine polygon models by recursively dividing polygons. It uses control points that define the

shape of the model, i.e where the subdivision is focused. There are two main approaches to subdivision surfaces: approximating and interpolating methods. The latter makes the refined surface points match the position of vertices in the original mesh, while the former arrangement will not necessarily adjust the points to the original mesh.

A spline is a parametric function used in Computer Graphics to generate smooth curves. It is defined as a piecewise polynomial function that takes values from an interval from *a* to *b* and maps them to a set of real numbers. The interval *[a, b]* is divided into *k* subintervals defined by $[t_i, t_{i+1}]$. These subintervals are called knots and have a polynomial *P* defined on each of them. The control points are values that determine the shape of the curve. Each point of the curve is taken by computing the weighted sum of control points. A spline can be seen in Figure 2.1

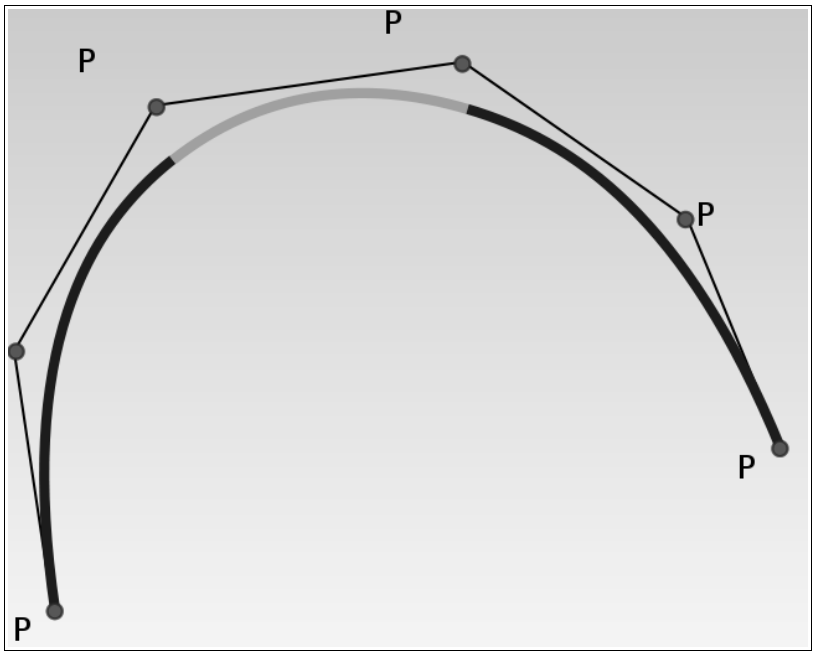


Figure 2.1 A spline with its control points.

This chapter covers the literature on representation of 3D polygon meshes, simplification methods, identification of features and methods used to measure the quality of models. Within the representation of 3D models, basic structures such as vertices, edges, meshes

and their relationships are reviewed. The chapter then briefly describes simplification methods in the literature.

2.1 - Polygonal Modelling

Polygon meshes are representations of shapes in 3D Computer Graphics. The shape is a polyhedral object composed of vertices, edges and faces. Geometric models are not necessarily composed of triangles, even though, usually, the faces are defined as triangles or quadrilaterals. This work focuses on polygon meshes formed by triangles. This is reasonable since most real-time applications use polygon meshes composed of triangles, and because all polygons can be decomposed into triangles.

The topological information about a model consists of the relations between vertices, edges and faces. A vertex is a 3D point in space with three coordinates $v_i = (x_i, y_i, z_i)$. An edge is a segment that connects two vertices. A triangle $f_i = (j, k, l)$ is formed by three edges and three vertices v_j, v_k and v_l. This can be seen in Figure 2.2.

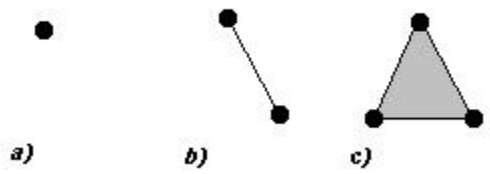


Figure 2.2 a) A vertex, b) an edge and c) a triangle.

Polygon meshes can be manifold or non-manifold (see Figure 2.3). The former occurs when the infinitesimal neighborhood around any point on the surface is topologically equivalent to a disk. A non-manifold mesh is identified when a vertex is shared by unconnected sets of triangles or when the edge of one triangle is spanned by the edges

from two other triangles [1]. Some simplification algorithms or modelling tools can only handle geometric models that are manifold.

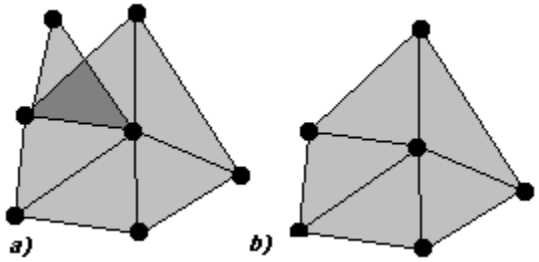


Figure 2.3 a) Non-Manifold mesh and b) manifold mesh.

The order by which the vertices of a triangle are specified determines the orientation of the polygon. This is useful in many ways e.g. it can help determine if a polygon is facing the viewer or if it can be culled.

A normal is a vector that is perpendicular to a surface plane. The normal is used to determine the orientation of the face. It can be calculated as the cross product between two vectors or edges incident to the face. The vertex ordering of the face and whether the coordinate system is left-handed or right-handed will determine the direction of the normal.

Suppose that a triangle t has the vertices v_1, v_2, and v_3. Depending on the order they are presented, e.g. v_1-v_2-v_3, the normal can point towards opposite directions. A vertex also has a normal that is obtained by the sum of the normals of the faces incident to the vertex.

The information obtained with the normals and edges is not only important in the rendering process but it also helps to determine other characteristics of a mesh. The curvature of a mesh is an aspect that can be calculated if a higher level of topological data is analyzed: the relationship between adjacent faces, edges and vertices determine the curvature of a region in the mesh. The length of edges and the polygon area are also

utilized to analyze a mesh's curvature. The curvature calculation will be explained in more detail later in this chapter.

2.2 – Level of Detail

In chapter 1, the importance of mesh simplification was discussed for the purpose of geometry representation in different fields such as gaming and the movie industry. The methods covered in this chapter are based on the concept of mesh simplification that deals with multiple resolutions of a model. This means that versions of the model are generated at different levels of simplification. In order to handle them, it is necessary to develop methods to decide which level to use and how to switch between different models. The simplification is based on a particular metric and the objective is to deliver faster rendering by reducing the polygonal complexity while preserving the most salient details of the model. This is called Level of Detail (LOD). A LOD system can essentially be divided into three stages:

1 – Obtain the simplified or optimized mesh (*generation*).

2 – *Select* at run-time an appropriate simplified mesh to be rendered.

3 – Seamlessly *switch* between models.

Generation is considered an important [1] and complex aspect of LOD. It is responsible for providing the necessary reduction of complexity of a model based on an error assessment or a metric. The quality of the simplified mesh will depend on this process.

After the g*eneration*, a LOD algorithm has to select the appropriate model level of simplification to be rendered. Selection uses a metric to change between models. The metric can be based on distance to the viewer, 2-dimensional image analysis, importance of the model in the viewpoint and view speed.

In the case of *switching*, which consists of changing the model resolution, there are two major approaches: Alpha Blending and Geomorphing.

2.2.1 - Discrete Multi-Resolution

An initial approach for LOD was provided by Clark et al [13]. The idea consisted of generating many versions of a particular model with different mesh resolutions. Since this form of generation is a pre-processing step, there is no way of knowing which part of the model will be facing the viewer, so this approach is used in conjunction with view-independent LOD. This differs from the view-dependent LOD that performs the selection by making the mesh more detailed the closer it is to the viewer, provided that is within the field of view. The switching occurs between discrete versions of the model and it can be perceptible depending on the view position and the number of versions of the model.

Although it appears rudimentary, this technique has been widely used, especially in interactive applications [1], where speed is the main concern. One clear advantage is that the generation process is independent of the rendering. This means that a slow and high quality simplification algorithm can produce better approximations, since time is not a concern during preprocessing. It also lets the model be converted to optimized formats such as triangle strips or vertex arrays that can be rendered faster by specialized hardware. It also works well with memory hierarchies, since what is going to be rendered is known beforehand.

2.2.2 – Continuous LOD & View-Dependent Rendering

Continuous LOD is considered to be an evolution [1] of Discrete Multi-resolution. It uses simplification operators to iteratively reduce the complexity of the mesh. Instead of relying on a predefined number of simplified versions of a model, the idea is to obtain a specific (or close enough) model resolution. There is a substantial advantage in this approach in the form of the generation of a precise resolution for a given metric. It also has a small memory footprint, since only one version of the model has to be stored.

Because it requires the simplified mesh to be generated at run-time, this scheme is more computationally intensive than Discrete Multi-resolution.

View-dependent LOD is a specialization of this technique. It lets the model be simplified according to the viewer position. This is very useful for large models like terrains or scientific data. In the case of the former, for example, it is possible only to render with detail a patch where the viewer is located, while coarser patches are used as they recede from the viewer.

2.2.3 - Switching with Alpha Blending & Geomorphing

As good as a generation method can be, the purpose of making visually imperceptible the simplification of a model is defeated if the transition between resolutions is not well performed. *Switching* is a concept in LOD that deals with this problem. *Alpha Blending* and *Geomorphing* are two common switching techniques.

Alpha Blending consists of changing the alpha value of a transparent version of the model to make it opaque. At the end of this process, the original version is made transparent. This means that two versions will blend until the second version's alpha value becomes 1, turning it fully opaque. The transition is applied through a certain range that can be based on distance or a period of time.

One drawback of this approach is that, during the transition, the render system will have to cache and draw the two versions of the model. On the other hand, it provides good visual results. This technique is best used with Discrete Multi-resolution methods, since it has to use model resolutions with significant differences between each other [1].

An alternative to this approach is *Geomorph*. It is a switching method that does not require rendering two model resolutions at the same time. It works by interpolating the positions of vertices between two different model resolutions.

2.2.4 – Simplification Approaches

There are two general ways to simplify polygonal models: through iterative edge contractions or triangulation of a model or set of points. The former method is the one of interest in this work. The latter is covered in a number of simplification algorithms; one of the most discussed is Hoppe's Mesh Simplification [14].

The simplification operators are basic operations applied to a polygon mesh with the purpose of decreasing its complexity. Many simplification algorithms [6, 9] use these collapsing operations to iteratively decrease the complexity of the model.

An *edge collapse* is a simplification operator performed by "contracting" an edge as can be seen in Figure 2.4. This means that two vertices incident to this edge are removed. As a result, a vertex replaces the removed vertices. This vertex can be placed on the middle point in the edge, at one of its extremes or in the collapsed edge surroundings.

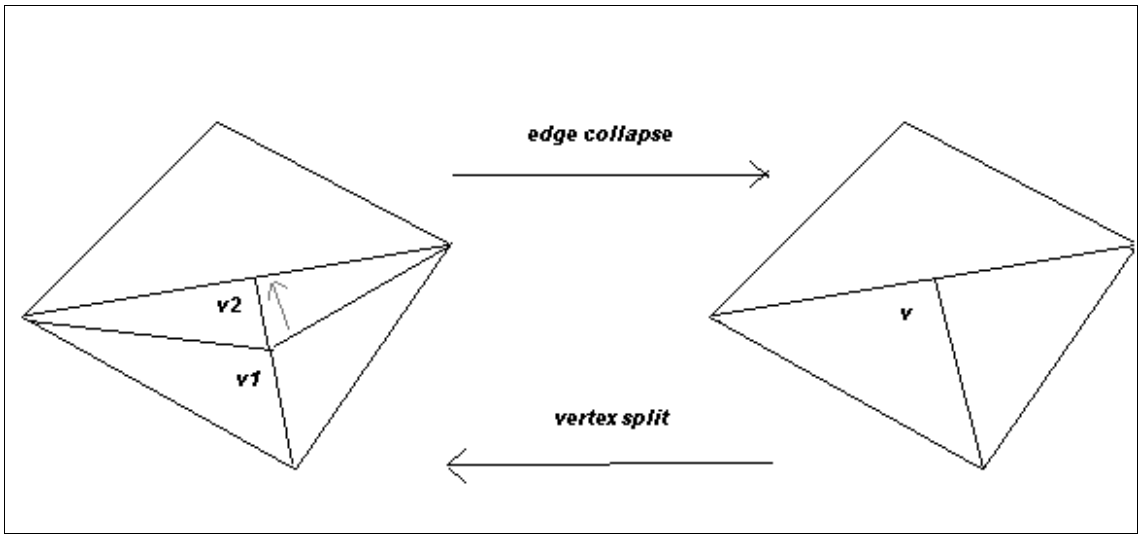


Figure 2.4 Edge collapse.

A *half-edge collapse* constitutes a special case of the edge collapse that works by removing a vertex and collapsing to one of the edges' endpoints.

In the case of the pair collapse shown in Figure 2.5, any two vertices can be selected for removal, whether they form an edge or not. Usually, the pair collapse is used with a predefined distance threshold so the algorithm does not have to consider all vertex pair combinations in the mesh. This form of collapse is useful to join disconnected meshes.

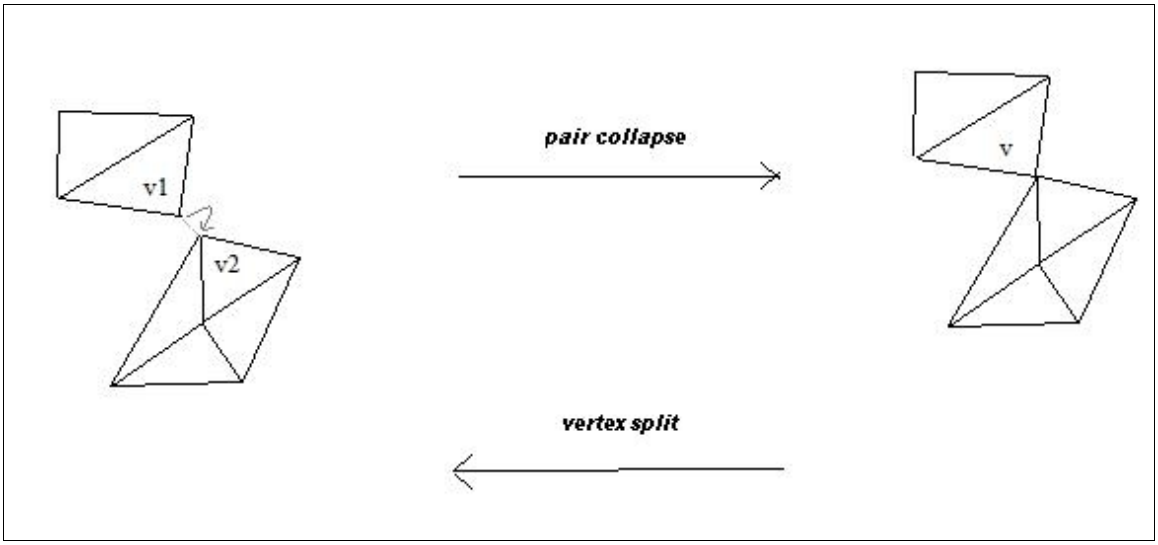


Figure 2.5 Pair collapse.

Some other simplification operators include the *triangle collapse* and *cell collapse*. They work by collapsing all the vertices incident to a triangle or within a cell to a single vertex. The latter is defined by an area usually defined by a radius. The resultant vertex could be new or chosen from the vertices to be collapsed.

Finally, the *vertex removal* operator works by removing a vertex and re-triangulating the affected area. A vertex removal can result in an operation similar to an edge collapse, but the re-triangulation process of the former could create edges and vertices that are different than the general process of collapsing edges of the latter.

Inconsistencies in the mesh can appear after the simplification operator is applied. Mesh *foldover* can occur after an edge collapse due to the newly created triangle. The new triangle "folds" over the other mesh triangles. This issue can be detected by checking for the normal changes of the triangles affected by the edge collapse. Another issue that can occur are topological inconsistencies that are generated by edge collapses that create non-manifold meshes.

2.3 – Metrics

In order to assess how the simplification process affects a geometric model, it is important to have some way of associating a cost with the decrease of model complexity. The strategy used by the algorithm depends on how this cost is calculated, i.e. the metric utilized by the algorithm.

Many different ways of assessing the cost of simplifying meshes have been suggested since the first simplification approaches appeared [1]. The general ideas were organized in three different groups that will be discussed in the following sections: Image-based differentiation, Euclidean Distance in Object Space and Curvature. These metrics are not mutually exclusive. On the contrary, many techniques involve the use of two or more of the discussed techniques. In fact, this work is also based on approaches of different metric groups.

2.3.1 – Image-Based Differentiation

This method compares different rendered points of view of a model with its simplified version. Each rendered pair of images (the original mesh and the lower resolution mesh) are differentiated at the pixel level.

Lindstrom and Turk [15] proposed a specific approach to this idea, where the edge collapse is driven by the resultant comparison of the rendered images of the mesh. The

camera is placed at 20 different vertices of a dodecahedron bounding the mesh and two groups of images are rendered: the original model group of images and the simplified version group. Then, the rendered images of the two different versions of the models are compared by computing the error of their pixel luminance values.

As pointed out by Luebke et al [16], this method can be very slow since it has to render the mesh 40 (2 x 20 camera positions) times whenever it compares them. It also can perform poorly if the rendering conditions (e.g. lighting, shading) of the model are different from the conditions of the original model. One solution proposed was to compare images by enabling predefined rendering conditions for both versions of the model. Another possible problem is that the number of sample images around the model could be insufficient to assess the model's complete shape, preventing a thorough comparison.

Since it computes the luminance difference at a pixel level this method is able to assess differences in normals, color, texture attributes and shading which is an advantage over metrics based only on Euclidean distance in object space.

2.3.2 – Euclidean Distance in Object Space

According to Luebke et al [16], simplification algorithms could be grouped by the metrics used to assess the model deviation from its original mesh. Four categories were created for the algorithms that use some kind of geometric inference: distance between vertices, distance between a vertex and a plane, distance between a vertex and a surface, and distance between surfaces. Each one of these classifications and their associated simplification algorithms will be explained in the following sections.

2.3.2.1 - Distance Between Vertices

This category considers the geometric distance between the original model vertices and the simplified model vertices. However, this type of approach can lead to some imprecision. If two triangles have their edges swapped, this will not be captured as a vertex deviation, even though it leads to a surface deviation.

Vertex Clustering [17] is an algorithm that fits in this category. It works by arranging the polygonal surface on a uniform spatial grid. The vertices that are within each grid cell (or cluster) are collapsed to a representative vertex that is calculated by a weighted combination of the vertices in the cluster. The redundant edges and triangles resulting from this operation as well as the isolated vertices are removed. The error is computed by mapping the maximum distance between the previous and current position of the vertices.

Some other algorithms introduced improvements by using hierarchical subdivision in the grid [18], choosing cell grids according to the best representative vertices, or even positioning cell grids based on curvature information [19].

2.3.2.2 – Distance Between Vertex And Plane

This category includes a method that is considered to be fast and efficient [4, 16] when compared to other mesh simplification methods. The distance of a point p with coordinates (x, y, z) to a plane, given the distance X to the origin and the unit normal n, is

$$d = n \cdot p + X \qquad (2.5)$$

This calculation is cheaper than the computation of the distance between two points in space.

Ronfard and Rossignac [20] proposed a method that uses edge collapses as the simplification operation. The algorithm associates to a simplified vertex a set of supporting planes. In the original mesh, each vertex has one supporting plane for each adjacent face. The maximum distance of the vertex to the set of planes is chosen as the error. After the edge is collapsed, the set of planes of each vertex are merged and the redundant planes are removed. As the simplification progresses the sets of planes of the preserved vertices keep growing.

Measuring the vertex-to-plane distance can lead to some inaccuracies due to the distance from a vertex to its supporting polygons being underestimated or overestimated. For example, the maximum distance to the planes could be the shortest distance to the supporting polygons in the original mesh. This can be seen in Figure 2.6.

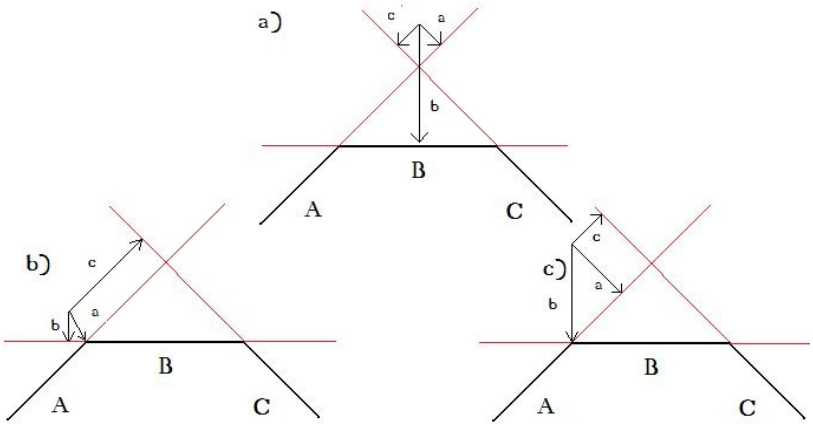


Figure 2.6 The distance of a simplified vertex to its supporting planes. (a) The shortest distance to the supporting polygons, b, is calculated as the maximum distance. In (b) the maximum vertex-plane distance, c, overestimates the shortest distance to the supporting polygons. In (c), the maximum vertex-plane distance, b, underestimates the shortest distance to the supporting polygons.

Garland and Heckbert [9] proposed a similar approach while modifying the way the planes would be carried over through the simplification steps and also by allowing the merging of nearby vertices. Instead of defining a union of planes, this approach sums the

squared vertex-plane distance. Since the QEFC approach makes use of this algorithm, this will be explained in more depth in Chapter 3.

2.3.2.3 – Distance Between Vertex And Surface

This class of approaches involves mapping one surface to another by relating the vertices of the model to their closest points on the simplified surface. The resulting distance between the mapped points is considered to be the error. According to Luebke [16], such algorithms tend to be slower than vertex-plane distance based techniques, because appropriate mappings between the original and simplified vertices have to be found by sampling the surface many times.

Mesh Optimization [14] works by comparing the distance between the surfaces of the original mesh and the simplified mesh. Later, Hoppe modified his initial algorithm to provide continuous LOD [21]. Progressive Meshes add a high-level data structure that would store the base model and the refinement operations. This algorithm also provided smooth *switching* between the different model resolutions, the capacity to refine selected areas of the model and the ability to compress the mesh.

Mesh Optimization works by defining inner and outer simplification procedures. The metric is defined by the sum of the squared distance from the set of vertices of the simplified model to the surface. The inner problem is solved first and the algorithm moves the vertices until the error is minimized. The outer problem is defined by the operations that change the number of vertices and their connectivity while minimizing the error. The outer problem results on topology changes to the mesh. The algorithm repeats these operations until convergence.

The Progressive Meshes algorithm works by evaluating a potential edge collapse. A new vertex v is generated with a defined initial position, e.g. one of the edge vertices or the midpoint of the edge. Every vertex of the original mesh is then mapped to the closest

vertex of the simplified mesh. The position of the vertex *v* is moved to the location where the sum of the squared distances of its corresponding mapped vertices returns a minimum distance. After the vertex *v* is moved, the mapping has to be updated, since the relation of proximity between the vertices could be affected. The vertex *v* is then moved again and the process is repeated until the solution converges.

The system may not converge when an optimal vertex position is at infinity or when none of the vertices of the original model map are close to the vertex *v*. In both cases, moving the vertex could decrease the error or not affect it at all. Hoppe's solution to deal with this is the *spring energy* term. This term allows the neighboring vertices to influence a pulling force to attract *v*. As the simplification progresses, the spring term will lose its force. Also, as with the Maximum Supporting Plane Distance algorithm described previously, the computation to determine the next edge collapse will increase, since more preserved vertices will be mapped to the faces incident to the edge. The energy function of the Progressive Mesh algorithm is represented as:

$$E(M) = E_{dist}(M) + E_{spring}(M) + E_{scalar}(M) + E_{disc}(M) \qquad (2.6)$$

E_{dist} and E_{spring} represent the distance deviation and the spring energy term, respectively. E_{scalar} represents the color and other scalar errors. The E_{disc} term protects the discontinuities on the attributes, like the normal, by defining a penalty. If there is a simplification operation that alters the discontinuity, the E_{disc} is assigned in order to increase the error caused by this operation.

2.3.2.4 – Surface-Surface Distance

The simplification algorithms included in this category are the ones that consider all points on the simplified and the original model. These methods are considered to be slower [16] than other Euclidean distance categories because they work by minimizing the maximum error.

Simplification Envelopes [22] [23] is a simplification method that uses the surface-to-surface distance as a metric. It works by creating the inner and outer versions of the geometry that are defined by a threshold. The vertices are placed in a queue and the algorithm tries to remove a vertex if the resulting triangulation does not intersect the envelope surfaces. The process goes on until there is no vertex to be removed.

Mappings in the Plane [24] is another surface-to-surface simplification method that orthogonally projects adjacent faces of a vertex onto a plane before and after its removal. Then it calculates the mutual tessellation of the two sets and finds the maximum error by computing the distance between edge crossing points of that mutual tessellation. Mappings in Texture Space [25] is a similar method but instead of using projections onto a plane, it uses the deviation of the texture coordinates.

2.3.3 – Curvature Measurement

Algorithms that make use of the euclidean distance between vertices, edges or faces to determine a metric for simplification often make large changes in high curvature regions of a model. This could lead to poor simplification results if the prominent features of a model are found in this type of region. The reason is because geometric distance does not always correlate with visual difference. In order to develop methods that take into account the high curvature regions, methods that measure the curvature were proposed.

Typically, model generation methods such as manual modelling or 3D scanning result in 3D meshes that often are irregularly sampled. This means that there is a variation of connectivity and distribution of triangles across the model. Techniques that provide curvature measurements, like feature detection, find estimations of differential quantities, providing information about the main features of a model that can help correct these issues.

Discrete curvatures have been proposed as a way to measure error in geometric features with high-curvature even though they have a small distance metric. Turk suggested a method [26] that simplifies shapes by using a sphere to approximate a surface to its radius to determine the curvature. Another method proposed by Hamman [27] estimates the curvature at each vertex by a least square fitting of a paraboloid to the vertex and its neighbors. This work focuses on the proposals that base the discrete curvature on the Gaussian and mean curvature known as the Gauss-Bonnet scheme. Among parabolic fitting, Surazhsky et al [28] considered Kim's work to be one that generated the closest results to the analytically computed values of the Gaussian and mean curvatures.

The Gaussian and integral mean curvatures are calculated by the use of the angle and face information that is related to a vertex v. The former curvature is defined by

$$K = \int_s K = 2\pi - \sum_{i=1}^{n} \alpha_i \qquad (2.7)$$

where α is the angle between two successive edges e_i and S is the area of the adjacent faces around vertex v. The integral mean curvature is computed using the formula

$$H = \int_s H = \frac{1}{4} \sum_{i=1}^{n} \|e_i\| \beta_i \qquad (2.8)$$

where $||e_i||$ is the length of the edges incident to v and β_i is the dihedral angle of the edge e_i. A dihedral angle of an edge e is defined by the angle between the normals of the edge's adjacent triangles.

The sum of the absolute principal curvatures, $|k_1|$ and $|k_2|$ can be computed from the relations

$$K = k_1 k_2 \qquad (2.9)$$

and

$H = (k_1 + k_2) / 2$ *(2.10)*

which results in

$$k_1, k_2 = H \pm \sqrt{H^2 - K} \quad (2.11)$$

Finally, the sum of the absolute principal curvatures $|k_1|$ and $|k_2|$ is defined in relation to the Gaussian and Mean Curvatures. Note that the equation

$$|k_1| + |k_2| = \begin{cases} 2H & if\ K \geq 0 \\ 2\sqrt{H^2 - K} & otherwise \end{cases} \quad (2.12)$$

result is always a real number, even if K is greater than H^2. This equation has a condition for K to guarantee that it will always result in a real number.

2.4 – Feature Based Mesh Simplification

Feature based mesh simplification or the integration of the QEM algorithm with feature detection approaches have been proposed by several authors [5, 6, 39]. It is important to discuss how methods that identify features based on curvature information work.

Chen and Nishita introduced an algorithm that segments the unstructured meshes of a geometric model into several parts by using feature detection methods, and then simplifying each part of the meshes iteratively.[39] The algorithm finds all feature and base edges on the mesh. The former are defined by the angle between the faces adjacent to the edge. The classification of a base edge occurs if the dot product of the normal of two vertices that share an edge is less than a predefined threshold. The feature and base edges are defined as un-removable so the simplification process never discards them. All other edges are sorted in order of their descending weight. Then, the algorithm selects an

edge on the top of the list and applies half-edge collapses to them. After a half-edge collapse, the neighboring edges have their weights re-evaluated. This process is repeated until there are no removable edges.

Although, Chen and Nishita's approach joins two neighboring feature edges and allows the simplification of the interior points, it does not provide identification of feature curves, thus avoiding their gradual simplification. The algorithm requires the use of different predefined thresholds during the simplification process. Those thresholds are set manually and depend on the model being simplified. Also, if the mesh is simplified and the feature points are not gradually discarded, then they could cause early discarding of vertices that cause great geometric deviation.

Kim et al [6] define the features based on local properties related to the Gaussian and Mean curvatures. They proposed the use of a distance metric with the calculation of the curvature of the model as another error metric. The curvature information for each vertex is summed with the Quadric Error to generate the total deviation. Another metric used in their work is the *tangential error* calculated by the magnitude of a difference vector between two normal vectors of tangent planes. The authors proposed associating the above curvature and tangential information to an error metric, where this value would be used to define how much a simplification step costs.

The function is defined as

$$DDEM(v) = Qv(p) + Tv(p) + Cv(p) \qquad (2.13)$$

where Qv is the quadric cost of the contraction. It is based on the concept of quadric error metric proposed by Garland et al [9]. Tv is the *tangential error metric* based on the magnitude of a difference vector between two normal vectors of tangent planes. Cv is the curvature based on the sum of the principal curvatures defined in (2.12). These costs are placed on a priority queue and the simplification step that provided the least cost would be on the top of the queue.

As with Chen and Nishita's method, the approach proposed by Kim et al does not provide global identification of features and feature curve simplification. If the feature's relative importance is too high, important faces based on geometric deviation could be discarded. The opposite would happen if the features are not given an appropriate weight in relation to the quadric error cost. This would require finding an appropriate parameter not only for the model but also for the level of simplification.

Wu et al suggested using an approach that is based on edge contractions and the QEM.[5] It works by finding feature edges based on the face angle in order to build feature curves. The edge weights are based on a predefined global weight and the number of feature edges of the curve. The algorithm uses the QEM method to simplify a mesh by applying an edge weight to the edge's contraction cost.

Although this algorithm takes a global approach to the feature detection and classification, it does not provide simplification of the feature curves. Since all edges in a curve are assigned the same weight, there is no way to delay the loss of edges that have a high relative important in the formation of the curve's shape compared to edges in the curve that are not as relevant to the curve.

Chapter 3 - Feature Based Geometry Simplification

In this thesis, the quadric error metric [9] algorithm is used as the basic simplification algorithm. This method is known as an efficient mesh simplification method [4, 6]. Feature Detection for Surface Meshes[7] is used as a complementary technique to identify parts of the important sections of the model that have to be preserved through the simplification process. The chosen feature points are represented by non-intersecting curves that are simplified by the Douglas-Peucker line simplification method [8] used in cartography.

Instead of considering only local importance of features, Feature Detection for Surface Meshes [7] identifies and builds features using a curve representation scheme. The curvature information is identified through a comparison within the local neighborhood in order to add edges to the feature curve. It also takes a global approach by breaking, merging and filtering curves depending on rules that take into consideration the curve's length, the end points' relative importance and other neighboring feature curves.

QEFC does not make use of predefined weights to promote the integration of techniques. The algorithm uses the Douglas-Peucker algorithm to simplify the feature curves. It sorts the feature vertices in order of their geometric deviation in relation to their respective curves. If an appropriate number of features is selected, then the algorithm will discard the least important vertices relative to the detected feature curves. The appropriate value for the feature importance will depend on the number of feature vertices detected and on how the feature curves match the model's saliences. We have found it difficult to give a universally useful value for the setting of that parameter and instead propose setting it interactively by the use of our system.

3.1 – Mesh Simplification Algorithms of Interest

In this chapter the relevant aspects of geometric model representation and simplification will be reviewed. This section will cover the techniques selected to be part of this thesis.

The quadric error based mesh simplification algorithm proposed by Garland was chosen as the basic simplification algorithm. This approach is known to be efficient [4] while providing good quality when compared to other simplification methods [6]. Some systems use the quadric based method as a standard to compare against other algorithms [30] or as the basis of some new approach [6, 5].

3.1.1 – Quadric Error Metric Based Mesh Simplification

Garland suggested an approach that would accumulate the distance distortion caused by iterative pair contractions. The idea is to sort all possible pair collapses by the quadric error metric and iteratively pick the contractions with increasing order of quadric error.

This algorithm could be briefly explained in the following table:

1 – Calculate Quadric Matrices for each vertex $\mathbf{v_i}$ in the original model. 2 – Find all edges or the vertex pairs ($\mathbf{v_i}$, $\mathbf{v_j}$) within distance ε from each other. 3 – For each vertex pair, compute the best position to place the new vertex $\mathbf{v_n}$. The cost represents the error $\Delta(v)$ of $\mathbf{v_n}$. 4 – Insert the vertex pairs in a priority heap. 5 – Remove from the heap the vertex pair with the least cost, contract selected vertex pair and recalculate the cost of the vertex pairs affected by the contraction. 6 – Repeat 5 until the heap is empty or the number of faces is equal to a selected value.

As with Hoppe's algorithm presented in the second chapter, Garland's approach works by iteratively simplifying the mesh. It organizes vertex pairs (within a distance defined by a threshold) in a heap with increasing order of cost. The metric used is based on the association of planes with the vertices. This method can also aggregate unconnected parts

of the mesh, which is believed to provide better quality when compared to the half-edge collapse version of the same algorithm.[9]

QEFC is based on the half-edge collapse but it does not create a new vertex like in an edge collapse; instead the vertex that is preserved is always the one of the two vertices remaining from the contraction. For this reason, QEFC uses a heap queue based on vertices instead of edges. Edge collapse is useful for some type of models but it has a drawback. It has been demonstrated that it is difficult to work with in practice, since it uses a parameter that has to be adjusted for different geometric models [29]. A threshold t defines the maximum distance between vertex candidates that has to have an appropriate value. Too large of a value makes the number of vertex pairs grow quadratically. If t is too small the technique would not have a noticeable effect since it would miss most of the vertex pairs.

The implementation developed in this work also preserves boundaries by applying large weights to boundary edges. However, it does not support texture or color information. It also does not necessarily prevent mesh fold-over, since this requires an appropriate threshold for each model.

The QEM algorithm associates a set of planes with each vertex of the model. These planes are represented as quadric matrices and initially coincide with the triangles that are incident to a vertex. As the simplification progresses the distance from a vertex to its set of planes increases. This distance is used as a guide to show which simplification step causes the minimum deviation. The algorithm repeats this operation until the desired number of polygons is reached or when the error reaches a defined value.

In order to calculate the distance from the vertex to the set of planes, the quadric error based method uses 4x4 symmetrical matrices called fundamental error quadrics. The initialization of this algorithm computes all the quadrics for the initial vertices.

Each plane associated to the vertex is defined as $p = [a\ b\ c\ d]^T$, where it represents the plane defined by the equation $ax + by + cz + d = 0$ where d is a scalar constant and $[a\ b\ c]^T$ is a unit normal. The distance is calculated by

$$\Delta(v) = v^T \left(\sum_{p \in planes} F \right) v \qquad (3.1)$$

where F is the fundamental error quadric defined as

$$\mathbf{F} = \mathbf{pp^T} = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \qquad (3.2)$$

The sum of all fundamental error quadrics of a vertex v can be represented by the quadric Q. Figure 3.1 shows that the quadrics have a geometrical meaning, as pointed out by Garland. They can be represented by ellipsoids that are centered on each vertex. They also show that the new vertex can be moved around the ellipsoid while respecting an error threshold.

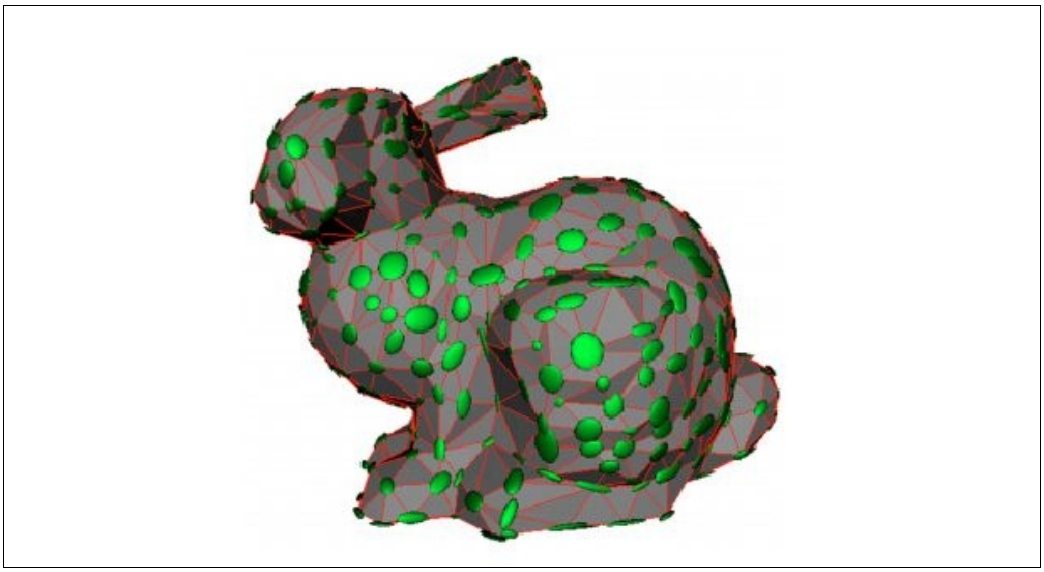


Figure 3.1 According to Garland [9], the error quadrics have a geometric meaning. They can be represented by ellipsoids that are centered around each vertex. They also mean that the new vertex can be moved around the ellipsoid while respecting an error threshold. Note that the ellipsoids are adapted to the shape of the surface (Image taken from [9]).

As can be seen in the equations, the quadrics can be summed, to represent the set of planes. $\Delta(v)$ represents the distance to the plane or the error metric. Initially, this distance is zero, given that the vertex is at the intersection of its planes. This means as more contractions are applied to the mesh, the distance of the vertex to its set of planes is likely to increase as the number of planes does.

This method of storing and summing the planes' information as quadrics can reduce the amount of memory and processor resources necessary to perform the simplification when compared to the approach by Ronfard and Rossignac [20]. However, this does not come without a disadvantage. There may situations when a plane is summed multiple times. But since a plane is the representation of a single triangle, it can only be overlapped three times. Garland suggests that this problem is not important when compared to the benefits in the small memory footprint and the small cost to sum the matrices instead of computing the more expensive set union operation proposed by Ronfard and Rossignac [20].

After the calculation of the initial quadrics, the algorithm selects valid vertex pairs. This means that the vertices of a pair have to be within a distance ε of each other or connected by an edge. The idea is to provide for the possibility of aggregation of different parts of the model. Then, for each vertex pair, the algorithm calculates the optimal position of the new vertex v_n using an equation to minimize the error shown in equation (3.3), aggregating each vertex's quadrics in the form of Q_n. The idea is to create a new vertex v_n where the error $\Delta(v)$ is minimal. Garland demonstrated by taking the partial derivatives that it is equivalent to solving the equation as in

$$v_n = \begin{bmatrix} q11 & q12 & q13 & q14 \\ q12 & q22 & q23 & q24 \\ q13 & q23 & q33 & q34 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad (3.3)$$

If the matrix is not invertible, the algorithm tries to find an optimal position along the edge formed by the vertex pairs. If this strategy does not work, then it chooses one of the endpoints of the edge or its midpoint.

The costs associated with each contraction are stored in a priority queue. This means that the vertex pairs are listed in an increasing order of the error their removal will produce. After all the vertex pairs have their cost evaluated, the algorithm selects the one with the least cost to be removed from the queue. The edge is contracted and the new vertex is placed on it or, if it is a half-edge, the edge is contracted to one of its endpoints. Then the vertices affected by the contraction have their costs reevaluated.

The quadric error algorithm uses some techniques in order to help maximize the quality of the simplification. If a mesh has boundaries, which means that an edge is part of only one triangle, the associated cost of the boundary edge will be increased significantly. This increase in the cost has to be defined by the user. A variant of the algorithm also applies weights to prevent discontinuities of color or texture [31]. Another technique associated is the one that prevents mesh fold over by penalizing or avoiding a contraction if a face

flipped. One way of doing this is to define a threshold to check if the face's normal changed significantly after a contraction. Another method used by Garland and Heckbert is to define perpendicular planes to the edges surrounding a vertex pair. This defines a region where the new vertex can be placed without inverting the mesh.

3.2 – Feature Detection

As explained in the second chapter, curvature measurement can lead to identifying features of a model. Usually 3D models have an uneven distribution of triangles and connectivity [32]. Also, models can have noise caused by techniques like mesh extraction or 3D scanning. Although there are algorithms to deal with these problems, like smoothing and denoising, they can cause deformations to the mesh. Another way to reduce noise is to detect patterns and identify features through the use of curvature information or vertex connectivity.

According to Zhong et al [32], the most popular feature extraction methods use a technique called point extraction. It consists of detecting feature points that have large curvature and then are connected to form a feature line. Another method detects the feature points by calculating the principal curvature of each vertex [33, 34]. These algorithms make use of the points' local neighborhood instead of looking at them globally. Surface extraction methods create a surface that shares geometric properties like normals and curvature with the original model's surface. The generated surface is intersected with the original surface. The third category of feature detection methods works with the use of the Mesh saliency approach [35, 32]. The technique finds unique regions of a model by comparing it to its surrounding, which is a mechanism present in models of human vision.

Feature Detection For Surface Meshes (FDSM) [7] defines a set of rules to identify features in models with specific properties. Some of these rules are used specifically on models with normal discontinuities, but at the same time, they are flexible enough to

identify features like ridges and corners on meshes with smoother properties. FDSM is sub-quadratic in time over the size of the input mesh ($n + mlogm$, where n is the number vertices and m is the number of feature edges) while generally it is capable of differentiating noise from a feature.

The algorithm details two classes of features: *1-dimensional* (*1-feature*) and *0-dimensional* (*0-feature*). The former represents the smooth feature curves that do not intersect each other at their interior points. A 0-feature is a feature point that is not contained in the interior of any 1-feature. 0-features, called corners, can connect 1-features. A 1-feature, also known as a ridge, cannot intersect another 1-feature without a corner. 1-features can be closed or open.

A mesh is composed of *0-dimensional cells* (vertices), *1-dimensional cells* (edges) and *2-dimensional cells* (faces). Feature detection identifies these feature cells to build feature curves.

The rank is the number of feature curves incident to the 0-feature. The feature points shown in Figure 3.2 are called tip (*rank 0*), terminus (*rank 1*), turn (*rank 2*) and junction (*rank 4*). A tip has no feature curves incident to it. The terminus, has one feature curve, while the turn has two and the junction, three or more. A 0-feature incident on only one closed 1-feature has rank 2, instead of rank 1. In general, the smaller the rank of the 0-features the harder is to identify the feature.

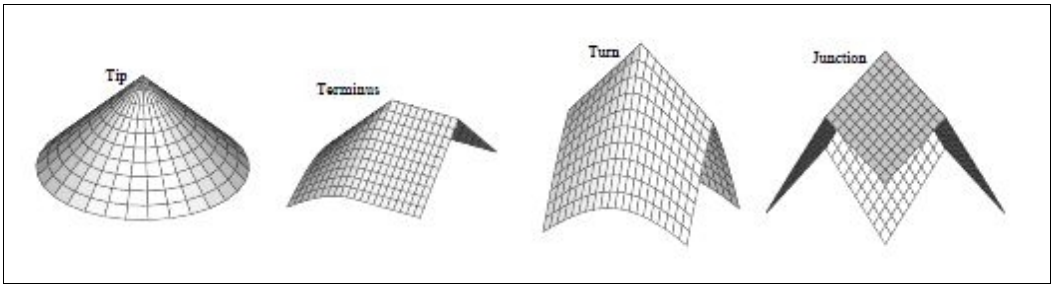


Figure 3.2 Image showing the various ranks of 0-features (Image taken from [7]).

The FSDM algorithm [7] is presented below:

```
1 -  Find all strong edges in the mesh and store them in a priority queue with decreasing
     order of strongness

2 – for each edge in the queue
          if it has not been visited, create a curve  γ
          while strongest edge in relation to both ends of γ has not been visited
                    find strongest edge g in relation to the front or back of γ
                    if g is not visited, store g in curve γ and mark it as visited
          end while
          break γ into sub-curves at strong vertices
          for each sub-curve
                    if the sub-curve is not false-strong then make sub-curve a feature curve
          end for
     end for
3 - break feature curves at rank 3 vertices
4 - merge curves at false-strong vertices of rank 2
```

Feature edges are used to identify strong curves. Strong edges are identified through numerical approximation to the curvature of the surface and of the feature curves.

To determine if an edge *e* is strong, the face (dihedral) angle has to be calculated. The face angle ^*e* of an edge is the angle between the normals of the incident faces of this edge. Given $\theta \in [0, \pi]$, the edge is strong if the angle is greater than θ.

According to the authors, this method is not sufficient to determine edge strongness due to its inability to handle noise, because the geometry information is very local. Also, it is hard to find an appropriate value for θ that can work for different models. For these reasons, the concept of relative *θ-strongness* is defined.

^*(e, g)* is the angle between edges *e* and *g*. If *e*^*g* forms a curve, ^*e* and ^*g* must be relatively large, meaning that ^*(e, g)* has to be relatively small. This motivated Xiangmin

and Heath [7] to define a weighted face angle as *w(g, e)* = |*cos* ^*(e, g)*| ^*g*. The representation of the face angle is shown in Figure 3.3.

Given a value *r* >= 1 and $\theta \in [0, \pi]$, the edge *g* is *r-θ-strong* in relation to *e* if ^*g* is equal or greater than ^*(e, g)* and *w(g, e)* equal or greater to θ, and *r* times larger than the weighted face angle of all the edges incident to *e*^*g*. This is represented in Equation 3.4.

$$s(g, e, r, \theta) \equiv \hat{} g \geq \hat{}(g, e) \text{ and } w(g, e) \geq max\{\theta, \max_{g \cap e \in h \text{ and } h \neq g}\{rw(h, e)\}\} \qquad (3.4)$$

An edge *g* is *url-strong* if it is θ_u*-strong* or *r-*θ_l*-strong,* where θ_u is the upper-bound of weaknesses and θ_l is the lower-bound of *strongnesses*. It is *url-stronger* than *h* with respect to *e* if it is θ_u-strong and its face angle is greater than at *h* or if it is r-θ_l-strong.

If *g* is the *url-strongest* in relation to *e*, it is said *e* is *url-connected* to *g*. If *e* is θ_u-strong and there is a url-connected path along a curve, this curve is said to be *url-strong*.

Still, perturbations in the mesh can make the algorithm erroneously classify edges as features. In order to detect these edges, the curve is broken into sub-curves at strong vertices.

The algorithm defines strong vertices by their ranks, which means the number of the incident feature edges of the vertices, or by calculating its edge angle as shown in Figure 3.3. The latter is the angle between the vertex's incident edges. One more time, the *url-strongness* concept is used.

Given a strong curve γ, a vertex *v* is considered to be r-θ-strong if its edge angle is greater than θ and if the edge angle is r times greater than the edge angles of the neighbor vertices in γ. A vertex is url-strong if it is θ_u-strong (edge angle equal or greater than θ) or *r-*θ_l*-strong*. At end vertices of the curve, the edge angle is considered to be π.

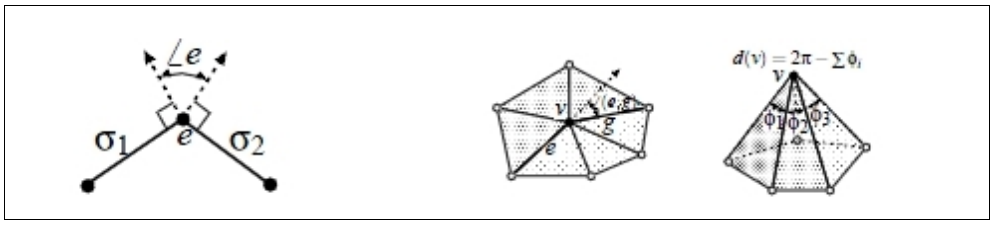


Figure 3.3 Edge Angle (left), Face Angle (center) and Angle Defect (right).

The two definitions of vertex strongness presented take into consideration its role in a given curve. Xiangmin and Heath propose another way of assessing the strongness of a vertex that is independent of the feature edges, meaning that it is useful to be applied with the filtration rules. The angle defect that is represented in Figure 3.3 is the difference between 2π and the sum of the incident faces of a vertex, given by *^(v, σ)*. The formula is shown in

$$ad(v) = 2\pi - \sum_{v \in \sigma} \hat{}(v, \sigma) \qquad (3.5)$$

It must be noted that the angle defect resembles the Gaussian curvature. It measures how much a set of incident surfaces of a vertex deviate from being flat. The url-strongness is also defined for the angle defect of a vertex *v*. A vertex is r-θ-strong if *ad(v)* is greater than θ and *r* times greater than the angle defect at its neighboring vertices.

Filtration rules are used to check if a sub-curve is false-strong. There are two rules: the short-falseness rule and the long-falseness rule. The first rule filters out short curves with end vertices weak in angle defect and at least one of the end vertices not connected to a feature. The rule states that, given a user-defined parameter *l*, and the number of weaknesses defined by *b*, a curve is false-strong if one of its end vertices is not connected to a feature, *b* is greater than 0, and the number of edges is smaller than *bl*.

The second rule is used to discard long curves that run along an equally long feature curve. This is caused by some mesh generation methods. A sub-curve is long-false if none of its end vertices is strong in angle defect or is a known feature, and every vertex of the sub-curve is adjacent to a feature curve.

3.3 – Line Simplification

In order to simplify the feature curves that were detected with the previous algorithm, the curve simplification algorithm called Douglas-Peucker [8] is used. This a method commonly used in cartography. This approach is considered to generally provide good quality results when compared to other simplification algorithms for non-intersecting curves [36].

Its implementation generates an approximation of a sequence of vertices $v_0 \ldots v_i \ldots v_n$ based on an error threshold ε, defined as a parameter. It starts by analyzing the segment v_0-v_n. It finds the vertex v_i that has the greatest distance to this segment given that it is greater than distance ε. The algorithm splits the curve in two segments (as shown in Figure 3.4) based on the vertex v_i , and calls itself twice using these segments (v_0-v_i and v_i-v_n) as parameters. Any point within the segment that is less than ε is discarded.

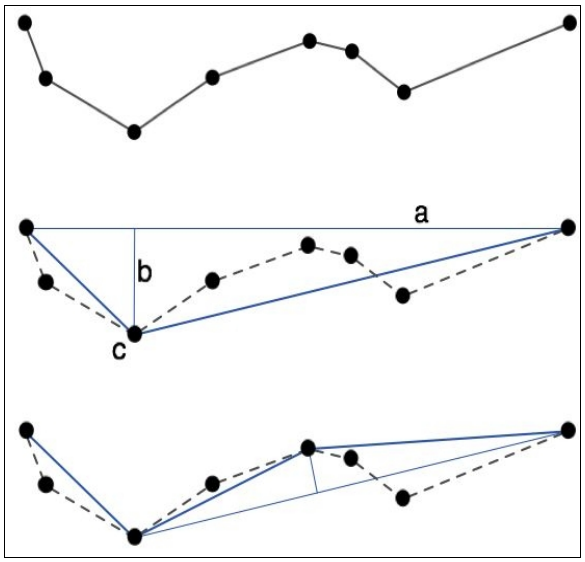


Figure 3.4 Douglas-Peucker simplification process. Point *c* has the greatest distance to the segment *a* (Picture taken from Wikimedia Commons).

The algorithm is described below:

```
Simplify(v0, vn)
        Find vi, the most distant vertex from v0-vn.
        If vi's distance is greater than ε, call simplify(v0, vi) and simplify(vi, vn)
        Else discard (vi)
End Simplify
```

The distance is calculated by the analysis of the position of the vertex v_i in relation to the segment. If the dot product of the segments v_i-v_0 and v_0-v_n is equal or less than 0, then the distance is v_i minus v_0. If it is positive and greater than the dot product of v_i-v_n by itself, then the distance is v_i minus v_n. Otherwise, the distance is v_i minus a point within v_0-v_n that forms with v_i a perpendicular segment to v_0-v_n.

3.4 – Integration of Techniques

The central idea presented in this thesis is to use the QEM algorithm to reduce the Euclidean deviations caused by the simplification process and at the same time prevent the Quadric metric from affecting high curvature portions of the model through the

preservation of the features. The algorithm will first identify feature curves. Then, these feature curve vertices will be simplified to a user-specified level using a specialized algorithm [8]. Finally, the remaining feature curve vertices will be "frozen" and the QEM algorithm will be applied. Figure 3.5 shows the basic algorithm flowchart.

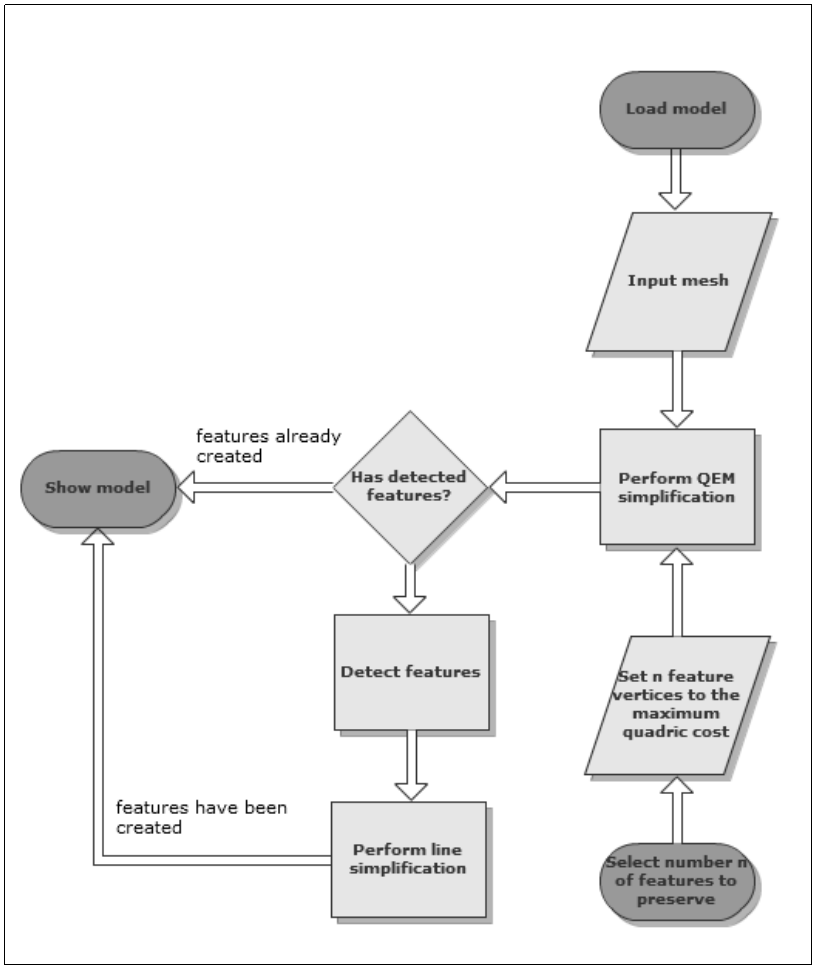


Figure 3.5 The flowchart for the QEFC algorithm. The dark grey elements represent the start/end states of the algorithm.

The goal is to obtain good results when compared to the Quadric Error Metric (QEM) algorithm [9]. The QEM algorithm has been demonstrated by Cignoni et al [4] to be a balanced solution among mesh simplification algorithms. Its flexibility also permits relatively easy integration with different approaches [6].

The feature detection algorithm sets well-defined rules and conditions that take into consideration isolated spatial structures and their relative importance in a model. It also provides its results as a simple and concise structure of curves with no intersections. Finally, the Douglas-Peucker algorithm, used in cartography, is employed to determine the importance of the points of these curves.

QEFC algorithm is described below:

First Pass

1 – Simplify the mesh using only the quadric error metric algorithm (create heap h_1)
2 – Run the Feature Detection algorithm to obtain feature curves in the original model
3 – Display original mesh and feature curves

Second Pass

1 – For each feature curve, execute algorithm in section 3.3 with ε set to zero
2 – Store all the results in one separate heap h_2 ordered by the descending Douglas-Peucker distance
3 – Select the number of feature vertices in h_2 by preserving *n* (defined by the user) vertices at the top of heap h_2
4 – Modify the half-edge collapse heap h_1 (quadric error cost) by setting the *n* preserved feature vertices to the maximum quadric cost
5 – Remove from the heap h_1 the vertex pairs with the minimum cost, contract selected vertices
6 – Display original mesh and feature curves

Some methods [5, 6] that are used in conjunction with the quadric error algorithm influence the importance of the edges through changes in the cost of the quadric error algorithm. These methods use multiplications or summations to the cost of the quadric error approach. However, the strength of this influence can be difficult to determine. Thus, it was defined that the solution was to let the user choose the best number of feature vertices to be preserved.

First, the QEFC implementation of the quadric error metric algorithm is executed and the feature curves are computed. The second pass executes the Douglas-Peucker algorithm

with the threshold ε set to value zero. This means that the line simplification algorithm will compute the distance of every vertex that is part of the feature curve. The distances are stored in a heap, and the top *n* vertices in this priority queue are marked. Since the initial quadric matrices do not need to be recomputed, the quadric algorithm part will start by the construction of the edge collapse structure. However, this time the quadric error of the marked vertices will be set to the maximum value allowed, which puts them at the bottom of the quadric priority queue, deferring as much as possible their simplification by the quadric algorithm.

If any parameter of the FDSM algorithm is changed to preserve a different number of feature vertices, the first and second passes have to be executed again. In case that only the number of feature vertices is changed, only the second pass has to be executed again.

In order for the user to be able to view the mesh simplification or refinement, after any of the two steps, it is possible to iterate over the edge contractions/splits interactively. This means that, after the computation of first and second passes, the user can quickly see the effect of the parameter selections on the model.

Chapter 4: Results and Performance Analysis

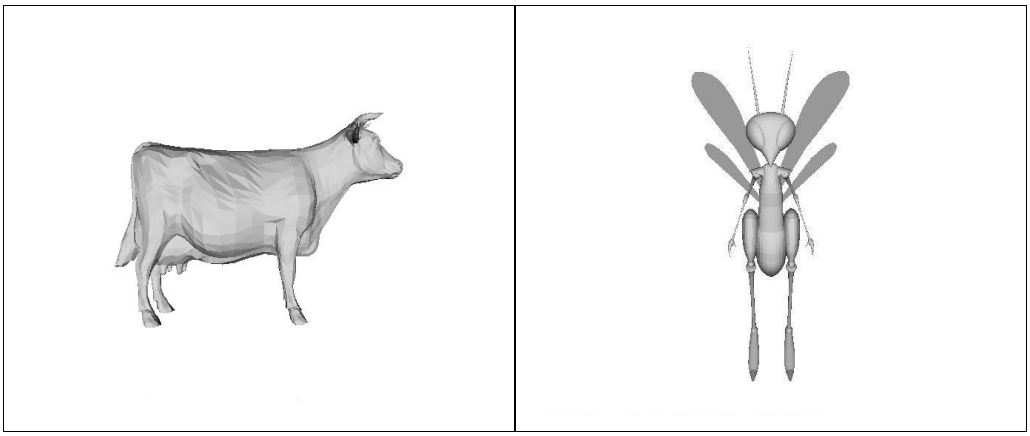
This chapter will experimentally evaluate the use of the algorithm introduced in Chapter 3. The efficiency of the method and the geometrical difference between the simplified model and the original will be analyzed. First, a brief and empirical evaluation of the time and space complexity will be given, and then the results will be presented. The quality will be measured both in terms of empirical evaluation and with the help of the Metro analysis tool.

4.1 Models Used

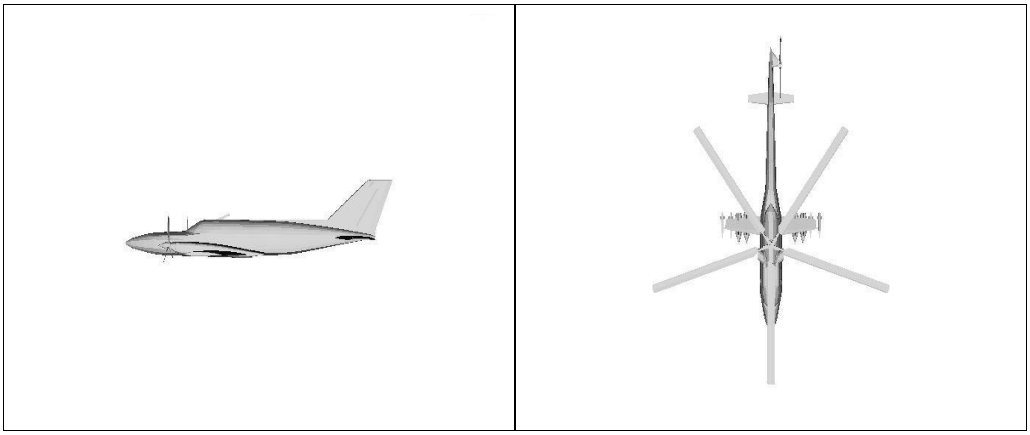
The models were chosen according to the number of polygons and shape. It was expected that models with normal discontinuities would have better results when compared with smoother models, since the feature detection rules are useful for the former. In fact, some models with smooth properties do not have any or very little improvement with this method when compared to quadric error metric. Tests with the widely used Stanford Bunny model will help demonstrate this.

The distinct features that are very well preserved by this method are the long, thin and flat parts of the models. This means that features such as propeller blades for helicopters and airplanes, and wings for characters have a high likelihood of being preserved with the QEFC algorithm. This is where the this algorithm can make the most difference when compared to other simplification algorithms.

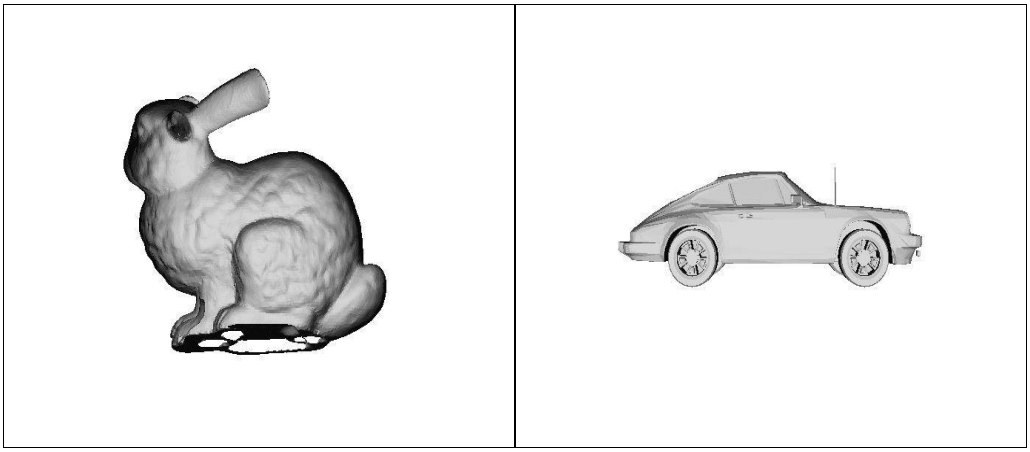
Figure 4.1 shows the models chosen to evaluate the algorithm. Some of the selected models have smooth properties while others have normal discontinuities.



(a) Cow (2903 vertices) (b) Bugman (12594 vertices)



(c) Cessna (6796 vertices) (d) Hind (3218 vertices)



(e) Bunny (34835 vertices) (f) Porsche (5247 vertices)

Figure 4.1 Models chosen to demonstrate the QEFC algorithm. They can be divided into two groups: Objects with significant normal discontinuities and objects which are largely smooth.

4.2 Time and Space Efficiency

Although this research was not only guided in terms of processor and memory requirements for the simplification method, it is important to analyze these factors. This is important to show if an algorithm presents good results, while being fast and having a small memory footprint (e.g. to be used in a game development pipeline), or if the algorithm is slow and the quality of the simplified models does not offer any comparative advantage over other simplification algorithms. First the time complexity of the QEFC algorithm will be analyzed. Then, information will be provided for the memory usage and empirical running times.

4.2.1 Time Complexity

Since the QEFC approach involves the integration of three algorithms already studied, this analysis of time and space complexity can be based on the previous information presented by the authors of the algorithms this work is based on. It also can be based on the information provided in the previous chapter.

As explained by Garland, the first algorithm (Quadric-Based Polygonal Surface Simplification [9]) has time complexity of $O(n\ logn)$, where n represents the size of the mesh.

The second algorithm used is FDSM [7] with runtime of $O(n + mlogm)$, where n is the size of input and is related to the original mesh faces, and m is the number of strong edges. For this algorithm, computing the face angles takes $O(n)$ time and it is considered the most complex task. Sorting the strong edges takes $O(mlogm)$ [7]. For most models the number of strong edges is small compared to the overall size of the model.

The Douglas-Peucker line simplification algorithm produces great variation between the best-case and worst-case scenarios. Its run-time is $O(pm)$, where p represents the number

of segments of the simplified curve and *m* is the number of segments of the original curve. Note that this *m* is directly related to the *m* in the FSDM algorithm. As explained in the last section, this algorithm sets the threshold ε to 0, meaning that the QEFC implementation will consider every segment of the curve. This produces a complexity of $O(m^2)$.

4.2.2 Memory Usage

Memory consumption is also an important factor in determining efficiency. According to Garland [37], his quadric error algorithm uses 268*v* bytes for storing the mesh and the simplification data, where *v* is the number of vertices of the model, *n* is the number of faces and *p* is the number of edges. For a closed manifold, the number of faces *n* is usually twice the number of vertices and the number of edges *p* is three times the number of vertices. Since the QEFC method uses an alternative implementation of the quadric algorithm, the representation of data is different. In this case, the vertex stores its position, the faces incident to it and the associated quadric. The faces store vertices and normals associated with it. All of the previous information is similar to Garland's implementation but the pair links information. This is used to compute the feature edges. There is also a difference regarding where the vertex stores the target vertex and the cost. This implementation also has another structure that records the contractions. It stores the *from* vertex, the *to* vertex, and also the face and vertex indices to be removed. This data structure permits to perform simplification or undo the simplification process on a step-by-step basis.

	Data Item	Stored As	Size (bytes)
Vertices	Position Face Links Quadrics + areas Target v Cost Q(v)	$3v$ floats $3f$ words $11v$ doubles v words v floats	$12v$ $12f$ $88v$ $4v$ $4v$
		Subtotal	$108v + 12f \sim= 132v$
Faces	Vertices Normal	$3f$ words $3f$ floats	$12f$ $12f$
		Subtotal	$24f \sim= 48v$
Pairs (Feature Detection)	Endpoints Dihedral Angle Edge Angle Curve Length Booleans	$2p$ words p doubles p doubles p doubles $5p$ Booleans	$8p$ $8p$ $8p$ $8p$ $5p$
		Subtotal	$37p \sim= 111v$
Edge Collapse	Pair Faces Removed Vertices Removed	$2p$ words n words v words	$8p$ $4f$ $4v$
		Subtotal	$8p + 4f + 4v \sim= 36v$
Line Simplification	Vertex Distance	v words v floats	$4v$ $4v$
		Subtotal	$4v + 4v = 8v$
		Total	$132v + 48v + 111v + 36v + 8v = 335v$

Table 4.1 The memory consumption of the QEFC algorithm is shown. Calculations assume that $f \sim= 2v$ and $p \sim= 3v$. It is assumed that the number of faces (f) usually is 2 times the number of vertices (v). In the case of the pairs, they are 3 times the number of vertices, for closed manifolds [37].

Since the QEFC algorithm deals with feature detection, the structure is treated as a segment with curvature information associated. Finally, there is the line simplification

data structure. This stores a heap with the vertices and their respective distances. The Table 4.1 shows that the memory requirements are linear in the size of the mesh.

4.2.3 Empirical Running Time

The runtime tests consist of measuring the time taken for the QEFC algorithm to perform the first pass and the second pass. The former is the time to simplify the model using the QEM algorithm and the latter is the time to select the features and simplify them using the Douglas-Peucker algorithm. Although this implementation uses the Quadric-based algorithm, it differs from Garland's work in the way it defines the processing steps. In the traditional quadric algorithm, there are two steps: the initialization and the simplification steps. In this case, the first-run involves the initialization of the matrices, the simplification using QEM and the FDSM algorithm. The second pass involves the selection of the vertices as features, the simplification of these curves using the Douglas-Peucker algorithm and the complete simplification of the mesh with the generation of iterative collapses. The running times for the first and second passes are shown in Table 4.2.

Model	Time(s) - Total	Time(s) – First Pass	Time(s) - Second Pass	# of Faces	# of Vertices	# of Feature Vertices
Cow	1.891	1.48	0.41	5804	2903	380
Bugman	14.125	11.71	2.42	25100	12594	1293
Hind	10.657	10.16	0.5	6448	3218	1171
Cessna	14.829	13.64	1.188	13546	6796	1243
Bunny	12.969	6.78	6.19	69451	34835	419
Porsche	10.35	8.13	2.22	10474	5247	1397

Table 4.2 Running times for the first pass (the initialization of the matrices, simplification using QEM) and the second pass (the selection of the vertices as features, the simplification of these curves). The times were taken on a Pentium M 1.7Ghz with 512 MB of RAM memory.

By comparing the running times in the Table 4.2, it can be concluded that, for the tested models, empirical running times depend not only on the size of the mesh, but also on the

first pass. For example, although the Bugman model has two times the number of faces of the Cessna model, they have similar execution times (~14s). This could be explained by the Quadric Error Metric algorithm and the FDSM execution times. Another example is the comparison between the cow model and the bunny. The former has almost 12 times fewer vertices than the latter though the first pass simplification time is only 6 times more for the bunny model. One more time, this could be explained by the Quadric Error Metric algorithm and the FDSM execution times.

The second pass involves selecting the feature vertices to be preserved. It also returns the mesh to its original state which happens after the user selects the percentage of feature vertices to be kept and the total number of vertices of the simplified model. The second pass time is shown in Table 4.2 which demonstrates how much time it takes to re-select the feature vertices for some models.

As expected, the second pass takes much less time than the first. It also scales linearly with the mesh complexity for the tested models. This shows that the selection of feature vertices after they have been detected in the first pass does not add much of a computational burden, since the computation of the priority heaps is done only once if the feature parameters are not changed.

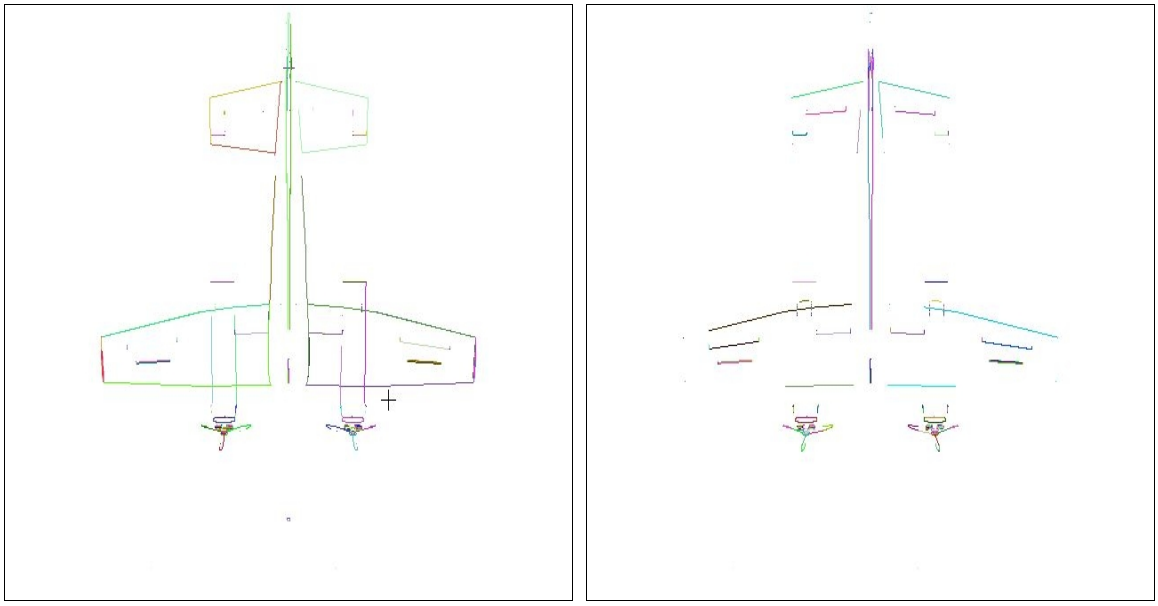
4.3 Parameters Used

Xianming and Heath [7] state that defining appropriate parameters for feature identification is not a trivial task. They provide some guidelines to find the upper and lower bound angles, although the signal/noise ratios indicted by *r* were harder to determine because they are more mesh dependent. Table 4.3 shows the parameters for the FDSM algorithm used in the this thesis.

	Dihedral Angle	Edge Angle	Angle Defect
theta-u	40	80	80
theta-l	15	15	20
r (signal/noise ratio)	2	4	3

Table 4.3 Parameters used to identify features of models. They were shown according to the suggestions of the feature detection paper [7].

In the case of the filtration rules, the number of weaknesses *b* in angle defect was defined as greater than 0 to enable the short-falseness rule and 2 for the long-falseness rule. The parameter *l* that defines if a curve has less than *bl* edges, to be considered false strong was set to 3. This value was chosen after testing with different parameters and models. The feature detection for all models was done with the short and long false-strong rules enabled, unless otherwise stated. The use of sub-optimal values for the parameters can lead to poor representation of the feature curves as it is shown in Figure 4.2. As was stated before, the threshold parameter ε was set to 0. This was intended to force the implementation to compute the Douglas-Peucker distance of all the vertices incident to a curve.



a) Cessna well-chosen parameters b) Model with poorly chosen parameters

Figure 4.2 Feature curves of the Cessna model.

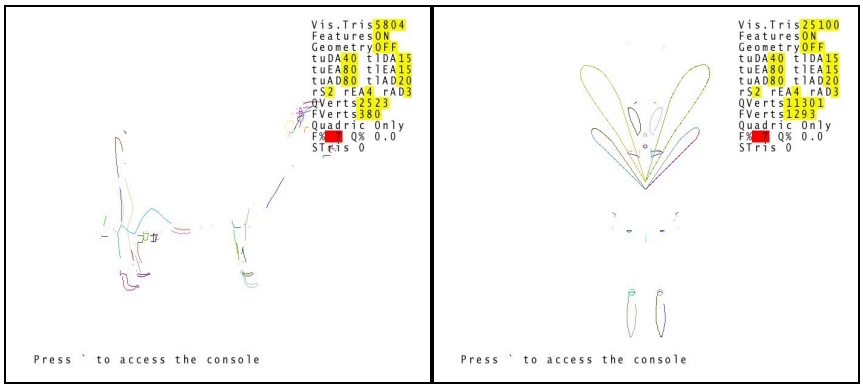
4.4 Feature Curves

As explained in Section 3, the feature curves are a result of the identification of edges that fulfill a predetermined set of rules. These rules involve calculating if the dihedral angle, edge angle and edge length to check if they satisfy certain parameters. The edge is selected as part of the curve if it is the one from a set of neighboring edges that best fulfills these rules. The curves can be closed or open.

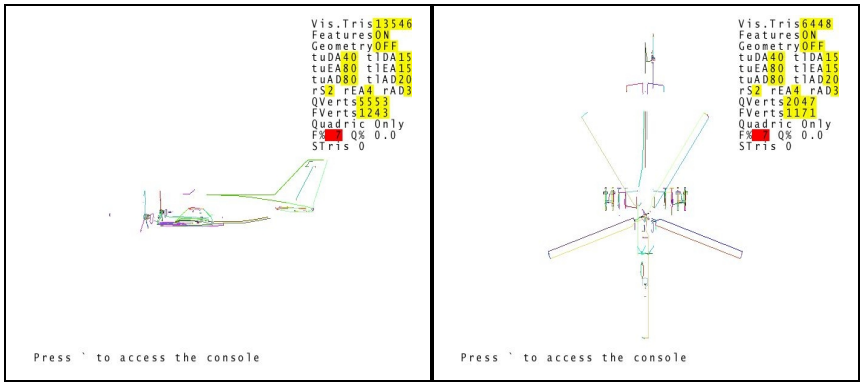
The features are displayed for the chosen models in Figure 4.3. In order to help distinguish them, each curve has a different color associated with it.

Even though the feature curves are a good indication of the shape of the model, this is not a guarantee that they are sufficient to maintain the volume of parts of the model. This will be better explained when the Porsche model is analyzed.

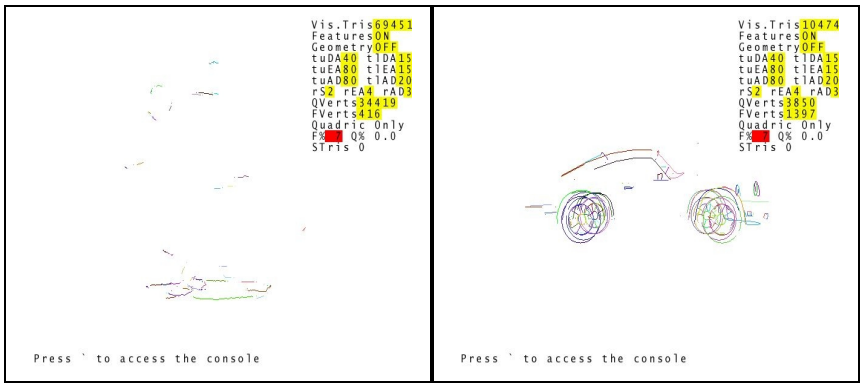
As can be observed, the feature curves appear as silhouettes in models that have normal discontinuities. The features of the Bunny model, for example, represent poorly the model, while, in the case of the Hind, they provide a good approximation to its shape.



(a) Features on Cow model (b) Features on Bugman model



(c) Features on Cessna model (d) Features on Hind model



(e) Features on Bunny model (f) Features on Porsche model

Figure 4.3 Feature curves are shown for the chosen models. It can be noted that objects with normal discontinuities provide better features. In fact, the features are so good in delineating the shape of the model that it is almost possible to see their silhouettes only by looking at the feature curves. On the other hand, models that lack discontinuities like the Bunny model are almost impossible to distinguish.

4.5 Geometric Quality and Approximation Error

Apart from developing a simplification method, one of the tasks in Mesh Simplification that presents a challenge is to compare models simplified with different approaches. Although many authors consider the Hausdorff distance [4] an important way of determining the quality of a mesh, there are some discrepancies between its numeric results and the visual fidelity of a model. In some cases, one could argue that a model can appear better than the other, even though the Hausdorff distance would tell another story.

One of the most used tools to compare two models is Metro [4]. It works by using a point to surface distance metric and a parameter P_v that determines the distance between the sampling points. The first model is sampled by Metro using points spaced according to the parameter value. Then, each point is mapped to the closest point on the second model. The tool determines the distance between each pair of mapped points. It returns the average distance and the maximum distance (Hausdorff distance).

Given two meshes, *OM* and *SM*, the former being the original mesh, and the latter being the simplified mesh, P_o is the set of points in the original mesh, P_s the set of points on the simplified mesh, the Hausdorff distance is computed using Equation 4.1.

$$Emax\,(OM, SM) = max\,(\,max_{a \in Po}(min_{b \in Ps}(d(a, b))),\, max_{b \in Ps}(min_{a \in Po}(d(a, b)))\,) \quad (4.1)$$

where, for a point y, $min_{a \in Po}(y)$ is the minimum Euclidean distance between all points in the set Po and x. $max_{a \in Po}(y)$ is the maximum Euclidean distance between all points in the set X and y.

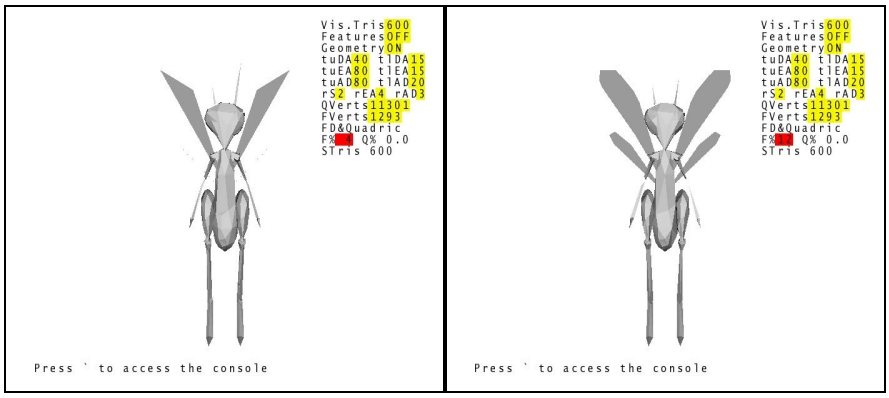
4.5.1 – Appropriate Number of Preserved Feature Vertices

In the second pass, the algorithm has already computed all contractions using the quadric error metric and identified all the features of the model. The edge contractions are stored in a data structure, allowing the algorithm to interactively display the simplification/refinement processes. Then the user has to select the percentage of feature vertices to be preserved.

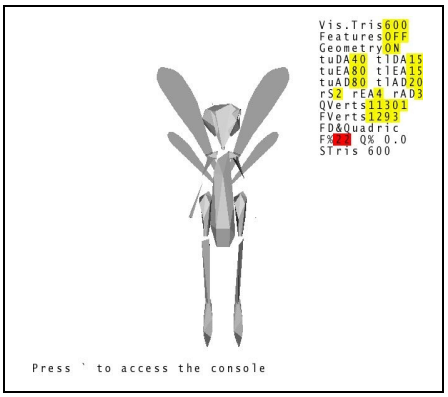
After many tests with different models, it was determined that setting values for the feature parameter that work for different models is difficult. Many factors are taken into account when selecting an appropriate parameter: the number of faces of the model, the number of feature vertices and even the shape of the detected features.

Since the definition of appropriate values for the feature selection parameter is hard to determine, it is important to show what an upper-bound would be. The maximum desired value of the feature selection parameter would be defined by the desired degree of simplification. Although in real-world tests, if the maximum desired value for the feature selection parameter is close to the total number of vertices, the results are not encouraging. This is explained by the fact that, as the feature selection parameter reaches the maximum value, the vertices chosen by the quadric error algorithm, which are crucial to represent the shape of the model, are discarded.

On the other hand, a very small value for the feature selection parameter can be of little effect, since the feature and quadric heaps might have the same vertices stored in them with similar priorities.



(a) Bugman model with 4% (a) and 12% (b) of the 1293 feature vertices preserved



(c) Bugman model with 22% of the 1293 feature vertices preserved

Figure 4.4 Bugman model simplified at 2% of the original triangles with different levels of preservation of features in relation to the number of faces.

As can be seen in Figure 4.4, the Bugman model shows noticeable differences depending on the value chosen for the feature selection parameter. A small value does not change much of the mesh, whereas a very large value will preserve most of the feature vertices (wings) detected, but at the cost of other important parts of the model like the arms and the eyes.

In order to assess the optimal parameter value for the feature preservation, it was necessary to use the tool to interact with the model through the selection of different number of feature vertices preserved until the optimal one is found.

4.5.2 - Visual Results

Assessing the fidelity of simplified models through visual comparisons is one of the most used and practical ways of showing that a method can give better results than another. The objective is to show the visual differences for given models and resolutions, and later provide the numerical results through the use of the Metro tool [4].

The first mesh presented is the Cessna model. The feature curves of this model were so well-defined that they could work as a silhouette for the original model. This means that there is a high probability that the feature preservation will postpone the removal of important parts of the model.

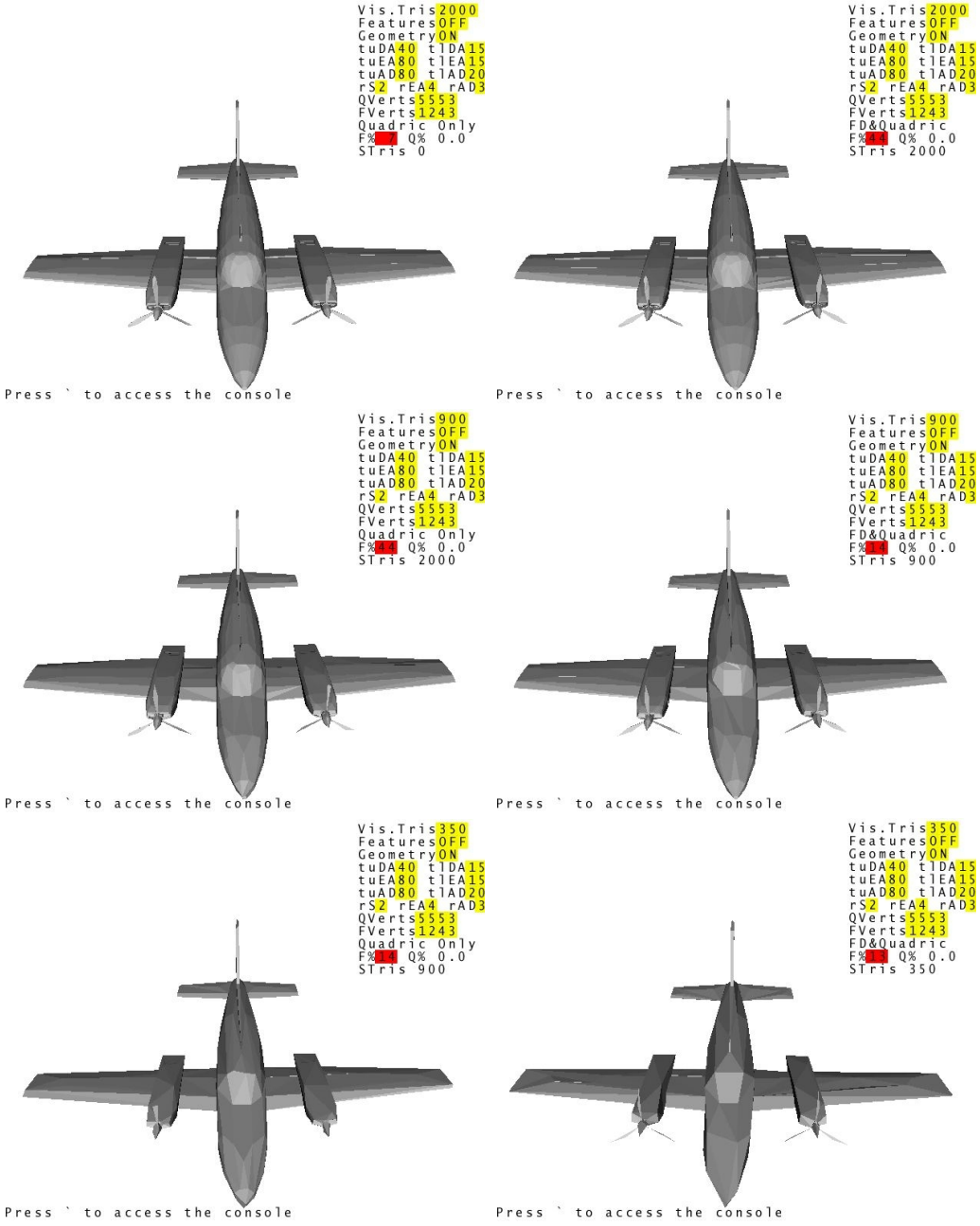


Figure 4.5 Cessna model at different resolutions (2000, 900, 350 faces). The images on the left were simplified using Quadrics only. The QEFC approach was used to obtain the simplifications of the ones on the right. In this case, 44%, 14% and 13% of the features were preserved, respectively.

As expected, the Cessna (shown in Figure 4.5) model shows a considerable improvement using the QEFC approach when compared to the traditional Quadric-based simplification

algorithm. This can probably be explained by the normal discontinuities of the object. Thin and crease features like the propellers are "protected" by the algorithm, meaning that they are preserved even in low resolution models (350-900 faces).

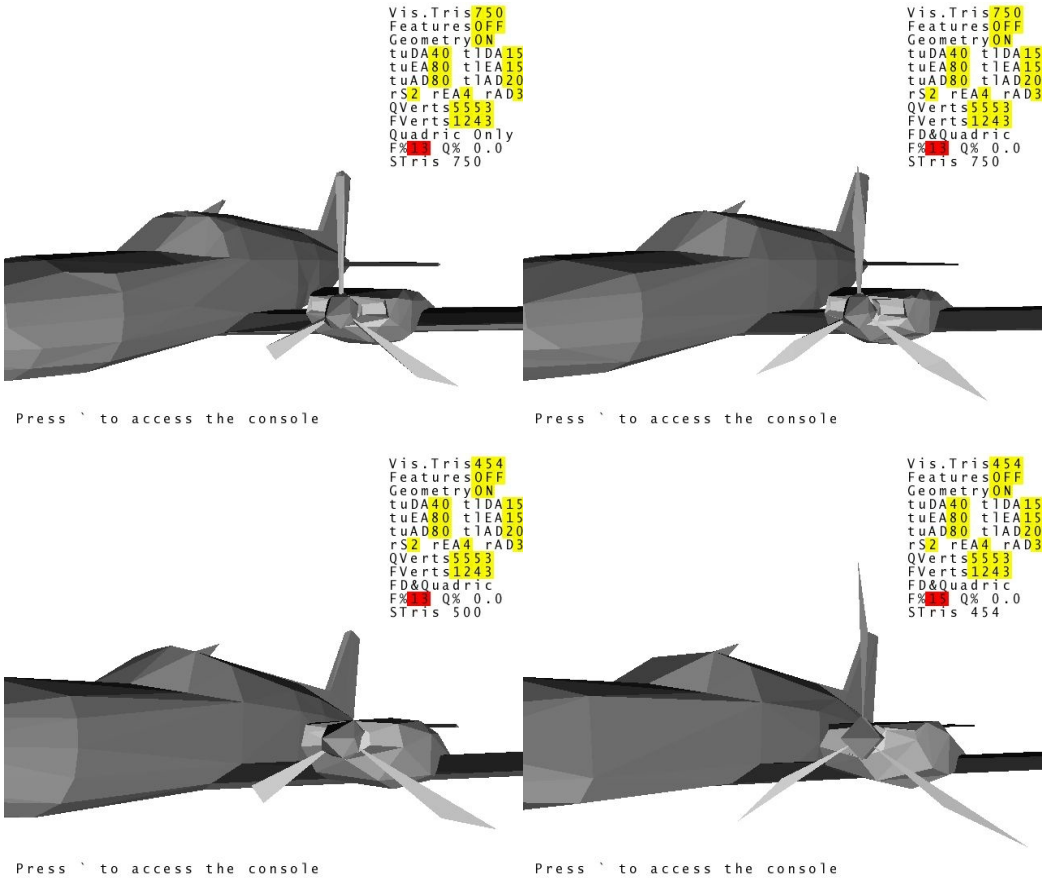


Figure 4.6 Close-up of the propellers of the Cessna model. Note that the propellers on the subfigures at the right are look better preserved than the ones on the left (QEM-simplified models) because their features were “frozen”. The models were simplified to 750 and 454 (top, bottom) triangles with the subfigures on the right having 13% and 14% of the vertices preserved, respectively.

A situation like the one on Figure 4.6 shows that the quadric error simplification could overlook an important part of the model. In this case, the FDSM algorithm identified the propeller blades of the Cessna model, so that their removal even at a low model resolution was avoided. The model that demonstrates the most differences in silhouettes is the Hind model. As can be seen in Figure 4.7, the blades and wingtips on the hind

simplified using Quadrics disappear at an earlier simplification stage when compared to the model reduced using the QEFC method. The features are useful to indicate the preservation of the large and small blades.

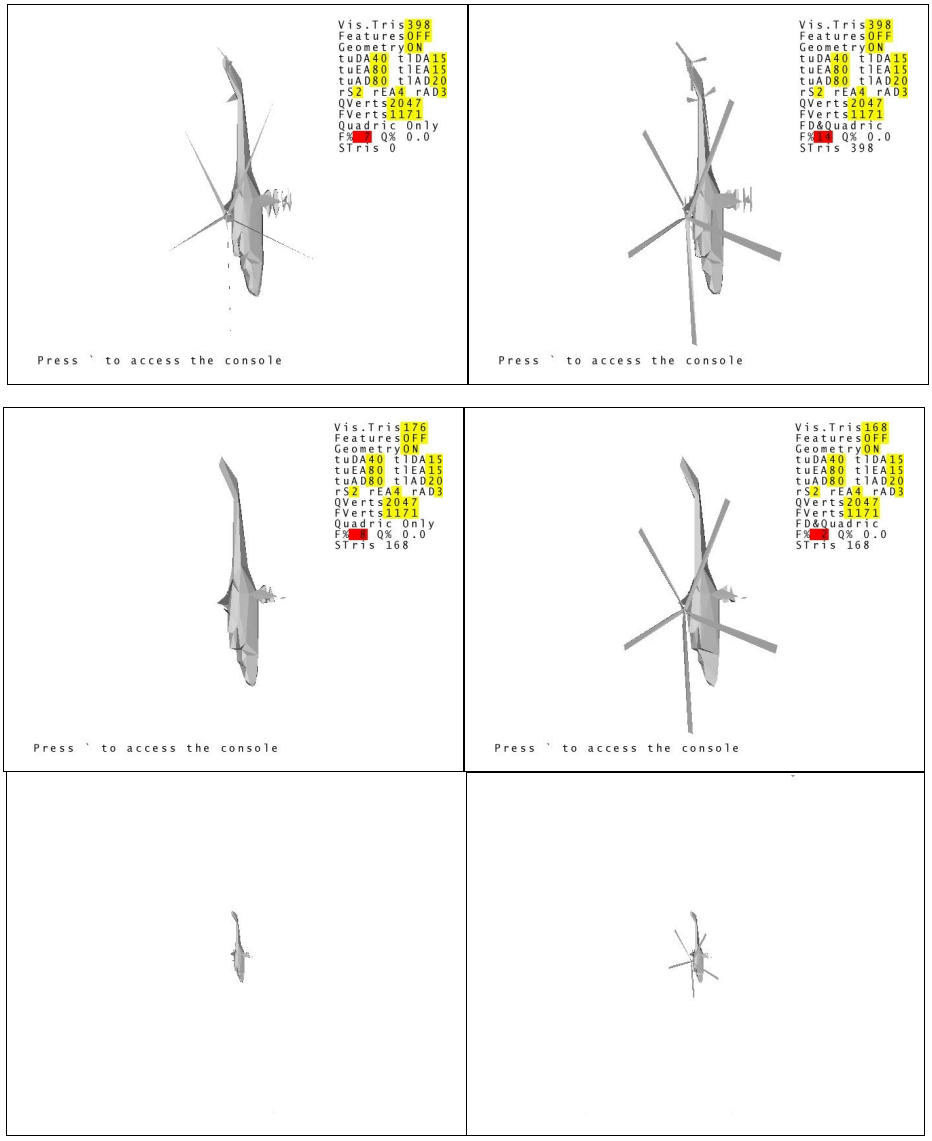


Figure 4.7 Hind model with different levels of simplification using Quadric Error (left) and the QEFC approach (right). Note that the blades are better preserved on the right. The models have 398 (top) and 168 faces with the subfigures on the right having 14% and 2% of the feature vertices preserved, respectively.

However, the Hind model simplified with the quadric error algorithm seems to offer better quality for small details like the cabin. One explanation is that some vertices from the quadric heap were discarded in favor of vertices in the feature heap. Also, since this approach uses the Douglas-Peucker method to simplify curves, it is expected that the longest features will be on the top of the feature heap.

The model simplified with the QEFC algorithm appears to provide better results visually, since this type of low resolution model would probably only be used when the object is far away from the viewer, meaning that small details do not contribute too much to the overall quality. In this case, it makes sense to think that the long and large parts of a model are the most important. It also gives better results in terms of the Hausdorff distance. This will be shown in section 4.4.3.

The next model to be considered is the Porsche, shown in Figure 4.8. The features identified for this mesh can provide a better indication of its shape, even though the model did not gain much quality by using the QEFC method. This could indicate that the Quadric Error algorithm is already preserving the identified features or that the QEFC method is not able to keep the volume of certain features like the wheels.

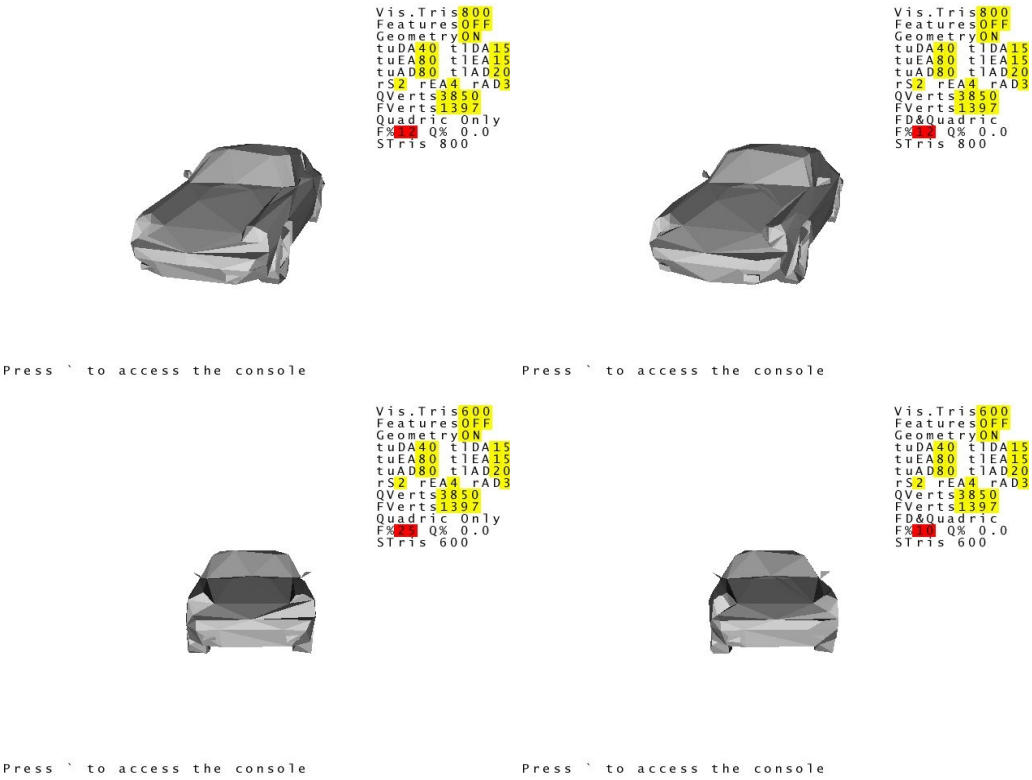


Figure 4.8 Porsche model with different levels of Simplification. The simplification using Quadric Error (left) preserves most of the features identified by the QEFC approach (right).

It must be pointed out that there are some minor differences in the pictures. First, it is possible to identify that the side mirror and the fog lights are better preserved with the QEFC algorithm, for the meshes with 800 faces. The meshes with 600 faces show differences on the hood geometry. The mesh simplified using quadric error metric shows less of the headlight and the hood.

In Chapter 3, the implementation of this method was described as an interactive simplification tool. The user is able to choose parameters related to the feature identification and preservation according to the simplification level. In the case of feature identification, choosing different parameters than the standard ones (indicated in section 4.3) could be used to increase the number of features detected. This could be useful when simplifying models without or with few normal discontinuities like the Bunny. As

explained before, in this case, the short and long falseness rules were disabled in order to end up with a silhouette that resembled the original model as can be seen in Figure 4.9, although this does not noticeably improve the quality of the simplified mesh.

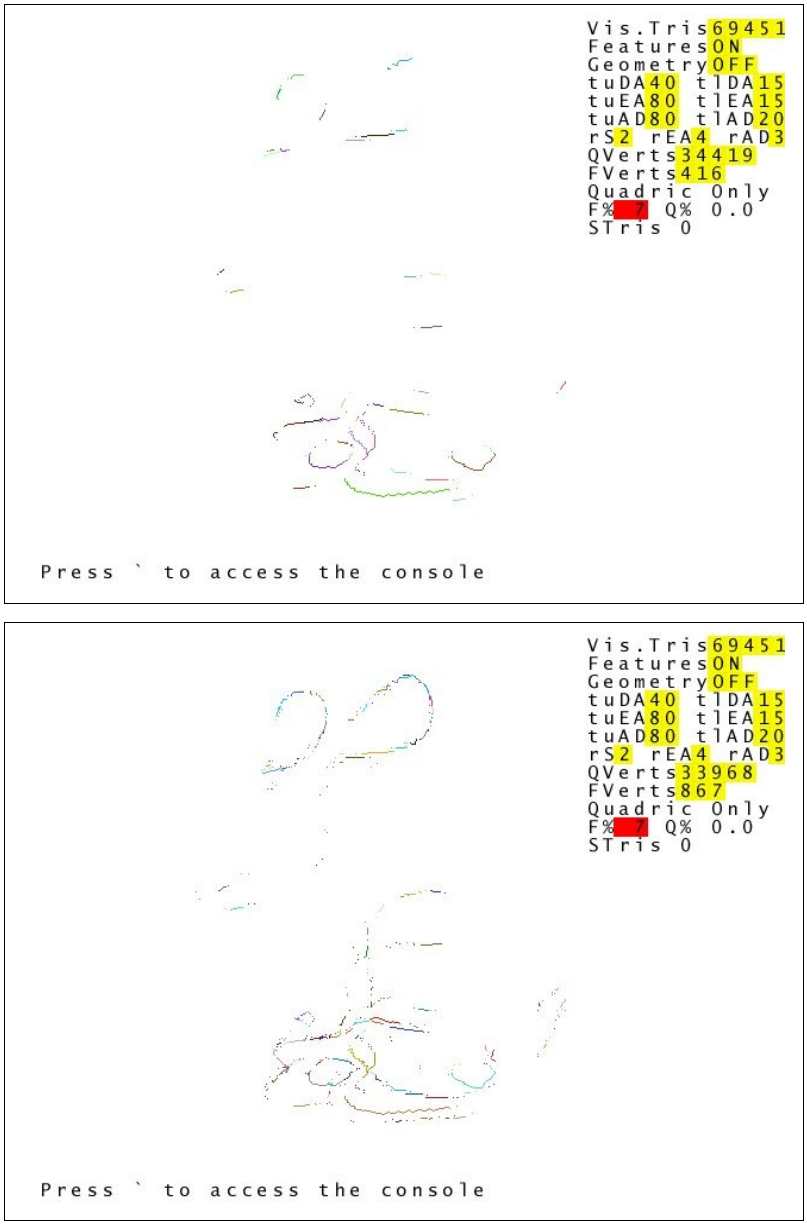


Figure 4.9 Bunny model feature curves with short and long falseness rules on (top) and off (bottom) using default parameters (shown in Table 4.3). Note that the second subfigure resembles better the silhouette of a bunny.

4.5.3 - Metro Scores

Metro [4] is a tool used to compare simplified models to their original state. However, because Metro provides different methods of computing results, some authors [38] prefer to compare models using Metro's Hausdorff (maximum) distance while others [6] opt for Metro's mean distance or the root mean squared distance. In some cases, they utilize both scores. The Hausdorff distance is explained as the maximum of the minimum distances between points in the original mesh and the ones in the simplified mesh. The mean distance and root mean squared distance represent the average distance between the points of two compared meshes. This could mean that while the former metric is useful for detecting the loss of a large part of the original mesh after it was simplified, the latter refers to the overall quality of the simplified model.

Excluding the Bunny model, all other models using the QEFC approach provided better approximation quality when compared to the QEM algorithm. For this reason, only the results that have generated relevant conclusions to this work will be shown.

Although Metro is an important reference to mesh quality, there are cases where two models have the same Hausdorff distance although their visual fidelity is different. Some results that correspond to this case are shown with a comparison between the quadric error metric and the QEFC approach.

This problem is exemplified by the Bugman model. The graph in Figure 4.10 shows that at a mesh resolution of 600 faces, the model simplified using the quadric error algorithm has the same Hausdorff distance as the model simplified using the QEFC method (shown in Figure 4.10). As seen in Figure 4.11, it becomes clear that the models look different.

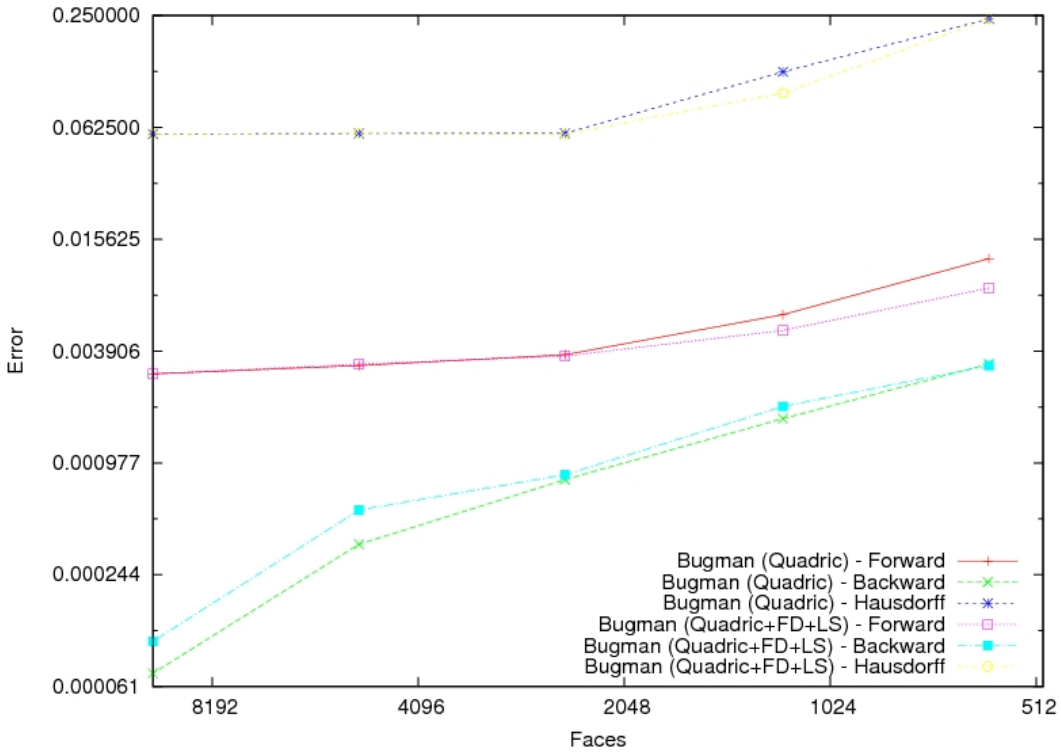


Figure 4.10 Hausdorff distance of the Quadric Error and QEFC for the Bugman model.

Looking at the pictures, it appears that the model that was simplified using the QEFC approach better preserves its original appearance. The wings and shoulders serve as proof of this. On the other hand, the antennas look better preserved in the model simplified with the Quadric metric.

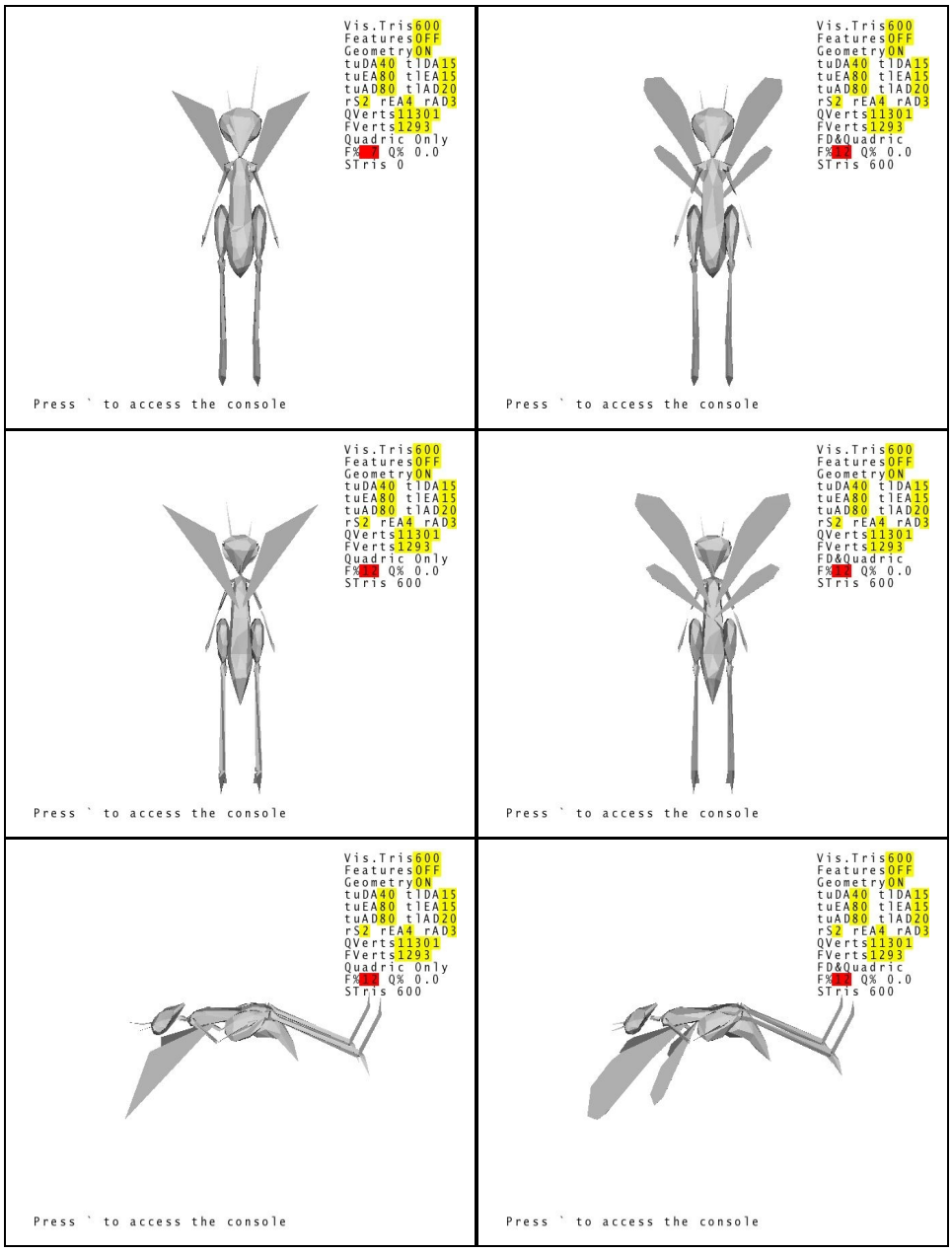


Figure 4.11 Discrepancies between Metro scores and the visual fidelity of the Bugman model. Even though the two simplification approaches result in similar Metro scores (Figure 4.10), visually the model simplified with QEFC looks superior.

One way to interpret Metro scores is to imagine that there is some important part of the original mesh that has been greatly simplified in both models. This was reflected in the

maximum distance. In this case, it makes more sense to analyze the mean score, since it takes into consideration the whole model instead of the maximum distance between the points of two models.

Bugman	Quadric Metric	Quadric+FD+LS
Forward (Mean Distance)	0.012297	0.008529
Backward (Mean Distance)	0.003334	0.003267
Hausdorff Distance	0.240072	0.240072

Table 4.4: Forward and backward mean distances for two Bugman models simplified to 600 faces with the same Hausdorff distance. One was simplified using the QEFC method and the other used the quadric error algorithm. FD means Feature Detection and LS means Line Simplification.

Table 4.4 shows that while there is some important part of the original model missing in both simplified models, but on average, the mesh simplified using the QEFC method has better quality than the one simplified using the quadric error algorithm. These results coincide with the visual analysis of the models.

The scores for the Cessna model are shown in Figure 4.12. Since this mesh has similar features as the model used as a reference in the FDSM algorithm [7] which this method is based on, it was expected that a good silhouette would be obtained. The question is if the detected features would translate into a simplified model with good quality.

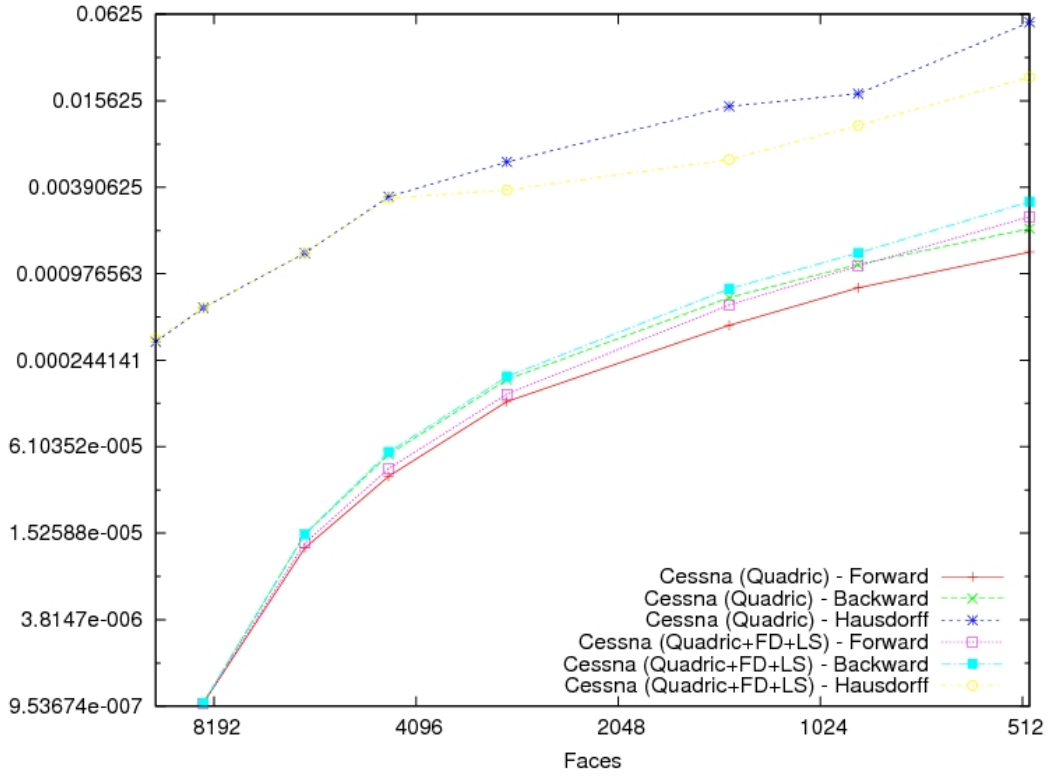


Figure 4.12 Metro scores for the Cessna model originally endowed with 13546 faces. This model can be included in the category of models with normal discontinuities, being the perfect candidate for the QEFC method.

The above results show an interesting situation. The quadric-based simplification method and the QEFC approach seem to make similar choices on resolutions of 10000, 8500, 6000 and 4000 faces. From that point on, the QEFC method provides results that make the Hausdorff curve on the graph less steep.

The Hind model has similar characteristics to the Cessna model as shown in Figure 4.13. The rotor blades have thin and long features as with the Cessna's wings and propeller blades. The well-defined silhouette detected by the feature identification algorithm indicates that this is a type of model that the QEFC algorithm can be used to improve the quality.

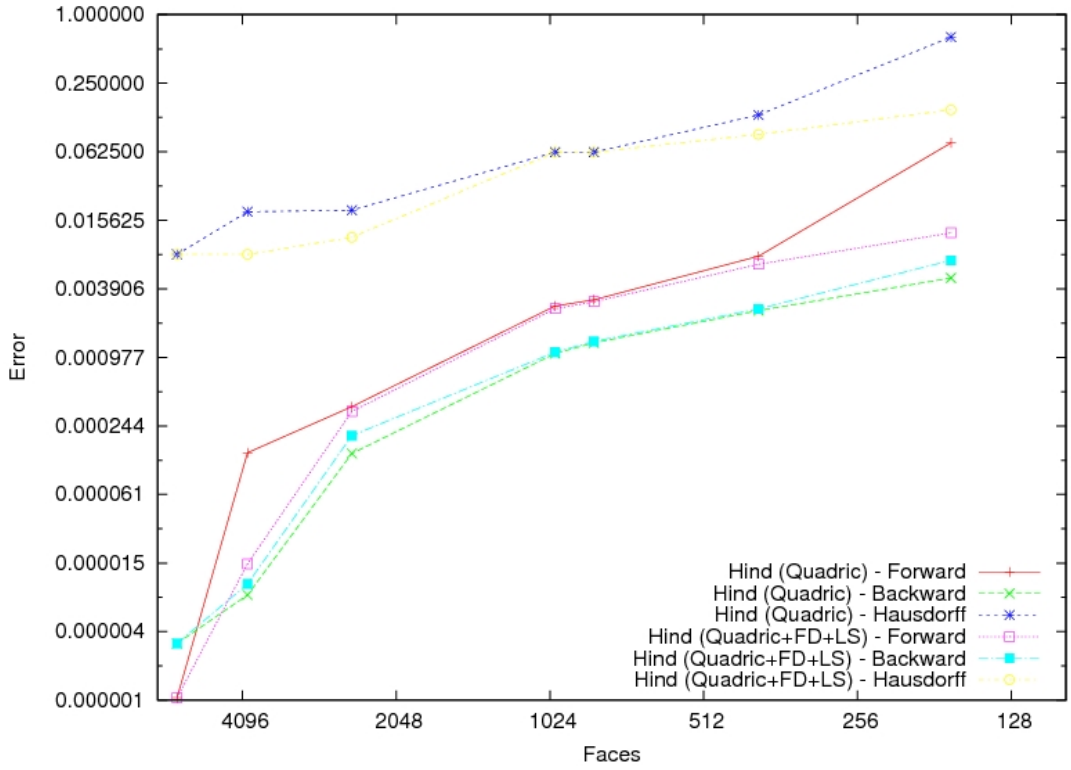


Figure 4.13 Metro scores for the Hind model originally with 5804 faces.

The scores show that the QEFC algorithm can provide better results than the quadric error algorithm alone. This is especially true when the simplified model reaches fewer than 700 faces. From that point on, the Hind simplified using the quadric algorithm starts losing the rotor blade details. At a low resolution, the QEFC algorithm shows the the best results, because it allows crucial feature vertices to be preserved in the simplified model.

However, at a resolution of 840 faces, the two methods show the same Hausdorff distance. The mean average error of the two models was analyzed to check which one would have the lowest score. Though, we note that visually the models showed no noticeable difference. This could demonstrate that the quadric-based simplification made similar choices as the QEFC method.

Mean Distance (Hind)	Forward	Backward	Hausdorff
Quadric Metric	0.003130	0.001314	0.061912
Quadric+FD+LS	0.003007	0.001348	0.061912

Table 4.5: Forward, backward and Hausdorff distances for two Hind models simplified to 840 faces. One was simplified using the QEFC method and the other used the Quadric metric.

The forward and backward mean distances in Table 4.5 show that the QEFC algorithm provides mixed results. Although, it can be inferred that on average the mean distance in with the QEFC simplified model is lower, visually both solutions seem to provide similar results.

It could be assumed from these results that the QEFC method chooses to keep important vertices that were already discarded by the quadric error algorithm. This occurs especially when a model is simplified to low resolutions. But it may be that if the model continues to be simplified, the quadric error algorithm and the QEFC method coincide in the preservation of vertices. This means that in some cases this method will not provide better results.

Given that the user chooses appropriate values for the feature selection parameter, the QEFC algorithm provides most of the time the same or better results as the quadric error algorithm. After analyzing the results, it is concluded that the QEFC method is more likely to provide superior results when the mesh has normal discontinuities and when the simplified mesh is at low resolutions.

CHAPTER 5: CONCLUSION

We have developed a novel algorithm for Mesh Simplification that combines the Quadric Error Metric [9] and FDSM [7] to provide good simplification results. The feature curves are simplified with the use of the Douglas-Peucker algorithm and the resulting points are "frozen" so they can be preserved during the simplification using the QEM algorithm. The QEFC algorithm has close to sub-quadratic complexity, given that it works as a combination of algorithms with sub-quadratic time.

Methods [5][6][39] that use feature detection or the integration of feature detection with QEM provide only local consideration of feature importance or do not provide a structured way of simplifying feature curves as the overall mesh is simplified. QEFC provides global feature identification. It also simplifies features through the use of the Douglas-Peucker algorithm which avoids inadvertently increasing too much the importance of a feature curve as the mesh is simplified. Methods that integrate feature and curvature detection with the QEM algorithm do the integration process by summing quadric error to the curvature values [5] or by applying a weight to the quadric error cost. [39] This could make the relative importance of features to be overestimated or underestimated depending on the simplification level. QEFC provides the integration by requiring that a parameter has to be set interactively by the use of our system to define how many feature vertices are to be preserved.

It has been demonstrated that the QEFC algorithm can provide better approximations of an original model when compared to the Quadric Error Metric implementation on certain models that have specific types of features. It gives better results on models that have sharp edges and continuous curves. This can be explained by the fact that the QEM algorithm accumulates error as the simplification progresses and it prematurely discards vertices that are important to the model's general structure.

It is important to state that the QEFC system should have its feature parameter defined depending on the type of model. In Chapter 4, some optimal parameter values were suggested for the models used. Nonetheless, when using a completely new model, this algorithm requires some experimentation before selecting the optimal number of feature vertices to preserve.

5.1 - Contributions

Identification of Simplification Methods – In order to reach the conclusions present in this work, it was necessary to study different simplification techniques and to identify their weaknesses and strengths. The QEM having been selected, it was coupled with a the FDSM algorithm. Based on that information, the research extended to areas beyond mesh simplification to identify useful algorithms to help in the simplification of the resultant feature curves.

Integration of Identified Techniques – A system was developed that identifies features and simplifies them by changing the priority of the simplification in the traditional QEM algorithm. This integration resulted in an algorithm that can be used to refine the simplification.

5.2 - Future Improvements

Integration of Color and Normals

Garland et al [31] suggested that color and normals could be related to Quadrics when they proposed an extended version of the QEM algorithm. The QEFC algorithm could be extended in the same direction using Garland's extended QEM algorithm or by providing a method to modify the FDSM algorithm to create curves on vertices that have color and normal discontinuities.

Comparison with Additional Algorithms

The Quadric with Feature Curves algorithm was compared directly to the QEM method. The comparisons could be extended to other algorithms that make use of features as well.

Image-based comparisons

It was shown in Chapter 4 that in some cases the Metro scores would incorrectly portray the visual differences between models simplified with QEM and QEFC. For this reason, image-based comparisons could help assessing the quality of the approximations.

REFERENCES

[1] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, R. Huebner, Level of Detail for 3D Graphics, Morgan Kaufmann, pp.6-14, 2003.

[2] W. Carlson, "A Critical History of Computer Graphics and Animation", <http://accad.osu.edu/~waynec/history/lessons.html>, accessed on 12/09.

[3] – D. Sevo, "History of Computer Graphics", <http://hem.passagen.se/des/hocg/hocg_1960.htm>, accessed on 12/09.

[4] P Cignoni, C Montani, R Scopigno, A Comparison of Mesh Simplification Algorithms, Computers & Graphics, Pisa, vol. 22 No.1, pp.37–54, 1998.

[5] Y. Wu, Y. He, H. Cai, QEM-based Mesh Simplification with Global Geometry Features Preserved, Computer Graphics, SIGGRAPH'04 Proceedings, ACM, pp.50-57, 2004.

[6] S. Kim, S. Kim, C. Kim, Discrete Differential Error Metric for Surface Simplification, Proceedings of 10th Pacific Conference on Computer Graphics, IEEE Computer Society, pp276-283 , 2002.

[7] J. Xiangmin, M. T. Heath, Feature Detection for Surface Meshes, Proceedings of 8th International Conference on Numerical Grid Generation in Computational Field Simulations, University of Illinois, pp.705-714, 2002.

[8] D. Douglas, T. Peucker, Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature, Canadian Cartographer, University of Toronto Press, vol. 10 No. 2, pp.112–122 , 1973.

[9] M. Garland, P. S. Heckbert, Surface Simplification Using Quadric Error Metrics, Proceedings of the 24th annual Conference on Computer Graphics and Interactive Techniques, ACM Press, pp.209-216, 1997.

[10] C. Machover, The CAD/CAM Handbook, McGraw-Hill, pp.69, 1996.

[11] D. LaCourse, Handbook of Solid Modelling. McGraw Hill. pp.25, 1995.

[12] A. H. Watt, 3D Computer Graphics, 2nd Edition, Addison-Wesley, pp.27-65, 123-141, 1993.

[13] J. J. Clark, Hierarchical Geometric Models for Visible Surface Algorithms, Communications of the ACM, ACM, vol. 19(10), pp.547-554, 1976.

[14] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Mesh optimization, ACM SIGGRAPH, ACM, pp.19-26 , 1993.

[15] P. Linstrom, G. Turk, "Image-Driven Simplification", ACM Transactions on Graphics, ACM, vol.19(3), pp.204-241, 2000.

[16] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, R. Huebner, Level of Detail for 3D Graphics, Morgan Kaufmann, pp.47-83, 2003.

[17] J. Rossignac, P. Borrel, Multi-Resolution 3D Approximations for Rendering Complex Scenes, Technical Report RC 17687-77951, IBM Research Division, T J Watson Research Centre, YorkTown Heights, NY 1992.

[18] D. Luebke, C, Erikson, View-Dependent Simplification of Arbitrary Polygonal Environments, Proceedings of SIGGRAPH 97, ACM Press/Addison Wesley Publishing Co., pp.199-208, 1997.

[19] K. Low, T. Tan, Model Simplification Using Vertex-Clustering, Proceedings of 1997 Symposium on Interactive 3D Graphics, ACM, pp.75-81, 188, 1997.

[20] R. Ronfard, J. Rossignac, Full-Range Approximation of Triangulated Polyhedra, Computer Graphics Forum, Blackwell, vol.15(3), pp.67-76, 462, 1996.

[21] H. Hoppe, Progressive Meshes, Proceedings of the 23rd annual conference on Computer Graphics and Interactive Techniques, SIGGRAPH, ACM, pp.99-108, 1996.

[22] A. Varshney, Hierarchical Geometric Approximations, Technical Report: TR94-050, University of North Carolina, Chapel Hill, NC, 1994.

[23] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, W. Wright, Simplification Envelopes, Proceedings of SIGGRAPH 96, ACM, pp.119-128, 1996.

[24] C. Bajaj, D. Schikore, Error-Bounded Reduction of Triangle Meshes with Multivariate Data, SPIE, vol. 2656, pp.34-45, 1996.

[25] J. Cohen, D. Manocha, M. Olano, Simplifying Polygonal Models Using Successive Mappings, Proceedings of IEEE Visualization 97, IEEE Computer Society Press, pp.395-402, 1997.

[26] G. Turk, Re-tiling Polygonal Surfaces, Proceedings of SIGGRAPH, ACM, vol.26 No.2, pp.55-64, 1992.

[27] B. Hamann, A Data Reduction Scheme for Triangulated Surfaces, Computer Aided Geometric Design, Elsevier Science, vol. 11 No. 2, pp.197–214, 1994.

[28] T. Surazhsky, E. Magid, O. Soldea, G. Elber, E. Rivlin, A Comparison of Gaussian and Mean Curvatures Estimation Methods on Triangular Meshes, 2003 IEEE International Conference on Robotics and Automation, Taipei, Taiwan, pp.1021-1026, 2003.

[29] D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, R. Huebner, Level of Detail for 3D Graphics, Morgan Kaufmann, pp.134, 2003.

[30] C. Bing-Yu, N. Tomoyuki, An Efficient Mesh Simplification Method with Feature Detection for Unstructured Meshes and Web Graphics, Proceedings of IEEE Computer Graphics International, The University of Tokyo, pp.34-41, 2003.

[31] M. Garland, P. S. Heckbert, Simplifying Surfaces with Color and Texture using Quadric Error Metrics, Proceedings of the conference on Visualization 98, IEEE Computer Society Press, pp.263-269, Carnegie Mellon University, 1998.

[32] L. Zhong, M. Lizhuang, Z. Zuoyong, T. Wuzheng, Feature Extraction From the Mesh Model with Some Noise, Smart Graphics, Springer-Verlag, pp.194-199, 2007.

[33] M. Milroy, C. Bradley, G. Vickers, "Segmentation of a Wrap-Around Model Using an Active Contour", Computer-Aided Design, Elsevier, vol.29 No. 4, pp.299–320, 1997.

[34] J. Goldfeather, V. Interrante, A Novel Cubic-order Algorithm for Approximating Principal Direction Vectors, ACM Trans Graphics, ACM, vol.23 No. 1, pp.45–63, 2004.

[35] C. Lee, A. Varshney, D. Jacobs, Mesh saliency, ACM Transactions on Graphics, Proceedings of SIGGRAPH 05, ACM, vol. 24 No. 3, pp. 659-666, 2005.

[36] J. Hershberger, J. Snoeyink, Speeding up the Douglas-Peucker Line Simplification Algorithm, In Proceedings 5th Intl. Symp. Spatial Data Handling, University of British Columbia, pp. 134-143, 1992.

[37] M. Garland, Quadric-Based Polygonal Surface Simplification, Thesis, Carnegie Mellon University, 1999.

[38] M. Hussain, Y. Okada, K. Niijima, LOD Modelling of Polygonal Models Based on Multiple Choice Optimization, Multimedia Modelling Conference, Kyushu University, pp.203-210, 2004.

[39] B. Chen, T. Nishita, An Efficient Mesh Simplification Method with Feature Detection for Unstructured Meshes and Web Graphics, Proceedings of IEEE Computer Graphics International (CGI03) l, University of Tokyo, pp. 34-43, 2003.