

Simplifying Spline Models ^{*}

M. Gopi

D. Manocha

{gopi,dm}@cs.unc.edu

Department of Computer Science

University of North Carolina at Chapel Hill

Chapel Hill, NC 27516

August 22, 1999

Abstract

We present a new approach for simplifying models composed of rational spline patches. Given an input model, the algorithm computes a new approximation of the model in terms of cubic triangular Bézier patches. It performs a series of geometric operations, consisting of patch merging and swapping diagonals, and makes use of patch connectivity information to generate C-LODs (curved levels-of-detail). Each C-LOD is represented using cubic triangular Bézier patches. The C-LOD's provide a compact representation for storing the model. We also present techniques to quantify the error introduced by our algorithm. Given the C-LODs, the tessellation algorithms can generate the spline model's polygonal approximations using static and dynamic tessellation schemes. The simplification algorithm has been implemented and we highlight its performance on a number of models.

^{*}Supported in part by a Sloan fellowship, ARO Contract P-34982-MA, NSF CAREER award CCR-9625217, ONR Young Investigator Award, and Intel.

1 Introduction

In the last few years, the problem of simplifying geometric models has received considerable attention in computational geometry, computer graphics and geometric modeling. Most of the literature has focussed on the simplification of polygonal models. A number of algorithms based on vertex removal, edge collapse, face collapse and vertex clustering have been proposed. Given a polygonal model, these algorithms produce levels-of-detail (LOD) of the original model. Different algorithms vary based on the error metrics used for surface approximation, the underlying representations used for the simplified model, or whether or not they preserve the topology of the original model.

In many applications, models are defined using rational parametric spline surfaces. These include Bézier patches and non-uniform rational B-spline surfaces (NURBS). Most current systems use tensor-product surfaces, though triangular surfaces have been gaining importance as well. Models composed of tens of thousands (or even more) of such surfaces are commonly used in CAD/CAM, surface fitting and scientific visualization applications. Because current graphics systems are optimized to render triangles, a number of methods based on static and dynamic tessellation have been proposed in the literature to generate polygonal approximations of spline models. However, these algorithms generate at least two triangles for each tensor-product patch and one triangle for each triangular patch. As a result, in such cases the lowest level-of-detail consists of tens of thousands of triangles. One possibility is to generate a polygonal approximation of the spline model and generate its LODs using polygon simplification algorithms. In practice, this approach has two *drawbacks*. First, it leads to data proliferation. Representing a polygonal approximation of the spline model and its LOD takes considerably more space as compared to the original spline model. This becomes a major issue in the representation of very large CAD environments (e.g. for an automobile or an entire submarine composed of hundreds of thousands of spline patches). The polygonal approximation of the entire model may not fit into the main memory. Transmitting such models over the network is also

time consuming. The second drawback relates to using static tessellations and a few discrete polygonal approximations. In particular, Kumar et al. [KML95, S. 97] have highlighted a number of advantages of algorithms based on dynamic tessellation over static tessellation. Besides reduced memory requirements, these include generating varying tessellations for different patches, gradual and smooth switching between two triangulations based on incremental techniques and use of visibility culling algorithms, including view frustum culling and back-patch culling to increase the frame-rate. To make use of dynamic tessellation algorithms, we will like to represent the LODs using spline patches.

1.1 Main Contribution

In this paper we present a new algorithm to simplify spline models. The algorithm takes a spline model as input, and generates a new representation in terms of triangular Bézier patches, known as the *C-Model*. Along with the boundary description in terms of triangular patches, it also stores topology and connectivity information as a graph. Given the C-Model, it performs a series of patch merging, vertex removal and swap diagonal operations to generate curved levels-of-detail (C-LOD). The algorithm attempts to minimize the deviation error. The graph representation is used to identify vertices on which to perform the geometric operations and update the topology information. Eventually it produces a series of C-LODs, each represented in terms of cubic triangular patches. A preliminary version of this paper has appeared as [GM98].

Some of the main advantages of this approach are:

- **Generality:** The algorithm can handle most geometric models represented using surface boundary. It has the ability to simplify all B-rep models that can be exactly represented or approximated using triangular spline patches.
- **Simplifying Spline Models:** A new algorithm for simplifying spline models and computing C-LODs.
- **Efficiency:** The algorithm can handle large objects composed of thousands of patches or polygons and simplify them in a few minutes.

- **Reduced Memory Requirements:** For spline models, the memory required to store C-LODs is at most two times the size of input models.

The algorithm has been implemented and applied to a number of models. Different models and their simplifications are shown in the color plates at the end of the paper. It works reasonably well in practice.

1.2 Paper Organization

The rest of the paper is organized in the following manner. We survey related work in model simplification, surface fitting and spline rendering techniques in Section 2. We give an overview of our approach in Section 3. Section 4 briefly introduces the reader to triangular patches, and presents algorithms to generate a C-model representation. In Sections 5 and 6 we present algorithms to generate C-LODs from the C-model representation. We describe our implementation in Section 7 and highlight the performance of our algorithm on a few models. Finally, we discuss some open issues in Section 8.

2 Previous Work

There is considerable literature on simplification of polygonal models, surface fitting and data compression, and tessellation of spline models. In this section, we briefly survey the state of the art.

2.1 Simplifying Polygonal Models

Given a polygonal model, a number of algorithms have been proposed to generate its levels-of-detail. These include algorithms based on vertex clustering [RB93], vertex removal [BS96, SZL92, Tur92] and edge collapsing [HDD⁺93, Hop96, GH97, Gue95, CMO97, COM98, EM98a]. They use different local and global error metrics for simplifying polygonal models. Cohen et al. [CVM96] and Eck et al. [EDD⁺95] have presented algorithms that preserve the topology of the original object and give a global error bound on surface deviation. [DLW93, EDD⁺95] have presented algorithms for multi-resolution analysis for surfaces of arbitrary topology types.

Other simplification algorithms include decimation techniques based on vertex removal [SZL92, Sch97] and controlled topology modification [ESV97]. [EM98b] have presented a topology simplification algorithm that produces high-quality simplifications. All these algorithms generate a few static LODs. Hoppe [Hop96] has introduced *progressive meshes* to incrementally represent various LOD. Based on progressive meshes, view-dependent simplification algorithms have been proposed [XESV97, Hop97]. Hoppe [Hop97] has also applied the resulting algorithm to small (in terms of number of patches) spline models. The algorithm pre-computes a polygonal approximation for a spline model, performs a series of edge-collapse operations on the polygonal model and stores the result as a progressive mesh. At run-time it refines the progressive mesh as a function of the viewpoint [Hop97]. While this approach has some nice properties, its main limitations arise from using static tessellation algorithms and having relatively high memory requirements.

2.2 Surface Fitting

The problem of fitting spline patches to a polygonal model has been extensively studied in computer-aided geometric design. A survey of different techniques has been given in [Die93]. These include algorithms that fit smooth spline surface over irregular meshes [Sar90, Pet95, Loo94, EH96], that use n-sided patches for fitting [LD90], and spline approximation algorithms [dBF73, dB74]. More recently, many algorithms have used subdivision surfaces for piecewise smooth reconstruction [HDD⁺94].

2.3 Tessellating Spline Surfaces

A number of algorithms have been proposed in the literature to tessellate spline models and generate polygonal approximations [AES94, FMM86, KML95, S. 97, RHD89, SC88]. Most of these algorithms dynamically tessellate a model as a function of the viewpoint. The tessellation criterion is based on the screen-space size of the triangles and on surface curvature. As compared to static tessellation schemes, dynamic tessellation results in fewer polygons for a given viewpoint and produces higher fi-

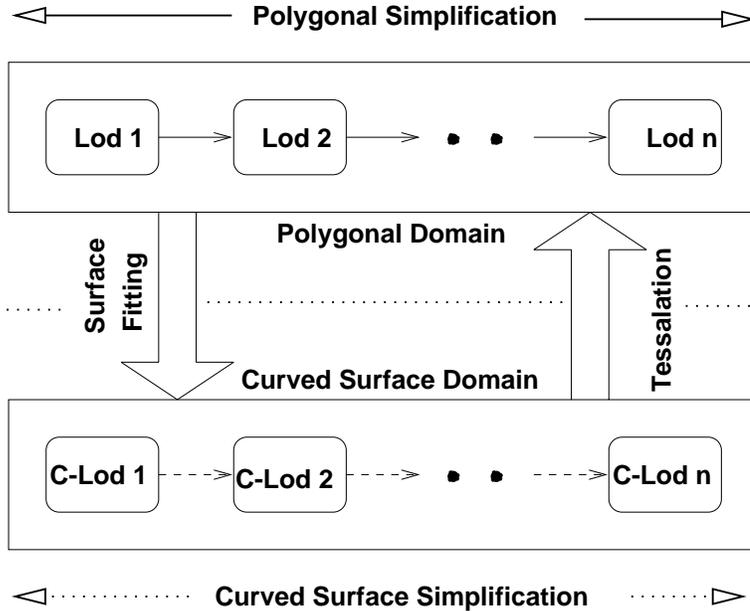


Figure 1: Relationship between Simplification, Surface fitting and Surface Tessellation

delity images. Recently, Kumar et al. [S. 97] have formulated interactive algorithms for rendering large spline models, composed of tens of thousands of spline patches on current high-end graphics systems. These algorithms incrementally triangulate a patch as a function of viewpoint. To efficiently handle large models, they group patches into super-surfaces and generate a low-resolution polygonal approximation for each of them. Furthermore, they compute a few static LODs for the polygonal approximation [S. 97]. The algorithm presented in this paper can be combined with the framework presented in [Kum96, S. 97] to efficiently handle very large models. As opposed to generating polygonal LODs, we can instead compute C-LODs of spline models.

3 Overview of our approach

Our approach for simplifying geometric models makes use of surface-fitting algorithms. It can handle polygonal as well as spline models in a unified manner. The relationship between earlier work on simplifying polygonal models (i.e. generating

LODs), tessellating spline surfaces, fitting surfaces and computing C-LODs using our algorithm has been shown in Figure 1. Our ultimate goal is to generate good approximations of the original model. As opposed to tessellating the spline models and generating LODs, we generate C-LODs directly. For interactive display, we generate a triangular approximation of these C-LODs using dynamic tessellation algorithms.

Our approach towards model simplification starts with generating a uniform representation of the polygonal and spline models in the form of a C-Model representation. The goal of the system is to generate various levels of details (C-LOD) for the C-Model. Our simplification algorithm makes use of vertex removal by merging the patches incident on that vertex. In many ways, these are extensions of simplification algorithms based on vertex removal and face removal for polygonal models. A sequence of such vertex removal operations on a C-Model will generate the next C-LOD. We define a few patch patterns and algorithms to merge the patches forming these patterns. A vertex can be removed only if the set of patches incident on that vertex matches with any of these patterns, so that they can be merged by applying that specific patch pattern merging algorithm. To identify the patterns, and thus all the removable vertices, we use a graph representation and make use of search algorithms. The major components of our system have been highlighted in Figure 2.

- **Representation Conversion:** The input to our system can be a polygonal model, tensor product patch model, or triangular patch model. Any of the above representations is initially converted to a common model representation (C-Model) in the form of a triangular patch model, with the complete adjacency information.
- **Computing mergeable edges:** *Mergeable edges* are those pairs of edges incident on a vertex that are amenable to merging. The conditions that the boundary curves of the patches should satisfy to become *mergeable edge pairs* are explained in Section 6.1. Vertices are tagged *retained* if it does not have any mergeable edge. All other vertices are tagged *undecided*.
- **Pattern Matching Graph Algorithm:** The goal here is to tag the status of

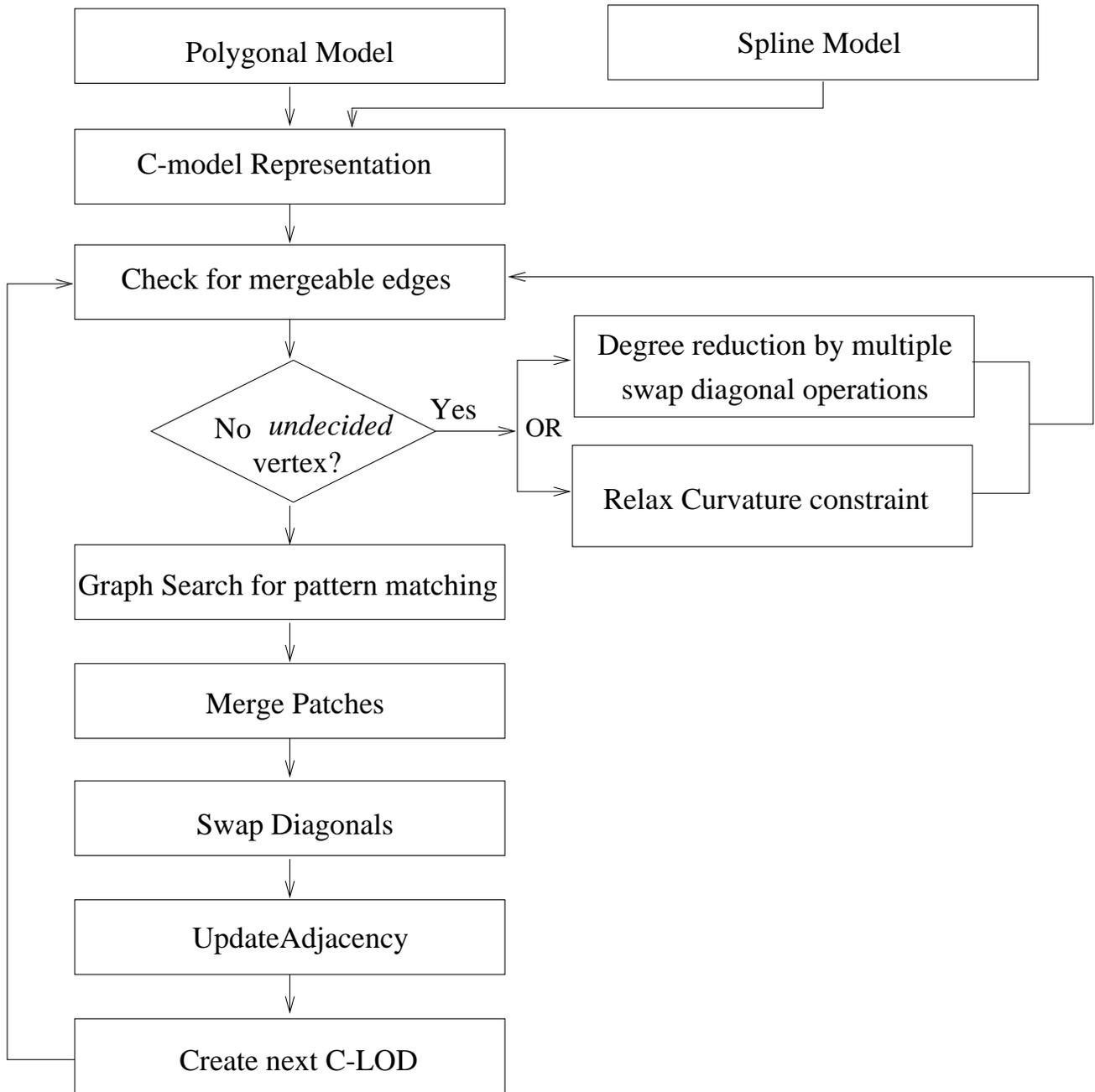


Figure 2: A Flow chart of our simplification system

all the *undecided* vertices into either *removed* or *retained*. Using the information about the *mergeable edge pairs*, patterns are matched around an *undecided* vertex. If the pattern matching is successful, then the vertex is tagged *removed* and the patches forming the pattern are tagged *merged*. The corners of the new patch formed out of merging are tagged *retained*. This process continues till there is no more *undecided* vertex.

- **Patch Mergings:** The patches that were identified for merging by the previous algorithm are merged here.
- **Swap Diagonal operation:** The *swap diagonal* operation, is performed on two adjacent triangular patches forming a rectangular patch structure. The common boundary between these patches is eliminated and a new boundary connecting the other two vertices of the rectangular structure is computed. This is equivalent to switching a diagonal line on a rectangle.
- **Adjacency Update:** After these operations, the modified connectivity and adjacency is updated only for those vertices and patches whose adjacencies have been affected.
- **C-LOD Generation:** This module creates the next C-LOD. This includes system cleanup and various bookkeeping operations to prepare the system for the next iteration.

The merging operation introduced in this section is based on the properties of the triangular patches and the well known de Casteljau subdivision algorithm for triangular patches.

4 Triangular Bézier Patches

The Bézier triangular surface [Far93] of degree n can be written as,

$$\mathbf{P}(u, v, w) = \sum_{(i,j,k), i+j+k=n} \mathbf{P}_{i,j,k} B_{i,j,k}^n(u, v, w) \quad (1)$$

where $\mathbf{P}(u, v, w)$ is a point on the surface, with the constraint $w = 1 - u - v$ and $i + j + k = n$. An additional constraint, $u + v \leq 1$, makes the parametric domain

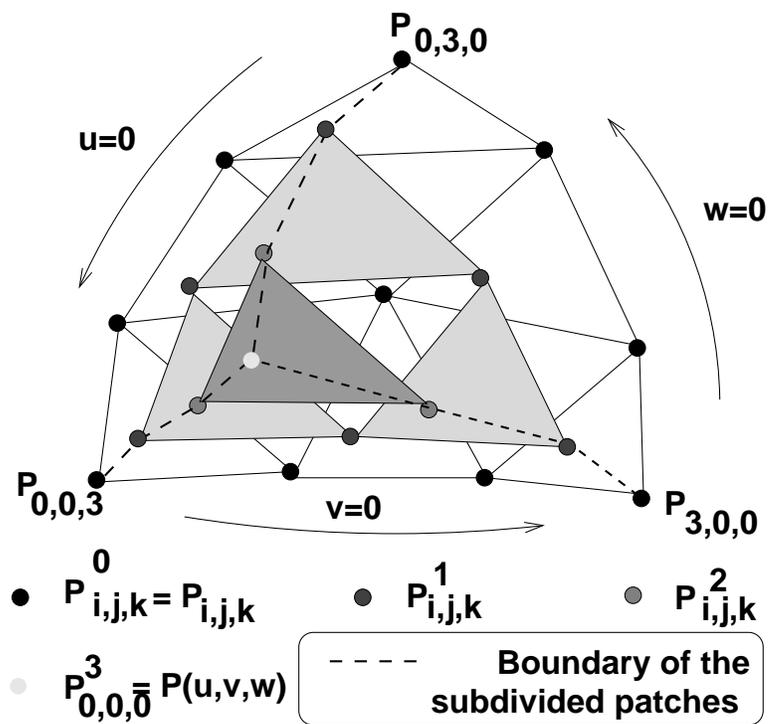


Figure 3: de Casteljau algorithm for subdivision

a triangle. The constants $\mathbf{P}_{i,j,k}$ are the control points which form a triangular net (Figure 3). The blending functions $B_{i,j,k}^n(u, v, w)$ are Bernstein basis functions of degree n given by

$$B_{i,j,k}^n(u, v, w) = \frac{n!}{i!j!k!} u^i v^j w^k \quad (2)$$

We consider only polynomial triangular patches, though the techniques given in this paper can be directly extended to the rational patches also. In the rest of the paper, we refer to Bézier triangular patches as just ‘patches’.

4.1 G^1 Continuous Triangular Patches

For two patches to be G^1 continuous, the triangles joining the control points along the border of the patches (i) should be pairwise planar (*coplanarity* condition) and (ii) should be an affine transformation of the domain triangles (*affine pairs* condition) (Figure 5). In other words, by condition (ii), in Figure 5, the quadrilaterals formed by the triangles $T1$ and $T2$, and $T3$ and $T4$, should be an affine transformation of the quadrilateral formed by $T5$ and $T6$. These two conditions are necessary and sufficient conditions for the patches to be G^1 continuous[Far93].

The three boundary curves of a triangular patch are Bézier curves formed by the boundary control points of the patch. It is important to note that two adjacent Bézier curves are G^1 continuous (their tangents are in the same direction, at the common end point), when the three control points – the common control point, and its adjacent control points on either curve – lie on a straight line. We refer to it as an *edge continuity* condition.

4.2 Subdivision and Merging Algorithms

A triangular patch can be subdivided into, in general, three triangular sub-patches using the de Casteljau algorithm. This is also the basis of the blossoming principle for triangular patches.

Given the control points $\mathbf{P}_{i,j,k}$ and a parametric vector $\mathbf{u} = (u, v, w)$, the de Casteljau algorithm computes, at each iteration, a sequence of sets of control points, using the following equation.

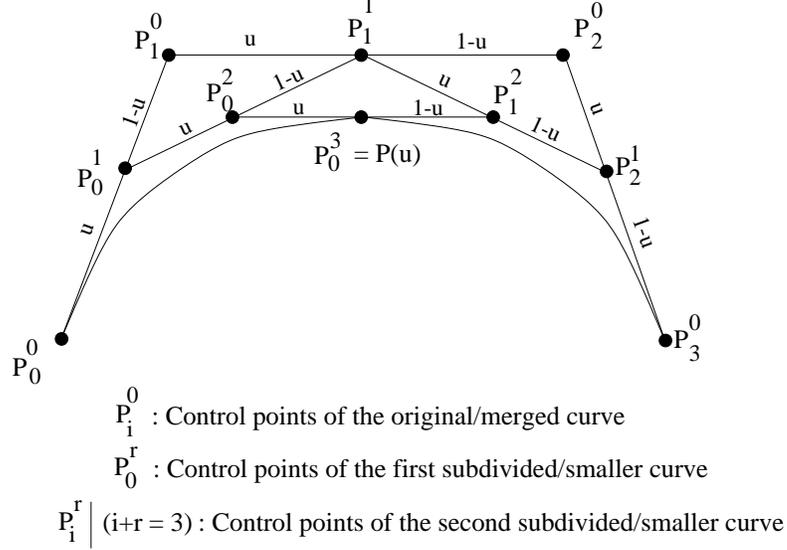


Figure 4: de Casteljau subdivision algorithm for Bézier Curves

$$\mathbf{P}_{i,j,k}^r(\mathbf{u}) = u\mathbf{P}_{i+1,j,k}^{r-1}(\mathbf{u}) + v\mathbf{P}_{i,j+1,k}^{r-1}(\mathbf{u}) + w\mathbf{P}_{i,j,k+1}^{r-1}(\mathbf{u}) \quad (3)$$

with the condition that $i+j+k = n-r$, where n is the degree of the patch and r is the iteration count. Further, $\mathbf{P}_{i,j,k}^0(u, v, w) = \mathbf{P}_{i,j,k}$. The result computed, $\mathbf{P}_{0,0,0}^n(u, v, w)$, is actually the point on the triangular patch for the parametric value (u, v, w) . As shown in Figure 3, the intermediate points generated by the above algorithm, are the control points of the sub-patches of the original patch.

If the patch is subdivided at a parameter (u, v, w) , where one parameter is zero, then the de Casteljau subdivision would yield two triangular patches instead of three. The new curve created, connecting a corner to a point on the opposite edge(curve), is called a *radial line*. Further, the boundary curve that is subdivided into two in this process, actually undergoes de Casteljau subdivision of a Bézier curve (Figure 4).

We formulate a method using the inverse of de Casteljau algorithm to merge patches. If three patches are obtained by the de Casteljau subdivision of a single patch, then by performing the exact reverse computation, we will be able to get the control points of the original patch. This requires computing the parametric value

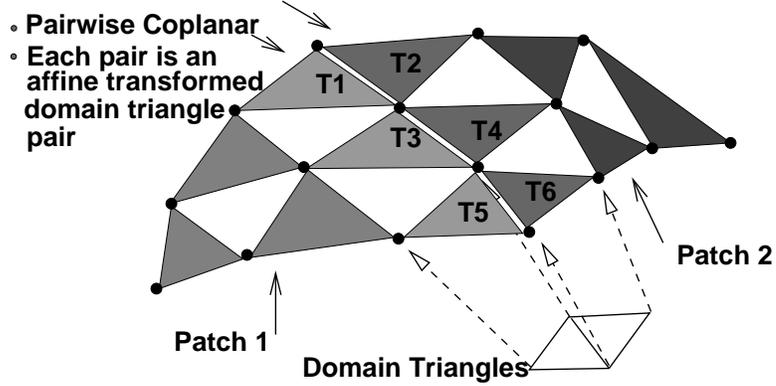


Figure 5: Conditions for G1 continuity

(u, v, w) at which these three patches were subdivided. We know that the point $\mathbf{P}_{0,0,0}^n$ is on the patch and also on the triangle formed by $\mathbf{P}_{1,0,0}^{n-1}$, $\mathbf{P}_{0,1,0}^{n-1}$, and $\mathbf{P}_{0,0,1}^{n-1}$. The barycentric coordinate of $\mathbf{P}_{0,0,0}^n$, on the above triangle gives the parametric value (u, v, w) . Using this, by reverse calculation, we can compute $P_{1,1,1}$ of the original patch. The boundary control points of the original patch are same as the boundary control points of the sub-patches. If the original patch was divided into two patches, the computation of (u, v, w) becomes simpler. One of the parameters is zero, and the others can be computed from the inverse de Casteljaun algorithm for Bézier curve, as outlined in [Far93]. This involves finding the parameter at which the common corner control point divides the straight line joining its adjacent two boundary control points (Figure 4). In this merging operation, three new control points, two on the boundary curve and one at the center, have to be computed for the new patch. These techniques are used in our algorithms for patch merging.

5 Generating C-Models

In this section we discuss algorithms to convert a few types of model representations to our C-Model representation. Our C-Model representation consists of a set of cubic Bézier triangular patches along with adjacency and topology information. It also contains other useful input information about patches that is explained in Section

8. We have chosen cubic triangular Bézier patches for various reasons. These include reduced space requirements and fast algorithms for tessellating them into polygons. Furthermore, the cubic patches provide us with sufficient degrees of freedom for surface fitting. Apart from having good mathematical properties, cubic patches have the right number of parameters and unknowns to work with for our algorithm. The following subsections explain various algorithms we use to convert polygons and tensor-product patches into triangular patches.

5.1 Triangular Patch Approximation for Polygonal Models

Though the simplification algorithm is designed to simplify cubic Bézier patch models, we can apply these algorithms on polygonal models also, by converting them to the C-Model representation. This gives a unified approach for simplifying both polygonal and spline models.

To convert the polygonal model into a C-Model, we start with fitting one triangular patch for every triangle. An algorithm to compute the control points for such a triangular patch, is given below.

A cubic patch has ten control points. We use the following naming convention: three ‘corner’ control points, six ‘boundary’ control points and one ‘center’ control point. The goal now is to position these ten control points, to ensure the coplanarity condition, with a reasonably ‘good’ surface fit over a polygon.

Choosing the corner control points: The three vertices of a triangle become the three corner control points of the patch.

Choosing the boundary control points: Each corner control point with its two adjacent boundary control points, define the tangent plane at that corner. We also know the tangent plane from the normal at each corner vertex. We choose one point on each of the two edges of the given triangle, that are incident on one corner, say A . The projection of these chosen points on the tangent plane at A gives us the two boundary control points adjacent to A . A reasonable choice of a point on the edge would be the one at a distance of one-third of the edge length away from A . The same method is repeated on the other two corners to get all the six boundary control

points.

It is important to note here that a vertex has multiple normals, if it is on an edge or crease. We can make use of this information to avoid G^1 continuity in such cases, to maintain the edges and creases. Now we need to choose the only remaining control point, the center control point. It is initialized to be the average of the six boundary control points. The computed position is refined after all the triangles are fitted with patches, making it a two-pass algorithm.

Refining the center control point: Let us consider one boundary curve, say edge A , of the patch. We know from the coplanarity condition that the center control point, two boundary control points of the edge A , and the center control point of the patch adjacent on the edge A , should be coplanar. For three edges we have three such planes, and the center control point is the intersection of these three planes. Let us construct the plane for the edge A . The cross-product of the line joining the boundary control points of A and the line joining the two center control points of the adjacent patches, defines the normal to the plane we are looking for. One of the boundary control points, with this normal, defines the plane. The three plane intersection point is then computed by solving three linear equations. If the system is close to being singular, then the original estimate of the center control point is retained. If the singularity is because of a planar surface, then the original estimate would retain the planarity. If a sharp curvature along the edges of the patches lead to the singularity of the system, then the original estimate would retain the sharp curvature.

Refining the center control point for a boundary patch: The polygons whose edges define a boundary of the surface, are called boundary polygons. Boundary patches are the patches fitted over a boundary polygon. In such cases, in the absence of adjacent patches, refining the center control point becomes an under-constrained problem. Under these circumstances, the center control point found in the first pass is projected on the line (in case of one boundary edge), or onto the plane (in case of two boundary edges).

5.2 Triangular Patch Approximation for Tensor Product Patch Models

This section explains our algorithm to convert a bi-cubic tensor-product Bézier patch into two triangular patches. If the patch degree is less than three, the patch is subjected to degree-elevation to make it bi-cubic. If the tensor-product patch is of a higher degree, then it is subdivided to smaller patches so that each subdivided patch can be approximated by a bi-cubic patch. The non-isoparametric curves of a bi-cubic patch are of degree six. So, the degree of the triangular patches would be six to exactly represent a bi-cubic patch. We approximate each bi-cubic patch with two cubic triangular patches.

The boundary control points of the tensor-product patch become the boundary control points the two triangular patches. The diagonal curve of the tensor-product patch, which is of degree six, is the third and the common boundary curve for the two triangular patches. We need to approximate the diagonal curve with four control points (a cubic curve).

A bi-cubic tensor product patch is given by the equation:

$$\mathbf{P}(u, v) = \sum_{i=0}^3 (\sum_{j=0}^3 \mathbf{P}_{i,j} B_j(v)) B_i(u)$$

The points computed within the parenthesis are the control points for the cubic isoparametric curve for a constant v . In the diagonal curve, $u = v$, and hence they are dependent. A simple extension of the algorithm highlighted above gives the approximation of the control points of the diagonal curve, as given below.

$$\mathbf{P}_i = \sum_{j=0}^3 \mathbf{P}_{i,j} B_j\left(\frac{i}{3}\right), \forall 0 \leq i \leq 3$$

where \mathbf{P}_i 's are the new control points. The center control points of the two triangular patches are found in the same way as given in Section 5.1. Similar methods can be devised to convert n-sided patches into triangular patches.

6 Patch Merging Algorithms

In Section 4, we reviewed methods to combine two or three patches into one big patch when the smaller patches were obtained by the de Casteljau subdivision of the big

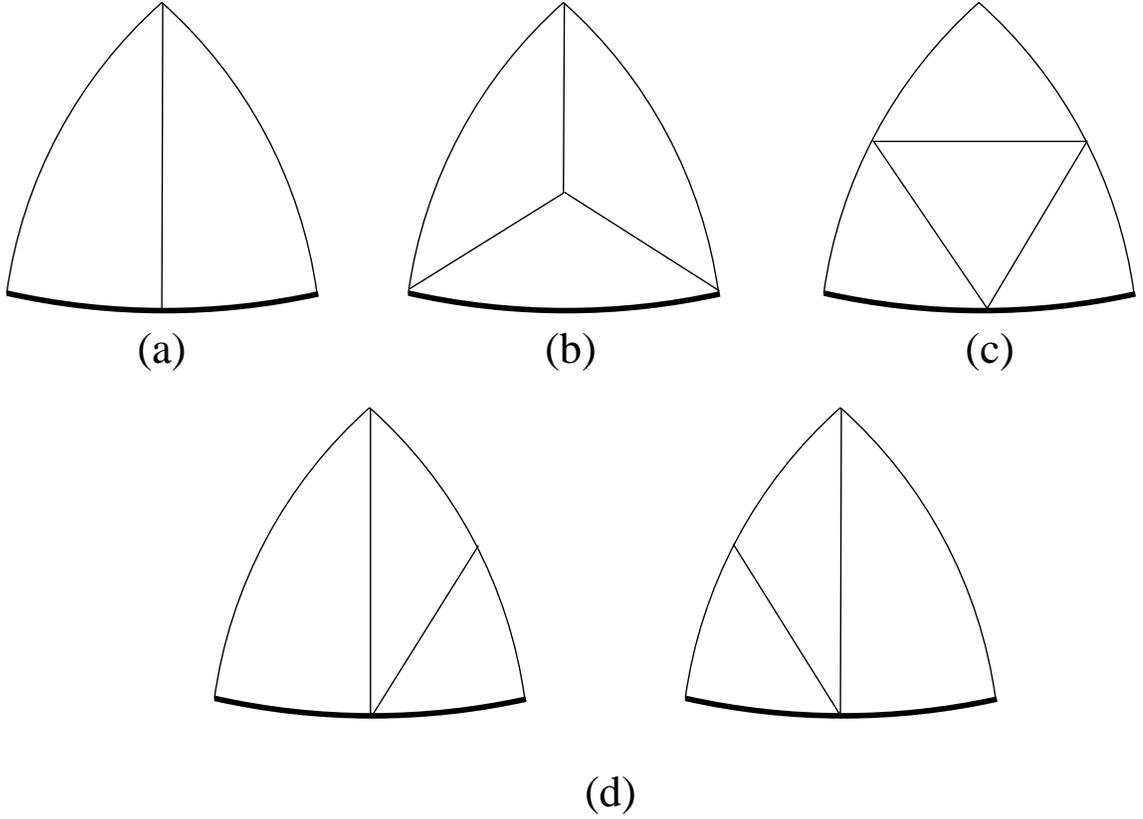


Figure 6: Patterns for merging: (a) T-pattern (b) Star Pattern (c) T-in-T pattern (d) 2T patterns. The dark edges are mergeable edge pairs

patch. In this section, we extend this idea to merge any combination of G^1 continuous patches. As this would introduce surface deviation error, we try to minimize this error by imposing various constraints on the patches that are merged.

The foremost requirement for the patches to be mergeable is G^1 continuity. In our application, the patches might not be G^1 continuous to start with. Making the triangular patches in a model G^1 continuous, especially to ensure the *affine pairs* condition, is an optimization problem [Man97]. So, in our system, we only ensure the *coplanarity* condition for geometric continuity. In practice, we obtain good results by only using this constraint.

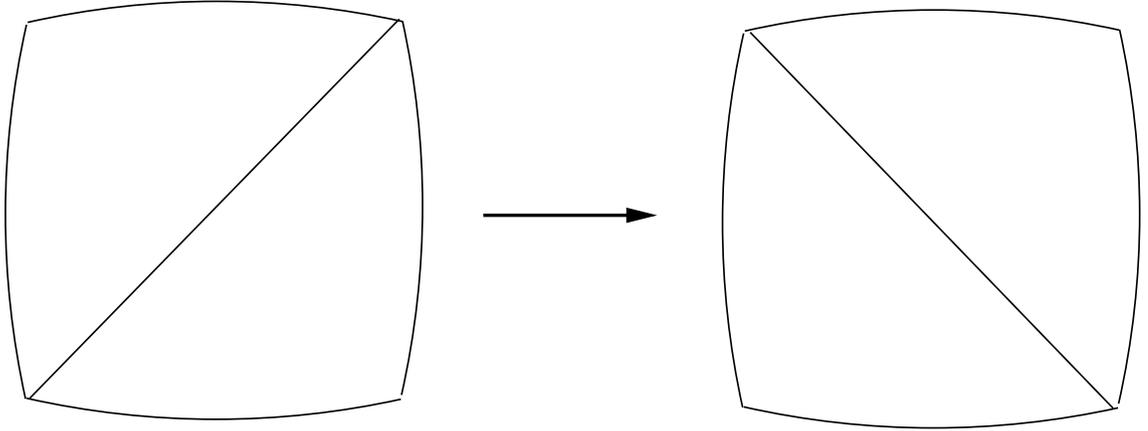


Figure 7: Swap Diagonal Operation

6.1 Merging Patterns

A set of patches can be merged only if they satisfy few geometric constraints and match with one of the defined patterns, in their local topology. Merging algorithms are defined only for these patch patterns. The merging patterns are illustrated in Figure 6. The dark edges are *mergeable edge pairs* for the common vertex of those edges.

We impose two conditions for two edges (curves) incident on a vertex V to be the *mergeable edge pairs* of V . The first is the *edge continuity* as explained in section 4.1. In other words, the common control point V should be on the line joining the two adjacent control points of the two curves. The second condition is the *restricted incident faces* condition. According to this condition, the edge pair should have either 0, 2, or 3 faces between them in both clockwise and counter-clockwise directions. It is possible to have no face between the edge pairs, if the vertex is a boundary vertex. For boundary vertices, only the boundary edges can be mergeable. The use of *restricted incident faces* condition will be made clear later in this section.

There are four patterns for merging, as shown in Figure 6. The natural classification of these patterns are based on the number of *mergeable edge pairs* each pattern has. The *star* pattern has no *mergeable edge pair*, the *T* pattern has one, *2T* patterns have two, and finally, the *T-in-T* pattern has three *mergeable edge pairs*. The pat-

terns T and $star$, are handled exactly the same way as described in Section 4.2. These patterns can be merged using inverse de Casteljau algorithm, as there is a direct de Casteljau algorithm to divide a triangular patch to get two or three patches.

The $star$ pattern has no mergeable edge pairs. But there are several ways to check whether the patches forming the $star$ are mergeable or not. One simple test is to check the deviation of the common control point for all three patches from the plane formed by its three adjacent control points. Another test is to find deviation of the normal of the common vertex, computed for all three patches. The presence of *mergeable edge pairs* is an essential condition for all patterns except the $star$ pattern. Although, we can decompose a $2T$ pattern into two T patterns, we use this prototype for implementation convenience.

Unlike the T and $Star$ patterns, there is no equivalent subdivision process for the patches merged by a T -in- T (Triangle-in-Triangle) pattern. The three *mergeable edge pairs* in this pattern are merged according to the inverse de Casteljau algorithm for Bézier curves, and the center control point is found by the algorithm described in Section 5.1. This pattern cannot be directly extended to higher degree triangular patches, because they have too many degrees of freedom.

Any merging pattern explained here allows not more than three faces around a vertex to be merged on one side of its *mergeable edge pair*. This explains the *restricted incident faces* condition required by the edge pair to qualify as *mergeable edge pair*.

6.2 Swap Diagonal operation

A *swap diagonal* operation shown in Figure 7, is not a patch merging operation. But it is very useful in reducing the degree of a vertex as shown in Figure 9. It is also useful in stopping the propagation of vertex removals by the graph algorithm which is explained in the next section. The algorithm to swap a diagonal is not as straight forward as in the case of two adjacent planar triangles. As we are dealing with curved surfaces, the diagonals are space curves. Our goal is to find the diagonal curve which is planar, or at least with minimum deviation from a plane.

The *swap diagonal* operation is performed in two steps (Figure 8). First the two

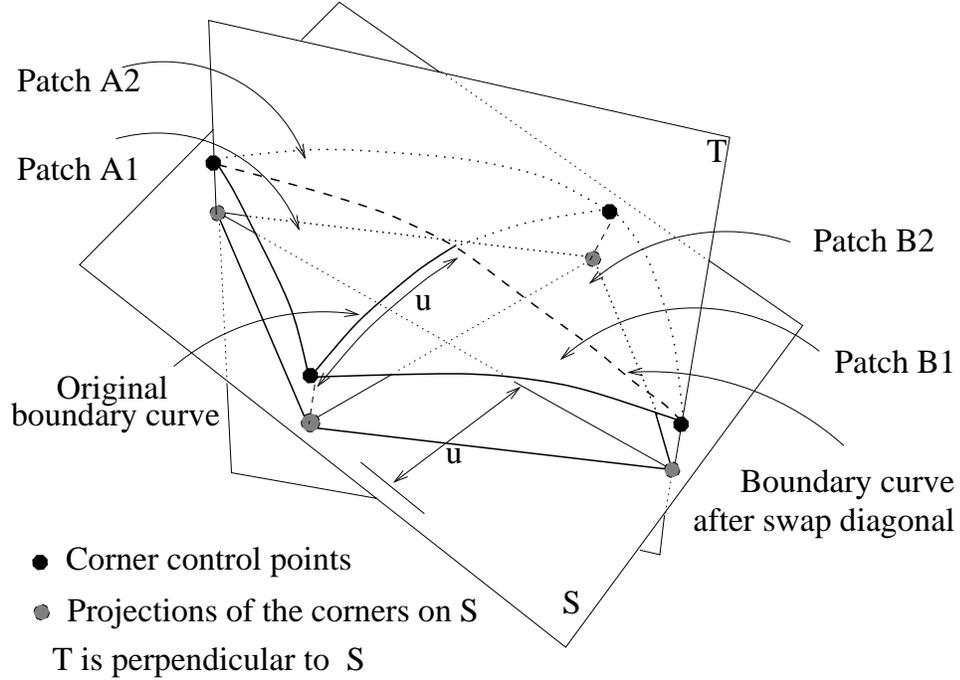


Figure 8: Swap Diagonal Operation

patches, say A and B , are subdivided into two patches each, say $\{A_1, A_2, B_1, B_2\}$, at the same point on the common boundary curve, using the de Casteljau algorithm. In the second step, A_1 is merged with its adjacent sub-patch of B , say B_1 , and A_2 with B_2 , using the inverse de Casteljau algorithm. This swaps the diagonal of the rectangular pattern (Figure 8). The main issue is to find the parameter at which the original patches A and B have to be split, so that in the second step while merging, the sub-patches satisfy the *edge continuity* condition.

We use a simple algorithm to approximate this parameter. We fit a plane, S , that approximates the four corner control points of the rectangular structure formed by A and B . The normal to the plane is computed as the cross product of the lines joining the opposite corners of the rectangle. On this plane, the four points are projected. The existing common boundary curve is represented as a straight line on the plane. The parameter at which this line is divided by the other diagonal is taken as the required parametric value for our subdivision step.

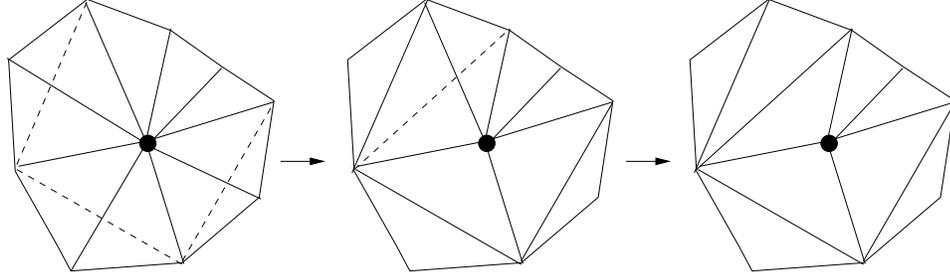


Figure 9: Degree Reduction using Swap Diagonal Operation

7 Graph Algorithm for Pattern Matching

In the previous section we introduced various patterns, or prototypes, of the triangular patches that can be merged. In this section, we describe an algorithm to identify such patterns in the boundary description. Once these patterns have been identified, the patches forming the pattern are merged according to that specific pattern merging rule to yield one single patch. This patch is substituted for the merged patches, thus reducing the patch count of the model to generate the next C-LOD.

Let us assume that we are given a set of vertices and their connectivity in the form of adjacency. A vertex can be tagged either *removed*, *retained*, or *undecided*, with obvious meanings. An *undecided* vertex can be made into a *retained* vertex or a *removed* vertex. But once they are tagged *retained* or *removed*, their status is not changed for that particular iteration.

As the presence of a *mergeable edge pair* is essential to finding a patch pattern around a vertex of degree greater than three, non-existence of *mergeable edge pair* would make the vertex to be tagged *retained*. As a result of the *restricted incident faces* condition, any vertex with degree (number of edges incident on the vertex) more than six will also be tagged *retained*. If all vertices are tagged *retained*, multiple swap diagonal operations have to be performed around a few vertices, to reduce their degree to less than or equal to six (Figure 9).

The vertices which are not tagged *retained* are tagged *undecided*. Let us also assume that for each *undecided* vertex, we are provided with its *mergeable edge pairs*.

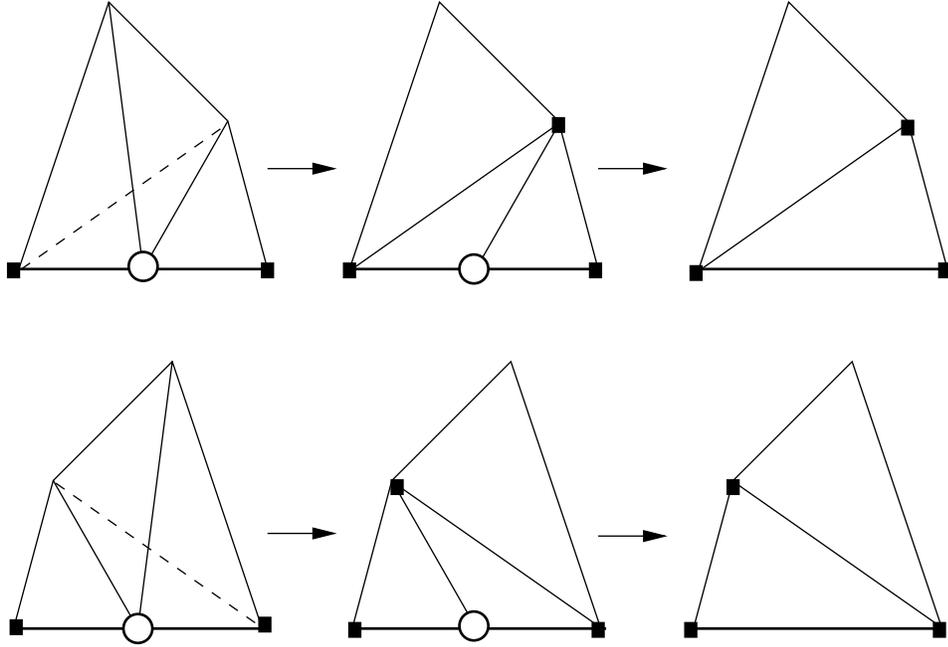


Figure 10: Generalized algorithm for three faces merging

The goal is to tag the *undecided* vertices as either *retained* or *removed*. If it is *removed*, then find patterns among the patches incident on that vertex, for patch merges, so that no patch has this vertex as its corner. It is clear to see that in T and *star* patterns there is one *removed* vertex, in a $2T$ pattern there are two *removed* vertices and in a T -in- T pattern there are three *removed* vertices. In all the cases there are three *retained* vertices.

With the topology information being coded in the form of *mergeable edge pairs*, we pose the problem of identifying patterns in this topology, as a graph searching problem. The vertices of the model are mapped to the vertices of the graph, and the boundary curves of the triangular patches, to the edges of the graph. We use depth first search algorithm to find patterns.

In a general case, this recursive algorithm takes as input, a vertex V , which is tagged *removed*, with its *mergeable edge pair*. The goal of the algorithm is to match a pattern on only one side of the *mergeable edge pair*. Hence it is assumed that that a pattern has been matched on the other side of the *mergeable edge pair*, using the

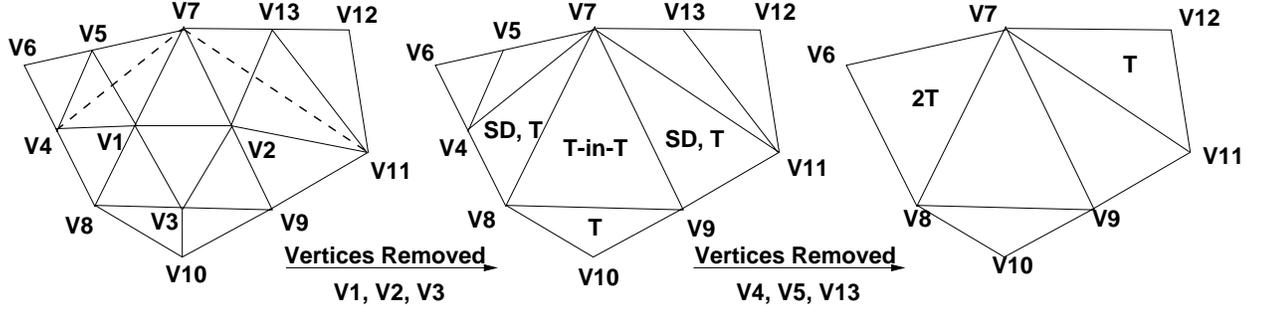


Figure 11: Illustration of the graph algorithm

same solution. We call the side at which a pattern has to be found as an *unmatched side*. It is important to note that the two adjacent vertices of V along its *mergeable edge pair*, are *retained vertices*.

There can be either 0, 2, or 3 faces on the *unmatched side* of V , as any other number of faces would contradict the *restricted incident face* condition for the given *mergeable edge pair*. If there is no face on the *unmatched side* of V , then we are done. If there are two faces then that portion of the geometry can be matched with a T pattern, and the recursion returns without any further recursive calls. In this pattern merging, if the third vertex, say R , is *undecided* or *retained*, is tagged *retained*. If R is already tagged *removed*, it is left unchanged. It can be proved that V is in the *unmatched side* of R , and after the removal of V , the pattern existing in the *unmatched side* of R is a T pattern. This T pattern will be merged, when the recursion returns to process R .

If there are three faces in the *unmatched side* of V , then we check the geometry on the *unmatched side* for one of the two $2T$ patterns or the T -in- T pattern. If none of these patterns match, then we adopt the following generalized algorithm to handle three faces on the *unmatched side* of V . The immediate neighborhood of any such vertex is *topologically* equivalent to the geometry shown in Figure 10. The vertex to be removed is denoted by a circle, and *mergeable edge pair* is shown as a thick edge. The shaded vertices are retained vertices. This geometry can be reduced to a T pattern by one swap diagonal operation, and the vertex under consideration can be removed,

without generating any more removable vertices. In this method, tagging the vertices to be *removed* or *retained* has to be done carefully. The vertices on the either side of the *mergeable edge pair* are always *retained* vertices. Among the other two vertices, one loses an edge and the other gains an edge because of this swap diagonal operation. The vertex that gains an edge should be either *undecided* or *retained* before the swap diagonal, and has to be tagged *retained* after the operation. The state of the vertex that loses an edge is not changed.

If a $2T$ pattern is matched, on the *unmatched side*, then a new *removed* vertex and a new *retained* vertex are generated. It is obvious that one side of the *mergeable edge pair* of the new *removed* vertex is already matched (with $2T$ pattern), and it is left with one *unmatched side*. This satisfies the generalized condition for this recursive algorithm to be applied on it. Similarly a $T\text{-in-}T$ pattern generates two such new *removed* vertices. The above pattern matching algorithm is applied to the removed vertices recursively. It is also possible that the other side of the new *removed* vertices are already matched in the course of recursion. In such a case, the recursion terminates.

It can be seen that even if a $2T$ or $T\text{-in-}T$ pattern is identified on the *unmatched side*, the generalized solution can be applied. The advantage of the generalized solution is that it generates no more removable vertices, and is applied to stop the propagation of vertex removals.

The recursion starts with a vertex where both sides of the *mergeable edge pairs* are *unmatched sides*. On one side, we match patch patterns, and in case of a three face pattern, we stop the recursion immediately using the generalized algorithm for handling three faces. Then the second *unmatched side* satisfies the required condition for the above explained algorithm. An example for pattern matching graph algorithm is given in Figure 11, where $V1$ is the first vertex to be removed, and SD denotes a swap diagonal. It can be seen that different sets of patch patterns are possible in the same topology. We attempt to find just one set of patch patterns out of various possibilities. As each vertex is checked once during the graph search algorithm, its

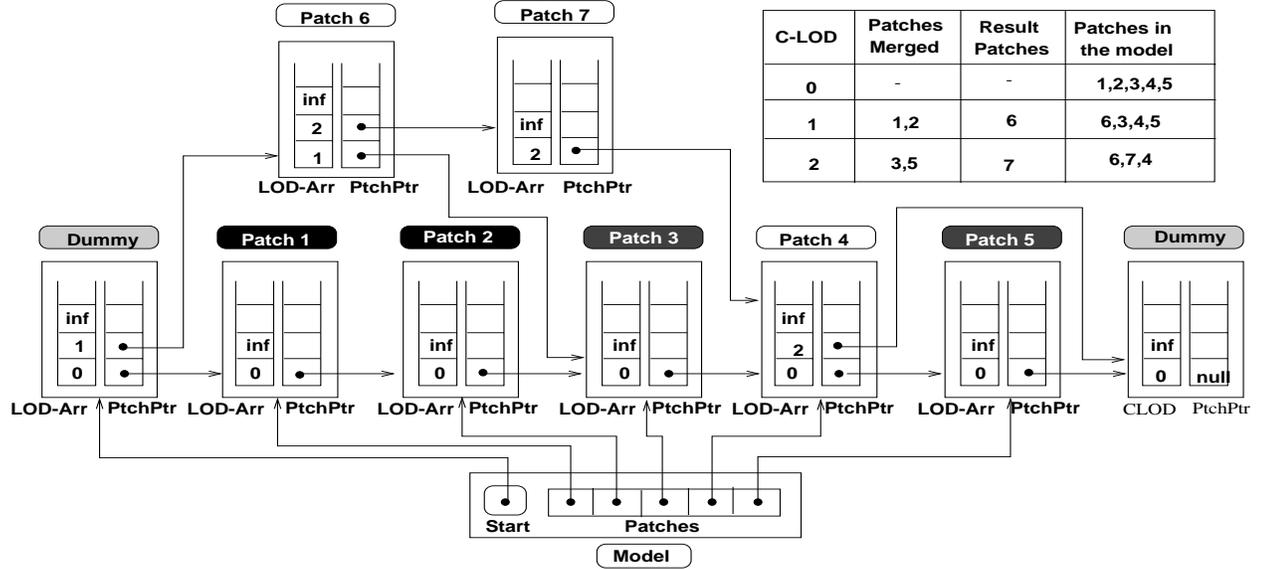


Figure 12: Patch traversal for various C-LODs

complexity is $O(n)$, where n is the number of vertices retained in the previous level of detail.

8 Implementation

In this section, we present implementation details of our simplification system. The algorithms described in this paper have been implemented in C++. We will discuss the system in stages as shown in Figure 2.

C-Model Representation: The polygonal and spline representations are initially converted to the C-model representation. The C-model also has information about the curvature and size of every cubic triangular patch, which are used to decide whether two adjacent patches can be merged or not. A high curvature edge is not merged with any other edge, and a relatively small patch is chosen for merging at the beginning of the pattern matching algorithm.

Computing mergeable edge pair: The *edge continuity* condition for *mergeable edge pairs* is checked by comparing with a tolerance value the angle of deviation of the two line segments connecting the common control point to its adjacent neighbors

in the two curves. At higher levels of detail, this tolerance is slightly relaxed. All vertices are *undecided* to start with. If a corner does not have any mergeable edges then it is tagged *retained*.

Pattern Identification by graph algorithm: As explained in Section 7, the graph searching algorithm identifies patches to be merged and the vertices to be removed. The actual merging operation and the vertex removal operation is not performed by the graph searching routine. The patches to be merged are logged in a data structure class called '*ToBeMerged*'. This is an array of sets-of-patches to be merged. Furthermore, whenever a swap diagonal operation is to be performed, it is logged into another class called '*ToBeSwapped*'.

Implementing MergePatches: The 'MergePatches' function merges every set of patches in the '*ToBeMerged*' array. The adjacency information can be updated from the data structures.

Implementing SwapDiagonal: The 'SwapDiagonal' function swaps diagonal of all the related patches logged in the '*ToBeSwapped*' array. Unlike the 'MergePatches' operation, the adjacency is updated immediately after each 'SwapDiagonal', to avoid an inconsistent state in the topological information introduced by the swap diagonal operation. This inconsistent state is due to the increase in the number of edges, hence the faces, of two of the vertices involved in the swap diagonal. Other merging operations will either maintain the edge count or decrease it.

Adjacency Update: Incremental update of adjacencies and connectivity is performed only for those vertices whose adjacency has been affected by the above operations. The adjacency information involves computing the connectivity between the corners and patches, ordering of edges and faces around a corner, and other book-keeping operations for the affected vertices. Every adjacency update is stored in an array in the vertex data structure, and is tagged with the present C-LOD number. If the C-LOD number of the array i is C_i and that of $i + 1$ is C_j , then for any other C-LOD number between C_i and C_j , array i stores the adjacency information.

Resetting the System: This involves tagging all vertices *undecided*, freeing the

unused memory in the structures related to removed vertices and merged patches, and resetting the counters and other system variables. After resetting, the system is ready for the next iteration.

8.1 Data Structures

In this section, we highlight the data structures used in our implementation. The main classes in the system are *Model*, *Corner*, and *Patch*. From the *Model* class, all C-LODs of the given model can be obtained. It has *Vertices[]*, an array of pointers to the *Corner* class, and *Patches[]*, an array of pointers to original *Patch* class. As no corner is added to the system during the simplification process, *Vertices* is not changed. If the corners are removed, they are tagged as *removed*, and not removed from the data structure. On the other hand, new patches are added to the system by the merging operation. Pointers exist in the *Model* class, only to the patches in the *original* model. No direct pointer is provided for each generated patch. Patches belonging to a specific C-LOD of a C-Model can be extracted by an interesting *Patch* class traversal using a linked list. In Figure 12, to get all patches belonging to a particular C-LOD, say i , the linked list is to be traversed from *start*, by choosing *PatchPtr[j]*, such that $LOD-Array[j] \leq i$, and $LOD-Array[j+1] > i$. Direct pointers to the patches are provided from *Model*, for random access of patches required in various components of the system.

The algorithms and data structures have been designed to save run-time memory usage. A good balance of speed against memory is achieved in the implementation. In all the functions, the option of parallel implementation is kept open by carefully designed data-access patterns.

8.2 Generating Levels of Detail

The above modules can generate subsequent C-LODs, while there are *undecided* vertices and patch patterns to remove them. The absence of *undecided* vertices may be for two reasons: the degree of the vertices are greater than six, or there is no *merge-*

(a)	(b)	(c)	(d)	(e)
Phone	Polygonal	43537	1623	235
Bunny	Polygonal	69491	1000	289
Lion	Tensor Patch	4632	402	10.7
Armadillo	Tensor Patch	5100	397	8.7

Figure 13: Performance of the system: (a)Model (b)Model representation (c)Initial patch count(d) Final simplified patch count (e) Total time taken (secs)

able edge pair. When the simplification process stops because of these conditions, the multiple swap diagonal operations are performed to reduce the degree of a few chosen vertices, as explained before (Figure 9). To make more edge pairs *merge-able*, we move the boundary control points of all the patches, without introducing cracks in the model, to make the boundary curves planar. Apart from increasing the possibility of edges becoming *mergeable*, this process also makes the quality of the merged patches better in terms of surface deviation error. Relaxing the curvature constraint also alleviates this problem.

9 Performance

The algorithms presented in this paper have been applied to various polygonal and spline models. Here we highlight the results on three models, two of which are polygonal models, and two are tensor product Bézier patch models. The performance numbers for these models are tabulated in Figure 13. The Phone and the Bunny model were given in ‘ply’ format with polygon adjacencies. The Lion and the Armadillo models was converted to a triangular patch model and adjacencies were generated off-line. The time given in the table, includes the C-Model representation conversion time and the total time to generate various levels of detail until the patch count given in the table is achieved. For the polygonal models, we computed 200 C-LODs and

for the tensor product patch models we had 14 C-LODs. Every patch in the Lion model has been hand-crafted to contribute to represent features of the model, so the simplification achieved in this model, cannot be directly compared with the simplification achieved in polygonal models. Further Lion model had many unconnected components, and further simplification introduced holes in the model. The Armadillo model, on the other hand, though had more number of patches than the Lion model, had a single connected component. Hence the Armadillo model, not only took lesser time because of easy propagation of patch mergings in a single large connected component, the algorithm was also able to do a better simplification in terms of number of patches than the Lion model.

Curved surface simplification is much more complicated than polygonal simplification, not just in terms of computation, but also in terms of dealing with degenerate models. In a polygonal model, when two triangles have same set of vertices, we know that they are coplanar triangles, and hence can handle such a scenario appropriately. But in case of triangular patches, even if the corner vertices of two patches are same, because of varying curvature, they need be the same patch. Any algorithm which uses the corners of the triangular patch to represent the topology of the model will have to do more comparisons on the other control points, if two patches have the same corner vertices, to decide whether they represent the same patch. In another scenario, in the polygonal model, there can be only one edge connecting two vertices. But in curved patches, there can be multiple curves (edges) connecting two vertices. Further complications are introduced, when a single triangular patch has two of corners same. Most of the degeneracies can be handled by our system.

The results shown here are from the prototype of our system. All the timings presented here were measured on an SGI-Onyx with an R10000 processor, 195 MHz clock.

10 Error Analysis

An important component of a simplification algorithm is to compute a tight error bound on the simplified model. In our algorithm no strict error metric had been imposed on the simplification, except in the form of *mergeable edge pairs*. The error metric we are working on is based on the surface deviation from the original model of the resulting patch after merging. Many polygon simplification algorithm keep track of deviation error metric during the simplification algorithm [Hop96, GH97, CMO97, COM98] and use a greedy strategy to minimize the error during each local operation. However, exact maximum surface-to-surface distance is rather expensive and we don't update after each patch merging operation. The results of the final error deviation computation can be used by a viewing program, that gives guarantees on the pixel deviation.

The error introduced in patch merging can be easily computed in the following manner. First the new patch obtained by the patch merging operation is subdivided using the de Casteljau algorithm at the same parameter at which it was merged. If there was no error in the merging operation, then these subdivided patches would exactly match original patches. Hence, a good error estimate is to find the maximum distance between the corresponding control points of these subdivided patches from those of the original patches. We can also use a weighted distance function, as the deviation of the corner control points would involve more error than the same deviation of the center control points. This error can be accumulated incrementally over various C-LODs.

All the patch patterns except the Tri-in-Tri pattern has a corresponding subdivision process. Hence for these patterns, the error can be estimated as above. But in the case of a Tri-in-Tri pattern, there is no such subdivision algorithm. Hence a better way of determining the error has to be devised. As the swap diagonal operation has been implemented as patch subdivision followed by two 2T patch merging operation, the error computation can be performed in the same way as that of a 2T pattern. The patch subdivision operation is accurate to the precision of the floating

point hardware.

We have observed that the swap diagonal operation, which does not contribute to the patch reduction, can introduce deviation error as it involves a patch merging operation. We are currently working towards a better *swap diagonal* algorithm and its implementation. Some error is also introduced by not satisfying the *affine pairs* condition for G^1 continuity. We are planning to implement the algorithm presented by [Man97]. This is based on local parametric scheme that improves surface shape by empirically proven improved settings of the free parameters.

11 Conclusion and Future Work

In this paper, we have presented a new algorithm for simplifying spline models without tessellating them into polygons. Our approach is general and also applicable to polygonal models. The resulting algorithms have been implemented and demonstrated on different polygonal and spline models. Our system is well designed and suitable for parallel implementation, on shared memory multiprocessing environments. As part of future work, we are trying to make the merging operation and the swap diagonal operation, in the simplification process more accurate and bound the error on merging operations. There are many open areas of future work. We will like to extend the algorithm to handle trimmed spline patches as well as models where complete adjacency information is not available. Furthermore, it may be useful to extend the topological simplification algorithm proposed in [EM98b] to curved models.

References

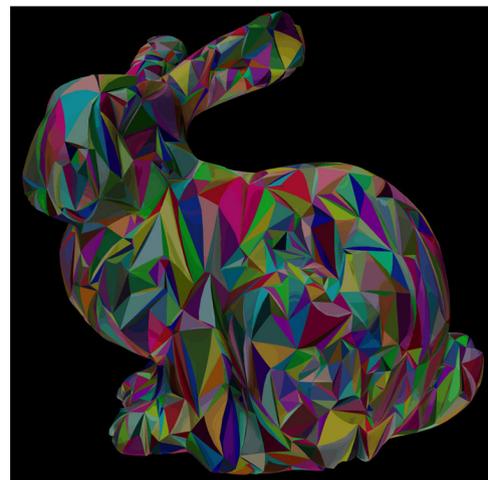
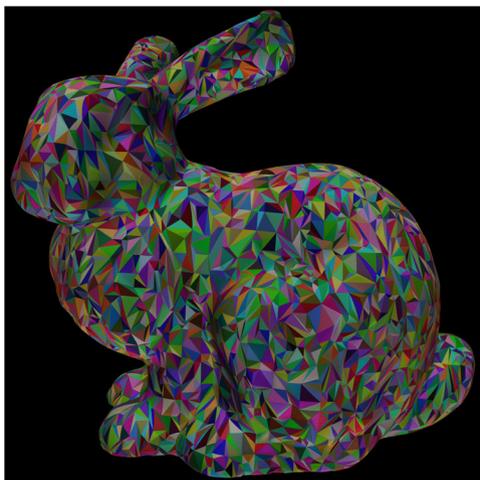
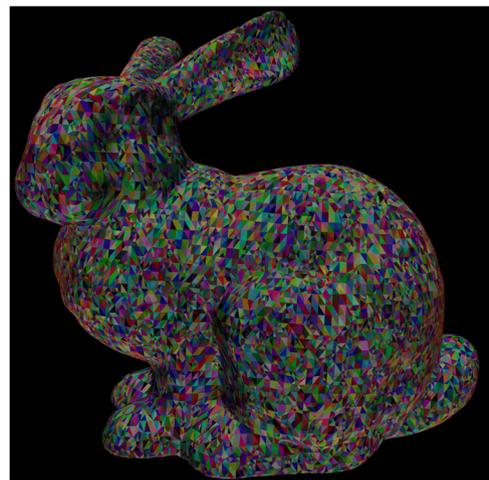
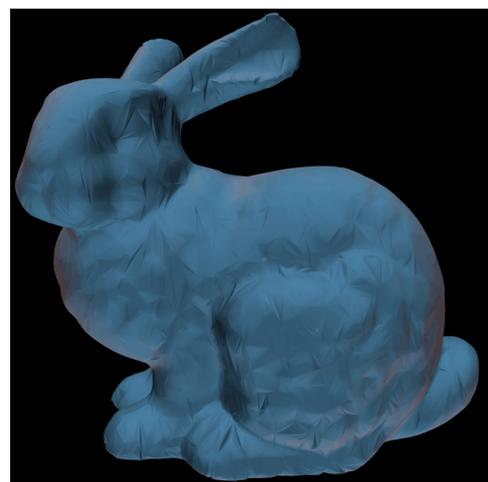
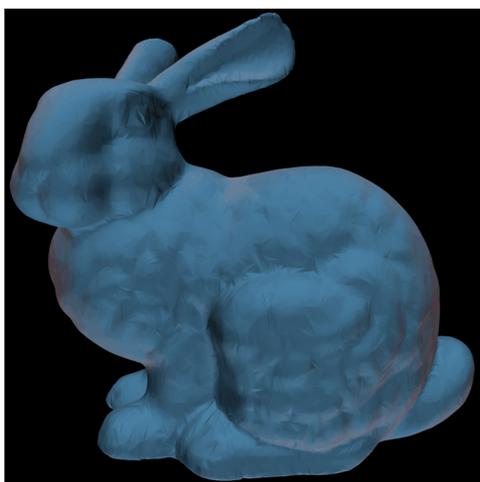
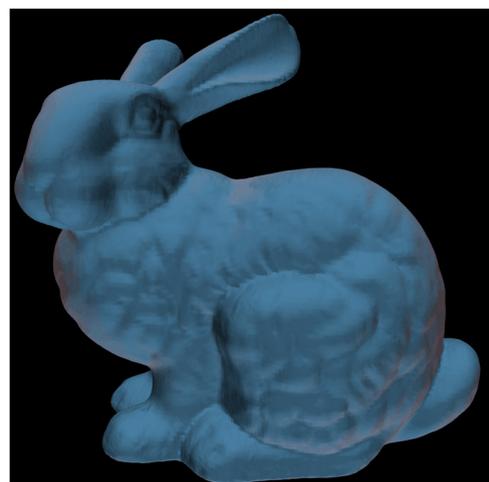
- [AES94] S.S. Abi-Ezzi and S. Subramaniam. Fast dynamic tessellation of trimmed nurbs surfaces. *Computer Graphics Forum*, 13(3):107–26, 1994. Proc. of Eurographics'94.
- [BS96] C. Bajaj and D. Schikore. Error-bounded reduction of triangle meshes with multivariate data. *SPIE*, 2656:34–45, 1996.

- [CMO97] J. Cohen, D. Manocha, and M. Olano. Simplifying polygonal models using successive mappings. In *Proc. of IEEE Visualization*, pages 395–402, 1997.
- [COM98] J. Cohen, M. Olano, and D. Manocha. Appearance preserving simplification. In *Proc. of ACM SIGGRAPH*, pages 115–122, 1998.
- [CVMe96] J. Cohen, A. Varshney, D. Manocha, and G. Turk et al. Simplification envelopes. In *Proc. of ACM Siggraph '96*, pages 119–128, 1996.
- [dB74] C. de Boor. Good approximation by splines with variable knots-II. In G. Watson, editor, *Numerical Solutions of Differential Equations*, pages 12–20. Springer-Verlag, New York, 1974.
- [dBF73] C. de Boor and G. Fix. Spline approximation by quasi-interpolants. *J Approx. Theory*, 8:19–45, 1973.
- [Die93] P. Dierckx. *Curve and Surface Fitting with Splines*. Oxford University Press, New York, 1993.
- [DLW93] T. Derose, M. Lounsbery, and J. Warren. Multiresolution analysis for surfaces of arbitrary topology type. Technical Report TR 93-10-05, Department of Computer Science, University of Washington, 1993.
- [EDD⁺95] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proc. of ACM Siggraph*, pages 173–182, 1995.
- [EH96] Matthias Eck and Hugues Hoppe. Automatic reconstruction of B-Spline surfaces of arbitrary topological type. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 325–334. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.
- [EM98a] C. Erikson and D. Manocha. Gaps: General and automatic polygon simplification. Technical Report TR98-033, Department of Computer Science, University of North Carolina, 1998. To appear in the Proc. of ACM Symposium on Interactive 3D Graphics, 1999.

- [EM98b] C. Erikson and D. Manocha. Simplification culling of static and dynamic scene graphs. Technical Report TR98-009, Department of Computer Science, University of North Carolina, 1998.
- [ESV97] J. El-Sana and A. Varshney. Controlled simplification of genus for polygonal models. *Proc. of IEEE Visualization*, pages 403–410, 1997.
- [Far93] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press Inc., 1993.
- [FMM86] D. Filip, R. Magedson, and R. Markot. Surface algorithms using bounds on derivatives. *CAGD*, 3:295–311, 1986.
- [GH97] M. Garland and P. Heckbert. Surface simplification using quadric error bounds. *Proc. of ACM SIGGRAPH*, pages 209–216, 1997.
- [GM98] M. Gopi and D. Manocha. A unified approach for simplifying polygonal and spline models. In *Proc. of IEEE Visualization*, pages 271–278, 1998.
- [Gue95] A. Gueziec. Surface simplification with variable tolerance. In *Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, pages 132–139, November 1995.
- [HDD⁺93] H. Hoppe, T. Derose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proc. of ACM Siggraph*, pages 19–26, 1993.
- [HDD⁺94] Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. Piecewise smooth surface reconstruction. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 295–302. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.
- [Hop96] Hugues Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996.
- [Hop97] Hugues Hoppe. View dependent refinement of progressive meshes. In *SIGGRAPH 97 Conference Proceedings*, pages 189–198. ACM SIGGRAPH, 1997.

- [KML95] S. Kumar, D. Manocha, and A. Lastra. Interactive display of large scale nurbs models. In *Proc. of ACM Interactive 3D Graphics Conference*, pages 51–58, 1995.
- [Kum96] S. Kumar. *Interactive Rendering of Parametric Spline Surfaces*. PhD thesis, Department of Computer Science, University of N. Carolina at Chapel Hill, 1996.
- [LD90] Charles Loop and Tony DeRose. Generalized B-spline surfaces of arbitrary topology. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 347–356, August 1990.
- [Loo94] Charles Loop. Smooth spline surfaces over irregular meshes. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 303–310. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.
- [Man97] S. Mann. Surface interpolation with triangular patches. In *Fifth SIAM Conference on Geometric Design*, 1997.
- [Pet95] J. Peters. C1 surface splines. *Siam J. of Numerical Analysis*, 32(2):645–666, 1995.
- [RB93] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering. In *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, June–July 1993.
- [RHD89] A. Rockwood, K. Heaton, and T. Davis. Real-time rendering of trimmed surfaces. In *Proceedings of ACM Siggraph*, pages 107–117, 1989.
- [S. 97] S. Kumar and D. Manocha and H. Zhang and K. Hoff. Accelerated walkthrough of large spline models. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 91–102, 1997.
- [Sar90] Ramon F. Sarraga. Computer modeling of surfaces with arbitrary shapes. *IEEE Computer Graphics and Applications*, 10(2):67–77, March 1990.

- [SC88] M. Shantz and S. Chang. Rendering trimmed nurbs with adaptive forward differencing. In *Proceedings of ACM Siggraph*, pages 189–198, 1988.
- [Sch97] W. Schroeder. A topology modifying progressive decimation algorithm. In *Proceedings of Visualization'97*, pages 205–212, 1997.
- [SZL92] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. In *Proc. of ACM Siggraph*, pages 65–70, 1992.
- [Tur92] G. Turk. Re-tiling polygonal surfaces. In *Proc. of ACM Siggraph*, pages 55–64, 1992.
- [XESV97] J. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):171–183, June 1997.



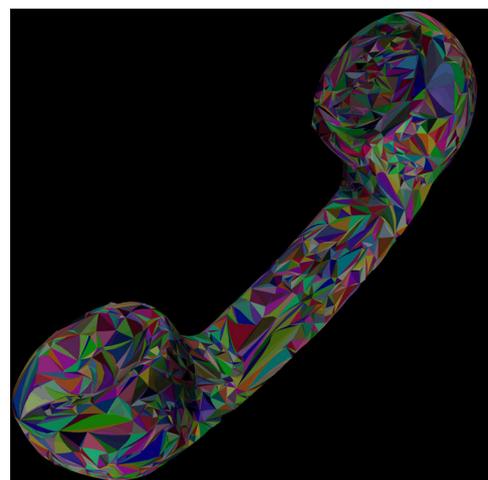
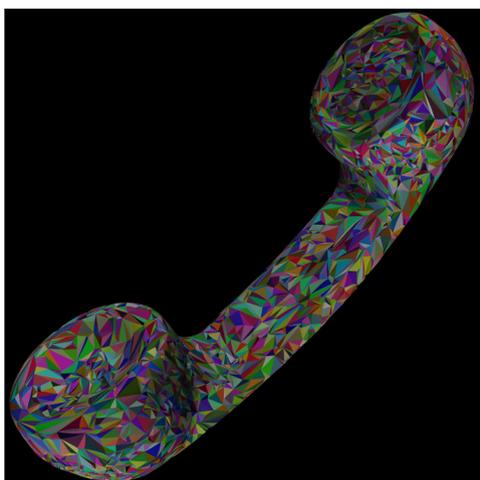
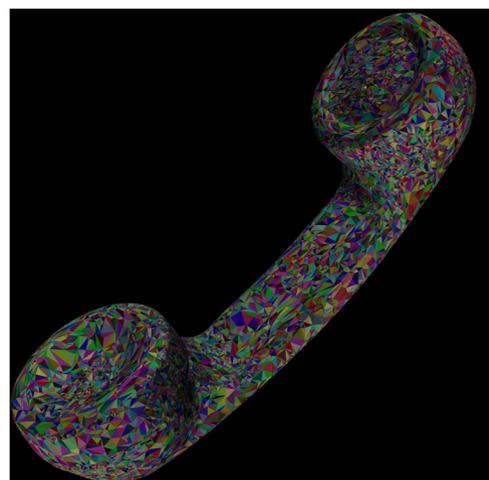
BUNNY MODEL

Top Row : Left : 69451 patches, Middle : 18356 patches, Right : 8038 patches
Bottom Row : Left : 36105 patches, Middle : 10448 patches, Right : 2909 patches



PHONE MODEL

Left : 43537 patches, Middle : 14740 patches, Right : 7996 patches



PHONE MODEL

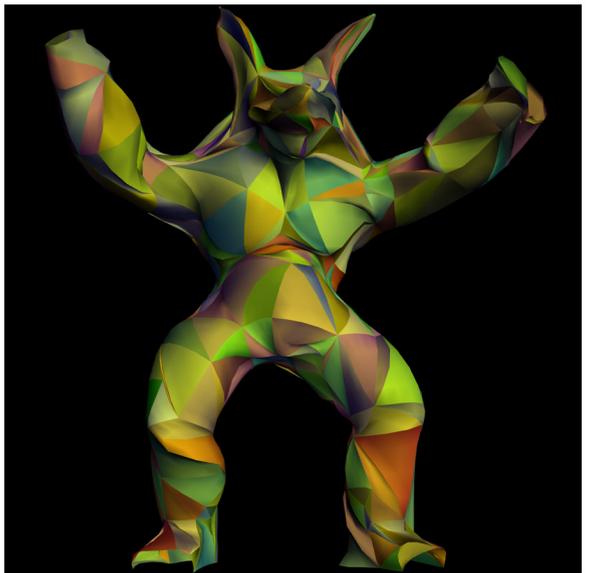
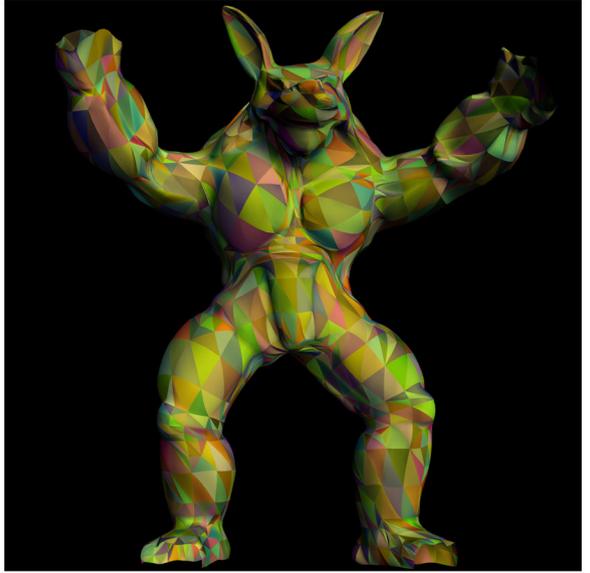
Left : 24627 patches, Middle : 9050 patches, Right : 4547 patches



LION MODEL

Top Row : Left : 4632 patches, Middle : 2356 patches, Right : 1342 patches

Bottom Row : Left : 3187 patches, Middle : 1779 patches, Right : 810 patches



ARMADILLO MODEL

Left Column : Top : 5100 patches; Middle : 1589 patches; Bottom : 836 patches.

Right Column : Top : 2233 patches; Middle : 1159 patches; Bottom : 450 patches.