To appear in Graphical Models and Image Processing, 1998.

Control of Polygonal Mesh Resolution for 3-D Computer Vision

Andrew E. Johnson

Jet Propulsion Laboratory
Mail Stop 107-102
4800 Oak Grove Drive
Pasadena, CA 91109
aej@robotics.jpl.nasa.gov

Martial Hebert

The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
hebert@ri.cmu.edu

Abstract

A common representation in 3-D computer vision is the polygonal surface mesh because meshes can model objects of arbitrary shape and are easily constructed from sensed 3-D data. The resolution of a surface mesh is the overall spacing between vertices that comprise the mesh. Because sensed 3-D points are often unevenly distributed, the resolution of a surface mesh is often poorly defined. We present an algorithm that transforms a mesh with an uneven spacing between vertices into a mesh with a more even spacing between vertices, thus improving its definition of resolution. In addition, we show how the algorithm can be used to control the resolution of surface meshes, making them amenable to multiresolution approaches in computer vision.

The structure of our algorithm is modeled on iterative mesh simplification algorithms common in computer graphics; however, the individual steps in our algorithm are designed specifically to control mesh resolution. An even spacing between vertices is generated by applying a sequence of local edge operations that promote uniform edge lengths while preserving mesh shape. To account for polyhedral objects, we introduce an accurate shape change measure that permits edge operations along sharp creases. By locally bounding the total change in mesh shape, drastic changes in global shape are prevented. We show results from many 3-D sensing domains including computed tomography, range imaging, and digital elevation map construction.

KEYWORDS: polygonal mesh, surface simplification, multiresolution modeling, shape approximation,3-D computer vision.

1 Introduction

Polygonal meshes are common representations in computer graphics because they can represent surfaces of almost any topology and complexity. By definition, a polygonal mesh is "a collection of edges, vertices and polygons connected such that each edge is shared by at most two polygons." (Foley et al.[5]) The recent growth in the availability of reliable 3-D sensors and sensing algorithms has made 3-D data much more common in computer vision research and algorithms. Because polygonal meshes are general representation for describing 3-D data, they are becoming as common in computer vision as they are in computer graphics. However, the uses of polygonal meshes by computer vision researchers may be quite different from those of computer graphics researchers. For instance, a graphics researcher may be interested in rendering a complex object represented as a polygonal mesh quickly and accurately, while a computer vision researcher may be interested in aligning two 3-D views of an object represented as polygonal meshes. In each field, the uses of polygonal meshes drives the operations that are commonly applied to them. Since the use of polygonal meshes is concentrated in computer graphics, the tools for manipulating polygonal meshes are tailored to the needs of computer graphics researchers. There is a conspicuous absence of tools for operating on polygonal meshes that are designed to aid computer vision researchers.

A particular operation that is necessary for computer graphics and computer vision is the control of resolution of polygonal meshes. The resolution of a polygonal mesh determines the amount of surface detail the mesh contains and is closely related to the number of vertices, edges and faces in the mesh. A coarse resolution mesh will contain a small number of vertices while a fine resolution mesh will contain a large number of vertices. However, because computer graphics is primarily concerned with accurately rendering objects, and computer vision is primarily concerned with measuring properties of sensed objects, the way in which resolution is controlled for computer graphics and computer vision will be different.

In computer graphics the control of mesh resolution is termed mesh simplification or mesh decimation. The main motivation for mesh simplification is to reduce the number of faces describing an object, while preserving the shape of the object, so that the object can be rendered as fast as possible. Shape preservation and face reduction are the two competing forces behind mesh simplifica-

tion. Some applications of mesh simplification include: removal of coplanar and adjacent faces from existing models [10] [14], creation of mesh hierarchies for level-of-detail rendering [3], and geometric compression [12] for storage and transmission of polygonal mesh models. Heckbert and Garland [9] have written a comprehensive overview of mesh simplification algorithms for computer graphics.

In contrast, an algorithm that controls the resolution of a mesh for computer vision applications should, in addition to reducing the number of faces while preserving the shape of the object, distribute the vertices of the mesh evenly over the surface of the object. The reason for this is as follows. A common operation in computer vision is to measure local properties of data, for example, calculating the gradient in a 2-D image at every pixel in the image or calculating the principal curvatures at every pixel in a range image. In these examples, an image is given, so "local" is determined by the parameterization of the image (i.e., adjacency of pixels). Unfortunately, for a polygonal mesh, locality is not well defined because a unique parameterization of the surface data does not necessarily exist. Examples of situations where well defined locality is needed in 3-D computer vision are point based registration of surface meshes [3][13], establishment of control points for computation of spline surfaces on a surface mesh [15], clustering of points for segmentation of range images [6], and calculation of surface normal [20].

A computationally efficient way to define locality is through the connectivity of vertices in the mesh. Vertices that are connected by an edge are local; however, this definition is problematic when the distance between connected vertices varies drastically, because the notion of locality cannot be associated with a fixed distance between vertices. Therefore, for local measurements on a polygonal surface mesh using connectivity to be meaningful, a method for normalizing the distance between connected vertices must be developed. Given such a method, by choosing the normalization distance appropriately, a mesh of any resolution can be generated. To our knowledge, no mesh simplification algorithms to date have been designed with the even distribution of vertices in mind. In this paper, we present a mesh simplification algorithm designed to normalize the distance between vertices on a surface mesh while preserving object shape. By controlling the normalization distance, this algorithm can produce meshes of arbitrary resolution facilitating the creating of mesh hierarchies useful in coarse-to-fine approaches to 3-D computer vision.

An example of application of our algorithm to a surface mesh generated from CT contours of a pel-

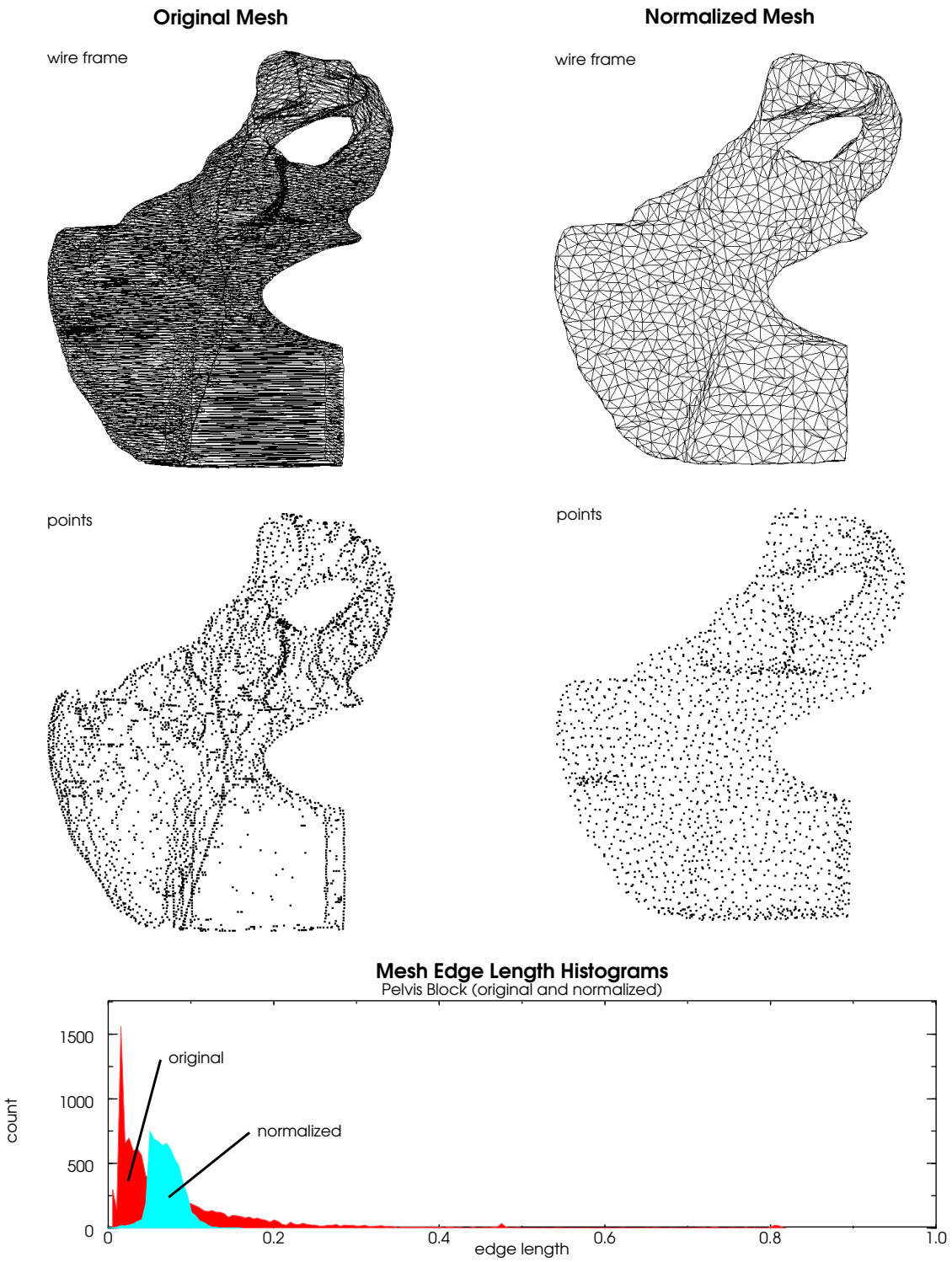

Figure 1: Demonstration of control of resolution. An initial mesh generated from CT contours of a pelvis bone phantom with holding block creates a mesh with very long edges and very short edges. Our algorithm generates a normalized mesh with edge lengths that are centered around the desired resolution. The histogram of edge lengths shows that the normalized mesh has a much smaller spread in edge lengths than the original mesh.

vis bone phantom with holding block, is shown in Figure 1. The lengths of the edges in the original mesh are widely distributed, but after application of our algorithm, the edges are compactly centered around the desired resolution. This is shown qualitatively through views before and after normalization, and quantitatively through histograms of edge lengths.

In addition to normalizing the distance between vertices, there are other requirements for any algorithm designed to control resolution of polygonal meshes used in computer vision. Below is a list of all of the necessary requirements:

- **Preserve shape**

  In general, the shape of the objects imaged is the property that is measured and compared in 3-D computer vision. Therefore, our algorithm must preserve shape if it is to produce meshes usable by 3-D computer vision algorithms.

- **Normalize distances between vertices**

  As stated above, our algorithm needs to make the distances between vertices regular so that mesh connectivity can be used to define the local neighborhood of a vertex. Normalizing distances also ensures that vertices are regularly distributed over the surface of mesh, making it possible to compute local properties evenly over the mesh. Furthermore, by normalizing the distance between vertices to a particular value, the resolution of the mesh can be controlled and hierarchies of meshes of increasing resolution can be generated.

- **Minimize number of vertices**

  In 3-D computer vision, the amount of computation is often proportional to the number of vertices or data points. Therefore, to minimize computation, our algorithm should minimize the number of vertices in the mesh while still meeting shape preserving and distance normalization requirements.

- **Handle free-form and polyhedral shapes**

  For our algorithm to be general, it must handle free-form (smooth, curved objects) and polyhedral objects because both forms are likely to imaged.

- **Handle mesh boundaries**

  The meshes being simplified in computer vision often have boundaries (e.g., a polygonal mesh generated from a single range image with range discontinuities) so our algorithm must be able to control the resolution of the mesh along the boundary of a mesh as well as in its interior.

Our algorithm meets all of these criteria given that the input surface mesh is simple (i.e., edges can be adjacent to no more than two faces) and contains only triangular faces. Most surface reconstruction algorithms create simple meshes, so the first condition is generally met in computer vision

contexts. If the second condition is not met, it is trivial to transform a mesh with non-triangular faces into one composed of triangular faces by breaking each planar polygonal face into multiple separate triangular faces using a constrained planar triangulation.

After describing some related algorithms that are representative of most mesh simplification algorithms developed to date, we will describe our mesh simplification algorithm. In particular, we will detail a new mesh edge-collapse criterion that preserves object shape while normalizing the lengths of edges in the mesh and a novel way of placing vertices after edge-collapse. We will also detail our method for propagation shape change errors; this method allows us to place a global bound on the maximum change in shape of a simplified mesh. We will follow our algorithmic discussion with a presentation of results from multiple 3-D sensing modalities including range images, marching cubes, computed tomography and digital elevation maps.

1.1 Related Work

Ideally, one would like to determine the mesh that contains the least amount of vertices and faces while conveying the shape of an object within some error bound of the original mesh. Unfortunately, the space of possible meshes is so large that searching for the globally best mesh is impractical. (In fact, the simpler problem of computing a minimal facet polyhedral terrain model is an NP-hard problem[1].) Therefore, most simplification algorithms to date are greedy, iterative algorithms that search for the best mesh by taking the current best step toward the global minimum. Like many mesh simplification algorithms, our algorithm is based on the iterative application of local mesh operations to transform the original mesh into a mesh meeting our simplification criteria. Some examples of mesh operations are edge-collapse, edge-split, edge flip and point removal followed by re-triangulation. There exist many published mesh simplification algorithms; below, we describe some representative algorithms. For a more comprehensive overview, see [9].

The general flow of iterative simplification algorithms is as follows: First, order mesh primitives (vertices, edges or faces) for simplification. Next, select the best mesh primitive for simplification and apply a mesh operation to that primitive. Then, update the shape of the mesh and reorder mesh primitives for simplification (if required). Repeat application of mesh operations until no more mesh primitives can be simplified. The differences in iterative simplification algorithms come from the primitives used in simplification, how the primitives are ordered for simplification, how

the mesh shape changes when a mesh operation is applied and what criteria are used to stop mesh simplification.

One of the first iterative mesh simplification algorithms was proposed by Schroeder et al. [17]. In this algorithm, vertices are the primitives used for decimation; they are removed from the mesh, and the local neighborhood surrounding the point is re-triangulated in the local plane of the vertex. A point is removed if its distance to the best fit plane of the surrounding points is small. The primitives are not ordered; all vertices with a planar fit error that are less than a threshold and meet topology preserving checks are removed. In a later version of the algorithm [18], the shape change in the mesh is limited by placing a global bound on the maximum allowable change in mesh shape. By using the point removal mesh operation, Schroeder's algorithm must shrink convex regions in the mesh and expand concave regions in the mesh. If the primitive with lowest planar fit error is removed at each iteration, the global change in shape will be kept as small as possible. In Schroeder's algorithm, primitives are not ordered for decimation, so a vertex with a greater planar fit error can be removed before a vertex with a lesser planar fit error. Therefore the global change in shape will not be kept as small as possible and the resulting reduction of number of points in the mesh will not be as great as it could have been had the vertices been ordered for decimation. Finally, the re-triangulation step in point removal is time consuming and complicated,

Guéziec's [8] mesh simplification algorithm improved on Schroeder's algorithm in many ways. Guéziec uses edges as the mesh primitive and edge-collapse to eliminate re-triangulation from the mesh simplification algorithm. The edges are ordered based on edge length and a single pass through the edges is performed. During edge-collapse, a new vertex is created. Guéziec intelligently places the new vertex in a position that preserves the volume of the object in the local neighborhood of the edge, preventing drastic changes in the shape of the object. Checks on topology preservation and creation of compact triangles are also used to preserve the shape of the mesh. Like our method, Guéziec's simplification method bounds the total change in mesh shape. However, mesh shape is bounded using a complex construction called a tolerance volume whose update requires a dynamic programming algorithm. The tolerance volume is a no less conservative than our shape change measure when bounding total change in surface shape, so its complex construction may not be justified. Furthermore, Guéziec's algorithm does not have any explicit control of the resolution of the meshes generated, and there is no explicit handling of the vertices along the

boundary of a mesh to prevent shrinking during simplification.

Hoppe et al. [11] use edge-collapse, edge-swap and edge-split to iteratively refine an initial mesh that is close to a set of 3-D data points. They use a global optimization procedure that attempts to find meshes that fit the input data and have a small number of vertices and edges that are not too long. They use a spatially random ordering of edges during simplification and perform a fixed number of iterations. Since they fit new vertex positions based on supplied data, the shape of the object is preserved. There algorithm can handle free-form as well as polyhedral objects. Their optimization procedure requires three nested loops and is subsequently quite slow, but it does produce very concise and accurate meshes. The inclusion of a term that penalizes long edges in the optimization is a step toward controlling the overall distribution of vertices in the surface mesh.

In later work[12], Hoppe developed a multiresolution mesh representation called a progressive mesh. This representation stores a mesh as a coarse mesh and a sequence of detail records that describe the order of edge split operations needed to refine the mesh. Progressive meshes not only refine surface shape but also scalar functions defined on the mesh surface. Progressive meshes are general multiresolution representations based on edge collapse and split operations whose sequence is determined by an algorithm similar to Hoppe et al. [11]. By combining our algorithm for sequencing edge collapse and split operations with the progressive mesh representation, our algorithm could generate multiresolution mesh representations. An issue addressed by our algorithm not addressed by Hoppe's algorithm is the bounding of global error. Hoppe's algorithm does not allow explicit control of edge length although his algorithm could be easily modified to do so by imposing a non-zero offset on the spring error term during mesh simplification.

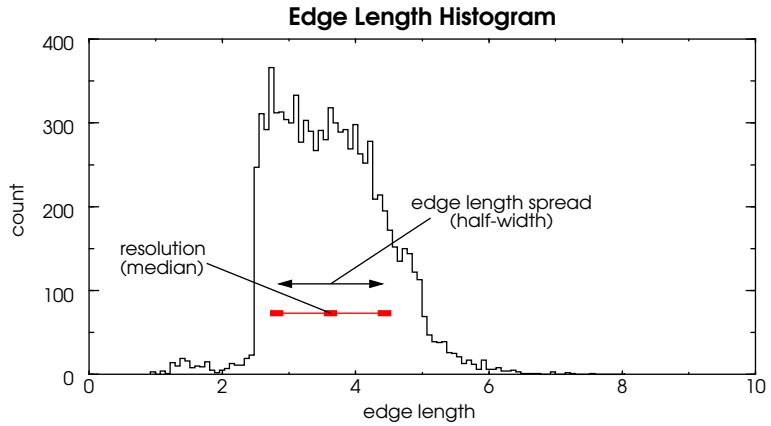

Figure 2: Histogram of edge lengths. The resolution of a mesh is the median of its edge length histogram and the edge length spread is the half-width (upper quartile minus lower quartile) of the histogram.

The issue of bounding global mesh error has been addressed by Cohen et al. [2] with their simplification envelopes representation. Simplification envelopes are surfaces that bound the absolute change in shape of a mesh during simplification. By applying local and global mesh operations their algorithm is able to produce surfaces guaranteed to remain within the simplification envelopes, while reducing the vertex count of the surface. Furthermore, their algorithm supports error bounds that vary over the surface which allows for adaptive refinement. However, because their algorithm produces simplified meshes which are a subset of vertices from the original mesh, and mesh operations are not ordered based on minimal change in mesh shape the simplified meshes will contain more vertices than meshes generated using other algorithms. Consequently, although their algorithm uses a less conservative error bound than ours, its benefit may be negated.

Garland and Heckbert [7] have developed an efficient surface simplification algorithm based on quadric error metrics. Their algorithm rapidly produces simplified meshes that can change topology to increase simplification. Like our algorithm, their algorithm uses edge collapse operations which are ordered based on applying the operation that induces minimal change in mesh shape. However, to speed computation, they use the average distance instead of the maximum distance between meshes and they do not address the issue of bounding total change in mesh shape. Furthermore, their algorithm has no way to explicitly control the length of edges in the mesh.

There exist some alternative particle-based approaches to generating a regular sampling of points on a surface. Turk [22] presents an algorithm that simplifies an initial mesh as follows. A small number of points are placed on the surface mesh and then pushed around the faces of the surface mesh by repulsive forces until they are evenly distributed. Once distributed, the points are added to the initial mesh and then all of the vertices in the initial mesh are iteratively removed. The end result is an even sampling of the original surface mesh. This algorithm will work best for smoothly varying surfaces and is quite complicated. Witkin and Heckbert [25] present an algorithm for obtaining a regular sampling of an implicit surface using a particle-based approach. They are able to control the sampling of an implicit surface by changing the properties of the forces controlling the particles and can maintain a regular sampling even while the surface is rapidly changing. However, their algorithm relies on an underlying implicit surface and does not address the issue of mesh generation. Shimada [19] presents an algorithm for controlling the sampling of 2-D and 3-D meshes using a physically-based triangulation method called the bubble mesh. His algorithm creates uni-

formly spaced vertices, but, when triangulating 3-D surfaces, requires that the surface have a 2-D parameterization. In the case of general surfaces, this requirement cannot be guaranteed.

2 Algorithm for Control of Mesh Resolution

For an arbitrarily shaped object it is impossible to make the distance between vertices exactly the same for all vertices in the mesh while still adequately describing the shape of the object. Therefore, we quantify the spacing between vertices using local and global statistics on the histogram of lengths of edges in the mesh. An example of an edge length histogram is given in Figure 2. We define mesh *resolution* to be the median of the lengths of all of the edges in the mesh, and we define the *edge length spread* to be the upper quartile of edge lengths minus the lower quartile of edge lengths (i.e., the half width) in the edge length histogram. Given these definitions, the goal of our algorithm is to adjust the resolution of the original mesh to a desired resolution while minimizing the edge length spread of the histogram. We call this process *length normalization*.

An additional constraint on length normalization is that the original shape of the object must be preserved; we assume that the original shape of the object is given by the mesh input into the algorithm. In our algorithm, two operations are iteratively applied to edges in the mesh to obtain the mesh of the desired resolution: edge-split is used to remove long edges, while edge-collapse is used to remove short edges. During edge-split, an edge is broken at its midpoint into two edges; the edge-split operation does not change the shape of the mesh. During edge-collapse, an edge is reduced to a point, so the local shape of the mesh is modified. In our algorithm, the position of the point resulting from edge-collapse is chosen to preserve the shape of the object, but some change in shape is unavoidable when edges are removed from the mesh. However, the change in shape of the mesh can be minimized at each iteration by intelligently ordering the edges for operation. More specifically, the edges in the mesh are ordered for operation by measuring the change in shape that results from application of each edge operation (*shape change measure*). Using this ordering, the edges that change the shape of the mesh the least are applied first, thus minimizing the change in shape of the mesh at each iteration. This best-first approach produces meshes that preserve shape better during length normalization than would an algorithm that chooses edges randomly.

To strike a balance between preserving shape and normalizing the lengths of the edges in the mesh,

the order of application of edge operations is also made a function of the length of the edge being operated on. More specifically, an *edge length weight* is computed for each edge; the order in which an edge is operated on is then determined by the product of its edge length weight and its shape change measure. For example, with this ordering, given two edges with the same shape change measure whose lengths are shorter than the desired resolution, the shorter edge will be collapsed first.

We prevent the shape of the mesh from changing too much by placing a bound on the maximum allowable change in mesh shape. Each time an edge operation is applied, the edge's shape change measure is added to the *accumulated shape change* (the accumulation of the change in shape that the edge has undergone in previous iterations) of all of the edges in the neighborhood of the edge. If the accumulated shape change of an edge is made greater than the maximum allowable change in shape, the edge is removed from consideration. This ensures that over the entire surface of the mesh, the change in shape remains below a user-defined bound. Furthermore, if the length of an edge is within some user defined bounds of the desired mesh resolution, the edge will not be decimated. The limit on the maximum allowable shape change and the length of edges being within some bounds of the desired resolution will eventually prevent all edges in the mesh from being decimated. This constitutes the stopping criterion for the algorithm. Since our algorithm attempts to normalize lengths while preserving shape, most, but not all, of the edge lengths will be within the desired bounds.

2.1 Definitions

Before describing the mesh normalization algorithm in detail, some definitions need to be established. Edge-collapse and edge-split operate on local neighborhoods in the mesh. The exact definitions of the local neighborhoods of an edge and vertex are as follows. Let the *EdgeStar(e)* be the local neighborhood in the mesh affected by the edge-collapse operation on an edge e. *EdgeStar(e)* contains all of the faces that contain at east one of the vertices making edge e, as well as the edges and vertices that make these faces. Let the *VertexStar(v)* be the local mesh neighborhood of the vertex v created when an edge-collapse operation is applied to an edge. It contains all of the faces that contain the vertex v along with the edges and vertices making up these faces. An illustration of *EdgeStar(e)* and *VertexStar(v)* are given in Figure 3. Let the *EdgeDiamond(e)* be the local neighborhood in the mesh effected by the edge-split operation on an edge e. It contains the faces that are

adjacent to the edge e. The edges and vertices that make these faces are also in *EdgeDiamond(e)*. Let the *VertexDiamond(v)* be the local mesh neighborhood of the vertex v created when an edge-split operation is applied to an edge. It contains the four faces that contain the vertex v along with the edges and vertices making up these faces. Illustrations of *EdgeDiamond(e)* and *VertexDiamond(v)* are given in Figure 3.

2.2 Overview of Algorithm

Input into length normalization is a desired resolution L_0 and the acceptable deviation in length L_D from the desired resolution for edges in the normalized mesh. The upper and lower bounds on edge lengths are then

$$L_{MIN} = L_0 - \frac{L_D}{2} \qquad L_{MAX} = L_0 + \frac{L_D}{2} \tag{1}$$

In addition, the maximum allowable change in shape C_{MAX} is input to the algorithm.

Given in detail, our algorithm is as follows: First, a priority queue (a dynamically ordered queue) is created from all the edges in the mesh whose lengths are not within the bounds (L_{MIN}, L_{MAX}). The position of an edge in the priority queue is the product of a edge length weight and the shape change measure of the edge. Next, the first edge in the priority queue is popped off the queue and operated on. If the length of the edge is greater than L_{MAX}, the edge is split at its midpoint. This split changes the neighborhood of the edge by adding an edge, a vertex and two new faces. If the edge length is less than L_{MIN}, the edge is collapsed into a point, changing the neighborhood of the

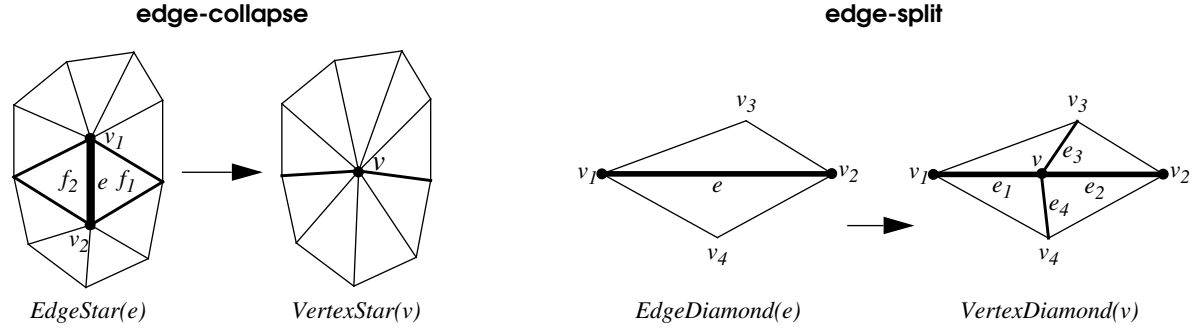

Figure 3: (Left) The effect of edge-collapse on the local neighborhood of a mesh. The edge e is collapsed to a vertex v, eliminating edge e, faces f_1 and f_2, and vertices v_1 and v_2. The local neighborhood of e is termed *EdgeStar(e)* and the local neighborhood of the vertex v is termed *VertexStar(v)*. (Right) The effect of edge-split on the local neighborhood of a mesh. The edge e is split at a vertex v, adding three new edges and two new faces to the mesh. The local neighborhood of e during edge-split is termed *EdgeDiamond(e)* and the local neighborhood of the vertex v after edge-split is termed *VertexDiamond(v)*.

edge by eliminating a vertex, two faces and an edge. When an edge is collapsed, its shape change measure is added to the accumulated shape change of the edges in the new neighborhood of the edge. After the operation is applied, the edges in the old neighborhood of the edge are removed from the priority queue Then, the edges in the new neighborhood of the mesh are added to the priority queue if they meet the following criteria: their lengths are outside the edge length bounds L_{MAX} and L_{MIN}; their accumulated shape change is not greater than C_{MAX}; and they meet additional checks that prevent changes in topology and shrinkage of the mesh boundary. Edges are iteratively popped off the queue and operated on until no more edges exist in the queue. A pseudo-code description of the length normalization algorithm is given in Figure 4.

2.3 Shape Change Measure

The shape change measure of an edge is defined as the distance between the current mesh and the

```
NormalizeLengths(LMIN,LMAX,CMAX,MESH)
    // Initialize edge priority queue
    PQ = InitializePriorityQueue()
    For all edges e in MESH
        C = ShapeChangeMeasure(e,MESH)
        W = EdgeLengthWeight(LMIN,LMAX,e,MESH)
        AccumulateShapeChange(e) = 0
        If (CanOperateOn(e,LMIN,LMAX,CMAX,MESH))
            InsertEdgeInPriorityQueue(e,W*C,PQ)
    // Normalize mesh
    While (!Empty(PQ))
        edge e = PopPriorityQueue(PQ)
        If (Length(e)>LMAX)
            For all edges oe in EdgeDiamond(e)
                RemoveEdgeFromPriorityQueue(oe,PQ)
            vertex v = SplitEdge(e,MESH)
            For all edges ne in VertexDiamond(v)
                C = ShapeChangeMeasure(ne,MESH)+AccumulateShapeChange(ne)
                W = EdgeLengthWeight(LMIN,LMAX,ne,MESH)
                If (CanOperateOn(ne,LMIN,LMAX,CMAX,MESH))
                    InsertEdgeInPriorityQueue(ne,W*C,PQ)
        If (Length(e)<LMIN
            For all edges oe in EdgeStar(e)
                RemoveEdgeFromPriorityQueue(oe,PQ)
            vertex v = CollapseEdge(e,MESH)
            For all edges ne in VertexStar(v)
                AccumulateShapeChange(ne) = AccumulateShapeChange(ne) +ShapeChange(e)
                C = ShapeChangeMeasure(ne,MESH)+AccumulateShapeChange(ne)
                W = EdgeLengthWeight(LMIN,LMAX,ne,MESH)
                If (CanOperateOn(ne,LMIN,LMAX,CMAX,MESH))
                    InsertEdgeInPriorityQueue(ne,W*C,PQ)
```

Figure 4: Pseudo-code description of length normalization algorithm.

mesh that results from operating on the edge. During length normalization, we want to place a bound on the maximum change in shape of the mesh. Therefore, our shape change measure is defined as the maximum distance between meshes before and after an edge operation is applied. In brief, shape change measure is a function of the maximum distance between the vertices in one mesh and their closest points on the faces of the other mesh. Since edge operations effect only a local neighborhood of the edge, the distance between meshes can be measured by only comparing the local mesh neighborhoods before and after application of the operation.

We consider mesh shape to be conveyed by the faces of the mesh (not just the vertices), so an accurate measure of distance between meshes must consider distance between mesh faces. We define the asymmetric distance between a mesh M_1 and a mesh M_2 to be the maximum of the distance between any point on M_1 and its associated closest point on M_2. Because meshes are composed of subsets of linear elements (points, lines and planes), the maximum distance between M_1 and M_2 will occur between a vertex of M_1 and a face of M_2. Therefore, the distance between M_1 and M_2 can be defined as the maximum Euclidean distance between a vertex v_i of M_1 and its closest point, $v_{closest}$, on the closest face f_j of M_2.

$$d(M_1, M_2) = \max_{v_i \in M_1} \left(\min_{f_j \in M_2} \left\| v_i - v_{closest}(v_i, f_j) \right\| \right) \tag{2}$$

The closest point on a triangle to a point in space is computed by first projecting the point along the triangle normal onto the plane defined by the triangle. If the projected point lies inside the triangle, then it is the closest point. Otherwise the point is projected perpendicularly onto the lines that determine the edges of the triangle. If the point projects onto the lines between two vertices of the triangle then this is the closest point. Otherwise, the closest point is one of the vertices of the triangle.

The distance $d(M_1,M_2)$ is not symmetric, so we define a distance metric between two meshes $D(M_1,M_2)$ to be the maximum of $d(M_1,M_2)$ and $d(M_2,M_1)$.

$$D(M_1, M_2) = max(d(M_1, M_2), d(M_2, M_1)) \tag{3}$$

In our length normalization algorithm, we use $D(M_1,M_2)$ as our shape change measure. Intuitively, the shape change measure will be zero when the surfaces described by the faces of the mesh coincide, even when the faces, edges and vertices of the two meshes are not exactly the same. Using

the maximum distance between meshes as our shape change measure gives our algorithm the ability to operate on edges along surface shape discontinuities, like ridges and corners, as long as the distance between meshes remains small after operation. Operating on ridges and corners is not possible with some mesh simplification algorithms because the shape change measures used (e.g., distance to local tangent plane in [17]) are over cautious and prevent simplification along surface shape discontinuities even when simplification will not change the shape of the mesh.

In the general case of comparing two meshes, computing the $D(M_1,M_2)$ is computationally expensive. However, as will be shown in the next section, computing the shape change measure between meshes before and after application of an edge operation is computationally feasible since there the change in mesh shape is restricted to a local neighborhood of the mesh.

2.4 Edge Operations

The first edge operation we will consider is edge-collapse. As shown in Figure 3, the effect of the edge-collapse operation is to shrink an edge in the mesh to a point thereby, removing the edge and its two adjacent faces from the mesh. Edge-collapse can be performed quickly through local mesh operations that remove the affected faces, edges and vertices from our surface mesh data structure and then update the pointers between adjacent faces, vertices and edges.

An important variable in the edge-collapse operation is the position of the new vertex that results from edge-collapse; the position of the other vertices in *EdgeStar(e)* remain fixed during edge-collapse. A simple method for positioning the vertex would be to place the vertex at the midpoint of the collapsed edge. However, as shown in 2-D in Figure 5, this simple placement of the new vertex keeps the vertex on the surface mesh, but can cause excessive shape change (shrinkage or expansion) in areas of high curvature in the mesh. Instead, we allow the new vertex to be placed off of the edge, in order to reduce the shape change measure of the edge-collapse. In particular, the position of the new vertex $\boldsymbol{v}$ is the average of the projection of the midpoint of the edge $\boldsymbol{v}_m$ onto the N planes defined by the faces in *EdgeStar(e)*.

$$\boldsymbol{v} = \boldsymbol{v}_m - \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{n}_i\boldsymbol{v}_m + d_i)\boldsymbol{n}_i \qquad \boldsymbol{v}_m = \frac{\boldsymbol{v}_1 + \boldsymbol{v}_2}{2} \tag{4}$$

The planes in *EdgeStar(e)* are defined by their surface normal $\boldsymbol{n}_i$ and offset d_i. As shown in Figure 5 for a 2-D example, placing the new vertex based on projections onto the planes of the sur-

rounding faces prevents shrinkage of the mesh by distributing the change in shape above and below the collapsed edge. This is in contrast to placing the new vertex at the midpoint of the collapsed edge where the change in mesh shape is not balanced because only shrinkage occurs. Figure 6 shows the cumulative effect of our shape preserving placement of the vertex during edge-collapse versus the placement of the vertex at the midpoint of the collapsed edge. A surface mesh model of a femur bone generated from CT contours is shown in Figure 6. Two 2-D slices through the model are shown for the original mesh: a mesh normalized using shape preserving vertex placement and a mesh normalized using midpoint vertex placement. From the slices it is apparent that, with respect to midpoint placement, shape preserving placement reduces the shrinkage that can occur in areas of high curvature during length normalization.

The shape change measure for an edge that is going to be collapsed can be computed using just the mesh primitives in *EdgeStar(e)* and *VertexStar(v)*. After edge-collapse the vertices along the border of *VertexStar(v)* are the same as the vertices on the border of *EdgeStar(e)*. Therefore, the shape change measure can be calculated as the maximum of: d_e, the distance between v and its closest point on the faces of *EdgeStar(e)*; d_{v1}, the maximum distance between v_1 and its closest point on the faces of *VertexStar(v); and* d_{v2}, the maximum distance between v_2 and its closest point on the faces of *VertexStar(v).*

$$d(VertexStar(v), EdgeStar(e)) = max\left\{\begin{array}{l} d_e = \min\limits_{f_j \in EdgeStar(e)} \left\| v - v_{closest}(v, f_j) \right\| \\ d_{v1} = \min\limits_{f_j \in VertexStar(v)} \left\| v_1 - v_{closest}(v_1, f_j) \right\| \\ d_{v2} = \min\limits_{f_j \in VertexStar(v)} \left\| v_2 - v_{closest}(v_2, f_j) \right\| \end{array}\right\} \quad (5)$$

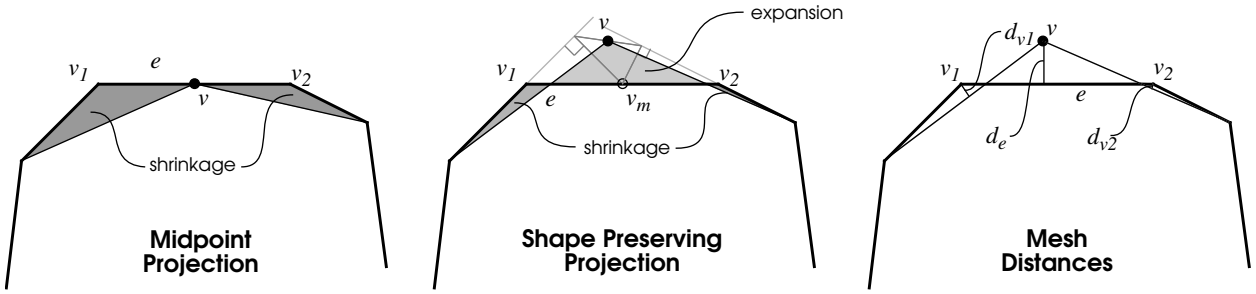

Figure 5: Placement of the new vertex generated during edge-collapse at the midpoint of the collapsing edge causes shrinkage of the mesh during normalizations (left). However, by placing the new vertex off of the edge during edge-collapse, shrinkage and expansion are combined to limit the shape change in the mesh (middle). The distances between meshes during edge collapse is the maximum of the distance between the new vertex v and the faces old mesh and the distances between the faces of the new mesh and the old vertices v_1 and v_2 (right).

A drawing of these distances for a 2-D example is shown on the right in Figure 5.

We use the edge-split operation to remove edges that are too long from the mesh. As shown Figure 3, the edge-split operation splits an edge at a vertex on the edge to produce three new edges, two new faces and a new vertex. The position of the new vertex is chosen as the midpoint of the edge being split. Since the mesh surface before and after edge-split is the same, the shape change measure for the edge-split operation is zero. Edge-split can be performed quickly through local mesh operations that add the new faces, edges and vertex to a surface mesh data structure and then update the pointers between adjacent faces, vertices and edges.

2.5 Accumulated Shape Change

Each time an edge is collapsed, the shape of the mesh changes slightly. The shape change measure we use is the maximum distance between the mesh before and after the edge was collapsed. We limit the amount of shape change that occurs during normalization by storing an *accumulated shape change* in mesh shape accrued so far by each edge during normalization. Initially each edge starts with zero accumulated shape change. When an edge *e* is collapsed, its shape change measure is added to the accumulated shape change of all of the edges in *VertexStar(v)*. By keeping track of the worst case change in mesh shape for each edge, we can limit the global maximum change in mesh shape. In other words, edges that have an accumulated shape change greater than a specified bound can be prevented from being collapsed. Note that the edge-split operation does not change

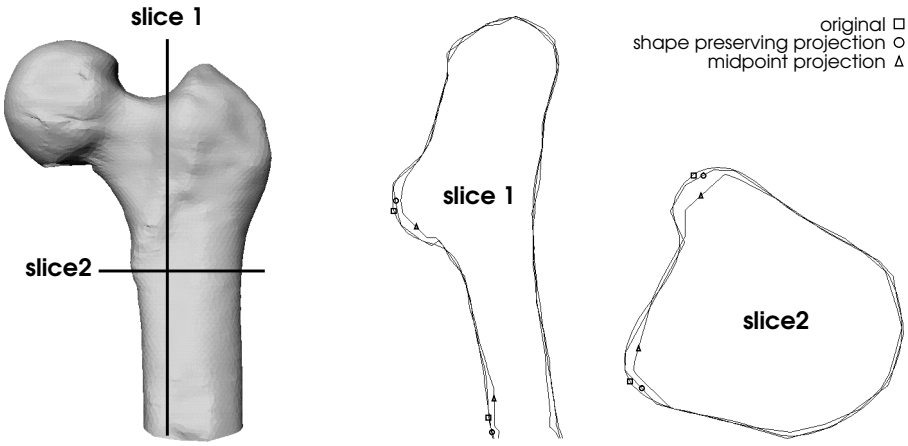

Figure 6: Illustration of the benefit of shape preservation vertex positioning during edge-collapse. Two 2-D slices through a femur model normalized using shape preserving and midpoint positioning show that midpoint positioning of vertices during edge-collapse shrinks the surface mesh in areas of high curvature while shape preserving positioning reduces shrinkage in high curvature areas. Shape preserving projection balances shrinkage and expansion to prevent excessive shape change.

the shape of the mesh, and it does not increase the accumulated shape change of the edges in the neighborhood of the edge.

The idea of accumulating shape change has been investigated by other authors. Schroeder [18], accumulates shape change by adding distances to local best fit planes after retriangulation. Distance to best fit plane is less accurate than distance between meshes, so his accumulation of total error is more conservative than ours. This will result in less mesh simplification at a given maximum error. Guéziec[8] limits total change in mesh shape using a construction called a tolerance volume. Inspection of the figures in his paper indicate that, in some cases, our method of measuring maximum distances between meshes is less conservative than limiting error using a tolerance volume. Cohen et al. [2] have developed a mesh simplification representation called a Simplification Envelope for limiting accumulation of error during simplification. Their measurement of accumulation of error is more exact and therefore less conservative than our method which adds maximum mesh distances. However, because their algorithm produces simplified meshes which are a subset of vertices from the original mesh, and mesh operations are not ordered based on minimal change in mesh shape, their simplified meshes will contain more vertices than meshes generated using other algorithms. Consequently, the benefit of a less conservative error bound may be negated.

The global bounds on accumulated shape change are illustrated in Figure 7. Placing a maximum

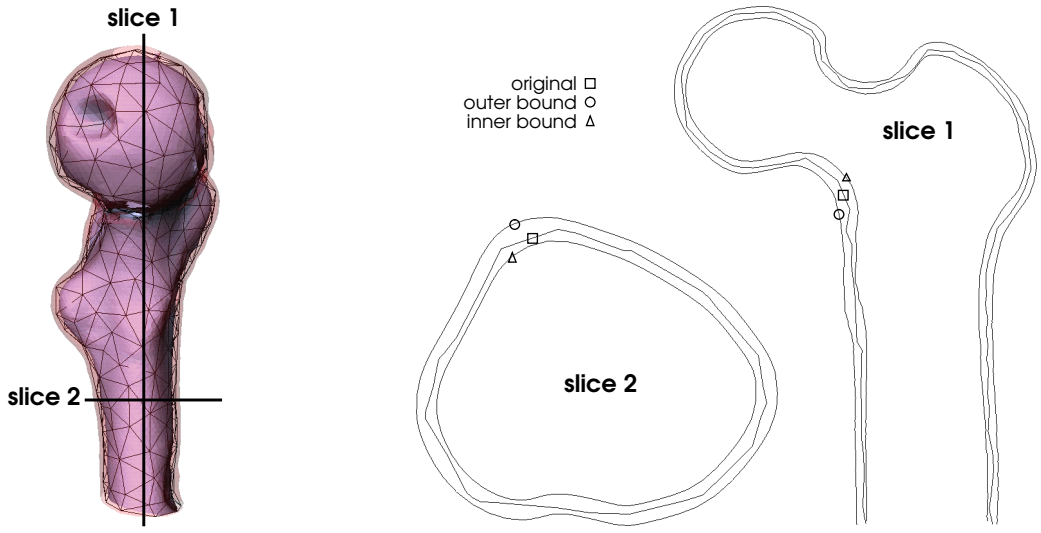

Figure 7: Visualization of global bounds on accumulated shape change for a model of femur. Normalization prevents excessive shape change by keeping the simplified surface mesh (wire frame, left) inside the inner and outer global error bound surfaces (shaded, transparent, left). Two 2-D slices through the normalized mesh and the inner and outer bound surfaces clearly show that the normalized mesh is within the error bound surfaces. NOTE: The bounding surfaces are for visualization only and are not used in the normalization algorithm.

bound on the total change in shape of the edges in the mesh can be visualized as two surfaces that contain the original mesh: an inner surface that bounds shrinkage and an outer surface that bounds expansion (Similar to simplification envelopes [2]). During normalization, the global bound on accumulated shape change prevents the normalized surface from moving outside of these bounding surfaces. In Figure 7 the bounding surfaces are expansions and contractions of the original surface mesh generated by projecting each vertex *v* in the original surface mesh along the surface normal of the best fit plane to the vertices in *VertexStar(v)*. The vertices are projected (out for outer bound and in for inner bound) a distance equal to the maximum allowable accumulated shape change C_{MAX}. Two 2-D slices through the normalized mesh and inner and out bounding surface clearly show that the normalized surface stays within its bounds. Note that these bounding surfaces are for visualization purposes only and do not enter into our algorithm.

2.6 Edge Ordering

During normalization we would like to operate on edges with lengths that are far from the desired resolution in order to reduce the edge length spread. We would also like to prevent operations on edges that have large accumulated shape change in order to prevent drastic changes in mesh shape. To implement these requirements, the edges in the mesh are ordered for operation by storing them in a priority queue. The order of an edge in the priority queue is determined by the product of the accumulated shape change *C* for the edge and an *edge length weight W* for the edge; edges with a small product *C*W* will be toward the top of the queue.

The edge length weight of an edge is generated from a Gaussian of edge length

$$W = \exp\left(-\frac{(l - L_0)}{L_D^{\,2}}\right) \tag{6}$$

where the length of the edge is l, the desired resolution is L_0, and the acceptable edge length spread is L_D. Using this edge length function will assign a small weight to edges that are much shorter or longer than the desired resolution. The accumulated shape change of an edge will be large for edges that have changed the shape of the mesh a great deal. By using the product of accumulated shape change and edge length weight, edges that are very short or very long and have not deviated from their original positions will be decimated before edges that are close to the desired resolution or that have deviated a great deal from their original position.

Using the edge length weight in addition to the accumulated shape change for ordering edges for operation is our main mechanism for generating length normalized meshes. If an edge is too long its edge length weight will be small and it will be split. If an edge is too short, its edge length weight will also be small and the edge will be collapsed. Therefore, over many iterations, the lengths of the edges in the mesh will be forced toward the desired resolution. By using accumulated shape change and not just the (immediate) shape change measure of an edge, edges that have been operated on a great deal (and have changed the shape of the mesh over many iterations) will be avoided. This has the effect of distributing the change in mesh shape over the entire surface of the mesh instead of concentrating the change in shape at specific places.

During normalization, edges are constantly removed and added to the priority queue. An edge can no longer be operated on, and hence will not be added to the priority queue, if its accumulated shape change exceeds the maximum allowable accumulated shape change C_{MAX} when operated on. Furthermore, an edge will not be added to the priority queue if its length is within the desired edge length bounds (L_{MIN}, L_{MAX}). These two conditions eventually cause the priority queue to become empty, so the iterations on the mesh edges must stop. Since some edges will achieve the accumulated shape change bound before their length is within the edge length bounds, not all the edges in the final mesh will have lengths inside of the bounds.

Initially, all edge-splits will be ordered before edge-collapse operations because the accumulated shape change measure C for edge-split operations is zero. Edge split operations do not change the shape of the mesh, so performing them first is not detrimental to normalization of edge lengths. Furthermore, by performing edge-split operations first, more edges are added to the mesh. This will give the algorithm more options (i.e., edges to operate on) when attempting to normalize edge lengths through edge collapse. As the algorithm progresses, long edges in the edge operation priority queue will have non-zero accumulated shape change, so edge-split operations will occur interspersed with edge-collapse operations.

2.7 Mesh Boundaries

In computer vision, the meshes generated from 3-D data will often have boundaries due to partial views of a scene or incomplete scene data. Like interior edges, boundary edges are inserted into the priority queue based on their accumulated shape change measure and edge length weight. If the

boundary edge is shorter than the desired mesh resolution, then it should be collapsed. Collapsing a boundary edge changes the shape of the mesh and the shape of the boundary. Since the boundary usually contains important information about the shape of the object (e.g., occluding contour), its shape needs to be preserved. Since our shape change measure determines the maximum distance between the mesh faces before and after collapse, it takes into account the change in shape of the mesh shape and its boundary. If collapsing an edge changes the boundary greatly, then the shape change measure of the edge is large and the edge is not operated on. As with interior edges, splitting a boundary edge does not change mesh shape, so its shape change measure is zero. The edge-collapse and edge-split operations as applied to boundary edges are given in Figure 8.

3 Discussion

There exists a distinct advantage in using the maximum distance between meshes before and after operation as our shape change measure; edges along creases or ridges in the surface mesh can have small shape change measure and hence be operated on. This is in contrast to more conservative measures of shape change, such as distance to best fit plane [17]. Along ridges, the distance to best fit plane (plane fit to the vertices in the edge star of the edge) is large, so these edges cannot be decimated. Figure 9 demonstrates the effectiveness of our shape change measure for such cases. A surface mesh representation of a cube is decimated using three different mesh simplification algorithms. After the original surface mesh, the result of an algorithm (similar to Schroeder et al. [18])is shown This algorithm simplifies meshes by removing points that are a small distance to the best fit plane of the point followed by re-triangulation of the local neighborhood. Since points on the 12 creases of the cube are a large distance to the plane fit to the local neighborhood of the point, they

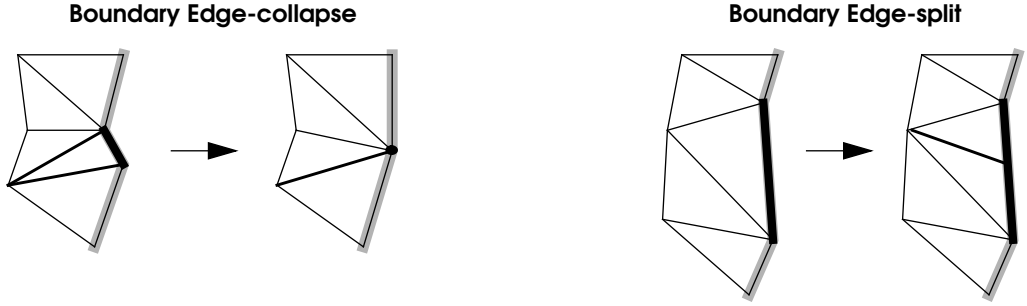

Figure 8: The effect of the edge-split and edge-collapse operations on edges on the boundary of the surface mesh. Special version of the operations are implemented because the local mesh neighborhood of an edge on the boundary is different from the local mesh neighborhood of an edge in the interior of the mesh.

are not removed. The result is that a large number of vertices are left along the creases of the cube, while few remain in the interior of the cube sides. In addition, the lengths of the edges in the surface mesh are widely distributed.

The next simplification result was generated using our algorithm without using the edge-split operation. The measure of shape change used was the average distance of the vertices in edge *e* from the best fit plane to *EdgeStar(e)*. The edges were ordered based on this distance to the best fit plane and edge length. Although there is less spread in edge length, the edges along the cube creases are not collapsed because the distance to the best fit plane is large along the creases. The edges along the creases show up in the edge length histogram as a spike at the short edge end of the histogram.

The final result shows normalization of the cube lengths using the algorithm presented in this paper. Using the distance between meshes before and after operation as the shape change measure allows edges along the creases of the cube to be operated on. The result is a much smaller spread in edge lengths than would be possible with the previous two implementations. This result clearly shows the ability of our algorithm to normalize surface mesh representations of polyhedral objects.

Figure 10 demonstrates, in the extreme, how much the distance between meshes shape change measure increases the simplification over that possible when using the planar fit error shape change measure. Using the distance between meshes criterion, the cube mesh shown at the top of Figure 9 can be reduced to 26 vertices, while the planar fit error criterion only allows a reduction to 309 vertices. In both cases, the algorithms are executed until no more simplification is possible without distorting the shape of the cube.

Overall the computation complexity of length normalization algorithm is *O(NlogN)* where *N* is the number of edges in the original mesh. Creating the priority queue of edges takes *N* insertions each into a dynamically sorted list (the priority queue). The priority queue is implemented efficiently as a Fibonacci heap [16], so each insertion takes *O(logN)* time. During normalization, each edge operation requires the re-insertion of a roughly fixed number of edges back into the priority-queue. If *M* edge operations are applied to the mesh, mesh normalization will take *O(MlogN)*. In our experience, the number of edge operations is on the order of the number of edges in the original mesh, so application of all of the edge operations takes *O(NlogN)*. Combining this with the time it takes to create the priority queue, the overall complexity of the algorithm is *O(NlogN)*.

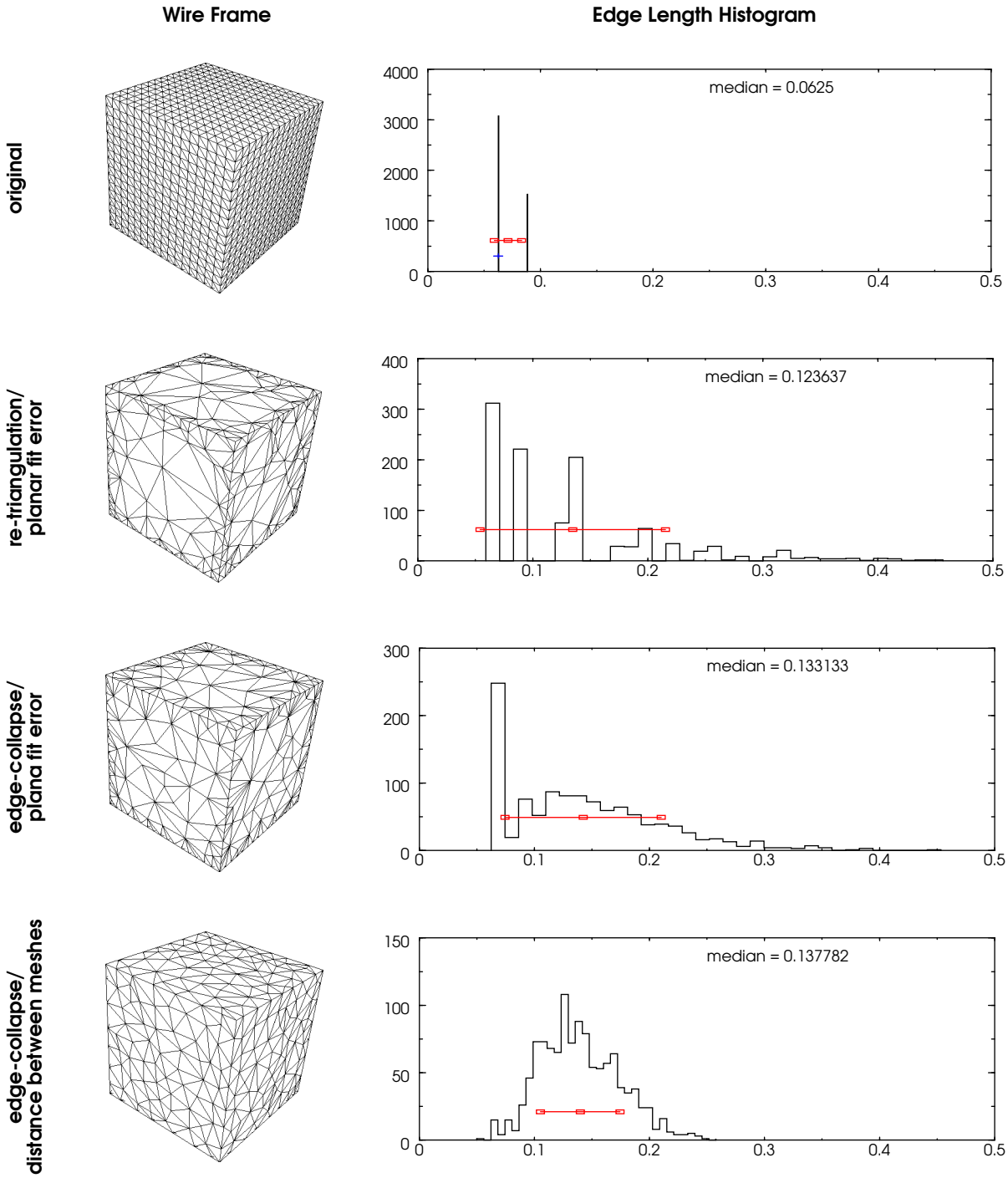

Figure 9: Comparison of length normalization for three mesh simplification algorithms. Mesh decimation using point removal followed by re-triangulation and a planar fit error results in a large spread in edge lengths and too many vertices on the creases of the cube. An algorithm that uses edge-collapse and planar fit error has a smaller spread in edge lengths, but still does not remove edges on the creases of the cube. Not until the distance between meshes is used to measure shape change are edges removed along the creases of the cube, resulting in a much more compact edge length histogram.

4 Results

In order to demonstrate the generality of our algorithm, we present results from multiple sensing domains common in 3-D computer vision. The results are represented as hierarchies of surface meshes generated from the original data set. Each level in a hierarchy is generated by applying the length normalization algorithm to the original data. The resolution of each level is set by adjusting the edge length bounds (L_{MIN}, L_{MAX}) and the maximum accumulated shape change C_{MAX} input into the algorithm. The edge length bounds input into the algorithm are shown as a line with boxes indicating the bounds and the desired resolution on the edge length histogram for each level of the hierarchy. The desired resolution doubles between each level of the mesh. This is validated by the doubling of the measured median of edge lengths between each level. The edges not within the edge length bounds have accumulated shape change that is greater than C_{MAX}. In the results, the ability of our algorithm to normalize edge lengths is shown visibly with wire frame meshes with hidden lines removed and shaded surface meshes.

Figure 11 shows a hierarchy of normalized surface meshes for a model of a rubber ducky. The in-

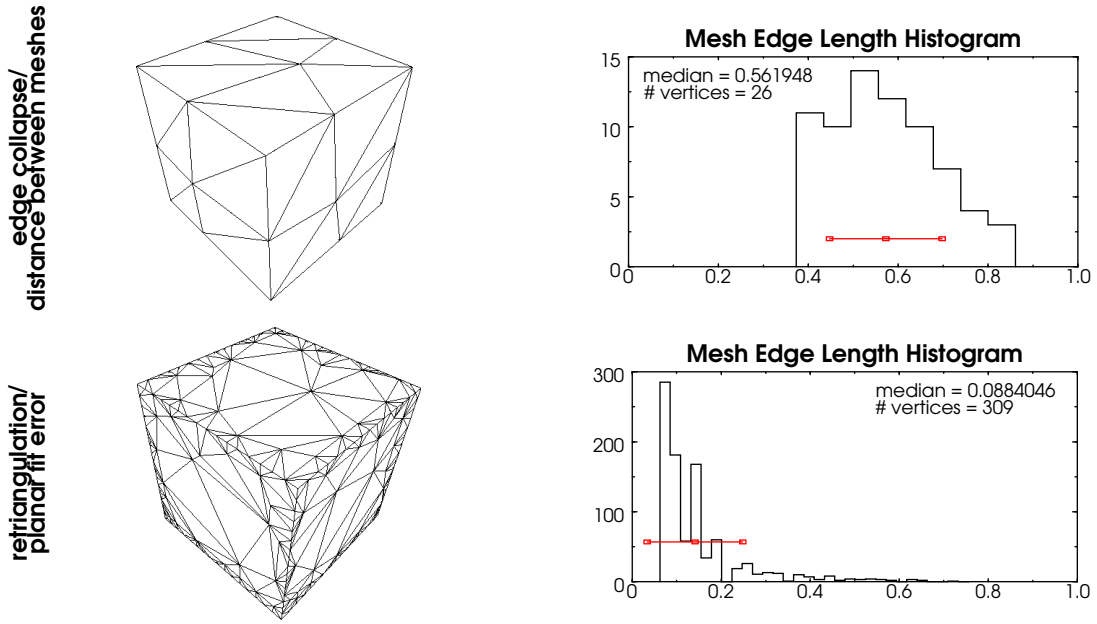

Figure 10: Comparison of shape change measure on the amount of simplification possible for polyhedral objects. Both of the cubes shown above were simplified as much as possible without deviating from the original cube shape (shown at the top of Figure 9) using two different shape change measures. Using the distance between meshes shape change measure allows for a much greater simplification of the cube than possible with the planar fit error shape change measure because planar fit error prevents simplification along the creases edges in the cube.

put surface mesh was generated using a volumetric range image merging algorithm [24]. Multiple views of the ducky taken with a structured light range sensor were inserted into a volumetric data structure that describes the surface of the duck. The seamless surface of the duck was then extracted from the volume using the Marching Cubes algorithm. A characteristic of Marching Cubes is the generation of many short edges; these short edges generate the aliasing noticeable in the original data. The first level of the hierarchy removes these short edges and subsequently the aliasing. This result demonstrates the ability of our algorithm to handle Marching Cubes data and curved surfaces without holes.

Figure 12 shows a hierarchy of normalized surface meshes for a model of a femur bone. The original data was created from Computed Tomography (CT) slices of a human femur bone. Surface contours were extracted from each CT slice, and the surface contours were subsequently linked to create the surface mesh. There are holes at the top and bottom of the femur bone due to incomplete 3-D data. This result demonstrates the ability of our algorithm to produce normalized surface meshes while preserving shape from surface meshes constructed from CT contours that contain boundaries.

Figure 12 shows a hierarchy of normalized surface meshes generated from a range image of an industrial scene. The scene contains three I-beams, a pipe with two elbow joints and a water heater tank. The original data was generated from the range image by making each pixel in the range image a vertex and connecting pixels in adjacent rows and columns with edges. Range discontinuities were eliminated by removing extremely long edges from the surface mesh. Specular reflections off of a few surfaces in the scene cause incorrect range computations and result in holes in the original surface mesh. This results demonstrates the ability of our algorithm to handle meshes with significant mesh boundaries and holes. It also shows that, without modifying parameters, our algorithm can normalize meshes that contain both polyhedral (I-beams) and free-form (pipes) objects.

The final result demonstrates the use of our algorithm for simplifying digital elevation maps. The original surface mesh is generated from a digital elevation map of the Lockheed Martin facility in Denver, Colorado by making each pixel in the map a vertex and connecting pixels in adjacent rows and columns with edges. A mesh hierarchy generated with our algorithm is then shown next to a hierarchy of meshes generated through simple sub-sampling of the original digital elevation map. The two meshes show at each level of the hierarchy each have the same number of points. It is clear

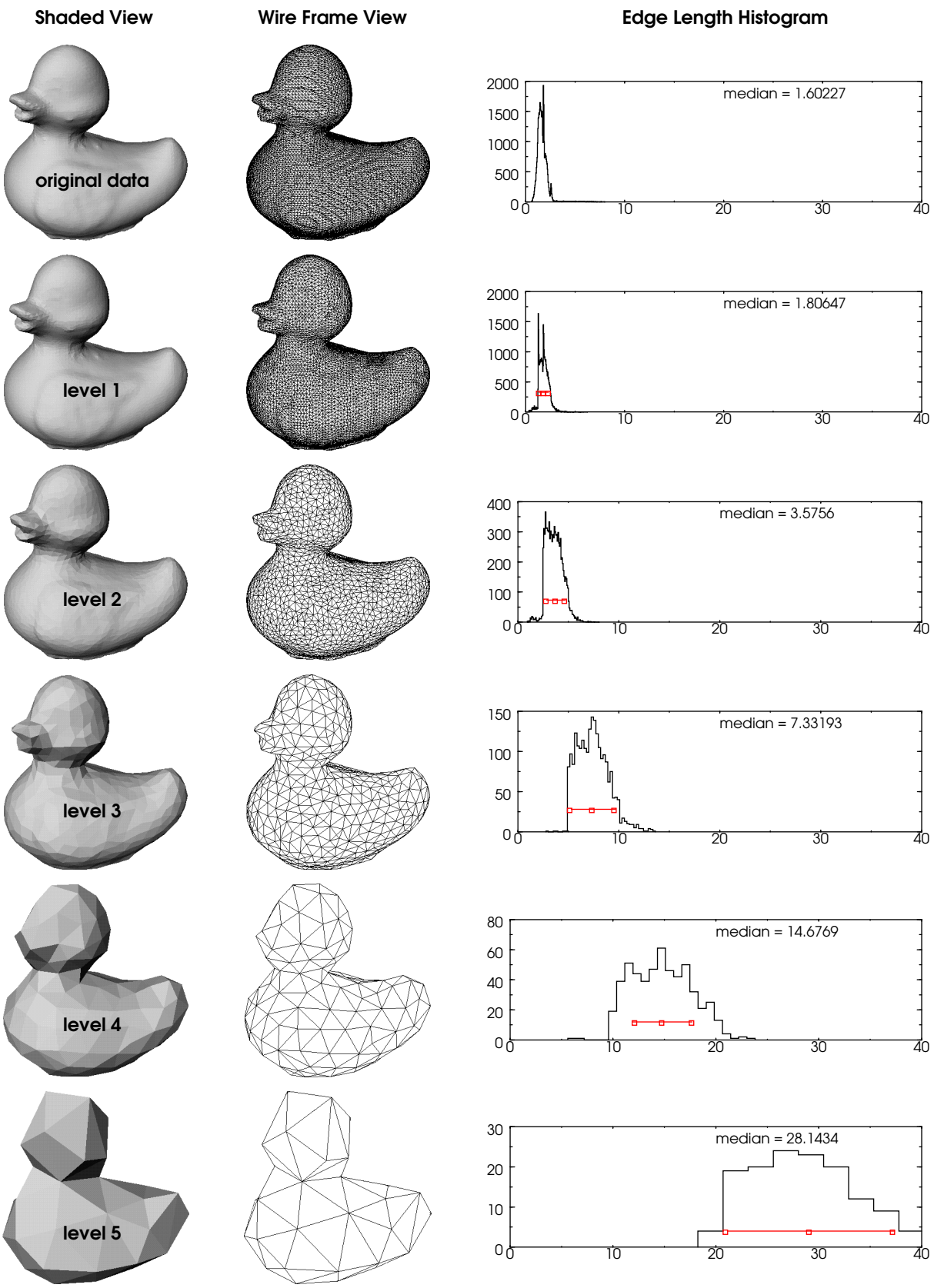

Figure 11: Hierarchy of duck meshes with original data generated from the from Marching Cubes algorithm.

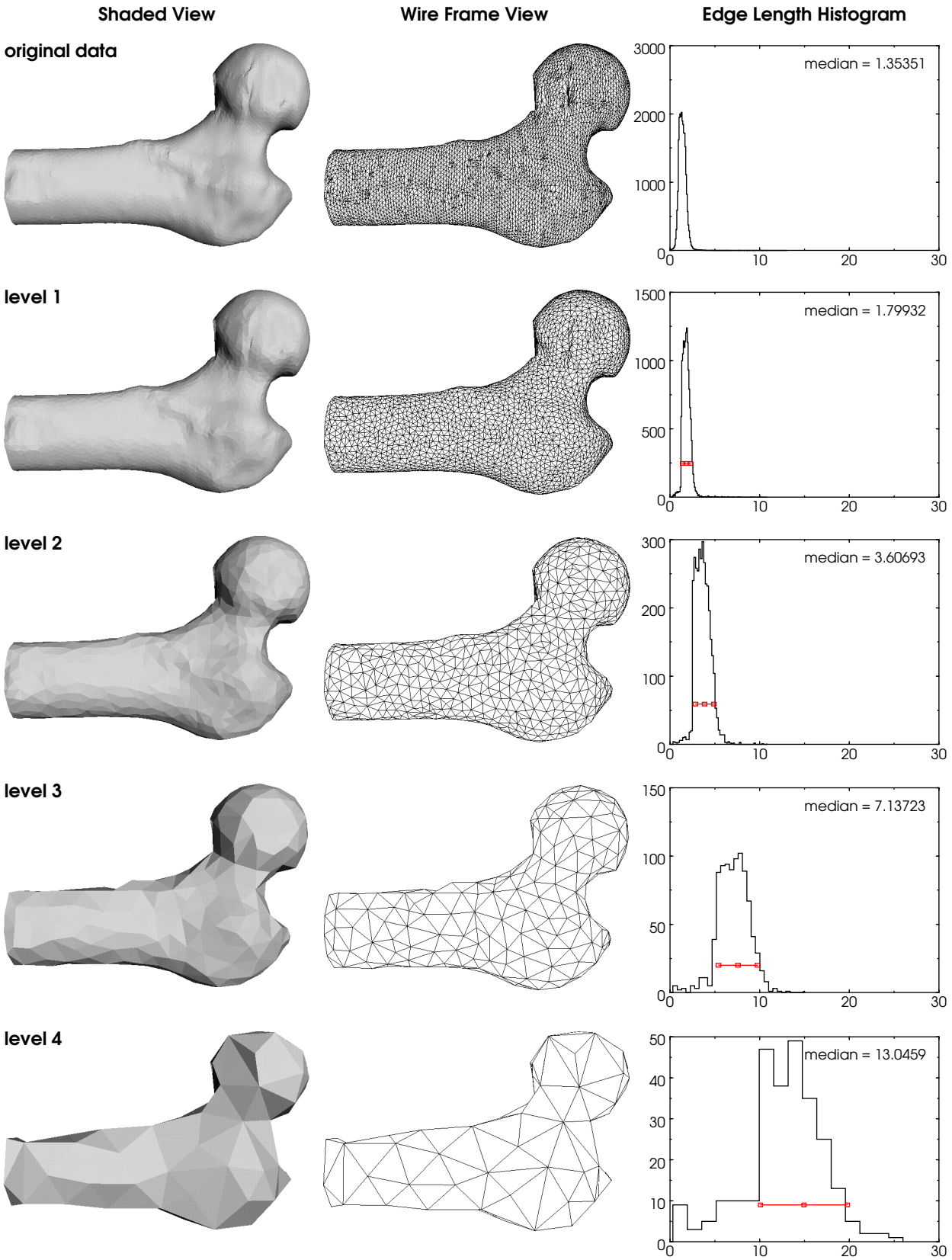

Figure 12: Hierarchy of femur meshes with original data from CT.

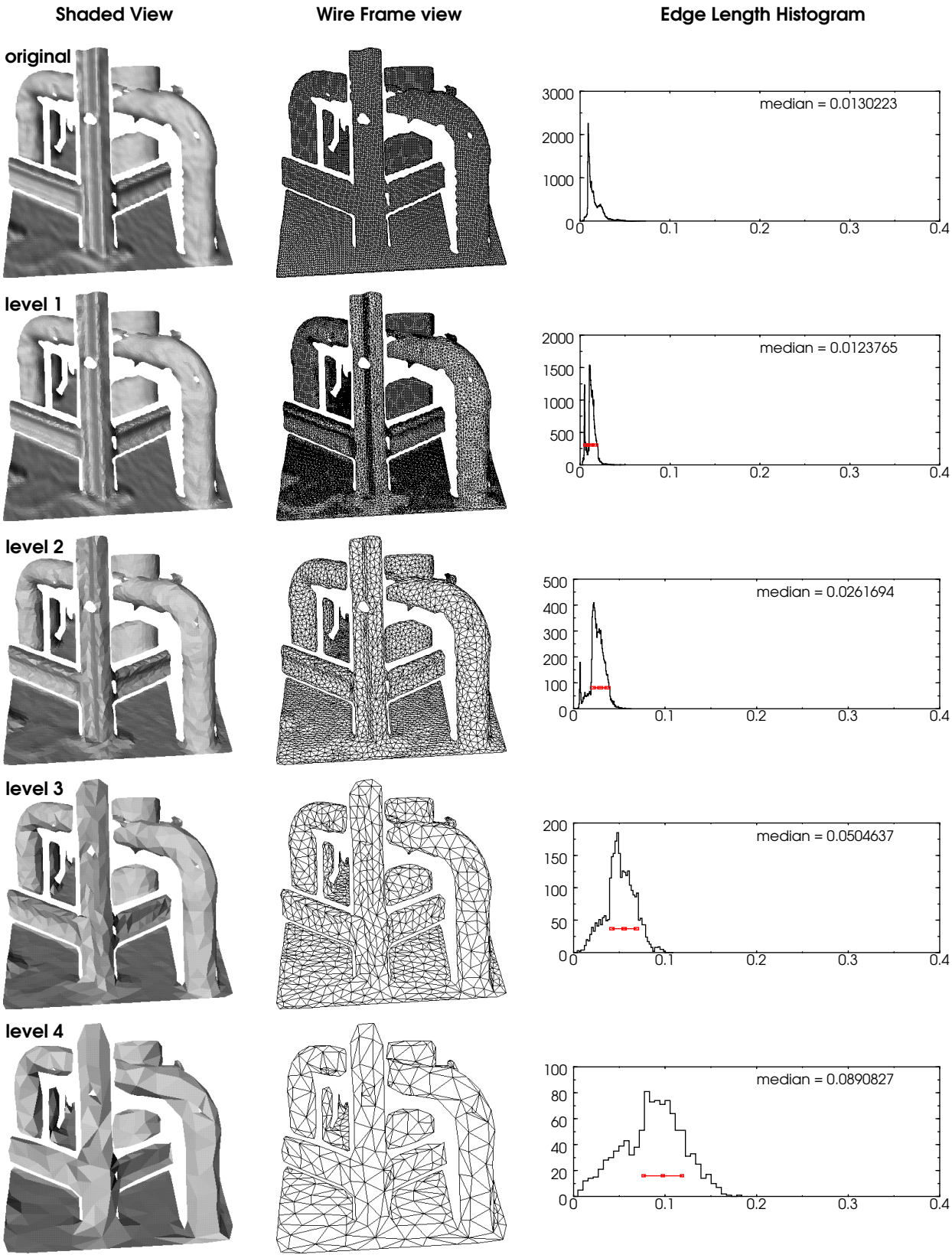

Figure 13: Hierarchy of range image with edges along range discontinuities removed.

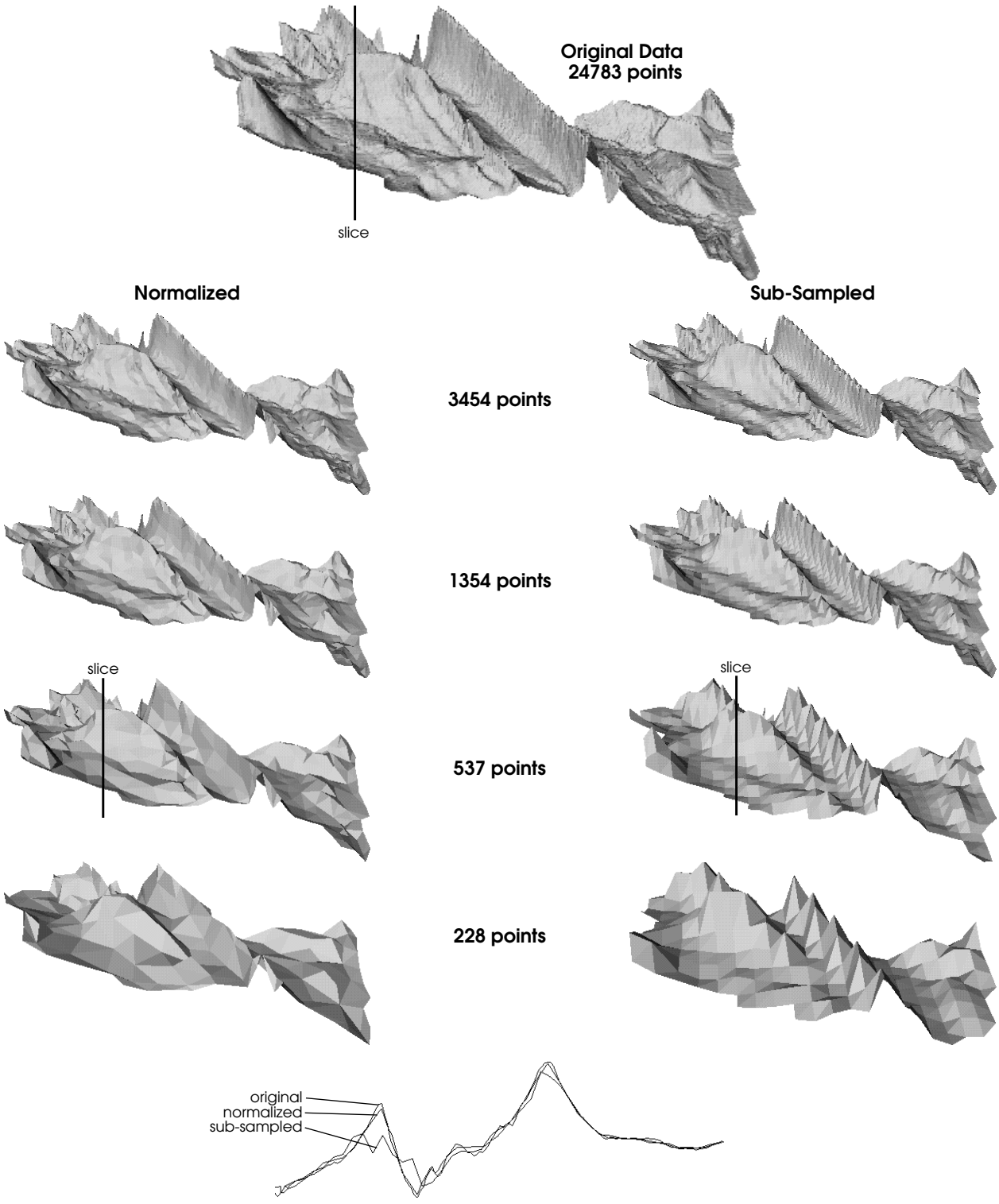

Figure 14: Hierarchy of digital image elevation maps showing the advantage of simplification over sub-sampling. For example, the meshes generated using normalization accurately convey the prominent ridge through the hierarchy while the ridge in the sub-sampled hierarchy turns into peaks and valleys as the sub-sampling increases. Slicing through the third level of the hierarchy provides a 2-D confirmation of the benefit of normalization over sub-sampling.

from shaded views of the meshes that sub-sampling the mesh drastically changes the shape of the terrain map. For example, the prominent ridge in the in the terrain remains a ridge in the normalized hierarchy, while the ridge turns into a sequence of peaks and valleys in the sub-sampled hierarchy. The ability of the normalized hierarchy to preserve shape is also demonstrated in 2-D slices taken through the data for the third level of the hierarchy.

5 Conclusion

We have developed an algorithm that controls the resolution of a surface mesh by normalizing the lengths of the edges in the mesh while preserving mesh shape. The algorithm was developed with special attention to the types of surface meshes encountered in 3-D computer vision. It works equally well on meshes representing curved and polyhedral objects with or without boundaries. Our algorithm is similar to other mesh simplification algorithms in that it iteratively changes the mesh by applying local mesh operators, which in our case are edge-collapse and edge-split. Our algorithm differs from others in that the order in which edge operations are applied depends on the shape change induced in the mesh as well as on the length of the edge. It also uses a novel shape change measure that more accurately predicts the effect of applying a mesh operation and consequently allows for simplification along surface ridges. Finally, our algorithm preserves the shape of the mesh during normalization by balancing expansion and shrinkage during edge-collapse and by applying a global bound on the maximum change in mesh shape.

In the future we plan to extend our algorithm in three directions. Instead of representing a mesh hierarchy using discrete levels, we would like to represent a hierarchy as a continuous stream of edge-collapse and edge-split operations. In this way, a mesh of arbitrary resolution could be generated from the stream by applying the operations in the stream until the desired resolution is reached. Representing the hierarchy as a stream of data would also allow gradual transmission of the model and continuous level of detail model generation. Another direction for this work is to explore the combination of geometry and texture or other surface properties in surface mesh normalization. The final direction we would like to explore is the introduction of topology changing operations into mesh normalization. Removing holes and merging surface patches could conceivably reduce the number of edges needed to describe a mesh at the desired resolution.

Acknowledgments

This research was performed in the Robotics Institute, Carnegie Mellon University and was supported by the Department of Energy under contract DE-AC21-92MC29104. We would like to thank Paul Heckbert for many conversations on possible algorithms for mesh simplification and in particular for pointing out to us the idea of bounding global shape change. In addition, we would like to thank David Simon for providing the CT data sets, Mark Wheeler for providing the rubber ducky data set and Jay Gowdy for providing the digital elevation map data set. Furthermore, this work would not have been possible without the support of Jim Osborn and all of the members of the Artisan project.

References

[1] P. Agarwal and S. Suri. "Surface approximation and geometric partitions." *Proc. Fifth Symp. Discrete Algorithms*, pp. 24-33, 1994,

[2] J. Cohen, A. Varshney, D. Minocha, G. Turk, H. Weber, P. Agarwal, F. Brooks and W. Wright. "Simplification envelopes." *Proc. Computer Graphics 1996 (SIGGRAPH '96)*, pp.119-128, 1996.

[3] C. Chua and R. Jarvis. "3-D free-form surface registration and object recognition." *Int'l Jour. Computer Vision*, vol. 17, pp. 77-99, 1996.

[4] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery and W. Stuetzle, "Multiresolution analysis of arbitrary meshes," *Proc. Computer Graphics 1995 (SIGGRAPH '95)*, pp. 173-182, 1995.

[5] J. Foley, A. van Dam, S. Feiner and J. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, New York, 1990.

[6] P. Flynn and A. Jain. "BONSAI: 3D object recognition using constrained search." vol. 13, no. 10, pp. 1066-1075, October 1991.

[7] M. Garland and P. Heckbert. "Surface simplification using quadric error metrics." *Proc. Computer Graphics 1997 (SIGGRAPH '97)*, pp. 209-216, 1997.

[8] A. Guéziec, "Surface simplification with variable tolerance," *Proc. Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, pp. 132-139, November, 1995.

[9] P. Heckbert and M. Garland, "Survey of polygonal surface simplification algorithms," Carnegie Mellon University School of Computer Science Tech. Report (CMU-CS-97-194), December 1997.

[10] P. Hinker and C. Hansen, "Geometric optimization," *Proc. Visualization '93*, pp. 189-195, October 1993.

[11] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, "Mesh optimization," *Proc. Computer Graphics 1993 (SIGGRAPH '93)*, pp. 19-26, 1993.

[12] H. Hoppe, "Progressive Meshes," *Proc. Computer Graphics 1996 (SIGGRAPH '96)*, pp. 99-108, 1996.

[13] A. Johnson and M. Hebert. "Surface registration by matching oriented points." *Proc. Int'l Conf. on 3-D Digital Imaging and Modeling,* May 1997.

[14] A. Kalvin and R. Taylor, "Superfaces: polyhedral approximation with bounded error," *SPIE Medical Imaging,* vol. 2164, pp. 2-13, 1994.

[15] V. Koivunen and R. Bajcsy. "Spline Representations in 3-D Vision." in *Object Representation in Computer Vision*, M. Hebert, J. Ponce, T. Boult and A. Gross, eds. Springer-Verlag, pp. 177-190, December 1994.

[16] S. Näher and Christian Urhig. *The LEDA User Manual: Version R 3.3*, Max-Planck-Institut für Informatik, 1996.

[17] W. Schroeder, J. Zarge and W. Lorensen, "Decimation of triangular meshes," *Proc. Computer Graphics 1992 (SIGGRAPH '92)*, pp. 65-70, 1992.

[18] W. Schroeder, "A global error bound for triangle decimation," *WWW download.*

[19] K. Shimada. "Physically-based automatic mesh generation." *Jour. Japan Society for Simulation Technology*, vol. 12, no. 1, pp. 11-20, 1993.

[20] F. Stein and G. Medioni. "Structural Indexing: efficient 3-D object recognition." *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 125-145, 1992.

[21] D. Terzopoulos and M. Vasilescu, "Sampling and reconstruction with adaptive meshes," *Proc. Computer Vision and Pattern Recognition (CVPR '91)*, pp. 70 - 75, 1991.

[22] G. Turk, "Re-tiling polygonal surfaces," *Proc. Computer Graphics 1992 (SIGGRAPH '92)*, pp. 55-64, 1992.

[23] M. Vasilescu and D. Terzopoulos, "Adaptive Meshes and Shells: Irregular triangulation, discontinuities and hierarchical subdivision," *Proc. Computer Vision and Pattern Recognition (CVPR '92)*, pp. 829-832, 1992.

[24] M. Wheeler. "Automatic modeling and localization for object recognition." *Ph. D. Thesis Carnegie Mellon University School of Computer Science CMU-CS-96-188*, October, 1996.

[25] A. Witkin and P. Heckbert, "Using particles to sample and control implicit surfaces." *Proc. Computer Graphics 1994 (SIGGRAPH '94)*, pp. 269-277, July 1994.