From: AAAI Technical Report SS-00-04. Compilation copyright © 2000, AAAI (www.aaai.org). All rights reserved.

Extraction of feature lines on triangulated surfaces using morphological operators

Christian Rössl, Leif Kobbelt, Hans-Peter Seidel
Max-Planck-Institut für Informatik, Computer Graphics Group
Im Stadtwald
66123 Saarbrücken, Germany
{roessl,kobbelt,hpseidel}@mpi-sb.mpg.de

Abstract

Triangle meshes are a popular representation of surfaces in computer graphics. Our aim is to detect feature on such surfaces. Feature regions distinguish themselves by high curvature. We are using discrete curvature analysis on triangle meshes to obtain curvature values in every vertex of a mesh. These values are then thresholded resulting in a so called binary feature vector. By adapting morphological operators to triangle meshes, noise and artifacts can be removed from the feature. We introduce an operator that determines the skeleton of the feature region. This skeleton can then be converted into a graph representing the desired feature. Therefore a description of the surface's geometrical characteristics is constructed.

Introduction

There are several different representations for surfaces used in computer graphics, ranging e.g. from parametric surfaces defined as functions over a parameter domain to surfaces described by constructive solid geometry that allows combination of simple geometric primitives. As usual in computer science, different representations are favored for different purposes.

Even so, there is a very simple but effective representation for many kinds of surfaces: triangle meshes. On the one hand they are capable of describing surfaces of arbitrary shape and topology. On the other hand they use the most basic graphic primitive for describing a surface, hence graphics hardware usually deals with sets of triangles.

In recent years triangle meshes have grown to become popular, as for example techniques have been developed to use such meshes directly for geometric modeling in a comfortable way rather than the usual parametric free-form surfaces, e.g. NURBS (Kobbelt et. al. 1998). Besides, a triangulated surface may be obtained from a laser range scanner sampling a real world object. The huge sets of data which arise as a result can now be reduced without losing too much accuracy of the surface representation (Kobbelt, Campagna, and Seidel 1998). For this reason, triangle meshes are the most appropriate representation when dealing with real world objects.

Copyright © 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

This paper focuses on extracting a structured description of an object represented by a triangulated surface from its geometry. Thus, we enrich an unstructured geometric representation with semantics of the object being investigated. The result may then be used as input to other algorithms.

Therefore we propose a technique to find feature lines on a triangle mesh. In feature regions the local geometry of the mesh is heavily bent. Feature lines approximately pass along the ridge of maximum inflection. Mathematically such local characteristics of geometry can be measured by the curvature of the surface. This kind of information is important for a variety of applications as e.g rating surface quality or classification of surfaces, patch layouting for reverse engineering, or avoidance of aliasing during remeshing by resampling the surface.

Applying some binary threshold operation on the obtained curvature values leaves us with a feature vector whose components represent the binary values assigned to the vertices. We then adapt morphological operators from digital image analysis in order to reduce noise and irritating artifacts on the feature vector, and we construct operators to extract a "feature skeleton". This skeleton can then be transformed into a graph afterwards. Its edges represent polygons on the mesh lying near the region of interest, i.e. the feature to be extracted. The polygons are connected by the nodes of this graph.

Thus, we transform an unstructured surface representation into a more structured description of surface characteristics defined by geometry. This description may be used as input to further algorithms.

Discrete curvature analysis

We use geometric curvature to extract features of a surface. As curvature is invariant to rotations or translations of an object, one can expect similar results from (not too) different views of the object or at least some overlap of features. E.g. registration of different views or scans can take advantage of this fact. Curvature also allows to classify different regions of an object. So geometric primitives that have been used to construct the object may be recovered.

Curvature cannot be evaluated directly for triangle meshes, because it is mathematically defined for smooth surfaces only. A triangle mesh is a piecewise linear surface, so it is not clear how to calculate any derivatives on such a

mesh.

Our approach locally estimates the first and second fundamental form of the surface $F(u, v)$ in every vertex of its triangulation. Deriving surface curvatures like principle curvatures from the fundamental forms is straightforward. An introduction to the basic concepts of differential geometry can be found e.g. in (Farin 1996; do Carmo 1976).

First, we construct a nearly isometric ($\|F_u\| \approx 1$, $\|F_v\| \approx 1$ and $F_u F_v \approx 0$) parameterization for a vertex and its neighbors by using an exponential map. Given this parameterization the surface can then be locally approximated by a second order Taylor polynomial. This is done by (least squares) solving a linear system, yielding the Taylor coefficients F_u, F_v, F_{uu}, F_{uv}, F_{vv} as solution. These derivatives enable us to estimate further differential parameters e.g. normal vectors, Gaussian, mean or maximal curvature at the vertex V of the triangulation as needed. Figure 1 (left) shows the maximal curvature on a technical model represented by a triangle mesh. Dark regions denote high curvature. For color coding the values are interpolated into the triangles.

Extraction of feature regions

With discrete curvature analysis we are able to estimate curvature values for every inner vertex of the mesh. After some prefiltering and a thresholding operation yielding a binary feature vector, morphological operators are used to produce a skeleton that lies close to the region of interest resp. the feature lines.

Filtering and thresholding

If the input data are point samples from a real object, we have to deal with high frequency noise. This effect is intensified by the use of second order derivatives. Operating directly on the values obtained from curvature approximation might give dissatisfying results. Therefore some filtering should be applied to the curvature data before.

A simple median filter that considers a vertex and its neighbors gives good results. For some applications, e.g. constructing a curvature histogram for thresholding or visualization, outliers on the input data may entail undesirable effects. A simple yet effective solution is to clamp the 5% highest and lowest values.

Our morphological operators will deal with binary values only. Therefore some classification scheme must be applied to the vertices. Let X_i be a certain curvature value (e.g. maximal curvature) of a vertex V_i of a triangle mesh with N vertices $V_i, \ldots, V_N$.. Now X_i shall be transformed to $F_i \in \{0, 1\}\,(1 \leq i \leq N)$, with F_i denoting the "feature" to be assigned to that vertex. We use a thresholding operation to determine the feature vector $F := (F_i)_i$:

$$1 \leq i \leq N: \quad F_i = \begin{cases} 1 & X_i \in [a, b] \\ 0 & \text{else} \end{cases} \tag{1}$$

The thresholding parameters $a, b \in \mathbb{R}$ are obtained in an application specific way. We cannot present an automatic thresholding scheme. In general no assumptions can be made like e.g. different intensities of an object and its background as often made in digital image processing. Though, if there is some knowledge about the surface being processed, automatic and/or multiple thresholding might be applicable. Figure 1 (middle) shows the result of thresholding. It also depends on the application whether the boundary of a mesh shall be included. In the following sections we will refer to F as the *feature vector* and $\{i \mid F_i = 1\}$ as *feature*. Such vertices V_i are said to be "marked".

Morphological Operators

Mathematical morphology has been being used in digital image analysis for quite a long time. Morphological operators are particularly interesting and often preferred to convolution operators because of their simplicity and the fact, that they can be efficiently implemented in hardware (Haralick, Sternberg, and Zhuang 1987).

We adapt morphological operators to operate on a binary feature vector F corresponding to a triangle mesh to extract feature lines from the previously thresholded curvature values. First some notations:

For convenience we introduce an alternative notation for the feature vector $F \in \{0, 1\}^N$. Let this vector be assigned the set

$$\mathcal{F} := \{j \in \{1, \ldots, N\} \mid F_j = 1\} \tag{2}$$

with $\overline{\mathcal{F}} := \{1, \ldots, N\} \setminus \mathcal{F}$. The two notations are equivalent since $i \in \mathcal{F} \Leftrightarrow F_i = 1$.

In addition we introduce the neighborhood relation **nhd**, assigning each vertex V_i the set of its 1-neighborhood (including V_i). For convenience **nhd** maps vertex *indices* only. It is defined as

$$\mathbf{nhd}\{i\} = \{i\} \cup \{j \mid \exists \text{edge}(V_i, V_j)\} \tag{3}$$

The radius of a neighborhood can be recursively enlarged by defining a n-neighborhood $\mathbf{nhd}^n$ as

$$\begin{aligned} &\mathbf{nhd}\{i_1, \ldots, i_k\} := \textstyle\bigcup_{1 \leq \mu \leq k} \mathbf{nhd}\{i_\mu\} \\ &\mathbf{nhd}^1\{i\} := \mathbf{nhd}\{i\} \\ &\mathbf{nhd}^{n+1}\{i\} := \mathbf{nhd}(\mathbf{nhd}^n\{i\}) \quad (n > 1) \end{aligned} \tag{4}$$

Dilation and Erosion. The classical definitions of dilation and erosion are based on addition in a (n-dimensional) Euclidean vector space E^n. E.g. the dilation operator generates from two given sets $A, B \subset E^n$ the union $A \oplus B = \{c \in E^n \mid c = a + b, a \in A, b \in B\}$. Here A is the image or pattern to be dilated, and B denotes the so called structure element. (Haralick, Sternberg, and Zhuang 1987)

Such definitions for a vector space E^n cannot be directly used on general triangle meshes. Since the topology of the mesh is unknown, there is no reasonable definition for an addition. Therefore we redefine morphological operators for triangle meshes, even though in a limited way.

Definition 1 (dilation) *Let $\mathcal{F} \subseteq \{1, \ldots, N\}$. The* dilation *of $\mathcal{F}$ by* **nhd**^{n} *is defined as*

$$\mathbf{dilate}^n(\mathcal{F}) := \{j \mid \exists i \in \mathcal{F} \,:\, j \in \mathbf{nhd}^n\{i\}\}$$

The n-neighborhood $\mathbf{nhd}^n$ is used as structure element for every vertex. Thus it adapts in a way to the local topology. One can utilize neighborhoods with different radii as

Figure 1: Left: Maximal curvature on a technical model, dark regions denote high curvature. Prefiltering by a median filter has been applied. Middle: After thresholding. Right: After an opening and a closing operation. The mesh boundary was added to the feature. Visual artifacts arise from color interpolation over large triangles.

structure element, but it is not possible to use only "certain neighbors" from the same neighborhood. This would not be definite for every local topology and every vertex. For that reason our operators resemble the classical ones defined in Euclidean vector space restricted to a disk-like structure element $\{(x, y) \mid -n \leq x, y \leq n\}$. As one single structure element $\mathbf{nhd}^n$ will be employed we use a quasi unary notation. In classical morphology the structure element would be the second operand of a binary operator $\oplus$.

The dilation operator adds vertices to the feature, $\#\mathbf{dilate}^n(\mathcal{F}) \geq \#\mathcal{F}$. $\mathcal{F}$ is grown in a way preserving its "shape" on the mesh. The dilation operator can therefore be effectively used to fill "holes" of unmarked vertices inside and at the boundary of the feature.

Now, another operation is needed to reverse the effect of dilation so that the original shape is recovered. Therefore we have to shrink $\mathcal{F}$. Apart from that, this shrinking or erosion operator cuts off undesired branches.

Definition 2 (erosion) *Let* $\mathcal{F} \subseteq \{1, \ldots, N\}$. *The* erosion *of* $\mathcal{F}$ *by* $\mathbf{nhd}^n$ *is defined as*

$$\mathbf{erode}^n(\mathcal{F}) := \{j \mid \mathbf{nhd}^n\{j\} \subseteq \mathcal{F}\}$$

Dilation and erosion can be applied with a n-neighborhood as structure element. For implementation it might be helpful to note that the effect of using a larger structure element can also be achieved by iteratively applying the respective operator. Hence $\mathbf{dilate}^n = \mathbf{dilate}^1 \circ \cdots \circ \mathbf{dilate}^1$ (n times). The same is true for erosion. Thus we can indeed use an unary notation with the structure element described by a multiply applied operator. This is also useful for implementation because walking around a vertex and enumerating its neighbors can be efficiently done, if appropriate data structures are used for the underlying triangle mesh (Campagna, Kobbelt, and Seidel 1999). This is the operation needed to construct a 1-neighborhood **nhd**.

Opening and closing. The dilation and erosion operators have been introduced to construct more powerful morphological operators that suppress noise on the feature.

Dilation and erosion indeed remove artifacts that may be left even after prefiltering, but they do not preserve the size of the feature. By consecutively shrinking and then growing $\mathcal{F}$, branches will be cut and the original shape will be preserved. The resulting operator is called opening.

Definition 3 (opening) *The* opening *operator is defined as*

$$\mathbf{open}^n := \mathbf{dilate}^n \circ \mathbf{erode}^n$$

The term opening refers to the property of the operator to open regions, where the feature just touches itself, furthermore small regions of marked vertices ("islands") are just removed as peninsular regions (branches).

By swapping the order of the application we obtain the closing operator. Therefore $\mathcal{F}$ is first grown and shrinked afterwards, filling holes in the inner region of the feature, and filling bays along the boundary.

Definition 4 (closing) *The* closing *operator is defined as*

$$\mathbf{close}^n := \mathbf{erode}^n \circ \mathbf{dilate}^n$$

Opening and closing can effectively be used to reduce noise and artifacts on the feature $\mathcal{F}$. Both operators are idempotent by their nature, that is multiple application of the same operator yields the same result as one single application. Figure 1 (right) shows the result of a closing operation.

Skeletonization and pruning. Our aim is to extract feature lines with every line segment corresponding to a triangle edge of the mesh. Although artifacts have been removed from the feature the marked region is in general still too coarse to be accepted as such a feature line.

We are therefore looking for the skeleton of $\mathcal{F}$. Mathematically, the skeleton of a set $S \in \mathbb{R}^n$ is defined as follows. For every $x \in S$ let $D(x)$ be the largest disk centered at x with $D(x) \subseteq S$. Then x is in the skeleton of S if there is no other disk $D' \subseteq S$ that also contains $D(x)$. As a result, the skeleton is the set of centers of the largest disks contained in a set S.

In our case the skeleton shall not be thicker than one vertex. Further on it must follow the topology of the original feature region. Therefore we iteratively "scratch off" several layers of the feature region, similar to the former shrinking process. In digital image analysis the terms skeletonizing, medial axis transformation or just thinning are commonly used for this process (Gonzales and Wood 1993). The difference to the erosion operator is, that there are parts of the feature that must not vanish. Defining a criterion, whether a vertex may be removed or not, is essential for skeletonizing.

Definition 5 (complex vertex, complexity)
Let $F \in \{1, 0\}^N$ *be the feature vector. Let* $(u_\mu^i)_{\mu=0}^{\mu=n_i-1}$

denote the sequence of indices of the n_i neighbors of vertex V_i ordered clockwise.
Now let $c_i := \sum_{\mu=0}^{\mu=n_i-1} |F_{u_\mu^i} - F_{u_{\mu+1 \bmod n_i}^i}|$.
A vertex V_i is defined to be complex, *iff $F_i = 1$ and $c_i \geq 4$. The number c_i is said to be the* complexity *of V_i.*

In order to determine if a vertex V_i is complex, one circulates through its neighbors $V_{u_\mu^i}$ $(0 \leq \mu < n_i)$ while counting the number c_i of transitions from marked to unmarked vertices and vice versa. Thus, complex vertices are either part of the feature line (*edge*) with width 1 ($c_i = 4$) or they belong to a *node*, where several of such lines meet ($c_i > 4$). For practical applications, it might be useful to define all boundary vertices as complex.

Definition 6 (center, disk) *Let $\mathcal{F} \subseteq \{1, \ldots, N\}$ denote a feature region and $i \in \mathcal{F}$ one of its vertices V_i. Then i is defined as* center, *if* $\mathbf{nhd}\{i\} \subset \mathcal{F}$.
The set $\mathcal{O}_i := \{j \mid i \text{ is center} \wedge j \in \mathbf{nhd}\{i\} \setminus \{i\}\}$ *is defined as the* disk *around the center i.*

Notice that as $i \in \mathbf{nhd}\{i\}$, only vertices V_i with $i \in \mathcal{F}$ can be centers. If all neighbors of such a marked vertex are also marked, then this vertex is called a center and its neighbors form the surrounding disk. Note that vertices on disks may also be centers themselves.

We now use the last two definitions to construct a skeletonize operator, which is a kind of erosion that respects certain unremovable vertices. Whether a vertex may vanish or not is expressed in terms of complex vertices, centers, and disks.

Definition 7 (skeletonize operator) *Let $\mathcal{C} \subseteq \mathcal{F}$ be the set of all complex vertices in a feature $\mathcal{F}$. $\bigcirc \subset \mathcal{F}$ denotes the union of all disks, and $\odot \subset \mathcal{F}$ denotes the union of corresponding centers. The* skeletonize operator *is defined as*

$$\mathbf{skeletonize}(F) := \mathcal{F} \setminus (\bigcirc \cap \overline{\mathcal{C} \cup \odot})$$

Figure 2 shows the effect of skeletonizing (middle) for the technical model.

With every iteration of the skeletonize operator the outmost layer is scratched off from the feature region, except for complex vertices. In the inner region, all vertices are centers, and therefore are not removed. The process becomes interesting at "thin parts" of $\mathcal{F}$. They must not be erased or cut off because this would result in a different topology for the obtained feature. Fortunately, such thin regions of $\mathcal{F}$ are defined to be complex and so are not removed.

The skeletonize operator is iterated until the feature does not change anymore. It is obvious, that $\mathcal{F}' := \mathbf{skeletonize}(F) \subseteq \mathcal{F}$. $\mathcal{F}'$ will be left unchanged if $\odot = \emptyset$, or if $\odot$ contains only such centers, whose disk vertices are all complex. Furthermore, disks may be erased from $\mathcal{F}$, therefore the former centers of $\mathcal{F}$ might not be centers anymore in $\mathcal{F}'$. It hast to be pointed out that it is not possible to create new centers or disks by the skeletonize operator. (Note: the number of complex vertices usually does increase, though.) Thus, there are no cycles while iterating **skeletonize**, until the resulting feature set does not change anymore. The *skeletonize algorithm* then terminates after a finite number of iterations of **skeletonize**, resulting in the skeleton of $\mathcal{F}$. Figure 2 (middle) shows a part of the extracted skeleton.

The resulting skeleton contains only complex vertices except for the ends of branches not connected to any node of complex vertices. Small branches are usually treated as undesired artifacts and therefore have to be pruned. Depending on the application, all branches are removed, or just branches up to a certain length.

A pruning operator is provided to iteratively shorten unwanted branches by one vertex each for every iteration. It is defined as

Definition 8 (pruning operator) *Let $\mathcal{S} \subseteq \mathcal{F}$ be a skeleton, and let $\mathcal{C} \subseteq \mathcal{F}$ denote its complex vertices. Then the* pruning operator *is defined as*

$$\mathbf{prune}(\mathcal{S}) = \mathcal{S} \setminus \overline{\mathcal{C}}$$

Figure 2 (right) shows a part of the skeleton of the technical model used in figure 1 after pruning.

Postprocessing

The extracted feature $\mathcal{F}$ has been reduced to a skeleton $\mathcal{F}'$. Assuming that pruning was applied, it therefore consists of "edges" of vertices with complexity 4 and nodes of vertices with complexity > 4 (Def. 5). A natural extension to the extraction of the feature by skeletonizing is to convert the set representation of $\mathcal{F}'$ to a graph.

For this purpose, node-vertices have to be grouped to nodes and edge-vertices to edges. In order to construct the graph, first the nodes are detected by iteratively grouping neighboring node-vertices. After that, edges can be identified by walking from node to node through the original triangle mesh. This will eventually leave rings, that are "closed edges" which never touch any node. These rings have to be processed extra.

Eventually, one is interested in patches of inner vertices (of $\overline{\mathcal{F}'}$) bounded by some edges in addition. These patches can be found by iteratively using a seed fill algorithm, filling and identifying every patch.

For now, the constructed edges are polygons that entirely consist of triangle edges from the mesh. Some fairing can be applied to these polygons, resulting in a smoother curve. If needed, such a piecewise linear curve can be approximated by e.g. a B-Spline curve.

Applications

There are several applications for the proposed feature extraction technique. Our initial aim was reverse engineering of triangulated surfaces. For CAD/CAM applications Spline surfaces have emerged to a de facto standard for surface representation during the last decades. So there is a demand for tools, which convert triangle meshes to such free-form surfaces. As there is no inherent parameterization for triangle meshes, the surface usually has to be partitioned into a couple of patches. For every patch a Spline surface is approximated, taking into account continuity between patches. Finding a suitable patch layout is a demanding task.

There are automatic approaches like (Eck and Hoppe 1996), but they do not really respect natural patch boundaries suggested by curvature. On the contrary the user is left

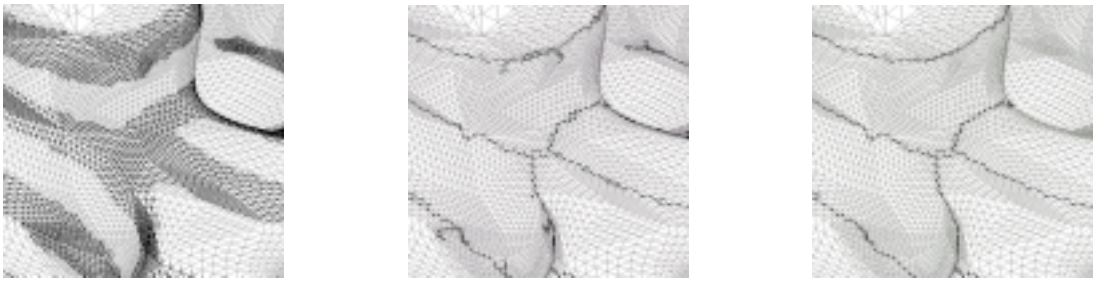

Figure 2: Left: After closing, magnified from figure 1. Middle: Skeleton. Right: After Pruning.

alone with manual "boundary painting" (Krishnamurthy and Levoy 1996). Utilizing feature extraction helps the user to get reasonable patch layouts. But there is still some manual work left.

Usually not all desired patch boundaries can be found in one step. On the one hand, there is need for multiple thresholding and combining the results. On the other hand, the user may interactively complete resp. connect or cut resp. disconnect feature regions. This way, the user can effectively construct a reasonable patch layout.

With the calculated discrete curvature being visualized, one can rate the quality of a triangulated surface. The extraction of feature lines can then be used to detect defects of the surface. Especially, if some knowledge about the surface is provided, e.g. if measured data from a real world manufactured object is compared to a reference model, automatic detection may be applicable.

The proposed feature extraction may also be used to classify surfaces sampled by a laser range scanner. But not only recognition of resp. the distinction between a number of different objects is an interesting application. In order to produce a digital 3D model of a real world object, usually several views of the object have to be registrated and merged. The extracted feature can help to detect overlapping regions between views and to match the different coordinate systems per view.

Conclusion

We presented a technique to extract feature lines from triangulated surfaces. Therefore, curvature values obtained from discrete curvature analysis were thresholded resulting in a binary feature vector. We adapted morphological operators from digital image analysis to triangle meshes so that noise and artifacts are suppressed, and we constructed a skeletonize operator that extracts the desired feature. The feature lines can then be transformed into a graph whose edges represent polygons lying near the region of interest resp. the feature.

A problem arises from reduced meshes (Kobbelt, Campagna, and Seidel 1998) with triangles of strongly varying edge length. The proposed morphological operators have to be extended to respect geometry as well as topology. Therefore one would define a neighborhood **nhd** of a vertex V as the vertices lying inside a sphere of a certain radius centered at V.

There are a number of applications in computer graphics for which feature extraction is useful. We are interested in patch layouting for reverse engineering in particular. The proposed technique effectively extracts the desired feature lines. Anyhow, there is still need for manual assistance, e.g. in order to determine reasonable thresholds or to manually fix the feature vector. At this point techniques from other disciplines in computer science such as AI may be able to help.

Furthermore, we transformed an unstructured surface representation of an object into a structured description of this object in terms of characteristics of its geometry. This description may be used as input to other algorithms concerned with semantics of the object rather than plain geometry.

References

Campagna, S.; Kobbelt, L.; Seidel, H.-P. 1999. Directed Edges — A Scalable Representation for Triangle Meshes. to appear in *ACM Journal of Graphics Tools*

do Carmo, M.P. 1976 *Differential Geometry of Curves and Surfaces*. Prentice Hall

Eck, M.; Hoppe, H. 1996. Automatic Reconstruction of B-Spline Surfaces of Arbitrary Topological Type. *Computer Graphics (Proc. SIGGRAPH '96)*: 325-334

Farin, G. 1996. *Curves and Surfaces for Computer Aided Geometric Design. A Practical Guide*. 4. ed., Academic Press

Gonzales, R.C.; Woods, R.E. 1993. *Digital Image Processing*. Reprint with corrections. Addison-Wesley

Haralick, R.M.; Sternberg, S.R.; and Zhuang, X. 1987. Image Analysis Using Mathematical Morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. PAMI-9, No. 4: 532-560

Kobbelt, L.; Campagna, S.; Seidel, H.-P. 1998. A general framework for mesh decimation. *Graphics Interface 98 Proc.*

Kobbelt, L.; Campagna, S.; Vorsatz, J.; Seidel, H.-P. 1998. Interactive Multiresolution Modeling on Arbitray Meshes. *Computer Graphics (Proc. SIGGRAPH '98)*: 105-115

Krishnamurthy, V.; Levoy, M. 1996. Fitting Smooth Surfaces to Dense Polygon Meshes. *Computer Graphics (Proc. SIGGRAPH '96)*: 313-324

Welch, W.; and Witkin, A. 1994. Free-Form Shape Design Using Triangulated Surfaces. *Computer Graphics (Proc. SIGGRAPH '94)*: 247-256