Appeared in Lake Tahoe Workshop, Lake Tahoe, California

Mesh Edge Detection

Andreas Hubeli, Kuno Meyer, Markus Gross
Department of Computer Science, ETH Zurich, Switzerland

Abstract

We present a framework to extract line-type features from unstructured two-manifold meshes. Our method computes a collection of piecewise linear curves describing the salient features of the mesh, such as edges and ridge lines. Our algorithms are semi-automatic, that is, they require the user to input a few control parameters and to select the operators to be applied to the mesh.

Our mesh edge detection algorithm can be used as a preprocessor for a variety of applications including mesh fairing and smoothing.

1 Introduction

Recent advances in acquisition systems have resulted in the ready creation of very large, densely sampled surfaces, usually represented as triangle meshes. The impossibility of real-time interaction with these large models has motivated many researchers in the computer graphics community to design advanced mesh processing methods including subsampling, restructuring, fairing and others. The early approaches, such as the *vertex removal* algorithm of W. Schröder [7] or the *progressive mesh* algorithm of H. Hoppe [4], use local error norms to construct multiresolution approximations of meshes by iteratively removing information from the input mesh.

More recent representations are based on the generalization of fairing techniques from signal processing [8], [6], [5]. They use multiresolution algorithms to improve the mesh approximation. In this paper we investigate a related problem: *mesh edge detection*. Our goal is to extract line-type features from meshes which can be used to construct more sophisticated multiresolution representations. The most important advantage of using line-type features is that we can force fairing algorithms to retain feature information, such as sharp edges or ridge lines - much in the same way it was accomplished for subdivision surfaces. We achieve this goal by generalizing well known computer vision techniques, such as [1], to meshes with arbitrary connectivity.

The paper is organized as follows: in section 2 we present the first major component of our framework, the set of classification operators. In section 3 we introduce the second component of our framework, the detection operators. In section 4 we describe some of the experimental results we obtained using this technique. Finally, we discuss some work in progress.

E-mail: hubeli@inf.ethz.ch
meyerk@student.ethz.ch
grossm@inf.ethz.ch

Address: Department of Computer Science
ETH Zentrum
CH - 8092 Zurich

2 Classification Phase

The classification operators basically assign a *weight* to every edge in the mesh. The weight is proportional to the importance of the edge: ideally, edges close to or on line-type features should be assigned large weights whereas all remaining edges should get small weights. The weights will then be used to extract the subset of the “most important” edges, the so-called *feature edges*. They are employed in the detection phase to construct the line-type mesh features.

In the following subsections we will describe some of the classification operators we designed and tested.

2.1 Second Order Difference (SOD)

This most simple operator assigns a weight to every edge in the mesh proportional to the dihedral angle defined by the normals of its two adjacent triangles. The idea is similar to the second order difference operator constructed by Guskov et al. in [2] which was used to fair meshes of arbitrary connectivity. The operator, described by equation (1), is locally bound and can be evaluated efficiently.

$$w(e) = \cos\left(\frac{\boldsymbol{n}_i}{\|\boldsymbol{n}_i\|} \cdot \frac{\boldsymbol{n}_j}{\|\boldsymbol{n}_j\|}\right)^{-1} \qquad (1)$$

$\boldsymbol{n}_i$ and $\boldsymbol{n}_j$ correspond to the normals of the two triangles that share edge e.

This technique is well suited for coarse, pre-optimized meshes. SOD however, performs poorly on very smooth or noisy meshes, since all computations are carried out within a small region of support.

The results in figure 5.a were generated using this operator.

2.2 Extended Second Order Difference (ESOD)

With only a little effort we can build a simple extension to the previous operator. Instead of using the normal of the two neighboring triangles, we take the average normals computed from the one-ring of the vertices opposite to the edge and apply them to equation (1). This is shown in figure 1.

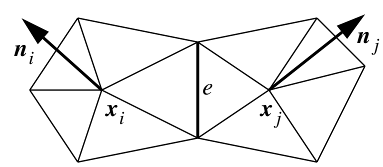


Figure 1: Support of the extended second order difference operator

The increase in the size of the support has the expected impact: the influence of noise on the classification process is significantly attenuated. However, as a drawback, ESOD does not perform that well on coarse meshes.

2.3 Best Fit Polynomial (BFP)

In this approach all the vertices used to evaluate the classification operator on an edge are projected onto a two-dimensional parameter plane. The projected vertices are then interpolated with a best fit polynomial $p(u)$ of degree n. Finally the curvature of the (planar) polynomial is evaluated at the edge position e, as described by equation (2):

$$w(e) = p''(e) \tag{2}$$

The major difficulties of this approach are the definition of the parameter plane and the proper projection of the initial vertices from 3-space. An intuitive definition of the parameter space is given in figure 2:

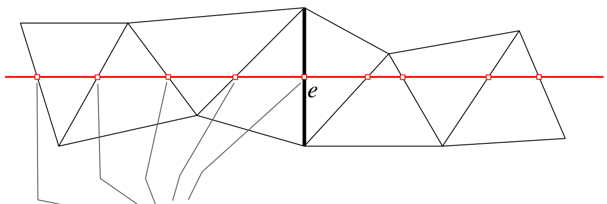


Figure 2: Illustration of the parameter plane used for the BFP method (top view)

We propose to set the parameter plane to be perpendicular to the edge being considered. In addition, the midpoint of the edge is defined to lie on the plane. The points used in the best fit process are computed from the intersection of the plane with a set of neighboring triangle edges. The most important advantage of this strategy is that the support of the operator can be chosen freely. That is, we can even adapt it locally for each edge. Furthermore, the degree of the fitting polynomial can be adjusted to the size of the support.

Both figure 5.b and figure 5.c were generated with variations of this operator.

2.4 Angle Between Best Fit Polynomials (ABBFP)

The last operator is an extension to the previously introduced polynomial fit. As in the previous case, polynomials are fitted through the parameter plane of every edge. This operator actually fits two polynomials: one for the vertices that lie on one side of the edge, and one for the vertices lying on the other. The weight assigned to the edge is then chosen to be proportional to the angle between the two curve tangents evaluated at the edge position. It yields according to equation (3):

$$w(e) = \cos\left(\frac{(1, p_l'(e))}{\|(1, p_l'(e))\|} \cdot \frac{(1, p_r'(e))}{\|(1, p_r'(e))\|}\right)^{-1} \tag{3}$$

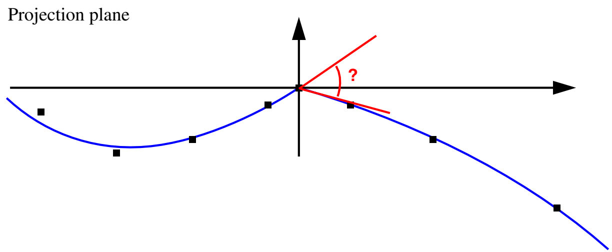


Figure 3: ABBFP operator

All the results in figure 6 were generated using this operator and with support sizes ranging from 0.25% (figure 6.a) to 4% (figure 6.c) of the size of the object bounding box.

3 Detection Phase

As previously discussed, the classification operators are used to assign a weight to every edge in the mesh. The larger the weight, the “more important” the edge. In a second phase, proper feature lines need to be constructed. This is accomplished in three steps:

- First, a subset of *feature edges* is constructed. Only edges in this set will be further considered to compute the final set of line-type features.
- Next, the feature edges are clustered into patches. These patches define mesh regions where line-type features are present.
- Finally, the *line-type features* are extracted using a skeletonizing algorithm that reduces the patches to piecewise linear curves.

3.1 Selection of Feature Edges

This process is heavily mesh and user dependent, since the number of feature edge candidates depends both on the size and geometry of the mesh and on the user’s intention. Hence, we require the user to select appropriate threshold values. We propose the following two different strategies for thresholding:

Standard Thresholding

The most simple thresholding approach analyzes every edge separately. Based on a user-provided parameter it decides whether an edge is a feature edge. The user can specify the threshold parameter both in percentage of edges that must be preserved, or as the minimal weight that an edge must have to be included into the subset of feature edges.

Hysteresis Thresholding

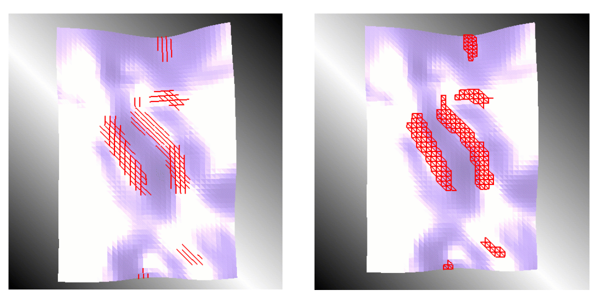
This type of approach uses two threshold values serving as a lower and an upper bound of a hysteresis. If the weight associated with an edge is larger than the upper value, the edge is automatically defined as a feature edge. If the weight is smaller than the lower bound, then the edge is automatically discarded. The remaining edges, whose weights lie between the lower and upper bounds, are only defined as feature edges if one or more of their neighbors are feature edges. The advantage of this approach is that it constructs smoother patches in regions where feature edges are present.

3.2 Construction of the Patches

The patches are generated from the subset of feature edges using the following approach:

- every edge that shares *both* of its vertices with feature edges is inserted into the subset of feature edges.

As an example consider the figure 4. The results illustrated in figure 4.b demonstrate that the patches are filled with feature edges and that the size of the patches is clearly bound.



a) b)

Figure 4: Construction of the patches:
a) Subset of feature edges computed by the hysteresis thresholding scheme
b) Set of patches generated using the method of section 3.2

3.3 Skeletonizing

As described the output of the previous step is a set of patches. As such, a patch is a collection of edge features describing the neighborhood of one or more line-type features. The final step of the detection phase is to reduce these patches to a set of line-type features. We accomplished this using the following two skeletonizing strategies:

Triangle Based Skeletonizing

This approach reduces patches to lines of a thickness smaller or equal to 2 by removing triangles from the patch - one at a time. In order to get the correct result, we have to impose constraints on the removal strategy. The drawback of this approach is that it might break up larger line-type features into multiple disconnected segments. The advantage is that can be computed very efficiently. A more detailed description has to be omitted for brevity.

Vertex Based Skeletonizing

In the second approach, we use potential fields to remove vertices from the patch. In a first step, a potential field is defined for every vertex whose strength is proportional to its distance from the patch boundary. Next, we discard all the vertices that have at least one neighbor with a larger potential. As a result we obtain a surviving subset of disconnected vertices. These vertices are subsequently glued together using certain assumptions on the potential fields. Again, further details have to be omitted.

4 Results

In this section we will present and discuss experimental results obtained on different meshes. We used both well known meshes, such as the bunny, mannequin, motor part, and the golf club, as well as a simple, smooth geological surface. Most of the meshes have an arbitrary connectivity, and they exhibit some sharp line-type features. The geological surface is the only exception, being a regular grid and not having any salient features.

In the first sequence of pictures presented in figure 5 we compare some of the operators defined in section 2. On this mesh we obtained the best results by using operators with small support. This is due to the fact that the model is only sparsely sampled. As a consequence, operators with large support include information that is geometrically too far away from the edge. This problem was addressed by constraining the support of the BFD operators. Both the SOD operator and the BFP operators generated good results.

In the second sequence illustrated in figure 6 we capture the influence of the support of the ABBFP operator. From left to right we show the influence of increasing the support from 0.25% to 4% of the size of the bounding box. In figure 6.a the support is too small, and as a consequence, the noise of the mesh corrupts the performance of the classification operator. At the other extreme, in figure 6.c the support of the operator is too large. Hence, the marked edges are clustered around only a few of the prominent mesh features. The best results have been achieved by setting the support to an intermediate value of 2%, as shown in figure 6.b.

Finally, figure 7 depicts the results obtained on different meshes by combining the classification and detection operators appropriately. Interestingly, the line-type features of the head mesh of figure 7.a include the outlines of the eyes, mouth, ears, nose and eyebrows - as we would expect it from a feature extraction algorithm. The features of the golf club mesh of figure 7.b have been recognized correctly, and all the sharp edges have been captured. Our last image in figure 7.c shows a geological surface. Although the surface does not contain any salient edges, our framework extracted a few features. By comparing these results to the actual structure of the surface, we found that the algorithm extracted local ridges and valley lines in the mesh.

5 Conclusions and Future Work

In this paper we presented a framework for the detection of line-type features in meshes of arbitrary connectivity. We proposed a two-stage process consisting of a *classification phase* and a *detection phase*. In order to handle a variety of different meshes, we provide a set of operators with different properties.

As already mentioned, the user must select the operators as well as a few parameters for the classification and detection steps. As such, the process is not fully automatic. It still requires manual assistance and tuning for optimal performance. We do not consider this as a major drawback, since all algorithms can be computed efficiently.

The presented results are encouraging, and we expect some major improvements in the near future. The most important ones include:

- Computational efficiency: although the operators are fairly efficient, it still takes a few seconds (up to a few minutes for very large meshes) to compute the features. This is certainly a drawback for interactive applications where a user might try different operators to optimize the results.
  We will tackle this problem by designing a multiresolution representation of the mesh. Since the multiresolution representation effectively strives towards maintaining model features, we can identify efficiently which edges need to be considered for the computation.
- Improved classification strategy: the most important problem encountered in the classification phase is that features are not necessarily high frequency information. This is well known from image edge detection. Although, intuitively, we tried to capture low-frequency information by extending the support of the operators improvements might be obtained by mesh decomposition [3].
  The advantage of such settings is that one could start the classification process on a coarse, low-frequency approximation and progressively improve the results while refining the mesh.
- For computational efficiency, our current skeletonizing strategies are based on topological distance. Taking geometric distances instead might further improve the performance of the framework.
- Our final goal is, of course, to provide a mesh analysis sophisticated enough to automatically determine the optimal operators and parameters.

Literature

[1] J. Canny. "A computational approach to edge detection." In IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-8, No. 6, pages 679-696, Nov. 1986.

[2] I. Guskov, W. Sweldens, and P. Schröder. "Multiresolution signal processing for meshes." In *SIGGRAPH '99 Proceedings*, Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH, ACM Press, Aug. 1999.

[3] M. Gross and A. Hubeli. "Eigenmeshes." Technical Report ETH Zurich, Mar. 2000.

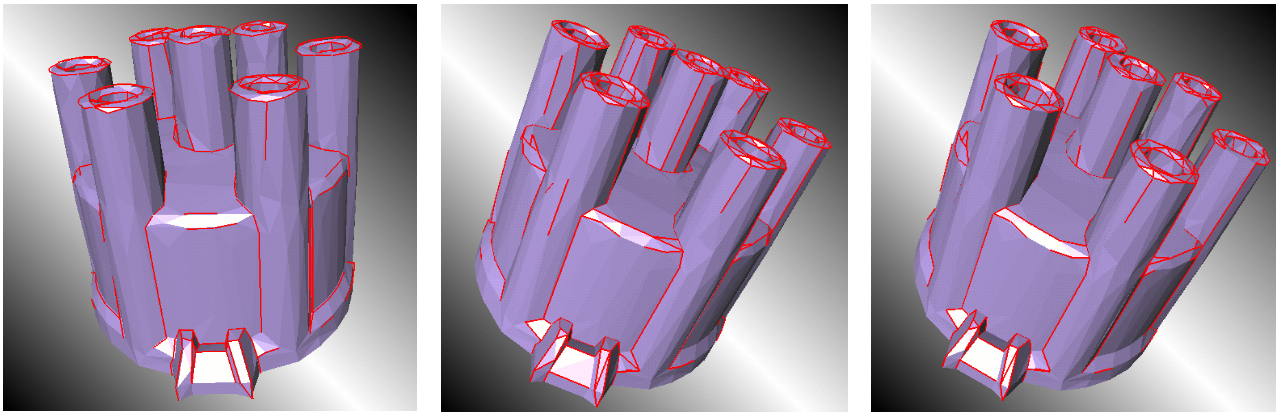
[4] H. Hoppe. "Progressive meshes." In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, Aug. 1996.

[5] A. Hubeli and M. Gross. "Fairing of non-manifolds for visualization." In Visualization 2000 Proceedings, Oct 2000.

[6] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. "Interactive multi-resolution modeling on arbitrary meshes." In M. F. Cohen, editor, *SIGGRAPH 98 Conference proceedings,* Annual Conference Series, pages 105–114. ACM Press and Addison Wesley, July 1998.

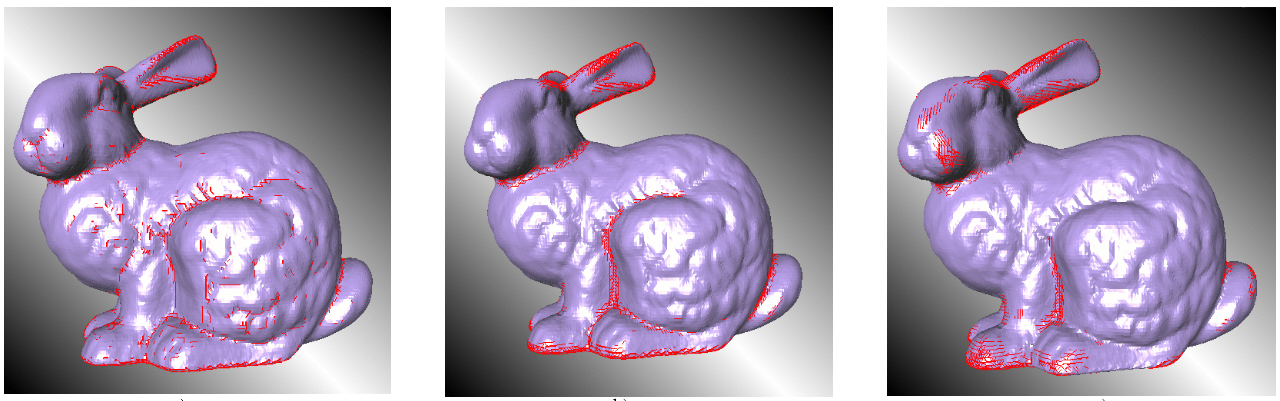
[7] W. Schröder, J. Zarge, and W. Lorensen. "Decimation of triangle meshes." In SIGGRAPH 92 Conference Proceedings, Annual Conference Series, pages 65–70, July 1992.

[8] G. Taubin. "A signal processing approach to fair surface design." In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 351–358. ACM SIGGRAPH, Addison Wesley, Aug. 1995.



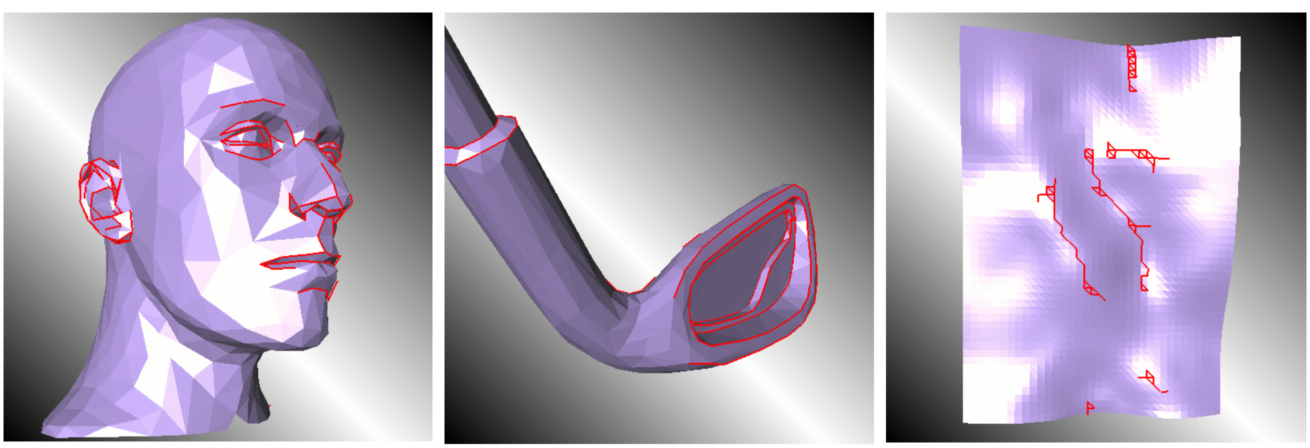
a) b) c)

Figure 5: Results computed on a motor part using different operators:
a) Line-type features computed using the SOD operator
b) Line-type features computed using an *extension to the BFP* operator
c) Line-type features computed using an *extension to the BFP* operator



a) b) c)

Figure 6: Results computed on the bunny model using the *ABBFP* operator:
a) Edges detected using a support of 0.25% of the size of the object bounding box
b) Edges detected using a support of 1.0% of the size of the object bounding box
c) Edges detected using a support of 4.0% of the size of the object bounding box



a) b) c)

Figure 7: Mesh edge detection applied to other models:
a) A mannequin head. Feature lines detected: eyes, nose, lips, ears, eyebrows
b) A golf club. All the sharp features have been detected correctly
c) A geological surface. Some interesting feature-lines could be extracted from this very smooth surface