Sharpen&Bend: Recovering curved edges in triangle meshes produced by feature-insensitive sampling

Marco Attene, Bianca Falcidino, Michela Spagnuolo, IMATI – GE / CNR
Jarek Rossignac, College of Computing and GVU Center, Georgia Institute of Technology

---

Various 3D acquisition, analysis, visualization and compression approaches sample surfaces of 3D shapes in a uniform fashion, without any attempt to align the samples with the sharp edges and corners of the original shape. Consequently, the interpolating triangle meshes chamfer these sharp features and thus exhibit a relatively large error in their vicinity. We introduce here two new filters that restore a large fraction of the straight or curved sharp edges missed by feature-insensitive sampling processes: (1) **EdgeSharpener** restores automatically the sharp edges by identifying and splitting the chamfer edges and by forcing the new vertices to lie on intersections of planes extending the smooth surfaces incident upon these chamfers and (2) **Bender** subdivides the resulting triangle mesh using a combination of the Butterfly subdivision scheme, for the smooth portion of the mesh, with a four-point subdivision scheme, for the sharp edges, in order to preserve the sharpness of the recovered sharp edges while bending their polyline approximations into smooth curves. This combined post-processing (named **Sharpen&Bend**) significantly reduces the error produced by feature-insensitive sampling processes. For example, we have observed that the L^2 error introduced by the SwingWrapper remeshing-based compressor can often be reduced down to a fifth by executing EdgeSharpener alone after decompression, with no additional information. For meshes produced by retiling shapes with curved edges, this error may be further reduced by two thirds if we follow the EdgeSharpening phase by Bender. Thus the combined Sharpen&Bend process takes a triangle mesh produced by a feature-insensitive sampling of a curved shape with sharp curved features and automatically refines the mesh to produce a more accurate approximation of the initial shape.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling – *Boundary representation, Geometric algorithms, languages and systems*

General Terms: Algorithms

Additional Key Words and Phrases: sharp feature, subdivision surface, remeshing

---

1. INTRODUCTION

The surfaces of 3D models are often represented by approximating triangle meshes. Their triangles are the simplest form of interpolant between surface samples, which may have been acquired with a laser scanner [Attene and Spagnuolo 2000; Amenta et al. 2001; Giesen and John 2002], computed from a 3D scalar field resolved on a regular grid [Lorensen and Cline 1987; Bloomenthal 1988], or identified on slices of medical data [Cheng and Dey 1999; Cong and Parving 2001]. Most acquisition techniques restrict each sample to lie on a specific curve whose position is completely defined by a pre-established pattern. For example, a laser-scanner measures distances along a family of parallel or concentric rays that form a regular pattern or grid. One may also argue that an iso-surface extraction uses three such patterns, aligned with the three principal directions. Because the pattern of these rays or stabbing curves is not adjusted to hit the sharp edges and corners of the model, almost none of the samples lie on such sharp features. Therefore, the sharp edges and corners of the original shape are removed by the sampling process and replaced by irregularly triangulated chamfers. The error between the original shape and the approximating triangle mesh may be decreased by using a finer sampling step. But, over-sampling will increase significantly the number of vertices, and thus the associated transmission and processing cost. Furthermore, as observed by Kobbelt et al. [2001], the associated aliasing problem will not be solved by over-sampling, since the surface normals in the reconstructed model will not converge to the normal field of the original object.

This work is part of the bilateral research agreement "Surface Analysis" – GVU/GATECH and IMATI-GE/CNR. IMATI-GE was partially supported for this work by the national FIRB project MACROGeo. Rossignac's work on this project was partly supported by a DARPA/NSF CARGO grant #0138420.

Authors' addresses:
M. Attene, B. Falcidieno, and M. Spagnuolo, Istituto per la Matematica Applicata e Tecnologie Informatiche, IMATI – CNR, Via De Marini 6, Torre di Francia, 16149 Genova, Italy; email: {attene, falcidieno, spagnuolo}@ge.imati.cnr.it;
J. Rossignac, IRIS cluster, College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, NW, Atlanta, GA 30332-0280; email: jarek@cc.gatech.edu.

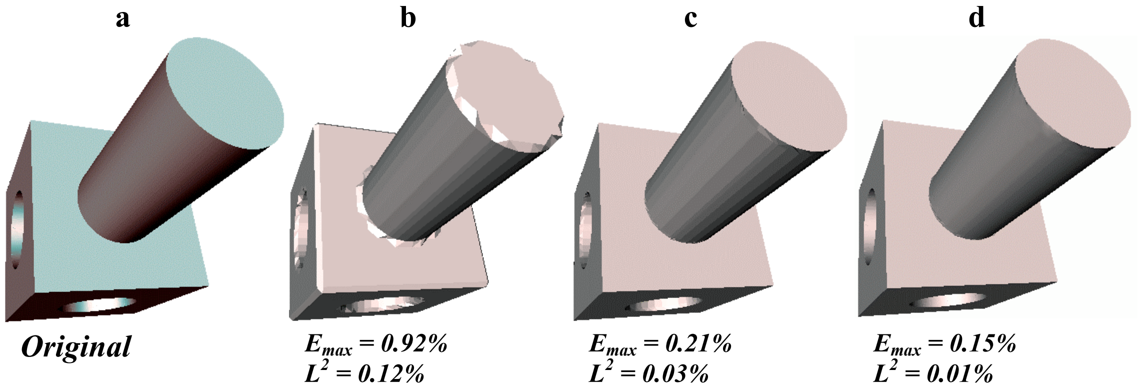

Figure 1: An original model (a) was re-meshed through a feature-insensitive algorithm (b). The sharp edges and corners were restored by EdgeSharpener (c). Then, Bender faired the smooth regions without rounding off the sharp features reconstructed by EdgeSharpener (d). For each model, the maximum distance from the original surface (E_{max}) and the mean-squared distortion (L^2) are reported. All the values are percent of the bounding-box diagonal.

Similar aliasing artifacts can be observed on models produced by surface remeshing, which is the basis of three of the most effective compression techniques published recently [Attene et al. 2003b; Szymczak et al. 2002; Guskov et al. 2000]. All three methods create a new mesh that approximates the original one. Vertices of the new mesh are placed on the original surface or at quantized locations near the surface, so that their position can be predicted more accurately and encoded with fewer bits. To reduce the encoding of each vertex to a single parameter, the vertices of the resampled mesh are restricted to each lie on a specific curve, which is completely defined by previously processed neighboring vertices. Unfortunately, almost none of the new vertices fall on sharp edges or corners. As a consequence, the sharp features are not captured in the new mesh and a significant error between the original surface and its approximating triangle mesh occurs near these sharp features.

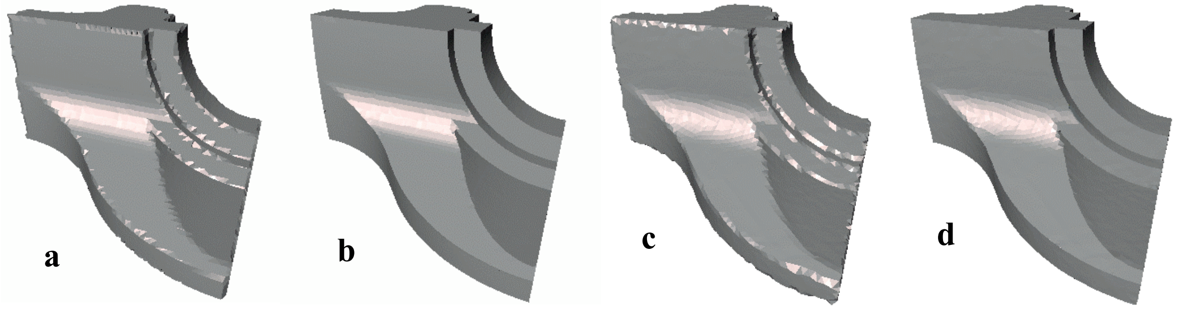

Figure 2: An aliased model (a) generated with Marching Cubes is improved automatically (b) by EdgeSharpener. A version (c) of the original model was generated through the lossy SwingWrapper compression [Attene et al. 2003b]. An improved version (d) was obtained with no additional information by running EdgeSharpener after decompression.

To reduce this error, one may choose to use a feature-sensitive remeshing process [Lee et al. 1998], which attempts to place the samples on the sharp edges and corners of the original model. Unfortunately, this solution requires a more verbose representation of the samples, which are no longer restricted to each lie on a specific curve and hence must be encoded using 3 coordinates each. In order to retain the full benefit of the compactness of a feature-insensitive retiling while reducing the approximation error, we present in this article the **EdgeSharpener** approach, which has been previously introduced in a conference publication [Attene et al. 2003a]. It automatically identifies the chamfers and replaces them with refined portions of the mesh that more accurately approximate the original shape, restoring a **piecewise linear approximation of the sharp edges**.

A significant number of publications have been focused on identifying sharp features in a 3D model [Hubeli et al. 2000; Hubeli and Gross 2001], even in the presence of noise. More recently, solutions were proposed for maintaining sharp features during remeshing [Kobbelt et al. 2001; Vorsatz et al. 2001]. In both cases, the features to be extracted or preserved are present in the model. In contrast to this body of previous work, the EdgeSharpener solution proposed here recovers sharp features in an aliased model, from which they have been removed by feature-insensitive retiling. Our edge-sharpening process works well for meshes generated through a variety of uniform sampling schemes and does not introduce undesirable side effects away from sharp features. For example, Figure 2 shows a model resampled with two different feature-insensitive approaches. Its sharp features have been correctly restored in both cases using EdgeSharpener.

After the sharp edges have been restored by EdgeSharpener, the error between the triangle mesh and the original model is distributed more uniformly and accounts for the difference between the original curved surface and its piecewise-linear approximation. When the original surface is smooth everywhere, the error may be further reduced by subdividing the approximating triangle mesh. An interpolating subdivision process [Dyn et. al 1990; Zorin et al. 1996] may be used to refine the triangle mesh globally, bending the triangles to smoothen the surface at the edges and vertices. Unfortunately, when the original model contains sharp edges, such a bending process would round or blend the sharp edges restored by EdgeSharpener, and hence would increase the error, annihilating the benefits achieved by EdgeSharpener. Thus, we introduce here a modified version of the Butterfly subdivision approach, which preserves the sharpness of the features restored by EdgeSharpener, while bending them so that they form smooth curves between sharp corners. This new refinement scheme, called **Bender**, uses the modified Butterfly subdivision scheme introduced in [Zorin et al. 1996] for all edges that are not incident upon a vertex bounding a sharp edge. The other edges, which include the sharp edges and the edges adjacent to sharp edges, are subdivided using the subdivision rules described in [Zorin and Schröder 2000], which are also used for subdividing possible boundaries. The combined effect of both Edge-Sharpener and Bender (abbreviated Sharpen&Bend) is shown in Figure 1.

An important application of Sharpen&Bend is the post-processing of laser-digitized models. Most surface reconstruction approaches, in fact, are not able to correctly reconstruct sharp features. Moreover, while sufficient sampling conditions have been studied for smooth 3D objects [Amenta et al. 2001], a guaranteed-quality reconstruction of surfaces with sharp features has remained a challenge, even in the 2D case [Dey and Wenger 2001].

The remainder of the paper is organized as follows: Section 2 outlines how existing resampling and subdivision methods tackle the problem of extracting or maintaining sharp features; in Section 3 the EdgeSharpener filter is described and several particular cases are discussed; Section 4 describes how the sharp edges tagged by EdgeSharpener are preserved and subdivided during our Bend process; Section 5 reports the results of our tests on typical models in various application fields; finally, in Section 6 we summarize our contributions and conclude the paper.

2. RELATED WORK

The problem of extracting sharp feature lines from 3D data has been studied extensively and remains an important issue for several applications, the most important being reverse engineering [Thompson et al. 1999]. The source of the data may be *unstructured*, as in the case of scattered point sets, *partially structured* as in the case of contours or profiles, or *structured* as in the case of polygonal meshes. We discuss them in this order. Then, we discuss feature-sensitive polygonization and re-tiling approaches that identify sharp features in the original surface and preserve them in the triangulated model.

2.1 Identifying sharp features in unstructured point clouds

When a point cloud is dense enough, sharp features may be inferred by analyzing the neighborhood of each point. Usually, this is done as a post-processing step of finding a manifold triangle mesh interpolating the point cloud [Adamy et al. 2000]. In [Gumhold et al. 2001], however, a method is described in which points that lie close to the sharp features can be identified directly from the cloud, without the need of a prior surface reconstruction. The point cloud is partially organized through a neighbor graph and penalty weights are assigned to the points depending on the configuration of the neighbors. A weight indicates the unlikelihood that a point is part of a sharp edge, thus a point which is coplanar with all its neighbors will be strongly penalized. Then, a sub-graph is extracted based on a minimization of the weights. Even after a proper pruning of this sub-graph, which represents a rough approximation of the final set of feature lines, the edges are likely to form zig-zag patterns. This happens because the input samples are rarely aligned with the sharp edges, and the sub-graph is made of arcs which cross these sharp edges. The authors propose to apply a final procedure that smoothens the zig-zag away by fitting low degree splines to the feature lines. This eliminates the zig-zag, but the resulting feature lines are likely to lie on what we call *chamfers*, rather than at the intersection of extrapolated surfaces. In practice, if the original solid has a convex sharp edge, the corresponding feature line extracted by this method is likely to be entirely inside the solid; if the original edge is concave, the feature line is probably outside.

In [Guy and Medioni 1997], the authors present a method to infer smooth surfaces, while detecting intersection between surfaces and 3D junctions by applying perceptual grouping rules. The input points are each associated with a correspondent surface normal. If normals are not provided, the method relies on an automatic normal estimation. Since, in this method, sharp edges are computed as intersections of surfaces, the result may be inaccurate if the normals are not provided. In general, since one cannot assume that enough samples fall exactly on sharp edges, the normals at samples near the edges will be polluted, and so will be the resulting intersection of the surfaces meeting at the edges.

These two approaches work well for dense point clouds and when no extreme accuracy is necessary in their output. Conversely, the Edge-Sharpening approach described here works well even if the vertices of the input triangle mesh are sparse. Moreover, each sharp edge reconstructed by our method is computed by extrapolating the smooth surfaces which meet at the sharp edge, hence one can expect to obtain more accurate results.

2.2 Recovering sharp features in scanned data sets

In most 3D data acquisition processes, the cloud of points captured by a scanner is organized as uniformly spaced samples along a series of nearly parallel rows. The spacing of the rows and of the points along a row is uniform in the scanner's parameter space (an angle that controls the orientation of the laser beam), but not in terms of Euclidean distance. Sharp features are typically extracted by detecting curvature extrema along each row and by matching them between successive rows [Biasotti et al. 2000]. Similarly, if the points are known to be captured along straight profiles, as in typical bathymetries, it is possible to join close vertices of adjacent profiles into feature lines by analyzing, and possibly matching, the shape of their neighborhood along the profile [Patané and Spagnuolo 2002].

Clearly, one could adapt the above approaches to triangle meshes by computing regular cross-sections and doing the 1-D analysis to find matching points. However, the cross-sections will typically not go through the sample points and hence would add sampling noise and reduce the reliability of the approach.

2.3 Recovering sharp edges in Triangle Meshes

In most situations, surface samples are sparse and the surface that interpolates them is defined by a triangle/vertex incidence graph. In these cases, the additional information given by the connectivity graph lead to significantly better sharp-edge recovery results when compared to methods dealing with sparse clouds of unstructured points. In [Hubeli and Gross 2001], for example, a method is described for extracting a multi-resolution organization of sharp edges from a triangle mesh. The method is based on the assignment of a weight to each edge, so that the weight is proportional to the dihedral angle, or to some measure of the dihedral angle which uses a bigger support. Then, the heaviest edges are used to form patches of the surface which are thinned through a skeletonization process. Since this process may become slow for dense meshes with many sharp features, the input triangulation is first turned into a progressive mesh [Hoppe 1996], and the feature extraction operates on the coarsest mesh. Higher resolution features are obtained by inverting the edge-contractions through vertex-splits, as described in [Hoppe 1996], while keeping track of the features.

This approach, however, may result in the identification of a set of lines corresponding to small radius blends in the input model, and it may prove difficult to tune the parameters in order to get only the features which are actually sharp.

The same difficulty prevented us to use the method described in [Watanabe and Belyaev 2001], in which the identification of perceptually salient curvature extrema was used to detect curvature features. A curvature feature is a portion of the model which has an extreme value of curvature in some direction, and is computed by grouping triangles with similarly curved neighborhoods. A thinning algorithm can be used to turn the curvature features into feature lines approximating small radius blends in the model.

2.4 Feature sensitive (re)meshing

Feature sensitive sampling techniques have been mainly developed for iso-surfaces and for polygonal meshes. When the model being sampled is an iso-surface and the resulting model interpolates the samples through a polygonal mesh, the process is called *feature sensitive meshing, tiling,* or *polygonization*. In the particular case where the input model is already a triangle mesh, the process is called *feature sensitive re-meshing* or *re-tiling*.

The loss of sharp features during the polygonization of iso-surfaces has been addressed in [Othake and Belyaev 2002; Othake et al. 2001], where the standard marching-cubes algorithm is improved by optimizing the location of sample points so as to snap some of them onto sharp features. In [Othake and Belyaev 2002], the initial mesh produced by marching-cubes is optimized by forcing its triangles to become tangent to the iso-surface. Such a constraint automatically eliminates the chamfers by moving each of their triangles to either one or the other side of the sharp edge. Similarly, in [Othake et al. 2001] each vertex is iteratively moved so that the normals at the triangles incident to the vertex converge to the normals of the underlying iso-surface. In a similar fashion, when remeshing an original triangulation, the aliasing problem may be avoided by snapping some of the evenly distributed vertices to sharp feature lines, as proposed in [Vorsatz et al. 2001].

During the triangulation of an iso-surface, an *extended* marching cubes (EMC) algorithm [Kobbelt et al. 2001] derives vertex normals from the original scalar field and uses them to decide whether a voxel contains a sharp feature. If so, additional vertices are created in the voxel and placed on intersections between planes defined by the vertices and their normals.

This EMC approach was subsequently improved in [Losasso et al. 2002], enabling it to accurately polygonize models with sharp features using an adaptive subdivision of the space (i.e. an octree), with the result of obtaining polygonal models with less faces.

These feature-sensitive surface triangulation approaches exploit information about the original surface. In contrast, the EdgeSharpener solution proposed here operates on a triangle mesh produced by a feature-insensitive sampling and yet is able to restore most of the sharp features automatically, without additional information. One may argue that an application of the EMC to a polygonal mesh may be used to infer and hence reconstruct the sharp features. In [Kobbelt et al. 2001], such an application (i.e. a remeshing) is discussed and, in fact, it is useful to improve the quality of meshes having degenerate elements or other bad characteristics. In some cases, the information at the edge-intersections makes it possible to reconstruct sharp features in an Edge-Sharpener like manner. For example, if a cell contains an aliased part that does not intersect the cell's edges, the normal information at the intersections is used to extrapolate planes and additional points are created on the inferred sharp feature. If, on the other hand, the cell's edges do intersect the aliased part, the normal information becomes noisy, and nothing can be predicted about any possible feature reconstruction. In contrast, our approach for the construction of the extrapolated planes makes EdgeSharpener less sensitive to such problems. Moreover, remeshing the whole model through the EMC approach can introduce an additional error on the regions without sharp features. Conversely, the local modification we propose only affects the aliased zones by subdividing only the triangles that cut through the original solid (or through its complement) near sharp edges.

2.5 Feature preserving subdivision and smoothing

The problem of preserving sharp features during the subdivision of polygonal surfaces has been tackled in [Hoppe et al. 1994], where the authors use a modification of the Loop's subdivision scheme [Loop 1987] to improve the quality of the results of their surface reconstruction algorithm. In their approach, after a coarse reconstruction which interpolates the input point cloud with a triangle mesh, edges of the mesh are tagged as *sharp* if their dihedral angle exceeds a threshold or if they lie on the boundary. Then, the mesh is used as the input to a subdivision process that generates a piecewise smooth subdivision surface fitted to the data through an iterative optimization process. The optimization computes an approximation of some limit points of the surface and transforms the base domain so that these points best fit the input data with respect to an energy function. All the modifications applied to the base domain preserve the tagged edges. For example, if a tagged edge is split, then the resulting two edges connecting the old end-points with the newly inserted vertex are tagged as well. The result of this process is a tagged base domain which can be subdivided through a modification of the Loop's subdivision scheme which preserves the tagged features. Since the subdivision scheme is not interpolating, a trade-off between conciseness and fit to the data is necessary.

In a different approach, when the sharp features are *selected* on a quad-mesh by the user, modified subdivision rules may be used to subdivide the mesh in order to obtain sharp features in the limit surface [Biermann et al. 2001]. This is particularly useful for multiresolution editing purposes where, in order to put a curved sharp edge on the limit surface, the user can simply *draw* a piecewise linear curve on the base domain. Then, this curve will be subdivided through the modified rules that guarantee its eventual sharpness.

The **Bender** algorithm introduced in this paper subdivides triangle meshes. It is inspired by ideas developed in [Hoppe et al. 1994]. However, our feature preserving subdivision scheme is interpolatory.

When the input triangle mesh interpolates noisy samples, subdivision may have no benefit. Instead, a smoothing process may be needed. Recent feature-preserving techniques for mesh smoothing [Jones et al. 2003; Fleishman et al. 2003] propose a penalty function that is based on the distance between a sample P and a tangent plane through sample Q to diminish the influence of P on Q when the two are separated by a sharp edge.

3. THE EDGE-SHARPENER ALGORITHM

The errors produced by feature-insensitive sampling approaches are concentrated in what we call **chamfer triangles**, which cut through the solid near sharp convex edges or through the solid's complement near sharp concave edges. Our objective is to identify these chamfer triangles and to replace them with a finer triangle mesh portion that better approximates the sharp features of the solid.

In order to preserve the integrity of the triangle mesh, we subdivide the chamfer triangles by inserting new vertices on edges between two chamfer triangles and also inside the corner triangles where several sharp features meet. Our approach, which does not split the edges that separate chamfer and non-chamfer triangles, involves three parts:

- Identify the chamfer edges and corner triangles
- Subdivide them by inserting new vertices,
- For each newly inserted vertex, estimate the sharp edge or corner that we are trying to restore and snap the vertex onto that sharp edge or corner.

3.1 Identification of the chamfer triangles

We have explored three different approaches (global, local and filter) for identifying the chamfer triangles. They produce nearly equivalent results, but offer different compromises between elegance of the formulation, code simplicity, and running time efficiency. To provide the reader with a more global perspective, we briefly outline all three before discussing the details of the filter approach that we have retained. All three approaches identify what we call **chamfer edges**, which are shown in blue in Figure 3, and all of them are based on the identification of **smooth edges**, which are the edges that tessellate smooth portions of the original model and are identified using the following simple heuristic.

In the remainder of the paper, an edge is said to be **smooth** if the angle between the normals to its two incident triangles is less than a given threshold, which we have chosen to be twice the average of such angles for the entire mesh. This choice is motivated by the following consideration: when an original piecewise smooth model is sampled with a nearly infinite density, the dihedral angle at edges not belonging to chamfer triangles is nearly □. Furthermore, the number of non-smooth edges is negligible with respect to the total number of edges, thus the average dihedral angle remains close to □ or, equivalently, the average angle, □, between the normals of two adjacent triangles remains close to 0. The influence of non-smooth edges on □ is small but not null, thus the actual angle for smooth edges is slightly smaller than □. In practice we do not have infinite samplings, so taking □ as threshold makes the algorithm too sensitive to small amounts of noise. We have experienced that doubling □ is a good compromise between theoretical correctness in the ideal case and robustness in all of the practical cases encountered.

The **global** approach that we have initially explored processes the entire mesh and identifies clusters of triangles that are connected by smooth edges and thus tessellate portions of smooth surfaces. **Chamfer edges** are edges that connect vertices in two clusters, and **chamfer triangles** are those bounded by one or two chamfer edges. Triangles bounded by three chamfer edges are called **corner triangles**. The global clustering phase is delicate [Garland et al. 2001], expensive, and unnecessary for our purpose. Furthermore, it does not easily differentiate between smooth surfaces and thin corridors (generalized triangle strips) of smoothly connected triangles, which although smooth, may correspond to the actual chamfers that we wish to replace.

The **local** approach, which we have also investigated and rejected, examines the neighborhood of each edge formed by its two incident triangles and by all their neighbors. It attempts to organize the ordered ring of neighbors into two or three strips of nearly coplanar and contiguous triangles, separated by chamfer triangles. If it succeeds, the edge is a chamfer edge. Although the process is local for each edge, its formulation is rather inelegant and its execution involves redundant steps. Furthermore, using only one ring of neighbors may wrongly identify chamfers in noisy regions that do not separate smooth portions of a surface.

We have finally opted for the **filter** approach, which is significantly faster, more robust, and easier to implement than the other two. This approach, which we have named EdgeSharpener, is based on the initial identification of the smooth edges and on a succession of six simple filters. Each filter colors the edges, vertices, or triangles, based on the colors of their adjacent or incident elements. (To simplify the presentation, we use colors instead of tags.)

The first step of the filter approach is to paint **brown** all of the smooth edges (we assume that all vertices, edges, and triangles are initially **gray**), then we apply the following sequence of six filters:

- *Paint* ***red*** *each* ***vertex*** *whose incident edges are all brown.* These red vertices are surrounded by a smooth portion of the surface.
- *Paint* ***red*** *each* ***triangle*** *that has at least one red vertex.* A connected component of these red triangles forms the core of a smooth region.
- Recursively paint **red** the triangles that are adjacent to a red triangle through a brown edge. Doing so extends the cores to smoothly abutting triangles.
- Paint **red** the edges and vertices of the red triangles. This filter adds to each smooth region its boundary.
- *Paint* ***blue*** *each non-red edge joining two red vertices.* Typically, these blue chamfer edges join the boundary vertices of two different smooth regions separated by a strip of chamfer triangles.
- *Paint* ***green*** *each triangle bounded by three blue edges.* These are the corner triangles where three strips of chamfer edges meet.

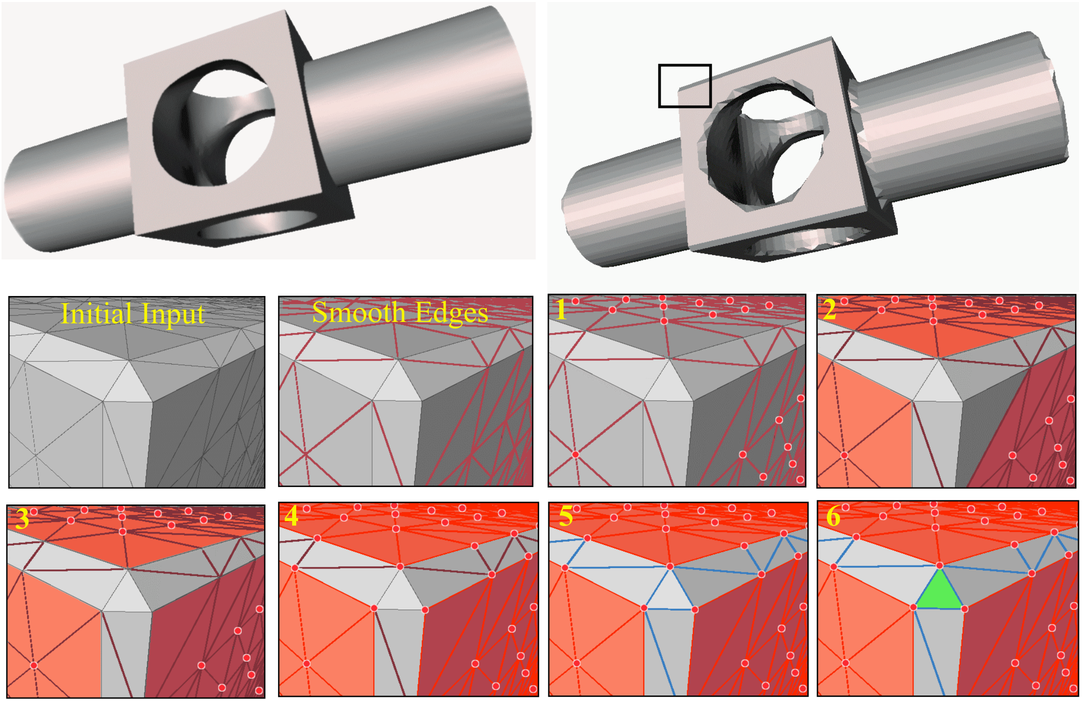

Figure 3: An original model (top-left) was re-meshed through a feature insensitive algorithm (top-right). The smooth edges in the aliased input model were detected and the six filters (1-6) have selected the chamfer edges and the corner triangles to be subdivided.

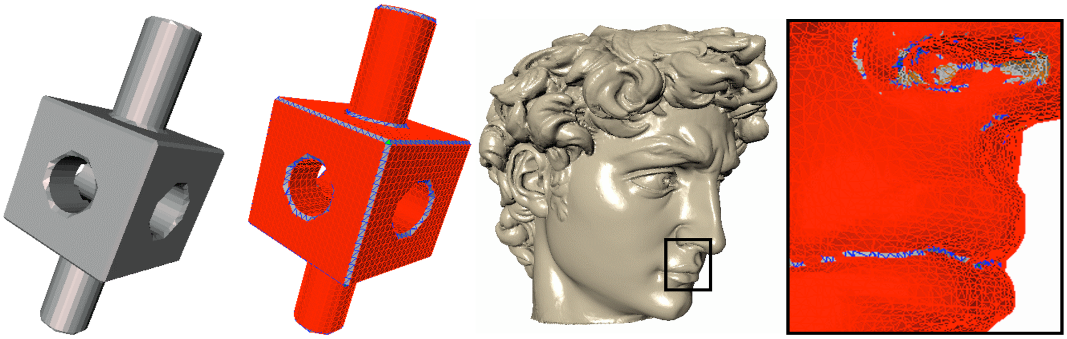

Figure 4: Chamfers identified by Edge-Sharpener on a Marching-Cubes generated model (left) and on the simplified version of a laser scanned model of Michelangelo's David (right). Some edges are still gray or brown.

The six steps are illustrated in Figure 3. **Filter 1** identifies the **interior vertices** of smooth regions. **Filter 2** identifies the **core triangles** of smooth regions. These core triangles are incident upon at least one interior vertex. **Filter 3** extends the **smooth regions** to include all of the triangles that are adjacent to a core triangle by a smooth edge. Note that we do not distinguish between the different components of the smooth portion of the mesh. **Filter 4** marks the edges that bound the smooth regions to ensure that they are not mistaken for chamfer edges in step 5. Note that these edges are **not smooth**. Filter 4 also identifies the vertices that bound the smooth regions. **Filter 5** identifies the **chamfer edges** as those that connect vertices on the boundary of smooth regions but do not bound a smooth region. Note that chamfer edges may, but need not, be smooth. Also note that some edges may still be gray and that some brown edges may neither be part of a smooth region nor be chamfer edges (Figure 4). Finally, **Filter 6** identifies the **corner triangles** that are bounded by three chamfer edges and have all of their vertices on the boundary of smooth regions. Thus, they are at the junction of at least three portions of smooth regions.

3.2 Subdivision of the chamfer triangles

To **subdivide** the chamfer triangles, including the corner ones, we insert a new vertex in the middle of each chamfer edge and in the middle of each corner triangle. Then, we re-triangulate the resulting polygons. We may have three cases (see Figure 5):

- A triangle with a single chamfer edge is split in two triangles.
- A triangle with two chamfer edges is split into three triangles.
- A corner triangle, which has three chamfer edges and an interior vertex is split into six triangles forming a fan around the interior vertex.

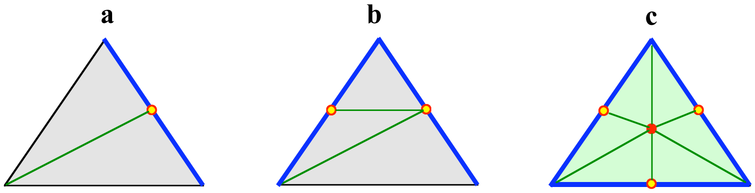

Figure 5: Subdivision of a chamfer triangle with one (a) two (b) or three (c) chamfer edges.

3.3 Snapping new vertices onto sharp features

Finally, we must find the proper position for each new vertex introduced in the middle of a chamfer edge or a corner triangle. We use an extrapolation of the smooth surfaces that are adjacent to these elements, as shown in Figure 6 and Figure 7 and explained below.

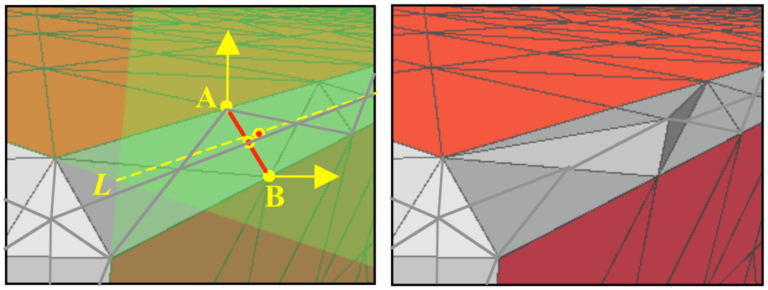

Figure 6: Re-location of a new vertex splitting a chamfer edge.

To find the position of a new vertex V inserted in a chamfer edge E, we consider the two original vertices, A and B, of E. We compute the weighted sum N of the normals to all of the red triangles incident upon A, normalize it, and define a plane P that is orthogonal to N and passes through A. As weights, we use the angle between the two edges of the incident triangle that meet at A [Mokhtarian et al. 1998]. Similarly, we compute the weighted sum M of the normals to all of the red triangles incident upon B, normalize it, and define a plane Q that is orthogonal to M and passes through B. Finally, we move V to the closest point on the line L of intersection between planes P and Q.

More specifically, the final position of V is $(A+B)/2+(h/k)H$, where $H=AB□(M□N)=(AB\bullet N)M+(BA\bullet M)N$, where $h=AB\bullet N$, and where $k=2(M\bullet N)(AB\bullet N)–2(AB\bullet M)$. The process is shown in Figure 6.

To find the position of a new corner vertex, W, inserted in a corner triangle with vertices A, B, and C, we proceed as follows. We first compute the weighted sum, N, of the normals to all of the red triangles incident upon A, normalize it, and define a plane P that is orthogonal to N and passes through A. Similarly, we define the plane Q through B with normal M and the plane R through C with normal S. Then, we move W to the intersection of planes P, Q, and R, which is the solution of the system of three linear equalities: $W\bullet N=A\bullet N$, $W\bullet M=B\bullet M$, $W\bullet S=C\bullet S$ (see Figure 7).

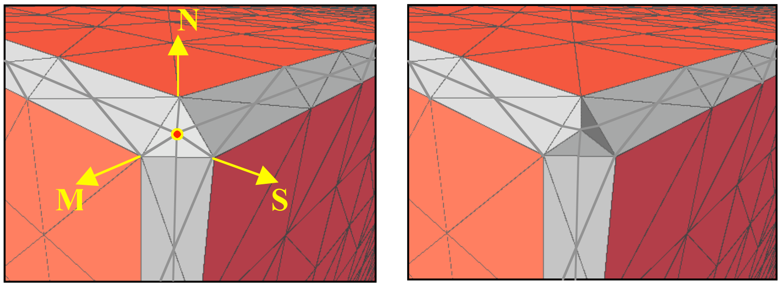

Figure 7: Re-location of a new vertex splitting a corner triangle.

For simplicity, we have omitted in the previous two paragraphs the discussion of **degenerate cases**. Such cases include situations where the pairs of planes are parallel or when the triplets of planes do not intersect at a single point, because their normals are coplanar. Moreover, since the algorithm is tailored for nearly uniform triangulations, we have chosen to avoid the creation of edges which are longer than the longest edge of the input mesh (see Figure 8). Thus, if the extrapolated position would require the creation of such a long edge, or if the position itself is not defined because of a linear dependency between the planes, we simply leave the newly inserted vertex in the middle of the chamfer edge or of the corner triangle.

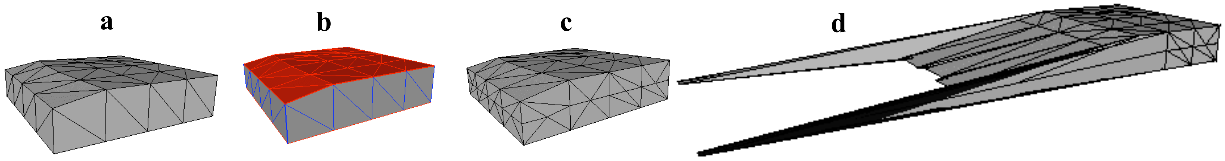

Figure 8: An input model (a) having some chamfer edges (b) were subdivided without moving the new vertices (c). The model (d) was obtained by skipping the edge-length check.

In some cases, a portion of a triangle strip that forms a chamfer is bordered by a concave edge on one side and by a convex edge on the other. We detect these situations easily by analyzing the configuration of the triangles incident on the end-points of the chamfer edge or triangle. We treat these cases as the ones discussed above, and simply do not move the newly inserted vertices.

Another degeneracy to be discussed includes all the configurations in which the original model has more than three sharp edges meeting at a corner. In these cases, a corresponding re-sampled model has a strip of chamfer edges for each original sharp edge, and these four strips meet at a region made of two or more corner triangles (see Figure 9). The new points that split these adjacent corners (and the chamfer edges in between) are moved to the same position, resulting in the creation of degenerate (zero-area) triangles. Therefore, when the sharpening is complete, it may be necessary to eliminate some degenerate (zero-area) triangles [Botsch and Kobbelt 2001]. We have tuned our implementation by considering as *degenerate* a triangle having at least one angle smaller than 1 degree; in this case we simply collapse the edge which is opposite to such an acute corner and update the connectivity graph consistently.

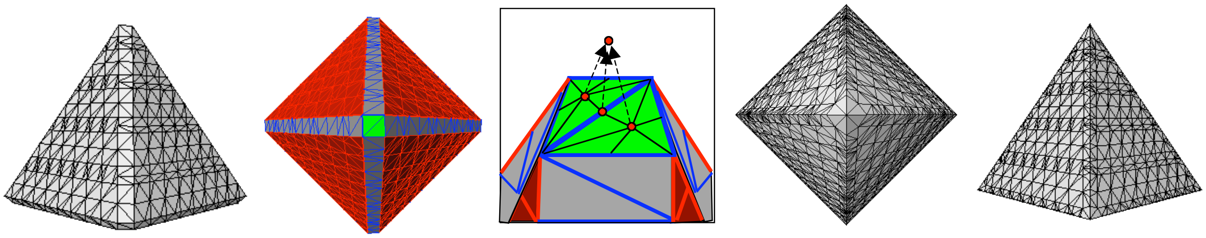

Figure 9: A pyramid was remeshed using the marching intersections [Rocchini et al. 2001]. Edge-Sharpener detected two adjacent corner triangles, has subdivided them, and has eliminated the resulting degenerate triangles.

The sharp edges and the sharp corners created by EdgeSharpener are marked, as such, for further processing, as discussed in the next section.

4. BENDING

The bending phase described here is optional. It is particularly beneficial when restoring **curved** models from a triangulation generated by a feature insensitive sampling.

The error between a curved smooth surface and a triangle mesh approximating it can often be reduced by inserting additional samples as vertices in the mesh. Standard subdivision approaches would round off the sharp features. Thus, we want to tag the sharp edges and use a Bender modified subdivision scheme that smoothens the surface and preserves the sharpness of sharp edges while bending chains of them into smooth curves.

Below, we first describe how to tag all of the sharp edges during the execution of EdgeSharpener and then provide the details of the Bender algorithm.

4.1 Tagging the sharp edges

In [Lee et al. 1998] and [Hoppe et al. 1994] a crude threshold on the dihedral angle was used to tag the edges of the mesh for further processing. We could use this method as a post-processing after running EdgeSharpener. Unfortunately, when the sampling is curvature-insensitive, such an approach could mistakenly tag, as sharp, some of the edges lying on smooth surfaces with high curvature.

To reduce the frequency of these false positives, we use a modified version of EdgeSharpener, in which, in addition to the sharp edges created by subdividing chamfer triangles, we also tag as sharp the non-smooth edges that bound two smooth faces. These edges may be easily identified using the coloring scheme introduced earlier. They may be precisely defined as the non-smooth edges that have both of their incident triangles painted red. To identify them during EdgeSharpener, we perform the first three EdgeSharpener filters as explained in the initial version above. After Filter 3 we insert a new filter, say Filter 3bis, which finds the non-brown edges with two adjacent red triangles and tags their ending vertices. Filter 3bis also tags all of the vertices having a non-manifold red neighborhood. Then, we execute the remaining filters and, at the end, we tag all of the vertices inserted by EdgeSharpener to subdivide the chamfer triangles. Finally, we tag as sharp all of the edges which link two tagged vertices. This process is shown in Figure 10.

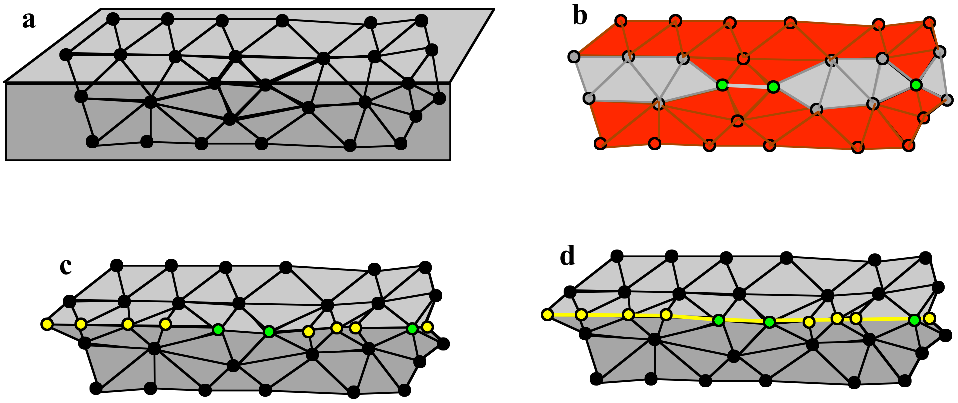

Figure 10: An original surface with a sharp edge is approximated by the triangulation of a feature-insensitive sampling [a]. Filter 3bis detected a non-brown edge having two incident red triangles and tagged its vertices. Filter 3bis also tagged a vertex having a non manifold red neighborhood (tagged vertices are shown in green in [b]). After the subdivision of the chamfer triangles, all of the newly inserted vertices have been tagged (yellow vertices in [c]). Finally, all of the edges joining two tagged vertices have been tagged as sharp (yellow edges in [d]). Note that the vertices of the tagged edges may have been tagged either in [b] or in [c].

4.2 The Bender algorithm

The **Bender** algorithm, introduced here, assumes that all of the sharp edges have been identified and tagged. Note that it can be executed on tagged meshes that are not necessarily produced by EdgeSharpener.

We wish to smoothen the triangle mesh to bring it closer to the original curved surface. Because we assume that the samples are on the original surface, we use an interpolatory subdivision scheme. We use a modification of the Butterfly subdivision [Dyn et al. 1990], which splits each triangle into four by inserting a new vertex in the middle of each edge, as shown in Figure 11.

Each newly inserted vertex p is then moved to a position that is a linear combination of the edge's end-points and six neighboring vertices. The configuration of the neighbors and their weights, which define the so-called *stencil* of the subdivision rule, are reported in Figure 13-R1, where the vertex p is marked by a black dot.

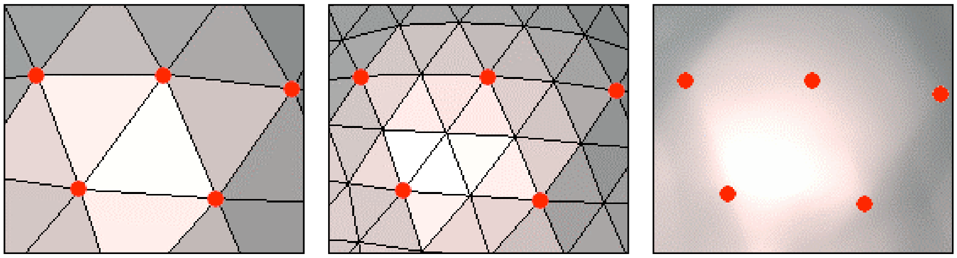

Figure 11: Butterfly subdivision: an example showing how an initial triangle mesh (on the left) is refined by a subdivision step (middle). On the right, the limit surface is shown. The red dots indicate the vertices of the initial triangle mesh, whose coordinates are not modified by the subdivision process.

When repeated, the Butterfly subdivision converges to a smooth surface everywhere, except near *extraordinary* vertices, which do not have six incident triangles. Furthermore, the Butterfly scheme is not defined for border edges. Both problems have been addressed by Zorin et al. [1996], who proposes to adapt the weights of the linear combination to take into account the valence, k, of each vertex (i.e. the number of incident edges) and the fact that some of the neighboring edges and vertices may be on the border of the surface. Zorin distinguishes several cases, shown in Figure 13-(R1-R5 and E1-E2). For each case, the position of p is defined by a particular stencil [Zorin et al. 1996; Zorin and Schröder 2000]. The values of the weights, corrected according to [Zorin 2003], are shown in the Figure.

Zorin's improved subdivision guarantees that the limit surface is smooth everywhere, including near the extraordinary vertices. Furthermore, his scheme can handle manifold triangle meshes with boundary. In this case, each boundary edge is subdivided using the one-dimensional four point stencil introduced in [Dyn et al. 1987] and depicted in Figure 13-R2, which ignores the valence of the vertices and only makes use of the neighboring vertices on the boundary. However, it does not make provision for sharp edges, which we want to bend into smooth curves while preserving their sharpness.

Our Bender algorithm is a modification of Zorin's scheme which, when iterated, is able to smoothen the mesh everywhere while preserving the sharpness of the tagged edges. Our modifications, described in the following paragraphs, take place when subdividing an edge having one or both end-points on a sharp edge. For all the other edges we use the Zorin's subdivision rules described in [Zorin et al. 1996; Zorin and Schröder 2000] (with the corrections provided in [Zorin 2003]) and reported in Figure 13-E2.

4.2.1 Modified rules for sharp edges

In the remainder of the description we make use of the notion of *manifold sharp vertex*, which we define to be a vertex with exactly two incident sharp edges. If an edge, E = (V,W), is tagged as sharp we may have three configurations:

- If both V and W are manifold sharp vertices, then we subdivide the edge using the one-dimensional four-point scheme described in [Dyn et al. 1987] and depicted in Figure 13-R2, where the sharp edges are shown in red.
- If both V and W are non-manifold, we leave the new point in the middle of the sharp edge E.
- If V is non-manifold and W is manifold, then let F be the sharp edge incident at W and different from E. In this case we *reflect* F on the other side of E, that is, we consider the plane P containing both E and F and, on P, we compute the mirror F' of F with respect to the axis of E. Then we consider F' as being the only other sharp edge incident at V and apply the one-dimensional four-point scheme (see Figure 13-R6).

4.2.2 Modified rules for edges having a vertex on a tagged edge

When subdividing an edge with one of its end-points, say V_0, on a sharp edge, we perform a topological cut along sharp edges. Specifically, if V_0 has *n* >1 incident sharp edges, we create *n-1* copies of V_0, say V_1, V_2, .., V_n. Then, we duplicate all of the sharp edges meeting at V_0 and, for each copy, we substitute V_0 with one of the V_i. Finally, we update the connectivity graph so that one of the two triangles adjacent to a sharp edge becomes adjacent to its copy. This process only involves topological operations and the coordinates of each V_i are exactly the same as those of V_0. This process is shown in Figure 12.

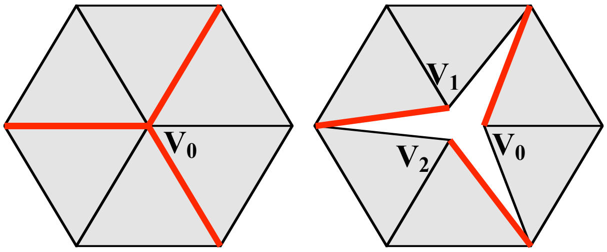

Figure 12: Example of topological cut along sharp edges (red on the left). In the image, V_0, V_1 and V_2 have been displaced to show the topological hole.

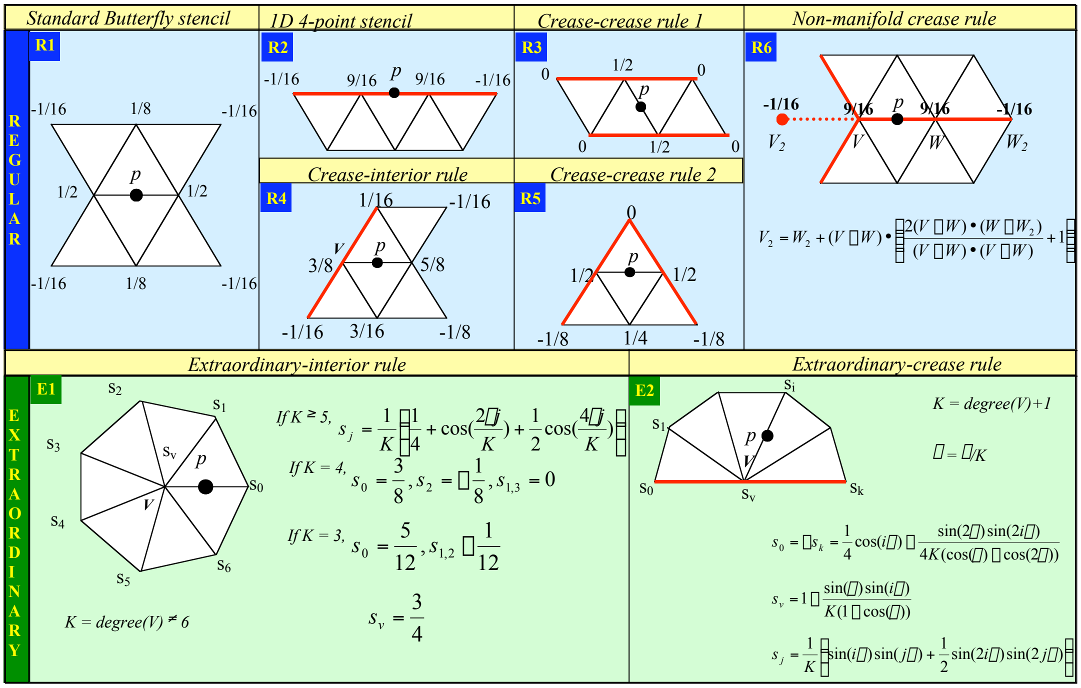
Standard Butterfly stencil
R1
-1/16
1/8
-1/16
1/2
p
1/2
-1/16
1/8
-1/16
1D 4-point stencil
R2
-1/16
9/16
p
9/16
-1/16
Crease-crease rule 1
R3
0
1/2
0
p
0
1/2
0
Non-manifold crease rule
R6
-1/16
9/16
p
9/16
-1/16
V_2
V
W
W_2
$V_2 = W_2 + (V - W) \bullet \left[\frac{2(V - W) \bullet (W - W_2)}{(V - W) \bullet (V - W)} + 1 \right]$
Crease-interior rule
R4
1/16
-1/16
V
p
3/8
5/8
-1/16
3/16
-1/8
Crease-crease rule 2
R5
0
p
1/2
1/2
-1/8
1/4
-1/8
REGULAR
Extraordinary-interior rule
E1
s_2
s_1
s_3
s_v
p
s_0
s_4
V
s_5
s_6
$K = degree(V) \neq 6$
If K ≥ 5, $s_j = \frac{1}{K}\left(\frac{1}{4} + \cos(\frac{2\pi j}{K}) + \frac{1}{2}\cos(\frac{4\pi j}{K})\right)$
If K = 4, $s_0 = \frac{3}{8}, s_2 = -\frac{1}{8}, s_{1,3} = 0$
If K = 3, $s_0 = \frac{5}{12}, s_{1,2} = -\frac{1}{12}$
$s_v = \frac{3}{4}$
Extraordinary-crease rule
E2
s_i
s_1
p
V
s_0
s_v
s_k
$K = degree(V)+1$
$\theta = \pi/K$
$s_0 = -s_k = \frac{1}{4}\cos(i\theta) - \frac{\sin(2\theta)\sin(2i\theta)}{4K(\cos(\theta) - \cos(2\theta))}$
$s_v = 1 - \frac{\sin(\theta)\sin(i\theta)}{K(1 - \cos(\theta))}$
$s_j = \frac{1}{K}\left(\sin(i\theta)\sin(j\theta) + \frac{1}{2}\sin(2i\theta)\sin(2j\theta)\right)$
EXTRAORDINARY

Figure 13: Stencils used by Bender. ‘p’ is the point being computed as a linear combination of the depicted neighboring vertices. All the other possible neighbors are assumed to have zero coefficient. Sharp edges are shown in red.

In the particular case where $n = 1$ (that is, V_0 is the dead-end of a chain of sharp edges) we do not duplicate any vertex or edge, and simply consider V_0 as if it were not on a sharp edge. In all the other cases, after the cut, the vertex V_0 becomes a boundary vertex. According to [Zorin and Schröder 2000] and Figure 13 we apply the suitable boundary rule and close the mesh back to its original configuration.

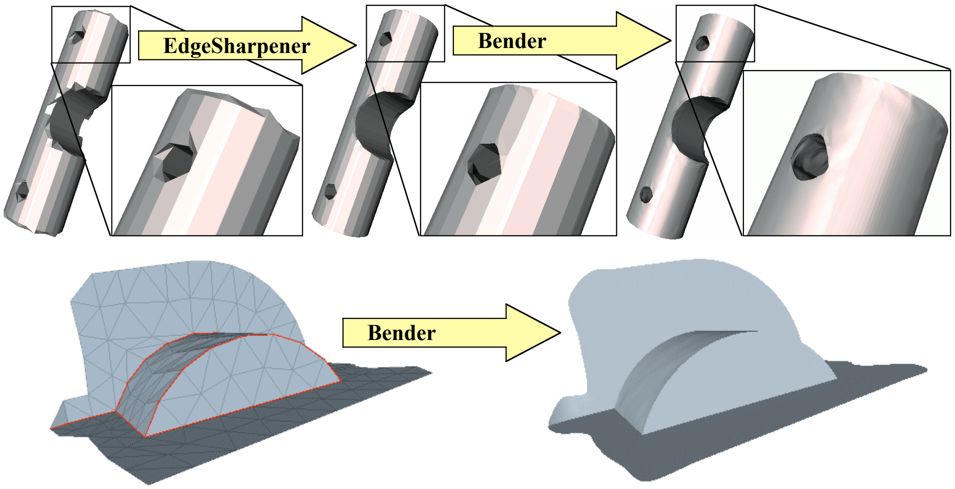

Figure 14: Top row: An example showing the improvement of a coarsely remeshed model after its processing through Sharpen&Bend. Bottom row: another example showing the behavior of Bender alone on a mesh with both boundary and sharp edges (shown in red on the leftmost model).

Finally, if a non-sharp edge has both its end points on some tagged edge, we sequentially perform the cut for both the vertices, apply the proper boundary rule, and close the mesh back.

Note that if Bender is to be used more than once (as in the examples shown in Figure 14), it is necessary to propagate the marking throughout the subdivision. Thus, when splitting a sharp edge, we tag as sharp the two new edges connecting the old end-points of the edge with the new vertex. Also, note that Bender can handle manifold triangle meshes with both sharp edges and boundary, as shown in Figure 14.

5. RESULTS AND DISCUSSION

We have tested EdgeSharpener extensively in conjunction with the SwingWrapper compression algorithm [Attene et al. 2003b]. In order to reduce the number of bits to encode the vertex locations, SwingWrapper performs a remeshing of an original dense triangle mesh, constraining the position of each vertex to lie on a circle defined by two previously created neighboring vertices.

Specifically, SwingWrapper grows the new mesh by attaching one new triangle at a time following an EdgeBreaker like traversal order [Rossignac 1999; Rossignac 2001]. When the new triangle has a new tip vertex, the location of this tip is computed as the intersection of a circle orthogonal to the gate with the original surface, forcing the two new edges to have a prescribed length L. This scheme allows one to encode the location of the tip vertices using a few bits that quantize the dihedral angle at the gate. The sequence of quantized angles is further compressed using an arithmetic coder. The SwingWrapper compression is *lossy*, since an error is introduced by the remeshing. Most of the discrepancy between the original and the re-sampled models is concentrated near the sharp edges and corners.

We have tested EdgeSharpener on models produced by the Piecewise Regular Meshes (PRMs) compression approach [Szimczak et al. 2002], which performs a different remeshing. Based on their orientation, it splits the triangles into 6 sets. The set of triangles whose normal is closest to the positive x-direction is sampled using a regular grid in the y-z plane. The other five sets are sampled similarly using the appropriate grids. The results are connected into a valid triangle mesh. Although in the PRM approach each vertex is actually represented by its 3 coordinates, the parallelogram prediction [Touma and Gotsman 1998] of two coordinates is perfect for all the interior vertices of a set. The zero corrections to perfect predictions can be encoded compactly.

The connectivity of the meshes produced by SwingWrapper and by PRM is encoded using modified versions of the EdgeBreaker compression scheme [Rossignac 1999].

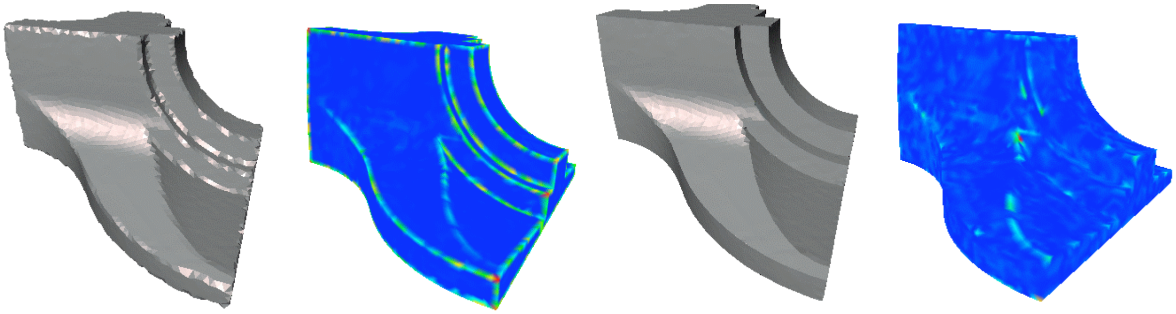

Figure 15: The maximum error in the fandisk encoded with SwingWrapper is 0.89% of the bounding-box diagonal. After the filtering through EdgeSharpener such error is 0.37%.

In both cases, we have observed that EdgeSharpener significantly reduces the error between the original shape and the one recovered after decompression. An example of this improvement is shown in

Figure 15, where the "fandisk" model was compressed using SwingWrapper. When no sharpening is applied, the maximum distance between the decoded mesh and the original model is 0.89% of the bounding box diagonal. It decreases down to 0.37% after the application of our new EdgeSharpener filter. The colored models have been produced by the Metro tool [Cignoni et al. 1998] used to measure the distortion. Metro uses a color spectrum to show the distribution of the error. Note that the spectrum is normalized to fit the whole range of errors, so that the blue color corresponds to the minimum error while the red indicates the maximum. Thus, the light color which appears after the sharpening in some regions must not be interpreted as an increase of the error, because it comes from a renormalization of the color spectrum in a more narrow range.

For all the models with sharp features that we have remeshed, we have observed a significant decrease of the L^2 distortion for all the bit-rates due to the **EdgeSharpener**. A further improvement of the rate-distortion curves can be achieved by **Bender** if the original model has curved smooth areas (such as, for example, the fandisk model depicted in Figure 15), and was coarsely remeshed.

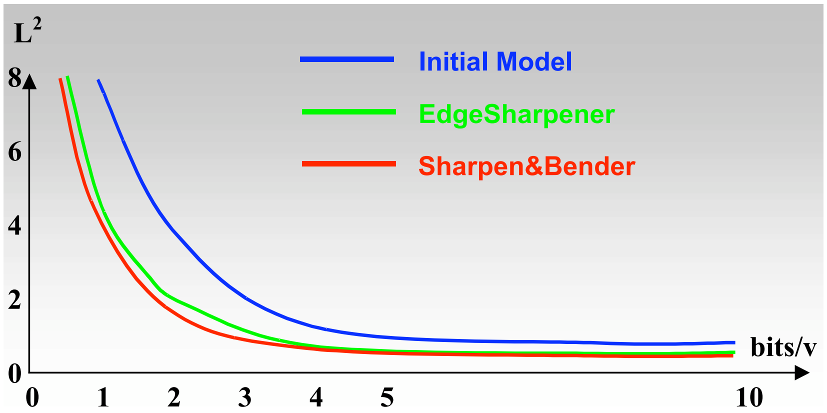

Figure 16: Reduction of the L^2 error for various sampling densities. Bit-per-vertex rates are relative to the # of vertices (6475) of the original fandisk model. Errors are expressed in units of 10^{-4} of the bounding-box diagonal.

We have also tested EdgeSharpener with and without Bender on a number of triangle meshes generated through the Marching-Cubes algorithm, through the SwingWrapper and the PRM remeshers and through the surface reconstruction method described in [Attene and Spagnuolo 2000]. We have found that in all the cases, when the original model was sampled with a sufficiently high density, most of the sharp features can be completely recovered, while the parts of the mesh that correspond to regions of the original model without sharp features are not modified by Edge-Sharpener.

In Figure 20 some results of the combined Sharpen&Bend algorithm are shown. The model of the face was laser digitized and reconstructed from the point cloud; the bone was created by running the marching-intersections [Rocchini et al. 2001] algorithm on a denser version of the same model; all the other models have been produced by the SwingWrapper remesher. In all of the examples (except for the face, for which we did not have an original surface to compare with), we have observed a significant reduction of both the maximum and the mean square distortions.

In order to test the robustness of the proposed approach in presence of noisy data, we have perturbed some models with various amounts of noise and we have observed that the sharpening does not produce unwanted side-effects. Clearly, if the amount of noise becomes comparable with the inter-sample spacing, its influence on the dihedral angles prevents the algorithm to identify some chamfer elements, but the results are still very good (Figure 17).

Moreover, we have concluded that the effectiveness of the proposed method is not restricted to uniformly sampled meshes. For example, Edge-Sharpener correctly restores the sharp features of typical meshes generated through interpolation of laser-captured point sets or through iso-surface polygonization procedures which exhibit a fair amount of variation in edge-length.

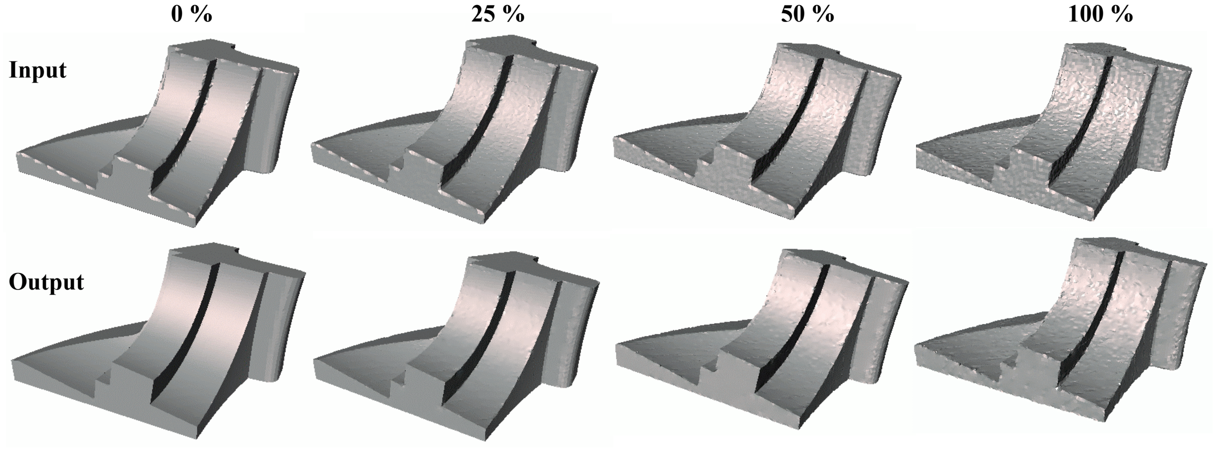

Figure 17: Processing a model with various amounts of noise. The input was sharpened and de-noised through constrained Laplacian smoothing (tagged vertices were not moved). The amplitude of the noise in the normal direction ranges from 0% (left) to 100% (right) of the maximum length of an edge in the input mesh.

5.1 Limitations

Clearly, Edge-Sharpener can miss sharp features that are smaller than the inter-sample spacing and may produce sharp edges where the original model has a feature that has been smoothed with a small-radius blend (Figure 18). There is simply not enough information in the sampling to recover such small features or blends.

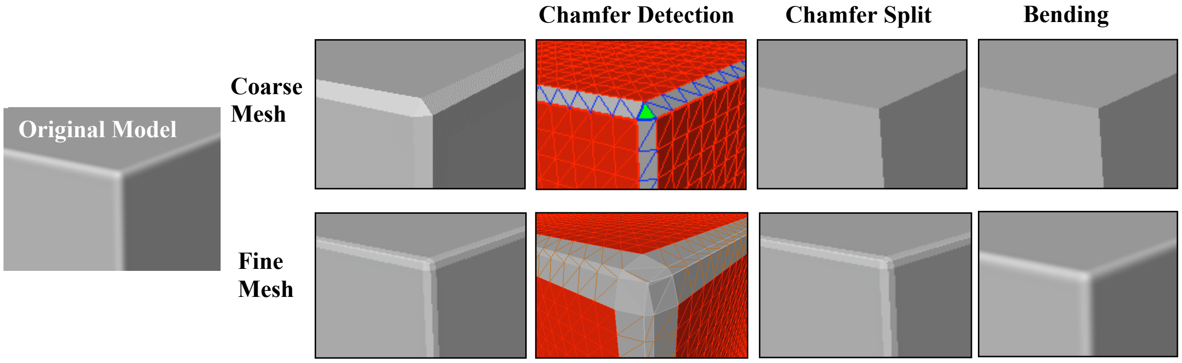

Figure 18: Unwanted creases may be produced if an original surface has blends whose radius is smaller than the inter-sample spacing (top row). If the sampling step is small compared to the blend radius, the blends are not modified by Edge-Sharpener (bottom row), while they are faired as expected by Bender.

Also, in extremely rare cases, the alias corresponding to a feature that blends smoothly into a flat area may be painted red, preventing the detection of some “desired” chamfer triangles. This situation, however, may happen only if the strip of such triangles is not aliased, which is very improbable in practical cases. Figure 19, for example, shows the correct reconstruction of such a blended feature from a retiled model having the typically “rippled” strips of chamfer triangles. On the other hand, if the same model was sampled through a grid exactly aligned with its sharp edges, the blended feature would not have been recovered because its corresponding strip of chamfer triangles would have been smoothly blended onto the flat face, and the red region would have expanded along the strip through brown edges.

If an original model has a smooth face that is thinner than 3 times the inter-sample spacing, Edge-Sharpener may not be able to identify a sufficient number of smooth vertices for it and hence may not be able to recover the sharp features which bound that face. As for the unwanted sharpening of small radius blends, this problem is a consequence of an insufficient sampling density, and may be easily solved by using a denser sampling.

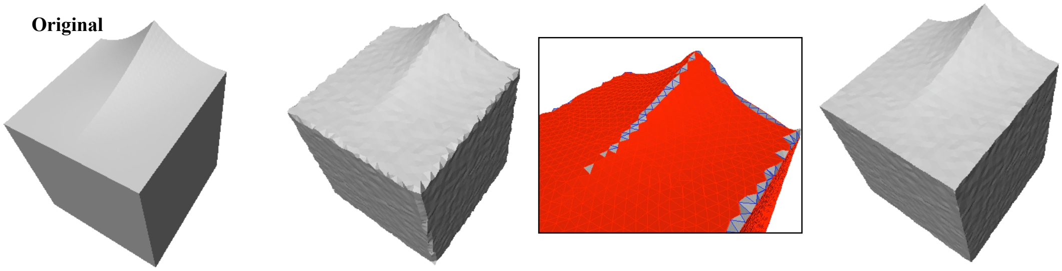

Figure 19: Reconstruction of a sharp feature which blends smoothly onto a flat surface. The "ripple" of the strip of chamfer triangles prevented the red region to expand on the chamfer.

5.2 Performance

Our experiments on a variety of meshes indicate that **Edge-Sharpener** is extremely fast and robust. For example, the sharpening of the models presented in this paper took less than 0.4 seconds each on a standard PC equipped with a 1.7Ghz CPU. The performance of **Bender** is comparable with the one of a typical subdivision scheme. Our implementation, which is not particularly optimized, subdivides an average of about 22000 triangles per second. Precise timings for the combined Sharpen&Bend algorithm are shown in Figure 20, where each model was subdivided once, except for the hand which was subdivided twice.

6. CONCLUSIONS

We have introduced a simple, automatic, and efficient edge-sharpening procedure designed to recover the sharp features that are lost by reverse engineering or by remeshing processes that use a non-adaptive sampling of the original surface. The procedure starts by identifying smooth edges. Then, it performs six trivial filters that identify chamfer edges, which in turn define chamfer and corner triangles. The chamfer edges and triangles are subdivided by inserting new vertices and moving them to strategic locations where the sharp feature is estimated through extrapolation of abutting smooth portions of the surface. Also, we have introduced (1) an automatic tagging approach which marks the sharp edges and (2) a Bender modified subdivision scheme that smoothens the surface and preserves the sharpness of tagged edges while bending chains of them into smooth curves.

We have run numerous tests on models coming from uniform remeshing, marching-cubes iso-surface generation, and surface reconstruction from nearly uniform clouds of points. In all of the cases, in addition to the correct reconstruction of sharp features, we have observed that the distortion between the mesh and the original model was significantly reduced by our sharpening process, while the parts of the mesh not corresponding to sharp features in the original model were not modified. Moreover, when the original model has curved areas, the application of Bender further decreases the distortion.

There are several directions for future research, the most interesting including:

- The recovering of conical tips which, though being *sharp*, are not at the intersection of any sharp edge;
- A proper handling of sharp edges which blend smoothly onto flat areas;
- The determination of a set of sampling conditions which guarantee that EdgeSharpener will be able to recover all of the sharp edges and corners.

ACKNOWLEDGEMENTS

The authors thank all the members of the Shape Modeling Group of the IMATI-GE/CNR and the reviewers for their helpful advice. Thanks are due to Denis Zorin for providing the correct coefficients of the extraordinary-crease rule.

REFERENCES

ADAMY, U., GIESEN, J. AND JOHN, M. 2000. New techniques for topologically correct surface reconstruction. In *Proceedings of IEEE Visualization '00*, 373–380.

AMENTA, N., CHOI, S. AND KOLLURI, R. 2001. The power crust. In *Proceedings of the 6th ACM Symposium on Solid Modeling and Applications*, 249-260.

ATTENE, M. AND SPAGNUOLO, M. 2000. Automatic surface reconstruction from point sets in space. *Computer Graphics Forum 19*, 3 (*Proceedings of Eurographics '00*), 457-465.

ATTENE, M., FALCIDIENO, B., ROSSIGNAC, J. AND SPAGNUOLO, M. 2003. Edge-Sharpener: Recovering sharp features in triangulations of non-adaptively re-meshed surfaces. In *Proceedings of the 1st Eurographics Symposium on Geometry Processing*, 63-72.

ATTENE, M., FALCIDIENO, B., SPAGNUOLO, M. AND ROSSIGNAC, J. 2003. SwingWrapper: Retiling triangle meshes for better EdgeBreaker compression. *ACM Transactions on Graphics 22*, 4, 982-996.

BIASOTTI, S., MORTARA, M. AND SPAGNUOLO, M. 2000. Surface Compression and Reconstruction Using Reeb Graphs and Shape Analysis. In *Proceedings of the Spring Conference on Computer Graphics (SCCG '00)*, 174-185.

BIERMANN, H., MARTIN, I.M., ZORIN, D. AND BERNARDINI, F. 2001. Sharp Features on Multiresolution Subdivision Surfaces. In *Proceedings of Pacific Graphics '01*, 140-149.

BLOOMENTHAL, J. 1988. Polygonization of implicit surfaces. *Computer Aided Geometric Design 5*, 341-355.

BOTSCH, M. AND KOBBELT, L. P. 2001. A Robust Procedure to Eliminate Degenerate Faces from Triangle Meshes. In *Proceedings of Vision, Modeling and Visualization (VMV '01)*.

CHENG, S. W. AND DEY, T. K. 1999. Improved construction of Delaunay based contour surfaces. In *Proceedings of the ACM Symposium on Solid Modeling and Applications*, 322-323.

CIGNONI, P., ROCCHINI, C. AND SCOPIGNO, R. 1998. Metro: measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (*Proceedings of Eurographics '98*), 167-174.

CONG, G. AND PARVING, B. 2001. Robust and Efficient Surface Reconstruction from Contours. *The Visual Computer 17*, 199-208.

DEY, T. K. AND WENGER, R. 2001. Reconstructing curves with sharp corners. *Computational Geometry Theory Applications 19*, 89-99.

DYN, N., GREGORY, J. AND LEVIN, D. 1990. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics 9*, 2, 160-169.

DYN, N., GREGORY, J. AND LEVIN, D. 1987. A four-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design 4*, 257-268.

FLEISHMAN, S., DRORI, I. AND COHEN-OR, D. 2003. Bilateral Mesh Denoising. In *Proceedings of ACM SIGGRAPH '03*, 950-953.

GARLAND, M. AND HECKBERT, P.S. 1997. Surface Simplification using Quadric Error Metrics. In *Proceedings of ACM SIGGRAPH '97*, 209-216.

GARLAND, M., WILLMOTT, A. AND HECKBERT, P.S. 2001. Hierarchical face clustering on polygonal surfaces. In *Proceedings of the ACM Symposium on Interactive 3D graphics*, 49-58.

GIESEN, J. AND JOHN, M. 2002. Surface reconstruction based on a dynamical system. *Computer Graphics Forum 21*, 3 (*Proceedings of Eurographics '02*), 363-371.

GUMHOLD, S., WANG, X. AND MACLEOD, R. 2001. Feature Extraction from Point Clouds. In *Proceedings of the 10th International Meshing Roundtable*, 293-305.

GUSKOV, I., VIDIMCE, K., SWELDENS, W. AND SCHRÖDER, P. 2000. Normal Meshes. In *Proceedings of ACM SIGGRAPH '00*, 95-102.

GUY, G. AND MEDIONI, G. 1997. Inference of Surfaces, 3D Curves and Junctions from sparse, noisy, 3D data. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 11, 1265-1277.

HOPPE, H. 1996. Progressive Meshes. In *Proceedings of ACM SIGGRAPH '96*, 99-108.

HOPPE, H., DEROSE, T., DUCHAMP, T., HALSTEAD, M., JIN, H., MCDONALD, J., SCHWEITZER, J. AND STUETZLE, W. 1994. Piecewise smooth surface reconstruction. In *Proceedings of ACM SIGGRAPH '94*, 295-302.

HUBELI, A. AND GROSS, M. H. 2001. Multiresolution feature extraction from unstructured meshes. In *Proceedings of IEEE Visualization '01*, 16–25.

HUBELI, A., MEYER, K. AND GROSS, M. H. 2000. Mesh Edge Detection. In *Proceedings of the Workshop Lake Tahoe* (Lake Tahoe City, California, USA).

JONES, T., DURAND, F. AND DESBRUN, M. 2003. Non-Iterarive, Feature-Preserving Mesh Smoothing. In *Proceedings of ACM SIGGRAPH '03*, 943-949.

JU, T., LOSASSO, F., SCHAEFER, S. AND WARREN, J. 2002. Dual Contouring of Hermite Data. In *Proceedings of ACM SIGGRAPH '02*, 339-346.

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U. AND SEIDEL, H-P. 2001. Feature Sensitive Surface Extraction from Volume Data. In *Proceedings of ACM SIGGRAPH '01*, 57-66.

LEE, A., SWELDENS, W., SCHRÖDER, P., COWSAR, L. AND DOBKIN, D. 1998. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *Proceedings of ACM SIGGRAPH '98*, 95-104.

LOOP, A. 1987. Smooth Subdivision Surfaces based on Triangles. *Master's thesis*, University of Utah, Department of Mathematics.

LORENSEN, W. AND CLINE, H. 1987. Marching Cubes: a high resolution 3D surface construction algorithm. In *Proceedings of ACM SIGGRAPH '87*, 163-169.

MOKHTARIAN, F., KHALILI, N. AND YUEN, P. 1998. Multi-Scale 3-D Free-Form Surface Smoothing. In *Proceedings of the British Machine Vision Conference*, 730-739.

OHTAKE, Y. AND BELYAEV, A.G. 2002. Dual/Primal Mesh Optimization for Polygonized Implicit Surfaces. In *Proceedings of the ACM Symposium on Solid Modeling and Applications*, 171-178.

OHTAKE, Y., BELYAEV, A.G. AND PASKO, A. 2001. Dynamic Meshes for Accurate Polygonization of Implicit Surfaces with Sharp Features. In *Proceedings of Shape Modeling International (SMI '01)*, 74-81.

PATANÉ, G. AND SPAGNUOLO, M. 2002. Multi-resolution and Slice-oriented Feature Extraction and Segmentation of Digitized Data. In *Proceedings of the ACM Symposium on Solid Modeling and Applications*, 305-312.

ROCCHINI, C., CIGNONI, P., GANOVELLI, F., MONTANI, C., PINGI, P. AND SCOPIGNO, R. 2001. Marching Intersections: an efficient resampling algorithm for surface management. In *Proceedings of Shape Modeling International (SMI '01)*, 296-305.

ROSSIGNAC, J. 1999. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1, 47-61.

ROSSIGNAC, J. 2001. 3D Compression made simple: EdgeBreaker with Wrap&Zip on a Corner-Table. In *Proceedings of Shape Modeling International (SMI '01)*, 278-283.

SZYMCZAK, A., KING, D. AND ROSSIGNAC, J. 2002. Piecewise Regular Meshes. *Graphical Models 64*, 3-4, 183-198.

THOMPSON, W.B., OWEN, J.C., DE ST. GERMAIN, H.J., STARK, S.R. AND HENDERSON, T.C. 1999. Feature-Based Reverse Engineering of Mechanical Parts. *IEEE Transactions on Robotics and Automation 12*, 1, 57-66.

TOUMA, C. AND GOTSMAN, C. 1998. Triangle Mesh Compression. In *Proceedings of Graphics Interface '98*, 26-34.

VORSATZ, J., RÖSSL, C., KOBBELT, L.P. AND SEIDEL, H.-P. 2001. Feature Sensitive Remeshing. *Computer Graphics Forum 20*, 3 (*Proceedings of Eurographics '01*), 393-401.

WATANABE, K. AND BELYAEV, A.G. 2001. Detection of Salient Curvature Features on Polygonal Surfaces. *Computer Graphics Forum 20*, 3 (*Proceedings of Eurographics '01*), 385-392.

ZORIN, D. AND SCHRÖDER, P. 2000. Subdivision for Modeling and Animation. *SIGGRAPH '00 Course Notes*, Course #23, 23-28 July, New Orleans, Louisiana, USA.

ZORIN, D., SCHRÖDER, P. AND SWELDENS, W. 1996. Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of ACM SIGGRAPH '96*, 189-192.

ZORIN, D. 2003. Private Communication.

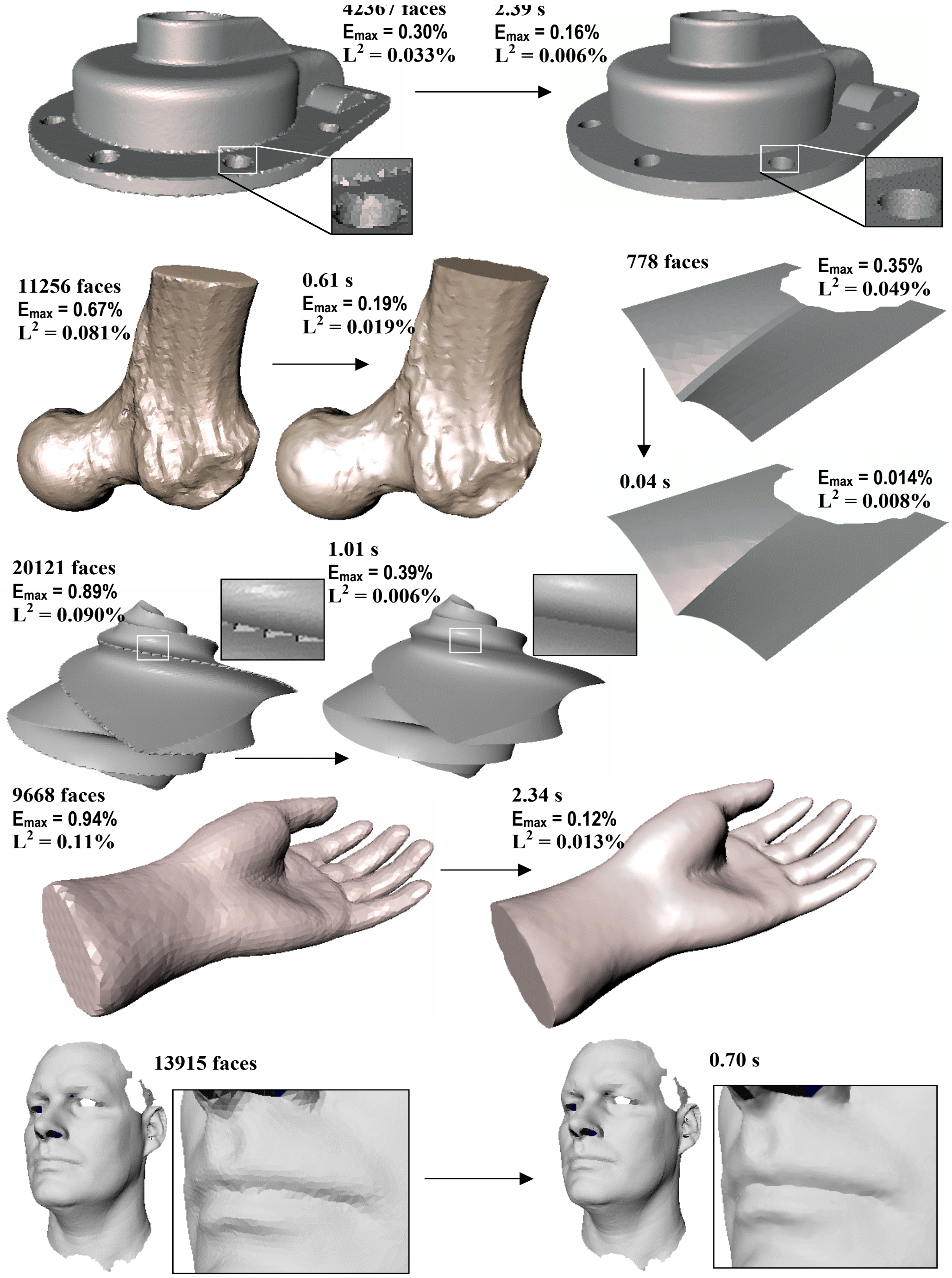

Figure 20: Some examples of models improved through Sharpen&Bend. For each input model the number of faces is indicated, while near each output model the processing time is reported. Furthermore, for all the input models produced by remeshing (all of the examples but the face) we report both the maximum and the mean-squared distortions before and after running Sharpen&Bend.