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

M. Attene, B. Falcidieno, J. Rossignac and M. Spagnuolo

Abstract - Various acquisition, analysis, visualization and compression approaches sample surfaces of 3D shapes in a uniform fashion, without any attempt to align the samples with sharp edges or to adapt the sampling density to the surface curvature. Consequently, triangle meshes that interpolate these samples usually chamfer sharp features and exhibit a relatively large error in their vicinity. We present two new filters that improve the quality of these re-sampled models. **EdgeSharpener** restores the sharp edges by splitting the chamfer edges and forcing the new vertices to lie on intersections of planes extending the smooth surfaces incident upon these chamfers. **Bender** refines the resulting triangle mesh using an interpolating subdivision scheme that preserves the sharpness of the recovered sharp edges while bending their polyline approximations into smooth curves. A combined **Sharpen&Bend** post-processing significantly reduces the error produced by feature-insensitive sampling processes. For example, we have observed that the mean-squared distortion introduced by the SwingWrapper remeshing-based compressor can often be reduced by 80% executing EdgeSharpener alone after decompression. For models with curved regions, this error may be further reduced by an additional 60% if we follow the EdgeSharpening phase by Bender.

Index Terms: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling – *Boundary representations, Geometric algorithms, languages and systems*

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 [2][3][18], computed from a 3D scalar field resolved on a regular grid [8][32], or identified on slices of medical data [10][12]. Most acquisition techniques restrict each sample to lie on a specific line or 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. [29], 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. 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 [5][20][1]. 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. To reduce this error, one may choose to use a feature-sensitive remeshing process [30], 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 compactness of a feature-

- *M. Attene, B. Falcidieno and M. Spagnuolo - Istituto di Matematica Applicata e Tecnologie Informatiche, sez. di Genova, Consiglio Nazionale delle Ricerche. E-mail: {attene, falcidieno, spagnuolo}@ge.imati.cnr.it.*
- *J. Rossignac – College of Computing, Georgia Institute of Technology. E-mail: jarek@cc.gatech.edu*

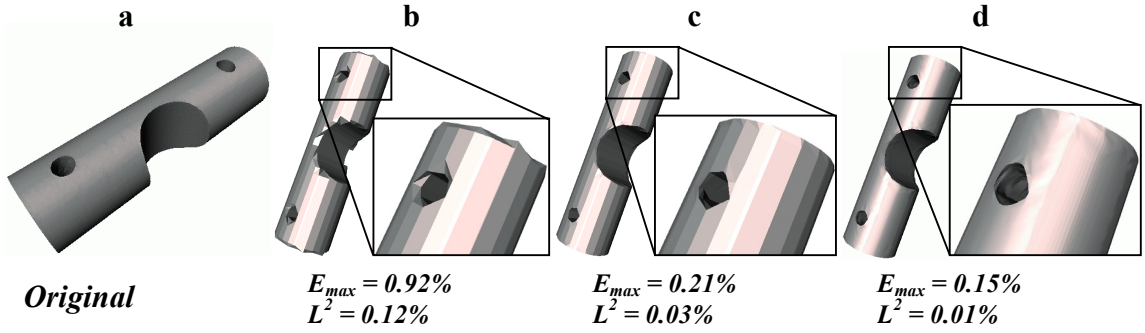


Fig. 1: An original model (a) was re-meshed through a feature-insensitive algorithm (b). The sharp edges and corners were restored by EdgeSharpener (c). Then, Bender faired the smooth regions without rounding off the sharp features reconstructed by EdgeSharpener (d). For each model, the maximum distance 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.

insensitive retiling while reducing the approximation error, we have developed the **EdgeSharpener** approach. 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. 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 [14][44] may be used to refine the triangle mesh globally, bending the triangles to smooth 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. Considering sharp edges as if they were boundaries, as suggested in [44], is not sufficient for retaining corners and sharp edges with dead-ends. Thus, we introduce here a new approach, called **Bender**, which preserves the sharpness of the features restored by EdgeSharpener, while bending them so that they form smooth curves between sharp corners. For edges that are not sharp and not adjacent to a sharp edge, Bender performs a modified Butterfly subdivision [43][44]. For other edges, we introduce new subdivision rules, which make it possible to properly refine sharp corners, sharp edges, and also smooth edges that connect to sharp features. The benefits of combining Edge-Sharpener and Bender (into a filter that we call Sharpen&Bend) are illustrated in Fig. 1.

A significant number of publications have been focused on identifying sharp features in a 3D model [24][25], even in the presence of noise. More recently, solutions were proposed for maintaining sharp features during remeshing [29][41]. In both cases, the features to be extracted or preserved are present in the model. In contrast to this body of previous work, our solution 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.

The above considerations reveal the importance of Sharpen&Bend for post-processing 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 [2], a guaranteed-quality reconstruction of surfaces with sharp features has remained a challenge, even in the 2D case [13].

1.1 Original contributions

A preliminary description of EdgeSharpener was first introduced in a conference publication [4]. In this article, however, a more thorough analysis of the state of the art is presented, along with a deeper investigation of the results and limitations of the method. Moreover, extensions to the original algorithm are introduced in order to tag sharp edges while reconstructing them. For non-adaptively sampled surfaces, this approach is more accurate than typical methods based on a threshold of the dihedral angle.

Here we also introduce **Bender**, which uses novel rules to maintain the sharpness of tagged edges while subdividing the mesh. We show that to properly handle sharp edges which are not closed 1-manifold curves it is not sufficient to treat them as if they were boundary curves, and hence we introduce novel special rules.

Most important, this paper proposes a new all-in-one *black box* to improve non-adaptively (re)sampled meshes. We show that, for a variety of popular re-meshed models, Sharpen&Bend significantly decreases their distortion with respect to the corresponding original shapes.

2 Related Work

Here, we successively discuss approaches to identify features in *unstructured data* (scattered points), *partially structured data* (contours or profiles), and *structured data* (polygonal meshes). Then, we discuss feature-sensitive polygonization, re-tiling, smoothing, and subdivision approaches that preserve sharp features.

2.1 Identifying sharp features in unstructured point clouds

When a scattered point sampling of a surface is sufficiently dense, sharp features may be inferred by analyzing the

neighborhood of each point. This analysis may be performed after a triangle mesh has been reconstructed [1] or directly from a point cloud [19], by first organizing it through a neighbor graph, then evaluating the flatness of the neighbors of each point, and finally extracting an optimized sub-graph spanning non-flat vertices. Even after pruning, the edges of that subgraph often form zig-zag patterns, because they connect input samples on opposite sides of sharp features.

Gumhold et al. propose to smooth the zig-zags by fitting low degree splines [19]. The resulting curves are likely to lie on *chamfers*, rather than on the intersections of extrapolated surfaces. For instance, if the original solid had a convex sharp edge, the chamfer produced by a feature-insensitive sampling would cut through the solid. The spline approximation would lie on the chamfer and hence inside the solid, rather than close to the original sharp edge.

Perceptual grouping rules, based on surface normals, have been used to infer smooth surface patches [21]. Sharp features are recovered as intersections of adjacent patches. When surface normals are not provided, the method estimates them at each sample from the locations of neighboring samples. Hence, normals at samples near sharp features are polluted by neighbors on other patches. Polluted normals lead to errors in the estimation of sharp features.

The two approaches described above work well for dense point clouds. The Edge-Sharpening approach proposed in this paper extends sharp feature recovery to cases where the vertices of the input triangle mesh are sparse. Moreover, the sharp edges recovered by our method lie precisely on the intersections of the estimated incident smooth patches.

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 [6]. 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 [36].

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 of the mesh being swept 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 [24], 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 [22], 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 [22], 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. The use of curvature extrema [42] suffers from the same limitation.

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 [34][35], 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 [34], 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 [35] 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 [41].

During the triangulation of an iso-surface, an *extended* marching cubes (EMC) algorithm [29] 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 [27], 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 [29], 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 [23], where the authors use a modification of the Loop's subdivision scheme [31] 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 [7]. 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 subdivides triangle meshes and is inspired by the approximant scheme developed in [23]. When the model to be improved does not contain noise, as in the case of most results of remeshing processes, an approximant scheme such as Hoppe's [23] would introduce an unnecessary error on vertices. Furthermore, such a scheme may require a considerable number of iterations to reach an acceptable fit to the data. In contrast, since our **feature preserving subdivision** scheme is **interpolatory**, we preserve the input data while bending the surface patches interpolating sample points.

When the input triangle mesh interpolates noisy samples, subdivision may have no benefit. Instead, a smoothing process may be needed after the Edge-Sharpening. Recent feature-preserving techniques for mesh smoothing [16][26] 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 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

Our approach identifies what we call **chamfer edges**, which are shown in blue in Fig. 2. It is based on the identification of **smooth edges**, which 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 of the threshold angle 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 π. 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 π or, equivalently, the average angle, ε, between the normals of two adjacent triangles remains close to 0. The influence of non-smooth edges on ε is small but not null, thus the actual angle for smooth edges is slightly smaller than ε. In practice we do not have infinite samplings, so taking ε as threshold makes the algorithm too sensitive to small amounts of noise. We have experienced that doubling ε is a good compromise between theoretical correctness in the ideal case and robustness in all of the practical cases encountered.

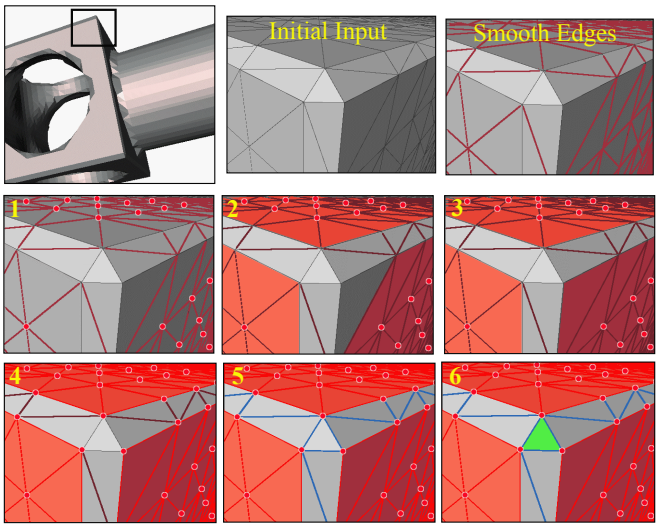


Fig. 2: An original model (top-left) was re-meshed through a feature insensitive algorithm (top-right). The smooth edges in the aliased input model were detected and the six filters (1-6) have selected the chamfer edges and the corner triangles to be subdivided.

Our approach to identify chamfer triangles 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 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.* It is surrounded by a smooth portion of the surface.
- *Paint **red** each **triangle** that has at least one red vertex.* They form the cores of a smooth regions.
- Extend the cores by recursively painting **red** the triangles adjacent to a red triangle through a brown edge.
- Paint **red** the edges and vertices of red triangles.
- *Paint **blue** each non-red edge joining two red vertices.* These are the chamfer edges.
- *Paint **green** each triangle bounded by three blue edges.* These are the corner triangles where chamfers meet.

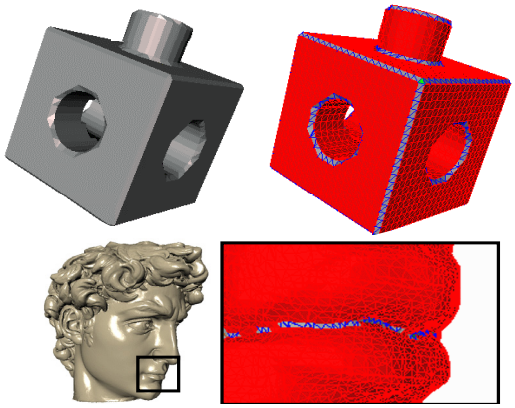


Fig. 3: Chamfers identified by Edge-Sharpener on a Marching-Cubes generated model (top row) and on the simplified version of a laser scanned model of Michelangelo's David (bottom row, model courtesy of the Digital Michelangelo Project , Stanford University). Some edges are still gray or brown.

The six steps are illustrated in Fig. 2. **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 need to 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 (Fig. 3). 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 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 Fig. 4):

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

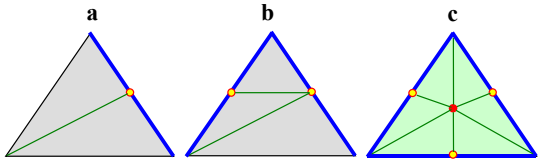


Fig. 4: Subdivision of a chamfer triangle with one (a) two (b) or three (c) chamfer edges.

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 of a corner triangle. We use an extrapolation of the smooth surfaces that are adjacent to these elements, as shown in Fig. 5 and Fig. 6 and explained below.

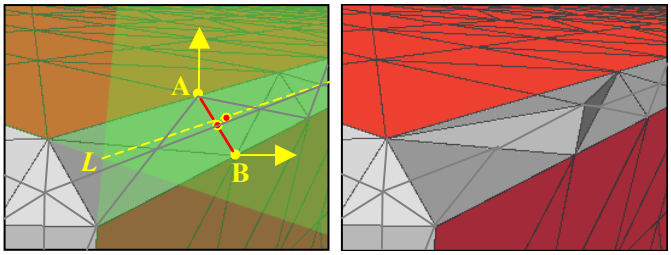


Fig. 5: Re-location of a new vertex splitting a chamfer edge.

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 (see Fig. 5). 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 [33]. Similarly, we compute the weighted sum M of the normals to 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. Specifically:

$$V = (A+B)/2+(h/k)H,$$

where $h=AB \bullet N$, $k=2(M \bullet N)(AB \bullet N)-2(AB \bullet M)$,

and $H=AB \times (M \times N)=(AB \bullet N)M+(BA \bullet M)N$

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 Fig. 6).

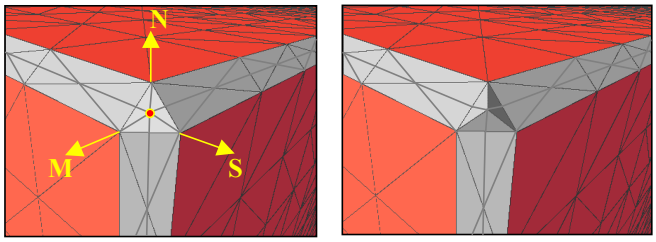


Fig. 6: Re-location of a new vertex splitting a corner triangle.

3.4 Dealing with degenerate situations

For simplicity, we have omitted in the previous sections the discussion of degenerate cases. We identify them here and explain how they are handled.

Nearly parallel planes: 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 Fig. 7). 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.

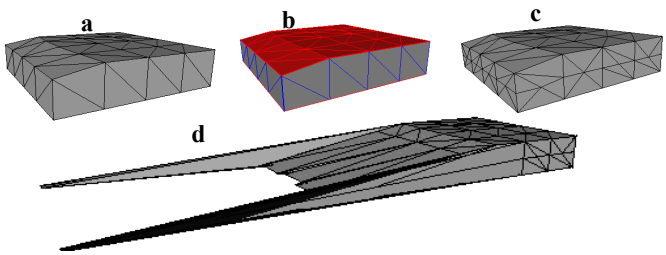


Fig. 7: In an input model (a) the chamfer edges joining nearly parallel surfaces (b) were subdivided without moving the new vertices (c). A wrong model (d) would be obtained without our edge-length check.

Step features: 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.

Multi-corners: The original model may have 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 strips meet at a region made of two or more corner triangles. 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 faces [9]. 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.

4 BENDING

The bending phase described in this section is particularly beneficial when restoring **curved** models from a triangulation generated by a feature insensitive sampling.

The error between a curved smooth surface and its triangle mesh approximation can often be reduced through subdivision. Because standard subdivision approaches would round off the sharp features, we have developed a new subdivision scheme that preserves the sharpness of sharp edges, while bending their piecewise linear approximations into smooth curves.

4.1 Tagging the sharp edges

As suggested previously [23] [30], we could attempt to recover the sharp edges from the mesh produced with Edge Sharpener, using a crude threshold on the dihedral angle. 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.

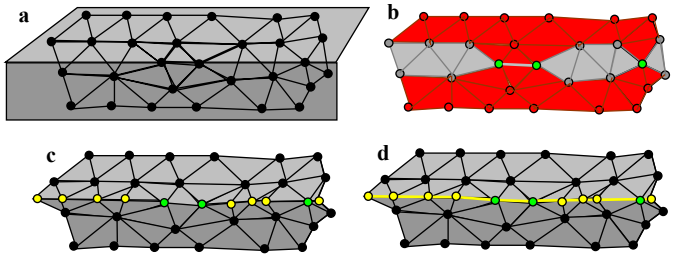


Fig. 8: An original surface with a sharp edge is approximated by the triangulation of a feature-insensitive sampling [a]. Filter 3bis has detected a non-brown edge having two incident red triangles and has tagged its vertices. Filter 3bis has also tagged a vertex having a non manifold red neighborhood (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].

To reduce the frequency of these false positives, we want to use the results of EdgeSharpener. However, the original version of EdgeSharpener only tags as sharp the new edges created by subdividing chamfer triangles. It does not explicitly tag as sharp the original edges where smooth patches meet. Hence, we have extended EdgeSharpener to also tag as sharp the non-smooth edges that bound two smooth faces. These pre-existing sharp edges are the non-smooth edges that bound two red triangles. To identify them during EdgeSharpener, we perform the first three EdgeSharpener filters as explained in the initial version above. After Filter 3, we execute 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 Fig. 8.

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 also can be executed on tagged meshes that are not necessarily produced by EdgeSharpener.

We wish to smooth the triangle mesh to bring it closer to the original curved surface. Because we assume that the samples (i.e vertices) lie on the original surface, we use an interpolatory subdivision scheme. We have selected to use a modification of the Butterfly subdivision [14], which splits each triangle into four by inserting a new vertex in the middle of each edge, as shown in Fig. 9.

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 stencil of the subdivision rule, are reported in Fig. 10-R1, where the vertex p is marked by a black dot.

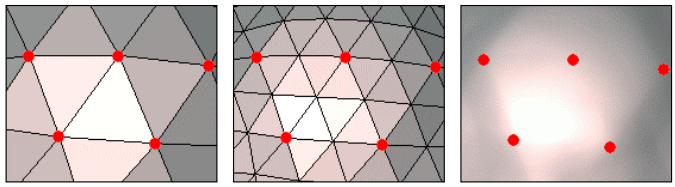


Fig. 9: Butterfly subdivision: an example showing how an initial triangle mesh (on the left) is refined by a subdivision step (middle). On the right, the limit surface is shown. The red dots indicate the vertices of the initial triangle mesh, whose coordinates are not modified by the subdivision process.

When repeated, the Butterfly subdivision converges to a smooth surface everywhere, except near *extraordinary* vertices, which do not have six incident triangles. The Butterfly scheme is not defined for border edges. Both problems have been addressed by Zorin et al. [44], who propose 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. They distinguish several cases, shown in Fig. 10-(R1-R5 and E1-E2). For each case, the position of p is defined by a particular stencil [43]. The values of the weights have been incorrectly reported in [43]. We have corrected them according to [45]. The improved subdivision

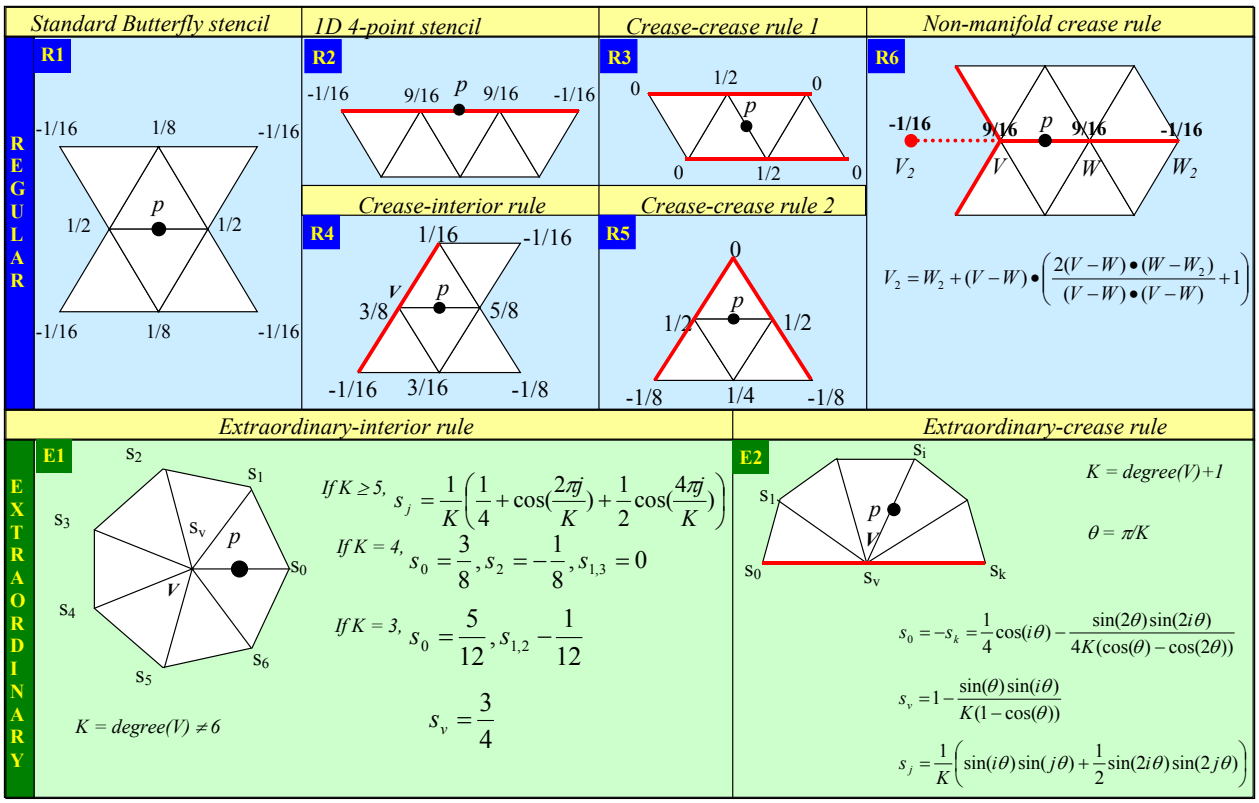


Fig. 10: Stencils used by Bender. 'p' is the point being computed as a linear combination of the depicted neighboring vertices. All the other possible neighbors are assumed to have zero coefficient. Sharp edges are shown in red.

guarantees that the limit surface is smooth everywhere, including near the extraordinary vertices. Furthermore, this 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 [15] and depicted in Fig. 10-R2, which ignores the valence of the vertices and only makes use of the neighboring vertices on the boundary. However, the approach of [43] 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 this scheme. It is able to smooth the mesh everywhere, while preserving the sharpness of the tagged edges. Our modification is limited to edges with one or both end-points bounding a sharp edge.

4.2.1 Modified rules for sharp edges

We say that a vertex with exactly two incident sharp edges is a *manifold sharp vertex*. If an edge, E, joining vertices V and W, is tagged as sharp, we may have three configurations:

- If both V and W are manifold sharp vertices, then we subdivide E using the one-dimensional four-point scheme described in [14] and depicted in Fig. 10-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 E.
- Now consider the case where only V is non-manifold. Let F be the sharp edge incident at W and different from E. We *reflect* F on the other side of E. To do so, we consider the plane P containing both E and F. On P, we compute the mirror F' of F with respect to the bisector axis of E. Then we consider F' as being the only other sharp edge incident at V and apply the four-point scheme (see Fig. 11-R6).

4.2.2 Other modified subdivision rules

When subdividing an edge with only one end-point, say V_0, on a sharp edge, we perform a topological cut along all the sharp edges incident upon V_0. Specifically, if V_0 has $n>1$ incident sharp edges, we create $n-1$ copies of V_0, say $V_1 \ldots V_n$, and duplicate all of the sharp edges meeting at V_0. This process only involves topological operations and is illustrated in Fig. 11.

When V_0 is the dead-end of a chain of sharp edges ($n=1$), 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 [43] and Fig. 10, we apply the suitable boundary rule and close the mesh back to its original configuration.

Finally, if a non-sharp edge has both end points on a

tagged edge, we perform the cut for both vertices, apply the proper boundary rule, and close the mesh back.

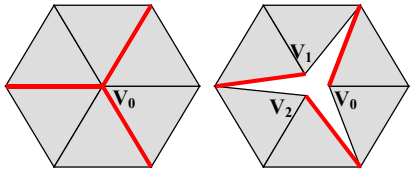


Fig. 11: Example of topological cut along sharp edges (red on the left). In the image, V_0, V_1 and V_2 have been displaced to show the topological hole.

Note that if Bender is to be used more than once (as in the examples shown in Fig. 12), it is necessary to propagate the sharp-edge markings 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 correctly handle manifold triangle meshes with both sharp edges and boundary, as shown in Fig. 12.

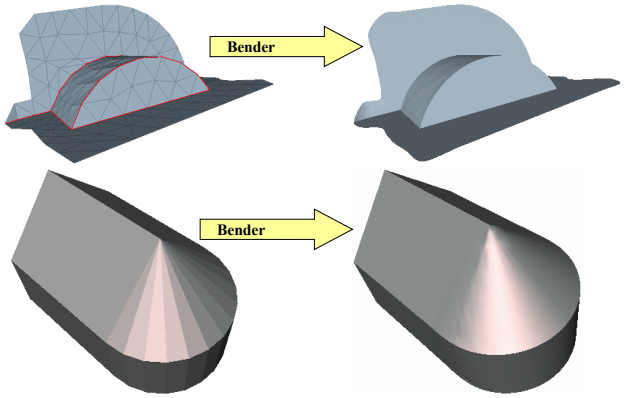


Fig. 12: Top row: an example showing the behavior of Bender alone on a mesh with both boundary and sharp edges (shown in red on the leftmost model). Bottom row: another example showing the bending around the dead-end of a sharp edge.

5 Results and Discussion

We have tested Sharpen&Bender extensively in conjunction with the SwingWrapper compression algorithm [5]. 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 [38][39]. When the new triangle has a new tip vertex, the location of this tip is computed as the intersection with the original surface of a circle orthogonal to the gate (edge where the new triangle is attached). Therefore, the two new edges have a prescribed length L. This scheme allows one to encode the location of the tip vertex 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*, and most of the discrepancy between the original and the re-sampled models is concentrated near the sharp edges and corners, and near regions of high curvature. The connectivity of the meshes produced by SwingWrapper is encoded using modified versions of the EdgeBreaker compression scheme. In [5], we report very aggressive compression ratios and show that the error is concentrated around sharp features.

We have also tested EdgeSharpener with and without Bender on a number of models generated through the Marching-Intersections algorithm [37], which performs a Marching-Cubes-like [32] re-tiling of an input mesh, and through surface reconstruction [3] applied on data which simulates the typical patterns used in laser sampling.

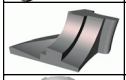
Original Shape	Remeshed through	L^∞ and L^2 Distortions	After EdgeSharpen	After Sharp.&Bend	Proc. Time
	SwingWrapper 31554 faces	0.41% 0.054%	0.18% 0.031%	0.16% 0.021%	1.94
	Reconstruction 11256 faces	0.67% 0.073%	0.26% 0.053%	0.19% 0.019%	0.61
	March. Inters. 778 faces	0.35% 0.049%	0.19% 0.017%	0.14% 0.008%	0.04
	March. Inters. 20121 faces	0.89% 0.090%	0.37% 0.014%	0.39% 0.006%	1.01
	SwingWrapper 9668 faces	0.94% 0.11%	0.23% 0.08%	0.12% 0.013%	2.34
	Reconstruction 7548 faces	0.81% 0.10%	0.35% 0.041%	0.31% 0.023%	0.49
	Original mesh 13915 faces	- -	- -	- -	0.70

Fig. 13: Experimental results showing the reduction of the L^∞ and L^2 distortions due to both EdgeSharpener and Bender. The number in the second column counts the faces of the remeshed models. All the errors are expressed as a percentage of the model's bounding box diagonal; errors are missing for the 'face' model because we did not have an original surface to compare with. In the last column the total running time is reported (in seconds). In all of the tests Bender was run once, except for the 'hand' for which two iterations were performed.

We have found that in all the cases tested, when the original shape 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, and correctly smoothed by Bender (see Fig. 17).

We have observed that Sharpen&Bend significantly reduces the error between the original shape and the remeshed one (Fig. 13). Consider two extreme cases: 1) when the original shape has no sharp edges EdgeSharpener has no benefit and 2) when all the faces of the original shape are flat, Bender has no benefit. Between these extreme cases, the error reduction varies depending on the shape and accounts to both EdgeSharpener and Bender. If the input model interpolates a dense enough sampling, EdgeSharpener is expected to have a strong impact on the

L^∞ distortion, while Bender will mainly reduce the L^2 error. EdgeSharpener, however, will also have an impact on the L^2 distortion, whose reduction may become significant if the original shape has only flat faces separated by sharp edges. Furthermore, if the input model was obtained from too coarse a sampling, EdgeSharpener may miss some sharp edges. In such a case, Bender will round these edges and may introduce a slight increase in the L^∞ distortion. Fig. 13 shows the results of our experiments which clearly agree with the above considerations. Errors have been computed through the publicly available Metro tool [11], except for the model of the face for which we did not have an original surface to compare with. The remeshed versions of the test models, along with their Sharpen&Bend improved versions, are shown in Fig. 17 and Fig. 18.

To analyze how sampling density affects their benefits, we have applied Edge-Sharpener and Bender to models of increasing resolution produced by SwingWrapper. As indicated in Fig. 14, the relative L^2 error reduction remains considerable at all scales. The top-left part of the figure includes an additional curve relative to the PGC coder [28]. Note that PGC uses a feature-sensitive remeshing engine [30], and thus outperforms SwingWrapper on models with sharp features. The error reduction due to Sharpen&Bend, however, widely compensate for this advantage.

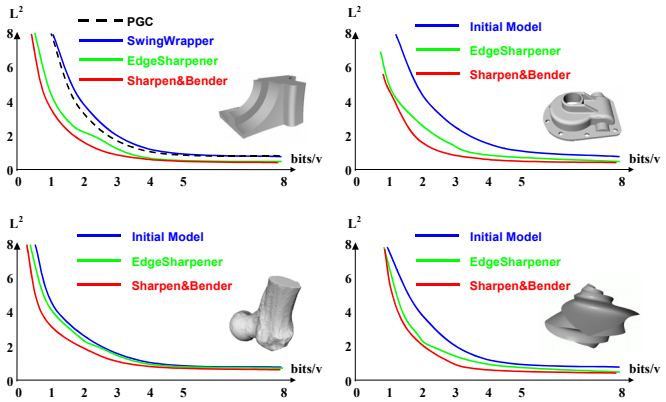


Fig. 14: Impact of EdgeSharpener and Bender on the rate-distortion curves of the SwingWrapper compressor. Bit-per-vertex rates are relative to the # of vertices of the original model. Errors are expressed in units of 10^{-4} of the bounding-box diagonal.

We conclude that the effectiveness of the proposed method is not restricted to sharply uniform meshes. For example, EdgeSharpener 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. Though they are not uniform, in fact, it is important to note that in these meshes the edge length is typically bounded (i.e. cell's diagonal in the Marching Cubes approach), therefore our detection of degenerate cases still offers good results.

Finally, we have found that small quantities of noise do not prevent EdgeSharpener to correctly restore sharp edges. Modern, well-calibrated laser scanners produce absolutely acceptable data. Clearly, if the amount of noise becomes comparable with the inter-sampling spacing, its influence on the dihedral angles prevents the algorithm to identify some chamfer elements, and some sharp edges may be missed. In [4] an example is depicted showing how the sharpening quality degrades as the noise increases. Furthermore, for such heavily perturbed data an interpolating subdivision is not appropriate, and it is preferable to follow the EdgeSharpening by a feature-preserving smoothing [16][26] or by an approximant subdivision of the tagged mesh [23].

5.1 Limitations

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 (Fig. 16). There is simply not enough information in the sampling to recover such small features or blends.

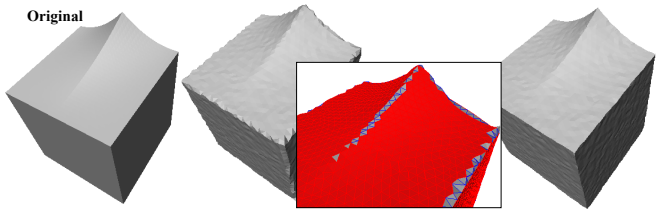


Fig. 15: Reconstruction of a sharp feature which blends smoothly onto a flat surface. The "ripple" of the strip of chamfer triangles prevented the red region to expand on the chamfer.

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. Fig. 15, 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.

Finally, 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 solved by using a denser sampling.

Unfortunately, we cannot provide measures of sampling density that guarantee that EdgeSharpener does not miss

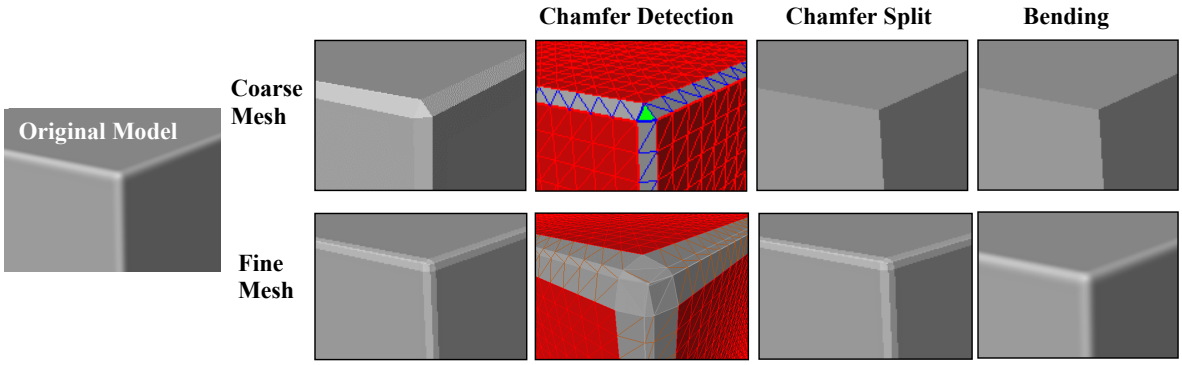


Fig. 16: Unwanted creases may be produced if an original surface has blends whose radius is smaller than the inter-sample spacing (top row). If the sampling step is small compared to the blend radius, the blends are not modified by Edge-Sharpener (bottom row), while they are smoothed as expected by Bender.

any feature. Sufficient sampling conditions for non-smooth geometry are hard to define [13] and, to the best of our knowledge, their formalization is still an open problem.

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 Fig. 13, where each model was subdivided once, except for the hand which was subdivided twice.

6 Conclusions

We have presented 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. Also, we have introduced (1) a new automatic tagging approach which marks the sharp edges and (2) a Bender modified subdivision scheme that smooths 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.

Acknowledgments

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 and by the EU Project AIM@SHAPE (Contract # 506766). Rossignac's work on this project was partly supported by a DARPA/NSF CARGO grant #0138420. 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 corrected coefficients of the extraordinary-crease rule.

References

[1] Adamy, U., Giesen, J. and John, M. 2000. New techniques for topologically correct surface reconstruction. In *Proceedings of IEEE Visualization '00*, 373–380.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] Bloomenthal, J. 1988. Polygonization of implicit surfaces.

Computer Aided Geometric Design 5, 341-355.

[9] 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)*.

[10] 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.

[11] 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.

[12] Cong, G. and Parving, B. 2001. Robust and Efficient Surface Reconstruction from Contours. *The Visual Computer 17*, 199-208.

[13] Dey, T. K. and Wenger, R. 2001. Reconstructing curves with sharp corners. *Computational Geometry Theory Applications 19*, 89-99.

[14] 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.

[15] Dyn, N., Gregory, J. and Levin, D. 1987. A four-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design 4*, 257-268.

[16] Fleishman, S., Drori, I. and Cohen-Or, D. 2003. Bilateral Mesh Denoising. In *Proceedings of ACM SIGGRAPH '03*, 950-953.

[17] Garland, M. and Heckbert, P.S. 1997. Surface Simplification using Quadric Error Metrics. In *Proceedings of ACM SIGGRAPH '97*, 209-216.

[18] Giesen, J. and John, M. 2002. Surface reconstruction based on a dynamical system. *Computer Graphics Forum 21*, 3 (*Proceedings of Eurographics '02*), 363-371.

[19] Gumhold, S., Wang, X. and Macleod, R. 2001. Feature Extraction from Point Clouds. In *Proceedings of the 10th International Meshing Roundtable*, 293-305.

[20] Guskov, I., Vidimce, K., Sweldens, W. and Schröder, P. 2000. Normal Meshes. In *Proceedings of ACM SIGGRAPH '00*, 95-102.

[21] 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.

[22] Hoppe, H. 1996. Progressive Meshes. In *Proceedings of ACM SIGGRAPH '96*, 99-108.

[23] 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.

[24] Hubeli, A. and Gross, M. H. 2001. Multiresolution feature extraction from unstructured meshes. In *Proceedings of IEEE Visualization '01*, 16–25.

[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).

[26] Jones, T., Durand, F. and Desbrun, M. 2003. Non-Iterarive, Feature-Preserving Mesh Smoothing. In *Proceedings of ACM SIGGRAPH '03*, 943-949.

[27] Ju, T., Losasso, F., Schaefer, S. and Warren, J. 2002. Dual Contouring of Hermite Data. In *Proceedings of ACM SIGGRAPH '02*, 339-346.

[28] Khodakovsky, A., Schröder, P. and Sweldens, W. 2000. Progressive Geometry Compression. In *Proceedings of ACM SIGGRAPH '00, 271-278.*

[29] 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.

[30] 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.

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

[32] Lorensen, W. and Cline, H. 1987. Marching Cubes: a high resolution 3D surface construction algorithm. In *Proceedings of ACM SIGGRAPH '87*, 163-169.

[33] Mokhatarian, 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

[40] Szymczak, A., King, D. and Rossignac, J. 2002. Piecewise Regular Meshes. *Graphical Models 64*, 3-4, 183-198.

[41] 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.

[42] 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.

[43] Zorin, D. and Schröder, P. 2000. Subdivision for Modeling and Animation. *SIGGRAPH '00 Course Notes*, Course #23, 23-28 July, New Orleans, Louisiana, USA.

[44] Zorin, D., Schröder, P. and Sweldens, W. 1996. Interpolating subdivision for meshes with arbitrary topology. In *Proceedings of ACM SIGGRAPH '96*, 189-192.

[45] Zorin, D. 2003. Private Communication.

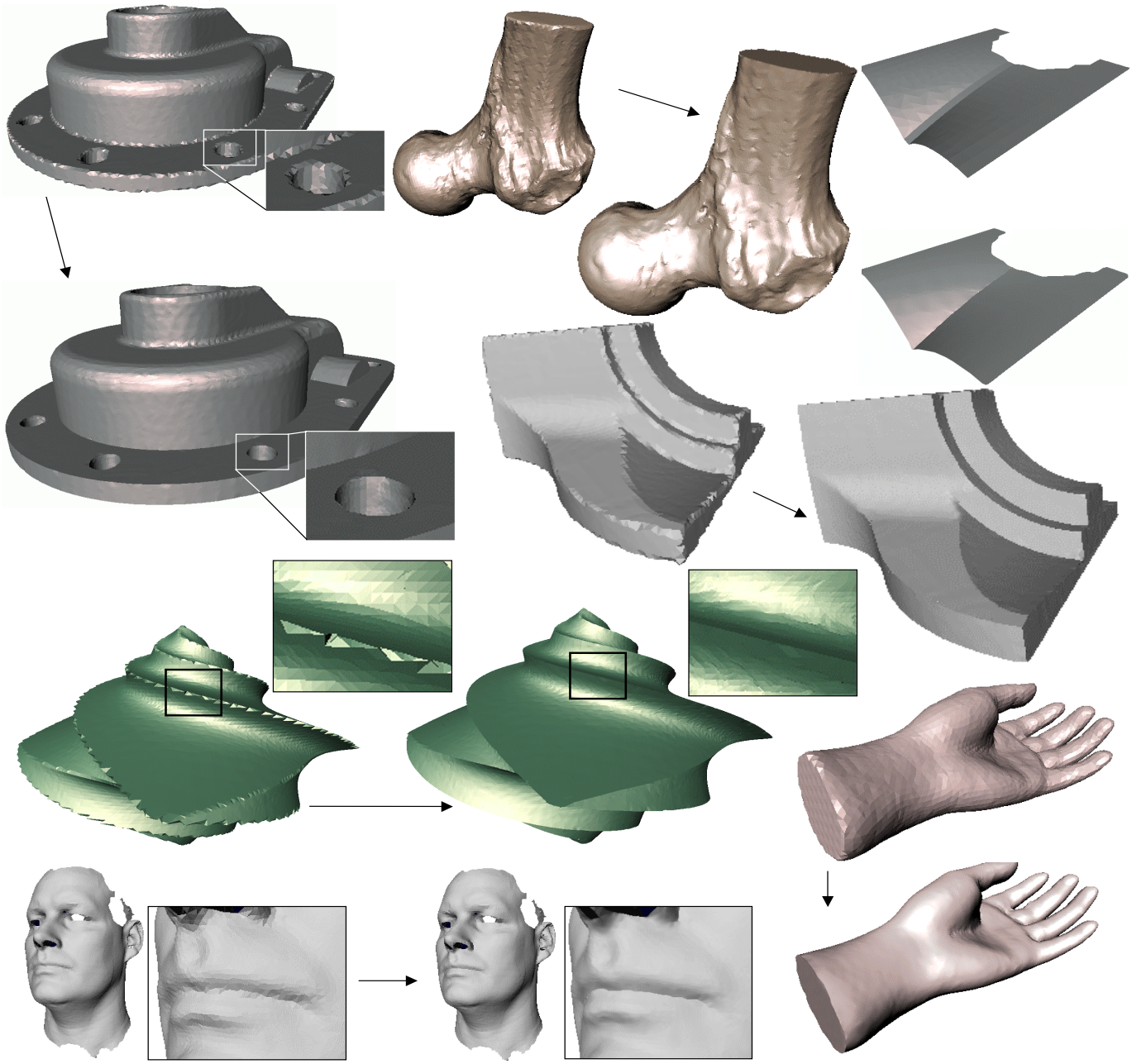


Fig. 17: Some examples of models improved through Sharpen&Bend. Corresponding numerical results are reported in Fig. 13. Original models courtesy of INPG (casting), Cyberware (ball_joint), H. Hoppe (fandisk), Y. Ohtake (octa_flower), Minolta (face) and Far Field Technology (hand).

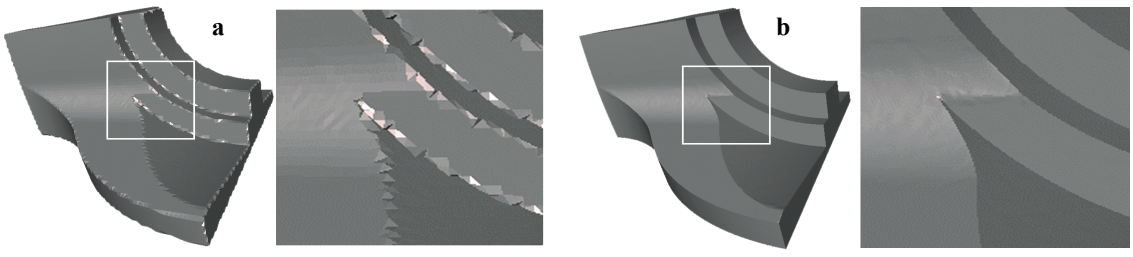


Fig. 18: The fandisk model was slightly tilted (30 degrees around the x-axis) before being re-sampled through a regular grid aligned with the coordinate frame (a). Although the sharp edges are not aligned with the sampling pattern, EdgeSharpener correctly reconstructed them. The final model improved by Sharpen&Bend is shown in (b).