EUROGRAPHICS 2001 / A. Chalmers and T.-M. Rhyne
(Guest Editors)

Volume 20 (*2001*), *Number 3*

Feature Sensitive Remeshing

J. Vorsatz, C. Rössl, L. P. Kobbelt[†] and H.-P. Seidel

Max-Planck Institut für Informatik
Saarbrücken, Germany

Abstract

Remeshing artifacts are a fundamental problem when converting a given geometry into a triangle mesh. We propose a new remeshing technique that is sensitive to features. First, the resolution of the mesh is iteratively adapted by a global restructuring process which additionally optimizes the connectivity. Then a particle system approach evenly distributes the vertices across the original geometry. To exactly find the features we extend this relaxation procedure by an effective mechanism to attract the vertices to feature edges. The attracting force is imposed by means of a hierarchical curvature field and does not require any thresholding parameters to classify the features.

1. Introduction

Surface representations based on polygonal meshes have become a standard in many geometric modeling applications. The reasons for the success of these mesh representations are the flexibility and generality with respect to shape and topology as well as the availability of efficient algorithms processing on meshes.

For a given geometric shape, there is obviously a multitude of possible mesh representations even if we prescribe a certain approximation tolerance. Depending on the application for which the mesh is supposed to be used, different requirements have to be satisfied. Such requirements can be as diverse as, e.g., the regularity of the connectivity, the triangle aspect ratio, or the normal jump between adjacent triangles.

Hence, to prepare a given mesh for a specific application, we need *remeshing algorithms* which take a triangle mesh and resample it such that the new tessellation still approximates the same geometric shape but additionally satisfies some quality requirements [6].

Typical examples are the remeshing algorithms proposed in [3, 11, 10], where an arbitrary input mesh is converted into a mesh with semi-regular subdivision connectivity. In this case, the remeshing is necessary to apply certain multiresolution decomposition operators to the geometry.

[†] Lehrstuhl für Informatik VIII, RWTH–Aachen, Germany

© The Eurographics Association and Blackwell Publishers 2001. Published by Blackwell Publishers, 108 Cowley Road, Oxford OX4 1JF, UK and 350 Main Street, Malden, MA 02148, USA.

Mesh decimation schemes [5, 4, 8, 1, 13] in general also belong to this class of algorithms since they change the mesh connectivity while reducing the overall complexity. Removing vertices from the mesh can be considered as mere subsampling but a true re-sampling is performed if new vertex positions are generated during the decimation. Geometric and topological quality criteria can be used to control the algorithm, i.e., to decide in a greedy fashion which decimation step to apply next.

Surface fairing techniques [16, 2, 14] can be understood as special mesh optimization schemes which preserve the mesh connectivity but change the vertex positions in order to remove noise or to equalize the vertex distribution.

In [17, 7] remeshing techniques are proposed which are based on a particle system approach. For a given mesh, a new tessellation is generated by randomly distributing sample points on the continuous piecewise linear surface. A global relaxation procedure balances the distribution of the samples by shifting them on the surface.

Amplifying the attraction forces of the particle system according to the local curvature of the underlying surface yields a non-constant sampling density with increased resolution in those parts of the surface that contain fine detail. This is a significant advantage compared to those remeshing techniques that generate semi-regular meshes. Here the adaption of the sampling density to the local geometric resolution can only be achieved by *selective refinement*, e.g., by locally changing the dyadic level of resolution.

A general drawback of most geometry resampling techniques is the *alias problem*. If the geometry has sharp features, i.e., creases where the original geometry does not have a continuous tangent plane, we can often observe surface reconstruction artifacts (cf. Fig. 1). Although some mesh decimation techniques are quite reliable in preserving feature edges, we generally have the problem of finding the exact location of the features when placing the surface samples on the mesh.

There are several reasons for these difficulties. First, on a mesh which is a discrete approximation to a continuous surface in the first place, it turns out to be rather difficult to estimate curvatures in a reliable and robust fashion. Although recently many powerful curvature discretizations have been proposed [12, 15, 2], these operators still suffer from random noise affecting the vertex positions. Hence simple thresholding techniques to detect sharp features almost never work.

Another source for problems near sharp features is the fact that placing some samples on the feature does not guarantee a correct reconstruction. To achieve this, we additionally have to align mesh edges to the feature. This, however, implies that for correct feature reconstruction we have to change the mesh connectivity in general. Hence the task of resampling a given mesh requires geometric *and* topological optimization.

In this paper we propose a new technique that performs a feature sensitive resampling of a given triangle mesh. A global relaxation and restructuring procedure is used in a first step to obtain an equal distribution of mesh vertices across the original geometry. To exactly sample the features we then extend the particle system approach by an effective mechanism to snap the vertices to feature edges.

In the next section we first recapitulate the notion of *dynamic connectivity meshes* as a technique for integrated connectivity and geometry optimization. Then we explain the class of *parameterization-based fairing operators* which perform a relaxation step in the domain of the original surface's parameterization. In Section 4 we present our method for snapping the sample points to nearby sharp features. We finally combine these basic techniques to implement an automatic feature sensitive resampling procedure (Section 5).

2. Dynamic connectivity Meshes

The initial step in our algorithm is to set the resolution of the triangle mesh. Starting from the initial bounded 2–manifold mesh $\mathcal{M}_0$, we apply simple topological operators that insert or remove vertices. By iteratively adjusting the mesh complexity to a desired target resolution, we obtain our initial version of the remesh $\mathcal{M}$. Mesh decimation is a standard technique that covers the coarsening part of our algorithm. To increase the resolution we can refer to a wide variety of subdivision schemes. Since we are only concerned with the mesh connectivity in this section, we can skip the geometric aspects of these techniques but focus on the topological part. In particular we want to obtain a good quality mesh according to the following criteria:

- No edge should be shorter than $\varepsilon_{\min}$.
- No edge should be longer than $\varepsilon_{\max}$.
- A vertex' valence should be six.

In order to achieve this, we apply an algorithm similar to *dynamic connectivity meshes* [7].

First, we apply a *half-edge collapse* — which removes two triangles by collapsing one edge into a single vertex — to edges that fall below a length of $\varepsilon_{\min}$. In the second step edges longer than $\varepsilon_{\max}$ are bisected by midpoint insertion, which in turn splits the adjacent triangle(s) into two smaller ones. To regularize the connectivity, we perform edge-flipping. This becomes necessary since both edge-collapse and edge-split affect the vertex balance, and according to Euler's formula we want most vertices to have valence six. For every two neighboring triangles $\triangle(A,B,C)$ and $\triangle(A,B,D)$ we flip the common edge between A and B, if the total valence excess $\sum_{p\in\{A,B,C,D\}}(valence(p)-6)^2$ is reduced.

Now the desired resolution can be set by increasing $\varepsilon_{\min}$ (thus ensuring a certain edge length) and by scaling down $\varepsilon_{\max}$, which results in a mesh with higher resolution. Notice that $\varepsilon_{\max} > 2\varepsilon_{\min}$ has to be satisfied in order to avoid generating two invalid edges during the split operation. Certainly, simply adjusting the connectivity without caring about the vertex positions can cause geometrically degenerated configurations. Hence, the subject of the next section is the shifting of the vertices, which eventually leads to a well structured mesh and evenly distributed vertices.

3. Parameterization-based fairing operators

Having optimized the connectivity of $\mathcal{M}$ with respect to the above quality requirements, we can now focus on the remaining degrees of freedom in a polygonal mesh, the vertex positions. The vertices are to be distributed according to some quality criteria, e.g., aspect ratio and uniform size of the faces. The physical model behind our algorithm is to interpret the edges connecting the vertices of $\mathcal{M}$ as springs of equal length. Certainly we have to impose additional boundary conditions to ensure that the spring mesh leaves the vertices on the input surface $\mathcal{M}_0$ instead of collapsing into a single point when letting go. Therefore, we restrict the relaxing force in such a way that the vertices are allowed to shift on the surface of $\mathcal{M}_0$ only. This working model can be implemented by iteratively applying a relaxing operator **R** to the mesh.

A straight forward approach is to decompose **R** into two operators applied sequentially. A relaxing operator **U** that does not take the boundary conditions into account but merely optimizes the vertex distribution and a projection

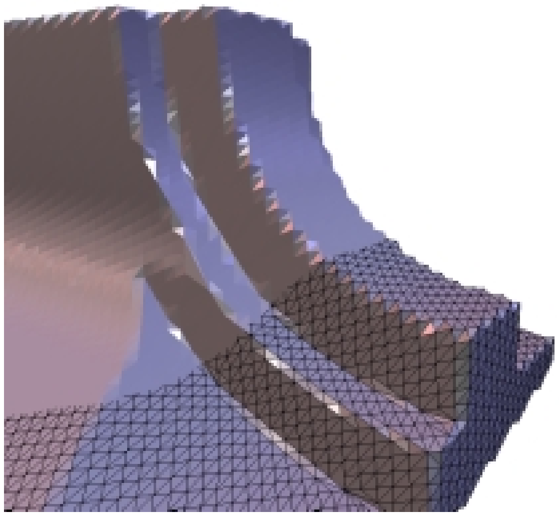

Figure 1: *Without taking care about feature lines, aliasing artifacts cannot be avoided even though a regular sample grid is used. Edge-flipping and aligning the sample grid with the features slightly improves the situation but still, sharp edges are lost.*

operator **P** that shifts a vertex **v** back onto $\mathcal{M}_0$. However, in particular in surface regions with high local curvature, **P** often fails or leads to counter productive results. Replacing **P** by an operator that shifts **v** to the closest point on $\mathcal{M}_0$ slightly improves the situation but the mapping is still not injective and surface artifacts like folding still occur.

This problem can be solved by *locally* parameterizing subregions of $\mathcal{M}_0$ and $\mathcal{M}$ in the vicinity of a vertex **v**. Doing that, **P** can be left out such that **R** and **U** are equivalent. Applying **U** to this planar configuration followed by an evaluation of the parameterization finally yields the updated 3D position of **v**. Since finding a parameterization is computationally involved and since this would have to be done for every vertex in $\mathcal{M}$ and for every iteration step, we propose the computation of a *global* parameterization in a preprocessing step.

Moreover, this global map provides additional benefits. Having differently detailed approximations $\mathcal{M}$ and $\mathcal{M}'$ to $\mathcal{M}_0$ linked by the parameterization gives direct access to multiresolution semantics.

The shrink wrapping approach to remeshing [10] is one instantiation of a scheme that uses a global parameterization of $\mathcal{M}_0$ combined with a relaxing operator to equally distribute the vertices of $\mathcal{M}$ on $\mathcal{M}_0$. It uses a spherical parameterization and thus works for genus–zero objects only. Additionally, the surface metric of $\mathcal{M}_0$ has influence on the relaxing operator **U**, since a spherical parameterization cannot be close to isometric for arbitrary input shapes.

For this reason, we use the MAPS parameterization[11]. The MAPS algorithm iteratively creates a coarse version of $\mathcal{M}_0$ that serves as a parameter domain $\mathcal{D}$ and uses conformal maps to get similar metrics on $\mathcal{D}$ and $\mathcal{M}_0$. In order to locally flatten out domain triangles into the plane, we adopt the arguments of the original paper and use hinge and exponential maps, respectively. Since the number of triangles in $\mathcal{D}$ is orders of magnitude lower than in $\mathcal{M}_0$, this can also be done in a preprocessing step with low additional storage costs. Having this at hand, we can plug in the uniform 2D Umbrella operator[9, 16] as relaxing operator **U**. Evaluating the parameterization maps the vertices back to $\mathcal{M}_0$. Due to the conformal mapping we get equally distributed and well shaped triangles.

4. Alias reduction by feature snapping

The previous step distributes the vertices equally, and the triangle mesh is considered perfectly shaped in the parameter domain and hence on $\mathcal{M}_0$. But as the relaxed vertices still do not sample feature regions accurately, the reconstructed mesh still suffers from aliasing (cf. Fig. 1). In this section we propose a technique that reduces aliasing by snapping vertices to features of the original surface. In Section 5 these techniques will be integrated in the overall algorithm.

The basic idea is to start a relaxation step with different forces that make the vertices move towards feature regions of $\mathcal{M}_0$. We do not explicitly classify these regions by thresholding curvature or more sophisticated techniques as used e.g. in the context of surface segmentation in reverse engineering (cf. [18]). This avoids the use of any application specific parameters. As a consequence we prefer the metaphor of vertices that move into direction of higher curvature to that of a force that is induced by the feature and pulls vertices. So the question is, what makes vertices move?

We define a scalar curvature field on the original surface $\mathcal{M}_0$ whose gradient provides the (candidate) directions towards which the vertices can move. With the MAPS parameterization there is a correspondence between the two meshes, and points on $\mathcal{M}_0$ can be mapped to points on $\mathcal{M}$ and vice versa. Having this one–to–one correspondence, directions defined on $\mathcal{M}_0$ can be transferred to directions or motion of vertices on $\mathcal{M}$. At first, we will focus on the definition of the curvature field and treat the details of vertex shifting after that.

Mathematically, curvature is defined for smooth, differentiable surfaces only. Regarding a piecewise linear triangle mesh as an approximation to a continuous surface leads to curvature discretizations as proposed e.g. in [12, 15, 2]. For our purpose a rather simple curvature measure is sufficient, as we do not need convergence to a smooth surface in the limit case. We require that flat regions have smaller curvature than feature edges that in turn have lower curvature than "corners". This setting allows vertices from flat regions to snap onto edges first, and edge vertices to snap to corners.

We calculate the curvature at a vertex **v** as follows: regard all edges in **v**'s 1-ring. For every edge we calculate the angle between the normals of the two attached triangles. The sum of all these angles is the curvature value of this vertex. It is straightforward that this measure is reasonable according to our requirements. This can be illustrated for a cube: the eight corner vertices have curvature $\frac{3}{2}\pi$, on edges the curvature is π, and all other vertices in the flat regions have zero curvature. Note that this measure is independent of the tessellation of the surface, as vertex valences etc., are not taken into account. Applying linear interpolation inside triangles we can assign a curvature value to every point on the surface.

Using this simple measure there is one circumstance we have to pay attention to. Suppose two vertices with high curvature are connected by an edge. Although this edge may be in a flat region, i.e., the normal deviation of its attached triangles is zero, it is assigned a high curvature due to linear interpolation between the two high curvature vertices. This situation can be avoided by preprocessing the original mesh $\mathcal{M}_0$: bisecting every edge (cf. Sec. 2) is a straightforward solution to this problem. In practice we restrict the splitting to regions with noticeable curvature. This is done for efficiency only. The induced curvature threshold does not affect the rest of the algorithm, in particular it is not used for classification but just truncates the very low curvature. Besides, this is no problem in principle, but just an artifact caused by linear interpolation of the simple curvature measure.

From the scalar curvature field we can construct a gradient field that indicates the direction in which a vertex should move in order to snap to some feature. However, the naïve approach of calculating the curvature gradient for every triangle greatly suffers from high frequency noise effecting more or less random vertex shifting. Imagine e.g. a planar region: even minimal geometric disturbance will produce a meaningless direction field. Our aim is to move vertices towards sharp feature edges of $\mathcal{M}_0$ while minimizing disturbance by noise.

In order to fit our needs we make use of a noise reduction scheme that smoothes the curvature and hence the gradient field. The initial scalar values are low pass filtered by weighted averaging. This is iterated several times resulting in a hierarchy of curvature fields on different frequency bands. So the gradient of the lowest frequency field points reliably towards sharp feature edges over a wide region, but smaller features may be missed. In contrast, the higher frequency fields respect smaller features and noise and are thus less reliable.

The idea is to first move vertices along the direction of the lowest frequency gradient. Higher frequency bands of the scalar curvature field are subsequently faded in by taking the maximum of the values of different levels in the hierarchy. This defines new gradients on successive levels respecting smaller features while the "support" of a feature depends on its significance. The filtered curvature field provides sharpness in the vicinity of features and smoothness elsewhere as illustrated by Fig. 2. We found that in practice it is sufficient to use the gradients of *this* scalar field for moving vertices rather than to successively shift vertices according to gradients on different smoothing levels.

Certainly, the number of smoothing levels could be used to suppresses small features and hence to serve as feature selection. This is not the intention of the algorithm: the main purpose of filtering is to provide a meaningful gradient field over a considerably large support with enough candidate vertices to be attracted to the features.

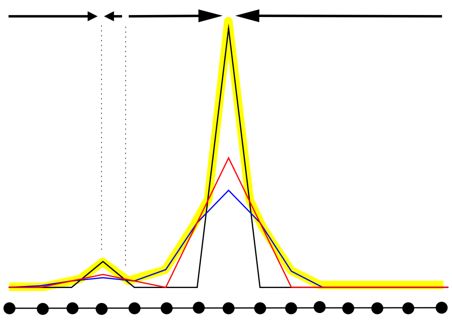

Figure 2: *Curvature filtering. This example shows a 1D setup of vertices with their curvature values at levels 1 (input values), 2 and 3 of the smoothing hierarchy (y-axis). We use the curvature field that is defined by the maximum curvatures over all levels (thick line). The arrows show the direction of the resulting gradient field in the different intervals. Notice how the smoothing has enlarged these intervals allowing more distant vertices to snap onto the sharp features.*

Ideally, a vertex is moved along the filtered direction field. If a local maximum in curvature is detected on its route, the vertex snaps to this position. In order to avoid a constant flow, the vertex will remain at its original position if there is no local maximum. As curvature maxima can appear only on the edges of the original mesh $\mathcal{M}_0$ it is sufficient to test for maxima on (intersections with) these edges only.

Using this snapping technique we can reconstruct features of the original surface by plain geometric optimization. Of course, the quality of the resulting remesh is unacceptable without introducing additional constraints that prevent degeneration such as orientation flipping of triangles. This is achieved by restricting a vertex' movement to the "kernel" of its 1-ring. The kernel is the convex subregion of the 1-ring that is bounded by the straight lines defined by outer edges. Fig. 3 shows an example.

When shifting the vertex inside its kernel along the gradient field, caps may arise (Fig. 3, center left). These are triangles with almost zero area and one inner angle close to π. We must avoid such configurations as neither a snapped vertex will move any further nor the topological optimization

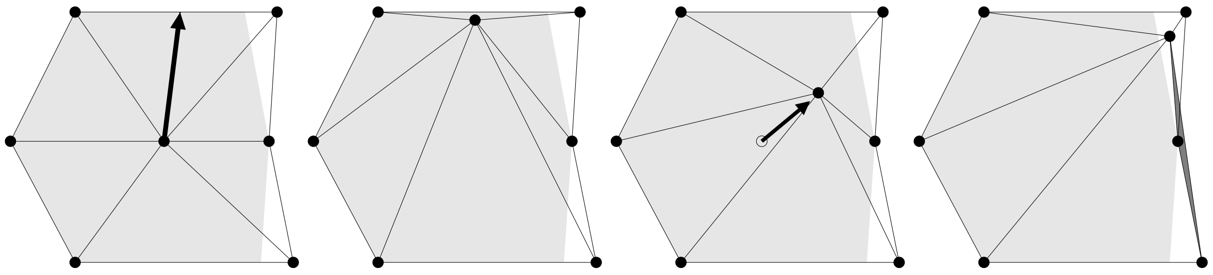

Figure 3: *Starting from the current position (left), the vertex is shifted with respect to the gradient field indicated by the black arrow. Moving in the direction of the gradient can cause caps in the vicinity of features (center left). This can be avoided by shifting along the edge that encloses the smallest angle with the gradient (center right). A potential short edge gets removed during the restructuring process (cf. Sec. 2). Note that the shift has to be restricted to the kernel of the 1-ring (grey area) to avoid flipping of triangles (right).*

(cf. Sec. 2) can remove them (in general). For this reason we discretize directions: curvature maxima are searched on the edge emerging from a vertex $\mathbf{v}_i$ of the current remesh $\mathcal{M}$ that encloses the smallest angle with the gradient at $\mathbf{v}_i$ (Fig. 3, center right). Only edges with an enclosing angle less then $\frac{\pi}{2}$ are allowed. This is always true for at least one edge inside a consistent mesh. Restricting the direction this way ensures that vertices are still moved along the gradient, merely the convergence to features is slowed down.

Hence, the direction along that a vertex $\mathbf{v}_i \in \mathcal{M}$ is attracted towards the feature always points to some neighbor $\mathbf{v}_j$. We now evaluate curvature on $\mathcal{M}_0$ at the intersections of the edge between $(\mathbf{v}_i, \mathbf{v}_j)$ with all edges of $\mathcal{M}_0$ (cf. Fig. 5). If a local curvature maximum is detected the vertex is moved to this position, else it remains untouched. In order to avoid constant flow, vertex movement is restricted: $\mathbf{v}_i$ may not move further than half of the length of the edge $(\mathbf{v}_i, \mathbf{v}_j)$. The rationale for this damping is that the neighbor $\mathbf{v}_j$ may reach the same position easier. In the extreme case $\mathbf{v}_j$ lies on the feature already and collapsing of vertices is prevented. Vertices that do not snap to a feature are relaxed (cf. Sec. 3) instead of moving along the gradient. This ensures that vertices remain equally distributed.

Vertices that have been shifted onto an edge of $\mathcal{M}_0$ due to a curvature maximum are restricted to move only on edges in $\mathcal{M}_0$ in the following stages of the algorithm. This allows us to use the same snapping technique in a one dimensional subspace of the parameter domain in order to resample corners of the original surface $\mathcal{M}_0$. Again, a vertex $\mathbf{v}_i \in \mathcal{M}$ tries to move towards high curvature, but this time along edges in $\mathcal{M}_0$. Its movement is still restricted to the kernel of the 1-ring. If a local maximum is found at a vertex $\mathbf{v} \in \mathcal{M}_0$ a corner is assumed, and $\mathbf{v}_i$ is moved to that position. $\mathbf{v}_i$ remains untouched either in case of there is no maximum or if there is another vertex $\mathbf{v}_j \in \mathcal{M}$ already sampling $\mathbf{v}$. All vertices that have not snapped to corners are finally relaxed on their feature edges in $\mathcal{M}_0$.

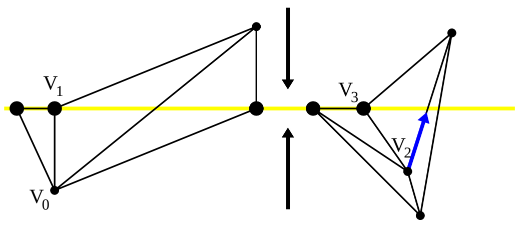

Figure 4: *The gradient field (black arrows) shifts vertices towards a feature (horizontal line). Vertex $\mathbf{v}_0$ is locked by an adjacent vertex $\mathbf{v}_1$ that is already residing on the edge (left). Flipping the edge that crosses the feature line aligns $\mathcal{M}$ with $\mathcal{M}_0$. On the right, vertex $\mathbf{v}_2$ moves onto the horizontal line, generating a degenerated triangle. Relaxing $\mathbf{v}_3$ fixes this problem.*

After feature snapping to edges (in 2D) and to corners (in 1D) the vertices of the remesh $\mathcal{M}$ are equally distributed, and they sample feature edges and corners of the original surface. Now we make sure, that edges connecting vertices in $\mathcal{M}$ are aligned to sharp features in $\mathcal{M}_0$. Due to the previously described discretization of directions for vertex movement, a vertex might be locked which leads to an edge crossing a feature of $\mathcal{M}_0$. Fig. 4 (left) illustrates the situation. These cases rarely happen due to the relaxation and can be solved by flipping the affected edge. Since the procedure can be applied iteratively, the algorithm works in the same fashion for several edges attached to one vertex.

In a final step we take care of triangles with poor aspect ratios. Such a configuration can occur, if the vertex is shifted towards the border of the kernel in a non-convex 1-ring. Fig. 4 (right) shows such a configuration. However, these cases can easily be solved by applying the relaxation operator to the vertex with the largest enclosing angle.

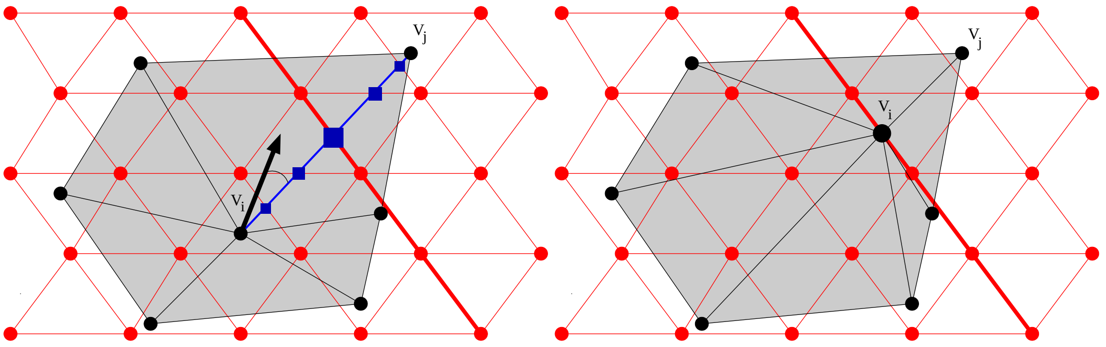

Figure 5: *Feature snapping of a single vertex* $\mathbf{v}_i$*. Left: The regular, light mesh denotes the original mesh* $\mathcal{M}_0$ *with a feature edge (thick). The dark mesh shows a 1-ring of* $\mathcal{M}$ *with the sampled curvature gradient (arrow). All vertices are in free state. The search direction is determined by the edge enclosing the smallest angle with the gradient. Curvature values* c_k *are sampled at the intersections with* $\mathcal{M}_0$*, displayed by the boxes that are scaled by* c_k*. Right: The center vertex snaps to the local maximum of the* c_k *and changes its state to snapped–to–edge.*

5. Overall algorithm

In the previous sections we discussed the components of our framework and elaborated on the different aspects. Now we are able to outline the overall algorithm:

Algorithm (Feature Sensitive Remeshing)

(a) Restructure the mesh.
(b) Apply parameterization based relaxation.
(c) Run feature snapping algorithm in 2D (to edges).
(d) Run feature snapping algorithm in 1D (to corners).
(e) Optimize the resulting mesh by edge flipping.

We assign every vertex $\mathbf{v}_i$ of the remesh $\mathcal{M}$ a state that will be evaluated and modified by the algorithm. A vertex can either have status *free*, *snapped–to–edge* or *snapped–to–corner* (Fig. 6). Initially, all vertices are *free*.

In step (a) an initial remesh is built using dynamic connectivity meshes as described in Sec. 2. The resulting mesh has the desired complexity in the number of vertices and is optimal with respect to the connectivity criteria. This step resets all vertex states to *free*.

The optimization only effects the connectivity of the triangle mesh but not its geometry. In the next step (b) we start the geometric optimization (cf. Sec. 3): the positions of all *free* vertices are shifted in the parameter domain by the fairing operator. This results in a mesh with a uniform vertex distribution.

Still, the surface is an unacceptable reconstruction of the original geometry due to aliasing in feature regions. Naïve local refinement cannot solve this problem. So we have to ensure that vertices of the remesh $\mathcal{M}$ lie on the features of $\mathcal{M}_0$.

Step (c) snaps vertices of $\mathcal{M}$ to feature edges of $\mathcal{M}_0$ (cf. Sec. 4). The filtered curvature field on $\mathcal{M}_0$ provides a gradient direction that guides vertices towards features. Note that no feature regions or edges are classified on $\mathcal{M}_0$, so no application specific parameters are needed. As a vertex $\mathbf{v}_i \in \mathcal{M}$ may only move towards a feature edge along an edge $(\mathbf{v}_i, \mathbf{v}_j)$ only intersections of $\mathcal{M}_0$ with this edge are new candidate positions for $\mathbf{v}_i$. Curvature is evaluated on $\mathcal{M}_0$ and used as the snapping criterion to find the target position.

If a local curvature maximum is encountered between $\mathbf{v}_i$ and $\mathbf{v}_j$ then a feature edge is assumed. As a consequence $\mathbf{v}_i$ is snapped to the intersection point of this maximum, and its state changes to *snapped–to–edge*. Fig. 5 illustrates this situation. If there is no such maximum, $\mathbf{v}_i$ stays at its position thus avoiding constant flow and degenerated edges. Also, $\mathbf{v}_i$ is not allowed to snap onto a *snapped–to–edge* vertex to avoid edge collapses.

Vertices that remain in state *free* may now be scheduled for another relaxation step. Steps (b) and (c) are iterated until no more *snapped–to–edge* vertices are generated. This ensures that there are enough candidates for snapping. Now vertices of the resulting remesh $\mathcal{M}$ sample edges of the original surface, but still corners are subject to aliasing.

It turns out that the corner problem can be solved just in the same way as the edge problem. We need to snap vertices to corners of the original surface, e.g., to vertices where feature edges meet. In contrast to step (c) the search space is restricted from the entire 2D parameter domain to the 1D feature edges. We have already detected these edges, as they connect *snapped–to–edge* vertices.

So step (d) can be formulated as follows: For every vertex $\mathbf{v}_i \in \mathcal{M}$ with state *snapped–to–edge* the curvature values at the two endpoints of the corresponding edge in $\mathcal{M}_0$ are used to select one of the two possible search directions. If the vertex happens to lie on a vertex in $\mathcal{M}_0$ and there are more than two candidate directions, the vertex remains untouched as it is assumed that a corner is found already.

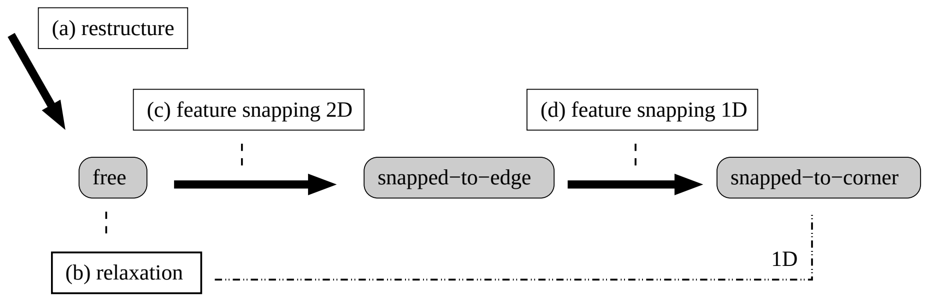

Figure 6: *Vertex states in overall algorithm.*

Again, we are looking for a local maximum between $\mathbf{v}_i$ and its neighbor $\mathbf{v}_j$ with the same state. If a curvature maximum is detected, $\mathbf{v}_i$ snaps to the associated vertex in $\mathcal{M}_0$ and its state changes to *snapped–to–corner*. If the maximum happens to be in $\mathbf{v}_j$, the vertex $\mathbf{v}_i$ is left unchanged. The vertices remaining in state *snapped–to–edge* are scheduled for relaxation on their feature edge. Corner snapping and 1D relaxation are iterated until there are no more vertices that snap to corners. Now $\mathcal{M}$ reconstructs edges and corners of the original surface $\mathcal{M}_0$ in so far as vertices are placed on feature edges resp. corners.

The last step (e) in the algorithm fixes edges that are still not aligned and cross a feature. This is achieved by edge flipping. This is in contrast to the requirements of balanced number of adjacent vertices, but has to be tolerated for the sake of an improved approximation to $\mathcal{M}_0$.

6. Results

We illustrate different stages of our algorithm on the fandisk model (13K△). This technical data set contains flat and smooth regions as well as sharp edges (straight and curved), and hence is ideally suited as a test data set for our algorithm. Fig. 7 (top left) shows the original surface. In a preprocessing step we first compute curvature values as depicted in Sec. 4.

The initial triangle mesh is then parameterized using the MAPS algorithm. Note that we do not tag any feature edges, e.g., by thresholding curvature or dihedral angles. For now, we do not adapt the number of vertices of the remesh (cf. Sec. 2), but start the algorithm with the initial connectivity.

Fig. 7 (top right) shows the remesh after application of the parameterization-based fairing operator. While this balances the vertex distribution, the features of the original surface are not reproduced anymore. This is fixed by snapping vertices to features followed by edge flipping (cf. Sec. 4). Fig. 7 (bottom left) shows the triangulation of the final remesh. Snapped vertices have been marked. Fig. 7 (bottom right) displays the flat shaded surface. The features of the original fandisk have been constructed nicely.

Fig. 8 shows a coarse and a fine approximation to the initial mesh in an enlarged view. We set the target resolutions to 2.8K△ and 40K△ respectively. Since we use the MAPS algorithm to preprocess a parameterization, the coarsest resolution is restricted by the resolution of the parameter domain. We always have to make sure, that a 1-ring in our remesh can be parameterized over a 2D plane.

7. Conclusions

We presented an automatic technique for feature sensitive remeshing of triangulated surfaces. This is done by extending a particle system approach with an effective mechanism to avoid resampling artifacts. Our algorithm does not rely on a threshold to detect sharp edges in the input mesh but operates without any model specific parameters. The target resolution can be changed dynamically. During the remeshing procedure, a global parameterization is generated that links all resulting remeshes. Hence we obtain a multiresolution structure where detail information can be transfered between different resolutions. This gives rise to many applications and future investigations such as multiresolution deformations. Also texture coordinates can be remapped to preserve surface attributes.

Acknowledgements

Thanks to Kolja Kähler for his helpful comments and for providing the MAPS implementation.

References

1. P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, 1998. 1

2. M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH 99 Conference Proceedings*, pages 317–324, 1999. 1, 2, 3

© The Eurographics Association and Blackwell Publishers 2001.

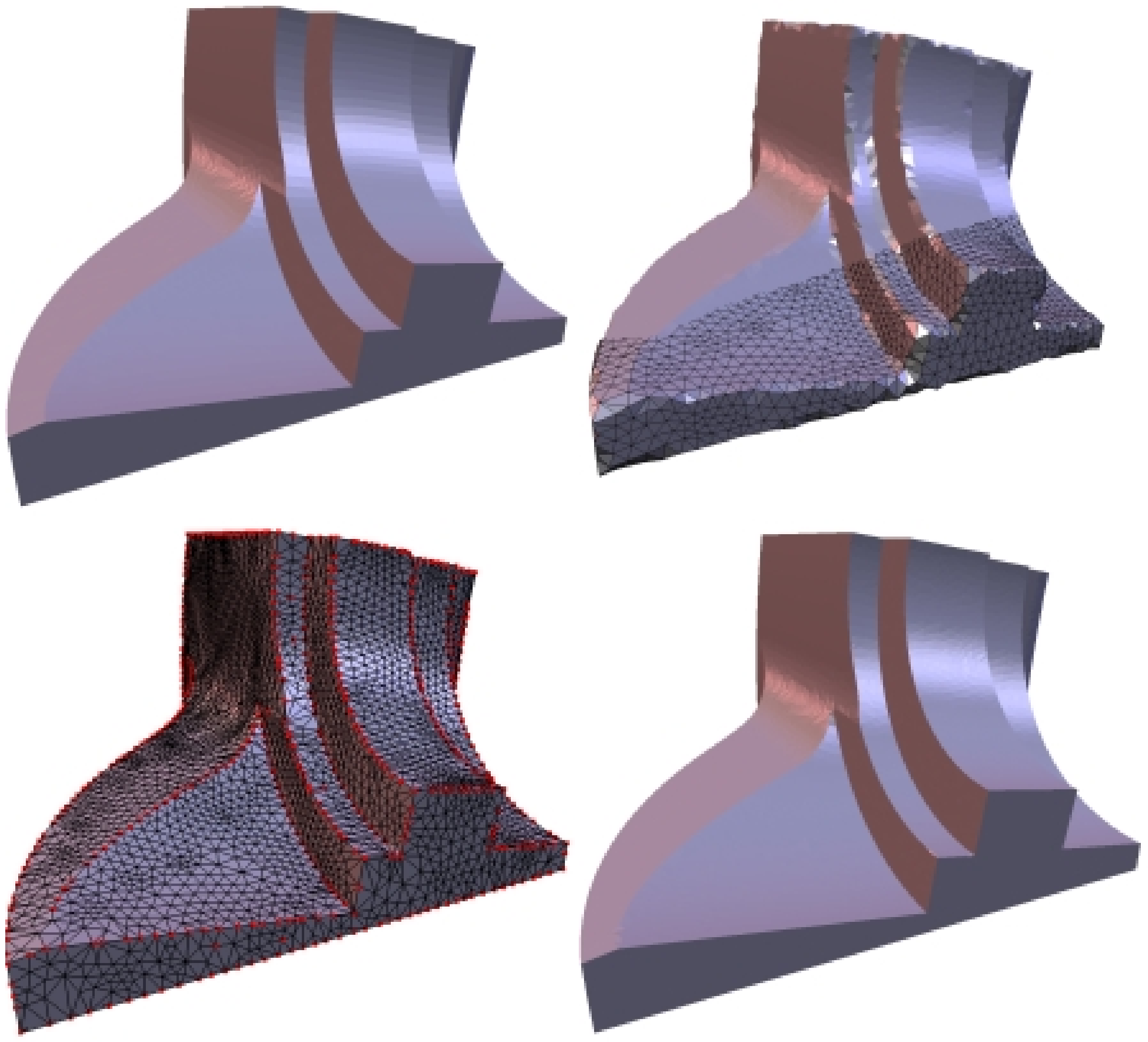

Figure 7: *Some steps of our algorithm. Top left: original surface (about 13K△). Top right: remesh at same resolution after parameterization-based fairing. Bottom left: resulting remesh after feature snapping; snapped vertices are marked. Bottom right: final result, flat shaded.*

3. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH 95 Conference Proceedings*, pages 173–182, 1995. 1

4. M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings*, pages 209–216, 1997. 1

5. H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 99–108, 1996. 1

6. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH 93 Conference Proceedings)*, pages 19–26, 1993. 1

7. L. Kobbelt, T. Bareuther, and H.-P. Seidel. Multiresolution shape deformations for meshes with dynamic vertex connectivity. In M. Gross and F. R. A. Hopgood, editors, *Computer Graphics Forum (Eurographics 2000)*, volume 19(3), pages 249–260, 2000. 1, 2

8. L. Kobbelt, S. Campagna, and H.-P. Seidel. A general framework for mesh decimation. In *Graphics Interface*, pages 43–50, 1998. 1

9. L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Sei-

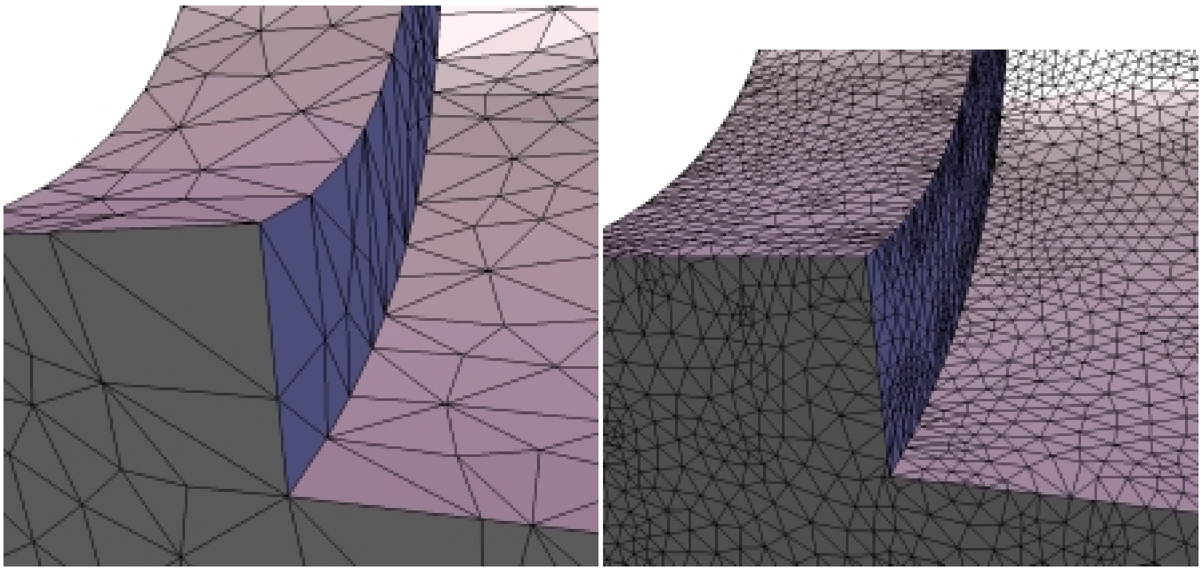

Figure 8: *Running the algorithm at different resolutions. Left: final remesh at lower resolution (2.8K△). Right: remesh with increased resolution (40K△).*

del. Interactive multi–resolution modeling on arbitrary meshes. In *SIGGRAPH 98 Conference Proceedings*, pages 105–114, 1998. 3

10. L. Kobbelt, J. Vorsatz, U. Labsik, and H.-P. Seidel. A shrink wrapping approach to remeshing polygonal surfaces. *Computer Graphics Forum*, 18(3):119–130, 1999. 1, 3

11. A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. In *SIGGRAPH 98 Conference Proceedings*, pages 95–104, 1998. 1, 3

12. H. P. Moreton and C. H. Séquin. Functional optimization for fair surface design. In *Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, volume 26, pages 167–176, 1992. 2, 3

13. J. Rossignac. Simplification and compression of 3d scenes. In *Eurographics '97 Tutorial*, 1997. 1

14. R. Schneider and L. Kobbelt. Generating fair meshes with G^1 boundary conditions. In *Proceedings Geometric Modeling and Processing*, pages 251–261, 2000. 1

15. G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral. In *Proc. International Conference on Computer Vision*, pages 902–907, 1995. 2, 3

16. G. Taubin. A signal processing approach to fair surface design. In *SIGGRAPH 95 Conference Proceedings*, pages 351–358, 1995. 1, 3

17. G. Turk. Re–tiling polygonal surfaces. In *Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, volume 26, pages 55–64, 1992. 1

18. T. Várady and P. Benkő. Reverse engineering B-rep models from multiple point clouds. In *Proc. of Geometric Modeling and Processing 2000*, pages 3–12, 2000. 3