Dynamic Remeshing and Applications

J. Vorsatz, Ch. Rössl, and H.-P. Seidel

Max-Planck-Institut für Informatik
Saarbrücken, Germany

ABSTRACT

Triangle meshes are a flexible and generally accepted boundary representation for complex geometric shapes. In addition to their geometric qualities or topological simplicity, intrinsic qualities such as the shape of the triangles, their distribution on the surface and the connectivity are essential for many algorithms working on them. In this paper we present a flexible and efficient remeshing framework that improves these *intrinsic* properties while keeping the mesh geometrically close to the original surface. We use a particle system approach and combine it with an incremental connectivity optimization process to trim the mesh towards the requirements imposed by the user. The particle system uniformly distributes the vertices on the mesh, whereas the connectivity optimization is done by means of *Dynamic Connectivity Meshes*, a combination of local topological operators that lead to a fairly regular connectivity. A dynamic skeleton ensures that our approach is able to preserve surface features, which are particularly important for the visual quality of the mesh. None of the algorithms requires a global parameterization or patch layouting in a preprocessing step but uses local parameterizations only. In particular we will sketch several application scenarios of our general framework and we will show how the users can adapt the involved algorithms in a way that the resulting remesh meets their personal requirements.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*

General Terms

Algorithms

Keywords

Remeshing,Dynamic Meshes,Multiresolution Modeling

1. INTRODUCTION

Nowadays the literature on triangle meshes comprises a huge amount of excellent work and is still growing rapidly. The virtual mesh-processing pipeline starting from acquisition down to rendering is well covered, thus triangle meshes have become an appealing surface representation.

Opposed to meshes that, e.g., represent characters in the animation-/game industry which are often hand-made and highly optimized with respect to triangle count, visual quality and kinematics, we turn our attention to densely sampled triangle meshes often stemming from 3D scanning devices or volume extraction [12, 18].

Even though a mesh might be 2-manifold without any unwanted holes or handles, in practice, the given tessellation often does not satisfy the requirements imposed by an application in a later step of the pipeline. Obviously, even for a prescribed approximation tolerance, there exist a multitude of tessellations. The requirements can be as diverse as a regular connectivity, bounded minimal angles of the triangles or the alignment of edges along surface features. In order to prepare a given mesh for a specific application, a variety of *remeshing algorithms* have been proposed. Among those are mesh decimation algorithms [10, 8, 21], semi–regular remeshing schemes [6, 19, 9, 15] and irregular remeshing [12, 3, 25].

The dynamic remeshing framework we present is in spirit similar to *Feature Sensitive Remeshing (FSR)* [25]. We link our remesh $\mathcal{M}$ to a domain mesh $\mathcal{D}$ and use a particle system on $\mathcal{M}$ for an equal vertex distribution. Moreover, $\mathcal{M}$ is a *Dynamic Connectivity Mesh (DCM)* [13] that provides incremental changes of the connectivity.

Our approach differs from [25] in that we are not building our particle system upon a (precomputed) global parameterization, instead we are using a set of *local* parameterizations of small, bounded parts of $\mathcal{D}$. This is done for the following reason. In order to be as flexible as possible, we make sure that we have a consistent mapping for each 1–ring and that we are not limited by the base-domain/chart-boundaries of some underlying global parameterization.

In the next section we describe a particle-system approach which lets vertices of $\mathcal{M}$ float on $\mathcal{D}$ in order to redistribute them equally. This way we achieve an isotropic remeshing of the original geometry. In this context we will show that different (local) parameterization methods influence the relaxation process and we will discuss how to construct parameterizations based on a minimal local domain. Section 3 briefly recapitulates the notion of *DCM*, a technique for integrated connectivity optimization. After that we explain how the vertex-relaxation in the particle-system and Dynamic Connectivity Meshes are combined. Section 4 introduces the notion of a skeleton that enables us to preserve important surface features. Finally, section 5 sketches some application scenarios for our dynamic remeshing approach. In particular we will go into interactive multiresolution modeling, semi-regular- and interactive remeshing.

2. RELAXATION ON THE DOMAIN

We optimize the vertex distribution of $\mathcal{M}$ by relaxing its vertices while restricting their positions to the domain $\mathcal{D}$. We define a particle-system that allows the vertices to float on the original surface represented by $\mathcal{D}$. A relaxation operator locally repositions a vertex with respect to its direct neighbors (1-ring). This optimization process iteratively applies the local relaxation, in a physical interpretation this minimizes the energy of a global system of spring-edges connecting the vertices.

The relaxation is done in a 2D parameter domain. So we need to be able to parameterize regions of $\mathcal{M}$ over the plane. For this purpose we first parameterize $\mathcal{M}$ over the domain mesh $\mathcal{D}$ by assigning to every vertex $v_i \in \mathcal{M}$ the domain triangle $\tilde{\Delta}_j \in \mathcal{D}$ that includes v_i and barycentric coordinates w.r.t. $\tilde{\Delta}_j$. We call the mapping

$$v_i \mapsto \left(\tilde{\Delta}_j, (\alpha_{i1}, \alpha_{i2}, \alpha_{i3})\right), \sum_{j=1}^{3} \alpha_{ij} = 1 \wedge \alpha_{ij} \geq 0$$

of all vertices $v_i \in \mathcal{M}$ the *link* between $\mathcal{M}$ and $\mathcal{D}$. So if we flatten a region of $\mathcal{D}$ to the plane, the link will provide us also a mapping of the associated vertices and triangles of $\mathcal{M}$ to the plane.

Now we define the local *relaxation operator* $\mathcal{U}$ as the so called *weighted* $2D$ *Umbrella* operator. It shifts a vertex v_i depending on a convex combination of its direct neighbors v_{ij}. Let p_i and p_{ij} be the respective parameter values obtained from the link to $\mathcal{D}$. Then $\mathcal{U}$ is defined as

$$\mathcal{U}(p_i) := p_i + \frac{1}{\sum_{j=0}^{n} \omega_{ij}} \sum_{j=0}^{n} \omega_{ij}(p_{ij} - p_i)\,, \quad \omega_{ij} \geq 0$$

The appropriate choice of the weights ω_{ij} will be discussed at the end of this section. The Umbrella operator $\mathcal{U}$ shifts vertices in the parameter domain, that are lifted back to 3-space using the link. This allows the restriction to $\mathcal{D}$ without expensive and error-prone projection operators.

The actual relaxation process is similar to parameter-based fairing applied in FSR [25] where a *global* parameterization (MAPS [19]) of $\mathcal{D}$ is used. In contrast to FSR (and also e.g. [2, 3]), we give up the global parameterization and replace it by a set of *local* parameterizations mapping regions of $\mathcal{D}$ to the plane. The benefit is that we do not have to rely on the quality of the global parameterization that may even be hard to construct. For a global parameterization of $\mathcal{D}$ that is defined over a coarse base domain, this prescribed domain limits the coarsest resolution of the remesh, as the size of a 1-ring in $\mathcal{M}$ is then restricted to that of a 1-ring in $\mathcal{D}$. On the other hand, for global flattening methods that induce cuts to the original surface, it is not clear how the relaxation operator should behave if a 1-ring intersects a cut. The price we have to pay with the new approach is the on-the-fly construction of the local parameterizations which cannot be done in a preprocessing step anymore. For this reason we keep the domains of the different local parameterizations as small as possible and apply caching whenever feasible.

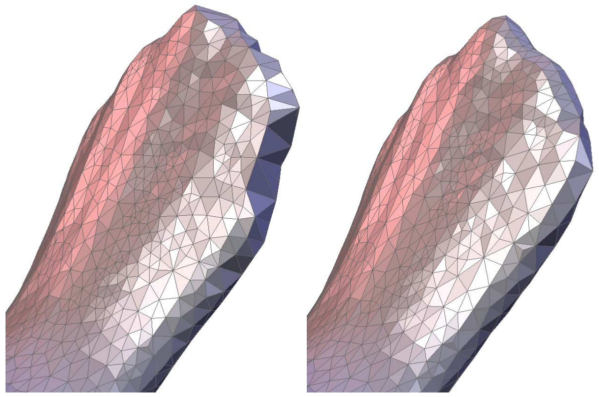
Figure 1: A closeup view of Tweety's tail. Projection into a fitting plane leads to degeneracies in areas with high curvature (left), whereas a local mapping with Floater's shape preserving parameterizations performs well (right).

We construct a set of local parameterizations $\{\Phi_i\}$. Each parameterization Φ_i provides a piecewise linear mapping from a set of domain triangles – the *local domain* $\mathcal{D}_L^i \subset \mathcal{D}$ – to the plane. With the *link* to the remesh $\mathcal{M}$ we can thus map associated vertices and triangles of $\mathcal{M}$. When we start remeshing with $\mathcal{M} = \mathcal{D}$ the initial link provides a 1-to-1 mapping between the corresponding vertices. Formally, for a vertex $v \in \mathcal{M}$ corresponding to $\tilde{v} \in \mathcal{D}$ we can choose a single triangle $\tilde{\Delta} \in \mathcal{D}$ that contains $\tilde{v}$ as its local domain.

As we start the optimization process, $\mathcal{U}$ will compute a shift vector for v from a convex combination of the direct neighbors v_j in the 1-ring. Hence shifting v requires a larger domain, since a parameterization of the 1-ring is needed. We try to keep this local domain small as well as the updates on it. So we restrict the shift to parameter points located in the intersection of the 1-ring $\Delta^1(v) \subset \mathcal{M}$ and the subset $\mathcal{D}_L^v \subset \mathcal{D}_L$ of triangles that this vertex v is assigned to. This way we ensure that the new position is well defined, i.e. has a mapping in the parameterization, and we avoid a more expensive point-in-triangle test.

This determines the *minimal domain* that has to be covered by the local parameterization. It consists of the union of

- triangles of $\mathcal{D}_L$ covered by the triangles of v's 1-ring, i.e. $\{\tilde{\Delta} \in \mathcal{D}_L | \Phi(\tilde{\Delta}) \cap \Phi(\Delta^1(v)) \neq \emptyset\}$,

and $\mathcal{D}_L^v$ which is defined as

- a single triangle $\tilde{\Delta} \in \mathcal{D}_L$, if the vertex is inside this triangle, i.e. $\Phi(v) \in \Phi(\tilde{\Delta})$, or
- two neighboring triangles $\tilde{\Delta}_1, \tilde{\Delta}_2 \in \mathcal{D}_L$, if v is located on their common edge, i.e. $\Phi(v) \in \Phi(\tilde{\Delta}_1 \cap \tilde{\Delta}_2)$, or
- a 1-ring in $\mathcal{D}_L$, if v corresponds to a vertex $\tilde{v}$ in $\mathcal{D}_L$, i.e. $\Phi(v) = \Phi(\tilde{v})$.

Fig. 2 shows the three different cases.

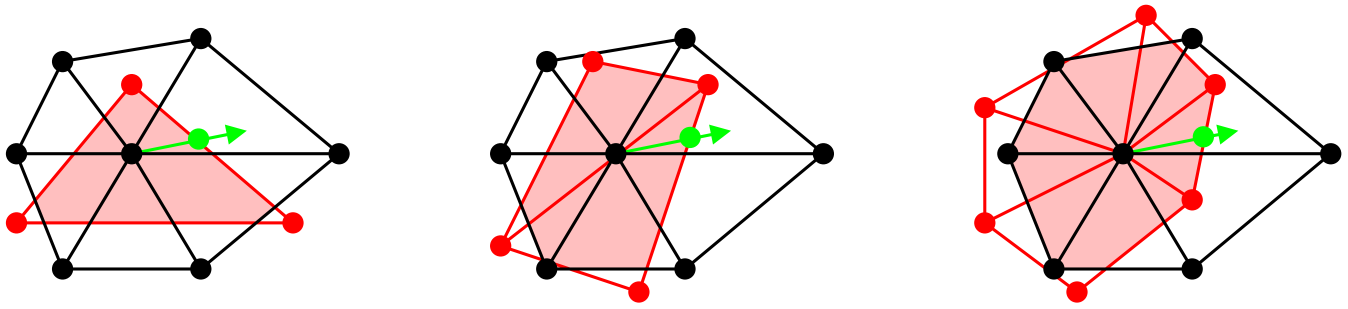

Figure 2: Example for calculating the shift vector for the vertex inside triangle (left), on edge (middle), and on vertex (right) case. The remesh $\mathcal{M}$ is drawn in wireframe, the domain mesh $\mathcal{D}$ is shaded. The arrow shows the vector calculated by the relaxation operator $\mathcal{U}$, the shift is restricted to the shaded area, the dot marks the final position of the center vertex.

The reasons for keeping $\mathcal{D}_L$ small are the lower costs for the construction of a local parameterization as well as the lower distortion that is induced. In fact, we might not even be able to construct a reasonable parameterization in certain situations. Thus, $\mathcal{U}$ cannot be computed for the corresponding vertex and we keep its position. This rarely happens, and as the neighborhood of the vertex is relaxed we are likely to find a parameterization in the next iteration.

If the cone of normals of the triangles in $\mathcal{D}_L$ has only a small opening angle we can even use a simple projection to a fitting plane for constructing the parameterization. This is especially useful in the case when $\mathcal{D}$ is relatively coarse compared to $\mathcal{M}$. (Just consider the trivial case when a 1-ring of $\mathcal{M}$ is completely contained in a single domain triangle of $\mathcal{D}$.) In general we have to apply a more sophisticated parameterization scheme. There are a number of methods for flattening disk-like 2-manifold meshes to the plane ([7, 5, 11, 23]. We use Floater's [7] shape preserving parameterization after projecting the boundary of $\mathcal{D}_L$ to a plane. Due to the definition of $\mathcal{U}$ the boundary is usually "fairly convex" and we almost always get a valid, bijective parameterization without foldovers of triangles. In case of an invalid parameterization we either could map the boundary to a circle and apply Floater's method again, thus ensuring a valid mapping or give up for this iteration as mentioned above. Our experiments show that the less expensive second alternative works well. We use the first alternative only after the construction of a local parameterization for this vertex has failed several times.

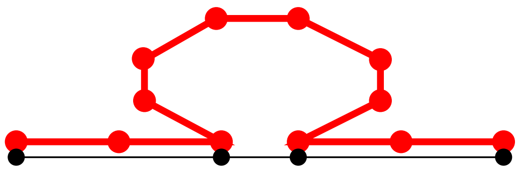

Figure 3: The remesh $\mathcal{M}$ (straight lines) might miss some feature part of the domain mesh $\mathcal{D}$ (curved) if we set $\omega_{ij} = 1$. We have to incorporate the area of the triangles of $\mathcal{D}$ that are covered by a 1-ring of a vertex in $\mathcal{M}$ into the relaxation operator to solve this problem and get a uniform sampling of $\mathcal{D}$.

Consider a configuration as illustrated in Fig. 3. Despite the fact that a projection parameterization is in general not a good choice for highly curved surface regions (cf. Fig. 1), there is the problem that the remesh $\mathcal{M}$ misses some feature part of the domain $\mathcal{D}$. This situation is likely to happen if $\mathcal{M}$ is coarse (cf. connectivity optimization described in the next section) compared to the surface feature on $\mathcal{D}$. In order to avoid this situation the relaxation operator $\mathcal{U}$ will take into account not only a 1-ring of $\mathcal{M}$ but also the covered domain triangles, i.e. the convex weights ω_{ij} used for relaxation depend on the area of domain triangles.This explains why the local domain $\mathcal{D}_L$ must cover a 1-ring in $\mathcal{M}$.

Formally, we can assign a local domain to every 1-ring. Practically, we assign to every mesh triangle $\Delta \in \mathcal{M}$ the set

$$C(\Delta) := \{\tilde{\Delta} \in \mathcal{D} | \Phi(\Delta) \cap \Phi(\tilde{\Delta}) \neq \emptyset\}$$

of domain triangles that have a non-empty intersection with Δ in the parameter domain. The application of $\mathcal{U}$ requires the union of all $C(\Delta_j)$ with triangles Δ_j from the respective 1-ring.

In every single relaxation step only a single vertex v is repositioned. So the corresponding $C(\Delta_j)$ have to be updated. We intersect each parameter triangle $\Phi(\Delta'_j)$ of the new 1-ring with all triangles of $\mathcal{D}_L^v$ and reassign them to the new triangles Δ'_j. This update step can be implemented efficiently since the intersection tests are done for all triangles of the 1-ring which are sharing common edges. Thus, intermediate intersection results can be reused. Moreover the outer hull of the 1-ring remains unchanged, hence there is no need to test against it. Also, we only need to recompute a parameterization if domain triangles have been added to the set of triangles assigned to a 1-ring. This caching of local parameterizations considerably speeds up our algorithm.

The choice of the weights ω_{ij} determines the energy to be minimized in the optimization process. Uniform weights will provide a uniform distribution of triangles in the local parameter domains. As the local domains are kept small, the resulting parameterizations are near isometric resulting in a uniform point distribution in 3-space. However, Fig. 3 illustrates, that surface features of $\mathcal{D}$ might be missed in the remesh. Hence we choose the weights ω_{ij} for the weighted umbrella proportional to the area (in 3-space) of all those domain-triangles that are covered by the 1-ring of v_{ij} and thus are able to capture those features.

Of course, this is just one possible choice for the weights ω_{ij} and there are many ways to adapt the ω_{ij}'s to a specific application. For instance one might also think of adapting the weights with respect to principal directions of curvature which would lead to an anisotropic remeshing. However, within the scope of this paper we just want to illustrate the core concept of our framework and will not go into detail here.

3. DYNAMIC CONNECTIVITY MESHES

Up to now, our remeshing scheme is able to reposition the vertices of $\mathcal{M}$ on $\mathcal{D}$. The other important component is to adapt the resolution of $\mathcal{M}$ while optimizing the connectivity. Starting from the initial connectivity of $\mathcal{M}$, we apply simple topological operators that insert or remove vertices and regularize the connectivity. This way we can incrementally adjust the complexity of $\mathcal{M}$ to a desired target resolution. In order to achieve this, we apply an algorithm similar to *DCM* [13], since we want to obtain a good mesh quality 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.

First, we apply a *half-edge collapse* — which removes two triangles (or one triangle for boundary edges) by collapsing one edge into a single vertex — to all edges that fall below a length of $\varepsilon_{\min}$.

In the second step we insert a vertex on every edge that is longer than $\varepsilon_{\max}$ and connect it with the opposite vertex in the adjacent triangle(s). The new vertex will be positioned on one of the endpoints of that edge *e* which was split (its parent-vertex). In fact, we are creating two geometrically degenerated triangles in the first place. But inserting the new vertex on the midpoint of *e* would require an expensive point-in-triangle-search (cf. [4]) in the parameter plane, since we have to establish the link to $\mathcal{D}$ for the new vertex.

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. Consequently, for every two neighboring triangles $\triangle(A,B,C)$ and $\triangle(A,B,D)$ we flip the common edge between A and B, if that reduces the total valence excess:

$$\sum_{p\in\{A,B,C,D\}} (valence(p)-6)^2$$

Note that all three topological operations are inexpensive and local since they do not need any global optimization and since they only affect a small region on $\mathcal{M}$. Moreover, $\mathcal{M}$ always remains a valid 2-manifold (opposed to the 1–to–4 triangle-split which is used in many subdivision schemes).

Now the desired resolution of $\mathcal{M}$ can be set by adapting $\varepsilon_{\min}$ and $\varepsilon_{\max}$. Notice that $\varepsilon_{\max} > 2\varepsilon_{\min}$ has to be satisfied in order to avoid generating two (with respect to $\varepsilon_{\min}$) invalid edges during the split operation.

In practice, both algorithms, i.e. relaxation and *DCM* work independently of each other. The relaxation operator $\mathcal{U}$ is iteratively applied to all the vertices. Here, the algorithm simply steps through the list of vertices in a linear order. As soon as an edge exceeds $\varepsilon_{\max}$ or falls below $\varepsilon_{\min}$, the corresponding topological operation is performed. Additionally, we schedule those vertices for immediate relaxation that were affected by the topological operation (the remaining vertex during the half-edge collapse and the newly inserted vertex introduced by the edge–split) thus ensuring a faster convergence of the particle system. Moreover, we particularly test the affected regions for possible edge-flips, since both half–edge–collapse and edge–split presumably amplify the local valence-excess.

Obviously, changes in the connectivity enforce an update of the local domains that are used in the relaxation step. This is trivial for the edge-split, since we introduce degenerated triangles which

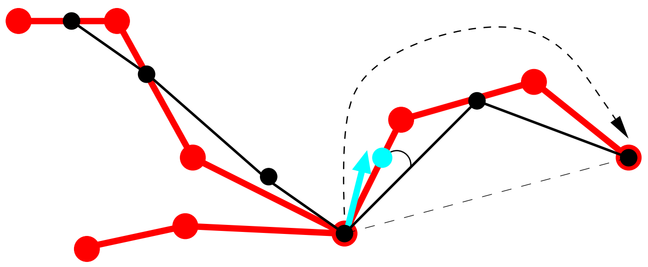

Figure 5: After a split of a bone-edge of $\mathcal{M}$ the newly inserted bone-vertex (light) gets attached to a bone-edge of $\mathcal{D}$ (thick lines). If the new vertex has a corner-vertex as its parent (central vertex), we attach it to that bone-edge that has the smallest enclosing angle with the bone-edge that was split. Additionally we require, that the opposite vertex of $\mathcal{M}$ can be reached via $\mathcal{D}$'s skeleton (dotted arc). After that, the new vertex is allowed to move on bone-edges of $\mathcal{D}$ exclusively.

do not have an intersection with domain triangles. During the half-edge collapse, we assign those domain triangles that were associated to the removed triangles to their respective neighbors. For the edge-flip, the reassignment is trivial.

4. FEATURE PRESERVATION

Aliasing is a fundamental problem for a remeshing algorithm which typically occurs at corners or sharp creases of $\mathcal{D}$. It is often mandatory to preserve such surface features but it can also be desirable to preserve additional user defined features such as certain vertices or edges.

Of course, refining $\mathcal{M}$ near the feature ad infinitum is not an appropriate solution. Instead we have to make sure that we represent those features with a limited number of faces of $\mathcal{M}$. In order to do so, it seems natural to align the edges along the features, i.e. orthogonal to the direction of maximal curvature. *FSR* [25] proposes an effective feature-snapping algorithm that is able to recapture features and is particularly useful if $\mathcal{M}$ and $\mathcal{D}$ differ.

Since we start with $\mathcal{M} = \mathcal{D}$, similar to [12, 3], we take a simpler approach and make use of skeletons that are attached to both meshes. In principle, a skeleton is a set of edges of $\mathcal{D}$ and can be either selected by an automatic algorithm (e.g. [22, 20]) or by an interactive selection done by the user. Additionally the users can add vertices to the skeleton that they want to be preserved during the remeshing.

In order to preserve this basic set of vertices and edges, we enhance the simple skeleton in a way that it consists of the following three primitives:

bone-edges are edges that were selected by the user/feature detection algorithm.

bone-vertices are vertices that have exactly 2 adjacent bone-edges.

corner-vertices are vertices that have $\neq 2$ adjacent bone-edges or vertices that are explicitly selected by the user.

The skeleton which is attached to $\mathcal{D}$ remains fixed during the whole remeshing process while its counterpart on $\mathcal{M}$ is modified. In order to ensure that the skeleton on $\mathcal{M}$ preserves the structure of $\mathcal{D}$'s skeleton, the key idea is to restrict the relaxation operator $\mathcal{U}$ for the bone-vertices of $\mathcal{M}$ to the bone-edges of $\mathcal{D}$. At the same time we ensure that the three topological operations during the retriangulation process (see Sec. 3) do not destroy this structure.

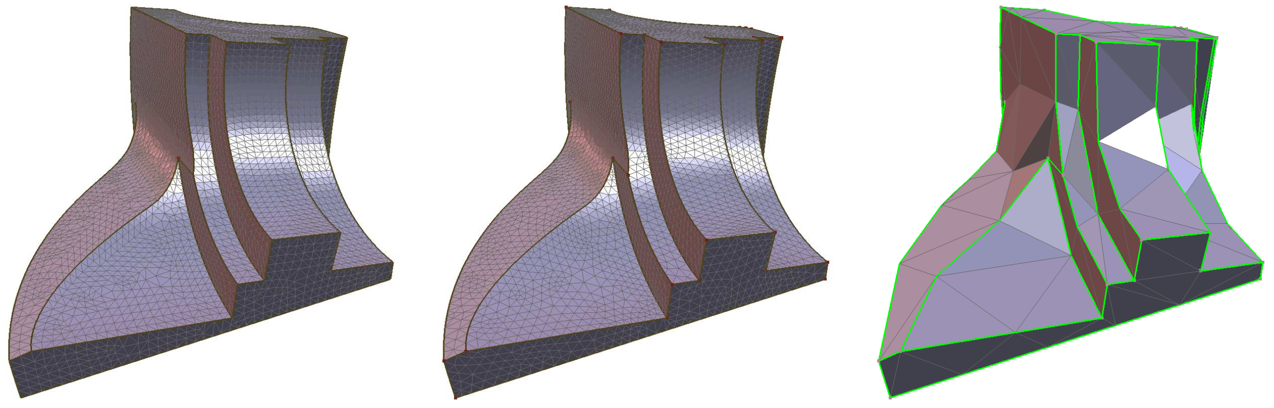

Figure 4: The original fandisk dataset with its skeleton and corner-vertices (left) gets remeshed (middle). Due to the restrictions imposed on the relaxation and topological operators, the skeleton is preserved even though we generated a really coarse approximation (right) of the original mesh.

Hence, we apply the following restrictions to the relaxation operator $\mathcal{U}$ and to the topological operations on $\mathcal{M}$.

- *Corner-vertices* remain fixed and never get touched by any topological operation.
- *Bone-vertices* are moving on *bone-edges* of $\mathcal{D}$ exclusively – $\mathcal{U}(p)$ is simply projected back to that *bone-edge*, that has the smallest enclosing angle with $\mathcal{U}(p)$.
- A half-edge-collapse of a *bone-edge* is allowed only if both endpoints belong to the skeleton (vertices that do not belong to the skeleton are allowed to collapse into the skeleton, but become part of it in that case (cf. Fig. 5)
- *Bone-edges* never get flipped.
- If a *bone-edge* *e* gets split, the new vertex is a *bone-vertex* as well. If the parent-vertex (see Sec. 3) happens to be a *corner-vertex*, it is allowed to move in the direction of that *bone-edge* of $\mathcal{D}$ that has the smallest enclosing angle with *e* and where the other endpoint of *e* is reachable (cf. Fig. 5).

5. APPLICATIONS

5.1 Multiresolution Modeling

The motivation for creating our remeshing environment is its application in our interactive multiresolution modeling framework *HUMID* (cf. [14]). In this paper, we just sketch the basic ideas, the detailed description of the whole modeling procedure will soon be available as separate technical report.

In our framework, the user selects the modeling-area $\mathcal{M}_{S0}$, a bounded region on the mesh which separates that part of the mesh which is subject to the modeling operation from the one that remains fixed. Inside the submesh $\mathcal{M}_{S0}$, the user transforms dedicated vertices and the algorithm adapts the remaining vertex-positions in such a way that $\mathcal{M}_{S0}$ transforms smoothly into $\mathcal{M}_{Si}$ while surface details are preserved (cf. [17]).

The original algorithm does not change the connectivity of $\mathcal{M}_{S0}$ (but just adapts the vertex positions). This might lead to badly shaped triangles in regions where extreme stretching/compression occurs. We have incorporated our remeshing technique into *HUMID* and are now able to do interactive multiresolution modeling with changing connectivity.

In order to do the dynamic remeshing, we first try to create a global parameterization of $\mathcal{M}_{S0}$. This is done by projecting the boundary of $\mathcal{M}_{S0}$ to a 2D polygon. The 2D position of the vertices in the interior of $\mathcal{M}_{S0}$ are obtained by the same relaxation procedure as it is described in Sec. 2 until the system converges (the boundary vertices remain fixed). We are choosing this parameterization to ensure that the tessellation does not change as long as the user does not apply any modification to $\mathcal{M}_{S0}$. Typically this step takes only a fraction of a second for a moderately large region of 5-10K$\triangle$.

If the parameterization algorithm fails, i.e. if foldovers occur (which rarely happens in our experiments), we fall back to successive local parameterizations (cf. Sec. 2) at the price of a lower frame-rate. Of course, computing a global parameterization only once in a preprocessing step is preferred compared to successively recomputing local parameterizations in every relaxation step. However, our local method enables us to process even those meshes that cannot be mapped to 2D with the method from above.

The actual modeling is done in two separate steps. We first perform a modeling operation (cf. [14]) and feed the modified surface $\mathcal{M}_{Si}$ to the remeshing algorithm. Instead of using the original modeling region $\mathcal{M}_{S0}$, we use $\mathcal{M}_{Si}$ as domain mesh $\mathcal{D}$. Since $\mathcal{M}_0$ and $\mathcal{D}$ are linked, we are always able to sample detail information from the original surface and thus ensure that the surface details are reconstructed correctly.

Note that the dynamic remeshing also ensures, that the triangulation of the whole mesh remains intact. No stitching [24] to reconnect the region which remained fixed with the remeshed modeling region $\mathcal{M}_{S0}$ is necessary.

5.2 Interactive Optimization

While the remeshing per se is fully automatic once the user has specified input parameters as e.g. the target resolution and/or a feature skeleton, we can also use it to provide a framework for interactive mesh optimization. Therefore, the optimization process is visualized by updating the displayed remesh after each iteration. The user can change input parameters during the optimization process, and he gets immediate visual feedback as the mesh converges to an optimal remesh.

Even more important in this context is the ability to specify only certain regions on the mesh that should be remeshed while the rest remains fixed. This can be regions on the original surface that include only few sample points for example. The optimization process runs just as before but schedules only vertices and edges in the specified regions for optimization. Again, this automatically ensures that the remeshed regions stay connected to the fix part of the mesh as its resolution and connectivity are optimized, and no additional zippering or stitching [24] is necessary.

In practice the user defined regions are often small enough to allow the parameterization of a whole region over the plane. As no local parameterizations have to be updated, this will significantly speedup the algorithm.

In addition, the user can influence the behavior of the relaxation operator $\mathcal{U}$ by adapting its weights ω_{ij}. This is done by defining a scalar-field for the domain vertices that is linearly interpolated on $\mathcal{D}$. The field can be manipulated interactively, e.g. to increase the local vertex density on $\mathcal{M}$. The new weights are computed by sampling and integrating the scalar values scaled by the inverse areas of the domain triangles. Of course, the user can always modify the feature skeleton during the remeshing process. Fig. 6 shows an example of an interactive remeshing session.

5.3 $\sqrt{3}$-Remeshing

Arbitrary (irregular) meshes are the most general and flexible boundary representation using triangles, but there is also the important class of *semi–regular* – or *subdivision–connectivity* – meshes often stemming from subdivision-algorithms (cf. e.g. [26]) that offer many advantages over the irregular setting. On the one hand this is due to their regular structure and on the other hand one can exploit their mathematical proximity to polynomial surfaces. Many algorithms, in particular in the context of rendering, filtering, texturing and compression (cf. [1]), can benefit from this special structure.

In the past, a number of algorithms [6, 19, 9, 15] have been proposed to convert an arbitrary input mesh into one having subdivision–connectivity. We can apply our dynamic remeshing framework for this conversion in the following way:

Instead of dyadic refinement (repeated 1–to–4–split operation), which implies dealing with 'intermediate' meshes containing red–splits, we follow the idea of Kobbelt [16] and perform $\sqrt{3}$-adic splits. This way we can reuse the basic topological operators and ideas from Sec.3.

The conversion is done in two phases. Similarly to the *MAPS*-algorithm (cf. [19]), we generate a coarse version of $\mathcal{M} = \mathcal{D}$ (cf. Sec. 2 and 3) by incrementally decreasing $\varepsilon_{\min}$and $\varepsilon_{\max}$. Theoretically, for genus zero objects, we can perform dynamic remeshing down to a tetrahedron as long as we find a valid parameterization for every 1–ring in $\mathcal{M}$. In practice, for objects with features, we are restricted by the rules imposed by the skeleton (see Sec. 4).

Once we have generated this coarse base-domain, we can start with the refinement phase. Again, this is done similarly to the above remeshing algorithm in that we apply topological changes combined with permanent relaxation. But instead of using the *DCM*-approach, we apply the topological operations as they are described in [16]. (The 1–to–3 split can be implemented analogue to the edge–split operation, i.e. we perform a topological 1–to–3 split of a triangle, place the newly inserted vertex on one of the corner vertices of the triangle which was just split and let the relaxation operator handle the repositioning.) Note that due to the hierarchical approach, the particle system converges quickly and we need just a few relaxation steps until the length of the relaxation vector falls below some user defined threshold.

We have to make one restriction in order to conserve the skeleton of $\mathcal{M}$. As mentioned in Sec. 4, a skeleton-edge is not allowed to flip in the first place. However, applying two refinement steps at once leads to a complete triadic split of all triangles (and edges). Thus we have a 1–to–1 correspondence of edges from the coarser level to those on the finer level and consequently are able to preserve the skeleton.

6. CONCLUSION

We presented a new algorithm for optimizing triangular meshes. This incremental remeshing process includes the optimization of the mesh connectivity based on the DCM approach as well as the optimization of the vertex distribution. The vertices are scattered regularly on the surface by simulating a particle system. This relaxation requires only local parameterization of the original surface, a global parameterization is explicitly avoided. In particular, we show how the local domains that are flattened to the plane can be efficiently constructed and updated after vertex shifts and connectivity changes.

The algorithm does not require to start with an initial mesh $\mathcal{M} = \mathcal{D}$ that is mapped 1-to-1 to the domain mesh, but we can also start with an *arbitrary* mesh. In that case, we first construct a valid link between $\mathcal{M}$ and $\mathcal{D}$ in a preprocessing step, and we require valid local domains, e.g. a mapping from triangles of $\mathcal{M}$ to sets of triangles of $\mathcal{D}$. This can be done by an appropriate projection operator, e.g. as used in [15]. However, for the sake of clarity we restricted ourselves to $\mathcal{M} = \mathcal{D}$.

We have applied our framework to various application scenarios. Results are shown on the next pages, a video with our dynamic remeshing framework in action is available on our web-site (www.mpi-sb.mpg.de/$\sim$ vorsatz). Even for the tooth model at highest resolution (77KΔ), the response times never exceeded 4-5 seconds.

7. REFERENCES

[1] Pierre Alliez and Mathieu Desbrun. Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH 2001 Conference Proceedings*, pages 198–205, 2001.

[2] Pierre Alliez, Mark Meyer, and Mathieu Desbrun. Interactive geometry remeshing. In *SIGGRAPH 2002 Conference Proceedings*, pages 347–354.

[3] Pierre Alliez, Éric Colin Verdière, Oliver Devillers, and Martin Isenburg. Isotropic surface remeshing. 2003.

[4] P. J. C. Brown and C. T. Faigle. A robust efficient algorithm for point location in triangulations. Technical report, Cambridge University, 1996.

[5] Mathieu Desbrun, Mark Meyer, and Pierre Alliez. Intrinsic parameterizations of surface meshes. In *Computer Graphics Forum, (Eurographics 2002)*, pages 209–218.

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

[7] Michael S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997. ISSN 0167-8396.

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

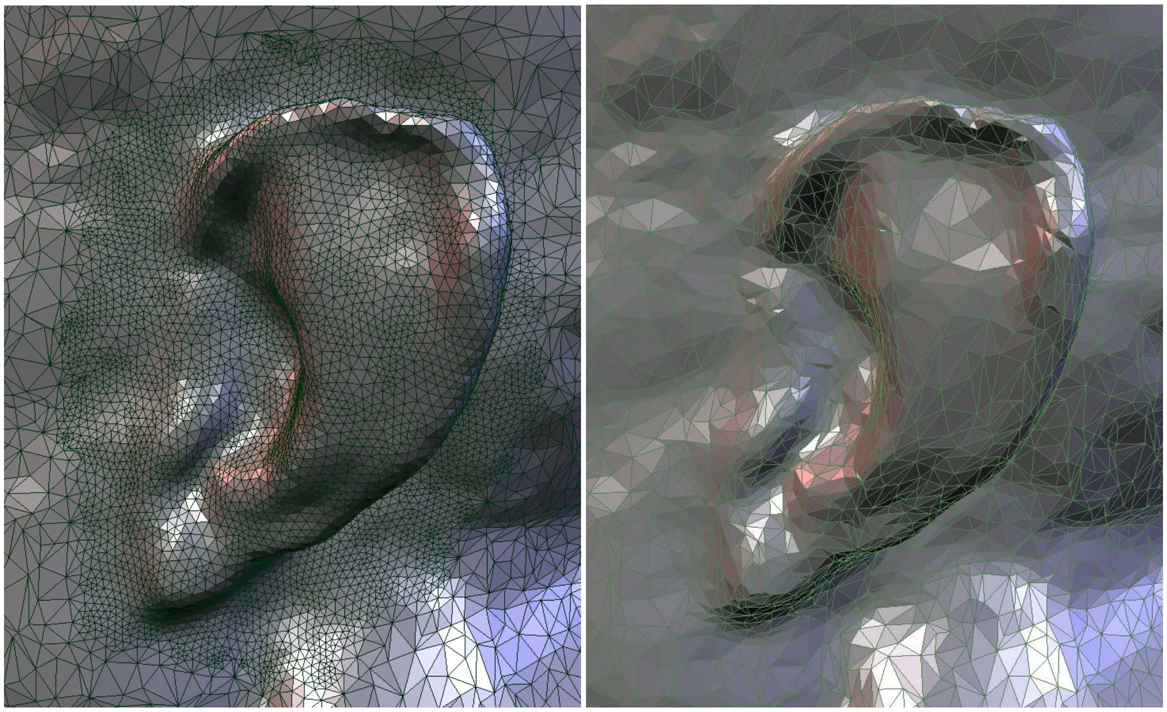

Figure 6: An interactive remeshing session operating on the ear of the Max-Planck model. The original triangulation on the left gets refined. Note that the partially remeshed area automatically connects to the fixed vertices on the boundary.

[9] Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. Normal meshes. In *SIGGRAPH 2000 Conference Proceedings*, pages 95–102, 2000.

[10] H. Hoppe. Progressive meshes. In *SIGGRAPH 1996 Conference Proceedings*, pages 99–108.

[11] K. Hormann and G. Greiner. MIPS: An efficient global parametrization method. In *Curve and Surface Design: Saint-Malo 1999*, pages 153–162. 2000.

[12] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *SIGGRAPH 2002 Conference Proceedings*, pages 339–346.

[13] L. Kobbelt, T. Bareuther, and H.-P. Seidel. Multiresolution shape deformations for meshes with dynamic vertex connectivity. In *Computer Graphics Forum (Eurographics 2000)*, volume 19, pages 249–260.

[14] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi–resolution modeling on arbitrary meshes. In *SIGGRAPH 98 Conference Proceedings*, pages 105–114.

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

[16] Leif Kobbelt. sqrt(3) subdivision. In *SIGGRAPH 2000 Conference Proceedings*, pages 103–112.

[17] Leif Kobbelt and Jens Vorsatz. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry*, 14(1-3):5–24, 1999.

[18] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature-sensitive surface extraction from volume data. In *SIGGRAPH 2001 Conference Proceedings*, pages 57–66.

[19] 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.

[20] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH 2002 Conference Proceedings*, pages 362–371.

[21] P. Lindstrom. Out–of–core simplification of large polygonal models. In *SIGGRAPH 2000 Conference Proceedings*.

[22] Christian Rössl, Leif Kobbelt, and Hans-Peter Seidel. Recovering structural information from triangulated surfaces. In *Mathematical Methods for Curves and Surfaces: Oslo 2000*, pages 423–432, 2001.

[23] Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. Bounded-distortion piecewise mesh parameterization. In *IEEE Visualization 2002 Conference Proceedings*, pages 355–362.

[24] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *SIGGRAPH 94 Conference Proceedings)*, pages 311–318.

[25] Jens Vorsatz, Christian Rössl, Leif Kobbelt, and Hans-Peter Seidel. Feature sensitive remeshing. In *Computer Graphics Forum (Eurographics 2001)*, pages 393–401.

[26] D. Zorin and P. Schröder. Subdivision for modeling and animation. In *SIGGRAPH 2000 Course Notes*.

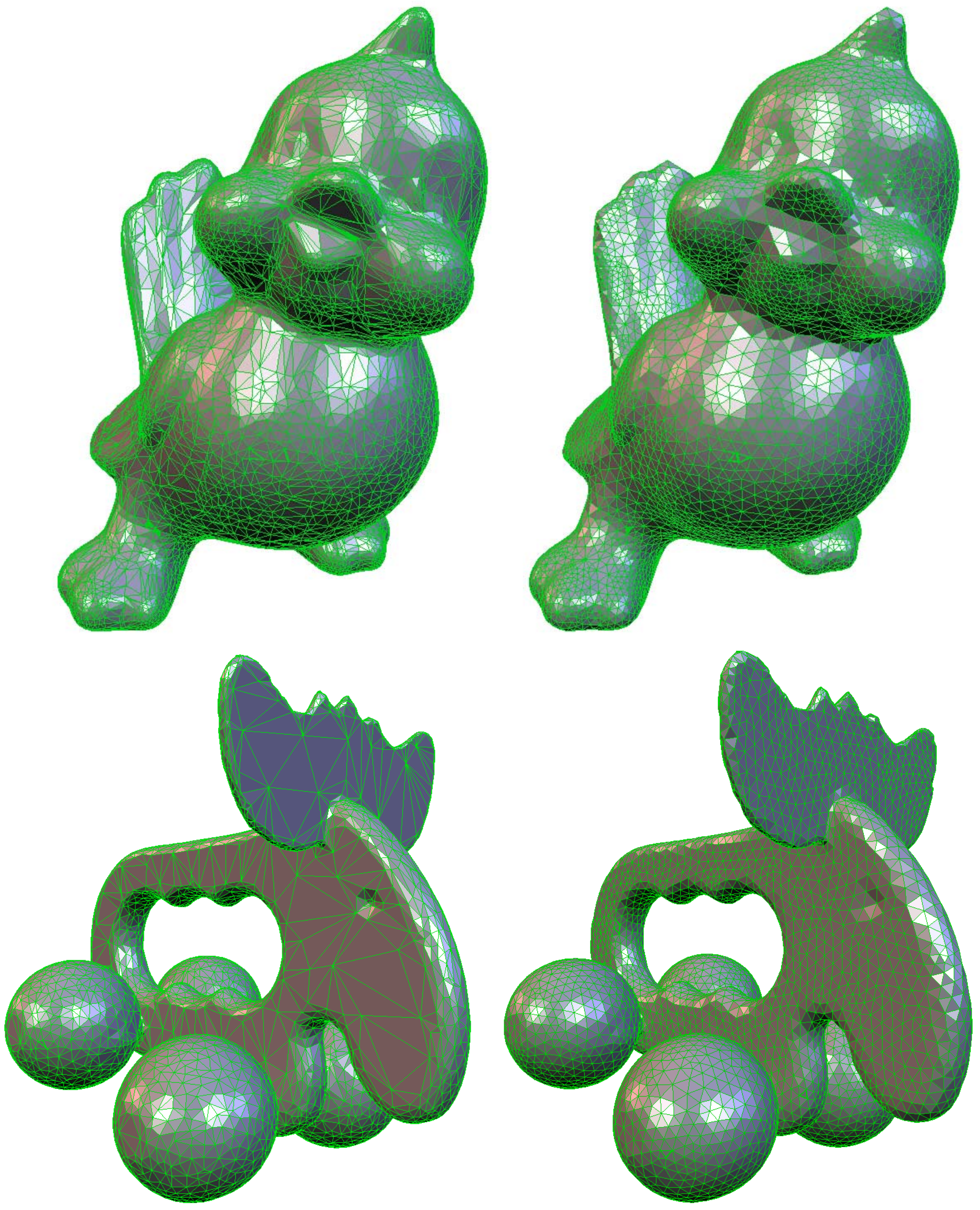

Figure 7: Remeshing of geometrically and topologically more complex models (original data-sets on the left, remeshed version on the right). Our algorithm works without an explicit patch-layouting for a global parameterization. A closeup view of Tweety's tail can be found in Fig. 1.

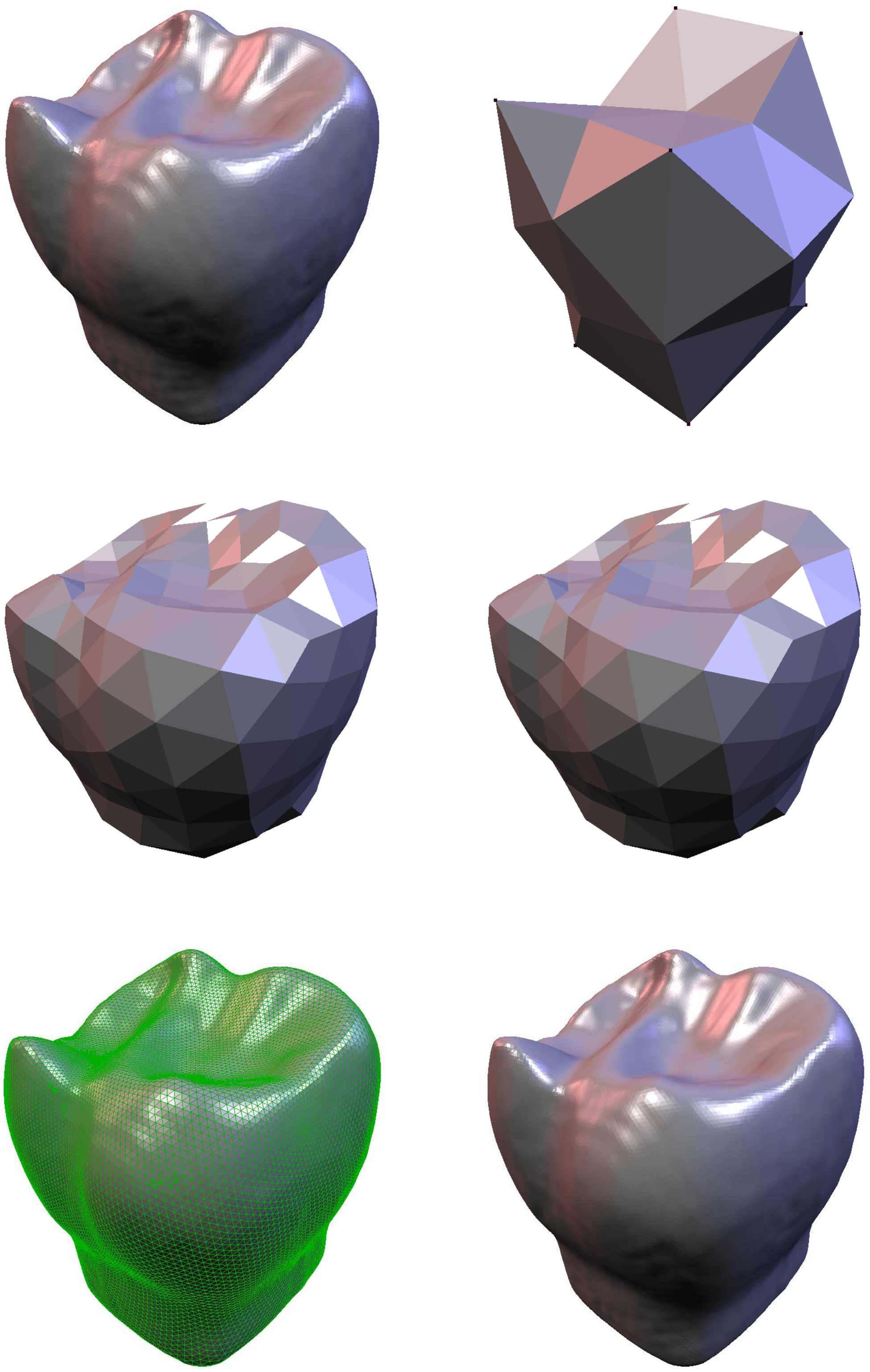

Figure 8: $\sqrt{3}$-remeshing of a tooth-model (original upper left). We first generate the coarse base domain (upper right) and start refining by applying the topological $\sqrt{3}$ split operator twice per step (middle row). The lower row shows the final remesh.