Interactive Geometry Remeshing

Pierre Alliez
USC / INRIA

Mark Meyer
Caltech

Mathieu Desbrun
USC

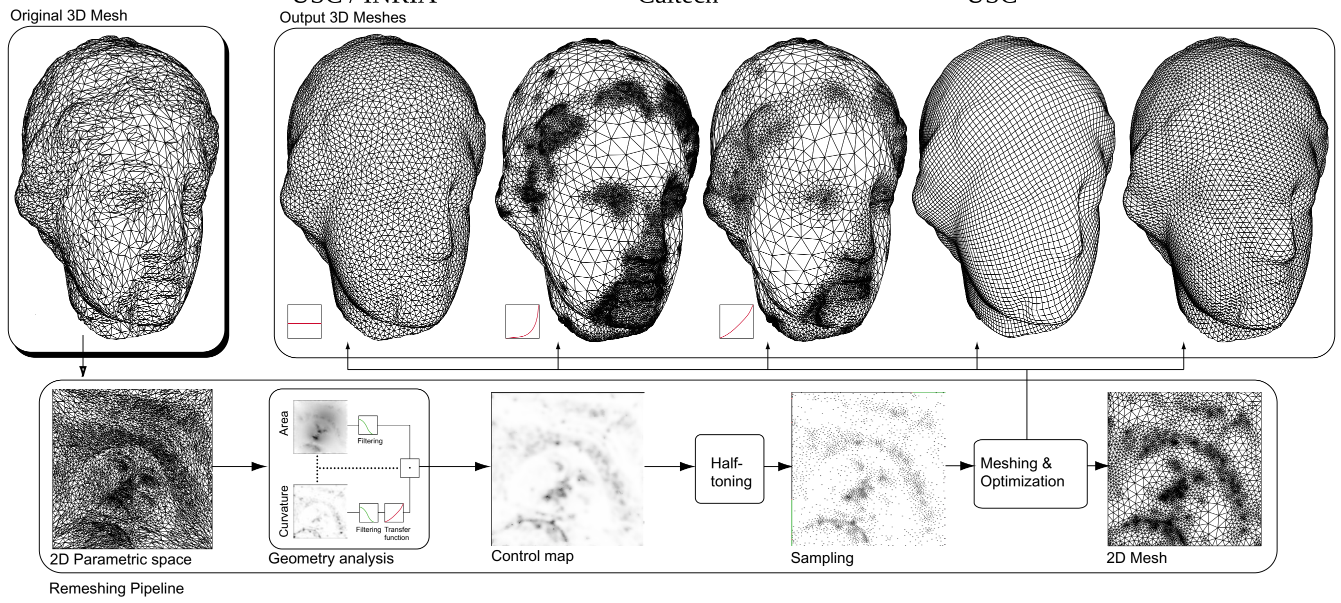


Figure 1: *A brief overview of our remeshing process: The input surface patch (top left) is first parameterized; Then geometric quantities are computed over the parameterization and stored in several 2D maps; These maps are combined to produce a control map, indicating the desired sampling distribution; The control map is then sampled using a halftoning technique, and the samples are triangulated, optimized and finally output as a new 3D mesh. A few examples of the various types of meshes our system can produce are shown (top, from left to right): uniform, increased sampling on higher curvature, the next with a smoother gradation, regular quads, and semi-regular triangles. After an initial pre-processing stage (∼1s), each of these meshes was produced in less than 2 seconds on a low-end PC.*

Abstract

We present a novel technique, both flexible and efficient, for interactive remeshing of irregular geometry. First, the original (arbitrary genus) mesh is substituted by a series of 2D maps in parameter space. Using these maps, our algorithm is then able to take advantage of established signal processing and halftoning tools that offer real-time interaction and intricate control. The user can easily combine these maps to create a control map – a map which controls the sampling density over the surface patch. This map is then sampled at interactive rates allowing the user to easily design a tailored resampling. Once this sampling is complete, a Delaunay triangulation and fast optimization are performed to perfect the final mesh.

As a result, our remeshing technique is extremely versatile and general, being able to produce arbitrarily complex meshes with a variety of properties including: uniformity, regularity, semi-regularity, curvature sensitive resampling, and feature preservation. We provide a high level of control over the sampling distribution allowing the user to interactively custom design the mesh based on their requirements thereby increasing their productivity in creating a wide variety of meshes.

1 Introduction

As 3D geometry becomes a prevalent media, a proliferation of meshes are readily available, coming from a variety of sources including 3D scanners, modeling software, and output from computer vision algorithms. Although these meshes capture geometry accurately, their sampling quality is usually far from ideal for subsequent applications. For instance, these (sometimes highly) irregular meshes are not appropriate for computations using Finite Elements, or for rapid, textured display on low-end computers. Instead, meshes with nearly-equilateral triangles, a smooth gradation of sample density depending on curvatures, or even uniform sampling are preferable inputs to most existing geometry processing algorithms. *Remeshing*, i.e., modifying the sampling and connectivity of a geometry to generate a new mesh, is therefore a fundamental step for efficient mesh processing.

We propose a precise and flexible remeshing technique for arbitrary geometry. Unlike previous techniques, we offer a high level of control over the sampling quality of the output mesh, as well as an unprecedented speed of execution. We will show that our remeshing engine can accurately generate any "tailored" sampling at interactive rates, and, if necessary, quickly optimize the quality of the resulting mesh, allowing the user to easily design a resampled geometry conforming to her requirements.

1.1 Background

Although studied in Computer Graphics for obvious reasons, surface remeshing has also received a lot of attention from various non-CG fields interested in mesh generation — mainly Computational Fluid Dynamics, Finite Element Methods, and Computational Geometry. However, the diverging goals resulted in vastly different, non-overlapping solutions as we now briefly review.

Mesh Generation Community Since the emphasis is generally on numerical accuracy, most of the tools developed in the non-CG communities focus on mesh quality. Remeshing procedures often use a parameter space to impose quantitative mesh properties such as local triangle sizes and shapes [8, 37, 16]. Others simply perform mesh simplification [30] or edge operations and vertex shifting [2] to conform to a global mesh property. However, most techniques heavily rely on mesh optimization [13, 33] to satisfy common requirements like equal angles for FE computations [3] or smooth gradation [4]; *accuracy* is therefore obtained at the price of rather slow computations.

Computer Graphics Community In contrast to the quality requirements of the other fields, CG work has focused mainly on *efficiency*. The majority of previous work has proposed semi-regular remeshing techniques [24, 20, 21, 22], based on an initial phase of simplification which could be used in itself for remeshing [14, 27] since it performs the aforementioned edge operations and vertex shifting. A noticeable body of work has also been recently proposed to accurately remesh sharp features [41, 6]. However, none of these methods can offer flexibility on the quality of the remeshing obtained, since issues such as area distortion or triangle shape distortion are not even considered: tailored output can only be produced through extensive trial-and-error by a patient user.

A controllable mesh re-tiling technique was proposed by Turk [38] to resample an input mesh using properties such as uniformity or curvature-based density, allowing a much more precise design of the output meshes. However, the algorithm requires the propagation of "particles" on the original mesh and a global relaxation of their positions until convergence, requiring heavy computation. Similarly, Bossen and Heckbert [5] proposed a 2D anisotropic mesh generation involving vertex insertions, vertex removals, and iterative relaxation. Again, output meshes conforming to various requirements can be generated but only after significant computational effort. Our goal is thus to attain accuracy, flexibility, and efficiency for resampling, as none of the techniques described above can offer such a combination.

1.2 Contributions & Overview

Our main contributions over previous remeshing techniques are in terms of **efficiency** as simple meshes can now be processed in real or interactive time through a novel resampling stage followed by an *output-sensitive* remeshing algorithm, and **flexibility** as we offer complete and precise control over the sampling rate and quality anywhere on the geometry. These two critical properties are obtained through the use of parameterization and conventional image processing tools such as filtering, transfer functions and error diffusion, in order to compute near-optimal resamplings in a matter of milliseconds. Previous approaches often worked directly on the mesh, resulting in either slow performance or little control over the remeshing quality.

The structure of this paper follows closely the overall algorithmic pipeline depicted at the bottom of Figure 1. We first describe the atlas of parameterization and geometry analysis we perform on the input mesh in Section 2, in order to generate a catalog of 2D maps as an alternate representation for the input mesh. We detail how these resulting maps are processed efficiently using standard signal processing tools to create a near-optimal resampling of the input mesh in Section 3. A final, rapid phase of optimization can then be performed to get accurate results as described in Section 4. Finally, we present a number of results to demonstrate the wide range of possible resamplings we can interactively obtain in Section 5, before concluding in Section 6.

2 Geometry Analysis

In this section, we explain in detail how we build a complete set of *maps* from the raw, input geometry. This will construct an alternative representation of the surface and all of its intrinsic properties in the form of convenient 2D images, which are easy to process. We demonstrate how simple and efficient this process is when graphics hardware is used appropriately. We show how to create a small set of tiling patches from a closed object of arbitrary genus, then give details on how to compute the geometry maps from these surface patches by flattening them onto isomorphic planar triangulations.

2.1 Creation of an Atlas of Parameterization

The first processing stage undergone by the input mesh consists in splitting the surface into disk-like patches, creating an *atlas of parameterization* [18]. A number of existing clustering algorithms such as [15, 31, 26] could be used successfully to achieve such a partition. Unfortunately, they do not produce smooth patch boundaries on the geometry as demonstrated in Figure 4, and therefore lead to poor-quality stitching across the remeshed patches. Note that one could also make some cuts in the geometry to turn it into a single patch, as often proposed in the last two years [23, 12, 34, 19]. All of these methods are valid ways to deal with arbitrary genus surfaces, and the resampling technique presented in this paper is mostly independent of the cutting/unfolding method chosen.

In the remainder of this paper, we use a variant of the mesh partitioning proposed by Eck *et al.* [11] (later improved by Guskov *et al.* [21]), that computes approximate Voronoï diagrams as an initial non-smooth partitioning of the mesh into genus-0 patches. This procedure, which we will extend in Section 2.5 to generate area-balanced patches, automatically produces a series of tiling patches from *input meshes of arbitrary genus*.

2.2 Parameterization

The second stage is to map each individual surface patch to an isomorphic planar triangulation. This operation, called *parameterization*, also has many solutions readily available ([11, 25, 26, 9] to name a few). Although most parameterization techniques would be adequate, one that guarantees visual smoothness of isoparametric lines and preserves the conformal structure of the input mesh is most preferable. We thus strongly advocate for the conformal parameterization as defined in [32, 11] since it behaves extremely well even on irregular triangulations [9]. This technique requires solving a simple, sparse linear system with coefficients based on the geometry of the mesh, and is usually handled in a matter of seconds using a Conjugate Gradient solver with good preconditioning. We fix the boundary to be a square (see Figure 2) or any convenient rectangular region so that our maps can be efficiently stored and processed as regular floating point images.

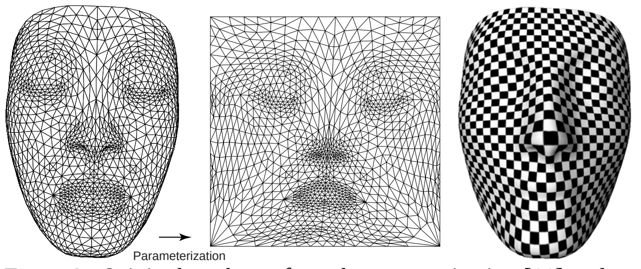


Figure 2: *Original mesh, conformal parameterization [11] and texture mapping of a checker-board. Notice the inevitable area distortion on the nose, which we will automatically compensate for during the resampling process (see Section 3.1).*

2.3 Geometry Maps

Once a parameterization has been found, we compute several scalar maps to serve as a *complete substitute* for the input geometry. This will allow us to work almost solely on the 2D images instead of on the original 3D mesh.

Catalog of Maps For our application, we have identified the following geometrical values as being relevant:

◇ *Area distortion map* $\mathcal{M}_{\mathcal{A}}$*:* since no discrete parameterization can (in general) preserve the area of every triangle, we need a piecewise constant scalar map indicating how each triangle has been shrunk or expanded during the parameterization. This is easily computed using the ratio $\mathcal{A}_{3D}/\mathcal{A}_{2D}$ of each triangle's surface area in 3D and its corresponding area in the 2D parameterization. Note that this map will *compensate* for any area distortion inevitably introduced by the parameterization (as depicted in Figure 2).

- ◇ *Curvature maps $\mathcal{M}_K$ and $\mathcal{M}_H$:* since any differential quantity on a smooth surface can be expressed as a (possibly nonlinear) combination of three invariants: area $\mathcal{A}$, Gaussian curvature K, and mean curvature H [17], we compute both a Gaussian curvature and a mean curvature map (in addition to the previously mentioned area distortion map). We use the discrete differential operators described in [28] to compute the mean and Gaussian curvatures at each vertex of the input mesh, though any reliable approximation of curvatures on piecewise-linear surfaces can be used. These two maps can then be combined to obtain other useful curvature maps: for instance, one can compute maps of minimum curvature κ_1, maximum curvature κ_2, or total curvature $\kappa_1^2 + \kappa_2^2$ by simple per-pixel operations on those two basic maps. Additional data, such as curvature tensors could also be computed on the surface and stored in maps, but we do not make use of them in this work;
- ◇ *Embedding Map $\mathcal{M}_{\mathbf{x}}$:* we also need the position $\mathbf{x} = (x, y, z)$ of each vertex, describing the exact geometry of the surface in 3D. These three maps (one per component) will provide a very efficient way of computing the mapping between a value $\mathbf{u} = (u_x, u_y)$ on the parameterization and its associated 3D point on the input mesh $\mathbf{x} = (x, y, z)$;
- ◇ *Face Index Map $\mathcal{M}_{index}$:* we also construct a face index map by assigning a color to each triangle in the parameterization corresponding to its face index in the mesh, as done by Botsch *et al.* [7]. Such a map turns out to be efficient for locating in constant time the triangle in which a given parametric value lies, saving potentially costly searches.
- ◇ *Additional Maps:* finally, any attribute (normal, texture, color, etc.) can also be mapped onto the parameterization to complete the catalog of maps.

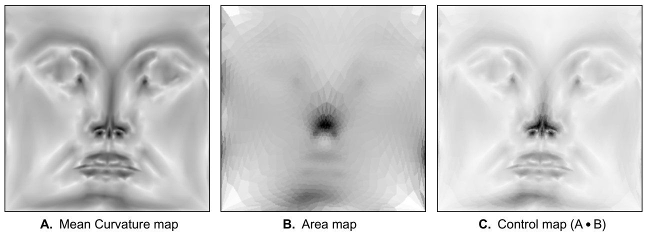


Figure 3: *Examples (in inverse mode for better visualization) of geometry maps for the mask in Figure 2. A. $\mathcal{M}_H$, the mean curvature map computed according to [28]. B. $\mathcal{M}_{\mathcal{A}}$, the area map; the nose has been compressed during the flattening process, while areas nearby the corners have been stretched. C. Sampling control map, using a per-pixel multiplication: $A \cdot B$.*

Hardware-Assisted Map Generation Piecewise-constant maps representing area distortions, face indices or per-face normals are efficiently generated using hardware accelerated OpenGL commands. Each floating-point or integer value is separated into the R, G, B, A color channels (similar to [7]), and all the triangles are rendered using OpenGL flat shaded triangle primitives in a back buffer. We assign a depth proportional to the surface area of each triangle to reduce the aliasing of small triangles in the map.

For linearly interpolated maps representing curvature, positions, per-vertex normals or attributes, we use the face index map and standard barycentric coordinates to compute the linear interpolation between the vertices in the parametric space. Note that the map creation could be simplified and optimized even further in the near future as soon as graphics boards implement full 32-bit floating point buffers for rendering (several OpenGL 2.0 proposals already require this feature). Nonetheless, generating the maps using current graphics hardware speeds up the map creation by *two orders of magnitude* compared to a naïve pixel-by-pixel implementation, and takes less than 100 ms for large meshes with thousands of triangles. Figure 3 depicts both a curvature and an area map, as well as a compositing of the two.

2.4 Features and Constraints

In addition to the geometry maps, we sometimes need to define specific features and/or constraints that the user wishes to enforce during the remeshing process. Typically, we want sharp features (present in mechanical parts for instance, see top left of Figure 6) to be preserved. Similarly, some particular points of the input surface may need to be constrained to become vertices of the remeshed version, for animation purposes for example.

Features We first assume that feature edges are either extracted using a simple dihedral angle thresholding, or directly input by the user by tagging existing input edges or creating arbitrary piecewise-linear feature curves. From this set of feature edges (Figure 6, top middle) we classify vertices by their number of adjacent feature edges, leading to two categories: we call *crease* vertices any vertex connected to exactly two feature edges, and *corner* vertices all the other vertices, connected to one or more than two feature edges. These feature edges are then chained together into a feature graph. This is very similar to the feature skeleton composed of "*backbones*" as introduced by Kobbelt *et al.*in a series of papers concerning geometry resampling and feature remeshing [7, 41, 6] (see Figure 6, top right, for an example). This **feature graph** requires little memory and can be computed in a straightforward way. We should note the following details that need to be addressed during the implementation: i) the graph can have cycles, ii) each patch boundary or cutting path is *also* added to the feature graph as a closed cycle (as being either a *sharp*, *boundary* or *seaming* backbone), iii) some features may meet at corners living on the boundary, and iv) a crease vertex should be classified as a corner if an important change of direction is detected along the feature. The latter corresponds to a feature inflexion point and is a rare occurrence. Once the feature graph has been properly constructed, the specified piecewise linear features will be exactly preserved by our remeshing technique as explained in Section 3.2.

Constraints We also allow the user to define a list of (u,v) values for which (s)he desires to get corresponding vertices in the output mesh. These values can be defined by the user by simply clicking on the input mesh. We save a list of all the constraints for later use during resampling.

2.5 Making the Atlas Area-Balanced

As mentioned in Section 2.1, we mostly use an existing technique to construct the atlas of parameterization. We, however, make use of our novel maps to improve this procedure. Eck [11] proposed to smooth patch boundaries iteratively by mapping two adjacent patches onto a 2×1 rectangular region using the discrete conformal mapping discussed in Section 2.2, and then re-defining the boundary between the two patches as the middle isoline in the parameterization (see Figure 4), which guarantees smoothness. However, this relaxation has a major inconvenience: it is *slippery* – since the

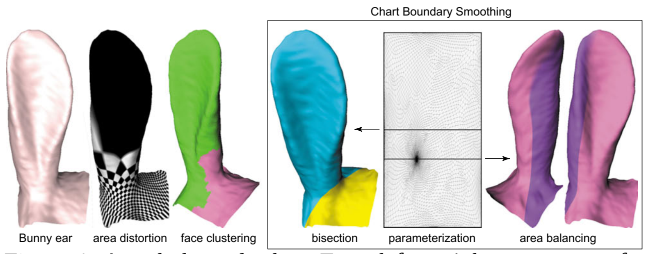


Figure 4: *Area-balanced atlas. From left to right: geometry of a Bunny ear; conformal parameterization and resulting area distortion visualized through a texture mapping of a checkerboard; face clustering obtained using [15]; partitioning obtained by simple bisection [11, 21]; the conformal parameterization, with the two medians; area-balanced and smooth partitioning, using the median line of its area map $\mathcal{M}_{\mathcal{A}}$ (computed in 50 ms).*

parameterization does not have any guarantee on area distortion, the middle isoline often splits the two patches into patches of two very different sizes, with a tendency to slip away from very curved features. As depicted on Figure 4, this often leads to patches with highly variable surface areas (compare the left and right areas after splitting) and with large parameterization distortion (note that one patch contains the entire ear, while the other is relatively flat).

Instead, we propose to construct the area distortion map of the 2×1 mapping as described in the previous section, and use it to find a good splitting line that creates equal sized patches. This is done by finding the median vertical line such that the sum of all pixel values on one side of the line is equal to the sum of the pixel values on the other side. Since a single sweep of the picture is sufficient to find the median, this operation takes little time – about $50ms$ for a 512×512 image. As demonstrated in Figure 4, this change in the original algorithm significantly enhances the quality of the partitioning, as no slipping occurs and each patch has the same surface area. Note that the dividing line is smooth thanks to the angle-preserving parameterization (*i.e.*, a straight line in parametric space corresponds to a smooth line on the surface).

Once the partitioning is done, we can compute the maps for each of the created patches as aforementioned. We use a lazy evaluation, computing a map only if needed to save both memory and time. We show in the next section the main contribution of this paper, *i.e.*, how these maps alone are used to resample the surface geometry at interactive rate.

3 Realtime Geometry Resampling

Now that the input geometry has been preprocessed and replaced by an equivalent series of maps, we can use these maps to design a proper resampling. In this section we propose a realtime technique to resample the geometry. This is achieved in two stages: first, the user designs a *control map* by combining different geometry maps to define the desired *density of samples*; then a simple *halftoning* technique is used to discretize this map and generate the exact, requested number of vertices. We show that this resampling is near optimal, and only a quick optimization will be needed to obtain a high quality mesh as output.

3.1 Designing the Control Map

To allow for a vast range of possible remeshings, we let the user design a control map that denotes the vertex density for the remeshing.

Area Map as Sampling Space Resampling the parameterization uniformly would not result in a regular 3D resampling of the geometry, due to the area distortion introduced during flattening. However, the area map $\mathcal{M}_{\mathcal{A}}$ does indicate the density of sampling needed on the parameterization to obtain a uniform sampling on the surface itself. The area map is therefore the sampling space we will use as *reference sampling density*.

Modulating the Sampling Density The final control map is obtained by multiplying the sampling space map by the **importance map** – a map denoting the desired sampling density across the patch. Many different maps can be used to tailor the sampling to the user's requirements, though we have mainly used curvature related maps in this work. To demonstrate the diversity of possible remeshing, we mention some canonical examples of importance maps that we have tried:

◇ constant, we will obtain a uniform vertex density on the 3D surface (see Figure 8),
◇ related to an estimation of curvature using $\mathcal{M}_K$ and $\mathcal{M}_H$, we will adapt the sampling rate to the local curvature (see Figure 11),
◇ any user-defined map, we will obtain a map with user specified sampling (useful for animation and displacement maps). See Figure 9 for such an example.

The resulting map is then rescaled to the unit interval, and inverted ($x \rightarrow 1 - x$) so that darker areas on the picture correspond to regions which require higher sampling. A simple example is depicted

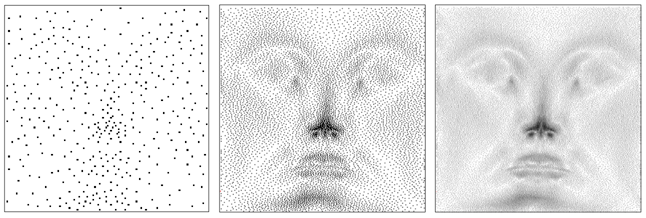


Figure 5: *Sampling of the map from Figure 3(C) using error diffusion with various numbers of requested samples (*$40\ ms$ *each).*

in Figure 3(C), where the area map is modulated with a mean curvature map (very light (white) areas correspond to flat and/or highly stretched regions of the mesh due to the flattening, and require few samples).

3.2 Halftoning the Control Map

Once the control map has been decided upon, we need to resample it with a local density of vertices in accordance with the control map, and with the exact number of samples the user requests. In other words, we need to transform the control map into a *binary image*, indicating the presence or absence of a vertex on the parameterization. In essence, our problem is directly related to the technique of *halftoning* grey-level images. Halftoning has been carefully studied for decades [39] and is still an active research field [29], mainly trying to improve the quality of dithering and printing. Different methods have been proposed to sample a continuous image with an adequate density, and to best statistically simulate an optimal blue noise signal in a single rasterization pass [39].

Discretizing the Control Map We use a recent error diffusion algorithm developed by Ostromoukhov [29], which samples an image using a *serpentine rasterization* (left to right on even lines, right to left on odd lines) with near-optimal quality. We add the following modifications to suit our purposes:

◇ while the original technique works on 8-bit images, we use 32-bit images to increase the range of densities;
◇ to avoid the well-known "dead zone" problem in error diffusion (large empty areas at the start of an error diffusion), we concatenate a vertically flipped copy of the control map above the control map and perform the halftoning for the total image, retaining only the bottom half of the image as the result;
◇ we also test for features and constraints (see Section 2.4), forcing a pixel to be black if it falls on one of the constraints, or forcing a pixel to be white if it falls on one of the features (as they will be sampled separately). The error diffusion accommodates for these forced selections by diffusing the error into nearby pixels.

The user simply chooses a given number of samples (which will be the final number of vertices) since an *exact number of vertices* can easily be reached by a simple linear scaling of the intensity of the control map [29] that preserves the ratio between the number of black pixels (*i.e.*, number of samples) and the image area. Note that the size of the maps determines the maximum number of samples (there cannot be more samples than there are pixels in the map). Therefore, we allow the user to select an appropriate image size having enough space for the sampler to work properly (though the choice of image size can easily be made automatically if desired). Such a technique turns out to be extremely efficient: a 512×512 image is sampled in only $40\ ms$ on a 1 GHz PIII. Examples of error diffusion are given in Figure 5.

Discretizing the feature graph A separate 1D error diffusion is performed along the boundaries and features in order to guarantee a consistent mesh density between the boundary and inner regions, as well as good feature preservation. After the initial sampling, we: i) gather all the pixels of the feature graph in a 1D array using Bresenham's line algorithm, ii) normalize their intensity according to the following law: $x \rightarrow 1 - \sqrt{(1-x)}$ (intuitively, the square root appears since if we want the fraction x of the samples to be black in 2D, it means we need the fraction $\sqrt{x}$ of the samples to be black in any 1D cross-section), $iii)$ apply a 1D error diffusion, and finally $iv)$ put the resulting samples into the sampled image. This guarantees an adequate feature sampling conforming to the control map, as demonstrated in Figure 8. The seams across patches are dealt with similarly to ensure an easy stitching.

3.3 User Control

Since our resampler runs at interactive rates, we can provide the user with a preview of the new mesh and allow for realtime editing of the control map to tailor the sampling to specific needs. An extremely powerful feature of our map based technique is that we can take advantage of many well-known signal processing tools for images. As a consequence, we can offer a multitude of tools still with realtime performance; for example:

◇ **Transfer Function** - Besides combinations obtained from filtering, scaling and shifting of the maps, we found it particularly useful to allow editing of a general transfer function over the importance map, or even direct editing of the importance map itself. For instance, a simple gamma function $f(x,\gamma) = x^\gamma$ over the curvature map gives the user control over the sampling with respect to the curvature. The user can also use pass-band filters or even a general transfer function to produce meshes with arbitrary sampling. Notice that the generality of this approach allows our system to simulate virtually any remeshing by choosing the maps and transfer functions appropriately (such as the $\mathcal{L}^2$-*optimal sampling* derived in [36]).

◇ **Smooth gradation** [3] of the vertex density can be achieved by *low pass filtering* of the importance map, using an optimized Gaussian filter routine. Changes over the global size of the filter kernel allow a fine and interactive tuning of the gradation. Note that in the ideal case, the local size of the filter kernel should be driven by the area map, making it a non-linear diffusion of the importance map.

◇ **Minimum Sampling** - A guaranteed minimum density of samples can be obtained by shifting the intensity of the importance map so that its minimum corresponds to the requested minimum sampling (*i.e.*, a minimum grey level).

Interactive Preview The error diffusion is fast enough ($40\ ms$ including the transfer function computation) to provide a real-time feedback of the sampling. Additionally, we provide the option of using the dithered map as a texture directly on the 3D original model since we already have the (u, v) parameterization. The samples thus appear on the mesh instantaneously, leading to a good preview of the current sampling.

4 Mesh Creation and Optimization

At this point, we are already able to *interactively* produce a resampling of an input mesh with a density proven to be statistically in agreement with the user's request. However, connectivity has not yet been computed. Additionally, the halftoning implies *quantized positions* for the vertices. Therefore, we now explain how to generate an initial connectivity and how a post-process optimization can greatly improve both connectivity and geometry in mere seconds. We emphasize that, contrary to [5] and most other remeshing techniques, we neither add, nor remove any vertex during the optimization since, in essence, the blue noise property already spreads "just enough" vertices everywhere. Consequently, the optimization is extremely efficient and consists of only a few edge swaps and local vertex displacements.

4.1 Mesh Creation

Once the control map has been sampled, we perform a 2D constrained Delaunay triangulation [1, 35] over the points sampled in the parametric space. Constrained edges correspond to an ordered list of points sampled using 1D error diffusion along backbones of the feature skeleton (see Section 2.4), as can be seen in Figure 6, bottom middle. The vertex coordinates are then mapped into 3D

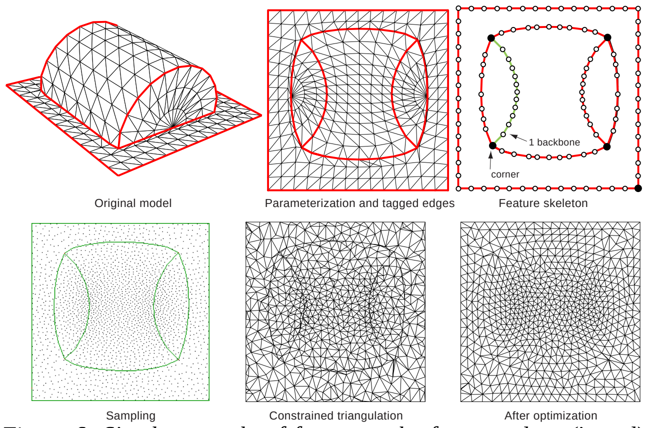


Figure 6: *Simple example of features: the feature edges (in red) are chained together to create the feature graph; a 1D error diffusion is then performed along the graph followed by a constrained Delaunay triangulation of the whole sampling; after a constrained mesh optimization, the feature edges are perfectly preserved, while blended in the new mesh.*

using the face index map (see Section 2.3) and barycentric coordinates within the triangle to find the accurate 3D position[1]. The constrained Delaunay triangulation and the reprojection onto the original 2-manifold typically take a total of $200\ ms$ for 3000 vertices generated. Notice that the connectivity generated by a Delaunay triangulation in the parameter plane may not be the most relevant one. However, since all triangulations with a given number of vertices are all isomorphic to each other through edge swapping, we use this triangulation as an initial "guess", and will perform connectivity optimization as necessary.

4.2 Connectivity Optimization

For a fixed set of vertices obtained by resampling, the connectivity can be arbitrarily modified by simple edge swapping. Many optimizations can be easily implemented (see, for instance, tightest triangulation [40], minimum curvature [10]). We also used the following two simpler criteria:

Regularity Edge swaps can be performed in order to favor valence 6 for interior vertices, and valence 4 on boundary vertices. This is implemented by randomly picking a non-feature edge and performing an edge swap only if it reduces the valence dispersion. A few additional constraints can be added in order to prevent face flipping in the parameterization, or large geometric distortions for instance. Note that we can also balance the valences on both sides of each inner backbone. The "rib effect" [6] can therefore be obtained by forcing exactly two neighbors on each side of a sharp edge whenever possible, as demonstrated in Figure 8.

Face Aspect Ratio Similarly, edge swaps can be performed to improve the aspect ratio of the triangles. In practice, we swap an edge between two triangles if it improves their *surface area/perimeter*[2] ratio (computed in 3D). This simple test often results in dramatic improvements, since the connectivity is now dependent on the embedding, and not solely on the parameterization.

4.3 Geometry Optimization

In addition to the connectivity optimization, we also perform a small geometry optimization to improve the geometric quality of the mesh. We perform a weighted Laplacian flow in the parameterization by moving every vertex $\mathbf{p}$ that does not belong to the feature graph by:

$$\Delta\mathbf{p} = \Delta t \sum_{i \in \mathcal{N}(\mathbf{p})} w_i(\mathbf{q}_i - \mathbf{p})$$

where Δt is a step chosen sufficiently small (*e.g.* 0.1), $\mathcal{N}(\mathbf{p})$ is the set of adjacent vertex indices to vertex $\mathbf{p}$, and $\mathbf{q}_i$ corresponds to the i^{th} adjacent vertex to $\mathbf{p}$.

[1]Although using $\mathcal{M}_\mathbf{X}$ would be faster, it is usually not accurate enough for small maps, and could therefore result in small noise in the reprojection.

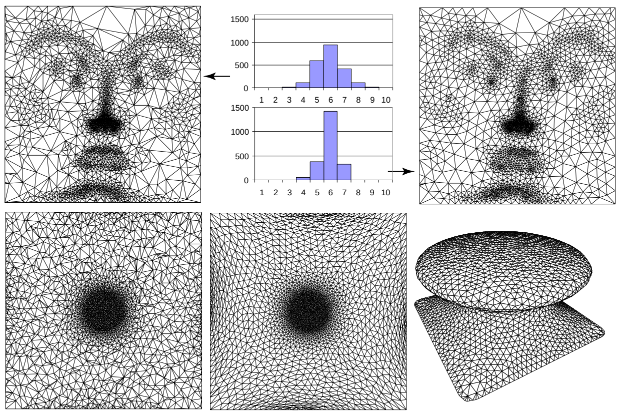
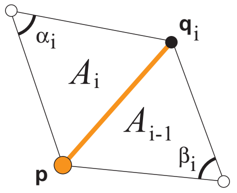


Figure 7: *Top: Left, Delaunay triangulation over the sampling. Right, after connectivity and geometry optimization. Middle, comparison of valence dispersion. Bottom: Left, Delaunay triangulation of a sampling performed upon the area map (leading to uniform mesh) of a mushroom-shape model. Middle, after minimization of local area dispersion. Right, the remeshed model. Note the uniformity obtained despite the strong area distortion due to the flattening process.*

Depending on the choice of remeshing that the user made when selecting the control map, we perform an adequate optimization by choosing the weights w_i so as to minimize an appropriate quantity. For example, if the users require a uniformly resampled mesh, we can minimize the local area dispersion by using the following weighting:

$$w_i = \frac{(A_i^{3D} \cdot cot(\alpha_i) + A_{i-1}^{3D} \cdot cot(\beta_i))}{\sum_{j=1}^{n} A_j^{3D}},$$



where α_i and β_i are the opposite angles in the parameterization as depicted, and A_i^{3D} and A_{i-1}^{3D} are the 3D face areas to the left and right of $\mathbf{pq}_i$.

This novel weighting has the quality of inducing no changes if the triangles are already of equal sizes, while producing a Laplacian smoothing ([28]) to iteratively improve the quality otherwise. The result of such an optimization can be seen in the bottom of Figure 8 for instance. The area distortion minimization is only a particular instance of the more general mesh optimization we offer. The area terms in the previous weights can be substituted by other values, based on the control map used. For a curvature-based map for instance, we replace the area terms by integrals of the control map over the associated triangles. Indeed, a single pass over the control map suffices to collect the integral of the map over each triangle. These integrals, measuring the ”amount of curvature” (or amount of anything the control map measures) contained in a triangle, are therefore appropriate weighting values if one wants to guarantee a triangulation adapted to the control map. This efficient smoothing generally happens in a matter of seconds, leading for instance to the results on Figure 11.

4.4 Combined Optimization

Our system can create a variety of optimizations by alternating between connectivity and geometry optimization stages. For instance, uniform meshes can be obtained by alternating edge swaps favoring regularity with geometry optimization iterations minimizing area dispersion (see Figure 7 bottom). If the user wishes to create the “rib” effect [6], she can simply alternate edge swaps which favor regularity and a univariate Laplacian smoothing of the feature vertices (Figure 6, bottom right). Additional results are given in the following section.

5 Remeshing Results

Our current implementation is written in C++ using a sparse matrix structure, biconjugate gradient and SSOR preconditioning for computing the conformal parameterization. All operations on the maps are performed using a standard image processing library, using OpenGL hardware whenever possible. The sampling previews use standard OpenGL texture mapping. All result timings are given for a 1 GHz PIII with 256 MBytes of memory. Figure 8 illustrates uniform remeshing of the *fandisk* at various resolutions using a 800×800 control map. Note how the 1D error diffusion performs well all the way from 200 vertices to higher complexity along the backbones of the feature skeleton. The conformal mapping is performed in $3.1s$ using SSOR with over-relaxation, and all the maps are computed in $1.2s$ total, while each sampling is done at an interactive rate in $160ms$. For the 2.5k vertex version the constrained Delaunay triangulation [35] takes $190ms$, and the optimization stage takes $5s$ overall. The final 3D mapping takes $250ms$. Note that our goal of sampling at interactive rates is achieved, greatly increasing the user's productivity and workflow.

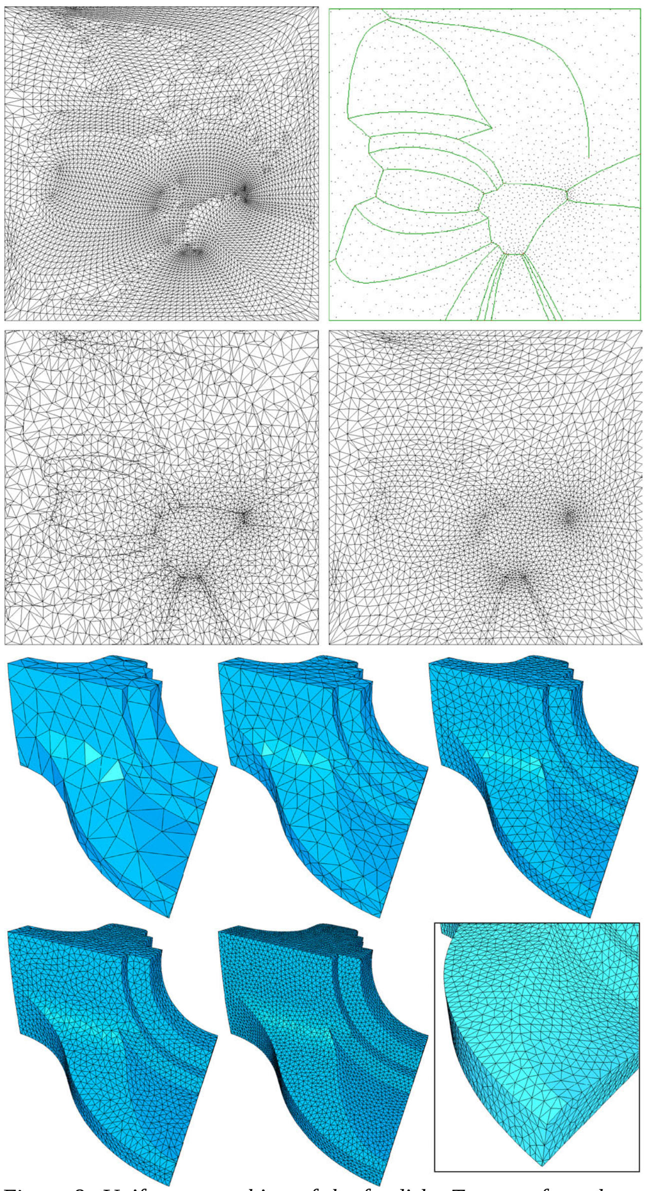


Figure 8: *Uniform remeshing of the fandisk. Top: conformal parameterization, and sampling obtained by error diffusion with 2.5k vertices with superimposed feature skeleton. Middle: result of constrained Delaunay triangulation before and after uniformity optimization. Bottom: several uniform remeshings with 0.2, 0.6, 1.4, 2.5 and 50k vertices respectively. Note the excellent behavior of the 1D error diffusion along the backbones, leading to consistent density between sharp edges and planar areas.*

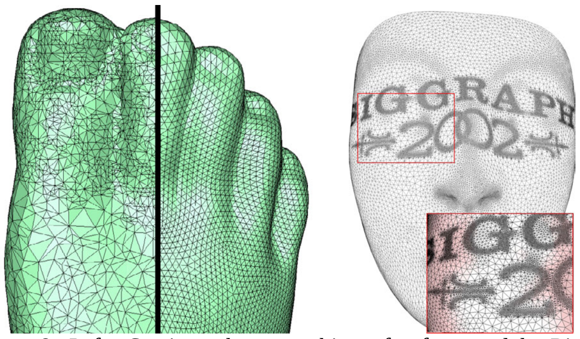


Figure 9: *Left: Semi-regular remeshing of a foot model. Right: Mesh created by pasting an image on the importance map (useful for animations and displacement maps).*

Figure 10 illustrates an example of uniform geometry remeshing of the *MaxPlanck* model using a 3 patch atlas. The original mesh (23kV) is uniformly remeshed to the requested 8.3kV. Notice that the final, remeshed model shows no signs that it was created using 3 independent patches. In Figure 11 the *MaxPlanck* model is remeshed with various transfer functions over the curvature map. The first example is uniform (*i.e.*, flat transfer function) with 15kV, and the three following examples are generated using a progressively increasing gamma function over the curvature map. All intermediate meshes ranging from uniform to adapted sampling can be obtained easily just by increasing the gamma or other custom transfer functions. Figure 9(left) shows a semi-regular remeshing of the *foot* model by applying regular subdivision in parametric space over a uniform base mesh. Figure 9(right) illustrates a custom-tailored sampling.

6 Conclusions and future work

We have demonstrated a novel, versatile technique for interactive geometry resampling that allows a very fine and easy control over the desired quality of the mesh. We substitute the original geometry by one or more 2D maps on which numerous operations such as halftoning and integration can be performed in real-time. Once an initial, near-optimal resampling has been designed, a fast optimization is performed to perfect the resulting mesh. We allow the user to custom design the mesh based on their requirements at interactive rates thereby increasing their productivity in creating a wide variety of meshes.

Many additional features can be added to our framework. We are investigating error diffusion in a quadtree data structure (to avoid the possibly large memory requirement of our current approach), anisotropic remeshing (possibly using ellipse packing) on a tensor control map of the principal curvatures, and hierarchical solving to accelerate the possibly slow parameterization stage. Finally, we plan to use our remeshing engine for other projects such as compression (how to remesh a surface to obtain the best rate/distortion tradeoff), as well as better geometric approximation.

Acknowledgements The work reported here was supported in part by the IMSC NSF Engineering Research Center (EEC-9529152), by the ECG project of the EU No IST-2000-26473, and by a NSF CAREER award (CCR-0133983). Special thanks to Victor Ostromoukhov for sharing his insights about error diffusion techniques, Yiying Tong for coding help with matrix preconditioning, and the SIGdraft reviewers for helpful comments. Models are courtesy of Hugues Hoppe, Leif Kobbelt and Peter Schröder.

References

[1] www.cgal.org: Computational Geometry Algorithms Library.

[2] BOROUCHAKI, H. Geometric Surface Mesh. In *2nd International Conference on Integrated and Manufacturing in Mechanical Engineering* (may 1998), pp. 343–350.

[3] BOROUCHAKI, H., GEORGE, P. L., HECHT, F., LAUG, P., AND SALTEL, E. Delaunay Mesh Generation Governed by Metric Specifications. *Finite Elements in Analysis and Design 25* (1997), pp. 61–83.

[4] BOROUCHAKI, H., HECHT, F., AND FREY, P. J. Mesh Gradation Control. In *Proceedings of 6th International Meshing Roundtable, Sandia National Labs* (oct 1997), pp. 131–141.

[5] BOSSEN, F., AND HECKBERT, P. A Pliant Method for Anisotropic Mesh Generation. In *5th Intl. Meshing Roundtable* (oct 1996), pp. 63–76.

[6] BOTSCH, M., AND KOBBELT, L. Resampling Feature and Blend Regions in Polygonal Meshes for Surface Anti-Aliasing. In *Eurographics proceedings* (sep 2001), pp. 402–410.

[7] BOTSCH, M., RÖSSL, C., AND KOBBELT, L. Feature Sensitive Sampling for Interactive Remeshing. In *Vision, Modeling and Visualization proceedings* (2000), pp. 129–136.

[8] DE COUGNY, H. L., AND SHEPHARD, M. S. Surface Meshing Using Vertex Insertion. In *Proceedings of 5th International Meshing Roundtable, Sandia National Labs* (oct 1996), pp. 243–256.

[9] DESBRUN, M., MEYER, M., AND ALLIEZ, P. Intrinsic parameterizations of surface meshes. In *Proceedings of Eurographics* (2002).

[10] DYN, N., HORMANN, K., S.-J. KIM, AND LEVIN, D. Optimizing 3D Triangulations using Discrete Curvature Analysis. *Mathematical methods for curves and surfaces, Oslo 2000* (2001), pp. 135–146.

[11] ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution Analysis of Arbitrary Meshes. In *Proceedings of SIGGRAPH* (1995), pp. 173–182.

[12] ERIKSON, J., AND HAR-PELED, S. Optimally cutting a surface into a disk. In *Proceedings of the 18th Annual ACM Symposium on Computational Geometry* (2002). to appear.

[13] FREY, P. J. About Surface Remeshing. In *Proceedings of the 9th Int. Meshing Roundtable* (2000), pp. 123–136.

[14] GARLAND, M., AND HECKBERT, P. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. In *IEEE Visualization Conference Proceedings* (1998), pp. 263–269.

[15] GARLAND, M., WILLMOTT, A., AND HECKBERT, P. Hierarchical Face Clustering on Polygonal Surfaces. In *ACM Symposium on Interactive 3D Graphics* (2001).

[16] GEORGE, P. L., AND BOROUCHAKI, H., Eds. *Delaunay Triangulation and Meshing Application to Finite Elements*. HERMES, Paris, 1998.

[17] GRAY, A., Ed. *Modern Differential Geometry of Curves and Surfaces*. Second edition. CRC Press, 1998.

[18] GRIMM, C. M., AND HUGHES, J. F. Modeling Surfaces of Arbitrary Topology using Manifolds. In *Proceedings of SIGGRAPH* (1995), pp. 359–368.

[19] GU, X., GORTLER, S., AND HOPPE, H. Geometry Images. In *Proceedings of SIGGRAPH* (2002).

[20] GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. Multiresolution Signal Processing for Meshes. In *Proceedings of SIGGRAPH* (1999), pp. 325–334.

[21] GUSKOV, I., VIDIMCE, K., SWELDENS, W., AND SCHRÖDER, P. Normal Meshes. In *Proceedings of SIGGRAPH* (2000), pp. 95–102.

[22] HORMANN, K., LABSIK, U., AND GREINER, G. Remeshing Triangulated Surfaces with Optimal Parameterizations. *Computer-Aided Design 33* (2001), pp. 779–788.

[23] LAZARUS, F., POCCHIOLA, M., VEGTER, G., AND VERROUST, A. Computing a Canonical Polygonal Schema of an Orientable Triangulated Surface. In *Proceedings of 17th Annu. ACM Sympos. Comput. Geom.* (2001), pp. 80–89.

[24] LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *Proceedings of SIGGRAPH* (1998), pp. 95–104.

[25] LÉVY, B. Constrained Texture Mapping for Polygonal Meshes. In *Proceedings of SIGGRAPH* (2001), pp. 417–424.

[26] LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. Least Squares Conformal Maps for Automatic Texture Atlas Generation. In *Proceedings of SIGGRAPH* (2002).

[27] LINDSTROM, P., AND TURK, G. Fast and Memory Efficient Polygonal Simplification. In *IEEE Visualization Proceedings* (1998), pp. 279–286.

[28] MEYER, M., DESBRUN, M., SCHRÖDER, P., AND BARR, A. H. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds, 2002. *http://multires.caltech.edu/pubs/diffGeoOps.pdf*.

[29] OSTROMOUKHOV, V. A Simple and Efficient Error-Diffusion Algorithm. In *Proceedings of SIGGRAPH* (2001), pp. 567–572.

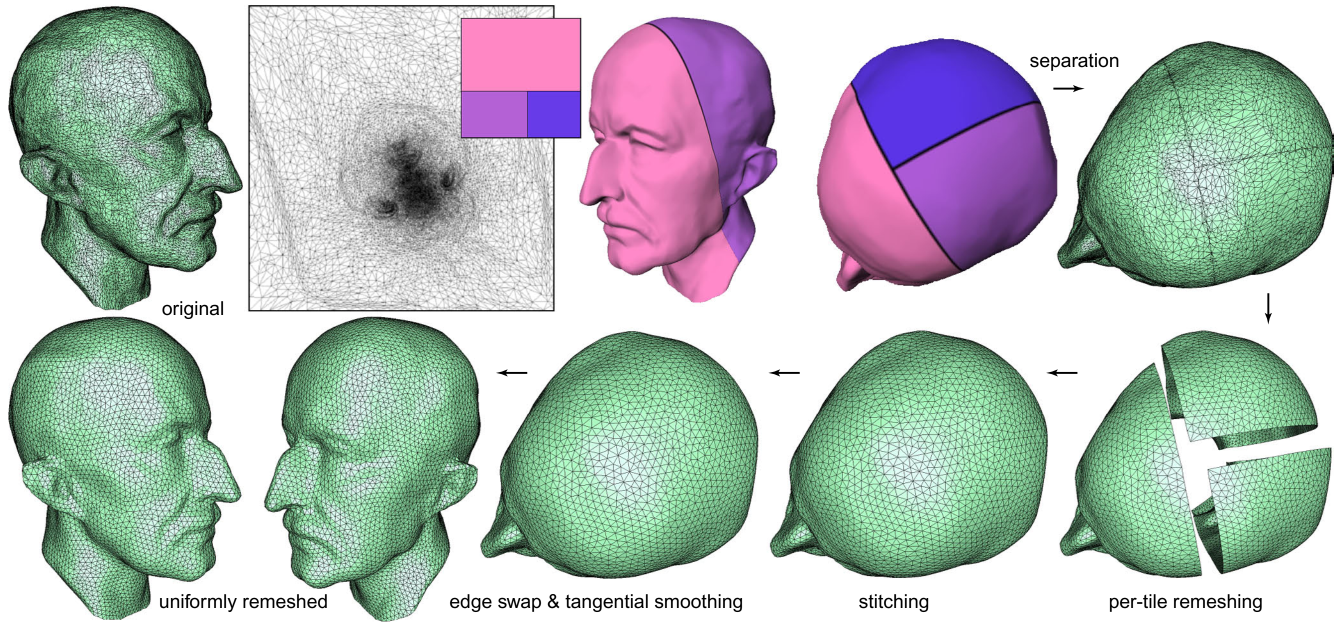


Figure 10: *Uniform remeshing of the MaxPlanck model. In clockwise order: Original mesh; Conformal parameterization; Parameterization-driven tiling with tree tiles requested; Three tiles meet at a corner; Mesh separation from the tiling; Burst view of the three tiles after independent uniform remeshing; The tiles put together require vertex stitching at the boundaries; A post-process swaps some edges and performs tangential smoothing along the stitching line; and the new model after uniform remeshing.*

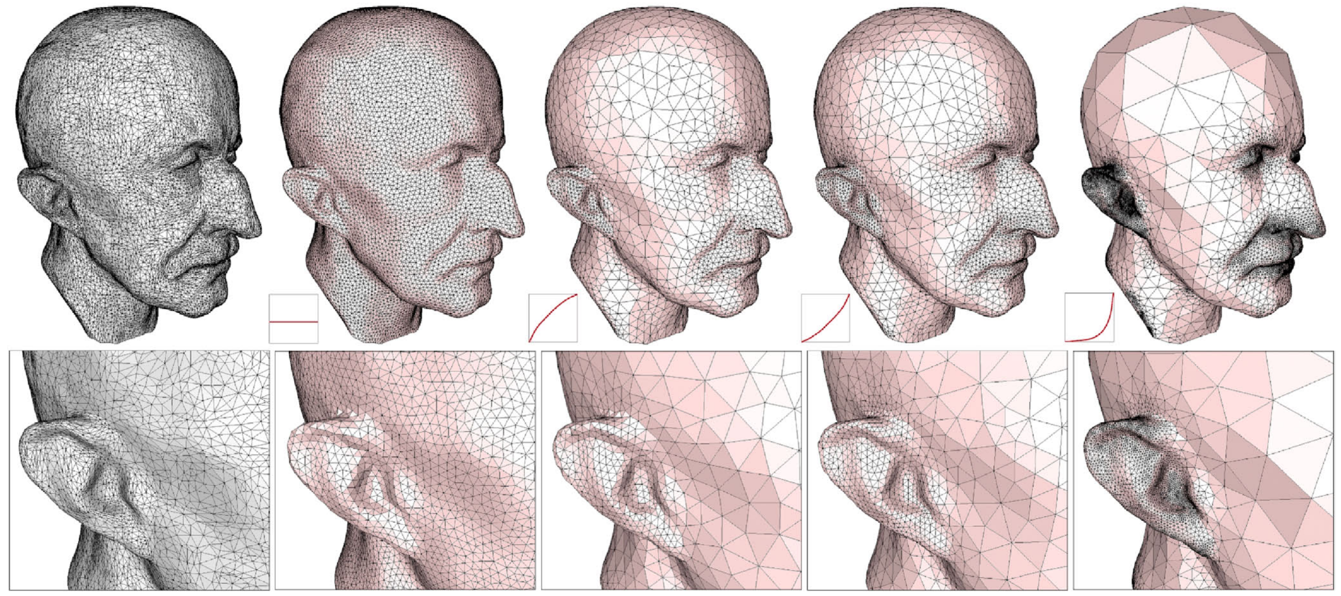


Figure 11: *Remeshing of the MaxPlanck model with various distribution of the sampling with respect to the curvature. The original model (left) is remeshed uniformly and with an increasing importance placed on highly curved areas (left to right) as the magnified area shows.*

[30] P. VÉRON, J.-C. L. Static Polyhedron Simplification using Error Measurements. *Computer-Aided Design 29(4)* (1997), pp. 287–298.

[31] PAULY, M., AND GROSS, M. Spectral Processing of Point-Sampled Geometry. In *Proceedings of SIGGRAPH* (2001), pp. 379–386.

[32] PINKALL, U., AND POLTHIER, K. Computing Discrete Minimal Surfaces and Conjugates. *Experimental Mathematics 2(1)* (1993), pp. 15–36.

[33] RASSINEUX, A., VILLON, P., SAVIGNAT, J.-M., AND STAB, O. Surface Remeshing by Local Hermite Diffuse Interpolation. *International Journal for numerical methods in Engineering 49* (2000), pp. 31–49.

[34] SHEFFER, A. Spanning Tree Seams for Reducing Parameterization Distortion of Triangulated Surfaces. In *Proceedings of Shape Modeling International* (2002). to appear.

[35] SHEWCHUK, J. R. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Proceedings of the First workshop on Applied Computational Geometry, Philadelphia, Pennsylvania* (1996), pp. 123–133.

[36] SIMPSON, R. B. Anisotropic Mesh Transformations and Optimal Error Control. *Appl. Num. Math. 14(1-3)* (1994), pp. 183–198.

[37] TRISTANO, J. R., OWEN, S. J., AND CANANN, S. A. Advancing Front Surface Mesh Generation in Parametric Space Using a Riemannian Surface Definition. In *Proceedings of 7th International Meshing Roundtable, Sandia National Labs* (oct 1998), pp. 429–445.

[38] TURK, G. Re-Tiling Polygonal Surfaces. In *Proceedings of SIGGRAPH* (1992), pp. 55–64.

[39] ULICHNEY, R. A. Dithering with Blue Noise. In *Proceedings of the IEEE* (1988), vol. 76(1), pp. 56–79.

[40] VAN DAMME, R., AND ABOUL, L. Tight Triangulations. *Mathematical Methods for Curves and Surfaces* (1995).

[41] VORSATZ, J., RÖSSL, C., KOBBELT, L., AND SEIDEL, H.-P. Feature Sensitive Remeshing. In *Eurographics proceedings* (sep 2001), pp. 393–401.