Direct Anisotropic Quad-Dominant Remeshing

Martin Marinov Leif Kobbelt

Computer Graphics Group
RWTH Aachen, Germany

Abstract

We present an extension of the anisotropic polygonal remeshing technique developed by Alliez et al. Our algorithm does not rely on a global parameterization of the mesh and therefore is applicable to arbitrary genus surfaces. We show how to exploit the structure of the original mesh in order to perform efficiently the proximity queries required in the line integration phase, thus improving dramatically the scalability and the performance of the original algorithm. Finally, we propose a novel technique for producing conforming quad-dominant meshes in isotropic regions as well by propagating directional information from the anisotropic regions.

1. Introduction

Anisotropic remeshing techniques are of great interest in at least two research areas in computer graphics. As proven in [21, 10], an optimal piecewise linear approximation of a smooth surface is achieved if one aligns the linear elements according to the principal curvature directions of the surface. In [6], normal noise due to piecewise linear discretization is reduced by exploiting the same property. Canonical shapes such as cylinders easily illustrate the power of the anisotropic alignment. Increasing uniformly the number of elements along the circumference results in better approximation, while splitting along the zero curvature direction does not improve the approximation quality. Therefore, for various computation intensive computer graphics and simulation applications, this purely geometrical argument renders anisotropic remeshing methods a very important mesh processing tool. Despite advances in raw computational and rendering power of recent hardware, polygon count is still seen as a main performance bottleneck, mainly due to bandwidth limitations.

Another important application for anisotropic remeshing algorithms is free-form surface modeling. Artists implicitly exploit the anisotropy of a model when drawing line strokes in a way which best describes the desired shape.

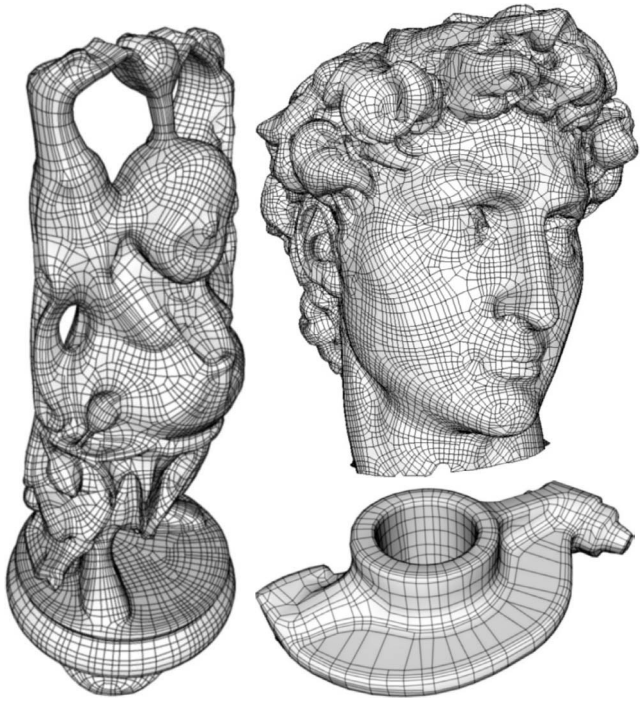


Figure 1. Anisotropic meshes.

Different rendering techniques [16, 18] simulating human-made drawings were proposed based on this observation. CAD professionals often compose a solid model by standard shapes such as cylinders, cones and spheres, thus again exploiting implicitly the natural anisotropy or isotropy of such primitives. In the view of this, anisotropic remeshing algorithms, that are able to automatically extract the semantical structure of a scanned object to a certain degree, are of great interest due to the significant time and cost savings achieved in the reverse engineering pipeline and the rapid prototyping production cycle. In addition, for many CAD applications quad-based representations are preferred, since they reflect better the symmetry of the modeled object.

1.1. Previous work

Many researchers in computational geometry, finite elements discretization and computer graphics explored methods for constructing anisotropic and quad-dominated meshes. A full review is beyond the scope of the paper, so we just outline some of the most related work. Initially quadrilateral remeshing algorithms were developed in [4, 19]. An interactive, user-guided method for generating quad-dominant meshes was proposed in [13]. In [12] a regular remeshing scheme for genus 0 surfaces is presented, but it does not take any anisotropic mesh alignment into account. Techniques for producing anisotropic triangle meshes were proposed in [5, 20]. A connection between quadric error metric simplification and the anisotropic alignment of the decimated mesh faces is derived in [15]. More recently, in [3] an anisotropic remeshing algorithm is proposed, based on the principal curvature directions estimation [9], which generates quad meshes inside anisotropic regions and conforming isotropic triangular meshes inside spherical and flat areas.

1.2. Anisotropic remeshing stages

The anisotropic remeshing technique introduced in [3] can be separated into several distinct subsequent steps (Fig. 2):

1. Initially a continuous, piece-wise linear curvature tensor field over the original triangle mesh is computed and filtered.
2. The original mesh is then sampled by building a network of curves following the principal curvature directions. A user-prescribed approximation tolerance in conjunction with the estimated curvature quantities defines their local density in different regions of the mesh.
3. The vertices of the newly generated mesh are obtained by intersecting the integrated curvature lines and the new mesh edges are defined along the curve segments connecting these intersection points. Finally, the mesh faces are generated from the extracted graph of vertices and edges, which results in a quad-dominated mesh due to the natural orthogonality of the principal curvature lines.

The specific contributions of our work are mainly concentrated in the second and most important step of the algorithm, while modifications of the first and the third part are presented in order to complement the changes introduced in the second stage. We improve the anisotropic remeshing technique in the following aspects:

Generality: Our algorithm does not require the construction of a global parameterization of the original mesh

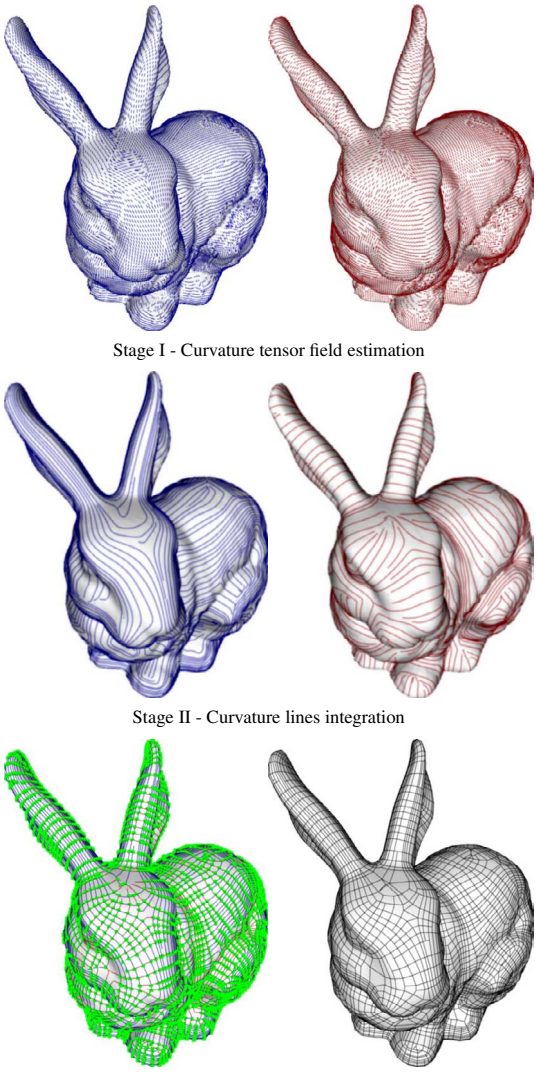


Figure 2. Anisotropic remeshing stages.

and therefore is inherently applicable to arbitrary genus surfaces. Avoiding the construction of a global parameterization, we also remove significant computational burden and implementation complexity from the remeshing process. It is well known that even employing sophisticated preconditioning and multi-grid techniques [17, 1], for high complexity meshes the global parameterization problem is hard to tackle.

Isotropic regions: The original algorithm uses a straightforward approach for dealing with isotropic regions by uniformly sampling them and building a constrained Delaunay triangulation using the samples. While there are geometric arguments in favor of this approach, it does not seem appropriate for meshing flat areas and tran-

sition regions between anisotropic patches (Fig. 3). Such configurations are often encountered in technical models, and thus are relevant in many applications. Therefore we propose a more sophisticated solution for sampling these regions. To obtain a coherent quad-mesh we propagate reliably estimated principal directions from the adjacent anisotropic patches into the isotropic region. Accordingly, we invert the line seeding strategy. Instead of placing the initial curvature line seeds in the *umbilic* regions, we place them in *strongly anisotropic* regions where the principal directions can be estimated reliably.

Performance: By proposing a simple and efficient data structure to speed up the proximity queries during the line integration phase, we improve significantly the scalability of the remeshing procedure. As a consequence our algorithm is able to process very large meshes such as the Buddha model ($1M$ triangles) and even the full resolution model of the head of Michelangelo's David ($4M$ triangles) in acceptable time. Unlike the original work, we avoid using exact precision arithmetics during the entire procedure. All the special cases we need to take care of are related to distinguishing the vertices, edges and faces of the (piecewise linear) input surface. We give specific details when describing the according step of our algorithm.

1.3. Overview

In Section 2 the curvature tensor field estimation and flattening procedures are briefly outlined. In Section 3 we describe a local parameterization approach used for integrating the curvature lines and an efficient data structure for performing the proximity queries required during the sampling phase. In Section 4, a novel technique for quad-dominant meshing of isotropic regions is presented. Our algorithm for directly meshing the extracted graph of vertices and edges in 3D is presented in Section 5. Finally, the results of our technique are discussed in Section 6.

2. Curvature tensor field

We compute the curvature tensor field of the mesh using the technique described in [9]. Since the obtained tensor field might be imperfect due to noisy surfaces, one usually applies a smoothing operator. To filter a tensor field, we use modified Laplacian smoothing directly in 3D. Our smoothing operator propagates curvature tensors from anisotropic areas into isotropic regions by applying weights which depend on the reliability of the local curvature direction estimates. We will elaborate on this in Section 4.

During the line integration phase (Section 3) one has to be able to evaluate and decompose the curvature tensor at every point in a (local) parameter domain $\Omega \subset \mathbb{R}^2$ associated with one or more faces of the original mesh. Since our curvature tensors live in $\mathbb{R}^3$ they have to be aligned to the tangent space, i.e., for every vertex v in Ω, we have to find a representation of its curvature tensor matrix in $\mathbb{R}^2$ (a process called flattening). Projecting an edge adjacent to that vertex to its estimated tangent plane in $\mathbb{R}^3$, we compute the angle ξ between the edge projection and the minimum principal tangent at the vertex. Rotating this edge's parameterization by ξ yields the minimum curvature direction in Ω. Since the maximum tangent vector is orthogonal, we can compose a 2×2 flattened matrix representing the curvature tensor in Ω. The curvature tensor at an arbitrary parameter point $(u, v) \in \Omega$ is calculated by linearly interpolating the flattened tensors at the vertices of the corresponding face.

Due to our local parameterization approach, not all edges adjacent to a flattened vertex are parameterized. Nevertheless, we always have at least two adjacent edges (Section 3.1) in the parameter domain and our flattening procedure picks the one that spans a smaller angle with the tangent plane. In order to make the projection to the tangent plane more robust after the tensor field computation and filtering we rotate the estimated principal directions at every vertex so that the normal derived from them coincides with the normal obtained through (weighted) averaging of the adjacent face normals. This is necessary especially after smoothing the tensor field, since the tensor-derived normals near strongly bended regions can deviate significantly from the actual surface normals and therefore the tensor flattening algorithm might yield incorrect results.

3. Curvature lines integration

Curvature lines tracing is the most essential part of the anisotropic remeshing procedure. In Section 3.1 we propose a robust method to integrate lines directly on the original surface using a local parameterization approach inspired by the technique described in [22]. At any time every line sample position is identified globally by the face f_i where it is located and its position inside it, i.e., by its *global barycentric coordinates* $[f_i, (u_i, v_i, w_i)]$.

In order to ensure that the lines satisfy some prescribed sampling density, proximity queries to the previously generated samples are performed at every integration step. Therefore the performance of the remeshing technique is very sensitive to the way how these queries are computed. We propose an efficient, yet simple solution for this problem in Section 3.2.

Four types of lines are tracked on the original surface. In addition to lines following *minimal* and *maximum* principal curvature directions, we distinguish also *boundary* and *feature* lines. The later are build by first constructing the feature graph [6], and then for every feature graph edge a line

consisting of its crease vertices is created. Similarly, boundary lines are defined for every boundary loop on the surface.

Line seeding: Since we do not have a global parametrization available, we have to place the seeds for the curvature line integration directly on the input mesh. Initially we start with a number of sparse random seeds that we place in the most anisotropic surface regions where the principal curvature directions can be estimated most reliably. Then while marching along a (minimum or maximum) curvature line we place additional seeds for the opposite type (maximum or minimum curvature resp.) along the way. The relative spacing of these additional seeds depends on the local curvature and the prescribed approximation tolerance. Later we re-start the curvature line integration at these seeds again spawning new curvature lines in orthogonal direction. This process is repeated until no new curvature lines can be generated without violating the estimated local line density.

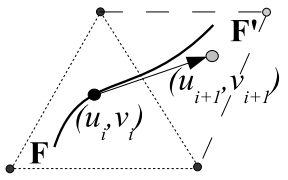
3.1. Local parameterization approach

Given a line seed, we parameterize isometrically the triangle where the seed resides by mapping one of its vertices to the parameter plane origin $(0, 0)$ and its outgoing half-edge to the u axis. Accordingly we flatten the three curvature tensors at the vertices of the seed triangle. Using the barycentric coordinates of the seed point inside the seed face, we find the initial parameter value (u_0, v_0). Now we start integrating the line from it.

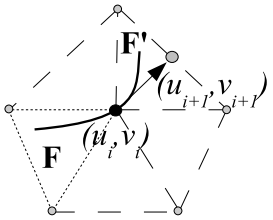
At any time t_i during the integration procedure there is only one operation which requires parameterization: given the current parameter value (u_i, v_i) and a corresponding update direction $d(u_i, v_i)$, *advance* the line (and its parameterization) to the point $(u_{i+1}, v_{i+1}) = (u_i, v_i) + h_i \cdot d(u_i, v_i)$, where h_i is an estimated integration step width [23]. We are then able to evaluate the surface point corresponding to (u_{i+1}, v_{i+1}) and decompose the (locally) flattened curvature tensor field at (u_{i+1}, v_{i+1}) to obtain the next direction $d(u_{i+1}, v_{i+1})$. The following special cases have to be considered:

1. If (u_{i+1}, v_{i+1}) is inside the parameterization domain of the current triangle then there is nothing to do.

2. If (u_{i+1}, v_{i+1}) lies across an edge of the current triangle F, we unfold its neighbor face F' at that edge. That is done isometrically by simply rotating the only one non-assigned vertex of F' to the parameter plane and correspondingly flattening the associated curvature tensor. This process is performed until that face is unfolded, which contains the point (u_{i+1}, v_{i+1}). To execute the in/out triangle check, we simply compute the barycentric coordinates of (u_{i+1}, v_{i+1}) with respect to the parameterization of the F'.



3. If during the unfolding in a face F we arrive exactly at a vertex v, we have to determine the adjacent face F' to continue. In general, there is no way to map isometrically all neighbor faces of v. Therefore we need to use an approximate solution. We compute the barycentric coordinates of (u_{i+1}, v_{i+1}) with respect to the parameterization of F. Mapping v to $(0, 0)$, we compute the polar map approximation [24] at it. Using the barycentric coordinates, we can recompute the parameterization of (u_{i+1}, v_{i+1}) with respect to the new map of F and find F'. F' is again re-mapped isometrically and we continue from it.



There is one more consideration, which leads to a substantial boost in performance when using the described approach. In case we need to compute the tensor at a predicted point during a higher order integration scheme (e.g. fourth order Runge-Kutta method), we need to have fast access to the parameterization of the current parameter point (u_i, v_i). We define as parameterization *context* the current triangle, its map and the flattened tensors at its vertices. For every prediction point step, we save the current context, execute the advance operation, evaluate the tensor at the predicted point, and then restore the context, thus returning quickly at our current position.

3.2. Proximity queries

An important aspect for the efficiency of the anisotropic remeshing scheme is the ability to quickly answer proximity queries of the following type: given a point p on the surface (with its global barycentric coordinates), find all previously computed curvature line segments which are inside a sphere with a given radius r centered at p. Here, the radius r depends on the local line density, while p is either a candidate line seed or a line sample generated during integration. A straightforward approach would be to use BSP-tree, octtree or another kind of a space-partitioning structure. However, global space-partitioning structures are not well suited for our case due to the fact that our search structure has to be updated frequently. At every integrated line point a query is performed, then once the line advances forward, the accepted sample is inserted into the structure, making it accessible for subsequent queries. Therefore we propose a simple and very effective solution which fits ideally to our needs.

During the integration of a line, we ensure that all of its segments are contained in just one face by splitting all line

segments which cross an edge of the input mesh. Every face keeps a list of all lines which pass through it and their segments inside it. This provides a natural, very localized decomposition of the samples in groups corresponding to the faces of the original surface.

Later on, once a query point p is given, we simply start conquering in a depth-first fashion all faces which have at least one vertex within r distance to p starting from the face where it resides. The set of all such faces we call a p-cell. We can traverse then every cached line sample inside the faces belonging to the p-cell and compute the distance to it, thus answering exactly the distance query. A subsequent query using p and larger r could be answered by simply extending the p-cell starting from its front faces. To avoid conquering large portions of the mesh when queries are executed during integration inside flat areas, we trim the query distance to some reasonable value set by the user, e.g., 5% of the bounding box diagonal of the model. Once p moves, we simply update the p-cell by removing all faces which are left out of the query distance and including all faces which now fall into it.

While it might seem that the proposed structure does more work than necessary, i.e., by evaluating distances to vertices of the original mesh and to line samples, this pays off even on very dense meshes, due to the fine granularity of the sample groups and the fact that a query does not have to traverse a tree like structure, but a simple array of faces. During the integration of the $2M$ line samples used for the high fidelity remeshing of the $1.09M$ triangles Buddha model, the structure answered on average $42K$ queries/second. The structure scales with respect to the original mesh density. This compensates the fact that the number of the integrated line points might be by magnitudes larger, especially during high fidelity remeshing. Additional advantages are its simplicity and especially the constant time insertion. The structure balances very well the different sampling density settings - during dense remeshing more samples are integrated, but the average query distance r is smaller, therefore less distance evaluations to the original mesh vertices are needed. Accordingly, in a sparse remeshing setting, the smaller number of integrated samples offsets the fact that r is larger. A possible pitfall are dense meshes which include a few very large faces - these however can be split easily in a preprocessing step.

4. Isotropic regions

Very few surface classes such as cylinders, tori and hyperboloids can be fully re-sampled exclusively using an anisotropic sampling technique. Complex technical models for example combine isotropic and anisotropic areas, ranging from flat regions to highly bended cylindrical shapes, often connected with hyperbolic blends. In the original work of Alliez et al., the remeshing algorithm employs a simple strategy for handling isotropic regions. They are uniformly sampled starting from the umbilic points which reside inside such regions and additional samples are added to saturate large flat areas. The samples are then meshed by building a constrained Delaunay triangulation which conforms to adjacent anisotropic mesh patches. It would be straightforward to adapt this approach in our setting, either by constructing the CDT directly in 3D [8], or mapping locally the isotropic surface areas to 2D and then building the CDT in the parameterization domain.

In this section we propose an alternative solution which propagates curvature directions from anisotropic areas to isotropic regions adjacent to them and seamlessly connects cylindrical and hyperbolic patches into a conforming quad-dominated mesh. In spherical regions the proposed technique is not so effective, and while it produces acceptable results, one would probably combine it with the original method. However for most flat and transition regions, most often encountered in technical modes (Fig. 5), our solution is clearly preferable.

In Section 4.1 we derive a robust criterion for distinguishing between reliable and non-reliable principal direction information and classify the faces of the original mesh according to it. Later on, during the line integration phase, whenever we have to cross an non-reliable region, we ignore the directional information inside it and continue the integration along a geodesic curve using a *trusted* direction, obtained during the tracking inside the last anisotropic area. Finally, once we leave the isotropic region, we snap the line to the most similar curvature direction, switching between minimum and maximum curvature lines if necessary.

4.1. Confidence estimation and propagation

We start by pointing out that geodesic disks around umbilic points are not sufficient to distinguish whether the region is isotropic or not. Sometimes the transition regions have the shape of long snakes which divide hyperbolic from cylindrical patches, even more complicated forms might exist (Fig. 3, left). Moreover, due to the smoothing operator, the estimated principal curvature values change significantly. In consequence, the umbilics of the filtered tensor field reside sometimes in incorrect locations, which do not correspond to the surface's true geometrical structure (Fig. 3, middle) and therefore we do not consider them as potential line seeds.

In [23, 3], the *deviator norm* quantity, based solely on the difference between the values of the principal curvatures at a point in the tensor field, is used to rate the *confidence* during the curvature line integration process. The norm of the deviator matrix shows correctly critical points of the tensor field, however we need an answer to a more general ques-

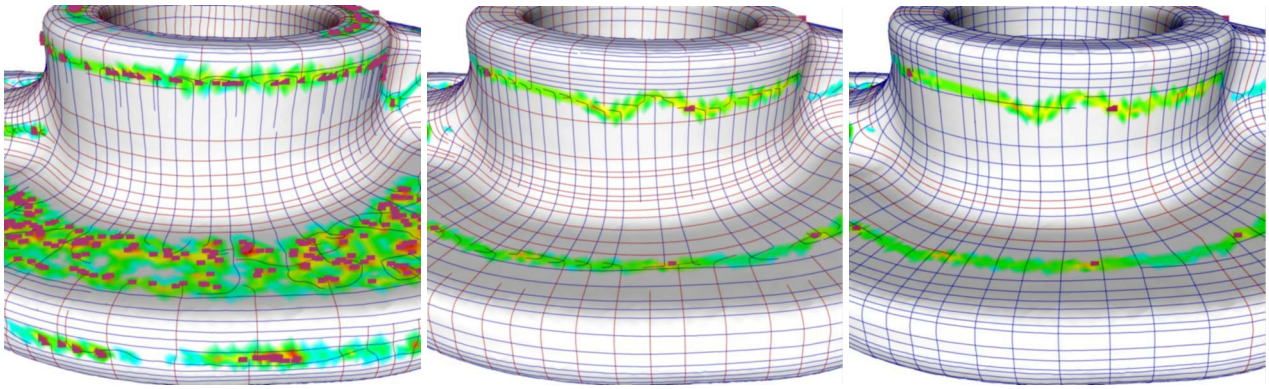


Figure 3. On the left we illustrate the integrated lines of curvature in a computed tensor field w/o any post-processing. Pink dots denote umbilic points. Blue lines follow the minimum curvature direction, and red lines the maximum one. Note the behavior of the lines inside the non-reliable (colored) areas on the surface. Our confidence estimation detects such areas even if there are no umbilics in proximity to them. In the middle, the change caused by smoothing the tensor field is shown. The number of umbilics is largely reduced, however the trajectories of the integrated lines are not improved - they still converge towards the remaining umbilics. On the right the result of our trusted direction integration is presented. The lines instantly change their type after leaving non-confident areas - their color represents now only the type of the initial principal direction used. The different sampling pattern is caused by the nature of the lines density estimates and the greedy seed placing strategy.

tion: how to detect regions where the principal directions do not align to the surface shape, but instead follow the directions leading to umbilic points? Therefore, away from umbilics, we base our confidence estimation not on a quantity related to the principal curvatures values, but on principal direction coherence.

We map every face of the original mesh individually to the parameter plane (as in Section 3.1) and flatten the minimal principal directions at its vertices. After computing the cosines of the three angles $\alpha_{j=0..2}$ between every pair of minimum directions, we define the confidence at this face by:

$$\Lambda(f_i) = \min_{j=0..2} |\cos(\alpha_j)| \ . \tag{1}$$

We now have a piecewise constant confidence scalar field on the original surface, ranging from 0 to 1, which we will use to alter the tensor field smoothing and the line integration phase. We extend the confidence definition (1) to the vertices of the original mesh by averaging the confidence of the faces adjacent to them. Now it is a simple matter to propagate curvature directions from confident regions to non-confident ones by weighting the coefficients in the discrete Laplacian operator:

$$T_i^{k+1} = \frac{\Lambda(v_i)}{2} T_i^k + \left(1 - \frac{\Lambda(v_i)}{2}\right) \frac{\sum_{j \in Nb(i)} \Lambda(v_j) T_j^k}{\sum_{j \in Nb(i)} \Lambda(v_j)} \ ,$$

where T_i^k is the 3×3 symmetric curvature tensor matrix at the vertex v_i during the k-th smoothing iteration. Note that due to the damping factor of $1/2$ for the confidence weight, the proposed operator not only propagates, but also smooths curvature tensors inside anisotropic regions.

4.2. Integration along trusted directions

We have now propagated trusted curvature tensors into isotropic regions, thus extending and smoothing the regions where we can integrate principal curvature lines reliably. However this is still not sufficient - some of these regions can never be annihilated completely since they represent significant discontinuities in the tensor field. The smoothing technique presented in the previous section only easies the identification of these regions and sharpens the differences between them and the adjacent anisotropic surface patches. We now alter the core of the anisotropic remeshing technique by swapping between minimum and maximum lines if necessary (Fig. 3, right).

Recall that we initially seed lines inside the confident, anisotropic areas. So we have a confident direction to start with. After entering a new face f_i during the line integration procedure, we check if the curvature directions inside f_i are sufficiently reliable by comparing $\Lambda(f_i)$ with some user-defined confidence threshold Λ_{min}. We found out that

the threshold value choice is not very critical, so we set it through all of our experiments to $cos(\pi/6)$. If $\Lambda(f_i) \geq \Lambda_{min}$ then we continue to follow the principal directions given by the tensor field. However, if $\Lambda(f_i) < \Lambda_{min}$, we instead follow the last *trusted* direction t_{tr}, defined as the average of the last few (reliable) directions used, thus integrating a geodesic curve crossing the non-reliable area. Once we leave the isotropic region, i.e., again $\Lambda(f_i) \geq \Lambda_{min}$, we check if t_{tr} is closer to either the minimum (t_{min}) or the maximum (t_{max}) principal curvature directions at the newly reached confident point and potentially alter the line type from here on according to the result of this test. More precisely, if $|t_{tr} \cdot t_{min}| \geq |t_{tr} \cdot t_{max}|$ we treat the following line segments as minimum line segments, otherwise as maximum line segments.

Several modifications are needed throughout the remaining components of the line integration phase to accommodate for the proposed change. The line integration procedure must be able now to change dynamically its behavior depending on the type of currently followed principle direction. In addition, since we do not have any notion of line type available inside isotropic regions, a modified proximity query is performed. Namely the trusted direction of the current line is compared with the (averaged) direction of the line segments of the already integrated lines within the current sampling density. If the angle between t_{tr} and the so computed approximate line directions is less than $\pi/4$, we then stop the integration of the current line, since continuing it, we will introduce small acute and large obtuse angles in the new mesh at the intersection point.

5. Meshing

Using the structure proposed in Section 3.2, it is extremely efficient to find the line intersections which constitute the new mesh vertices. For every confident face we just intersect all line segments inside it belonging to different line types. Inside non-confident faces we have to intersect line segments of the same curvature type too. Also if two lines intersect exactly at a vertex of the original mesh, then in general the same intersection point can not be computed in any of the faces adjacent to that vertex. Therefore we also keep for every vertex of the original mesh a list of the line segment identifiers coincident with it. Edges are defined along every line by connecting all subsequent intersection points on it. We merge all intersection points which are very close to each other and then remove all dangling and isolated vertices.

5.1. Building the half-edge structure

Since we will perform meshing directly in 3D, we have to remain consistent with the original surface orientation. Therefore, for every vertex in the new mesh we compute a normal vector based on the original mesh. If the vertex is a line intersection inside a face, we assign to it the face normal, and if the vertex coincides with an original vertex, then we assign to it the average normal at that original vertex. In order to build a half-edge (HE) mesh structure [7], for every vertex, we project all edges connected to it into the tangent plane defined by its precomputed normal. We select one of these edge projections and then compute the angle from it to all other edge projections. We sort the edges with respect to their angles in a counter-clockwise direction. Now we can easily define the HE structure around that vertex. The next HE for every incoming HE is the outgoing HE of the next edge in the sorted array. Correspondingly the last incoming HE is connected to the first outgoing HE.

5.2. Creating mesh faces

We construct the set of *free* HEs, which initially consists of all HEs, except those that correspond to a boundary line in the original mesh. Starting from one free HE, we traverse all next HEs determined by the next-previous relation defined above, until we reach the starting edge again. A new mesh face is created using the so extracted HEs, which are then removed from the free set. This operation is executed until no more free HEs remain.

In some cases, especially if we perform very coarse sampling, it is possible that the HE structure we obtained is not consistent everywhere and therefore the free HEs list cannot be emptied using the above operation. This situation is detected if the free list is no empty, but it is not possible to find another loop. Then we remove all edges which have two free HEs since these are exactly the inconsistently oriented edges. A final pass of the face creation operation takes care of the remaining mesh gaps.

Convex partitioning: During the face creation procedure, some extracted faces might be concave. Concave faces are undesired due to numerous problems when rendering and modeling them. We propose a simple algorithm for partitioning them directly in 3D, aiming at generating quad, anisotropic elements. It is well-known that convex partitioning in 3D is significantly more complicated problem than its 2D counterpart. Fortunately, in our case it is simplified by the fact that we can use the correct normals evaluated on the original mesh. Given a polygonal face of the remeshed mesh, for every of its HEs we define the sector normal at the vertex pointed by that HE as the cross product of its next HE vector and its opposite HE vector. Comparing the direction of the sector normal with the original normal at the vertex (by the sign of their dot product) reveals whether it is convex or concave.

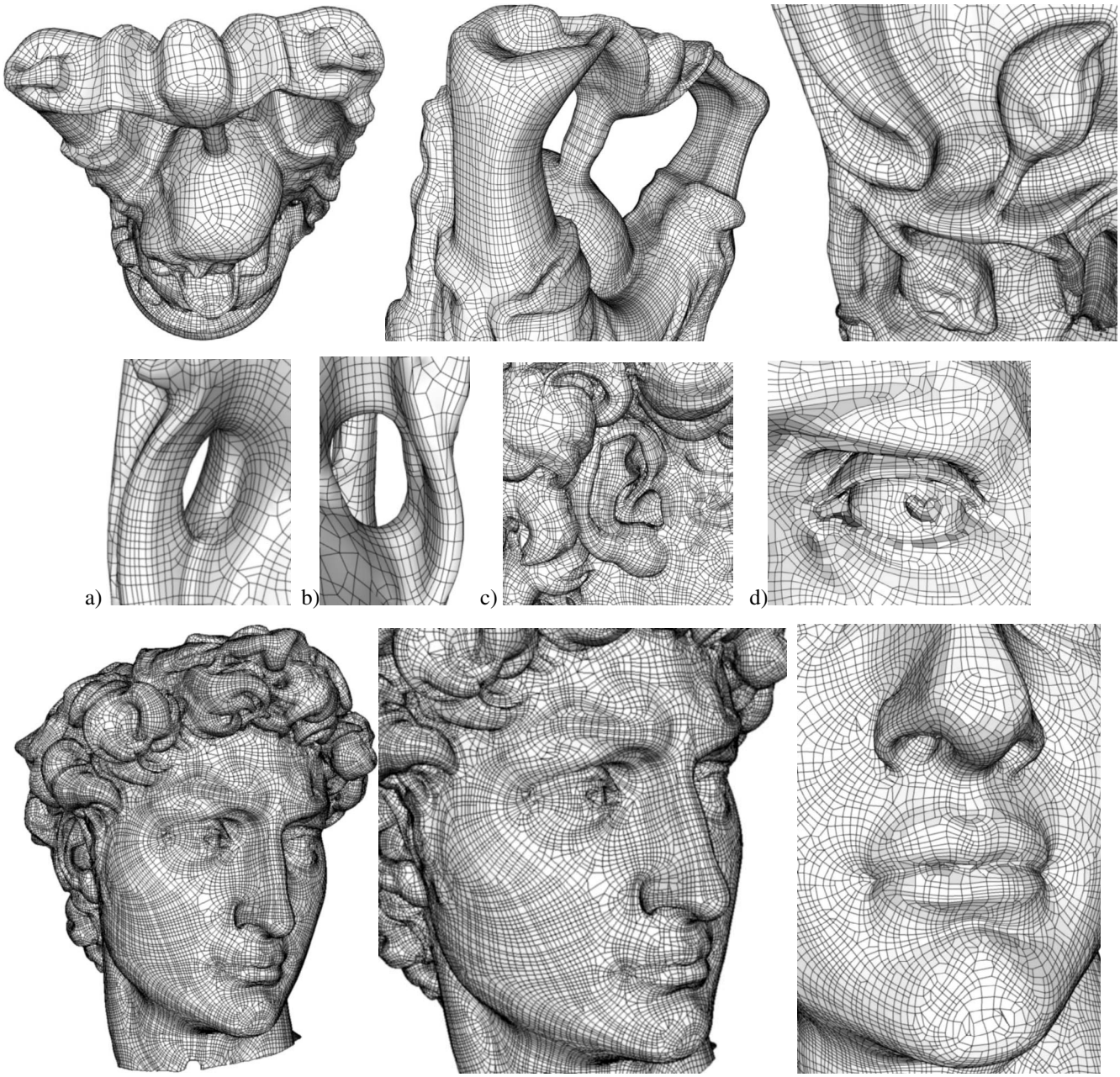


Figure 4. High fidelity remeshing of the Buddha (top) and Michelangelo's David (bottom) models.

Given a concave vertex in a face, we find all face diagonals going out from it, and choose one of them based on a score evaluated for each of the two faces which are produced if we split the concave face at it. The score function favors rectangular elements by rating highly angles which do not deviate significantly from $\pi/2$. Once a diagonal is chosen, we split the face at it, and then recursively run the algorithm on the resulting two new faces. Splits are performed until no more concave vertices remain. The same algorithm is used to partition faces which exceed some used-defined maximum valence, splitting them at the diagonal which yields the highest score.

6. Results and discussion

The choice of the examples in this section reflects the three main contributions of our work. The Buddha model is a well-known, high genus closed surface and therefore il-

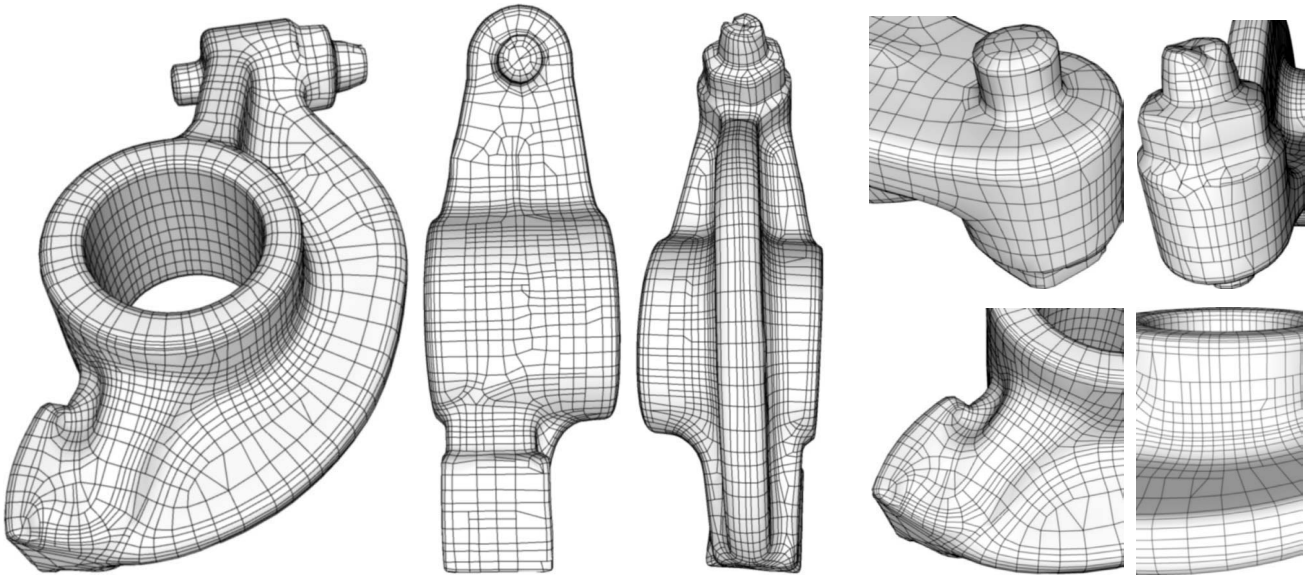


Figure 5. Anisotropic remeshing of the Rocker Arm model.

lustrates the ability of our scheme to handle arbitrary genus meshes. We remeshed the head of Michelangelo's David in its original resolution, thus focusing on the scalability of our technique. Finally, the Rocker Arm is a technical object which exhibits many isotropic regions and demonstrates the results of our reliable direction propagation algorithm.

Input Complexity			
Model	Buddha	David	Rocker Arm
Faces	1.09M	4M	80K
Vertices	545K	2M	40K
Output complexity: Low resolution (Fig. 1)			
Faces	18K	30K	2.3K
Vertices	19K	31K	2.3K
Output complexity: High resolution (Fig. 4, 5)			
Faces	120K	135K	5K
Vertices	125K	140K	5.2K
Timings	3-4min	8-9min	20-25sec
Memory	460MB	1.6GB	49MB

Timings depend on the input mesh complexity and slightly vary w.r.t. the output.

Table 1. Input, output complexity and timings.

Low resolution re-meshes of the three models are shown on Fig. 1. On Fig. 4, 5 we illustrate detailed, high-fidelity results produced by our technique. For quality comparison with [3], one could use the zoom-ups on Fig. 4c,d. Since we are able to process the original resolution input mesh of Michelangelo's David's head (in [3] a $80K$ vertices decimated version is used - [2]), the remeshing technique is able to capture more detail, which is visible on a fine scale, e.g., around the mouth, the eyes and the ears. On the other hand the increased level of detail causes a slightly less uniform line density in less curved regions, e.g., the cheeks. The radius of the averaging area for computing the curvature tensors was set to 1% of the bounding box diagonal for the Buddha and Rocker Arm models and 0.5% for the David's head model. The maximum valence was set to 5 for all examples because this leads to output meshes consisting of quads, T-joints and triangles. Depending on the prescribed approximation tolerance, it is possible that the output mesh has smaller genus than the original since tiny handles might not be sampled during the line integration process. This can be seen as a side effect which removes topological noise from the input mesh [14].

As noted in Section 3.2, the performance of our algorithm scales with respect to the complexity of the input mesh and only slightly varies depending on the output density. In the view of the timings (Table 1), we consider the performance of our algorithm to be competitive even with greedy decimation algorithms such as [11]. On the same computer (Pentium IV 2.8GHz, 2GB RAM), decimating the full resolution David's head mesh to $140K$ vertices takes $7min$. Note that some of the parameters of the anisotropic remeshing algorithm, e.g., the tensor field averaging area, the amount of curvature adoption, etc, affect the performance considerably.

Implementation details: We used as a base mesh library the freely available OpenMesh [7]. Information re-

quired during the remeshing is on-demand attached and detached to the original mesh entities, using the convenient OpenMesh property mechanism. After the tensor field filtering, the vertices keep their normal vector, the minimal principal direction, the principal curvature values and a list of pointers to line segments passing through them. Temporary valid distance and parameterization properties of the vertices are frequently reused by different functions during the integration phase. For every face a list of line segments residing inside it is maintained. Standard double precision (64 bits) was used for performing the calculations and storing geometry information.

Future work: Our next goal is to investigate methods for improving the distribution of the integrated lines. While the local sampling density is estimated with respect to the prescribed approximation tolerances, in some applications one would like to be able to impose global uniformity requirements. A multi-resolution approach could help tackling such constraints - our experiments show that it is significantly easier to capture uniformly the object shape on smoothed, coarse versions of the models, while high frequency features could be re-sampled separately on finer resolutions. Another line of research would be to investigate the improved approximation by using asymptotic directions instead of principal directions in hyperbolic regions of the surface.

Acknowledgements: The used models are courtesy Cyberware, Stanford University and Igor Guskov. We would like also to thank the anonymous reviewers for their constructive comments.

References

[1] B. Aksoylu, A. Khodakovsky, and P. Schröder. Multilevel solvers for unstructured surface meshes. *In review, SIAM J. Sci. Comput.*

[2] P. Alliez. Private communication, 2004.

[3] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. Anisotropic polygonal remeshing. *ACM Transactions on Graphics. Special issue for SIGGRAPH conference*, pages 485–493, 2003.

[4] H. Borouchaki and P. J. Frey. Adaptive trangular-quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 41:915–934, 1998.

[5] F. Bossen and P. Heckbert. A pliant method for anisotropic mesh generation. In *5th International Meshing Roundtable*, 1996.

[6] M. Botsch and L. P. Kobbelt. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. *Computer Graphics Forum*, 20(3), 2001. ISSN 1067-7055.

[7] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. OpenMesh – a generic and efficient polygon mesh data structure. *OpenSG Symposium*, 2002.

[8] D. Cohen-Steiner, E. C. de Verdière, and M. Yvinec. Conforming Delaunay triangulations in 3D. In *18th Annu. ACM Sympos. Comput. Geom.*, pages 237–246, 2002.

[9] D. Cohen-Steiner and J.-M. Morvan. Restricted Delaunay triangulations and normal cycle. In *19th Annu. ACM Sympos. Comput. Geom.*, pages 237–246, 2003.

[10] E. F. D'Azevedo. Are bilinear quadrilaterals better than linear triangles? *SIAM J. Sci. Comput.*, 22(1):198–217, 2000.

[11] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics*, 31(Annual Conference Series):209–216, 1997.

[12] X. Gu, S. J. Gortler, and H. Hoppe. Geometry images. In *ACM Trans. Graph. 21(3)*, pages 355–361, 2002.

[13] I. Guskov, A. Khodakovsky, P. Schröder, and W. Sweldens. Hybrid meshes: multiresolution using regular and irregular refinement. In *Proceedings of the 18th annual symposium on Computational geometry*, pages 264–272. ACM Press, 2002.

[14] I. Guskov and Z. Wood. Topological noise removal. In B. Watson and J. W. Buchanan, editors, *Proceedings of Graphics Interface*, pages 19–26, 2001.

[15] P. Heckbert and M. Garland. Optimal triangulation and quadric-based surface simplification. In *Journal of Computational Geometry: Theory and Applications, 14(1-3)*, pages 49–65, 1999.

[16] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526. ACM Press/Addison-Wesley Publishing Co., 2000.

[17] N. Ray and B. Lévy. Hierarchical least squares conformal maps. In *11th Pacific Conference on Computer Graphics and Applications*, page 263, 2003.

[18] C. Rössl and L. Kobbelt. Line-art rendering of 3d-models. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, page 87. IEEE Computer Society, 2000.

[19] K. Shimada, J.-H. Liao, and T. Itoh. Quadrilateral meshing with directionality control by packing square cells. In *The 7th International Meshing Roundtable*, pages 61–75, 1998.

[20] K. Shimada, A. Yamada, and T. Itoh. Anisotropic triangulation of parametric surfaces via close packing of ellipsoids. In *International Journal of Computational Geometry and Applications, Vol.10, No.4*, pages 400–424, 2000.

[21] R. B. Simpson. Anisotropic mesh transformations and optimal error control. In *Proceedings of the third ARO workshop on Adaptive methods for partial differential equations*, pages 183–198. Elsevier North-Holland, Inc., 1994.

[22] O. Sorkine, D. Cohen-Or, R. Goldenthal, and D. Lischinski. Bounded-distortion piecewise mesh parameterization. In *Proceedings of IEEE Visualization '02*, pages 355–362, 2002.

[23] X. Tricoche. *Vector and Tensor Field Topology Simplification, Tracking, and Visualization*. PhD thesis, Schriftenreihe Fachbereich Informatik (3), Universität Kaiserslautern, Germany, 2002.

[24] W. Welch and A. Witkin. Free-form shape design using triangulated surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 247–256. ACM Press, 1994.