

# GEODESIC-BASED SURFACE REMESHING

Oren Sifri

Alla Sheffer

Craig Gotsman

*Center for Graphics and Geometric Computing  
Technion – Israel Institute of Technology  
{orensi/sheffa/gotsman}@cs.technion.ac.il}*

## ABSTRACT

Generation of surface meshes remains an active research problem despite the many publications addressing this topic. The main issues which must be treated by a good remeshing algorithm are: element quality, sizing control, approximation accuracy, robustness and efficiency. One reason surface meshing is such a challenging problem is the fact that using the Euclidean metric to measure distances between points on the surface can generate large discrepancies between the original surface and the constructed mesh. We solve this problem by using geodesic distances on the surface. The ability to accurately and efficiently compute geodesic distances, and propagate them across the mesh, permits us to generate quality surface meshes which closely approximate the input without using costly parameterization techniques.

**Keywords:** Triangle mesh, surface meshing, geodesic distances.

## 1. INTRODUCTION

3D models are used in applications ranging from animation and cinematography to heavy industry and scientific visualization. However, most existing surface mesh models can hardly even be called satisfactory. Most of them are not sampled properly and their basic elements — triangles — have poor, almost random shapes because of the 3D mesh acquisition process. Whether this is done using interactive solid modeling software or semi-automatically using a scanning device, it remains a tedious and error-prone procedure. Models generated by CAD software usually reflect regular sampling of the underlying parametric domain instead of the model features. The process of simplifying scanned models with millions of points is primarily concerned with preserving model geometry and topology and does not emphasize the quality of sampling and triangles. This results in meshes which usually cannot be used as-is for 3D applications. An intermediate step, correcting the basic mesh geometry and connectivity, while preserving features, is required. Such corrections, commonly known as *remeshing*, are a fundamental component in the field of *digital mesh processing*.

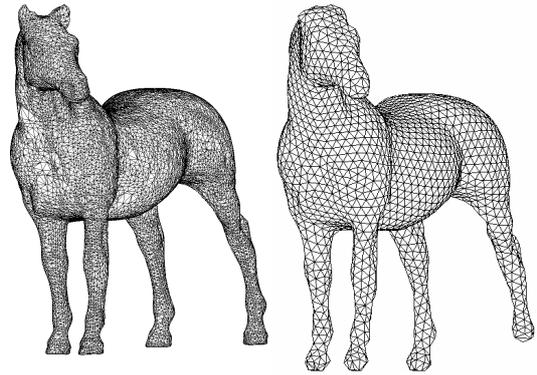
Remeshing tries to accurately approximate the model geometry with well-shaped elements. It also adjusts the sampling rates locally to match the amount of detail present. High-quality meshes are necessary for engineers performing numerical computations, such as *finite element analysis* that, for example, calculate mechanical stress, solve heat and flow differential equations or simulate such systems. A high-quality mesh conditions the system well, eliminating numerical errors and singularities that might otherwise arise. Hence, within the engineering community the emphasis is on quality. The computer graphics and modeling community, on the other hand, is concerned with another aspect of remeshing. Their focus is on the tradeoff between the visual quality of the result, the speed of the remeshing operation, and the optimization of the number of polygons in order to achieve interactive rendering speeds.

Over the last decade, an abundance of remeshing algorithms have been proposed. One group of algorithms, e.g. [10,11], is based on partitioning the 3D mesh into patches, and treating each patch separately, usually with subdivision techniques. While these techniques yield reasonable results, they are very sensitive to the patch structure, and the vertex sampling (or distribution) is difficult to control. More recent remeshing algorithms, e. g. [2,9,12,14] are based on global parameterization of the original mesh, followed by a resampling of the parameter

domain. After this, the new triangulation is “projected” back into 3D space, resulting in an improved version of the original model. The main drawback of the global parameterization methods is the sensitivity of the result to the specific parameterization used. Embedding a non-trivial 3D structure in the parameter plane severely distorts this structure, and important information, which is not specified explicitly, may be lost on the way. Even if the parameterization minimizes the metric distortion of the 3D original in some reasonable sense, it is impossible to eliminate it completely. Moreover, global parameterization is very slow, usually involving the solution of a large set of (sometimes nonlinear) equations. Because of the size of the system, this solution may be numerically imprecise, especially in regions where the connectivity has bad isoperimetric ratios. These regions correspond to protruding extremities in the 3D mesh (e.g. the legs of an animal), and they may be lost in the process. Additionally, 2D parameterization requires the surface to be cut to a disk-like topology, both to parameterize closed surfaces or surfaces with genus higher than zero, and to reduce the parametric distortion. These cuts introduce visible artifacts in the mesh. The main alternative to global parameterization is to work directly on the surface. Remeshing algorithms using this approach [2,8,16,18] usually involve difficult, inefficient and limited optimizations in 3D. For example, Frey and Borouchaki [8] perform local modifications in the tangent plane. In a subsequent work, Frey [7] uses a paraboloid to obtain a better approximation of the surface. These complex approximations are extremely slow and not always robust.

The common denominator of all the meshing algorithms is that they use Euclidean metrics, in the sense that the distance between points, even on the surface, is measured as the Euclidean distance between them. Since in most cases we should actually be using the geodesic distance, i.e. the distance between the points along the surface, this may introduce error culminating in distortion. The Euclidean distance may be considered a good approximation to the geodesic distances only at short ranges and in regions of low curvature. Otherwise it is quite different. Nonetheless, the published methods use Euclidean distances because they are much simpler to compute.

Assuming a constant sizing function, the vertices of the new mesh should be positioned on the surface such that the distances along the surface between close vertices are approximately equal. One way of achieving this is to build a “front” of vertices which advances across the surface at uniform “velocity” [3,17]. The front forms strips of triangles as it advances. The main problem with this method is that the front may meet itself, hence split and merge as it propagates. This complicates matters significantly, and a number of heuristics are required to control the process. The fact that Euclidean distances are employed here as well results in suboptimal results. Figure 1 shows the result of an advancing front technique implemented in a commercial package. The loss of high curvature features is a typical artifact. The result of applying our method on the same model with similar uniform sizing is shown in Figure 2(e). Due to these difficulties, advancing front techniques are not considered to be very attractive.



**Figure 1: Horse model (left). Original (right). Advancing front remesh, produced by an anonymous commercial mesh generator. Note how the details of the ears and hooves are lost.**

### 1.1 Our Contribution

This work introduces a novel remeshing method which operates directly on the 3D surface in a manner similar to the “advancing front” methods. So on the one hand, it does not involve any costly parameterization methods. On the other hand, it avoids all the pitfalls of the existing advancing front methods. Firstly, it uses geodesic distances instead of Euclidean distances. Secondly, it does not have to deal with major topological changes in the advancing front. This is achieved by segmenting the mesh into regions such that the treatment of each region is relatively straightforward. This also allows us to process meshes with arbitrary topologies.

## 2. ALGORITHM OVERVIEW

Our mesh generation procedure avoids the need for costly planar parameterization by computing accurate geodesic distances directly on the surface. The geodesic distances are computed using the “fast marching on triangulated domains” technique of Kimmel and Sethian [13]. We provide a brief overview of the technique in Section 3.

The basic meshing technique we use was first proposed by Adi [1]. It is based on generation of equidistant curves on the surface. An equidistant curve is the locus of all points on the surface at some fixed geodesic distance from a given point. Adi computed such equidistant curves from a single root point on the surface and then simply triangulated the strips between consecutive curves.

The difficulty with this simple approach, as with the advancing front techniques, is that equidistant curves may have complex topologies. The saddle regions where a single curve splits into several components can have arbitrary shape (Figure 2(a)). Without special treatment, the mesh in such regions will both disobey the sizing requirement and contain badly shaped triangles (Figure 2(b)). Adi did not provide a satisfactory solution to these problems, hence he was able to generate good meshes only for very simple models. We avoid this pitfall by first segmenting the surface into regions, such that the distance function is mono-

tone inside each region and therefore does not contain saddle points. Once the regions and the equidistant curves inside them are computed, each strip between two adjacent equidistant curves is meshed using a Voronoi tessellation of vertices distributed on the two curves. The next four sections describe the main components of the algorithm:

1. Computation of geodesic distances and equidistant curves.
2. Mesh preprocessing.
3. Segmentation into regions.
4. Triangle generation within each region.

The various stages of the algorithm are illustrated in Figures 2 and 4.

### 3. COMPUTING GEODESICS

The easy, but inaccurate, way to compute a "geodesic" distance between two vertices of a triangle mesh surface is to run a shortest-path algorithm on the mesh graph, where the weight associ-

ated with an edge is its length. Efficient algorithms, such as the Dijkstra algorithm [15], can compute these path lengths very efficiently, but can be shown to produce paths quite different from true geodesic paths. This is because the geodesic path does not necessarily pass through the mesh vertices, rather takes shortcuts through edges. See Figures 3(a) and 3(b) for a comparison. In our work we utilize the "fast marching on triangulated domains" algorithm of Kimmel and Sethian [13]. It computes approximate geodesic paths between two vertices in  $O(n \log n)$  time per path ( $n$  is the number of vertices in the mesh). Unfortunately, this algorithm does not always guarantee a correct result, in particular when the mesh contains triangles with obtuse angles. Kimmel and Sethian offer a solution to this problem, but it is rather complex and not always correct. Alternatively, the problem may be reduced by a preprocessing step which reduces the relative number of obtuse triangles in the mesh. The standard way of doing this is to refine the obtuse triangles so that most of the affected area is covered by smaller, but less problematic triangles. This is not guaranteed to remove all obtuse angles, but removing just the worst cases suffices to produce reasonably accurate geodesic distance computations. Fortunately, obtuse triangles do not occur so frequently, so the

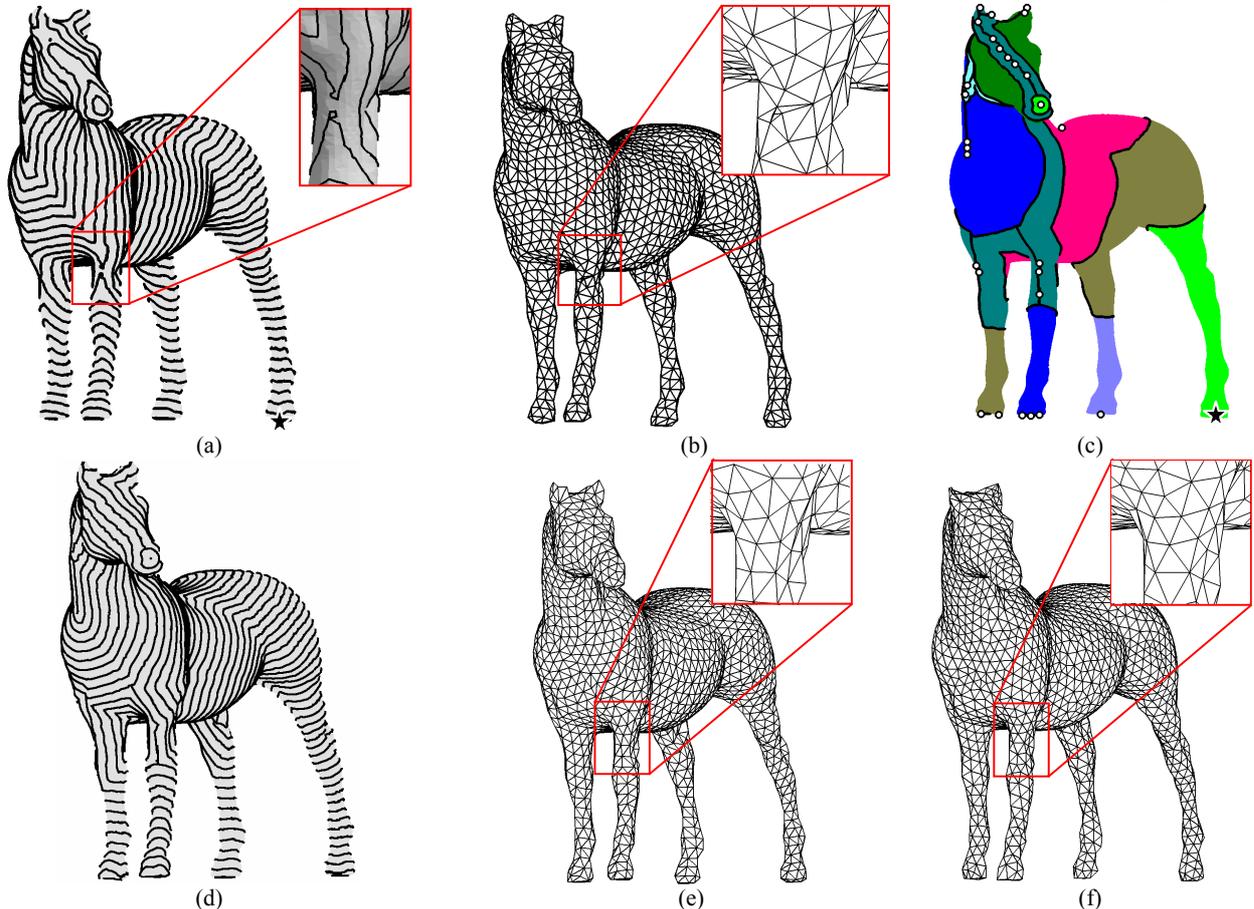
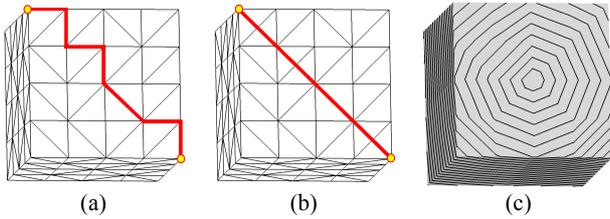


Figure 2: The effect of mesh segmentation on geodesic remeshing with uniform sizing of the horse model of Figure 1(a). (a) Geodesic curves and zoom on a saddle region. The root vertex is on the back left foot (marked by a star) (b) Resulting mesh with artifacts in saddle regions. (c) Segmented regions. The centers of leaf regions are highlighted by circles. (d) Geodesic curves formed with two-site distances on segmented mesh. Note that the saddle of (b) has disappeared. (e) Resulting mesh with no artifacts. (f) Mesh after post-processing.

algorithm produces quite good results in general, even without these two workarounds, although we employ them both.

The fast-marching method can also be adapted to non-uniform geodesic distances. The input may contain an arbitrary weight per vertex, where larger weights mean that the region surrounding that vertex is "harder" to pass through. The geodesics then take this information into account. This is a feature which is very useful to us, as we will see later.

We use the fast-marching method to compute the equidistant curves. This is done by computing the geodesic distance from the source to all other vertices of the mesh. The equidistant curve is then formed by connecting linear segments between points on the edges at a given distance, these too interpolated linearly between vertex distances. This approximation is obviously not correct when large triangles are involved, since the equidistant curves will be quite different from what they should be. See Figure 3(c); the equidistant curves on a plane are not circles, as would be expected. Here too, a possible solution, which we adopted, is to refine the mesh to contain many smaller triangles, forcing the algorithm to output more detailed geodesic information. Obviously, this increases the computation complexity.



**Figure 3: Shortest paths between the two yellow vertices. (a) Dijkstra. (b) Geodesic. (c) Equidistant curves based on geodesic distances.**

#### 4. MESH PRE-PROCESSING

Our meshing algorithm supports different sizing requirements on different regions of the mesh. Curvature-based sizing defined for each vertex is usually used to provide more accurate geometry approximation. Other per-vertex sizing data, such as those derived from analysis requirements, can be incorporated similarly. This data is sent as input to the fast-marching method, which conveniently, is able to use it.

We approximate the curvature at the mesh vertices using the method described in [5]. The sizing is then based on a combination of Gaussian and mean curvature. The relative weight of the two components is controlled by the user. In addition the user also controls the contrast or the gradient of the sizing gradation. This is achieved by transforming the sizing by some polynomial magnifying function.

The required number of triangles determines the desired edge length in the case of uniform sizing. This in turn determines the distance between consecutive equidistant curves on the surface.

This distance  $C_d$  is the desired edge length scaled by  $\sqrt{3/4}$  (the ratio between the height and the side in an equilateral triangle).

Now the root vertex for the distance computations is located. We compute the two vertices on the surface forming the maximal surface distance between them (the diameter) and select one of them as the root  $r$ . The computation of these two vertices is done using the following well known iterative procedure [6]:

1. Set  $r$  to some arbitrary vertex. Set  $D_{max}$  to zero.
2. Find the farthest vertex  $t$  from  $r$ . Set  $D_t$  to the distance between  $r$  and  $t$ .
3. If  $D_t > D_{max}$ , set  $r := t$ ,  $D_{max} := D_t$ , and goto 2.

This procedure is actually not guaranteed to find the mesh diameter, as it might get stuck in a local minimum, but for reasonably well-behaved meshes, this is rare. The root vertex  $r$  and the maximal distance  $D_{max}$  are used in the following stage to segment the mesh into regions. Figures 2(a) and 4(b) show the root vertex and the equidistant curves surrounding it for the horse and cactus models.

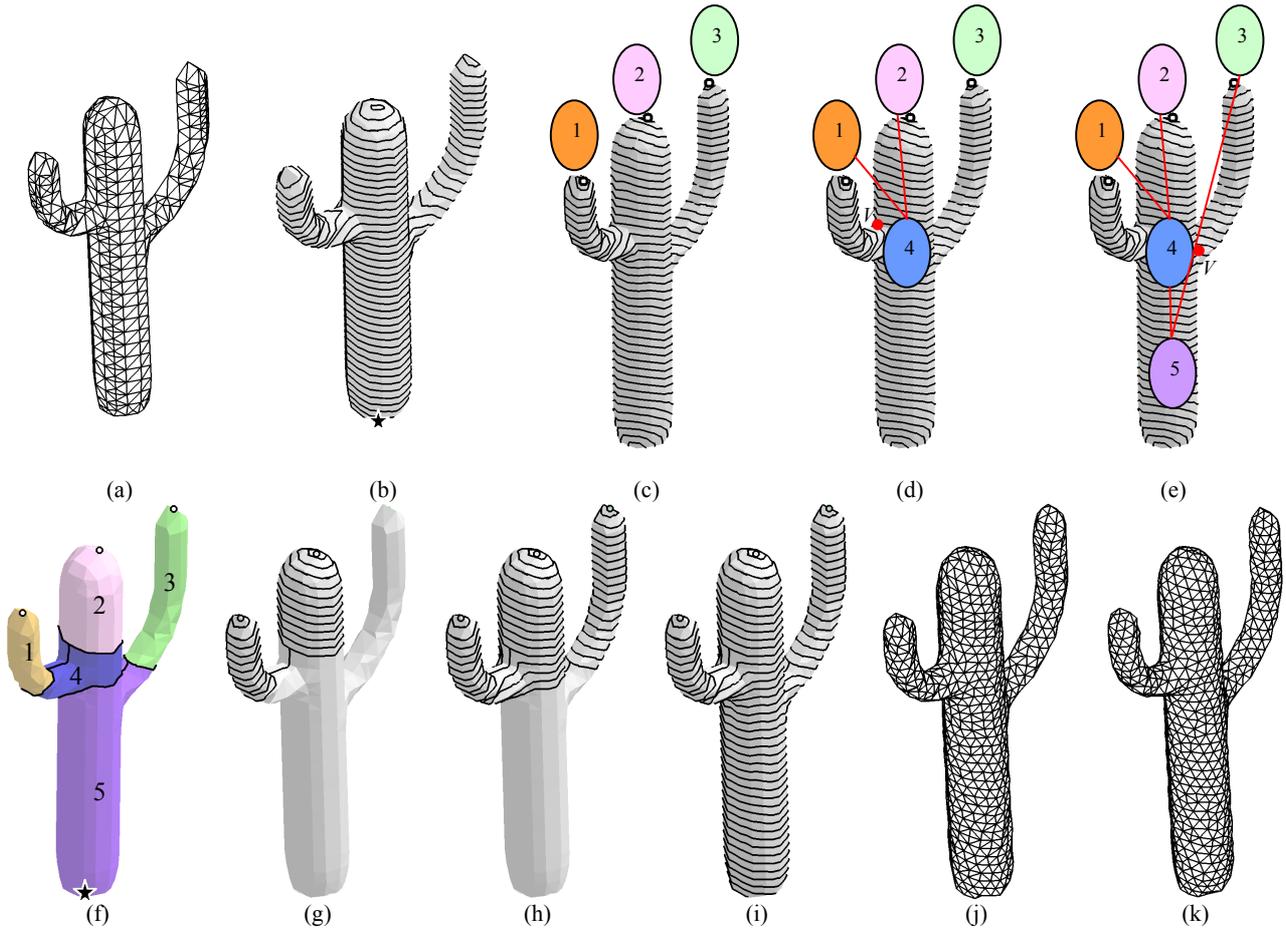
#### 5. REGION SEGMENTATION

Once the root vertex  $r$  is found we compute the geodesic distance  $D(v)$  from  $r$  to every other vertex  $v$  on the mesh.  $D(v)$  is then used to segment the mesh into regions to avoid mesh artifacts like those seen in Figure 2(b). The regions are constructed such that each equidistant curve within the region will be well-behaved. To guarantee this, each region must be a simply connected region. See Figures 2(c) and 4(f).

The regions form a tree structure containing three types of region nodes: leaves, interior nodes, and a root. The distinction between different types of regions reflects the properties of the distance function  $D$ . Leaf regions are formed around the maxima of  $D$ . Interior regions roughly correspond to the saddle points of  $D$ , and the root region is formed around the root vertex  $r$  (the global minimum of  $D$ ). The tree construction algorithm runs in two stages. First the topological structure of the tree (Figure 4(e)) is determined, and then the mesh regions corresponding to each node are computed (Figure 4(f)).

**Tree structure construction:** The set of vertices  $L$  which are the local maxima of  $D$  define the tree leaves (Figure 4(c)). Initially each leaf defines a degenerate tree consisting of a single node, producing a forest. A bottom up construction is performed with groups of trees connected by interior region nodes. Finally all the trees are joined into a single tree with a single root region node.

The tree construction uses a front propagation procedure on the surface starting from the set of leaf vertices  $L$ . The front propagation is based on the distance metric  $D$ . The fronts emanating from the leaf vertices are propagated so that at each step the vertex with smallest value of  $D$  is added to the front of the appropriate leaf  $l$ . When two fronts meet at a vertex, an interior region node is added to the tree as the parent of the two leaves (Figure 4(d)). The vertex is stored for further processing. The two fronts are merged, and the new front continues to advance using the minimal distance  $D$  from among its leaves. Whenever two fronts meet, interior nodes are created joining the sub-trees (Figure 4(c)). When only one front remains, i.e. all the leaves are connected into a single tree, the root node is introduced as the common parent. At the end of this procedure a region tree



**Figure 4: Algorithm stages. (a) Input cactus model. (b) Root vertex (highlighted by star) and curves equidistant from it. (c)-(e) Computing the region tree structure: (c) Maxima of distance from root. These are leaves of the region tree. (d) Propagating regions around maxima till regions 1 and 2 meet at interior region 4. (e) Continuing propagation of regions 4 and 3 until they meet at root region 5. (e) Resulting tree structure. (f) Resulting mesh regions. Regions 1, 2 and 3 are leaves, region 4 is interior, and region 5 is the root. (g)-(i) Computing equidistant curves within the regions: (g) Propagating equidistant curves down regions 1 and 2. (h) Propagating equidistant curves down region 4. (i) Complete set of strips. (j) Mesh after strip triangulation. (k) Final mesh after smoothing.**

structure (Figure 4(e)) is defined, but the boundaries of the regions still need to be determined.

**Tree regions construction:** The region boundary definition proceeds yet again bottom up, from the leaves to the root. When considering a region, the *top* of the region is defined as:

- the center vertex  $l$  for a leaf region;
- the boundary curve between the region and its children for an interior or root region.

The region boundary construction proceeds as follows:

While not all the boundaries have been constructed, find an interior node at which both the following hold:

- the top boundary is not defined,
- in both of its sub-trees the top boundaries are defined for all the nodes.

Note that at the beginning of the procedure all interior nodes with two leaf children satisfy this condition. Let  $t_1$  and  $t_2$  be the tops of the root nodes of the sub-trees. The vertex  $v$  at which the fronts of the sub-trees meet is known from the tree structure construction stage above. We now use it to compute the top boundary of the interior node which is the parent of the two sub-trees, as follows:

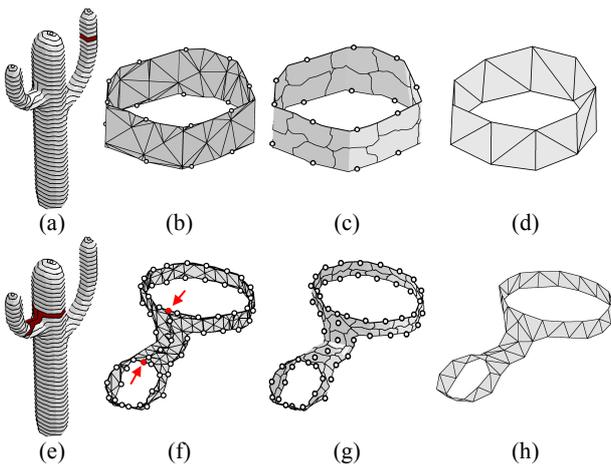
- Compute the equidistant curve  $C$  at distance  $D(v)$  from the root  $r$ . Note that  $C$  will contain  $v$ .
- Find two vertices  $v_1$  and  $v_2$  on  $C$  at minimal distance from  $t_1$  and  $t_2$  respectively.
- Compute the two equidistant curves  $C_1$  and  $C_2$  at distance  $C_d \lfloor D(t_1, v_1) / C_d \rfloor$  from  $t_1$  and at distance  $C_d \lfloor D(t_2, v_2) / C_d \rfloor$  from  $t_2$  respectively. This distance roundoff makes the distance from the tops of the two sub-tree root node regions to their bottom a multiple of  $C_d$ . It will result in an even curve distribution inside the regions at the meshing stage.

- Connect  $C_1$  and  $C_2$  into a “spectacles” shape using the shortest path between them which passes through  $v$ . This provides a single loop as the top boundary of the parent node.

The procedure is continued upwards inside the tree until all the region boundaries are computed. Figure 4(b)-(e) illustrates this procedure on the cactus model, resulting in the regions in Figure 4(f). Figure 2(c) shows the regions for the horse model. Note that many leaf patches on the head and legs of the latter are generated around minor local maxima. As a result  $D(l,v) < C_d$  and the patch degenerates to its center vertex  $l$ .

## 6. MESH GENERATION

We now proceed to place equidistant curves inside each region. The distance from the top of each leaf or interior region to its bottom is, by construction, a multiple of  $C_d$ . Hence curves can be spread evenly within each such region. Within the root region an error of up to  $C_d/2$  can occur. The algorithm spreads it equally during curve distribution by slightly increasing or decreasing the curve offset. This is illustrated in Figure 4(g)-(i). The equidistant curves constructed during this process are incorporated into the surface and used to generate the final mesh.



**Figure 5: Meshing strips on the cactus model. (a) Simple strip. (b) Strip with original triangles and new mesh vertices (circles). (c) Voronoi cells with new vertices as sites. (d) Dual Delaunay mesh. (e) “Spectacles” strip (f)-(h) Delaunay meshing. The T-junctions are marked by arrows.**

After the curves are computed the mesh can be generated using a final bottom-up sweep thru the tree. Starting from the leaves, vertices are distributed along each equidistant curve at edge length intervals. These curves are the boundaries of the strips (and spectacles). These vertices are added to the surface mesh (Figure 5(b) and 5(f)). When distributing vertices on spectacles they are first placed at the T-junctions, which are the meeting points of two or more equidistant curves (Figure 5(f)). The vertices placed on the curve connecting these form vertices interior to the strip. The vertex placement is then adjusted to optimize the element shape. The final mesh is constructed by triangulating the strip between consecutive curves. To generate the triangles, the algorithm computes the Voronoi cells around each vertex using

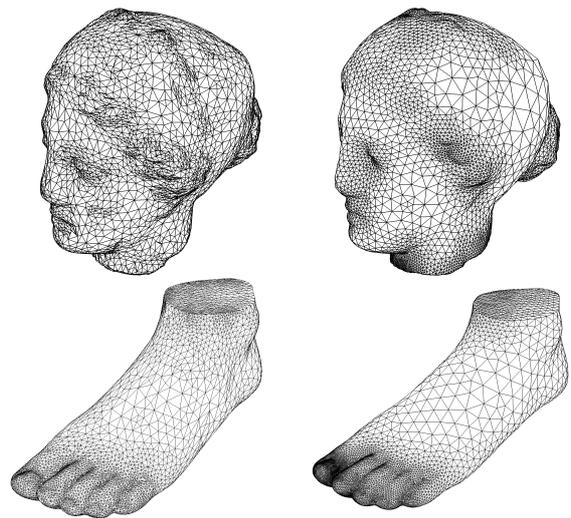
geodesic distances (Figure 5(c),(g)). The Delaunay triangulation dual to this Voronoi cell structure is computed and used as the mesh for the strip (Figure 5(d),(h)). The meshing stage can preserve surface features by incorporating feature edges into the mesh (see the gear model in Figure 7). Feature edges are detected using a dihedral angle criterion. The features are handled similarly to equidistant curves, with vertices distributed along them. Vertices are placed at intersections of feature edges with equidistant curves and the vertex distribution is adjusted accordingly. When meshing the strips, each piece of a feature edge which falls inside a strip is handled as part of the strip. The resulting strip triangulation contains both vertices distributed along the curves forming the strip and vertices distributed on the parts of feature edges overlapping the strip.

Once all the strips are meshed, the resulting mesh (Figures 2(e) and 4(j)) can be improved further using standard smoothing and local edge flip techniques (resulting in Figures 2(f) and 4(k)). This final stage distributes the vertices better on the surface, removing the visual “lines” formed by vertices placed along the equidistant curves. Even though these techniques improve the mesh regularity, they cannot be used excessively as they also cause damage to the mesh fidelity to the original.

## 7. EXPERIMENTAL RESULTS

We demonstrate the results of our remeshing method on several examples (in Figures 6 and 7). Figure 6 also compares to the result from the algorithm of [7]. The statistics for the resulting meshes are summarized in Table 1.

The sizing gradation control is demonstrated on the Venus head model (Figures 6 and 7). Our algorithm can successfully mesh both smooth and CAD-type models with corners and creases, as demonstrated by the foot and gear models. The foot model also demonstrates our method’s ability to handle significant sizing gradations resulting from large variations in curvature across short distances.



**Figure 6: Remeshing methods comparison. All models contain approximately 5,400 vertices. (Left) Local technique [7]. (Right) Our method. The statistics for both appear in Table 1.**

The example models showcase the method's ability to correctly capture sock-like shapes such as the animal's legs without requiring global parameterization. As is evident in the griffin and figure eight models, our method does not require any special treatment to handle models with genus greater than one. This, in addition to generating seamless meshes is yet another advantage of this technique over methods which utilize a parameter domain [2,9,12,14]. Our advantage over local methods [7] is in the mesh quality (Figure 6), with only a slight penalty in approximation error in some cases.

Both approximation and quality measures are shown in Table 1. The quality is demonstrated by the statistics of the minimal angle. The angles in the inputs are arbitrarily bad, but in most of the results, not many angles are less than 30° and the average angle is consistently above 50°. The statistics also include the Hausdorff distance from the original model measured using the Metro tool [4]. This is approximately 0.5% of the bounding box diagonal for most models, which is quite negligible. The superiority of using geodesic distance based advancing front instead of classical advancing front techniques in terms of approximation is clearly demonstrated by the horse meshes in Figure 1.

	Input size (#ver)	Result size (#ver)	Contrast	Input Angles (deg.)			Result Angles (deg.)			Appr Error (%)
				Min	%<30	Avg.	Min	%<30	Avg.	
Horse	19,851	2,205	0	1.60	24.9	35.8	22.9	0.14	51.5	1.48
Eight	766	985	0	21.4	37.7	47.9	32.9	0	51.9	0.48
Foot	10,016	5,421	0.5	2.07	27.7	37.7	10.0	0.63	51.2	0.27
Foot [7]		5,427					3.6	9.45	42.1	0.21
Venus 1	8,268	5,269	0	0.24	34.2	34.7	25.6	0.01	52.2	0.73
Venus 2		5,300	0.3				23.5	0.02	52.1	0.82
Venus 3 (Figure 6)		5,385	0.6				24.9	0.01	51.2	0.54
Venus [7]		5,319					0.24	23.81	37.4	0.21
Griffin	49,864	10,508	0.6	3.83	15.9	40.0	21.8	0.04	51.8	0.71
Dino	14,070	10,292	0.5	1.47	32.0	35.4	16.8	0.12	52.0	0.45
Camel	39,074	20,157	0.35	0.55	40.3	47.9	22.3	0.13	52.5	0.34
Gear	3,721	11,152	0	0.04	85.4	18.5	12.8	0.64	51.3	0.46
Tree (without segmentation)	13,778	988	0	3.2	42.7	30.3	0.07	5.7	46.6	2.1
Tree (with segmentation)	13,778	1,009	0				14.0	0.7	50.0	1.8

**Table 1: Statistics on some remeshed models. Contrast indicates the impact of curvature on sizing. Approximation error measures the Hausdorff distance between the resulting mesh and the input as a percent of the bounding box diagonal.**

The advantage to doing mesh segmentation as opposed to none at all is illustrated on the tree model in Figure 8. The corresponding statistics for the two results, as shown in Table 1, shows that the segmentation allows the algorithm to achieve a better result, both in terms of mesh quality, and in terms of approximation quality.

The run-time for the models varies between 60 seconds for the horse model to 600 seconds for the griffin on a 1.7 GHz Pentium 4 PC with 512 MB RAM. The time reflects the input and output sizes as well as the level of complexity which determines the number of regions generated by the algorithm. As demonstrated by the examples, the main advantage of our method is the ability to generate the number of triangles desired by the user while simultaneously approximating the input well and generating high quality elements.

## 8. CONCLUSION

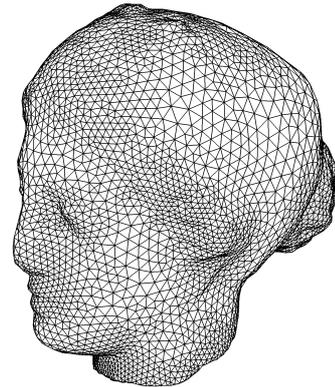
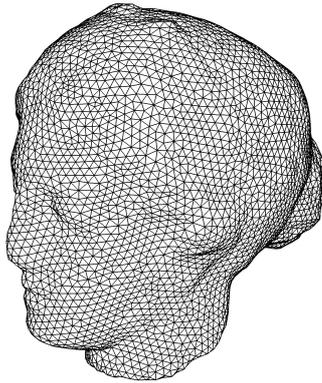
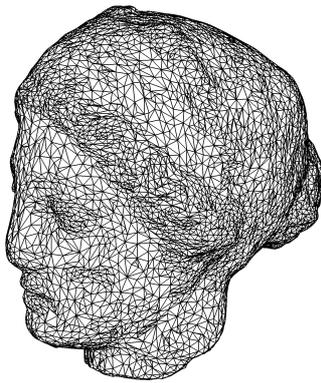
We have presented an efficient remeshing method which operates directly on the model surface without resorting to any sort of parameterization. At the heart of our algorithm lies convenient segmentation of the mesh according to geodesic distances. This allows us to incorporate any sizing function to control the distribution of mesh triangles. Each segment is meshed independently, and the result delicately smoothed. The results seem to be superior on all the models we have tested.

Future work will extend this method to mixed quad and triangle meshing, as well as pure quad meshing.

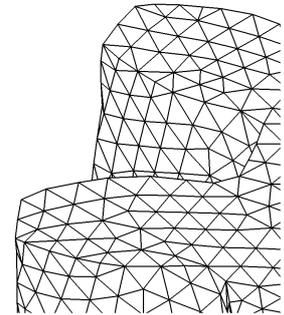
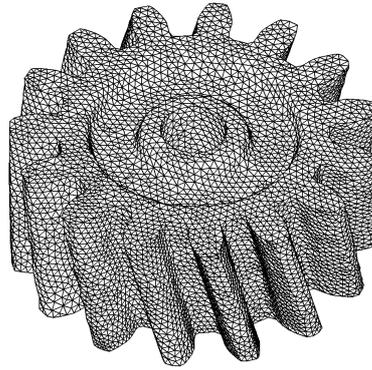
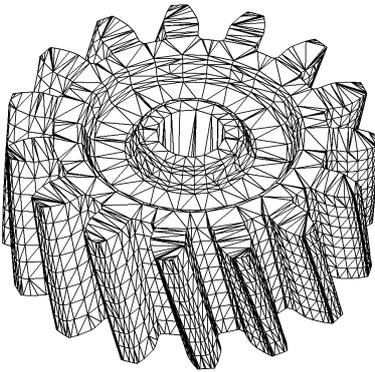
## REFERENCES

- [1] L. Adi. Fast computation of geodesic distances: Graphic applications. MSc. Thesis, Computer Science, Technion, 2002.
- [2] P. Alliez, M. Meyer, and M. Desbrun, Interactive geometry remeshing, ACM Transactions on Graphics. Special issue for SIGGRAPH conference, 21(3)(2002), pp. 347–354.
- [3] R.J. Cass, S.E. Benzley, R.J. Meyers and T.D. Blacker. Generalized 3D paving: An automated quadrilateral surface mesh generation algorithm. International Journal for Numerical Methods in Engineering, (1996).
- [4] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. Computer Graphics Forum (1998), 17(2):16-174.
- [5] N. Dyn, K. Hormann, S.-J. Kim and D. Levin, Optimizing 3D Triangulations using discrete curvature analysis, Innovations in Applied Mathematics, Vanderbilt University Press, (2001).
- [6] A Fischer, A. Manor, Y. Barhak. Adaptive parameterization for reconstruction of 3D freeform objects from laser-scanned data. Proceedings of Pacific Graphics (2002).
- [7] P.J. Frey, About surface remeshing, Proceedings of 9th International Meshing Roundtable, (2000), pp. 123–136.
- [8] P.J. Frey and H. Borouchaki, Geometric surface mesh optimization, Computing and Visualization in Science, 1 (1998), pp. 113–121.

- [9] X. Gu, S.J. Gortler and H. Hoppe, Geometry images, ACM Transactions on Graphics. Special issue for SIGGRAPH conference, 21(3), (2002), pp. 355–361.
- [10] I. Guskov, K. Vidimce, W. Sweldens, and P. Schroeder, Normal meshes, Computer Graphics. SIGGRAPH 2000 Proceedings, (2000), pp. 95–102.
- [11] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, Mesh optimization, Computer Graphics. SIGGRAPH '93 Proceedings, 27 (1993), pp. 19–26.
- [12] K. Hormann, U. Labsik and G. Greiner, Remeshing triangulated surfaces with optimal parameterizations, Computer-Aided Design, 33 (2001), pp. 779–788.
- [13] R. Kimmel and J.A. Sethian. Computing geodesic paths on manifolds. Proceedings of National Academy of Sciences, 95(15) (1998), pp. 8431-8435.
- [14] A. Sheffer and E. de Sturler. Parameterization of faceted surfaces for meshing using angle based flattening, Engineering with Computers, 17(3), (2001), pp. 326-337.
- [15] S. Skiena. The algorithm design manual. Springer, 2000.
- [16] V. Surazhsky and C. Gotsman. Explicit surface remeshing. Proceedings of the ACM/Eurographics Symposium on Geometry Processing (2003), pp. 17-27.
- [17] J.R. Tristano, S.J. Owen and S.A. Canann. Advancing front surface mesh generation in parametric space using a Riemannian surface definition. Proceedings of the 7th International Meshing Roundtable, 1998.
- [18] G. Turk. Re-tiling polygonal surfaces. Computer Graphics 26:2, SIGGRAPH 1992 Proceedings, pp. 55-64.



Venus



Gear

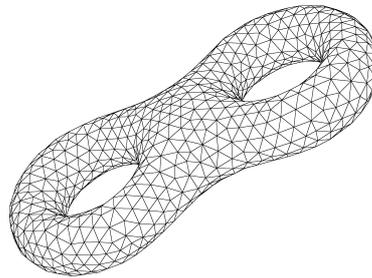
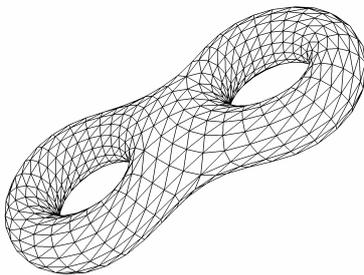
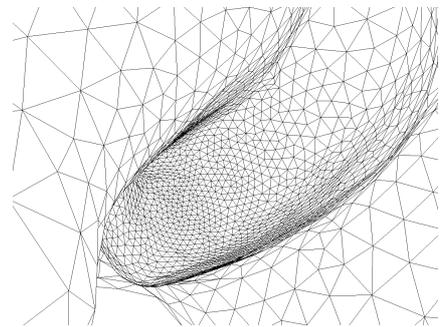
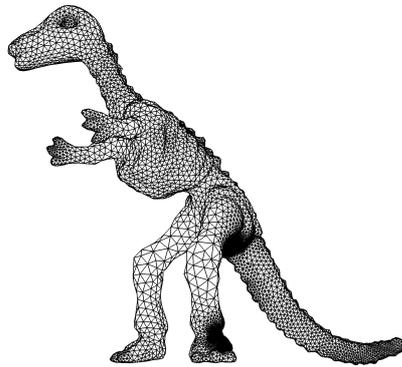
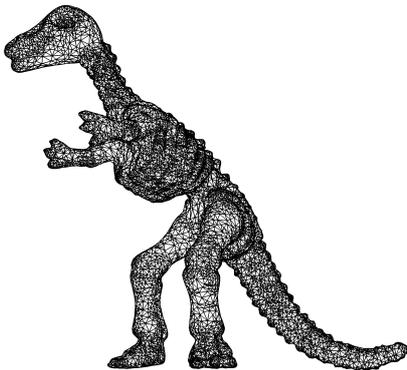
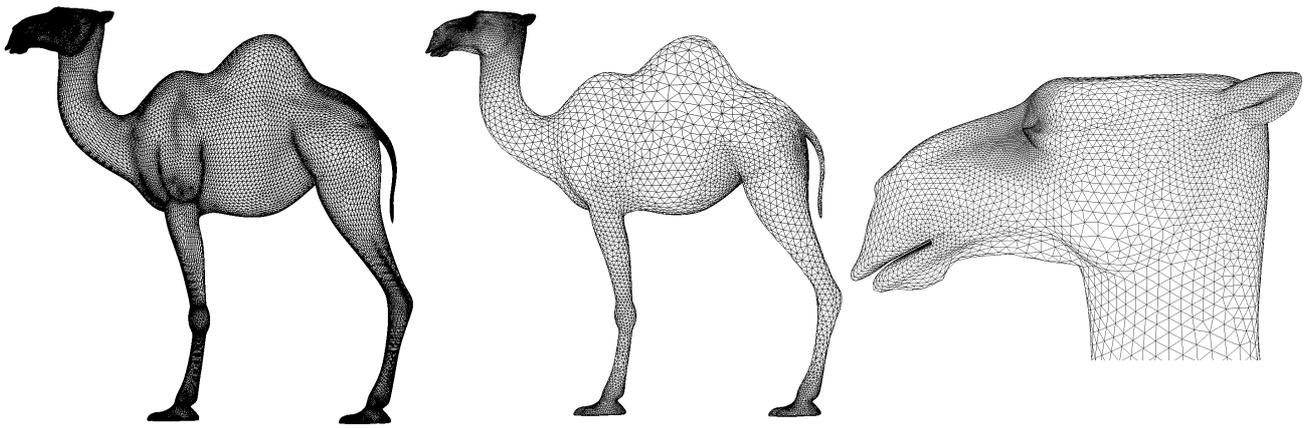


Figure Eight

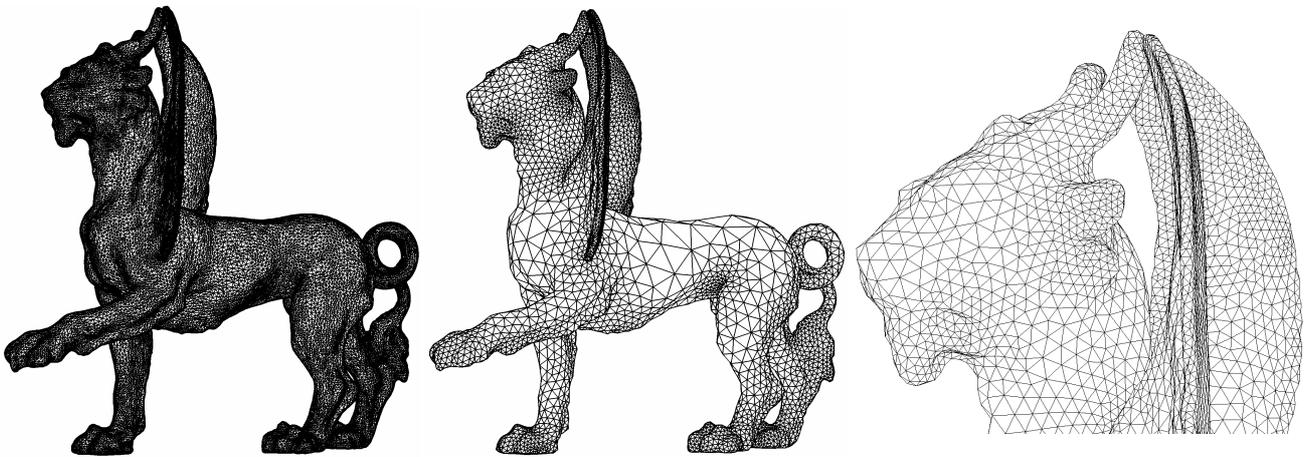


Dino

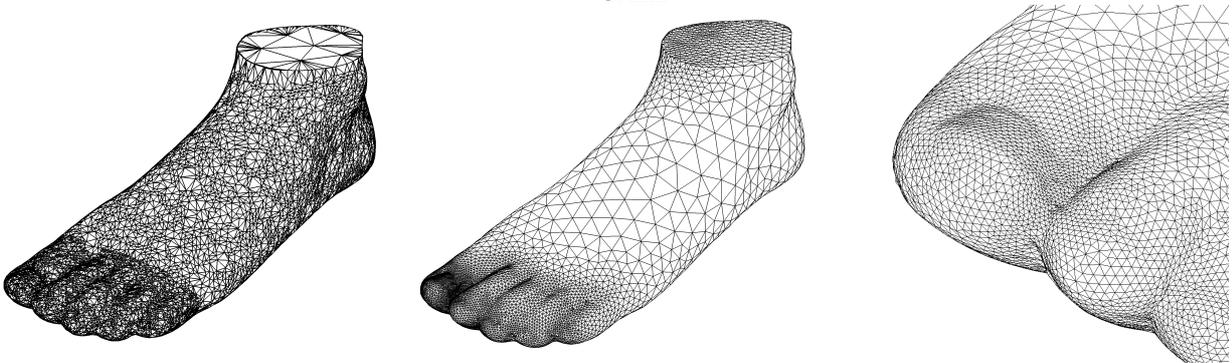
Figure 7: Remeshes. Left: Input mesh. Middle: Remesh. Right: Zoom on remesh (except the Venus model which has a different contrast).



Camel

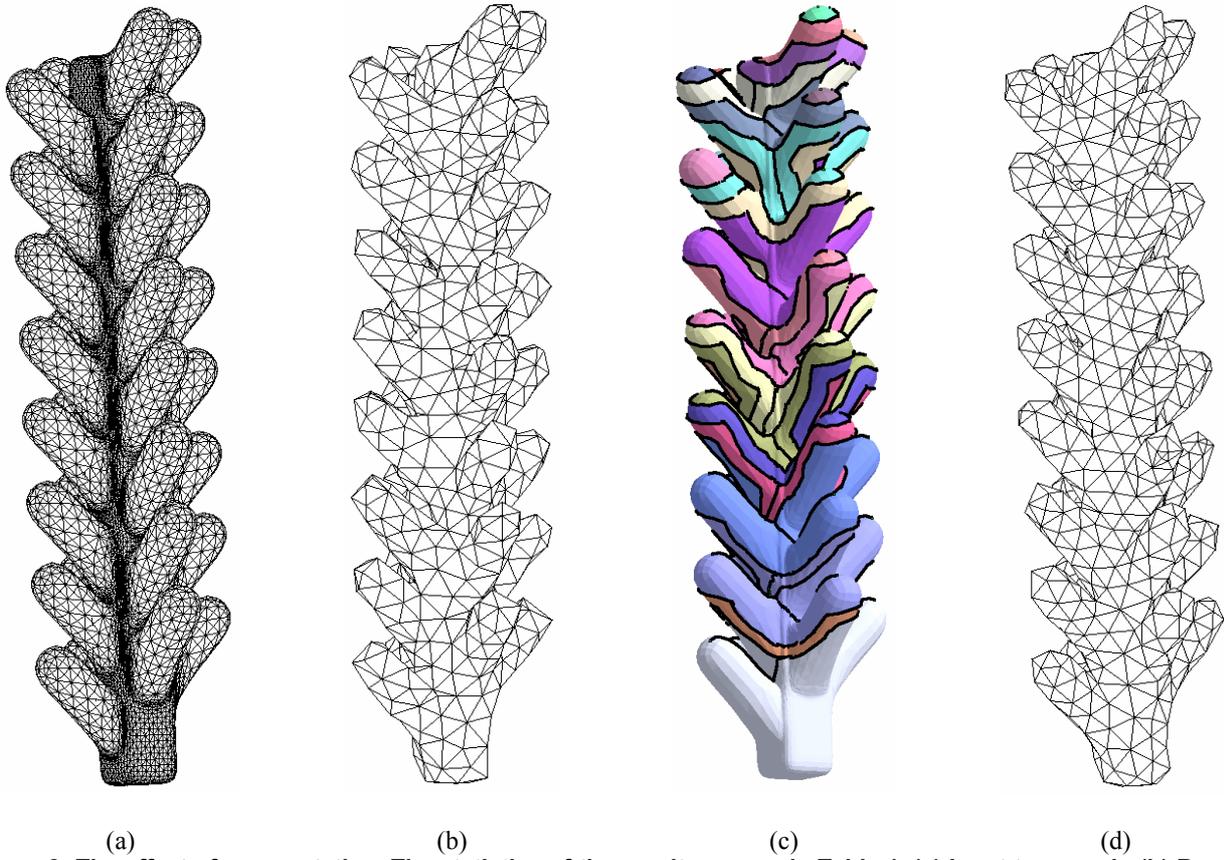


Griffin



Foot

Figure 7 (cont.): Remeshes. Left column is the input mesh. Middle is the remesh and right is a zoom.



**Figure 8: The effect of segmentation. The statistics of the results appear in Table 1. (a) Input tree mesh. (b) Remesh without segmentation. (c) Segments generated by our algorithm. (d) Remesh with segmentation.**