Surface Segmentation Using Geodesic Centroidal Tesselation

Gabriel Peyré
CMAP
Ecole Polytechnique
91128 Palaiseau, France
peyre@cmapx.polytechnique.fr

Laurent Cohen
CEREMADE, UMR CNRS 7534
Universite Paris Dauphine
75775 Paris, France
cohen@ceremade.dauphine.fr

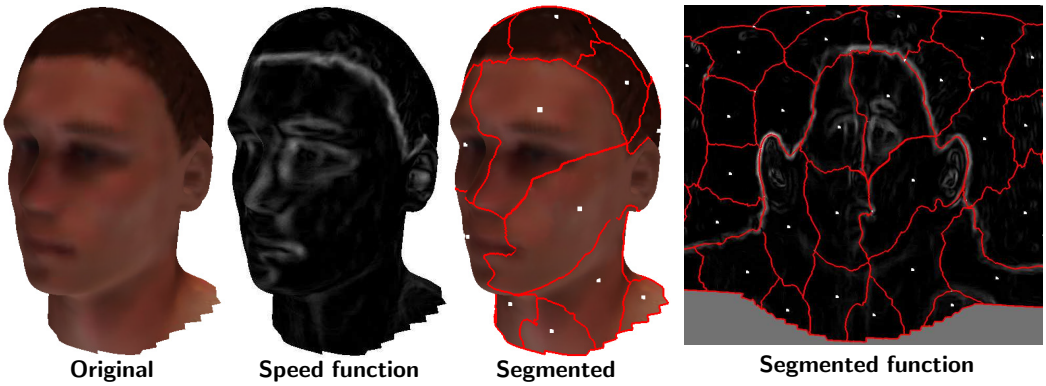

Figure 1. Segmentation of a textured head.

Abstract

In this paper, we solve the problem of mesh partition using intrinsic computations on the 3D surface. The key concept is the notion of centroidal tesselation that is widely used in an eucidan settings. Using the Fast Marching algorithm, we are able to recast this powerful tool in the language of mesh processing. This method naturally fits into a framework for 3D geometry modelling and processing that uses only fast geodesic computations. With the use of classical geodesic-based building blocks, we are able to take into account any available information or requirement such as a 2D texture or the curvature of the surface.

1 Introduction

The applications of 3D geometry processing abound nowadays. They range from finite element computation to computer graphics, including solving all kinds of surface reconstruction problems. The most common representation of 3D objects is the triangle mesh, and the need for fast algorithms to handle this kind of geometry is obvious. Classical 3D triangulated manifold processing methods have several well identified shortcomings: mainly, their high complexity when dealing with large meshes, and their numerical instabilities.

To overcome these difficulties, we propose a geometry processing pipeline that relies on *intrinsic* information of the surface and not on its underlying triangulation. Borrowing from well established ideas in different fields (including image processing and perceptual learning) we are able to process very large meshes efficiently.

1.1 Overview

In section 2 we introduce some concepts we use in our geodesic computations. This includes basic facts about the Fast Marching algorithm, the computations of Voronoi diagrams on surfaces, and a recently proposed greedy algorithm for manifold sampling.

Then, in section 3, we will introduce the notion of *geodesic centroidal tessellation* in order to compute a segmentation of the manifold. There, a fast algorithm based on a geodesic gradient descent is derived, and we will explain how to take into account special features of the mesh, or user defined constraints.

In the conclusion, we will apply this partioning algorithm to texture each patch of the mesh. We will then give a complete study of the timings of each part of our algorithm, including a comparison with classical methods.

1.2 Related Work

Geodesic Computations. Distances computation on manifolds is a complex topic, and a lot of algorithms have been proposed such as *Chen* and *Han* shortest path method [2] which is of quadratic complexity. *Kimmel* and *Sethian*'s *Fast Marching* algorithm [15] allows finding numerically the geodesic distance from a given point on the manifold, in $O(n\log(n))$ in the number of vertices. They deduce minimal geodesics between two given points. Some direct applications of geodesic computations on manifolds have been proposed, such as in [16], which applies the Fast Marching algorithm to obtain *Voronoi diagram* and *offset curves* on a manifold.

Mesh Segmentation. The segmentation of a complex surface is the first step for constructing a parameterization of a whole mesh. In [18] such a segmentation is performed to build an atlas for mesh texturing. In [31] the segmentation borrows ideas from Morse theory. The work of [13] decomposes the object into meaningful components using fuzzy clustering. The importance of texture seams reduction is studied in [26]. The geometry images paradigm [11] uses a single chart to parametrize the mesh on a 2D image. In [24], the authors also use a segmentation procedure to build geometry images for complex meshes. This segmentation is computed using a form of combinatorial Lloyd iteration. In section 3 we will describe a more powerful approach that uses real geodesic computations. The main issue of these techniques is the discontinuities of the parameterization along patch boundaries. These discontinuities can be nearly removed as explained in [14] at the cost of additional complexity.

2 Geodesic-Based Building Blocks

2.1 Fast Marching Algorithm

The classical Fast Marching algorithm is presented in [25], and a similar algorithm was also proposed in [29]. This algorithm is used intensively in computer vision, for instance it has been applied to solve global minimization problems for deformable models [5].

This algorithm is formulated as follows. Suppose we are given a metric $P(s)\mathrm{d}s$ on some manifold $\mathscr{S}$ such that $P > 0$. If we have two points $x_0, x_1 \in \mathscr{S}$, the weighted geodesic distance between x_0 and x_1 is defined as

$$d(x_0,x_1) \stackrel{\text{def.}}{=} \min_{\gamma} \left(\int_0^1 \|\gamma'(t)\| P(\gamma(t)) \mathrm{d}t \right), \qquad (1)$$

where γ is a piecewise regular curve with $\gamma(0) = x_0$ and $\gamma(1) = x_1$. When $P = 1$, the integral in (1) corresponds to the length of the curve γ and d is the classical geodesic distance. To compute the distance function $U(x) \stackrel{\text{def.}}{=} d(x_0,x)$ with an accurate and fast algorithm, this minimization can be reformulated as follows. The level set curve $\mathscr{C}_t \stackrel{\text{def.}}{=} \{x \setminus U(x) = t\}$ propagates following the evolution equation $\frac{\partial \mathscr{C}_t}{\partial t}(x) = \frac{1}{P(x)}\overrightarrow{n_x}$, where $\overrightarrow{n_x}$ is the exterior unit vector normal to the curve at x, and the function U satisfies the nonlinear *Eikonal* equation:

Figure 2. Front Propagation on the David, level sets of the distance function and geodesic path.

$$\|\nabla U(x)\| = P(x). \qquad (2)$$

The function $F = 1/P > 0$ can be interpreted as the propagation speed of the front $\mathscr{C}_t$.

The Fast Marching algorithm on an orthogonal grid makes use of an upwind finite difference scheme to compute the value u of U at a given point $x_{i,j}$ of a grid:

$$\begin{aligned} &\max(u-U(x_{i-1,j}),u-U(x_{i+1,j}),0)^2 \\ &+\max(u-U(x_{i,j-1}),u-U(x_{i,j+1}),0)^2 = h^2P(x_{i,j})^2. \end{aligned}$$

This is a second order equation that is solved as detailed for example in [4]. An optimal ordering of the grid points is chosen so that the whole computation only takes $O(N\log(N))$, where N is the number of points.

In [15], a generalization to an arbitrary triangulation is proposed. This allows performing front propagations on a triangulated manifold, and computing geodesic distances with a fast and accurate algorithm. The only issue arises when the triangulation contains obtuse angles. The numerical scheme presented above is not monotone anymore, which can lead to numerical instabilities. To solve this problem, we follow [15] who proposes to "unfold" the triangles in a zone where we are sure that the update step will work [15]. To get more accurate geodesic distance on meshes of bad quality, one can use higher order approximations, e.g. [19], which can be extended to triangulations using a local unfolding of each 1-ring.

Figure 2 shows the propagation of a front and the calculation of a geodesic path computed using a gradient descent of the distance function.

We should emphasize the fact that the Fast Marching computations are consistent with the continuous setting, which is not the case for traditional graph-based methods. When the triangulation becomes denser, geodesics

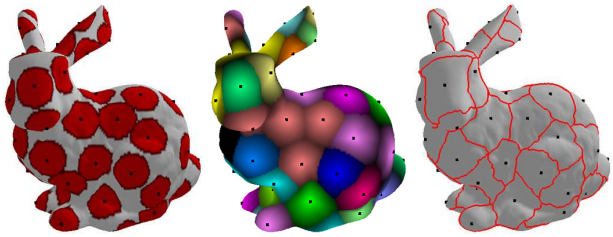

Figure 3. Progression of the fronts, Voronoi diagram, and resulting tessellation.

will converge to the exact geodesics of the underlying continuous manifold. For that reason, we will work with this continuous setting in mind, almost without any reference to the underlying triangulation that supports the computations.

2.2 Extraction of Voronoi Regions

It is possible to start several fronts from points $\{x_1,\dots,x_n\}$ and make them evolve together, as shown on figure 3. The areas with various colours define the *Voronoi diagram* of the starting points, namely the tessellation into the regions, for $i \in \{1,\dots,n\}$

$$V_i \stackrel{\text{def.}}{=} \{x \in \mathscr{S} \setminus \forall j \neq i, \quad d(x,x_j) > d(x,x_i)\}.$$

To accurately compute the boundaries of the Voronoi regions, we allow an overlap of the front on one vertex. Suppose a front a arrives at a vertex v_1 with time arrival t_1^a and another front b arrives at a vertex v_2 (connected to v_1) with time t_2^b. Allowing an overlap of the fronts, we record the time arrival t_2^a of a at v_2, and t_1^b of b at v_1. Then the two fronts collapse at $(1-\lambda)v_1+\lambda v_2$ where $\lambda = \frac{d_2^a - d_1^a + d_1^b - d_2^b}{d_1^b - d_1^a}$.

2.3 A Greedy Algorithm for Sampling a Manifold

A new method for sampling a 3D mesh was recently proposed in [21] that follows a farthest point strategy based on the weighted distance obtained through Fast Marching on the initial triangulation. This is related to the method introduce in [4]. A similar approach was proposed independently and simultaneously in [20]. It follows the *farthest point* strategy, introduced with success for image processing in [9] and related to the remeshing procedure of [3]. This greedy solution for sampling has been used with success in other fields such as computer vision (component grouping, [4]), halftoning (void-and-cluster, [30]) and remeshing (Delaunay refinement, [23]).

This approach iteratively adds new vertices based on the geodesic distance on the surface. Figure 4 shows the first steps of our algorithm on a square. The result of the algorithm gives a set of vertices uniformly distributed on the surface according to the geodesic distance.

Once we have found enough points, we can link them together to form a *geodesic Delaunay triangulation*. This is done incrementally during the algorithm, and leads to

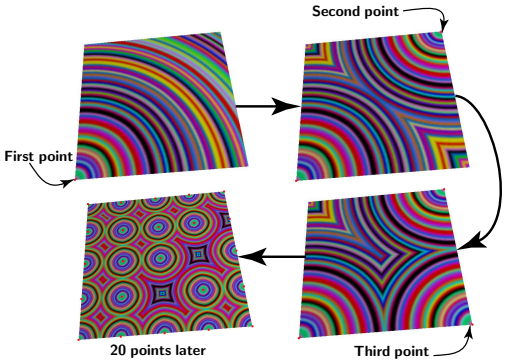

Figure 4. A n overview of the greedy sampling algorithm.

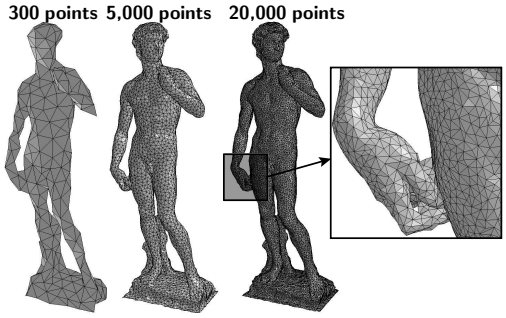

Figure 5. Geodesic remeshing with an increasing number of points.

a powerful remeshing method that can be either uniform (figure 5) or adaptive.

2.4 Adaptive Sampling Using Texture Information

To introduce some adaptivity in the sampling performed by this algorithm, we use a speed function $F = 1/P$ (which is the right hand side of the Eikonal equation (2)) that is not constant across the surface.

When a mesh is obtained from range scanning, a picture I of the model can be mapped onto the 3D mesh. Using a function F of the form $F(x) = \frac{1}{1+\mu|\text{grad}(I(x))|}$, where μ is a user-defined constant, one can refine regions with high variations in intensity. Figure 6 shows such a 3D model. On figure 1, one can see a 3D head remeshed with various μ ranging from $\mu = 0$ (uniform) to $\mu = 20/\max(|\text{grad}(I(x))|)$ (highly adaptive). Local density based on curvature information can be used, see [21].

In the following section, we will make heavy use of adaptive sampling based on the curvature of the surface. Let us denote by $\tau(x) \stackrel{\text{def.}}{=} |\lambda_1| + |\lambda_2|$ the total curvature at a given point x of the surface, where λ_i are the eigenvalues of the second fundamental form. We can introduce two speed functions

$$F_1(x) \stackrel{\text{def.}}{=} 1+\varepsilon\tau(x) \quad \text{and} \quad F_2(x) \stackrel{\text{def.}}{=} \frac{1}{1+\mu\tau(x)}, \qquad (3)$$

where ε and μ are two user-defined parameters. Figure 7 (a) shows that by using function F_1, we avoid putting more

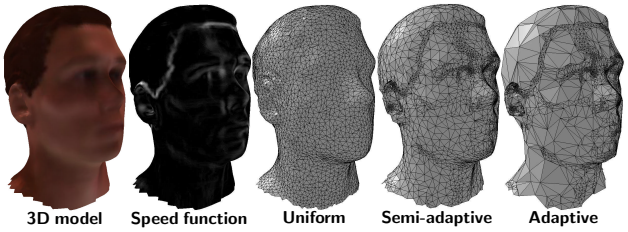

Figure 6. Remeshing of a 3D model using increasing weight for the speed function.

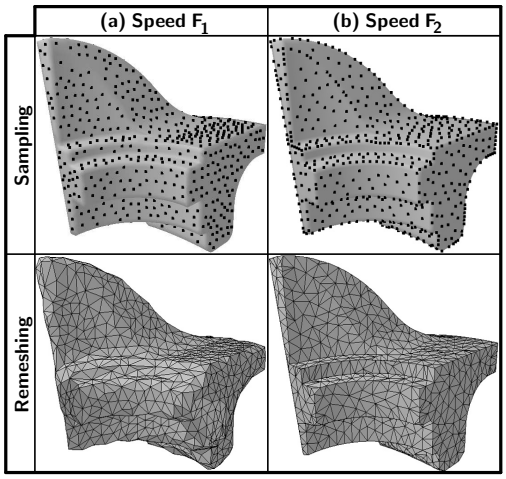

Figure 7. Uniform versus curvature-based sampling and remeshing.

vertices in regions of the surface with high curvature. The speed function F_1 can be interpreted as an "edge repulsive" function. On the other hand, function F_2 could be called "edge attractive" function, since it forces the sampling to put vertices in region with high curvature such as mesh corners and edges. Figure 7 (b) shows that this speed function leads to very good results for the remeshing of a surface with sharp features, which is obviously not the case for the "edge repulsive" speed function (figure 7 (a)).

3 Mesh Segmentation Using Centroidal Tessellation

In this section, we present an automatic mesh segmentation method which is the first part of our surface parameterization pipeline. This method is adapted to the tessellation of a complex manifold in elementary domains topologically equivalent to a disk.

In this method the mesh is cut into regions that best satisfy the following properties:
(C1) Boundaries of the regions agree with sharp features of the surface.
(C2) Regions are as compact as possible (the ratio area/perimeter should be large), enclosing equal areas.
Condition (C1) is natural because the resulting parameterization will have discontinuities on the boundaries of the patches, and it has no consequences if these singularities coincide with singularities of the surface. Condition (C2) prevents degenerated patches and minimizes the length of the singularities which occur along the boundary. These two conditions are contradictory, and a trade-off can be found using an iterative approach.

To achieve this trade-off, we first make an initial choice of regions using our greedy sampling algorithm to satisfy (C1). Then we use these regions to seed a geodesic-based centroidal tesselation to match requirement (C2).

3.1 Initial Choice of Regions

The goal of the algorithm is to build a segmentation $\mathscr{S} = \bigcup_{i=1}^{n} V_i$ of a triangulated manifold $\mathscr{S}$. The V_i will be the Voronoi regions associated with a given set of points $\{v_1, \ldots, v_n\}$.

These points are chosen using the sampling algorithm of section 2.3. To avoid the clustering of base points v_i near regions of high curvature, we choose the "edge repulsive" speed function F_1 of equation (3) for the sampling.

Once the v_i are computed, we can build the Voronoi regions V_i, using the method described in section 2.2. In order to force the boundaries of the regions V_i to follow the discontinuities of the surface, we use the "edge attractive" speed function $F_2(v)$ defined in equation (3) for the Voronoi cells extraction. This will allow us to "freeze" the front in regions with high curvature. This way the resulting Voronoi regions will have boundaries aligned with sharp features of the surface, and condition (C1) will be satisfied.

We also put a topological constraint on the shape of the Voronoi regions so that each cell is a topological disk. In order to maintain this constraint, we keep track of the shape of the front during the Fast Marching procedure. When this front fails to enclose a topological disk (e.g. on a sphere with a single base, the front will collapse onto itself), we stop the process and add a new base point at the location of the failure.

3.2 Centroidal Voronoi Tessellation

The segmentation provided by the previous section satisfies condition (C1), but does not match the requirements of (C2). In order to make the size of regions more homogeneous, we use a clustering technique to refine the Voronoi segmentation. This method is well known in the case of a planar segmentation, and allows the building of *centroidal Voronoi diagrams* [7].

In a formal fashion, the planar segmentation corresponds to minimizing the energy function

$$E(v_i, V_i) \stackrel{\text{def.}}{=} \sum_{i=1}^{n} \int_{V_i} \rho(y) d(y, v_i)^2 \mathrm{d}y, \qquad (4)$$

where ρ represents a density function, and d is a distance on the plane. If one uses $d(x,y) = \|x-y\|$ the Euclidean distance, with $\rho = 1$, and we can prove that:
• The V_i regions must be the Voronoi cells of the v_i points.

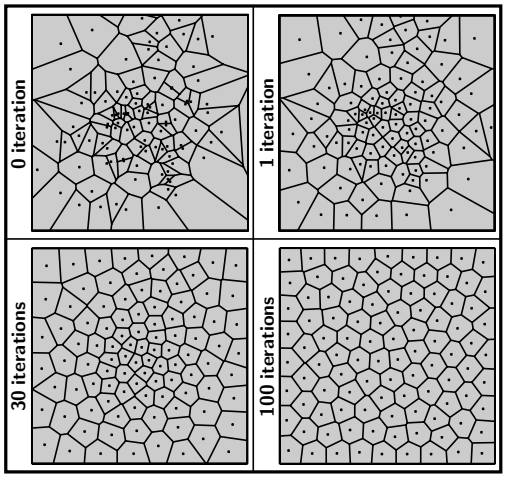

Figure 8. Iterations of Lloyd algorithm on a square.

• The v_i points must be the centers of mass of the V_i cells.

One can show that such a segmentation exists (although it need not be unique), and it is called *a centroidal Voronoi tessellation*. This segmentation can be computed using the *Lloyd* algorithm as follows. We are given an integer n and an initial configuration $\{(v_i, V_i)\}_{i=1}^{n}$. The algorithm iteratively updates the V_i and the v_i using these steps:

(i) Compute the centers of mass v_i^* of V_i.

(ii) Replace v_i with v_i^*.

(iii) Compute the Voronoi regions V_i^*, associated with v_i, for the metric d.

(iv) Replace V_i with V_i^*. If $\forall i,\ V_i = V_i^*$, terminate the algorithm. Otherwise, go back to (i).

Figure 8 shows some iterations of the algorithm, restricted to a square. If we omit the problem on the boundaries, the centroidal segmentation corresponds to the well-known *bee-hive structure* which gives both an optimal compactness and an equal repartition of area among the regions. It is important to note that we have started the algorithm with a random distribution of points. If we had used the output of the previous section (using our greedy sampling method), the number of iterations would have been reduced significantly (approximately 5 iterations instead of 100 for the same result).

The distribution of the points v_i really corresponds to the notion of isotropic sampling. For example, suppose we want to construct a Delaunay triangulation with the size of the triangles controlled by a function $H(x)$. Then, one can show that we may solve this problem asymptotically using a centroidal tessellation by choosing $\rho(x) \propto 1/H(x)^4$ [7].

3.3 Geodesic Lloyd Algorithm

In this paragraph, we explain how to generalize the centroidal Voronoi segmentation to surfaces. To that end we consider, on a triangulated manifold $\mathscr{S}$, the weighted geodesic distance d defined by equation (1).

The centroidal tessellation paradigm has already been recognized as a powerful tool for remeshing a triangulated manifold. In [1], an isotropic remeshing is performed using a Lloyd algorithm in parameter space. We, however, propose to compute the Lloyd iterations directly on the mesh. This allows us to handle meshes with arbitrary genus and without the distortion introduced by the parameterization of the surface. In [28] such a parameterization is avoided by using a local Lloyd scheme. However, due to the scaling induced by the conformal parameterization, this is not equivalent to the computation of a geodesic centroidal tesselation. In [24] a heuristic is used (back-propagation from the boundary) which does not solve the real center of mass equation.

To perform the Lloyd algorithm, we need to define the *intrinsic center of mass* v_i^* of V_i, which is the minimizer of the energy function

$$E_i(w) \stackrel{\text{def.}}{=} \int_{x \in V_i} d(x, w)^2 \mathrm{ds}.$$

where ds is the area element on the surface. On a Riemannian surface, such a minimizer exists and is unique under some mild assumptions on the local curvature, which are given in details in [12]. It is important to note that this intrinsic center of mass is different from the classical Euclidean center of mass (which in general does not even lie on the surface). The computation of the intrinsic center of mass has been introduced in the computer vision community in [27] and is closely related to the generalised procrustes analysis, see [17]. We note, however, that this center of mass has never been explicitly used for mesh processing, for example in [8], the authors uses a projection of the center of mass, and not an intrinsic barycenter.

In order to compute v_i^*, we perform a gradient descent of the energy function using

$$\overrightarrow{\nabla E_i}(w) = \frac{1}{2} \int_{x \in V_i} d(w, x) \overrightarrow{n_w(x)} \, \mathrm{ds},$$

where $\overrightarrow{n_w(x)}$ stands for the unit vector tangent at w to the geodesic path joining x and w. To estimate $\overrightarrow{\nabla E_i}(w)$, we first compute the distance map from w to all points in V_i. Tracing back the geodesics, we are able to compute $\overrightarrow{n_w(x)}$ for each vertex x of V_i. At last, we use a first order numerical integration formula to approximate the integral. We compute the boundary of V_i by interpolation as explained in section 2.2.

Figure 9 shows the results we obtain with a uniform speed function (so d is just the classical geodesic distance). We obtain a kind of *geodesic bee-hive* structure. The initialization of the seed point is done at random. With our greedy sampling algorithm for initialization, only one Lloyd iteration instead of three is enough to get the same centroidal tessellation.

But the most powerful feature of our method is that we can use a varying speed function to define the geodesic distance. In the following, we will use the "edge attractive"

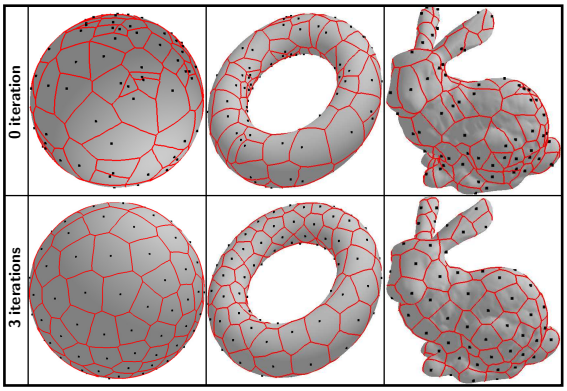

Figure 9. Lloyd iterations on various models.

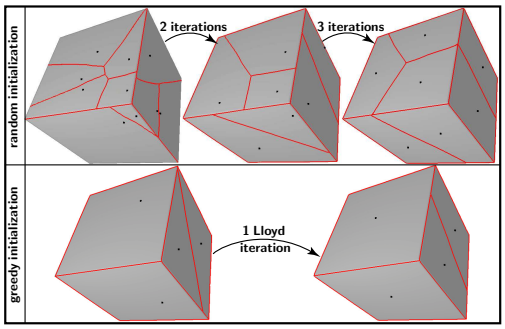

Figure 10. Greedy initialization lead to an optimal segmentation of the cube.

speed function F_2, that we have already used in subsection 3.1 to construct the initial regions. This way we ensure that condition (C1) will still be satisfied while the Lloyd iterations improves the segmentation with respect to condition (C2).

To perform a case study of our method, the figure 10 takes the example of a cube with 10 base points:

- On the top row, we use a *random* initialization. After 2 Lloyd iterations, we get a tessellation that respects the faces of the cube, so it is compliant with condition (C1). This is due to the fact that we use the "edge attractive" speed function. After 3 more iterations, we get a perfectly centered segmentation, and condition (C2) is respected on each face.
- With a random initialization, we do not get an optimal tessellation, since the distribution of base points is as follow: 1 face with 3 points, 2 faces with 2 points, and 3 faces with 1 point. On the bottom row, we use our *greedy* initialization. Thanks to the use of the "edge repulsive" speed function, the seeds are nearly optimally placed (no faces have 3 seeds anymore). We then perform the segmentation and Lloyd iterations using the "edge attractive" function. Only one iteration instead of three is enough to get a perfect segmentation.

Figure 11 shows the segmentation we obtain on more complex models. In the close-up we can see that the cell boundaries try to follow the edges of the mesh whenever it is possible.

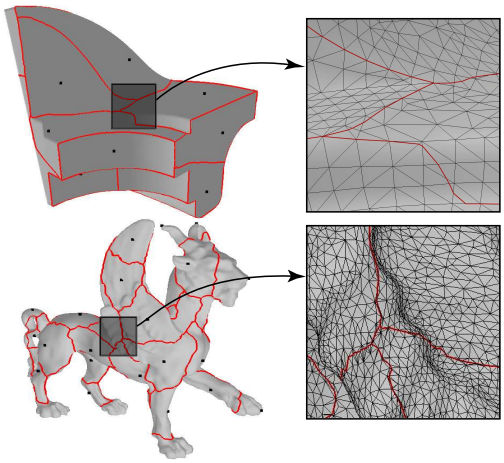

Figure 11. Segmentation of two complex models.

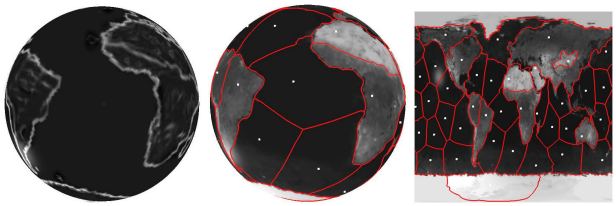

Figure 12. Speed function, segmentation of the earth and resulting regions on the texture map (right).

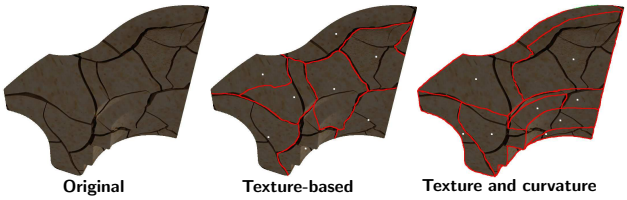

Figure 13. Segmentation using both texture and curvature information.

3.4 Segmentation using texture information

Following the ideas of the section 2.4, we can use a texture function to modulate the speed functions F_1 and F_2. The resulting segmentation will take in account both the texture intensity and the curvature information, according to the user will. Figure 12 shows the segmentation of a textured earth, and the resulting segmentation of the texture.

On such a simple model (a sphere), one could perform a segmentation directly on the 2D image (with periodic boundary conditions), with a special speed function that takes into acount both the gradient of the image and the length distortion due to the cylindrical coordinates. However, this cannot be easily extended to models with complex topology such as the one shown on figure 1. One could also use both the texture and the curvature information, as shown on figure 13.

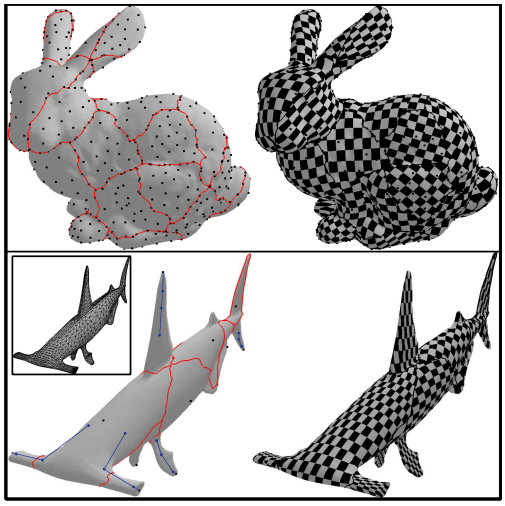

Figure 15. Texturing a bunny and a shark.

4 Results and Discussion

Texturing of a Complex Model.

Once we have performed the segmentation of a 3D mesh, we can flatten each patch and compute texture mapping on this atlas. Although the study of this step is outside the scope of this paper, we note that most of these classical methods come from graph-drawing theory, [10] gives a survey of these techniques. The boundary-free formulation of [6] and [18] (which are mathematically equivalent) is very interesting for texture mapping. We rather choose the boundary-free formulation of [22] that is not conformal but seems to better preserve area across the parameterization. This flattening scheme also uses only geodesic computations and naturally fits into an intrinsic framework for mesh processing together with our segmentation procedure.

On figure 14 one can see the whole pipeline in action. This includes first a centroidal tessellation of the mesh, then the extraction and flattening of each cell, and lastly the texturing of the model.

Figure 15 shows a bunny and a shark that have undergone the same texturing process. On the bunny, left image, all the base points for all patches have been depicted (30 point per patch). On the shark, left image, we have shown in blue the movement of some seed points during the Lloyd iteration process (4 iterations). This clearly shows one of the main features of our segmentation method, which is able to drive its base points to the most relevant areas with respect to the tessellation quality.

To store the texture into memory, we pack all the patches into a single square image, as shown on figure 16. We use a simple strategy to minimize the lost space, as suggested in [18]. It is important to note that the textures we use are mapped directly on the parameter space, without taking care of the junctions between boundaries. Although it is out of the scope of this article, a more complex texturing process could take the output of our pipeline and use it to perform real time painting directly on the mesh.

Discussion. Our tests on various 3D models enlight the strengths of our approach:

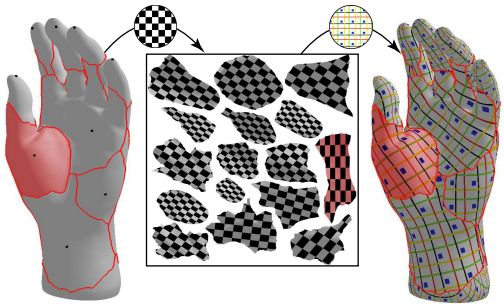

Figure 16. Example of chart packing.

- The boundary of our patches are smooth, almost straight in regions with low curvature variations. We can see that in parameter space, the base domain of a patch located in the middle of the David is a convex polyhedron with straight edges.
- The Voronoi cells of the tessellation are very compact, with almost equal areas. In regions with low curvature variations, this leads to convex base domains, and in regions with sharp features, the boundary of the cells follows the discontinuities.
- Using our greedy initialization, the convergence of the geodesic Lloyd algorithm is very fast (3 iterations).

Our future works include a theoretical study of the convergence of the geodesic Lloyd algorithm. We also would like to analyze experimentally the quality of the parameterization. A good way of evaluating the efficiency of such a scheme is to use its output to perform mesh compression. The mesh atlas provided by our algorithm is an ideal pre-processing step for performing wavelet transform in parameter space, in a fashion similar to [24].

5 Conclusion

We have described a new algorithm to perform the segmentation of a triangulated manifold. The main tool that allows to have a fast algorithm is the fast marching on a triangulated mesh, together with some improvements we added. This segmentation step is the first stage of most mesh processing pipelines. Our contribution includes a geodesic extension of the Lloyd algorithm that is able to construct a geodesic centroidal tessellation. This iterative algorithm takes into account curvature of the surface as well as texture information and is very well suited to building a set of base domains for mesh flattening.

References

[1] P. Alliez, E. C. de Verdière, O. Devillers, and M. Isenburg. Isotropic Surface Remeshing. *International Conference on Shape Modeling and applications*, 2003.

[2] J. Chen and Y. Hahn. Shortest Path on a Polyhedron. *Proc. 6th ACM Sympos. Comput Geom*, pages 360–369, 1990.

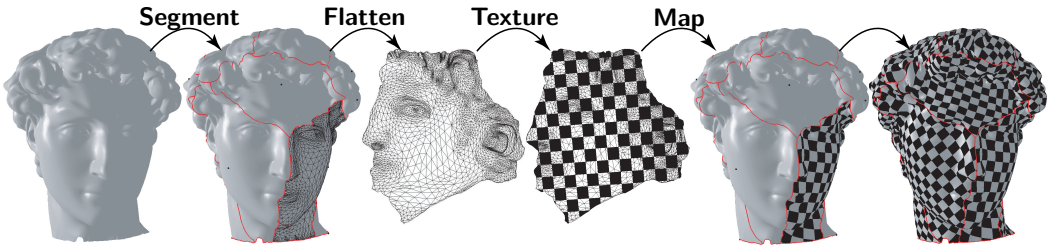

Figure 14. An overview of our pipeline. The mesh is first segmented using a weighted geodesic centroidal tessellation. Each resulting patch is then flattened using the Geodesic LLE procedure. At last, we can perform texture mapping on each base domain.

[3] L. P. Chew. Guaranteed-Quality Mesh Generation for Curved Surfaces. *Proc. of the Ninth Symposium on Computational Geometry*, pages 274–280, 1993.

[4] L. Cohen. Multiple Contour Finding and Perceptual Grouping Using Minimal Paths. *Journal of Mathematical Imaging and Vision*, 14(3):225–236, May 2001.

[5] L. D. Cohen and R. Kimmel. Global Minimum for Active Contour models: A Minimal Path Approach. *International Journal of Computer Vision*, 24(1):57–78, Aug. 1997.

[6] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic Parameterizations of Surface Meshes. *Eurographics conference proceedings*, 21(2):209–218, 2002.

[7] Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi Tessellations: Applications and Algorithms. *SIAM Review*, 41(4):637–676, Dec. 1999.

[8] Q. Du, M. Gunzburger, and L. Ju. A Constrained Centroidal Voronoi Tessellations for Surfaces. *SIAM J. Scientific*, 24(5):1488–1506, 2003.

[9] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi. The Farthest Point Strategy for Progressive Image Sampling. *IEEE Trans. on Image Processing*, 6(9):1305–1315, Sept. 1997.

[10] M. S. Floater, K. Hormann, and M. Reimers. Parameterization of Manifold Triangulations. *Approximation Theory X: Abstract and Classical Analysis*, pages 197–209, 2002.

[11] X. Gu, S. Gortler, and H. Hoppe. Geometry Images. *Proc. ACM SIGGRAPH 2002*, pages 355–361, 2002.

[12] J. Jost. *Riemannian Geometry and Geometric Analysis, 3rd edition*. Springer Verlag, 2001.

[13] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *Proc. SIGGRAPH 2003, ACM Transactions on Graphics*, 22(3):954–961, Jul. 2003.

[14] A. Khodakovsky, N. Litke, and P. Schröder. Globally Smooth Parameterizations with Low Distortion. *ACM Transactions on Graphics. Special issue for SIGGRAPH conference*, pages 350–357, 2003.

[15] R. Kimmel and J. Sethian. Computing Geodesic Paths on Manifolds. *Proc. Natl. Acad. Sci.*, 95(15):8431–8435, 1998.

[16] R. Kimmel and J. A. Sethian. Fast Voronoi Diagrams on Triangulated Surfaces. In *Proc. of the 16th European Workshop on Comp. Geom. (EUROCG-00)*, pages 1–4, 2000.

[17] H. Le. Mean Size-and-shape and Mean Shapes: a Geometric Point of View. *Adv. Appl. Prob.*, 27:44–55, 1995.

[18] B. Levy, S. Petitjean, N. Ray, and J. Maillot. Least Squares Conformal Maps for Automatic Texture Atlas Generation. In ACM, editor, *Special Interest Group on Computer Graphics - SIGGRAPH'02, San-Antonio, Texas, USA*, Jul. 2002.

[19] S. Manay and A. Yezzi. Second-order Models for Computing Distance Transforms. *Proc. IEEE VLSM 2003*, Sept. 2003.

[20] C. Moenning and N. A. Dodgson. Fast Marching Farthest Point Sampling. *Proc. EUROGRAPHICS 2003*, Sept. 2003.

[21] G. Peyré and L. D. Cohen. Geodesic Remeshing Using Front Propagation. *Proc. IEEE VLSM 2003*, Sept. 2003.

[22] G. Peyré and L. D. Cohen. Geodesic Computations for Fast and Accurate Surface Flattening. *Preprint available at* `http://www.cmap.polytechnique.fr/~peyre/upload/flattening/`, May. 2004.

[23] J. Ruppert. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, 18(3):548–585, May. 1995.

[24] P. Sander, Z. Wood, S. Gortler, J. Snyder, and H. Hoppe. Multi-chart Geometry Images. *Proc. Symposium on Geometry Processing 2003*, pages 146–155, 2003.

[25] J. Sethian. *Level Sets Methods and Fast Marching Methods*. Cambridge University Press, 2nd edition, 1999.

[26] A. Sheffer and J. Hart. Seamster: Inconspicuous Low-Distortion Texture Seam Layout. *Proc. IEEE Visualization 2002*, pages 291–298, 2002.

[27] A. Srivastava, W. Mio, X. Liu, and E. Klassen. Geometric Analysis of Constrained Curves for Image Understanding. *Proc. IEEE VLSM 2003*, Sept. 2003.

[28] V. Surazhsky, P. Alliez, and C. Gotsman. Isotropic Remeshing of Surfaces: a Local Parameterization Approach. *Proc. 12th International Meshing Roundtable*, Sept. 2003.

[29] J. Tsitsiklis. Efficient Algorithms for Globally Optimal Trajectories. *IEEE Trans. on Automatic Control*, 1995.

[30] R. Ulichney. The Void-and-Cluster Method for Generating Dither Arrays. *Proc. IS&T Symposium on Electronic Imaging Science & Technology, San Jose, CA*, 1913(9):332–343, Feb. 1993.

[31] E. Zhang, K. Mischaikow, and G. Turk. Feature-based surface parameterization and texture mapping. *accepted to ACM Transaction on Graphics*, 2004.