Geodesic Remeshing Using Front Propagation

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

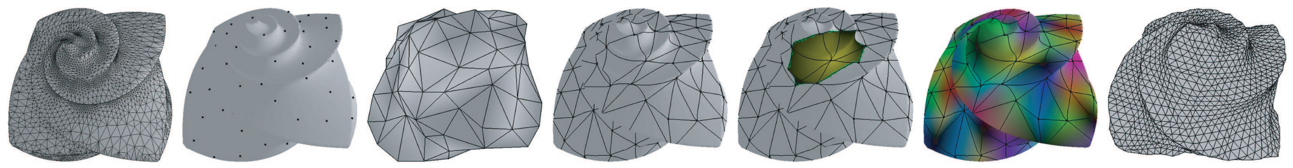

Figure 1. Different steps in mesh parameterization.

Abstract

In this paper, we present a method for remeshing triangulated manifolds by using geodesic path calculations and distance maps. Our work builds on the Fast Marching algorithm, which has been extended to arbitrary meshes by Sethian and Kimmel in [17]. *First, a set of points that are evenly spaced across the surface is automatically found. A geodesic Delaunay triangulation of the set of points is then created, using a Voronoi diagram construction based on Fast Marching. At last, we use the distance information to find a simple parameterization of the manifold. Marching algorithm makes this method computationally inexpensive, and gives very good results. Examples are shown for synthetic and real surfaces.*

1 Introduction

The applications of 3D manifold sampling are nowadays almost everywhere. It ranges from finite element computations to computer graphics, including all kinds of questions related to surface reconstruction techniques used in 3D image segmentation, like deformable models. The most common representation of 3D objects is the triangle mesh, and the need for the construction of a nice triangulation of a given surface is obvious. Thus it is often needed to enhance a given 3D structure (for example obtained from a 3D medical image or artist modeling) using a so-called remeshing algorithm.

The manifold parameterization problem is important as it is the basis for building "good" mesh representations. For example, obtaining a parameterization that minimizes certain distortion measures is the first step used in texture mapping or semi-regular remeshing. Also, once a semi-regular representation of the mesh is built, it is easy to compute wavelet transforms [19, 15] and perform data compression [11].

1.1 Overview

In this paper, we introduce in section 3 an isotropic remeshing algorithm that provides automatically a set of points that are spaced on the surface either uniformly or adaptively with a given density. Thiese points are defined iteratively, adding each point according to geodesic (weighted) distance map to the current set of points. A fast algorithm is proposed that updates by Fast Marching the distance map at each iteration. Basic facts about Fast Marching and geodesic computations are recalled in section 2. At last, in section 4, we propose a novel parameterization method based on geodesic information.

In order to parameterize a manifold, we first calculate a coarse mesh by defining a Voronoi diagram on the surface. The parameterization is then interpolated inside each geodesic triangle, by using appropriate distance data. Figure 1 shows the different steps involved in the parameterization process:

– Original mesh.
– Automatic determination of basis points.
– Determination of associated Delaunay triangulation.
– Calculation of the corresponding geodesic triangles.
– Distance data calculation.
– Parameterization interpolation.
– Semi-regular remeshing.

1.2 Related work

Geodesic calculations on manifolds is a large topic, and numerous algorithms have been proposed. Some of the most interesting exact methods are:

– *Chen and Han*'s shortest path algorithm [3] is of quadratic order, and difficult to implement.

– *Polthier* and *Schmies*'s straightest geodesic algorithm [12] allows solving the problem with initial conditions (a starting point, and direction). This problem is well defined (uniqueness), but cannot be used for remeshing and building surface parameterizations.

– *Sethian* and *Kimmel*'s *Fast Marching* algorithm [17] allows solving the problem with boundary conditions (start and end points), which has not necessarily a unique solution. The algorithm is fast, of order $O(n\log(n))$ in the number of vertices on the manifold.

Some applications of geodesic computations on manifolds have been proposed, such as [13], which applies the Fast Marching algorithm to obtain *Voronoi diagram* and *offset curves* on a manifold. This idea is used in our algorithm to calculate *Delaunay triangulations*.

Remeshing methods roughly fall into two categories:

– *Isotropic remeshing*: a surface density of points is defined, and the algorithm tries to position the new vertices to match this density. For example the algorithm of *Terzopoulos* and *Vasilescu* [20] uses dynamic models to perform the remeshing.

– *Anisotropic remeshing*: the algorithm takes into account the principal directions of the surface to align locally the newly created triangles and/or rectangles. The algorithm proposed in [1] used lines of curvature to build a quad-dominant mesh. Finite element methods make heavy use of such remeshing algorithms [14].

Our method relies on an isotropic distribution of points so that it can be cast into an *Eikonal* problem.

The last point presented in this paper is mesh parameterization. Some of the most effective methods available are:

– *Energy based approach*: first introduced by *Floater* [10], these methods consist in solving a linear system in order to minimize a certain energy measure. Geodesic paths are often chosen as boundaries to partition the the mesh.

– *Geodesic surface flattening*: in [22] *Zigelman et al* use geodesic information. They use *multidimensional scaling* to flatten the manifold while attempting to maintain the accuracy of distance information.

2 Geodesic Calculations

2.1 Fast Marching on an orthogonal grid

The classical Fast Marching algorithm is presented for example in [16, 4] for finding 2D paths or in [8, 6] for 3D extension and improvements. A similar algorithm was also proposed in [21]. Fast Marching enables solving the non-linear *Eikonal* equation:

$$\|\nabla U(x)\| = P(x), \tag{1}$$

where U is the weighted distance function to a given set of points in the plane, and $F = 1/P \geq 0$ is the speed of front propagation. In this section, we consider the orthogonal triangulation as represented on the left of figure 2. The Fast

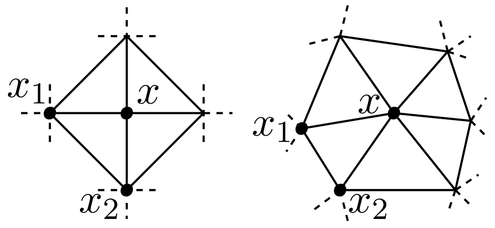

Figure 2. Orthogonal and arbitrary triangulations.

Marching algorithm makes use of an upwind finite differences scheme to compute the value u at a given point $x_{i,j}$ of the grid:

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

This is a second order equation that is solved as detailed in [5]. 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.

2.2 Study over a more general triangulation

Following [17], to generalize the above construction to an arbitrary triangulation, we consider the neighborhood of a point x represented by right side of figure 2. More specifically, we try to calculate a value for $U(x)$ in the triangle $\langle x,\, x_1,\, x_2\rangle$, if possible. Since several triangles around x may yield valid solutions, we consider only the smallest one. The problem that we face in approximating the gradient of U over the triangle $\langle x,\, x_1,\, x_2\rangle$ is that there is no "natural" coordinate system that we can use (unlike the case of an orthogonal grid). In [17] such a system is judiciously chosen, and trigonometric calculations are performed.

In a more general manner, [18] proposes a gradient calculation method in arbitrary dimension. It consists in considering the matrix M of size 2×2 whose lines are constituted of the vectors $\overrightarrow{xx_1}$ and $\overrightarrow{xx_2}$. We can then calculate the directional derivatives of U in x according to the vectors $\overrightarrow{xx_1}$ and $\overrightarrow{xx_2}$ with the first order formula:

$$\nu := \begin{pmatrix} U(x) - U(x_1) \\ U(x) - U(x_2) \end{pmatrix} = U(x)a + b,$$

where $a = \begin{pmatrix}1\\1\end{pmatrix}$ and $b = -\begin{pmatrix}U(x_1)\\U(x_2)\end{pmatrix}$. Now, the derivative in the direction of $\overrightarrow{xx_k}$ is by definition equal to $\langle \nabla U, \overrightarrow{xx_k}\rangle$, hence the formula $\nabla U \simeq M^{-1}\nu$. By plugging this into the Eikonal equation (1), and denoting $Q := (MM^{\mathrm{T}})^{-1}$, we get:

$$(a^{\mathrm{T}}Qa)U(x)^2 - (2a^{\mathrm{T}}Qb)U(x) + b^{\mathrm{T}}Qb = P(x)^2.$$

This is a second order equation, its discriminant being positive by the *Cauchy-Schwartz* formula. The upwind propagation condition we impose, as is the case in the orthogonal grid, is that $-\nabla U(x)$ points towards the interior of the triangle. We can show that if the angle in x is acute, then the scheme is monotone.

Figure 3 shows the propagation of a front and the calculation of a geodesic path (see 2.3).

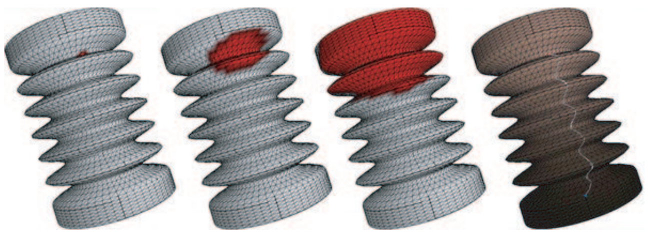

Figure 3. Front Propagation and geodesic path (on the right).

If, on the other hand, the triangulation contains obtuse angles, then the numerical scheme presented above is not monotone anymore, which can lead to numerical instabilities. To solve this problem, we follow *Sethian* who proposes to "unfold" the triangles in a zone where we are sure that the update step will work [17].

2.3 Geodesic Extraction

Once we have calculated the function U, which is the geodesic distance to a vertex v, we need to determine the geodesic path starting from a point v_0 and reaching the point v. To that effect, we look for the parametric curve $C(t)$ that verifies back propagation equation [4]:

$$\begin{cases} \frac{dC}{dt} = -\overrightarrow{\nabla U} \\ C(0) = v_0 \end{cases}.$$

We solve this equation by a numerical method such as *Runge Kutta 4* [9]. To that end, we need to calculate the value of $\overrightarrow{\nabla U}$ at any given point on the surface.

To obtain a continuous variation of $\overrightarrow{\nabla U}$ over the surface, we introduce a novel interpolation scheme that differs from the one used in [17]. First, we calculate the value of the gradient of U at every vertex v on the surface. Then, we linearly interpolate the value of $\overrightarrow{\nabla U}$ on each face. To estimate the value of $\overrightarrow{\nabla U}$ at a vertex v, we consider the 1-ring represented in figure 4 (left), where the v_i's are the neighbors of v. By replacing each angle α_i by $\widetilde{\alpha_i}$ in order to have a sum of 2π, we flatten the neighborhood of v on a plane (figure 4, on the right). We can thus approximate the

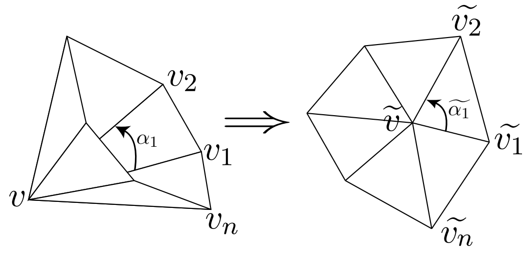

Figure 4. Flattening of a neighborhood.

gradient along each edge (v, v_{i+1}):

$$\nabla U^{(i)} = \frac{U(v_i) - U(v)}{|v_i - v|}.$$

The gradient at v_i satisfies the n equations:

$$\langle \nabla U, e_i\rangle = \nabla U^{(i)},$$

where $e_i := \frac{\widetilde{v}-\widetilde{v_i}}{|\widetilde{v}-\widetilde{v_i}|}$. We solve this over-determined system by least squares, which correspond to solving the 2×2 system:

$$\begin{pmatrix} \sum e_{i0}e_{i0} & \sum e_{i0}e_{i1} \\ \sum e_{i0}e_{i1} & \sum e_{i1}e_{i1} \end{pmatrix} \begin{pmatrix} (\nabla U)_0 \\ (\nabla U)_1 \end{pmatrix} = \begin{pmatrix} \sum e_{i0}\nabla U^{(i)} \\ \sum e_{i1}\nabla U^{(i)} \end{pmatrix},$$

where (e_{i0}, e_{i1}) represent the coordinates of e_i in a local frame of the flattening plane.

Figure 5 shows the geodesic path calculation for different models. We can also start several fronts and make them

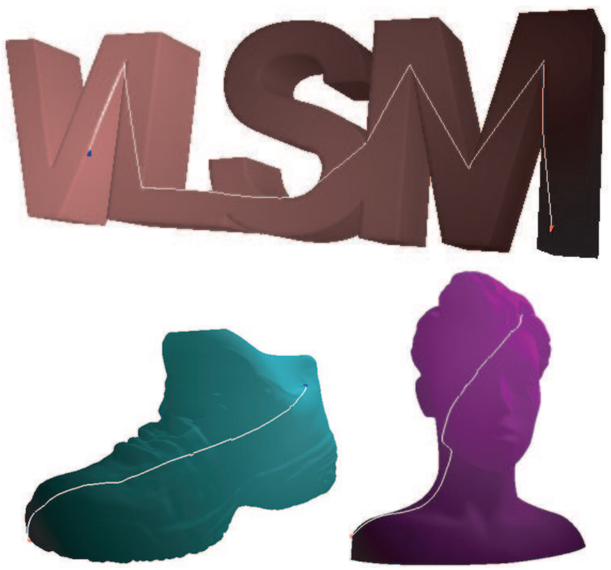

Figure 5. Geodesic paths.

evolve at the same time, as it is shown in figure 6. The different colored regions form a *Voronoi* diagram of the starting points, and the intensity represents the geodesic distance to the closest point.

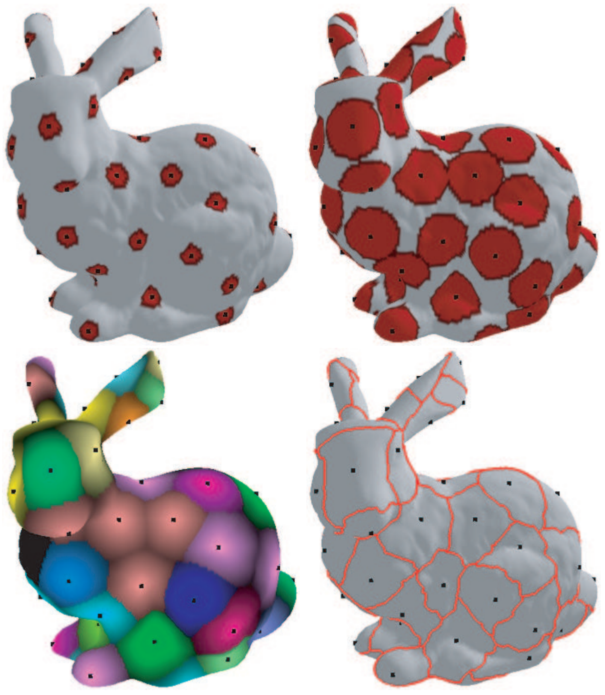

Figure 6. Front propagation (top), geodesic distance map (bottom, left), and Voronoi diagram (bottom, right).

3 Isotropic remeshing of a triangulation

Our approach iteratively adds new vertices based on the geodesic distance on the surface. The result of the algorithm gives a set of vertices uniformly distributed on the surface according to the geodesic distance. Taking into account a local density of vertices will be done in sections 3.3 and 3.4.

This is an extension of an idea of the algorithm of [5] that updated a front propagation from a set of points in order to iteratively find pairs of points to link together. In our context, there are two main differences. The first is that we find points on a triangulated surface instead of an image. The second is that we give a precise approximation of the point where the maximal geodesic distance is reached.

3.1 Iterative choice of basis points

We now describe how to automatically build an evenly spaced set of points on a triangulated surface. A first point x_1 is chosen at random on the mesh and its geodesic distance map U_1 computed by fast marching. A more elaborate choice consists in replacing this random point by the point with maximum distance from it.

Then we assume we have already computed a set of points $S_n = \{x_1, \dots, x_n\}$, together with U_n the geodesic distance map to S_n. To add a new point x_{n+1}, we simply select a point on the manifold that is furthest away from S_n. To compute the new distance map U_{n+1}, we use the fact that $U_{n+1} = \min(U_n, U_{x_{n+1}})$, where we have noted $U_{x_{n+1}}$ the distance map to x_{n+1}. So we simply need to update U_n by starting a front from x_{n+1} (using the Fast Marching algorithm exposed in section 2) and to confine it on the set $\{x \; ; \; U_{x_{n+1}}(x) \leq U_n(x)\}$. This assures that the whole remeshing process roughly takes less than $O(N \log(N)^2)$ operations (which would be the case if we desire as many points as the original mesh has, i.e. N).

At each iteration, the new point x_{n+1} needs not to be a vertex of the original mesh. It can be positioned accurately by interpolating the distance map. To be more precise, we add a new vertex $v = x_{n+1}$ in one of the triangles $\langle v_1, v_2, v_3 \rangle$ around the vertex of greatest distance v_1. We choose v_2 the point around v_1 of greatest geodesic distance, and similarly for v_3 but on the neighbors of the edge $[v_1, v_2]$. It happens most often that the three vertices v_i are reached by three different fronts (the other cases are trivial). We denote t_1, t_2, and t_3 the arrival times of the three fronts at the three vertices. We calculate the point w_3 at which the fronts 1 and 2 meet by linearly interpolating the distance map along edge $[v_1, v_2]$ (here we assume a front speed of 1):

$$w_3 := \lambda v_1 + (1 - \lambda) v_2$$

with:

$$\lambda := \frac{|v_1 - v_2| + t_2 - t_1}{2|v_1 - v_2|}$$

We then choose for v the center of mass of the three intersection points w_i, as shown in figure 7. Note that v is also

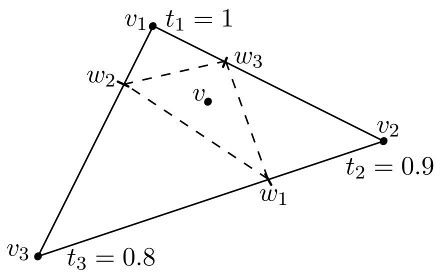

Figure 7. Determination of intersection points of three propagating fronts.

added to the set of vertices of the original mesh.

We choose to stop the algorithm either when the last added point x_{n+1} satisfy $U_n(x_{n+1}) \leq \delta$, where δ is a given threshold, or when a given number of points have been distributed.

Figure 8 shows three stages of the process of inserting points on a square. (in addition to the triangulation obtained, which is explained in the next section). The intensity reflects the geodesic distance to the nearest basis point (white for value 0).

3.2 Calculation of the geodesic triangles

Once we have found the complete set S_{n_0}, we must determine which vertices to link together to obtain our basis

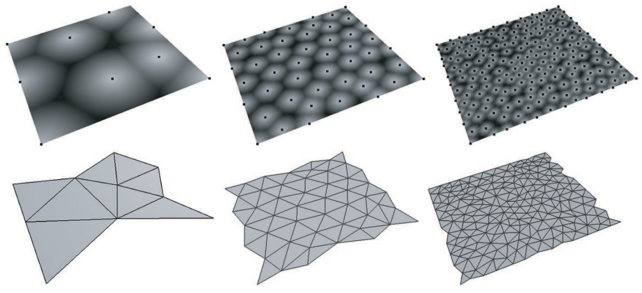

Figure 8. Insertion of points in a square (top), and corresponding triangulation (bottom).

triangulation $\mathcal{T}$ which is built incrementally during the algorithm. To that end, during the point distribution process we keep track of *saddle points*, which are vertices v that satisfy these two criterions:

– When the value of $U(v)$ is set by the Fast Marching algorithm, two fronts coming from different basis points x_i and x_j must meet for the *first* time at v (see [5]).

– Adding edge $[x_i, x_j]$ to the basis triangulation $\mathcal{T}$ must keep the triangulation valid (e.g. the edge must have less than two adjacent faces).

Note that when we update a distance map U_{n+1}, a previously found saddle point v can disappear (if v is reached by the front coming from x_{n+1}), and of course new saddle points can be created.

The set of saddle points tells us which vertices should be linked together to obtain a valid triangulation $\mathcal{T}$. We can also trace on the original mesh the geodesic path corresponding to the edges of $\mathcal{T}$. If we have a saddle point v where two fronts from x_i and x_j meet, we simply perform the gradient descent described in section 2.3 from v in both the direction of x_i and the direction of x_j.

Figure 9 shows progressive remeshing of the bunny.

At last, we can note that some holes might still be found in the resulting triangulation, for instance if there are holes in the original mesh. If we want to avoid such holes (e.g. if we want to use the resulting mesh for parameterizing the manifold), we must fill in the gaps. This can be performed by first unfolding and projecting the hole on a plane, and then computing a constrained Delaunay triangulation. This method is illustrated in figure 10.

3.3 Adaptive remeshing

In the algorithm presented in sections 3.1 and 3.2, the fronts propagate at a constant speed which results in uniformly spaced mesh. Our approach can be extended to work with an arbitrary speed function $F > 0$ for the front propagation, thus computing a geodesic distance weighted by $P = 1/F$. Since vertices are added at maximal values of the geodesic weighted distance, the resulting triangulation will be dense in regions with smaller F, and in regions with higher F the triangulation will be sparse. This is due to the fact that the algorithm distributes points in such a way that their weighted geodesic distances to neighbors are

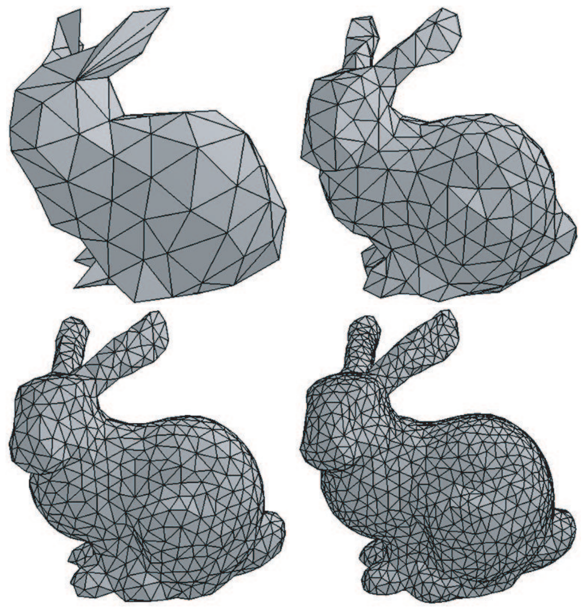

Figure 9. Geodesic remeshing with 100, 300, 800 and 1500 points.

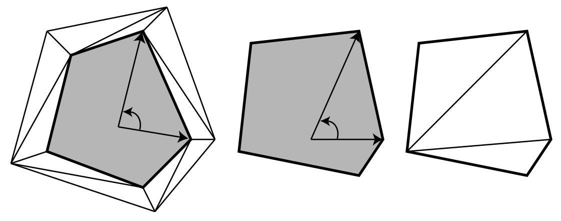

Figure 10. Original hole, flattened hole, and constrained Delaunay triangulation.

almost equal. The geodesic distance to vertices in a region with higher value of P is thus smaller. Function F can reflect the need of the user to refine some specific regions with more vertices.

Figure 11 shows a uniform distribution of points on the head and the distribution of points with a split of the mesh into two regions, one with high F and the other with low F. Similar results are shown for the bunny on the left and middle images of figure 13.

When a mesh is obtained from range scanning, a picture of the model can be mapped onto the 3D mesh. Using a function F that is inversely proportional to the norm of the gradient of the image, the user can refine regions with high variations in intensity. Figure 12 shows such a 3D model.

3.4 Curvature-based remeshing

The local density of vertices can also reflect some geometric proprerties of the surface. The most natural choice is to adapt the mesh in order to be finer in regions where the local curvature is larger. To that end we choose the speed F to be inversely proportional to $\max(|\lambda_1|, |\lambda_2|)$, where λ_i are the eigenvalues of the local curvature tensor. This tensor is computed in a pre-processing step using least square fitting of a quadric. The evaluation of the curvature ten-

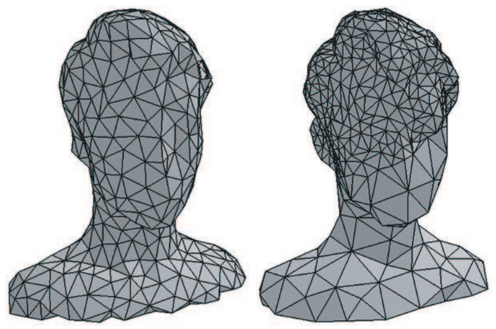

Figure 11. Uniform (left) and user-defined remeshing (right).

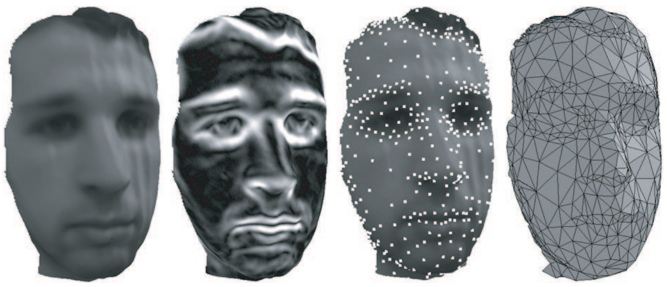

Figure 12. Original model, F function, adaptive distribution of points, and triangulation. Thanks to authors of [22] for providing data.

sor is a vast topic we used a robust construction proposed recently [7]. Figure 13 shows on the right a curvature-based distribution of points on the bunny. Figure 14 shows

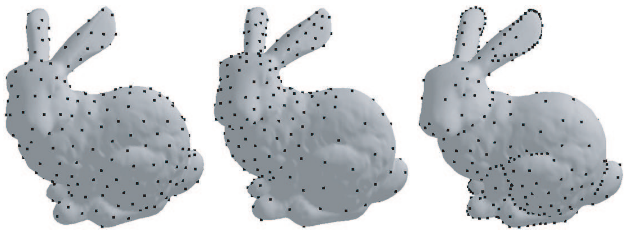

Figure 13. Uniform, user defined, and curvature-based distribution of points.

remeshing of two models with sharp features. Clearly the curvature-based method gives better reconstruction results.

4 Application to mesh parameterization

4.1 Problem statement

A parameterization f of a 3D triangulated manifold $\mathcal{V}$ corresponds to a set of mappings $f_i : \mathcal{U}_i \to \mathcal{V}$, where the $\mathcal{U}_i \subset \mathbb{R}^2$, and the f_i are homeomorphisms whose images cover all of $\mathcal{V}$. To simplify, we will consider a triangulated manifold $\mathcal{V}_0$ that contains very few triangles, but that has the same topology as $\mathcal{V}$. The maps f_i will then be piecewise linear, and the $\mathcal{U}_i$ are the triangles of $\mathcal{V}_0$. Figure 15 shows

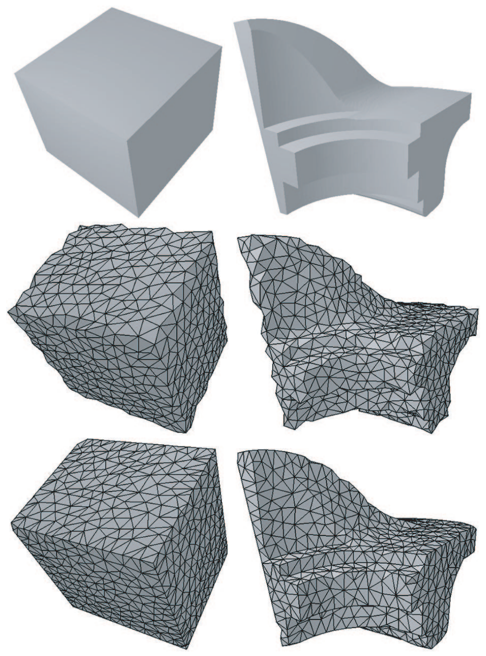

Figure 14. Original model (top), uniform remeshing (middle) and curvature-based remeshing (bottom).

such a parameterization. The functions f_i^{-1} are affine on every small triangle of $\mathcal{V}$, and the image of each triangle $\mathcal{U}_i$ of $\mathcal{V}_0$ by f_i delimits a region of $\mathcal{V}$.

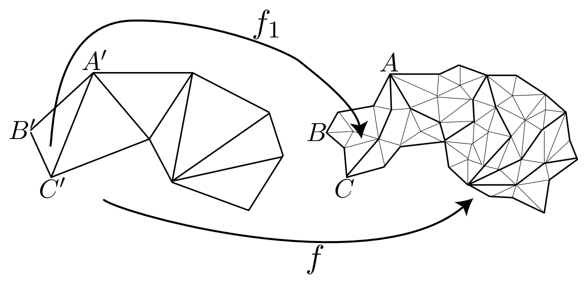

Figure 15. Piecewise linear parameterization

As basis domain for the mapping f we choose the basis mesh constructed with our algorithm in section 3. So the borders of each region $f_i(\mathcal{U}_i)$ are geodesic paths, and these paths need to be added to the original mesh. Figure 16 shows how to perform this inclusion.

Since inside each small triangle of a region $f_i(\mathcal{U}_i)$ the function f_i^{-1} is affine, for every vertex x of $\mathcal{V}$, we only need to know the *barycentric coordinates* of $f_i^{-1}(x)$ in the triangle $\mathcal{U}_i$. The remaining question is how to calculate those barycentric coordinates judiciously.

4.2 Parameterization interpolation

last, we need to calculate the barycentric coordinates (ν_1, ν_2, ν_3) of $f^{-1}(v)$ inside a given triangle $\langle x_1, x_2, x_3 \rangle$, for each vertex v. Note, the points $v_1 = f(x_1)$, $v_2 =$

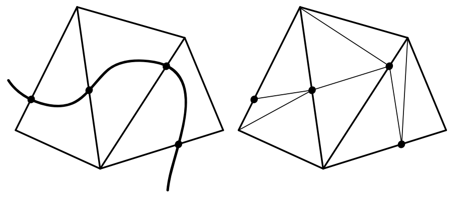

Figure 16. Triangulation before inclusion of the path (left) and after (right).

$f(x_2)$, and $v_3 = f(x_3)$ are the basis points determined in paragraph 3.1. To that end, we will use the geodesic distances (t_1, t_2, t_3) between v and each of the v_i. We will in fact place the point $f^{-1}(v)$ inside the triangle $\langle x_1, x_2, x_3\rangle$ whith conservation of the calculated geodesic distances.

Therefore, knowing the distances (t_1, t_2, t_3) of a point x to the three vertices (x_1, x_2, x_3) of the triangle, we need to calculate the barycentric coordinates (ν_1, ν_2, ν_3) of x in a manner analogous to the case of a planar triangle. We denote (l_1, l_2, l_3) the lengths of the geodesic edge opposite to each point (for example, l_1 is given by the arrival time in v_3 of the front departing from v_2). Now, in a plane, *Heron*'s formula says that the area A_3 of the triangle $\langle x, x_1, x_2\rangle$ is:

$$A_3 = \sqrt{p(p-l_3)(p-t_1)(p-t_2)},$$

with $p := (t_1 + t_2 + l_3)/2$. Figure 17 shows the triangle considered. Similarly, we can calculate A_1 and A_2, which

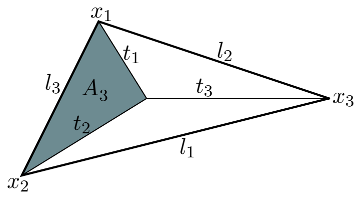

Figure 17. Determination of the barycentric coordinates of a point x

gives us the following barycentric coordinates:

$$\nu_1 = \frac{A_1}{A_1 + A_2 + A_3} \qquad \nu_2 = \frac{A_2}{A_1 + A_2 + A_3}$$
$$\nu_3 = \frac{A_3}{A_1 + A_2 + A_3}$$

Those formulas give us a parameterization of the point x, as long as there is no other point that has the same values for (t_1, t_2, t_3). This condition is met in practice as long as $\mathcal{V}_0$ is topologically equivalent to $\mathcal{V}$, and that the basis meshing is fine enough.

To compute the three distances (t_1, t_2, t_3) for each vertex, we simply start a front and confine it to the geodesic neighborhood of each basis vertex, as depicted on the left of figure 18. Doing this, we are sure that each point will be reached by three fronts and only three. On the right of figure 18, we can see the geodesic parameterization obtained. We simply assign a random color to each vertex, and interpolate the color using the parameterization.

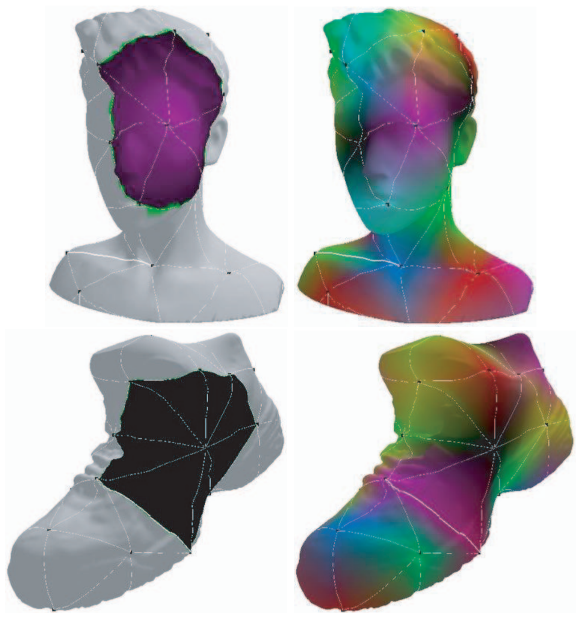

Figure 18. Examples of parameterizations.

4.3 Application to multiresolution constructions

We can sample each basis triangle according to a regular subdivision scheme such as those depicted in figure 19. The resulting triangulation is a so-called *semi-regular* mesh.

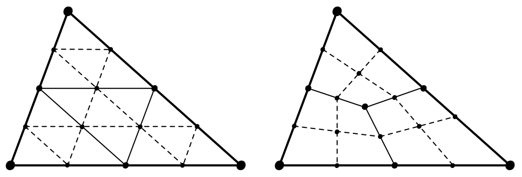

Figure 19. Triangle-based and quad-based subdivision schemes.

These kinds of triangulations are of primary importance, both because the connectivity is known in advance (which is a great advantage for compression), and because the subdivision process gives a natural multiresolution representation of the mesh. Wavelet transforms can then be built using the *lifting scheme* [19]. Wavelet coefficients can be encoded using a *zerotree coder* [11], which gives good distortion results.

Figure 20 show the process of subdivision. The parameterization of the mesh is shown in figure 18 (bottom right).

5 Final remarks and future work

The running times of our algorithm are very reasonable, and it takes about 10 seconds to distribute 500 points on a model of about 5000 vertices. The distribution of more points is almost immediate, thanks to the incremental structure of our method. The coarse mesh construction and the

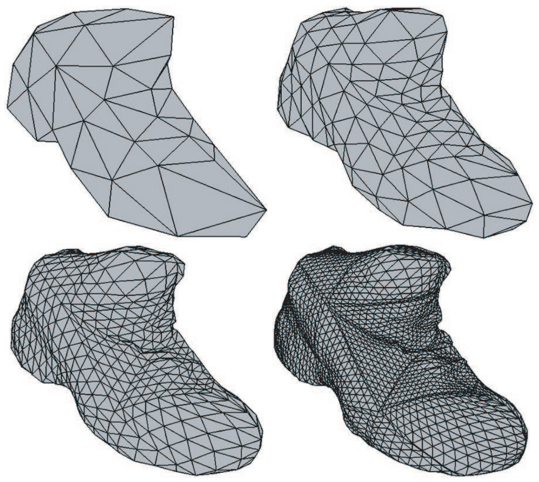

Figure 20. Four steps of regular subdivision.

parameterization steps take about the same time. So our current implementation allows us to process a whole model in less than one minute.

The isotropic remeshing algorithm proposed here gives very good results, similar to those given in [20] for a 2D mesh. The advantage of our method is that it is conceptually simple and very fast. The main component is the Fast Marching algorithm, which is pretty well understood.

The mesh parameterization scheme proposed in this paper gives a smooth map in regions of the manifold that are smooth. Tests and distortion measures remain to be made to compare this method with classical ones, such as *Floater*'s scheme [10].

Our future directions of research include taking into account special features (such as sharp edges) during the remeshing and the parameterization. One could define a linear density of vertices on these 1D features. An interesting question is finding theoretical bounds for the parameterization distortion in case of smooth surfaces.

6 Conclusion

We have described an iterative algorithm for distributing a set of points on a surface according to a local density function. This algorithm is fast and efficient, and provides a simple way to automatically find a geodesic Delaunay triangulation of the original surface. We have applyed this construction to build a coarse basis domain for a parameterization of the triangulated manifold. This method provides a smooth mapping from the basis mesh to the original triangulation, allowing multiresolution constructions and wavelet compression.

References

[1] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Levy, and M. Desbrun. Anisotropic polygonal remeshing. *ACM Transactions on Graphics*, 2003.

[2] M. Bertalmio, G. Sapiro, L-T. Cheng, and S. Osher. Variational problems and PDE's on implicit surfaces. *Proc. IEEE VLSM 2001*, pages 186–193, 2001.

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

[4] Laurent 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.

[5] L.D. Cohen. Multiple contour finding and perceptual grouping using minimal paths. *Journal of Mathematical Imaging and Vision*, 14(3), 2001. Presented at VLSM01.

[6] L.D. Cohen and T. Deschamps. Grouping connected components using minimal path techniques. Application to reconstruction of vessels in 2D and 3D images. In *Proc. of IEEE CVPR'01*, Hawai, 2001.

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

[8] T. Deschamps and L.D. Cohen. Fast extraction of minimal paths in 3D images and applications to virtual endoscopy. *Medical Image Analysis*, 5(4), December 2001.

[9] W.H. Press et Al. *Numerical Recipes in C : the art of computer programming*. Cambridge University Press, 1988.

[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] I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proc.*, Ann. Conf. Series, pages 95–102, 2000.

[12] K. and M. Schmies. Straightest geodesics on polyhedral surfaces. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization*, pages 135–150. Springer Verlag, 1998.

[13] 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.

[14] G. Kunert. Towards anisotropic mesh construction and error estimation in the finite element method. *Numerical Methods in PDE*, 18:625–648, 2002.

[15] P. Schröder and W. Sweldens. Spherical wavelets: efficiently representing functions on the sphere. *Computer Graphics*, 29(Ann. Conf. Series):161–172, 1995.

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

[17] J.A. Sethian and R. Kimmel. Computing geodesic paths on manifolds. *Proc. Natl. Acad. Sci.*, 95(15):8431–8435, 1998.

[18] J.A. Sethian and A. Vladimirsky. Fast methods for the Eikonal and related Hamilton-Jacobi equations on unstructured meshes. *Proc. Natl. Acad. Sci. USA*, 97(11):5699–5703, 2000.

[19] W. Sweldens. The Lifting Scheme: A construction of second generation wavelets. *SIAM Journal on Mathematical Analysis*, 29(2):511–546, 1998.

[20] D. Terzopoulos and M. Vasilescu. Adaptive meshes and shells: Irregular triangulation, discontinuities, and hierarchical subdivision. In *Proc. IEEE CVPR '92*, pages 829–832, Champaign, Illinois, 1992.

[21] J. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Trans. on Automatic Control*, 1995.

[22] G. Zigelman, R. Kimmel, and N. Kiryati. Texture mapping using surface flattening via multi-dimensional scaling. *IEEE Trans. on Visualization and Computer Graphics*, 8(1):198–207, 2002.