Submitted to COMPUTER GRAPHICS Forum. The definitive version is available at onlinelibrary.wiley.com.

Geodesic Polar Coordinates on Polygonal Meshes

Eivind Lyche Melvær[1,2] and Martin Reimers[1,2]

[1]Centre of Mathematics for Applications, University of Oslo, Norway
[2]Department of Informatics, University of Oslo, Norway
eivindlm@ifi.uio.no, martinre@ifi.uio.no

Abstract

Geodesic Polar Coordinates (GPCs) on a smooth surface S are local surface coordinates that relates a surface point to a planar parameter point by the length and direction of a corresponding geodesic curve on S. They are intrinsic to the surface and represent a natural local parameterization with useful properties. We present a simple and efficient algorithm to approximate GPCs on both triangle and general polygonal meshes. Our approach, named DGPC, is based on extending an existing algorithm for computing geodesic distance. We compare our approach with previous methods, both with respect to efficiency, accuracy and visual qualities when used for local mesh texturing. As a further application we show how the resulting coordinates can be used for vector space methods like local remeshing at interactive frame-rates even for large meshes.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Geometric algorithms, languages, and systems I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Colour, shading, shadowing, and texture

Keywords: parameterization, geodesic coordinates, decal texturing, discrete exponential map, remeshing

1. Introduction

Local surface parameterization has many applications, both for smooth surfaces and for meshes. Examples are surface approximation [WW94], generalization of vector space algorithms to surfaces [WD05, Pen06, SSCO06, SM09], numerical optimization on surfaces [Pen06], manifold learning [BWHK05, LZ08], 3*D* face recognition [MMS07] and computer graphics applications such as texturing [SGW06], as in Figure 1.

The exponential map (exp-map) is a well known local surface parameterization from differential geometry. It relates points in a neighbourhood of a base point **s** to points in the tangent plane at **s**, so that length and direction of geodesics are preserved, see Figure 2 for an illustration. The exp-map is a bijection in a region around the base point and has many useful properties as a local surface parameterization within its region of injectivity. One can consider it a generalization of arclength parameterization for curves. It is a radial isometry, that is preserves distance to **s**, and has low distor-

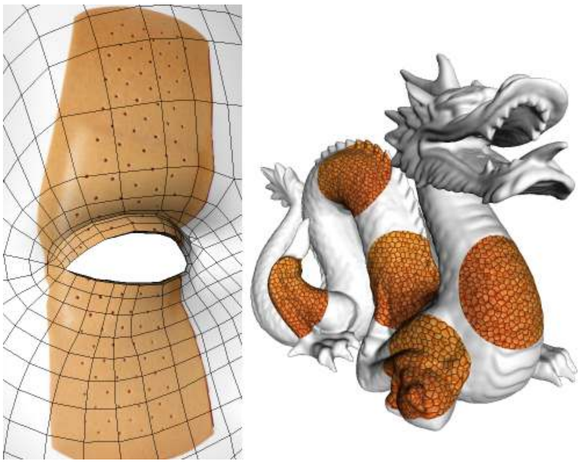


Figure 1: *Our approach for local surface parameterization admits high-quality interactive local texturing of large polygonal meshes. The eye is part of a polygonal mesh with* 3224 *vertices and an interior hole, while the dragon is a triangle mesh with* 437*k vertices.*

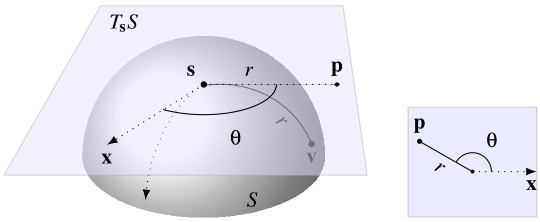


Figure 2: *Geodesic Polar Coordinates with respect to a base point* $\mathbf{s}$ *on a smooth surface. A point* $\mathbf{v} \in S$ *is parameterized by a polar coordinate* $\mathbf{p}$ *in* $T_{\mathbf{s}}S$*, or equivalently in* $\mathbb{R}^2$ *(right), corresponding to the length and direction of a shortest geodesic curve connecting* $\mathbf{v}$ *to* $\mathbf{s}$*.*

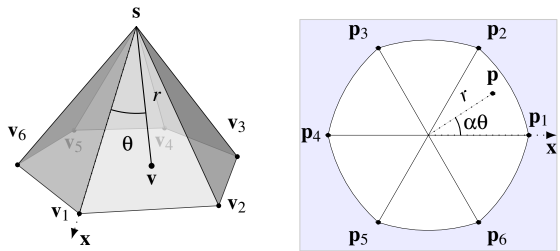


Figure 3: *Geodesic Polar Coordinates* $\mathbf{p} = P_{\mathbf{s}}(\mathbf{v})$ *on a triangle mesh differs from the smooth case by an angular scaling factor* $\alpha = \sum \theta_i / 2\pi$ *for triangle angles* θ_i*.*

tion near the base point. The exp-map can be used to define local coordinate systems in a region around a base point, by defining some coordinate system for the tangent plane $T_{\mathbf{s}}S$. A natural choice is to use polar coordinates. Hence we map each surface point $\mathbf{v}$ to a polar coordinate (r, θ), relative to some chosen reference tangent direction as in Figure 2. The radial distance r is the geodesic distance from $\mathbf{v}$ to $\mathbf{s}$, and the polar angle θ is the angle at $\mathbf{s}$ of the corresponding (shortest) geodesic curve connecting $\mathbf{v}$ and $\mathbf{s}$. The resulting map is called a Geodesic Polar Map (GPM) and the coordinates Geodesic Polar Coordinates (GPCs). Our goal is to efficiently approximate GPCs for 3*D* polygonal meshes. We will refer to our approximation as *Discrete GPC*, or *DGPC*.

There are many ways to parameterize a surface mesh, see [FH05, SPR06] for recent surveys. By applying global parameterization methods such as [Flo03, DMA02] to a submesh one are often guaranteed to obtain one-to-one parameterizations, in some cases minimizing some global distortion measure. However, local distortion around a central base point is not easily controlled. Moreover, such methods often require solving systems of equations or CPU-intensive optimization techniques and finding a suitable neighbourhood of a given point can be a nontrivial task.

There exist several methods for constructing local surface parameterizations. In [SS09] the Fast Marching Method (FMM) [KS98] is applied to construct a patch around a base point, and then Mean Value Coordinates [Flo03] are used to parameterize the interior of the patch. In [SGW09] vertex tracing is used to lay out a grid on the surface, and a mass-spring network is used to relax the grid afterwards. Whereas both these approaches are inspired by geodesic distance, neither of the resulting constructions are parameterized according to geodesic distance. The FMM is also used in [MMS07] to obtain geodesic circles around a base point. These are in turn parameterized and aligned to form local surface coordinates at non-interactive rates. In [SGW06] Dijkstra's algorithm is used to approximate geodesic curves, which are projected to local tangent planes to obtain geodesic coordinates that can be used for local texturing.

1.1. Contributions

In this paper we approximate GPMs on polygonal meshes by extending the local polar map presented in [WW94] to a larger region of the mesh. We extend an existing method for approximating geodesic distance on a triangle mesh to also compute polar angles. We have selected the method proposed in [Rei04] since it yields accurate approximations near a base point. Other methods, such as the popular and efficient FMM [KS98], could in principle be extended in a similar way by the extension velocity method [AS99]. However, we found that methods based on linear interpolation tend to give inadequate results for GPCs. Our approach on the other hand is efficient and accurate, and yields visually pleasing results. We also propose a simple approach to approximate geodesic distance on a general polygonal mesh without explicitly triangulating the mesh. This has to our knowledge not been done before.

We will start by introducing discrete GPMs in Section 2. In Section 3 we describe the approximation method presented in [Rei04] for computing geodesic distance on a triangle mesh, and show how it can be extended to also compute the angular part of the GPCs. In Section 4 we describe our algorithm which is also directly applicable for general polygonal meshes. In Section 5 we present numerical examples and applications, before we summarize and conclude in Section 6.

2. GPMs on Triangle Meshes

We first introduce our notation and define the problem we are considering. Let $\mathcal{T}$ be a triangle mesh with vertices $V = \{\mathbf{v}_i\}_{i=1}^{n}$ in $\mathbb{R}^3$ and triangles $T_{ijk} = [\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k]$. We consider an arbitrary base point $\mathbf{s} \in \mathcal{T}$ and a tangent vector $\mathbf{x}$ which we will call the base direction, see Figure 3. The Geodesic Polar Map $P_{\mathbf{s}} : \mathcal{T} \to \mathbb{R}^2$ with respect to $\mathbf{s}$ and $\mathbf{x}$

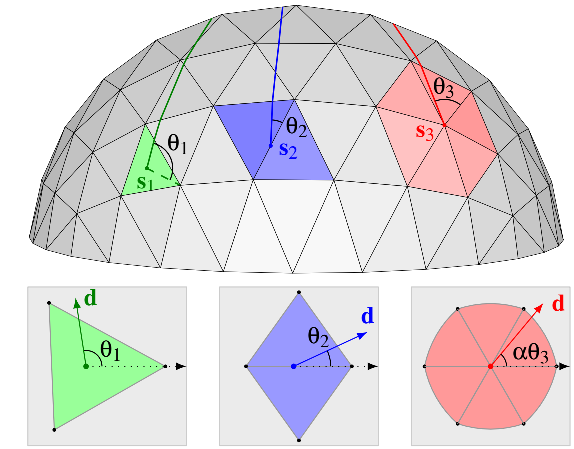


Figure 4: *Geodesic Polar Maps (GPMs) on the three types of base points* **s***. Tangent vectors correspond to geodesics.*

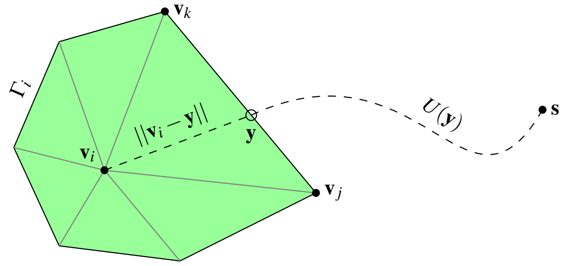


Figure 5: *The distance in* $\mathbf{v}_i$ *is required to be locally optimal, and satisfy* $U_i = \min_{\mathbf{y}\in\Gamma_i}\{U(\mathbf{y}) + \|\mathbf{v}_i - \mathbf{y}\|\}$.

associates each point $\mathbf{v} \in \mathcal{T}$ to a corresponding Geodesic Polar Coordinate $P_{\mathbf{s}}(\mathbf{v}) = \big(r_{\mathbf{s}}(\mathbf{v}), \theta_{\mathbf{s}}(\mathbf{v})\big)$ in $\mathbb{R}^2$. The radius $r_{\mathbf{s}}(\mathbf{v})$ equals the geodesic distance from $\mathbf{v}$ to $\mathbf{s}$, and the polar angle $\theta_{\mathbf{s}}(\mathbf{v})$ reflects the direction of the geodesic curve from $\mathbf{v}$ to $\mathbf{s}$ relative to the base direction $\mathbf{x}$. Thus $r_{\mathbf{s}}(\mathbf{s}) = 0$ and $\theta_{\mathbf{s}}(\mathbf{x}) = 0$.

For a smooth surface, the GPM is the composition of polar coordinates and the inverse exp-map, and so the relative direction θ of a geodesic at $\mathbf{s}$ is well defined. On a triangle mesh, however, tangent direction is not a clear concept. We therefore define the local GPM for the immediate neighbourhood of the three possible types of base points, see Figure 4:

- If $\mathbf{s}$ is in the interior of a triangle, the local GPM is defined as the isometric mapping to the plane.
- If $\mathbf{s}$ is in the interior of an edge we map the containing triangles to $\mathbb{R}^2$ by the so-called isometric hinge-map, i.e. by triangle flattening.
- If $\mathbf{s}$ is a vertex of $\mathcal{T}$ we use the polar map proposed in [WW94]. Each triangle in the one-disk of $\mathbf{s}$ is mapped so that distance to $\mathbf{s}$ is preserved and angles are scaled uniformly by $\alpha = \sum\theta_i/2\pi$ where the θ_i are the triangle angles at $\mathbf{s}$. More precisely, $r_{\mathbf{s}}(\mathbf{v}) = \|\mathbf{v}-\mathbf{s}\|$ and $|\theta_{\mathbf{s}}(\mathbf{v}) - \theta_{\mathbf{s}}(\mathbf{w})| = \alpha\angle(\mathbf{v}-\mathbf{s}, \mathbf{w}-\mathbf{s})$ for two points $\mathbf{v}$ and $\mathbf{w}$. If $\mathbf{s}$ is a boundary vertex we include the open boundary angle in the definition of α.

The so defined local GPM is a local parameterization which maps a triangle neighbourhood of $\mathbf{s}$ to $\mathbb{R}^2$ by preserving length and direction relative to the neighbouring vertices of $\mathbf{s}$. We extend the local GPM to the whole of $\mathcal{T}$ as follows. Consider a point $\mathbf{v} \in \mathcal{T}$ with a shortest geodesic curve C from $\mathbf{v}$ to $\mathbf{s}$. As usual we define $r_{\mathbf{s}}(\mathbf{v})$ to be the geodesic distance from $\mathbf{v}$ to $\mathbf{s}$. The polar angle is required to be constant on a geodesic curve, hence we set $\theta_{\mathbf{s}}(\mathbf{v}) = \theta_{\mathbf{s}}(\mathbf{y})$ for any point $\mathbf{y} \in C$ in the immediate triangle neighbourhood of $\mathbf{s}$ where $\theta_{\mathbf{s}}$ is already defined.

By the above definition of the GPM on a triangle mesh, a direction in $\mathbb{R}^2$ corresponds to a tangent direction on $\mathcal{T}$ and a vector in $\mathbb{R}^2$ corresponds to a geodesic in $\mathcal{T}$. The GPM is continuous and bijective in a neighbourhood of $\mathbf{s}$ defined by the so-called *injectivity radius*. Outside this region there can be singular points for which there are more than one shortest geodesic to $\mathbf{s}$. Moreover, there could be more than one point mapping to the same parameter point in $\mathbb{R}^2$. Therefore, the GPM is better suited in a local neighbourhood of the base point, than for global parameterization. The GPM is a radial isometry, i.e. preserves distance with respect to $\mathbf{s}$. Angular distortion, however, is unavoidable on a general non-planar mesh.

3. Approximating GPCs

GPCs can in principle be found by computing shortest geodesic curves. However, computing such curves for a whole mesh is inefficient and impractical, as we will show later. Instead we propose to approximate GPCs by extending an algorithm for approximating geodesic distance on a triangle mesh to also compute polar angles. We thus obtain approximate GPCs for the vertices of the mesh.

The method to approximate geodesic distance presented in [Rei04], is based on the dynamic programming method from [GR85] to solve the Eikonal equation on $\mathcal{T}$,

$$||\nabla u(\mathbf{x})|| = 1, \qquad u(\mathbf{s}) = 0. \tag{1}$$

The algorithm is similar to Dijkstra's algorithm, but instead of updating a vertex distance from a neighbour vertex we use the geometric information from a neighbouring triangle. We require that each vertex distance is locally optimal, i.e.

$$U_i = \min_{\mathbf{y}\in\Gamma_i}\{U(\mathbf{y}) + \|\mathbf{v}_i - \mathbf{y}\|\} \tag{2}$$

where $U(\mathbf{y})$ is a local approximation to geodesic distance from $\mathbf{s}$ to a point $\mathbf{y}$ on Γ_i, the boundary of the one-ring around $\mathbf{v}_i$, as illustrated in Figure 5. The approach also results in an

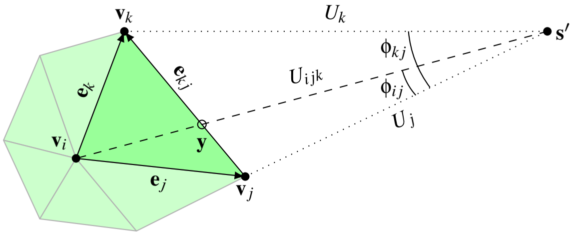


Figure 6: *The Euclidean triangle update: $U_{ijk} = \|\mathbf{v}_i - \mathbf{s}'\|$, where the virtual source $\mathbf{s}'$ is computed using previously computed distances U_j and U_k.*

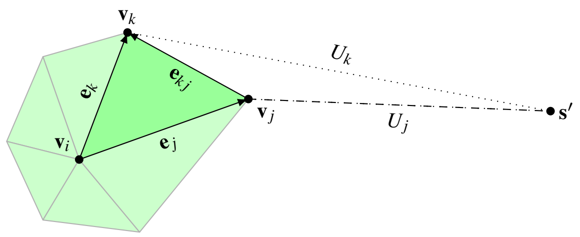


Figure 7: *The Dijkstra update $U_{ijk} = \|\mathbf{e}_j\| + U_j$ is used for a non-convex quadrilateral, here for $x_k < 0$.*

estimate for an optimal geodesic curve locally, given by the line segment $[\mathbf{v}_i, \mathbf{y}]$ for the optimal $\mathbf{y}$ in (2).

The choice of local approximation U in this approach is crucial. An Euclidean scheme was proposed in [Rei04] (see also [NK02]), i.e. one that yields correct distance with respect to a point in the plane. The basic idea can be illustrated by the local construction depicted in Figure 6. We construct a virtual base point $\mathbf{s}'$ with distance U_j from $\mathbf{v}_j$ and distance U_k from $\mathbf{v}_k$, and let $U(\mathbf{y}) = \|\mathbf{y} - \mathbf{s}'\|$. Hence, for a planar mesh where $U_j = \|\mathbf{v}_j - \mathbf{s}\|$ and $U_k = \|\mathbf{v}_k - \mathbf{s}\|$ we get the correct distance $U(\mathbf{y}) = \|\mathbf{y} - \mathbf{s}\|$.

More specifically, consider a regular triangle T_{ijk} as in Figure 6 with previously computed distances U_j, U_k which satisfies the triangle inequality

$$|U_j - U_k| \leq \|\mathbf{v}_j - \mathbf{v}_k\| \leq U_j + U_k. \tag{3}$$

Then one can find a "virtual source" point $\mathbf{s}'$ in the plane spanned by T_{ijk} such that

$$\|\mathbf{v}_j - \mathbf{s}'\| = U_j \qquad \|\mathbf{v}_k - \mathbf{s}'\| = U_k. \tag{4}$$

The distance in $\mathbf{v}_i$ can then be approximated as

$$U_{ijk} = \min_{\mathbf{y} \in [\mathbf{v}_j, \mathbf{v}_k]} \{\|\mathbf{y} - \mathbf{s}'\| + \|\mathbf{v}_i - \mathbf{y}\|\}, \tag{5}$$

which corresponds to the distance between $\mathbf{v}_i$ and $\mathbf{s}'$ in the interior of the planar quadrilateral Q with vertices $\mathbf{v}_i, \mathbf{v}_j, \mathbf{s}'$ and $\mathbf{v}_k$. The so-called Euclidean update from T_{ijk} is consequently given as

$$U_{ijk} = \begin{cases} ||\mathbf{s}' - \mathbf{v}_i|| & \text{if Q is convex;} \\ \min_{\ell=j,k}\{U_\ell + ||\mathbf{v}_\ell - \mathbf{v}_i||\} & \text{otherwise.} \end{cases} \tag{6}$$

Note that the latter case, illustrated in Figure 7, is the single vertex update used in Dijkstra's algorithm. This is also used if U_j and U_k does not satisfy the triangle inequality (3), i.e. when $\mathbf{s}'$ cannot be constructed.

Let us describe the minimization of (5) in detail. The vectors $\mathbf{e}_j := \mathbf{v}_j - \mathbf{v}_i$ and $\mathbf{e}_k := \mathbf{v}_k - \mathbf{v}_i$ are linearly independent and we can express the virtual base point $\mathbf{s}'$ on the form

$$\mathbf{s}' = \mathbf{v}_i + x_j \mathbf{e}_j + x_k \mathbf{e}_k \tag{7}$$

for unique scalars x_j, x_k. The conditions (4) yields two quadratic equations in x_j, x_k,

$$\begin{pmatrix} x_j - 1 \\ x_k \end{pmatrix} M \left(x_j - 1,\ x_k\right) = U_j^2 \tag{8}$$

$$\begin{pmatrix} x_j \\ x_k - 1 \end{pmatrix} M \left(x_j,\ x_k - 1\right) = U_k^2, \tag{9}$$

where M is the 2×2 matrix defined by

$$M := \begin{pmatrix} \mathbf{e}_j \cdot \mathbf{e}_j & \mathbf{e}_j \cdot \mathbf{e}_k \\ \mathbf{e}_j \cdot \mathbf{e}_k & \mathbf{e}_k \cdot \mathbf{e}_k \end{pmatrix}. \tag{10}$$

The solution we are interested in can be expressed as

$$x_j = \frac{A\left(\mathbf{e}_{kj}^2 + U_k^2 - U_j^2\right) + \mathbf{e}_k \cdot \mathbf{e}_{kj} H}{2A\mathbf{e}_{kj}^2}, \tag{11}$$

$$x_k = \frac{A\left(\mathbf{e}_{kj}^2 + U_j^2 - U_k^2\right) - \mathbf{e}_j \cdot \mathbf{e}_{kj} H}{2A\mathbf{e}_{kj}^2},$$

where $A := \|\mathbf{e}_j \times \mathbf{e}_k\|$ and $\mathbf{e}_{kj} := \mathbf{v}_k - \mathbf{v}_j$ and

$$H = \sqrt{\left(\mathbf{e}_{kj}^2 - (U_j - U_k)^2\right)\left((U_j + U_k)^2 - \mathbf{e}_{kj}^2\right)}. \tag{12}$$

The latter can be identified as Herons formula for the area of the triangle $[\mathbf{s}', \mathbf{v}_j, \mathbf{v}_k]$ by its edge lengths.

It is easy to check that $x_j + x_k \geq 1$ and therefore $\mathbf{s}'$ and $\mathbf{v}_i$ lies on opposite sides of the line through $\mathbf{v}_j, \mathbf{v}_k$ as in Figure 6. Moreover, H is a real number if the triangle inequality (3) holds. The quadrilateral Q is convex iff x_j, x_k are both positive. Therefore the Euclidean update U_{ijk} is given by

$$U_{ijk} = \begin{cases} \|x_j \mathbf{e}_j + x_k \mathbf{e}_k\| & \text{if } x_j, x_k > 0; \\ \min_{\ell=j,k}\{U_\ell + ||\mathbf{e}_\ell||\} & \text{otherwise.} \end{cases} \tag{13}$$

3.1. Polar angle extension

The Euclidean update can be extended to also compute a polar angle θ_i in $\mathbf{v}_i$ using the same design principle: correctness in the planar case. We assume given polar angles θ_j, θ_k in $\mathbf{v}_j$ and $\mathbf{v}_k$. There are two cases to consider, corresponding to the two cases in (13) and Figures 6 and 7. In the first case we interpolate the given polar angles linearly (modulo 2π) by

$$\theta_i = (1-\alpha)\theta_j + \alpha\theta_k, \tag{14}$$

where $\alpha = \phi_{ij}/\phi_{kj}$ and ϕ_{kj} is the (positive) angle between the vectors $\mathbf{v}_k - \mathbf{s}'$ and $\mathbf{v}_j - \mathbf{s}'$, and similarly for ϕ_{ij}. If U_i is computed by the Dijkstra update from $\mathbf{v}_j$, we simply let $\theta_i = \theta_j$, and similarly for updates from $\mathbf{v}_k$. It is easy to show that the angles thus computed are correct in the planar case, i.e. if $U_j, U_k, \theta_j, \theta_k$ are correct and Q is convex, and is consistent with the local GPM with uniformly scaled polar angles.

The above extended scheme is exact for planar meshes by construction, i.e. it yields the correct GPCs for meshes for which $u(\mathbf{x}) = \|\mathbf{x}-\mathbf{s}\|$ everywhere, e.g. a planar convex mesh or a mesh which is star-shaped with respect to $\mathbf{s}$. In general, the resulting approximation is exact in a neighbourhood of $\mathbf{s}$ including the triangles containing $\mathbf{s}$ and its edge neighbours.

4. Algorithm

In this section we present the overall approximation algorithm and explain how the extended Euclidean scheme is used to approximate GPCs. We use a dynamic programming approach similar to the Dijkstra algorithm. It works by propagating distances according to (2) in an ordered fashion, from vertices close to the base point to those farther away.

We first initialize the vertices neighbouring the base point $\mathbf{s}$ to their exact GPCs, relative to some chosen base direction $\mathbf{x}$ on $\mathcal{T}$, see Figure 4. The remaining vertex distances are initialized to some large value, while polar angles are not initialized. The algorithm next extends the initialized GPCs to the remaining mesh vertices, requiring that the distance component satisfies

$$U_i = \min_{jk} U_{ijk} \tag{15}$$

for each vertex $\mathbf{v}_i$, where the minimum is taken over all triangles containing $\mathbf{v}_i$. The polar angles are required to satisfy (14) for the triangle T_{ijk} which yields the above minimum.

Pseudocode for the overall algorithm is listed in Algorithm 1. The function *initializeNeighbourhood(s)* initialize the neighbourhood of $\mathbf{s}$ with GPCs according to the three different cases shown in Figure 4. The set *candidates* is a heap data structure containing vertices sorted according to distance $U[i]$. It is used to identify the vertex which is most likely to yield improvements to its neighbours, i.e. the vertex in *candidates* with the smallest distance. The test on line 13 prevents nodes outside a given radius *Umax* from entering *candidates*, in effect limiting the algorithm to run on a disk with radius *Umax*. The distance update U_i in (15) is computed in *computeDistance* and the corresponding polar angle θ_i in (14) is computed in *computeAngle*.

The algorithm stops when no further improvement can be found, i.e. when *candidates* is empty. At this point all vertex distances can be shown to be optimal, and the polar angles are computed from the triangle that yields the smallest distance.

Algorithm 1 Pseudo code for computing DGPC on a mesh

```
1:  for i = 1, ··· , n do
2:      U[i] = ∞
3:  end for
4:  initializeNeighbourhood(s)
5:  candidates.push( neighbourhood( s ) )
6:  while candidates.notEmpty() do
7:      j = candidates.getSmallestNode()
8:      for i ∈ neighbours(j) do
9:          newUi = computeDistance(i)
10:         if U[i] / newUi > 1 + eps then
11:             U[i] = newUi
12:             theta[i] = computeAngle(i)
13:             if newUi < Umax then
14:                 candidates.push(i)
15:             end if
16:         end if
17:     end for
18: end while
```

Algorithm 1 is very similar to Dijkstra and FMM type algorithms, however it differs in one important respect. The latter algorithms are *label setting*, meaning that a solution is constructed in a single pass. This is only possible for special types of problems: those for which each distance value depends on neighbouring values that are strictly smaller. Unfortunately, this so-called *causality property* holds only in special cases, e.g. numerical schemes based on linear interpolation for Eikonal equations discretized on an acute mesh. A solution to our problem does not satisfy the causality property, and so we could not use a strictly one-pass algorithm. Algorithm 1 is instead a so-called *label correcting* algorithm: it updates the solution until no improvement can be found. Although we have no theoretical bound on the number of steps necessary to compute a solution, it is well known that such algorithms can be very efficient in practice [Ber93]. In order to limit the number of steps we discard improvements smaller than a small user defined threshold *eps* in line 10 of the algorithm. One should set *eps* larger than machine precision, and according to the desired accuracy.

4.1. Implementation

Algorithm 1 is very easy to implement. The structure resembles the classical Dijkstra algorithm, but with the Euclidean update scheme in (13). In our implementation of *computeDistance*, we compute updates for U_i from the two triangles sharing the edge $[\mathbf{v}_i, \mathbf{v}_j]$. Alternatively one can update from all triangles containing $\mathbf{v}_i$, but we found this less efficient. We compute for each triangle T_{ijk} the corresponding coefficients x_j, x_k by (11) and check their signs. In practice the

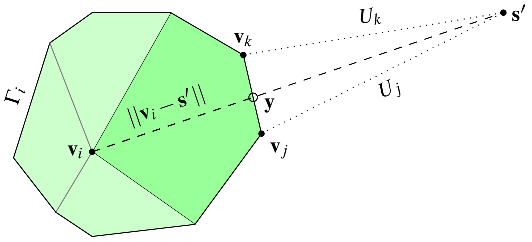


Figure 8: *The polygonal face update for a polygonal mesh use the triangle update per face-edge.*

division by the common denominator can be postponed until the computation of U_{ijk}. Note that Herons formula (12) is known to be numerically unstable for certain data. To avoid this one can use the approach proposed in [Kah]: We define $a \geq b \geq c$ to be the the sorted values U_j, U_k and $||\mathbf{e}_{kj}||$, and compute H by

$$H = \sqrt{(a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))}.$$

The computations of geodesic distance in Algorithm 1 is independent of the polar angles. The final computed angles do on the other hand depend on the resulting computed distances. In fact, an alternative approach to obtain the DGPCs is to first compute the geodesic distances for all vertices, and then the angles in a separate pass. However, we found that the latter approach was more complex and less efficient than computing both in the same pass as in Algorithm 1.

4.2. Polygonal meshes

Our method can easily be extended to meshes with arbitrary polygonal faces. In the case of a polyhedral mesh, i.e. where each face is a planar convex polygon, the term $\|\mathbf{v}_i - \mathbf{y}\|$ in (2) still represent the exact geodesic distance locally. Moreover, the local Euclidean approximation $U(\mathbf{y})$ can be computed for an edge $[\mathbf{v}_j, \mathbf{v}_k] \in \Gamma_i$, although the triangle T_{ijk} is not part of the mesh topologically as illustrated in Figure 8. Otherwise, for a non-planar polygonal face the term $\|\mathbf{v}_i - \mathbf{y}\|$ in (2) is an approximation to local geodesic distance, while $U(\mathbf{y})$ still can be computed per edge.

As a consequence we can use Algorithm 1 to approximate geodesic distance and GPCs also on polygonal meshes without explicitly triangulating the faces. To accommodate this the function *neighbours(i)* in Algorithm 1 must simply return all nodes which share a face with node $\mathbf{v}_i$, and *computeDistance(i)* should consider the faces containing $\mathbf{v}_i$ and $\mathbf{v}_j$. In our implementation we updated from the two edges of Γ_i that contains $\mathbf{v}_j$.

5. Results and Applications

We implemented our algorithm in C++, and ran experiments on a Linux computer with a 2.66 GHz Intel Core 2 processor and 4 Gb RAM. Additionally we adapted the ExpMap code kindly provided by the authors of [SGW06] in order to compare their method with ours. We performed various tests of efficiency and accuracy of the two algorithms, with a very conservative threshold of *eps*$= 10^{-12}$. We used both simple meshes for which we could find exact solutions for the GPCs, and a series of more complex meshes where this is not possible. We also provide visual results for the two methods applied for decal texturing. We conclude this section with applications of DGPC to local mesh remeshing and smoothing. A selection of the meshes used in our tests are depicted in Figure 9.

5.1. Efficiency

In our first test we computed parameterizations on a number of meshes with varying number of nodes, and recorded the resulting CPU time for our DGPC method and ExpMap respectively. We also applied a conceptually simple algorithm, denoted *TraceGPC*, which first computes exact geodesic distance by the method presented in [SSK*05], and subsequently polar angles by tracing geodesic curves back to the source. Note that although the methods are suitable for local parameterizations only, we ran the algorithms for the whole mesh in order to check the scalability.

In Table 1 we report for each mesh the total number of nodes n, the CPU time t and the number of vertices computed per millisecond. For DGPC we also report the *step ratio*, defined as the number $r = m/n$ where m is the total number of steps in the while loop in Algorithm 1. The step ratio is relevant for comparing our method with the FMM which guarantees an optimal $m = n$ and hence a step ratio $r = 1$. Our algorithm is label correcting, which implies that it may require more than n steps to compute the coordinates. It is thus of interest to verify that the step ratio does not become too large as this would imply a loss of efficiency. The step ratio will in general depend on the geometry of the mesh and on the chosen threshold *eps*.

In our global parameterizations with $eps = 10^{-12}$, the step ratio is in general very close to optimal, with the worst case $r = 1.25$ on the Bunny mesh. With $eps = 10^{-6}$, the step ratio decreased to $r = 1.16$ on this mesh. Note that for local parameterizations, which is the intended application of our method, the step ratio will typically be even closer to optimal than in our tests.

Our method appears to be comparable in efficiency with ExpMap on the Bunny mesh, but is significantly more efficient on meshes with better triangle layout. Our experiments indicate that the total computation time grows roughly linearly with n. Note that TraceGPC is inferior to the two other algorithms with respect to efficiency, particularly since backtracing of geodesic paths scales badly with increasing mesh size. We observed that the performance of our algorithm on polygonal meshes, such as the Head mesh depicted in Figure 9, is comparable to the performance on triangle meshes.

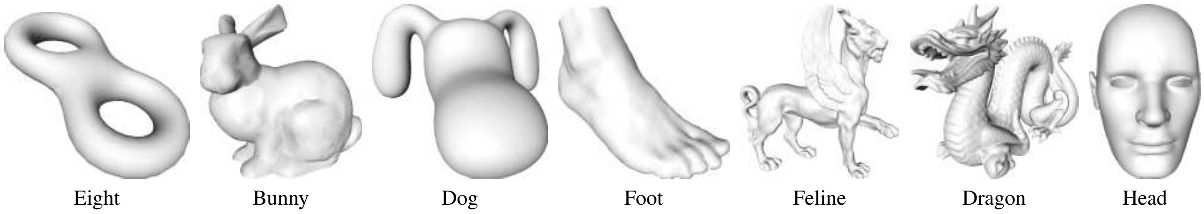


Figure 9: *A selection of meshes used in our tests. The Head mesh has polygons with up to six nodes per face, while the other meshes are triangle meshes.*

Table 1: *The efficiency of TraceGPC, ExpMap and DGPC globally. Here t denotes time in ms. For the TraceGPC algorithm, $t := t_d + t_a$ where t_d is the time to compute distances and t_a is the time to compute angles by path backtracing. The step ratio for DGPC is defined as $r := m/n$.*

		TraceGPC			ExpMap		DGPC		
Mesh	n	t_d	t	n/t	t	n/t	t	n/t	r
Eight	766	16	46	17	3	296	2	324	1.14
Noise	1089	37	199	5	4	308	3	402	1.02
Bunny	1355	26	128	11	5	290	5	292	1.25
Tri^6	2145	67	424	5	8	286	4	522	1.00
Dog	3716	123	589	6	13	296	10	381	1.01
Foot	5092	289	1.9k	3	17	297	13	380	1.01
Cone^6	13k	572	3.4k	4	46	273	30	416	1.00
Feline	50k	2.5k	55k	1	192	260	148	337	1.07
Dragon	437k	72k	4.6m	0.1	1894	231	1426	307	1.17
Head	3224						9	352	1.01

Table 2: *Distance and angle errors d_i and a_i for the triangle, cone and hemisphere meshes. The exponent after the mesh name notes the number of dyadic refinements.*

Distance error		max d_i	avg d_i
Plane^6	ExpMap	$5.9 \cdot 10^{-8}$	$1.3 \cdot 10^{-8}$
	DGPC	$6.7 \cdot 10^{-16}$	$6.7 \cdot 10^{-17}$
Cone^6	ExpMap	$8.1 \cdot 10^{-3}$	$2.1 \cdot 10^{-3}$
	DGPC	$6.7 \cdot 10^{-16}$	$6.7 \cdot 10^{-17}$
Hemisphere^8	ExpMap	$1.7 \cdot 10^{-2}$	$2.6 \cdot 10^{-3}$
	DGPC	$3.5 \cdot 10^{-4}$	$8.9 \cdot 10^{-5}$

Angle error		max a_i	avg a_i
Plane^6	ExpMap	$2.6 \cdot 10^{-8}$	$5.0 \cdot 10^{-9}$
	DGPC	$8.7 \cdot 10^{-13}$	$2.0 \cdot 10^{-14}$
Cone^6	ExpMap	$6.8 \cdot 10^{-2}$	$2.7 \cdot 10^{-2}$
	DGPC	$8.7 \cdot 10^{-13}$	$2.0 \cdot 10^{-14}$
Hemisphere^8	ExpMap	$8.1 \cdot 10^{-2}$	$2.3 \cdot 10^{-2}$
	DGPC	$1.4 \cdot 10^{-3}$	$2.4 \cdot 10^{-4}$

5.2. Accuracy

We first evaluate the accuracy of DGPC and ExpMap on meshes for which we can find exact or very accurate solutions. Our first test mesh is a single triangle in the xy-plane, with the source in the origin and the two other vertices in $(0.5, 1)$ and $(-0.5, 1)$ respectively. Our second test mesh is a six faced cone with the six boundary nodes evenly distributed on the unit circle in the xy-plane, and the base point at the apex in $(0, 0, 1)$. Both meshes were dyadically refined six times, resulting in meshes with 2145 and 12481 nodes respectively. The exact GPCs for these simple surfaces are easy to compute directly from the definition since geodesics on both meshes are straight lines, i.e. the meshes are star-shaped with respect to $\mathbf{s}$ and hence $r_{\mathbf{s}}(\mathbf{v}) = \|\mathbf{v} - \mathbf{s}\|$. We compared the exact angular and distance value with the approximations computed by the DGPC and ExpMap algorithms. The distance error d_i and angular error a_i of DGPC was close to machine precision for both meshes as shown in table 2, while the error with ExpMap was significantly higher, in particular for the non-planar cone example.

Our third test mesh is a convergence test on discretizations of the half unit sphere with the base point in the central point $(0, 0, 1)$. We ran our algorithm on a sequence of successively finer meshes approximating the half-sphere. The cone mesh was repeatedly refined dyadically and the vertices projected to the unit sphere. We compared the resulting DGPCs with the exact GPCs for the smooth half-sphere after each refinement. Note that the latter is straight forward to compute, but not for the approximating meshes. However, the approximating meshes converge to the half-sphere with a second order rate, and it is therefore relevant to compare the DGPCs with those of the half-sphere. Figure 10 shows the evolution of maximum error in both distance and angle for both DGPC and ExpMap. Initially the distances on the six faced cone are bad approximations to distances on the half sphere. While the DGPC distance errors decrease under mesh refinement, ExpMap does not appear to improve further after the first three steps. Both methods are initialized with exact polar angles at the coarsest level. After one level of dyadic refinement, angular errors appear for both methods. Thereafter the angular errors in the ExpMap method are roughly constant, while the DGPC angular errors are reduced and appear to converge under refinement. The ExpMap method does not appear to converge. Table 2 shows the max and average er-

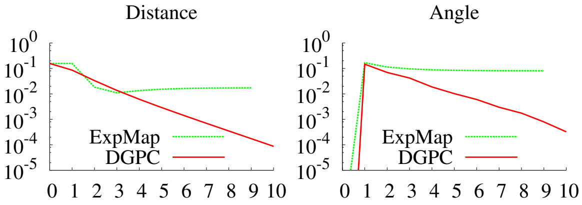


Figure 10: *Absolute maximum errors for ExpMap and DGPC under dyadic refinement of a hemisphere.*

ror after eight refinements, where the mesh has about 200k vertices.

We now proceed to evaluate the accuracy on meshes where the exact GPCs are nontrivial. As mentioned earlier, our DGPC parameterizations are applicable only in a region around a base point, and it is therefore of particular interest to measure the accuracy of our approximations locally. In our next test we computed local DGPC parameterizations on a selection of the meshes in Figure 9. For each mesh we did the following. For each vertex $\mathbf{v}_i$ we set $\mathbf{s} = \mathbf{v}_i$ and computed DGPC and ExpMap coordinates in a disk with a fixed geodesic radius. We chose the radius such that for all base points $\mathbf{v}_i$, at least ten vertices were inside the corresponding disk. The method presented in [SSK*05] was used to compute exact reference GPCs for comparison, by measuring the length and polar angles of geodesic curves. The error for each vertex in the disk with respect to the chosen base point $\mathbf{v}_i$ is thus defined to be the difference in angle and distance between the reference and the approximated GPCs. We then recorded the average error d_i^{avg} and a_i^{avg} and maximum error d_i^{max} and a_i^{max} for vertex $\mathbf{v}_i$, for both DGPC and ExpMap.

Table 3 shows the resulting distance errors. The worst case errors of DGPC is approximately 10% of the disk radius on the Foot and on the Bunny mesh. For the other meshes, the worst case error is approximately 1% of the disk radius. The average worst case error, and the worst case average error are both below the order of 0.1%. The errors of ExpMap are in general approximately one order of magnitude larger than DGPC, and worst case error for ExpMap is larger than the disk radius on the Bunny mesh. This mesh is difficult due to ill shaped triangles.

Table 4 shows the corresponding angular errors, measured in radians. The worst case error was 0.82 radians for DGPC and 3.1 radians for ExpMap, both on the Bunny mesh. The large errors occurs for the ill shaped triangle configurations. The average maximum error is however much lower. The angular error for ExpMap appears in general to be more than twice the angular error for DGPC.

Table 3: *Distance errors for vertices inside disks of fixed radius, scaled by the size of the disks to make the errors comparable among the meshes.*

Distance error		$\max d_i^{\text{max}}$	avg d_i^{max}	$\max d_i^{\text{avg}}$	avg d_i^{avg}
Eight	DGPC	$1.0 \cdot 10^{-2}$	$4.7 \cdot 10^{-4}$	$2.0 \cdot 10^{-3}$	$9.7 \cdot 10^{-5}$
	ExpMap	$3.6 \cdot 10^{-2}$	$4.8 \cdot 10^{-3}$	$1.5 \cdot 10^{-2}$	$1.2 \cdot 10^{-3}$
Bunny	DGPC	$1.2 \cdot 10^{-1}$	$5.2 \cdot 10^{-3}$	$7.8 \cdot 10^{-3}$	$3.9 \cdot 10^{-4}$
	ExpMap	$1.4 \cdot 10^{0}$	$1.1 \cdot 10^{-1}$	$7.6 \cdot 10^{-2}$	$4.5 \cdot 10^{-3}$
Dog	DGPC	$1.3 \cdot 10^{-2}$	$5.9 \cdot 10^{-4}$	$3.7 \cdot 10^{-4}$	$1.8 \cdot 10^{-5}$
	ExpMap	$6.0 \cdot 10^{-2}$	$3.1 \cdot 10^{-3}$	$4.2 \cdot 10^{-3}$	$2.7 \cdot 10^{-4}$
Foot	DGPC	$1.1 \cdot 10^{-1}$	$7.4 \cdot 10^{-3}$	$1.6 \cdot 10^{-3}$	$6.4 \cdot 10^{-5}$
	ExpMap	$3.9 \cdot 10^{-1}$	$2.4 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$5.4 \cdot 10^{-4}$
Noise	DGPC	$1.4 \cdot 10^{-2}$	$1.0 \cdot 10^{-3}$	$1.2 \cdot 10^{-4}$	$9.4 \cdot 10^{-6}$
	ExpMap	$9.9 \cdot 10^{-1}$	$8.3 \cdot 10^{-2}$	$1.2 \cdot 10^{-2}$	$1.4 \cdot 10^{-3}$

Table 4: *Angular errors for vertices in disks of fixed radius.*

Angle error		$\max a_i^{\text{max}}$	avg a_i^{max}	$\max a_i^{\text{avg}}$	avg a_i^{avg}
Eight	DGPC	$2.3 \cdot 10^{-2}$	$1.5 \cdot 10^{-3}$	$9.5 \cdot 10^{-3}$	$5.7 \cdot 10^{-4}$
	ExpMap	$7.8 \cdot 10^{-2}$	$6.2 \cdot 10^{-3}$	$3.3 \cdot 10^{-2}$	$3.4 \cdot 10^{-3}$
Bunny	DGPC	$8.2 \cdot 10^{-1}$	$5.2 \cdot 10^{-2}$	$5.4 \cdot 10^{-2}$	$2.9 \cdot 10^{-3}$
	ExpMap	$3.1 \cdot 10^{0}$	$3.2 \cdot 10^{-1}$	$1.4 \cdot 10^{-1}$	$9.7 \cdot 10^{-3}$
Dog	DGPC	$1.0 \cdot 10^{-1}$	$5.6 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$	$1.8 \cdot 10^{-4}$
	ExpMap	$1.5 \cdot 10^{-1}$	$1.3 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$8.6 \cdot 10^{-4}$
Foot	DGPC	$7.8 \cdot 10^{-1}$	$3.2 \cdot 10^{-2}$	$1.4 \cdot 10^{-2}$	$6.1 \cdot 10^{-4}$
	ExpMap	$1.2 \cdot 10^{0}$	$7.0 \cdot 10^{-2}$	$3.2 \cdot 10^{-2}$	$1.9 \cdot 10^{-3}$
Noise	DGPC	$7.1 \cdot 10^{-2}$	$5.5 \cdot 10^{-3}$	$6.3 \cdot 10^{-4}$	$5.1 \cdot 10^{-5}$
	ExpMap	$3.6 \cdot 10^{-1}$	$2.8 \cdot 10^{-2}$	$9.8 \cdot 10^{-3}$	$1.2 \cdot 10^{-3}$

5.3. Local texturing

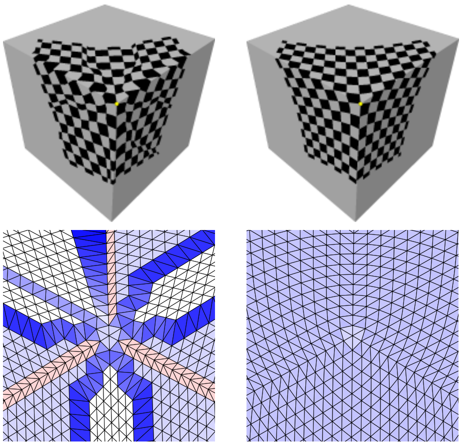
The objective of this section is to evaluate our coordinates for local texturing, so-called *decals*. In particular, we are interested in assessing the visual quality of the texturing and the local area distortion.

Local texturing amounts to pasting an image onto a surface mesh in a neighbourhood around a chosen base point. This requires a correspondence between the surface neighbourhood and the 2*D* image, such that we can give each vertex in the neighbourhood a 2*D* texture coordinate. The GPCs provide such a local parameterization, and their basic properties ensure low distortion near the base point. After Algorithm 1 has terminated, each vertex in the region of interest has an associated approximate GPC. These are easily converted to Cartesian texture coordinates with respect to some user defined coordinate system, i.e. scaled and aligned with a preferred direction. The resulting coordinates can be used directly in a rendering system such as OpenGL.

In order to measure the distortion of a local parameterization we used the logarithm of the Jacobian determinant of the mapping of a triangle τ_i in parameter (GPC) space to the corresponding triangle T_i on the 3*D* mesh. This simplifies to

$$\sigma_i := \log(\text{area}(T_i)/\text{area}(\tau_i)),$$

and measures how much the area of a piece of the texture



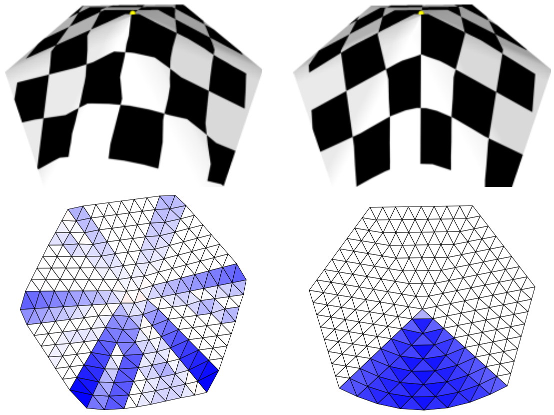
ExpMap DGPC

Figure 11: *A cube with the base point on a corner. The ExpMap max distortion is* 1.24*. The DGPCs are exact to machine precision, and with max area distortion at* 0.32 .

image is changed when pasted onto the mesh. In the following figures of the 2D parameter domain, triangles are coloured according to area distortion σ_i. We let white correspond to $\sigma = 0$, blue to stretch ($\sigma > 0$) and red to compression ($\sigma < 0$), and scale the colour intensity according to the largest area distortion among the parameterizations that are compared. Triangles that fold over in the parameter domain are coloured black. This occurs where the local GPM is not one-to-one and hence not a valid parameterization.

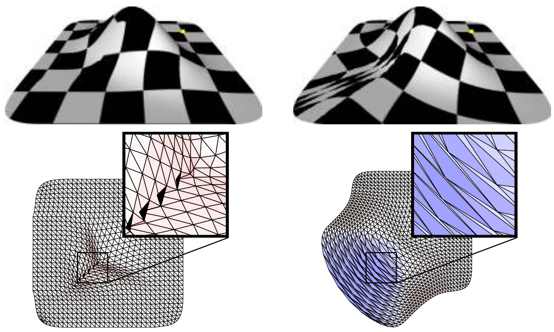
We now proceed with a number of examples where we compare our method with ExpMap [SGW06], applying the two methods for local texturing of different meshes, and showing both the 3D mesh with a chequerboard texture, and the 2D mesh parameterization coloured as explained above. Our first example is depicted in Figure 11. The source is on the corner of a cube, and the texture extends onto the three adjacent faces. The DGPC method yields coordinates that are exact to machine precision and the distortion is distributed uniformly. The ExpMap method yields large error and an irregular and significant distortion. Figure 12 depicts the six faced cone rendered with the base point on one of the faces, behind the apex. Both methods have significant distortion on the face opposite the source, but the DGPC method again shows a more regular distortion. We also tested with the source at the apex (not shown), with results similar to the cube example. In fact, the DGPC are in this case exact to machine precision (see Table 2), and the observed distortion with DGPC is inherent with geodesic coordinates.

Figure 13 depicts a smooth ridge. When the base point is behind the ridge as shown in the figure, both methods result in high distortion in the region in front of the ridge. This is expected, since there are singular points in this region. DGPC yields highly stretched triangles, but the triangles do not fold over as they do for the ExpMap method. Also, the distortion and the triangles in the DGPC parameterization appear to be evenly and symmetrically distributed, and this is also reflected in the texturing of the surface.



ExpMap DGPC

Figure 12: *A six faced cone, with the base point behind the apex. The max distortion is* 0.70 *and* 0.67 *for ExpMap and DGPC respectively.*



ExpMap DGPC

Figure 13: *A smooth ridge with base point behind the ridge. The max distortion for ExpMap is* 6.51 *and* 2.48 *for DGPC. Note that ExpMap yields triangles (filled black) that fold over in the parameter domain.*

The mesh depicted in Figure 14 is a simplified version of the Stanford Bunny, with 1355 vertices and some particularly ill shaped triangles. ExpMap creates triangle foldovers in the parameter domain, and visual artefacts in the texture. No foldovers or similar artefacts can be seen for the DGPCs.

Our next mesh is a flat surface contained in the unit square in the xy-plane, shown in Figure 15. Random noise with magnitude 0.01 is added to the z-component of the vertices. Although the DGPC method is more accurate than the ExpMap method on this example, see Table 4, the latter nevertheless yields the smallest distortion. We believe the reason is that the ExpMap method average out the noise when

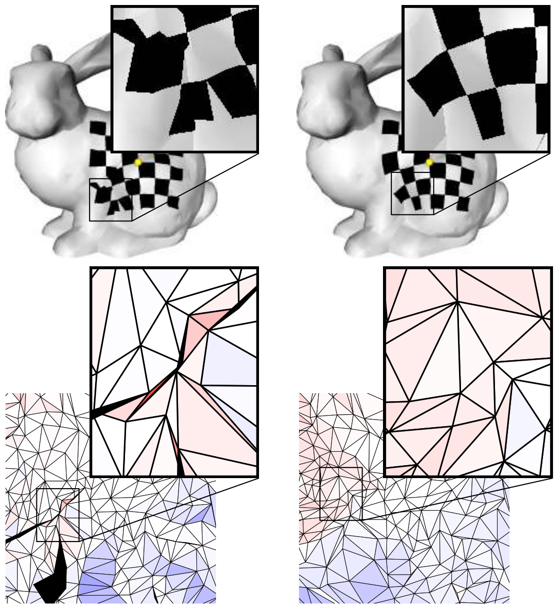


Figure 14: *Local texturing of a mesh with ill-shaped triangles. The ExpMap method yields triangle foldovers in the parameter domain and corresponding texturing artefacts. The max distortion is* 3.55 *for ExpMap, and* 1.18 *for DGPC.*

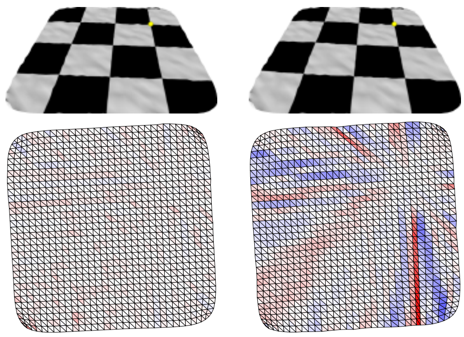


Figure 15: *Random noise added to a planar triangulation. The maximum distortions are* 0.37 *and* 0.47 *respectively.*

tangent planes are estimated. Thus, the distortion in DGPC is inherent in the GPM parameterization, and not due to approximation error.

In Figure 16 we have applied our algorithm to a polygonal mesh. As shown, the method can treat a hole in the surface as a polygonal face, and thus enable the parameterization to extend naturally across holes. Figure 17 illustrates that our approach can handle degenerate triangles and ill-shaped polygons well.

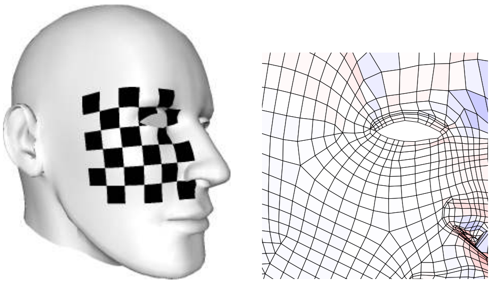


Figure 16: *DGPC for a polygonal mesh can extend naturally across holes in the surface.*

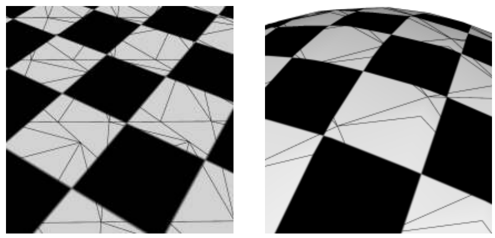


Figure 17: *Our method can handle degenerate triangles (left) and ill-shaped polygons (right).*

In our final test, we compare our results with local parameterizations based on the popular Mean Value Coordinates (MVC) proposed in [Flo03]. We conducted one test on a discretization of the unit half-sphere with 817 vertices. The boundary was mapped to a circle with radius $\pi/2$, corresponding to the exact GPC for the half-sphere with the base point in the top point. The MVC method extrapolates the boundary parameter ($2D$) coordinates to the interior as shown in Figure 18 (right). The MVC parameterization is smooth, with the highest distortion near the base point (compression) and near the boundary (stretch). The corresponding DGPC parameterization has negligible distortion

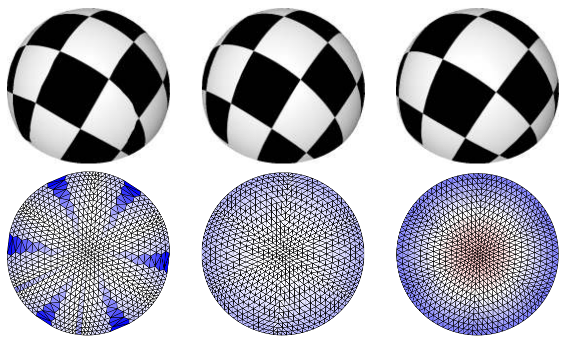


Figure 18: *A half sphere with the base point at the top. The max distortion is* 1.58, 0.46 *and* 0.78 *respectively.*

near the source point, but increases gradually towards the boundary. The maximum distortion with MVC is higher than the maximum distortion with DGPC. The ExpMap method shows higher and more irregular distortion and yields visual artefacts. The parameterization took about 2 ms to compute with ExpMap and DGPC, and 31 ms with MVC.

In practice, if MVC is to be used for local parameterizations, the user must define a suitable boundary, i.e. the region to be parameterized. The boundary has a significant effect on the resulting distortion, and must be chosen with care. A natural approach is to choose a geodesic disk, which in principle requires the same amount of work as our full approach. In addition one must set up and solve the resulting linear system in order to obtain the MVC. This approach obviously requires more computations than our method. However, the result is guaranteed to have no foldovers if a valid disk shaped region is used.

5.4. Mesh resampling and smoothing

Local mesh parameterizations can be used for local operations on a mesh, e.g. by applying vector space algorithms. As an example, we show how our GPMs can be used in combination with Laplacian smoothing for local resampling of a polygonal mesh in order to improve its quality.

A common remeshing strategy is to construct a mesh with nice properties in the parameter plane and use a parameterization of the 3D mesh to resample it on the 2D mesh, resulting in a 3D mesh with corresponding high quality. A good parameterization with low distortion is necessary to ensure that well-shaped faces in the parameter domain are mapped to well-shaped faces in $\mathbb{R}^3$. Our approach is applicable for such an approach since it yields high quality parameterizations of a region around a source point $\mathbf{s}$. Let us briefly explain how. We first make a local parameterization around $\mathbf{s}$ by using our DGPC algorithm. This results in a planar mesh with vertices $\mathbf{p}_i$ which correspond to the computed DGPCs. We denote by $P_\mathbf{s}$ the resulting piecewise linear map from $\mathbb{R}^3$ to $\mathbb{R}^2$. Then we apply Laplacian smoothing to this mesh, i.e. iteratively compute new $2D$ vertex positions according to

$$\mathbf{p}_i' = \mathbf{p}_i + \beta \frac{1}{n} \sum_{j=1}^{n} (\mathbf{p}_j - \mathbf{p}_i), \tag{16}$$

keeping its boundary fixed. Here the $\mathbf{p}_j$ are the mesh neighbours of $\mathbf{p}_i$, and $\beta \in [0,1]$ is a constant weight factor. Finally we find new vertex positions $\mathbf{v}_i^{new} \in \mathbb{R}^3$ by sampling the inverse GPM

$$\mathbf{v}_i^{new} = P_\mathbf{s}^{-1}(\mathbf{p}_i). \tag{17}$$

The resulting $3D$ mesh $\mathcal{T}^{new}$ will maintain the same topology as $\mathcal{T}$, but with vertices $\mathbf{v}_i^{new}$ in new positions on the original mesh surface.

The result of this approach is illustrated in the right part of Figure 19. The new mesh have much better face shapes and also appears smoother, with most of the geometric detail intact. If repeated, this operation tend to result in a smoother mesh, with new vertices on the surface from the previous iteration. This will preserve much of the mesh geometry between iterations, as opposed to traditional 3D Laplacian smoothing, which can be rather aggressive and remove significant details from the mesh, and also result in mesh shrinking. Taubin smoothing [Tau95] is designed to prevent mesh shrinking (shown in the middle part of the figure), but still removes more detail than our approach.

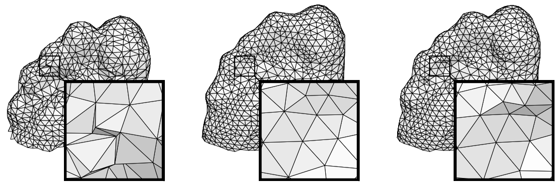


Figure 19: *Left: The Foot mesh has very irregular faces under the toes. Middle: Taubin smoothing applied locally to the mesh improves triangle shapes but removes important geometric detail. Right: Laplacian smoothing of the GPCs combined with GPM resampling increase the mesh quality without degrading details.*

As we demonstrated in the previous section, our algorithm can compute local coordinates for hundreds of nodes per millisecond. Since also Laplacian smoothing is very fast, the above approach can be used for an interactive "Magic Wand" tool for polygonal meshes, by brushing over areas with ill-shaped faces and instantly improve the mesh quality, see the multimedia material at [MR].

6. Summary

We have presented a generalization of the classical exponential map to polygonal meshes in 3D, by extending the local polar map from [WW94] to form GPMs and corresponding polar coordinates. The coordinates are approximated by extending the Euclidean scheme for computing geodesic distance on a triangle mesh proposed in [Rei04]. By this, we obtain a very efficient and accurate method to compute local mesh parameterizations with low metric distortion in a region of interest around a base point. The extension of our algorithm to approximate geodesic distance on a general polygonal mesh is to our knowledge the first of its sort. We have published an implementation of the algorithm and some multimedia material at [MR].

With regards to efficiency, our algorithm appears to be faster than the ExpMap method and comparable to a single pass algorithm like the FMM in practice. For local parameterizations the running time appears to be independent on the total number of nodes in the mesh. Our implementation of the algorithm could compute DGPCs for more than

300 nodes/ms, which makes it suitable for real-time applications. A naive approach based on computing geodesic curves turned out to be several orders of magnitude slower than our algorithm.

The numerical examples show that our approach computes GPCs exact to machine precision in cases where all geodesics are straight lines, e.g. for meshes that are planar or star shaped with respect to the base point. For general meshes the errors where small and appeared to converge under mesh refinement, both in the distance and polar angle components. This is in contrast to the ExpMap method whose errors were 1-2 orders of magnitude larger and with no indication of convergence. The angular error for the DGPCs was reasonable in most cases, although we observed relatively large angular errors in extreme cases. The visual results were in general very pleasing.

We compared DGPC visually to the ExpMap method by applying the coordinates for local texturing. While distortion is unavoidable in general, the two methods had different characteristics. We found that the DGPC method resulted in smaller and more evenly distributed distortion and smooth textures, while the ExpMap method often had artefacts and more irregular distortion. In comparison, parameterizations based on Mean Value Coordinates yields smooth parameterizations but admits no control of distortion around the base point. Moreover, this approach requires a user defined boundary and is significantly slower than our method.

We also showed how the DGPCs could be used for local mesh processing algorithms, by resampling a 3*D* polygonal mesh in order to improve its quality. This is an example of a 2*D* vector space algorithm that can be applied to a 3*D* mesh when one has a good local parameterization at hand. We believe our approach has many more applications that can leverage on this.

References

[AS99] ADALSTEINSSON D., SETHIAN J. A.: The fast construction of extension velocities in level set methods. *J. Comput. Phys. 148* (January 1999), 2–22. 2

[Ber93] BERTSEKAS D. P.: A simple and fast label correcting algorithm for shortest paths. *Netw. 23* (1993), 703–709. 5

[BWHK05] BRUN A., WESTIN C.-F., HERBERTHSON M., KNUTSSON H.: Fast manifold learning based on Riemannian normal coordinates. In *Proceedings of the SCIA;05* (Joensuu, Finland, June 2005), pp. 920–929. 1

[DMA02] DESBRUN M., MEYER M., ALLIEZ P.: Intrinsic parameterizations of surface meshes. In *Eurographics 2002 proceedings* (2002), vol. 21, Blackwell Publishers, pp. 209–218. 2

[FH05] FLOATER M. S., HORMANN K.: Surface parameterization: a tutorial and survey. In *In Advances in Multiresolution for Geometric Modelling* (2005), Springer, Berlin, pp. 157–186. 2

[Flo03] FLOATER M. S.: Mean value coordinates. *Comput. Aided Geom. Des. 20* (March 2003), 19–27. 2, 10

[GR85] GONZALEZ R., ROFMAN E.: On deterministic control problems: an approximation procedure for the optimal cost. I. The stationary problem. *SIAM Journal on Control and Optimization 23*, 2 (1985), 242–266. 3

[Kah] KAHAN W.: Miscalculating area and angles of a needle-like triangle. Preprint (1997), available at http://www.cs.berkeley.edu/~wkahan/Triangle.pdf. Accessed 6 August 2012. 6

[KS98] KIMMEL R., SETHIAN J. A.: Computing geodesic paths on manifolds. In *Proc. Natl. Acad. Sci. USA* (1998), pp. 8431–8435. 2

[LZ08] LIN T., ZHA H.: Riemannian manifold learning. *IEEE Trans. Pattern Anal. Mach. Intell. 30*, 5 (May 2008), 796–809. 1

[MMS07] MPIPERIS I., MALASSIOTIS S., STRINTZIS M.: 3-D face recognition with the geodesic polar representation. *Information Forensics and Security, IEEE Transactions on 2*, 3 (sept. 2007), 537 –547. 1, 2

[MR] MELVÆR E. L., REIMERS M.: DGPC project page. http://folk.uio.no/eivindlm/dgpc/. Accessed 6 August 2012. 11

[NK02] NOVOTNI M., KLEIN R.: Computing geodesic distances on triangular meshes. In *In Proc. of WSCG'2002* (Plzen-Bory, Czech Republic, 2002), pp. 341–347. 4

[Pen06] PENNEC X.: Intrinsic statistics on Riemannian manifolds: Basic tools for geometric measurements. *J. Math. Imaging Vis. 25*, 1 (2006), 127–154. 1

[Rei04] REIMERS M.: *Topics in Mesh based Modeling*. PhD thesis, University of Oslo, 2004. 2, 3, 4, 11

[SGW06] SCHMIDT R., GRIMM C., WYVILL B.: Interactive decal compositing with discrete exponential maps. *ACM Trans. Graph. 25* (July 2006), 605–613. 1, 2, 6, 9

[SGW09] SCHNEIDER J., GEORGII J., WESTERMANN R.: Interactive geometry decals. In *VMV'09* (Braunschweig, Germany, 2009), pp. 197–204. 2

[SM09] SUBBARAO R., MEER P.: Nonlinear mean shift over Riemannian manifolds. *Int. J. Comput. Vision 84*, 1 (2009), 1–20. 1

[SPR06] SHEFFER A., PRAUN E., ROSE K.: Mesh parameterization methods and their applications. *Found. Trends. Comput. Graph. Vis. 2* (January 2006), 105–171. 2

[SS09] SHAPIRA L., SHAMIR A.: Local geodesic parametrization: an ant's perspective. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, Möller T., Hamann B., Russell R. D., (Eds.), Mathematics and Visualization. Springer Berlin Heidelberg, 2009, pp. 127–137. 2

[SSCO06] SHAMIR A., SHAPIRA L., COHEN-OR D.: Mesh analysis using geodesic mean-shift. *Vis. Comput. 22*, 2 (2006), 99–108. 1

[SSK*05] SURAZHSKY V., SURAZHSKY T., KIRSANOV D., GORTLER S. J., HOPPE H.: Fast exact and approximate geodesics on meshes. *ACM Trans. Graph. 24*, 3 (2005), 553–560. 6, 8

[Tau95] TAUBIN G.: A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 351–358. 11

[WD05] WALLNER J., DYN N.: Convergence and C^1 analysis of subdivision schemes on manifolds by proximity. *Computer Aided Geometric Design 22*, 7 (2005), 593 – 622. 1

[WW94] WELCH W., WITKIN A.: Free-form shape design using triangulated surfaces. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1994), SIGGRAPH '94, ACM, pp. 247–256. 1, 2, 3, 11