Computing the Riemannian Center of Mass on Meshes

Claudio Mancinelli[a,*], Enrico Puppo[a]

[a] *DIBRIS - University of Genoa (Italy)*

Abstract

The Riemannian center of mass (a.k.a. Karcher mean or Fréchet mean) provides the equivalent to the Euclidean affine average on manifolds. In spite of its many potential applications in computer graphics and geometric modeling, there exist surprisingly few algorithms to compute it. We present a direct method for computing the Riemannian center of mass on a triangle mesh. Our method works in the polyhedral metric and uses a piecewise-linear interpolation of gradients of the distance fields from a set of control points. We present applications for tracing splines on a surface, comparing to other methods at the state of the art, and showing that we produce quality results while supporting user interaction.

Keywords: Riemannian center of mass, Karcher mean, Fréchet mean, Geodesics, Triangle meshes

1. Introduction

The *Riemannian center of mass* (RCM) has been defined by Grove and Karcher (1973) as the solution of an optimization problem on a Riemannian manifold, which provides a sort of *weighted average* of a finite set of points distributed on the manifold. In fact, in a Euclidean space, the RCM reduces to the standard affine average. See Fig. 1 for an example and Sec. 3 for a formal definition.

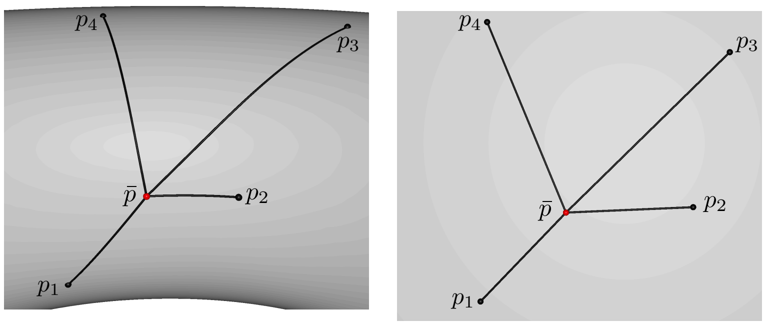


Figure 1: Left: the Riemannian center of mass $\bar{p}$ of four points p_1, p_2, p_3, p_4 on the surface of a torus with weights $0.47, 0.30, 0.08, 0.15$, respectively; the center of mass is joined to the control points with shortest geodesic paths, whose lengths minimize the energy. Right: a similar configuration of points in the plane and their affine average with the same weights.

Given the many applications of affine averages in computer graphics and geometric modeling, the RCM is a natural candidate to support computations on surfaces. This approach represents a more direct alternative to the traditional choice of relying on a parametrization of a surface. Indeed, as discussed by Yuksel et al. (2019); Nazzaro et al. (2022), parametrization-based methods suffer from a number of drawbacks, which could be overcome by working in the intrinsic metric of the surface.

*Corresponding author
Email addresses: `claudio.mancinelli@edu.unige.it` (Claudio Mancinelli), `enrico.puppo@unige.it` (Enrico Puppo)

Panozzo et al. (2013) discuss applications of the RCM to splines on surfaces, remeshing, Laplacian smoothing, texture mapping, texture transfer, etc. In spite of that, computing the RCM has long been considered too hard to be of practical interest. In fact, while Euclidean affine averages are computed in closed form, the RCM requires resolving an optimization problem on a manifold, which in turn involves computing distance fields and their gradients, as well as tracing geodesic curves. For this reason, there exist surprisingly few algorithms in the literature for its computation. While Panozzo et al. (2013) circumvent the problem, using an embedding in a higher-dimensional Euclidean space to find an approximate solution, only recently Sharp et al. (2019b) proposed a direct solution based on gradient descent, which exploits the relation between the gradient of the squared distance field and the logarithmic map at each point.

In this paper, we present a method for the direct computation of the Riemannian center of mass on meshes, which is based on an implementation of a general Newton's method on Riemannian manifolds, as described by Absil et al. (2008). Our method tackles the problem on the input mesh endowed with the polyhedral metric. Given the distance fields from the control points at the vertices of the mesh, we estimate the gradient and the Hessian of the energy function inside each triangle, and we trace the geodesic line from a given point and following a given tangent vector to proceed towards the solution.

Our method is oblivious of the algorithm used to compute the distance fields; of course, there may be some trade-off between accuracy and computational cost, which will affect the results of the RCM, too. Overall, the cost of computing the RCM once for a given set of weights is negligible with respect to the initial cost of computing the distances fields. In most applications, however, the RCM is computed many times with varying weights, so this cost may become relevant. We discuss these aspects in the experiments.

We apply our method to curves on surfaces, comparing our results with results obtained with other methods at the state of the art to address the same problem. In particular, we compare with the *Vector heat* (VH) method by Sharp et al. (2019b), showing that our method is more stable and faster by several orders of magnitude; and with the *Weighted average* (WA) method by Panozzo et al. (2013), showing that we produce more accurate results, offering a direct solution to the RCM problem, while being slightly slower. Besides, our method is much more scalable with respect to the size of the mesh. In fact, while we just rely on local computations, the VH method requires pre-conditioning a large linear system, which is of the same size of the input mesh; and the WA method also requires a cumbersome pre-processing step, which may become unfeasible for large meshes. An implementation of our method is released in open source at `https://github.com/Claudiomancinelli90/RCM_on_meshes`.

2. Related work

Riemannian center of mass. The RCM was introduced first by Grove and Karcher (1973) while Karcher (1977) provides a more extensive treatment of the subject; variations of the same concept are also known as *Karcher mean* or *Fréchet mean* in the literature. Absil et al. (2008) provide algorithms to find the minimum of functions on a Riemannian manifold, which can be applied to compute the RCM; they take for granted the necessary differential quantities without addressing their computation, though.

Panozzo et al. (2013) consider the direct computation of the RCM too computationally involved and circumvent the problem, relying on a high-dimensional embedding, where Euclidean distance can approximate the geodesic distance on the input manifold. Instead of resolving the RCM on the input, they compute in closed form a standard affine average in the embedding space; next, they use a *Phong projection* (which is a main contribution of their work) to project the result to the embedded surface; finally, they pull the result back to the original surface. While their method is effective and fast, it requires a cumbersome pre-processing, it is prone to artifacts from projection, and it does not scale to large meshes. Besides, it provides no insight for a direct computation of the RCM.

More recently, Sharp et al. (2019b) proposed a first direct algorithm for computing the RCM, as an application of their *Vector heat method*. They exploit the relation between the gradient of the squared distance field and the log map at each point to provide a gradient descent algorithm. Unfortunately, their algorithm is very slow, since it requires recomputing the log map centered at each point traversed during descent, and it is also sensitive to perturbations due to computing log maps of moving points.

In Sec.5, we directly compare our results to these two methods.

Geodesic queries. The literature about computing geodesics on meshes is vast. Surveys can be found in Bose et al. (2011); Crane et al. (2020). Here we review just some results relevant to our work.

The computation of the *distance field* from a source is a fundamental component that we take for granted. Exact methods for polyhedral surfaces stem from the works of Mitchell et al. (1987) and Chen and Han (1990) and produce a data structure which, when queried at a generic point, returns its *exact* distance from the source in the polyhedral metric. Graph methods restrict the possible paths to chains of arcs in a pre-computed graph, thus providing an approximate solution. See, e.g., Adikusuma et al. (2020); Nazzaro et al. (2022) for discussions on the trade-off between accuracy and speed. PDE methods compute approximate geodesic distances on a (unknown) smoothed surface just at the vertices of an input mesh: Kimmel and Sethian (1998) proposed the FMM, which is based on wavefront propagation; and Crane et al. (2013) proposed the *heat* method, which is based on diffusion. In our experiments, we rely alternatively on the exact polyhedral algorithm VTP of Qin et al. (2016), or on a simple graph-based solver.

The *two-points boundary value problem* consists of finding the shortest path between two points. The data structure built with the polyhedral methods mentioned above may be used to quickly extract the exact polyhedral shortest path, too. When the distance field is known just at the vertices, as in PDE methods, it is customary to extend it linearly inside triangles and trace the shortest path by descending the piecewise-constant gradient of this function. Several other works address the direct computation of the shortest path without pre-computing the distance field, some notable examples being Sharp and Crane (2020); Xin and Wang (2007). Such algorithms need an initial guess and provide just a *locally shortest* solution that is homotopic to the initial guess. In our experiments, we consider the two-points boundary problem as an application of the RCM for just two control points.

The *initial value problem* consists of tracing a geodesic line from a given point, with an initial tangent, and for a given length. This problem was studied in the polyhedral metric by Polthier and Schmies (1998): a geodesic proceeds straight inside triangles and across edges, by unfolding adjacent triangles to the same plane; if it hits a vertex v, they impose that the ingoing and outgoing directions split the total angle about v in two halves. We use this solution in the context of our algorithm.

3. The Riemannian center of mass

Let M be a compact and piecewise-smooth manifold[1] endowed with the geodesic distance $d : M \times M \to \mathbb{R}$,

$$d(x, y) = \min_{\gamma} \text{length}(\gamma),$$

where γ varies over all curves joining x and y on M. Given points $p_1, \ldots, p_h \in M$, henceforth called the *control points*, and weights $w_1, \ldots, w_h \in \mathbb{R}$, we define the energy function $K_{p_1,\ldots,p_h;w_1,\ldots,w_h} : M \to \mathbb{R}$

$$K_{p_1,\ldots,p_h;w_1,\ldots,w_h}(p) = \sum_{i=1}^{h} w_i d(p, p_i)^2. \tag{1}$$

The *Riemanninan center of mass* (RCM) of the control points with the given weights is defined

$$RCM(p_1, \ldots, p_h; w_1, \ldots, w_h) = \arg\min_{p \in M} K_{p_1,\ldots,p_h;w_1,\ldots,w_h}(p). \tag{2}$$

In the following, whenever no ambiguity arises, we will denote the energy simply with K while omitting the subscripts. If M is a Euclidean space, then the solution to Eq. 2 is the usual affine average $\bar{p} = \sum_{i=1}^{h} w_i p_i$.

Karcher (1977) provides a condition of existence and uniqueness of the solution of Eq. 2, which requires all points p_i to be contained inside a strongly convex ball. While this requirement is very strong, in practice numerical solutions can be found provided that the control points are not too far from each other and the space in between is not too “bumpy”. This makes the computation of the RCM inherently sensitive to the roughness of M in a variational sense. We discuss the limitations that follow in Sec.5.

[1] In Grove and Karcher (1973), the RCM is defined for a Riemannian manifold, hence smooth. We relax this definition to a piecewise-smooth manifold to encompass the polyhedral case. While the definition and the method proposed in this work are well-posed even in this relaxed setting, it remains intended that all the theory of Grove and Karcher (1973); Karcher (1977), e.g., about conditions for the energy to be convex, refers to the smooth Riemannian setting.

Algorithm 1: Computation of the RCM with Newton's algorithm

```
   Input: Surface $M$, Control points $p_i$, Weights $w_i$, Warm start $q$
   Output: Riemannian center of mass $\bar{p}$
1  if $q \neq$NONE then
2      $\bar{p} \leftarrow q$
3  else
4      k $\leftarrow$ argmax$_i w_i$
5      $\bar{p} \leftarrow p_k$
6  while $\|\mathrm{grad}K(\bar{p})\| > \varepsilon$ do
7      solve the Newton equation $\mathrm{Hess}K(\bar{p})\mathbf{s} = -\mathrm{grad}K(\bar{p})$
8      $\gamma \leftarrow$ trace_geodesic($\bar{p}$, s)
9      $\bar{p} \leftarrow$ argmin$_{p\in\gamma}\|\mathrm{grad}K(p)\|$
10 return $\bar{p}$
```

4. RCM in the polyhedral metric

Under the assumption that energy K is convex, the RCM can be found by starting at an arbitrary point and descending the gradient of K. If M is smooth up to the second order, then the Hessian of K can be computed, too, and a Newton's method can be applied for faster convergence. While our input is just piecewise-smooth, we first discuss the solution in the smooth setting, then we discuss how to walk from one smooth piece to another to find the minimum.

Absil et al. (2008) provide the basic algorithms for finding the minimum of a generic convex function on a Riemannian manifold. In Algorithm 1, we adapt their Riemannian Newton's method to the computation of the RCM. While the general schema is a standard iterative walk towards the minimum, this version has the following peculiarities:

1. lines 1-5 initialize the search: if a warm start q is not provided, the search starts at the control point with the largest weight; the warm start will be crucial in the applications presented in Sec.5;

2. at lines 6 and 7, we need the gradient grad K and the Hessian Hess K, which are intended in the Riemannian sense;

3. the Newton equation at line 7 is a 2×2 linear system that takes the Hessian matrix and the gradient of K expressed in the tangent plane $T_{\bar{p}}M$ and returns a vector $\mathbf{s}$ in the same tangent plane;

4. at line 8, we need to trace a geodesic from point $\bar{p}$ of M with given tangent vector $\mathbf{v}$, i.e., solving the initial value problem (see Sec.2);

5. line 9 implements the line search algorithm (see Absil et al. (2008) 4.2); in our case, the argmin can be found while tracing the geodesic γ, hence it requires constant time after executing line 8.

From now on, we assume that M is a triangle mesh with vertex set V. In the following, we present solutions for the items 2 and 4 above, also discussing how to deal with the non-smoothness of M across edges and vertices. We assume that the distance fields sourced at the control points

$$d_{p_i}(p) = d(p, p_i)$$

for $i = 1 \ldots h$ are given at all vertices of V. These distance fields can be computed with any method at the state of the art (see Sec.2). Note that, in the applications, the RCM is evaluated many times for the same control points with varying weights, thus the cost of computing the distance fields will be amortized.

We estimate the gradient and Hessian of the distance fields of the control points $p_1, \ldots, p_h$ once when they are given. Then we compute the gradients and Hessian of energy K on-the-fly by combining them for varying weights $w_1, \ldots, w_h$.

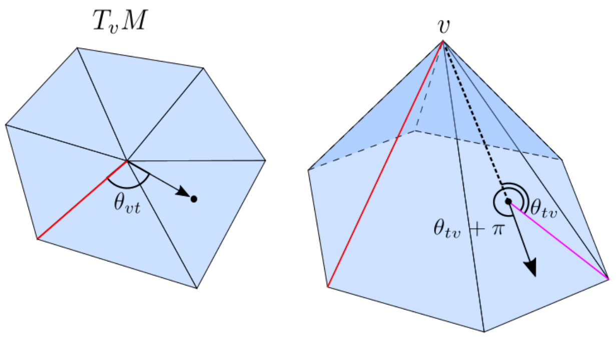


Figure 2: Discrete parallel transport by Knöppel et al. (2013). For a given triangle t incident at vertex v, we consider the centroid of t (black bullet); the red and magenta lines represent the reference directions in the tangent space T_vM of v (left) and on $t \in M$ (right), respectively. The straight line segment connecting v to the centroid of t (black arrow in T_vM and dashed line on M), form angles θ_{vt} and θ_{tv} with such reference directions, respectively. To parallel transport a vector from the tangent space of v to the tangent space of t, we rotate it by the angle $\theta_{tv} + \pi - \theta_{vt}$, where π takes care of keeping the orientation consistent when moving from one tangent space to the other. This imposes that the angle that such vector forms with the (trivial) geodesic connecting v to the centroid of t remains constant when measured in the tangent space of t.

In order to estimate such differential quantities, we endow M with a discrete differential structure, by assuming the Euclidean metric on each triangle. The tangent plane at each point in a triangle is identified with the plane of the triangle itself. The metric can be extended through the edges by unfolding the incident triangles to the same plane. At the vertices, however, an isometric unfolding is not possible in general. Under a discrete approach, a tangent plane is defined at each vertex v, by rescaling the angles of its incident triangles for a factor $2\pi/\Theta_v$, where Θ_v is the total angle about v spanned by such triangles. This construction allows us to define a discrete parallel transport. The approach proposed by Knöppel et al. (2013), which is illustrated in Fig. 2, resulted effective for our purposes; more sophisticated solutions have been proposed by Liu et al. (2016), which could be used alternatively. Note that, all quantities needed for the parallel transport can be pre-computed on the input mesh.

Given the values $f_1, \dots, f_n$ at all vertices of V of an otherwise unknown function f (a distance field from one of the control points in our case), the easiest and very popular approach to extend f to the rest of M is by piecewise-linear interpolation on triangles; this implies that the gradient of f is constant in each triangle while the Hessian is null. Such an approach cannot be applied to our problem, because the minimum of energy K would always occur at vertices of M, thus leading to discontinuous solutions for a fixed set of control points and weights $w_1, \dots, w_h$ that vary with continuity. We rather need to estimate a gradient that is continuous over the domain. To this aim, we estimate the gradient of any sampled function f at each vertex v_i based on its value at v_i and at all its neighbors v_j. We start with a constant gradient $\mathbf{v}_t$ at each triangle t, as above; then we parallel transport such vectors from each triangle t_j incident at v_i to the tangent plane $T_{v_i}M$ by means of the discrete connection; and we finally estimate the gradient at v_i by computing their average weighted by the areas of triangles:

$$\mathrm{grad} f_i = \frac{1}{\sum_{t_j \in \mathcal{N}(v_i)} A_j} \sum_{t_j \in \mathcal{N}(v_i)} A_j P_{t_j, v_i}(\mathbf{v}_{t_j}) \tag{3}$$

where A_j is the area of t_j, $\mathcal{N}(v_i)$ denotes the 1-ring of v_i, and P_{t_j,v_i} denotes the parallel transport of vectors from $T_{t_j}M$ to $T_{v_i}M$. Note that this is equivalent to estimate $\mathrm{grad} f_i$ as an average of $\mathrm{grad} f$ on $\mathcal{N}(v_i)$ by applying Gauss-Green with piecewise linear interpolation of f along its boundary. Also note that, since each $\mathbf{v}_j$ is a linear combination of the values of f at the vertices of t_j, Eq.3 is a linear combination of the values of f at the vertices of V, too, where the coefficients in linear combinations depend just on the geometry of M. Thus, the computation of the gradient for all vertices is efficiently expressed as a matrix vector multiplication $\mathrm{grad}\mathbf{f} = G_M \mathbf{f}$ where G_M is a precomputed sparse matrix that depends just on M, and $\mathbf{f} = [f_1, \dots, f_n]^T$ is the vector of values of f at the vertices of V.

Next, we linearly interpolate the gradient vectors at the vertices to obtain a piecewise-linear vector field X over M, which will we adopt as an estimate of $\text{grad} f$. To this aim, we parallel transport the vectors defined at the vertices of a triangle t to the tangent plane T_tM and we compute their affine weighted average, where weights are given by the barycentric coordinates of a generic point p in t. Let X_t denote the linear restriction of X to triangle t. Since X_t is not necessarily irrotational - as a gradient field must be - we estimate the Hessian of f inside t by first computing the Jacobian of X_t and then taking the symmetric matrix that minimizes the Frobenius distance from it. In formulas:

$$\text{Hess} f_t = \begin{pmatrix} a & \frac{b+c}{2} \\ \frac{b+c}{2} & d \end{pmatrix} \quad \text{where} \quad \begin{pmatrix} a & b \\ c & d \end{pmatrix} = J(X_t).$$

After obtaining the step $\mathbf{s}$ from line 7 of Algorithm 1, we compute the geodesic γ with the method of Polthier and Schmies (1998). The result is a polyline having its joints just at intersections with edges of M. We compute the gradient of K at all such points while tracing γ, and we select the next point $\bar{p}$ accordingly. In case $\bar{p}$ lies on an edge or at a vertex, we simply treat it as a point belonging to one of its incident triangles, using the differential properties at $\bar{p}$ as computed from such triangle. Our implementation of Polthier and Schmies (1998) deals with such corner cases while propagating the geodesic line.

Note that, since the gradient field X_t is defined analytically inside each triangle t, we can test whether it vanishes inside t, by resolving a simple 2×2 linear system. We optimize our search by adding this check each time we switch a triangle during the search, right after line 9 of Algorithm 1. In case the zero of X_t is found, we directly jump to the corresponding point and the algorithm ends.

As discussed in Sec. 3, if the control points do not lie in a strongly convex ball, the energy K may be non-convex, thus the RMC may not be well defined or unique. In these cases, pathological situations may occur, for specific combinations of weights, in which a whole valley of minima appear; small perturbations of such weights make the minimum jump from one end of the valley to the other; see, e.g. examples in Mancinelli et al. (2022). We can detect such situations by testing at each cycle whether the Hessian is positive-definite. If it is not, we just stop the search.

5. Results

We present applications to curves on a surface. Although this is certainly not exhaustive in terms of the potential applications, it involves resolving many instances of the RMC for varying weights, providing a suitable scenario to demonstrate how our method improves the state of the art.

For the computation of distance fields, we rely alternatively on the exact polyhedral algorithm VTP of Qin et al. (2016), or to a graph-based solver: VTP provides a more accurate reference, allowing us to reduce any bias from errors in the evaluation of the distance fields; the graph-based solver provides a much faster solution, allowing us to support user interaction. All experiments are executed on a Mac PowerBook M1 with 16GB memory, running on a single core.

We compare our results with results obtained with the *Vector heat* method of Sharp et al. (2019b) (henceforth VH), which is the only algorithm to date for the direct computation of the RCM on meshes, to the best of our knowledge. And we compare with the indirect and approximate *Weighted averages* method of Panozzo et al. (2013) (henceforth WA), which is based on embedding in a high-dimensional space, Euclidean distances, and Phong projection. For VH, we use the code from the *Geometry Central* library of Sharp et al. (2019a); for WA we use code provided by the authors. Note that, while our method and WA admit control points at any location of the mesh, the VH software supports just control points at vertices. For this reason, we apply this limitation in all results concerning comparisons. For the simplest case of tracing a shortest geodesic path (i.e., a linear spline), we also compare with a straightforward, yet popular, approach based on the piecewise-linear interpolation of the distance field.

Two points boundary value problem. For any two points $p, q \in M$, which are connected with a unique shortest path $\gamma_{p,q}$ with $\gamma(0) = p$ and $\gamma(1) = q$, their RCM with weights $(1 - w)$ and w is always defined and lies at $\gamma_{p,q}(w)$. Thus, the RCMs of p and q, for w varying in $[0, 1]$, can be used to trace the shortest path

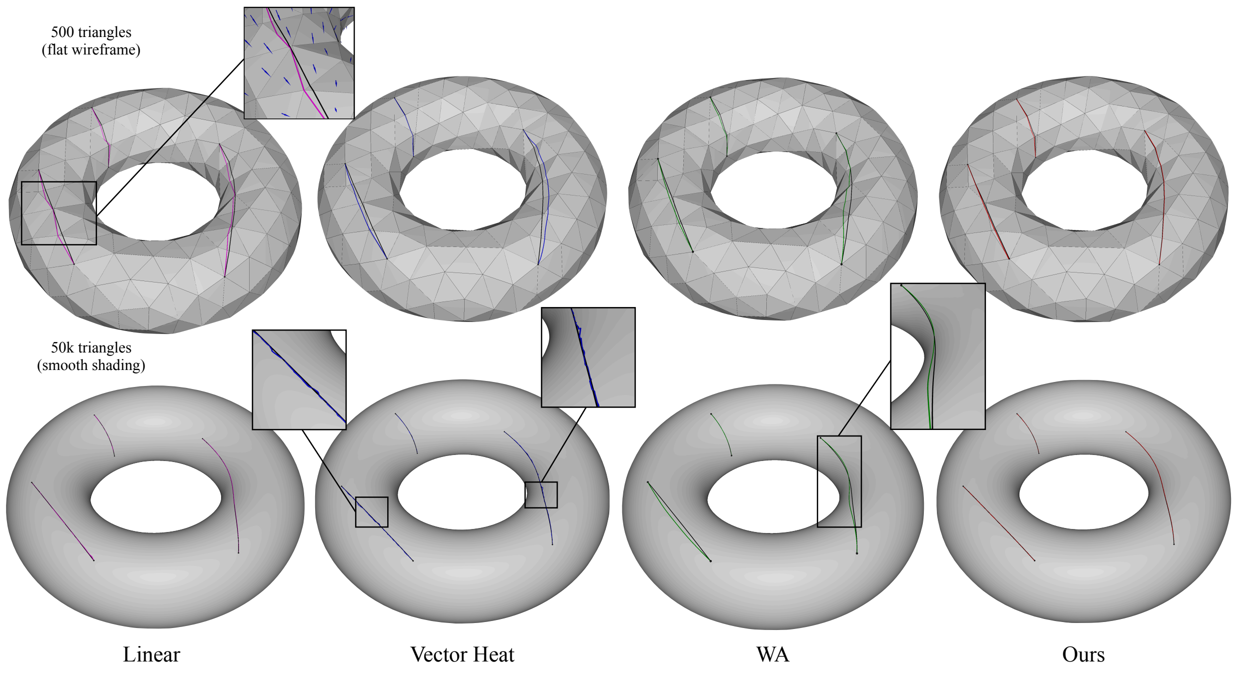


Figure 3: Shortest geodesic paths between two points on two meshes at different resolutions obtained with: gradient descent on a piecewise-linear interpolation of the distance field (magenta); RCM with VH (blue); WA (dark green); RCM with our method (red). The black line is the reference shortest path computed with an exact polyhedral solver. Blue arrows in the blow-up denote the constant gradients at triangles.

between them. Although this is certainly not the fastest way to address this problem, it provides a good test for the accuracy of our algorithm.

In Fig. 3, we compare our solution with the solutions computed with a simple descent of the piecewise-constant gradient of the distance field from p, which is linearly interpolated inside each triangle, with the RMC algorithm based on Vector heat and with the WA. For all these experiments, distance fields have been computed with the VTP algorithm of Qin et al. (2016); we also take as a reference path the solution obtained with the same solver (black lines). We present results on two meshes representing a torus: a mesh consisting of 50k triangles with a regular tessellation; and a tessellation containing just 500 triangles, which has been obtained with a drastic simplification of the former. We choose this synthetic model because it provides a smooth shape on which the trajectory of geodesics is rather intuitive.

For our method, as well as for VH, we use a warm start, which is initialized at point p in the first call of the RMC algorithm; while the solution from the previous call is used as a warm start for the next. Since the weight w is changing for small steps at each iteration, the warm start is always expected to lie very close to the solution.

The simple gradient descent presents artifacts on the coarse mesh, while it performs well on the hi-res mesh. The artifacts are mainly due to the discontinuity of the gradient field across edges. For example, it may happen that the gradient directions at two adjacent triangles point towards the edge shared by these two triangles. In this case, we are forced to follow the edge direction and restart from one of its vertices (top-left close up in Figure 3). With other datasets we have experienced drifts from the exact polyhedral solution on high resolution meshes, too, which are similar to those of VH on the coarse mesh. On the other hand, this method is extremely fast, with running times in the order of milliseconds for all curves we traced.

The solution obtained with VH is a bit unstable, producing wiggly lines in most cases. Such oscillations are probably due to an unstable estimate of directions in the log map for a moving point. Note that, VH does not aim at approximating geodesics in the polyhedral metric, but rather on an underlying smooth

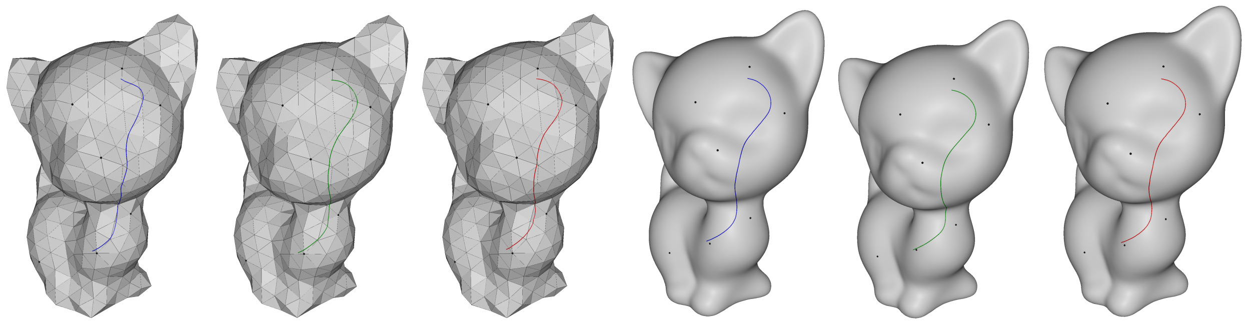


Figure 4: Splines computed with Vector heat (blue), WA(dark green) and our algorithm (red) on two different meshes representing the same object, with 1000 triangles and 50k triangles, respectively. Each spline consists of four segments, each sampled with 4096 points.

manifold. Thus, it should not be compared to the reference curve in terms of trajectory, but rather in terms of "visual smoothness". Both WA and our method produce smooth curves, but the results obtained with WA visibly deviate from the exact result in the middle portion of the curve, this behavior being consistent with the results on spline curves (see below).

In terms of performances, VH is very slow, while WA is very fast, but only after a rather expensive pre-processing time. The high computational cost of VH is due to the need of computing the log map at each point p during gradient descent, which is equivalent to estimating the distance field from p and occurs many times during curve tracing. Conversely, our method computes the distance fields from the control points just once in the beginning, while Algorithm 1 requires just local computations based on such fields during the search.

The first two rows of Table 1 show a comparison of the running times. Note that, the times for tracing in our method include also the computation of the distance fields, besides the cost of Algorithm 1. While on the coarse mesh such costs are of the same order of magnitude, on the hi-res mesh the cost of VTP dominates by one order of magnitude while the time taken by Algorithm 1 is comparable with that of WA.

Splines. We now present applications to cubic *rational Bézier* and *B-spline* curves. Spline curves in the Euclidean settings are all defined as weighted averages of a set of control points, with weights given by parametric functions, where the parameter varies in a given interval on the real line. See, e.g., Farin (2001). Spline curves on a surface may be defined by just replacing the affine weighted average with the RCM. In a nutshell, a generic spline on a manifold M can be described by equation

$$s(t) = \arg\min_{p \in M} \sum_{i=0}^{n} N_i(t) d(p, p_i)^2 \tag{4}$$

where the N_i's are cubic real functions forming a basis for the given spline scheme, the p_i are the control points of the curve, and parameter t varies on the interval of definition of the curve. Hence, $s(t)$ can be evaluated for a given value of t by resolving the RCM problem. We trace curve $s(t)$ by taking a given number of samples (4096 in our experiments) regularly distributed in its interval of definition, and evaluating $s(t)$ at each of them. Points corresponding to consecutive values of t are finally connected with shortest geodesic paths. This provides a "polygonal" approximation that converges to the curve while increasing the number of samples.

In the following, we compute all distance fields with a graph-based geodesic solver. The underlying graph is built once when the mesh is loaded and searched in all subsequent computations. The graph has one node per vertex of the mesh; each vertex is connected to the vertices in his k-ring with arcs; each arc is weighted with the corresponding polyhedral geodesic distance between its endpoints, which is computed with a variation of the algorithm of Xin and Wang (2007). We use $k = 6$ in all experiments. A field from a point p, which is not a vertex, is computed by augmenting the graph on-the-fly, with arcs that connect p to the vertices of the triangle t containing it, as well as of triangles surrounding t.

model		curve	VH		WA		ours		
name	triangles	segs.	pre (s)	trace (s)	pre (s)	trace (ms)	pre (s)	trace (ms)	update (ms)
torus	500	1	0.04	3.6–3.9	3	14	(VTP)	5–7	–
torus	50k	1	0.4	750–955	129	14	(VTP)	370	–
kitten	1k	4	0.04	31	2	29	3	19	8
kitten	50k	4	0.4	3458	133	23	22	106	41
fertility	100k	16	–	–	274	173	45	288	63
bunny	140k	17	–	–	388	51	64	905	110
armadillo	350k	23	–	–	–	–	185	2937	268
nefertiti	496k	6	–	–	2085	87	233	1374	402
lion	1M	15	–	–	–	–	456	6850	942

Table 1: Compared time performances of tracing curves with the VH method, the WA method, and ours. Sampling rate is 4096 points per curve segment. We report the size of the mesh, the number of curve segments, and the time for tracing each curve. Pre-processing times concern operations executed just once per dataset: pre-factorization of a matrix for VH; computation of the embedding for WA; and construction of the graph solver for our method (VTP does not require any pre-processing). The tracing times are given in seconds for VH and in milliseconds for WA and our method. The times for our method include the computation of the distance fields from the control points, too. The last column shows the time needed to update a curve upon dragging one of its control points. We used the VTP algorithm for the first two rows (shortest paths) and the graph solver with $k = 6$ for the remaining rows (B-splines). We did not perform experiments with VH on large meshes due to the long processing times. WA failed to produce the embedding for the armadillo and lion models.

We compare our solution with those obtained with VH and WA, by tracing curves of different length on meshes of different size. We use the same warm start, as in the previous case, for VH and our method. Again, our solution and WA are fast and have a smooth appearance, by they have slightly different trajectories; while the solutions obtained with VH are more similar to ours, but the algorithm is very slow, for the same reason outlined above. WA requires cumbersome times of pre-processing on large meshes and cannot scale beyond the sizes that we report in these experiments, while our method has no such limitations. See Fig. 4 for a visual comparison on two meshes at different resolutions and Table 1 for the running times. As before, the times reported in Table 1 for tracing a curve with our algorithm also include the computation of the distance fields from the control points, which are computed just once per curve. In the last column, we also report the time to update a curve upon dragging one of its control point p_i, which requires recomputing the distance field only for p_i as well as tracing just the portion of curve affected by p_i (four segments of curve for a cubic B-spline).

Our method can support interactive editing of control points on meshes up to several hundred thousand triangles. In Fig. 5 we show a few results on the meshes of Table 1. Note that, in some cases, generating long splines consisting of many segments may take more than one second; however, update times remain compatible with interaction in most cases. It is worth pointing out that while we present results obtained with a fine sampling rate (4096 points per segment), which is aimed at testing the robustness of the methods, a much coarser sampling would be sufficient for interactive usage. For instance, with 256 samples per segment, the curve on the lion dataset can be updated after moving one control point in just 197 ms, of which 161 ms are required to update the distance field, and roughly 10 ms to update each segment of spline with Algorithm 1.

Rational Splines. Being able of computing weighted averages directly translates in widening the range of curves that one can trace on a surface mesh. This is one of the main improvements with respect to Mancinelli et al. (2022), where the authors proposed two algorithms to trace Bézier curves on triangle meshes. Since their approach is based on subdivision schemes, it cannot be applied to rational splines. Our algorithm, on the other hand, can be used to trace both rational Bézier and B-spline curves, widening the editing possibilities of the user.

For example, we can edit a rational Bézier curve with control points $\{p_1, p_1, p_2, p_4\}$ by acting on the weights $\{w_1, w_1, w_2, w_4\}$: Fig. 6 a) shows the uniform case, i.e. $w_1 = w_2 = w_3 = w_4 = 1$, while Fig. 6 b) shows the curve obtained by setting $w_1 = w_4 = 0.05$ and $w_2 = w_3 = 5$. Recently, Ramanantoanina and

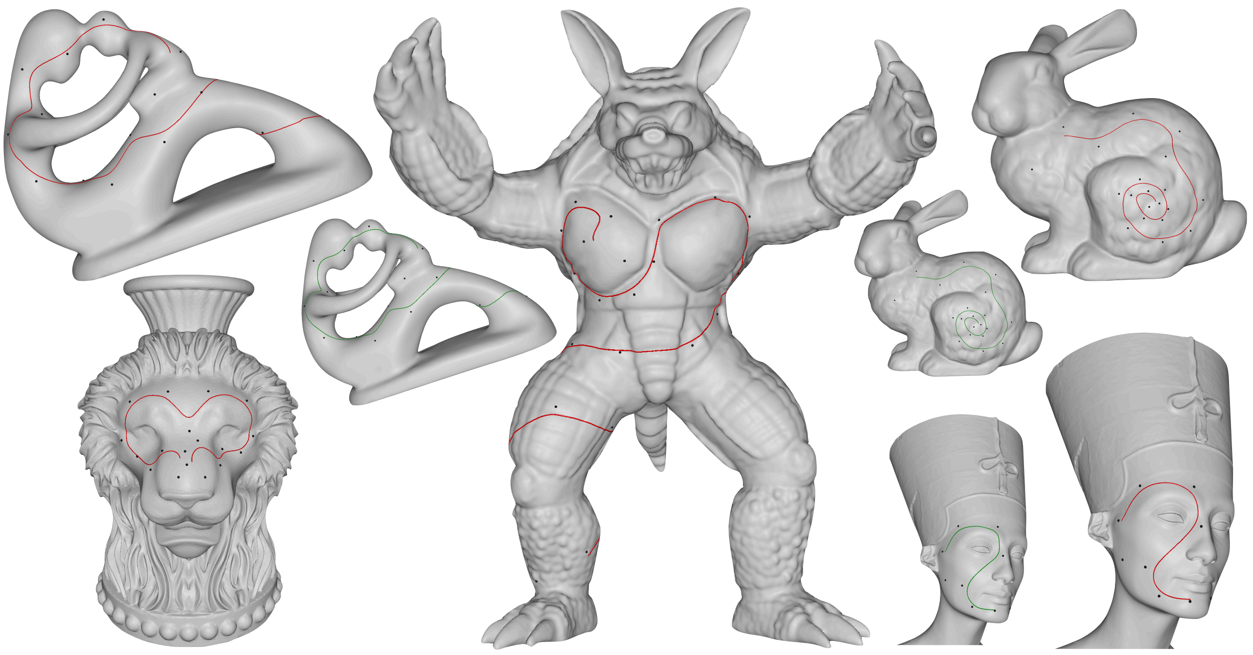
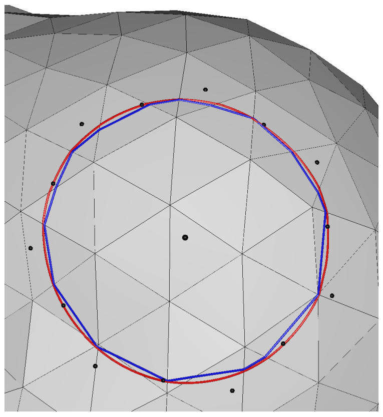


Figure 5: Splines traced with our method on a variety of models (red). In all cases, user interaction is supported. Computation times are reported in Table 1, compared with times for computing the same curves with WA (dark green).

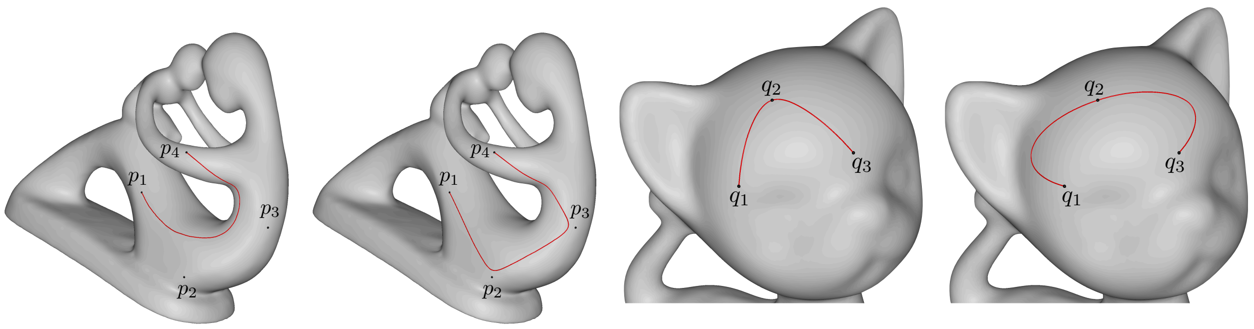
Hormann (2021) showed that expressing a rational Bézier curve in barycentric form may increase the leeway one has in terms of editing. In a nutshell, the curve is expressed as a weighted average of some *interpolation points* $q_1, \ldots, q_n$ and weights depending on $\beta_1 \ldots, \beta_n$. Figure 6 c) and d) shows two examples of these curves in the surface setting, where we set $\beta_2 = 4$ and $\beta_2 = 0.5$, respectively, obtaining results consistent with the ones in the Euclidean setting. See Figure 5 in Ramanantoanina and Hormann (2021). It should be noticed that, acting on the weights of a rational curve is way more efficient than acting on its control points, since, in the former case, no distance field needs to be recomputed.



Of course, since rational Bézier curves may be used to reproduce conic sections, it seems interesting to investigate to which extent the porting of these curves to the surface setting preserves their properties. For example, it is natural to ask if the construction of a circle with rational Bézier curves performed on a surface consists of points that are equidistant from a given point, i.e. a geodesic circle. This may be useful when one is interested in estimating the isolines of some distance field sourced at some point p. Usually, such contours are computed by detecting the edges with endpoints v_0, v_1 such that $d_p(v_0) < r$ and $d_p(v_1) > r$, where r is the desired radius, and finding the point q along such edge such that $d_p(q) = r$ through linear interpolation. The isoline is then computed by connecting with straight line segments the points thus obtained. Since this procedure assumes the linearity of the geodesic distance, it performs poorly on coarse meshes (blue line in the inset). On the other hand, the curve obtained by tracing circular arcs with our algorithm seems to produce a much smoother result.

5.1. Limitations

As already outlined, the RCM has inherent limitations from the requirement that all control points must lie inside a convex ball. Mancinelli et al. (2022) studied the Bézier curves on surfaces, showing that the RCM



a)$w_1 = w_2 = w_3 = w_4 = 1$ b)$w_1 = w_4 = 0.05, \;\; w_3 = w_4 = 5$ c)$\beta_1 = \beta_3 = 1, \;\; \beta_2 = 4$ d)$\beta_1 = \beta_3 = 1, \;\; \beta_2 = \frac{1}{4}$

Figure 6: Example of a rational Bézier curves traced with our methods: a)uniform rational cubic Bézier curve, b)cubic rational Bézier curve with weights $w_1 = w_4 = 0.05$ and $w_2 = w_3 = 5$, c) quadratic rational Bézier curve in barycentric form with weights $\beta_1 = \beta_3 = 1$ and $\beta_2 = 4$, d) same curve of c) with $\beta_2 = \frac{1}{4}$. For more detail about the definition of c) and d), we refer to Ramanantoanina and Hormann (2021).

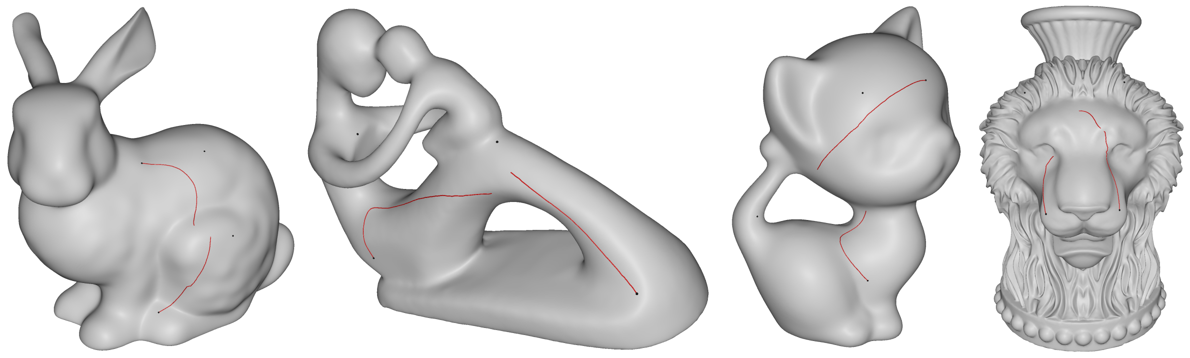
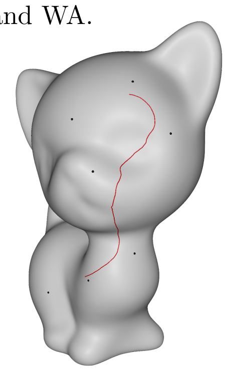


Figure 7: Examples of failure: broken curves are generated when control points are too far from each other. The same limitations apply to the other methods, too.

can fail, producing broken curves, if the control points are too far from each other. This is independent of the method used to compute the RCM: indeed, they show failure cases with both VH and WA.



On a sufficiently smooth surface, since each segment of a spline is determined by few consecutive control points, the RCM is a viable solution as long as consecutive control points are placed relatively close to each other, as in the examples shown above. However, on a relatively rough surface, convex balls may become tiny, to the point of coinciding with the 1-ring of vertices. While we have experienced quite some resiliency of our method to the roughness of surfaces (see, e.g., the armadillo in Fig. 5), trying to resolve the RCM when the mutual distance of the control points is large inevitably leads to unstable results. Fig. 7 shows some examples of failure. Concerning specific limitations of our algorithm, we have experienced some sensitivity to the accuracy of the distance fields and, especially, of their differential quantities. We have obtained the most stable results by working on meshes with fairly regular triangles, and computing the distance fields with VTP, which is however too slow to support interaction. We have achieved a good trade-off between accuracy and performance in the evaluation of the distance fields by using a graph solver, where each vertex is connected with exact distances to all the vertices in its k-ring, by setting $k = 6$. This graph has a moderate cost of pre-processing and is fast enough to traverse when computing the distance fields. As a comparison, the inset shows how a less accurate estimate of the distance field, computed on a graph with $k = 2$, affects the same curve of Figure 4 on the 50k kitten.

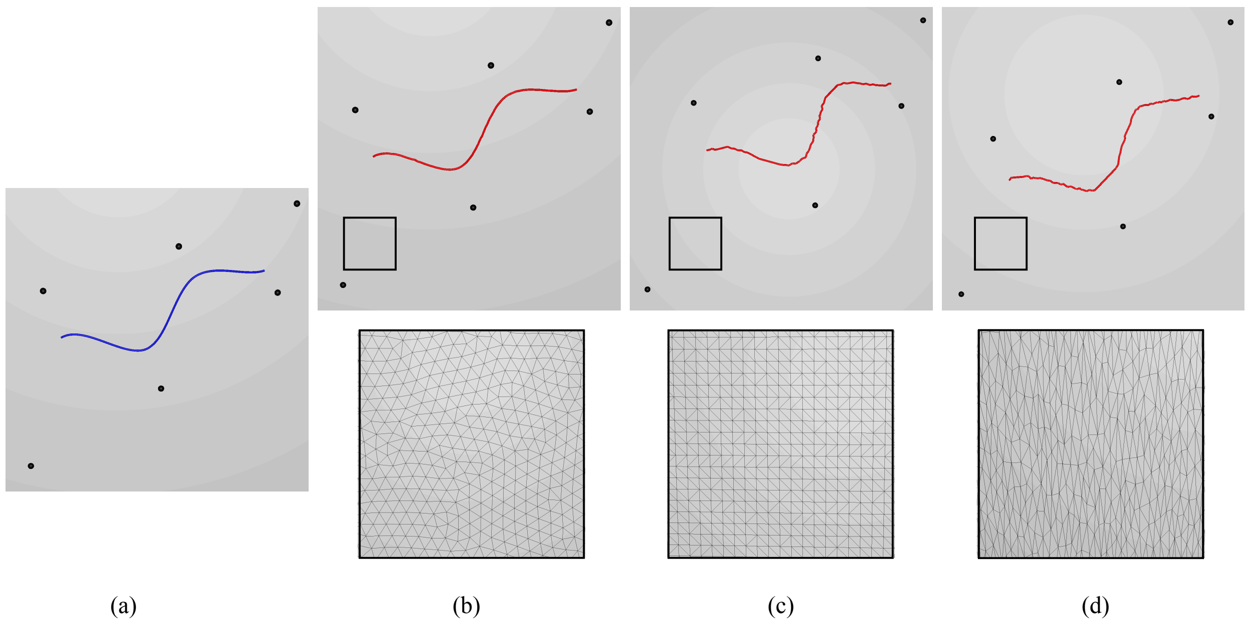
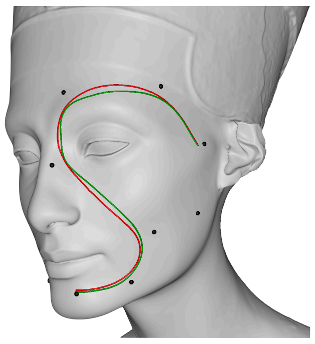


Figure 8: Sensitivity to the meshing: a cubic B-spline on the plane, ground truth (a); our results: on a Delaunay tessellation (b); on a triangulated regular grid (c); and on a highly anisotropic tessellation (d). While we perform well on the Delaunay mesh, our results may become rather unstable on highly biased meshes, producing wiggly curves.

The quality of the mesh may severely affect the local method that we use to estimate the gradient field. In Figure 8, we compare the ground truth of a spline computed in the plane, with results computed with our method on three different tessellations of the plane. On a Delaunay tessellation from a fairly regular distribution of points, our algorithm produces a result consistent with the ground truth. On a regularly triangulated quad grid, and on a highly anisotropic tessellation, however, the results become unstable, producing wiggly curves. This is due to a poor resiliency of the gradient estimator of Eq. (3) to these tessellations, in which the 1-ring of each vertex has an uneven, biased distribution. In fact, the average technique we use to estimate the gradient is very sensitive to the local geometry around a vertex, especially near critical points of the underlying distance field (see, e.g., ?). This is particularly true on anisotropic meshes, where the algorithm may even fail detecting the minimum at some steps, thus produces broken curves. A more robust gradient estimator may make our method more resilient to bad meshes with irregular 1-rings and skinny triangles. We plan to investigate this more thoroughly in future works.

5.2. Discussion



Our method clearly demonstrates superior performances compared to VH, which is the only other direct method for computing the RCM. WA, on the other hand, is fast and generates visually pleasing results in most cases, but it provides just a heuristic approximation of the RCM while it cannot guarantee that an embedding can be found, in which the Euclidean distance can approximate the original geodesic distance well enough. Therefore, it provides no control on the correctness of the result. In fact, the embedding procedure fails with some meshes, because no acceptable embedding can be found. This can be clearly seen in Fig. 3, where the geodesics computed with WA are quite different from the ones computed with the exact polyhedral solver. The inset shows an example on spline curves from Fig. 5, where we plot the curve obtained with WA (dark green) together with our results (red). Note how our curve follows a smoother trajectory. Moreover, the pre-processing times of WA are high, and the method can hardly scale to large meshes also because of the large memory footprint used during pre-processing.

6. Concluding remarks

We have presented a direct method for the numerical computation of the Riemannian Center of Mass on a triangle mesh. Our method is based on a discrete Riemannian Newton's method, using a piecewise-linear estimate of the gradient of the distance fields from the control points, and a piecewise-constant estimate of their Hessian. Our method returns stable results under the conditions of existence and uniqueness of the RCM, beating the only other existing direct method in terms of both accuracy and time performance. Coupled with a geodesic graph solver, our RCM algorithm provides a viable solution to support the interactive design of splines also on large meshes; while an exact geodesic solver may offer the best quality for final rendering at the cost of some more time. We applied our algorithm to spline tracing, supporting both rational Bézier and B-spline curves. Under these premises, we believe that the RCM may become an effective and efficient tool for geometric design and geometry processing by working directly in the intrinsic metric of manifolds. Other applications, e.g., texture mapping and texture transfer, involve resolving also the inverse problem, i.e., given a generic point, find its weights (coordinates) with respect to the given control points. Rustamov (2010) provided a solution for this problem, which is however quite unstable if the control points are many and distributed over a large area, as landmarks for texture transfer should be. For this reason, more investigation is needed to find a viable solution for these applications, too.

Our current algorithm is still sensitive to the quality of the input mesh, especially for the specific differential estimator we use. We believe that a more robust differential estimator, possibly relying on a smooth estimate of the underlying surface, could produce results much more stable and resilient to the quality of the underlying mesh. We plan to address such topics in our future work.

References

Absil, P.A., Mahoney, R., Press, R.S., 2008. Optimization Algorithms on Matrix Manifolds. Princeton University Press.

Adikusuma, Y., Fang, Z., He, Y., 2020. Fast Construction of Discrete Geodesic Graphs. ACM Trans. Graph, 39, 1–14.

Bose, P., Maheshwari, A., Shu, C., Wuhrer, S., 2011. A survey of geodesic paths on 3d surfaces. Computational Geometry 44, 486–498. URL: https://www.sciencedirect.com/science/article/pii/S0925772111000459.

Chen, J., Han, Y., 1990. Shortest paths on a polyhedron, in: Proceedings of the Sixth Annual Symposium on Computational Geometry, Association for Computing Machinery, New York, NY, USA. pp. 360–369. URL: https://doi.org/10.1145/98524.98601.

Crane, K., Livesu, M., Puppo, E., Qin, Y., 2020. A survey of algorithms for geodesic paths and distances. arXiv:2007.10430.

Crane, K., Weischedel, C., Wardetzky, M., 2013. Geodesics in heat: A new approach to computing distance based on heat flow. ACM Trans. Graph. 32.

Farin, G., 2001. Curves and Surfaces for CAGD: A Practical Guide. 5th ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Grove, K., Karcher, H., 1973. How to conjugate c1-close group actions. Mathematische Zeitschrift 132, 11–20.

Karcher, H., 1977. Riemannian center of mass and mollifier smoothing. Communications on Pure and Applied Mathematics 30, 509–541.

Kimmel, R., Sethian, J.A., 1998. Computing geodesic paths on manifolds. Proceedings of the National Academy of Sciences of the United States of America 95, 8431–8435.

Knöppel, F., Crane, K., Pinkall, U., Schröder, P., 2013. Globally optimal direction fields. ACM Trans. Graph. 32.

Liu, B., Tong, Y., Goes, F.D., Desbrun, M., 2016. Discrete connection and covariant derivative for vector field analysis and design. ACM Transactions on Graphics 35, 1–17.

Mancinelli, C., Nazzaro, G., Pellacini, F., Puppo, E., 2022. b/surf: Interactive bézier splines on surfaces. IEEE Transactions on Visualization and Computer Graphics doi:10.1109/TVCG.2022.3171179.

Mitchell, J.S.B., Mount, D.M., Papadimitriou, C.H., 1987. The discrete geodesic problem. SIAM J. Comput. 16, 647–668.

Nazzaro, G., Puppo, E., Pellacini, F., 2022. geoTangle: interactive design of geodesic tangle patterns on surfaces. ACM Trans. Graph. 41, 12:1–12:17.

Panozzo, D., Baran, I., Diamanti, O., Sorkine-Hornung, O., 2013. Weighted averages on surfaces. ACM Trans. Graph. 32, 60:1–12.

Polthier, K., Schmies, M., 1998. Mathematical visualization, Springer. pp. 135–150.

Qin, Y., Han, X., Yu, H., Yu, Y., Zhang, J., 2016. Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. ACM Trans. Graph. 35, 125:1–125:13.

Ramanantoanina, A., Hormann, K., 2021. New shape control tools for rational bézier curve design. Computer Aided Geometric Design 88, 102003.

Rustamov, R.M., 2010. Barycentric coordinates on surfaces. Computer Graphics Forum 29, 1507–1516.

Sharp, N., Crane, K., 2020. You can find geodesic paths in triangle meshes by just flipping edges. ACM Trans. Graph. 39, 249:1–15.

Sharp, N., Crane, K., et al., 2019a. geometry-central. Www.geometry-central.net.

Sharp, N., Soliman, Y., Crane, K., 2019b. The vector heat method. ACM Trans. Graph. 38, 1–19.

Xin, S.Q., Wang, G.J., 2007. Efficiently determining a locally exact shortest path on polyhedral surfaces. CAD Computer Aided Design 39, 1081–1090.

Yuksel, C., Lefebvre, S., Tarini, M., 2019. Rethinking Texture Mapping. Comp. Graph. Forum 38, 535–551.