b/Surf: Interactive Bézier Splines on Surface Meshes

Claudio Mancinelli, Giacomo Nazzaro, Fabio Pellacini, and Enrico Puppo

Abstract—We present a practical framework to port Bézier curves to surfaces. We support the interactive drawing and editing of Bézier splines on manifold meshes with millions of triangles, by relying on just repeated manifold averages. We show that direct extensions of the de Casteljau and Bernstein evaluation algorithms to the manifold setting are fragile, and prone to discontinuities when control polygons become large. Conversely, approaches based on subdivision are robust and can be implemented efficiently. We implement manifold extensions of the recursive de Casteljau bisection, and an open-uniform Lane-Riesenfeld subdivision scheme. For both schemes, we present algorithms for curve tracing, point evaluation, and approximated point insertion. We run bulk experiments to test our algorithms for robustness and performance, and we compare them with other methods at the state of the art, always achieving correct results and superior performance. For interactive editing, we port all the basic user interface interactions found in 2D tools directly to the mesh. We also support mapping complex SVG drawings to the mesh and their interactive editing.

Index Terms—geometric meshes, spline curves, user interfaces, geometry processing.

1 Introduction

Vector graphics in 2D is consolidated since decades, as is supported in many design applications, such as Adobe Illustrator [1], and languages, like Scalable Vector Graphics (SVG) [2]. Bézier curves are the building blocks of most vector graphics packages, since most other primitives can be converted into *Bézier splines* (chains of Bézier curves) and edited as such [3].

In many design applications, it would be beneficial to edit vector graphics directly on surfaces, instead of relying on parametrization or projections that have inherent distortions [4], [5]. Yet, bringing vector graphics to surfaces is all but trivial, since basic rules of Euclidean geometry do not hold under the geodesic metric on manifolds; and distances, shortest and straightest paths cannot be computed in closed form. In particular, in spite of several attempts to define curves under the geodesic metric, a complete computational framework that supports their practical usage in an interactive design setting is still missing.

In this work, we present the first *practical* method, which supports the *interactive* and *robust* design and editing of Bézier splines on high resolution meshes, *without any limitation on the position of their control points.* For the sake of simplicity, we restrict our study to cubic Bézier curves. Our contributions tackle different aspects of the problem, as outlined in the following.

Curve schemes in the manifold setting (Sec. 3): To the best of our knowledge, all existing extensions of Bézier curves to the manifold setting have been proven to work just "in the small", i.e., when control points are sufficiently close to one another. However, such limitation is incompatible with a practical usage. We show that, indeed, the manifold extensions of the de Casteljau and Bernstein evaluation algorithms may fail and lead to discontinuous curves for general sets of control points. On the other hand, we show that some existing subdivision schemes may be generalized to manifolds "in the large", too. We show that the recursive de Casteljau (RDC) subdivision scheme proposed by Noakes [7] indeed works for any set of control points, always producing C^1 curves. And we propose a manifold extension of an open-uniform Lane Riesenfeld (OLR) subdivision scheme; we elaborate on results of Duchamp et al. on the manifold extension of the standard Lane Riesenfeld scheme for B-splines [8], to show that our OLR scheme is the first one to produce C^2 Bézier segments in the manifold setting. Curves from both the RDC and the OLR schemes can be joined with C^1 continuity to form Bézier splines.

Algorithms (Sec. 4): We provide the basic tools for curve design and rendering with the RDC and OLR subdivision schemes. To the best of our knowledge, all previous proposals in the manifold setting provided only algorithms for curve tracing, i.e., to produce a discrete approximation of the curve. Besides such algorithms, we give the algorithms for evaluating the curve at a given parameter; and for splitting a curve at a given point, by approximating a single Bézier segment with a spline of two segments.

Computational framework and system (Sec. 5): While the curve schemes and related algorithms are defined on smooth manifolds, our implementation addresses triangulated surfaces. As remarked by Wallner and Pottmann [9], *"after discretization the question of smoothness does no longer make sense in the strict mathematical sense. Even so, it is important to know that ... the ideal geometric object one tries to approximate is smooth."* All our algorithms rely on the computation of repeated manifold averages, which involve finding geodesic shortest paths. Such computations are known to converge to the equivalent measures in the smooth setting as the geometric mesh is refined, e.g., through subdivision [10]. In order to target meshes with millions of triangles, we need a very efficient framework for geodesic computations. We develop an algorithm for computing locally shortest paths that is robust and beats the performances of the other methods at the state of the art. In particular, we greatly improve the step to find an initial guess,

- *Claudio Mancinelli and Giacomo Nazzaro are joint first authors.*
- *Claudio Mancinelli and Enrico Puppo are with the Department of Informatics, Bioengineering, Robotics and Systems Engineering of the University of Genoa, Genoa, Italy.*
  E-mail: enrico.puppo@unige.it, claudio.mancinelli@dibris.unige.it
- *Giacomo Nazzaro and Fabio Pellacini are with the Department of Computer Science of Sapienza University of Rome, Rome, Italy.*
  E-mail: {nazzaro, pellacini}@di.uniroma1.it

Manuscript received December XX, 2021; revised XXXXXX.

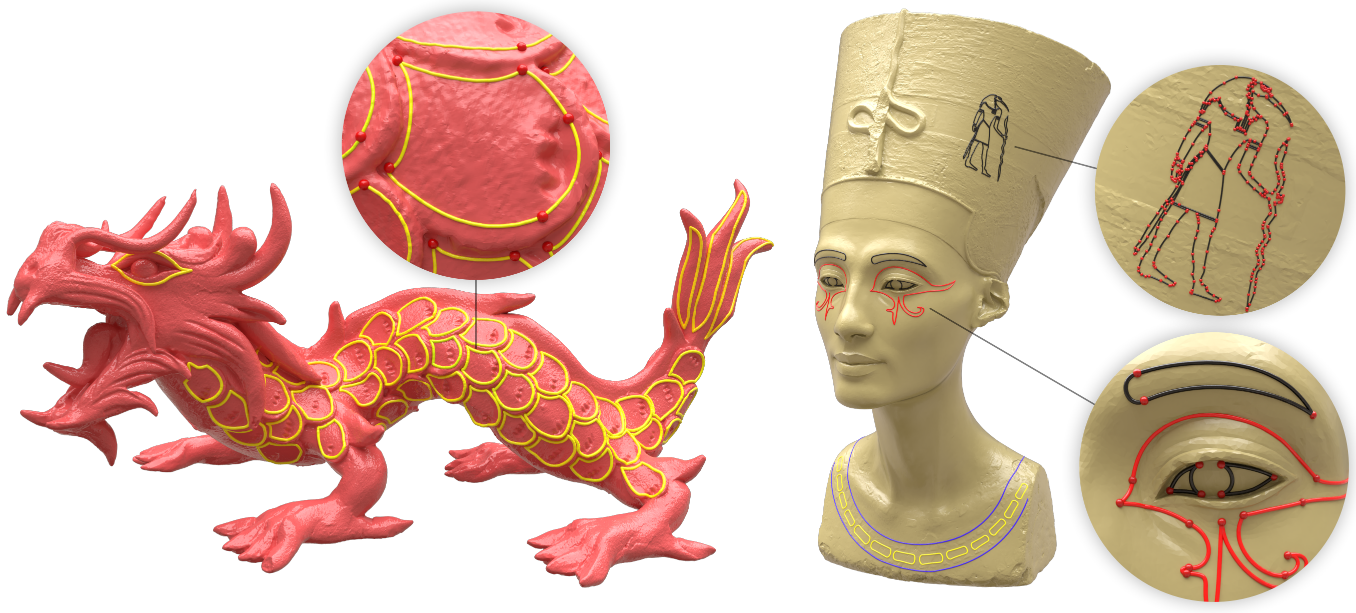

Fig. 1. We propose algorithms to interactively edit Bézier splines on large meshes, including curve editing, curve transformations and import and editing complex SVG drawings. All computations occur in the intrinsic geodesic metric of the surface. All splines in this figure have been drawn interactively. Control points and tangents of curves under editing are shown in the zoomed insets. Asian Dragon ∼7.2M triangles; Nefertiti ∼500K triangles.

which is the bottleneck of all local methods.

We integrate our algorithms in a user interface, thus providing the first prototype system that supports the robust interactive design of Bèzier splines on manifold meshes for any choice of control points. We support all basic operations that 2D editors have, including: click-and-drag of control points and tangents; point insertion and deletion; and translation, rotation and scaling of curves. We also support mapping of 2D SVG drawings onto the surface. Fig. 2 shows the interface of our system with some simple curves traced on a model. The supplemental video shows a full editing session. Our system remains interactive on meshes made of millions of triangles, such as the ones shown in Fig. 1.

Assessment and comparisons (Sec. 6): To assess the robustness and performance of our algorithms, we trace curves on the more than five-thousands watertight, manifold, meshes of the Thingi10k repository [11] with one hundred randomly generated control polygons for each mesh. Our algorithms handle all cases well. We discuss the sensitivity of our algorithms to the input mesh, and how to deal with critical meshes containing nearly degenerate triangles. We run an extensive comparison to the methods at the state of the art, in terms of both robustness and time performance, consistently beating their results.

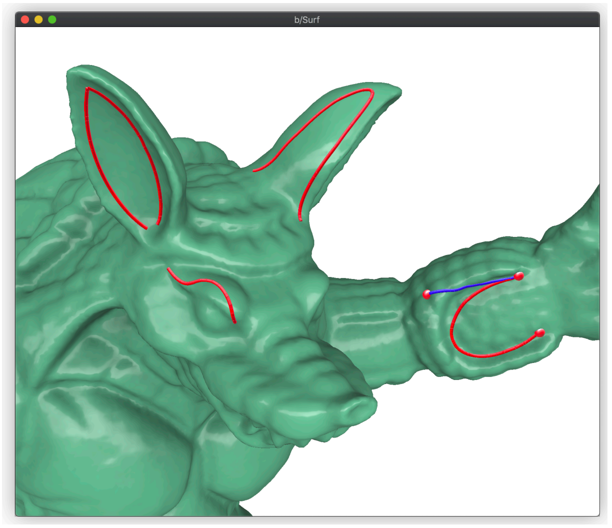

Fig. 2. The GUI of our system. Curves on the eye and on the arm consist of a single cubic segment each, while the two splines on the ears consist of two cubic segments each, with a sharp corner to the left and smooth junction to the right. A control tangent (in blue) is depicted on the curve under editing.

2 Related work

The design of spline curves on manifolds has been addressed by several authors, both from a mathematical and from a computational perspective. We review only methods addressing general surfaces.

A traditional approach to circumvent the problems of the Riemannian metric consists of linearizing the manifold domain via parametrization, designing curves in the parametric plane, and mapping the result to the surface. Parametrization introduces seams, and drawing lines across them becomes problematic. Moreover, distortions induced by parametrizations are hard to predict and control. The exponential map can provide a local parametrization on the fly for the region of interest [12], [13], [14], [15], [16]. However, its radius of injectivity can be small (e.g., in regions of high curvature), while control polygons and curves may extend over large regions. Even curves as simple as the ones depicted in Fig. 3 may be hard to control using either local or global parametrizations.

As reported in [9], another common approach consists of relaxing the manifold constraint, resolving the problem in Euclidean

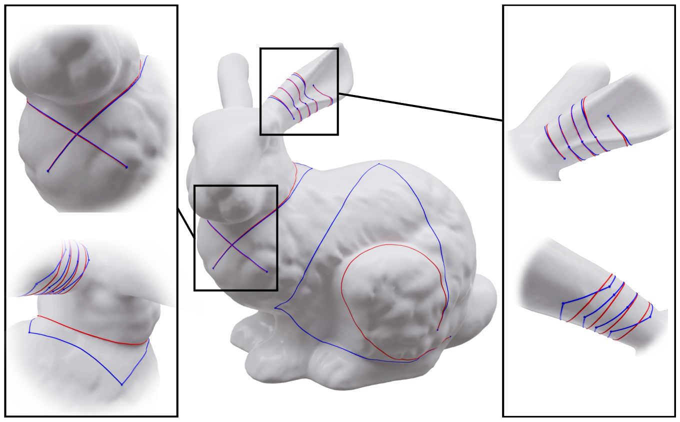

Fig. 3. Curves that wind about the object or require large control polygons may be challenging to draw with an approach based on parametrization. The collar and the curl consist each of a single cubic segment, while the spiral is a spline of four segments joined with smooth (C^1) continuity. Control polygons are depicted in blue.

space, and projecting the result back to the surface. Panozzo et al. [17] use an embedding in a higher-dimensional Euclidean space, followed by Phong projection. These methods may support user interaction, but they provide only approximate results, are prone to artifacts, and are hard to scale to large meshes.

The design of curves can also be addressed as an optimization problem in a variational setting. Noakes et al. [18] and Camarinha et al. [19] provide the basic variational theory of splines on manifolds. This approach is adopted in several other papers [20], [21], [22], [23], [24], [25], [26]. While most such works do not address implementation and performance, [23] and [24] eventually resort to projection methods. Overall, the variational approach is too computationally expensive to support user interaction on large meshes. Moreover, these curves are harder to control interactively than traditional Bézier splines.

Concerning the specific case of Bézier curves, Park and Ravani [27] first extended the de Casteljau algorithm to Riemannian manifolds, without developing the computational details. Later on, the de Casteljau algorithm on surfaces has been explored by several other authors [22], [28], [29], [30], [31]. Among these, Morera et al. [29] extend the recursive de Casteljau bisection, and Sharp et al. [32] achieve interactive performance on the same algorithm, by using a fast method for evaluating locally shortest geodesic paths [33]. We adopt the same structure of [29] for curve tracing with the recursive de Casteljau (RDC) subdivision. In Section 6.3, we further discuss the method of [32], [33] and compare their results and performances with our method. Absil et al. [34] define Bézier curves both with the de Casteljau algorithm and with the Riemannian center of mass (RCM), and show that they may produce different results. A method for the direct computation of the RCM through gradient descent has been proposed in [35], which is computationally intensive, though. Conversely, the method proposed in [17] is very efficient (after pre-processing), but provides just an approximation of the RCM. In Section 6.3, we compare to both such methods in terms of robustness and performance.

Several authors have investigated the theoretical aspects of the subdivision approach to splines in the manifold setting. We refer to Wallner [36] for a detailed analysis, reporting just the results most relevant to our work. Noakes [7] proves that the recursive de Casteljau subdivision converges and produces a C^1 curve in the cubic case, subject to strong constraints on the control polygon. Most recent results [8], [37], [38] focus on Lane-Riesenfeld schemes and show that a scheme of order k is convergent and C^k in the manifold and functional settings. These latter works motivate our approach to the open-uniform Lane-Riesenfeld (OLR) subdivision. We exploit observations reported in [36] to show that such schemes can be generalized to work on any control polygon.

3 Bézier Curves on Manifolds

We consider different constructions of Bézier curves, all of which produce the same curves in the Euclidean setting, and we analyze their extensions to the manifold setting. We provide just the basics of each construction, referring the reader to [3], [39] for further details. We assume the reader to be familiar with the geodesic metric on manifolds; details can be found in any book about Riemannian geometry [40]. All definitions are given in general, while results are given just for cubic Bézier curves, for the sake of simplicity; extensions to curves of a different order are just outlined. In the following, we will denote with M a smooth, compact and connected surface embedded in $\mathbb{R}^3$, endowed with the Riemannian metric induced by the embedding.

3.1 Preliminaries and notations

In the Euclidean setting, a Bézier curve is the image of a polynomial parametric function of degree k

$$\mathbf{b}^k : [0,1] \longrightarrow \mathbb{R}^d,$$

which is defined by means of a *control polygon* $\Pi = (P_0, \dots, P_k)$, where all $P_i \in \mathbb{R}^d$. Curve $\mathbf{b}^k$ interpolates points P_0 and P_k, and it is tangent to Π at them. All constructions of Bézier curves in the Euclidean setting rely on the computation of *affine averages* of points of the form

$$\bar{P} = \sum_{i=0}^{h} w_i P_i \tag{1}$$

where the w_i are non-negative weights satisfying the partition of unity. For $h = 1$, the affine average reduces to linear interpolation

$$\bar{P} = (1-w)P + wQ. \tag{2}$$

By analogy with the Euclidean setting, a control polygon Π in the manifold setting consists of a polyline of shortest geodesic paths, connecting the control points that lie on M.

Affine averages are not available on manifolds, but they can be substituted with the Riemannian center of mass [41], [42]. Given points $P_0, \dots, P_h \in M$ and weights $w_0, \dots, w_h$, their *Riemanninan Center of Mass* (RCM) on M is defined

$$RCM(P_0, \dots, P_h; w_0, \dots, w_h) = \operatorname*{argmin}_{P \in M} \sum_{i=0}^{h} w_i d(P, P_i)^2 \tag{3}$$

where $d(\cdot,\cdot)$ is the geodesic distance on M. If M is a Euclidean space, then the solution to Eq. 3 is the usual affine average of Eq. 1.

The RCM requires that Eq. 3 has a unique minimizer. Karcher [42] provides a condition of existence and uniqueness of the solution, which requires all points P_i to be contained inside a strongly convex ball, whose maximum radius depends on the curvature of M. In the following, we will refer to this condition as the *Karcher condition*. If such condition is satisfied, then the RCM is smooth in both the P_i's and the w_i's [43]. Unfortunately, the Karcher condition restricts the applicability of the RCM to relatively small neighborhoods in the general case.

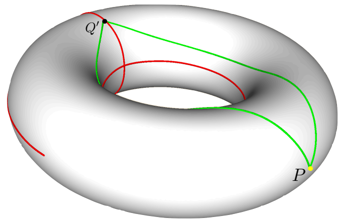

Fig. 4. The cut locus $C(P)$ of a point P on a torus (red). For a point Q' on $C(P)$ there exist two different shortest geodesics joining P to Q' (green).

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 $\gamma_{P,Q}(w)$. This provides the analogous of the affine average of Eq. 2 for pairs of points that do not lie on each other's cut locus [9].

3.2 Extension of the weighted average

Figure 4 shows an example of cut locus of a point P lying on a torus. If point Q lies on the cut locus of P, then there is ambiguity on which shortest path should be taken to compute their average. We extend the pairwise average to the cut locus, too, by picking one *arbitrary,* but *deterministically selected,* shortest path connecting P to Q. We thus define the *manifold average between two points*

$$\mathscr{A} : M \times M \times [0,1] \longrightarrow M; \quad (P, Q; w) \mapsto \gamma_{P,Q}(w) \tag{4}$$

where $\gamma_{P,Q}$ is a (deterministically selected) shortest geodesic path joining P to Q. We have that $\mathscr{A}(P, Q; w) = RCM(P, Q; (1-w), w)$ as long as P and Q do not lie on each other's cut locus; while at the cut locus it returns a point, which depends on the selected shortest path $\gamma_{P,Q}$. The averaging operator of Eq. 4 provides the analogous of Eq. 2 in the manifold setting for *any* pair of points.

Notice that, the operator $\mathscr{A}$ remains smooth everywhere in its w parameter, but it fails to be continuous at pairs $(P, Q) \in M \times M$ that lie on each other's cut locus. Such a discontinuity may affect the manifold constructions, as we will see in the following.

3.3 de Casteljau point evaluation

The de Casteljau construction provides a recursive definition, which evaluates a Bézier curve as $\mathbf{b}^k(t) = \mathbf{b}_0^k(t)$, where

$$\begin{aligned} \mathbf{b}_i^0(t) &= P_i \\ \mathbf{b}_i^r(t) &= (1-t)\mathbf{b}_i^{r-1}(t) + t\mathbf{b}_{i+1}^{r-1}(t) \end{aligned} \tag{5}$$

for $r = 1, \ldots, k$ and $i = 0, \ldots, k-r$. A curve is traced by computing Eq. 5 for t varying in $[0,1]$.

This construction can be extended to the manifold setting in a straightforward way by substituting the affine averages between pairs of points with the manifold average $\mathscr{A}$ defined above. This extension was proposed first by Park and Ravani [27]. As shown by Popiel and Noakes [31], if all consecutive pairs of points in the control polygon Π lie in a totally normal ball, then the resulting curve is smooth. However, if the constraint is violated, the resulting curve can be discontinuous. In fact, even if all shortest geodesics paths in Π are unique, some pairs of intermediate points involved in the construction may lie on each other's cut locus, for some value of the parameter t. As t passes such critical value, the manifold average $\mathscr{A}$ returns a discontinuous result, thus causing a discontinuity in the curve. Fig. 5(top) illustrates the construction near failure points; Fig. 6(a) provides another example of failure on a more complex shape.

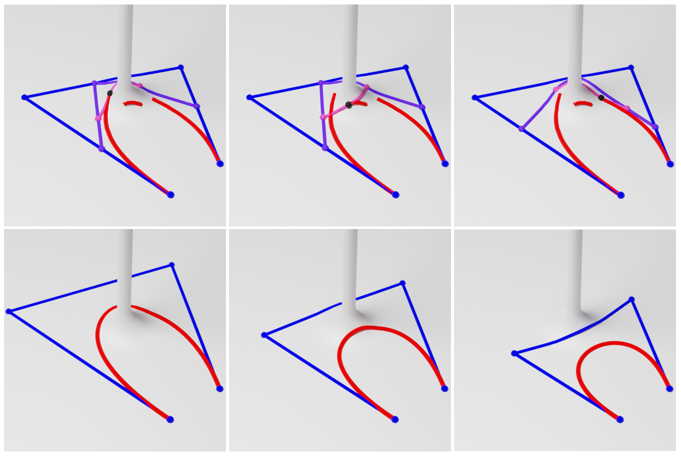

Fig. 5. Top: example of a failure case of direct de Casteljau evaluation. The black bullets at the discontinuities correspond to consecutive parameter values near a critical value, and the blue/purple/pink lines provide the de Casteljau construction. Note how the pink line jumps from one side of the pole to the other as t passes critical values, causing discontinuities. Bottom: our method always produces a smooth curve regardless of the positioning of the control points. The same control polygon of the top figure generates the curve in the center; dragging the handles we may force the curve to pass behind the pole (left) or further shrink (right). Note how the control polygon to the right also switches to the front of the pole, while leaving the smoothness of the curve unaffected.

3.4 Bernstein point evaluation with the RCM

A Bézier curve can be evaluated in closed form as an affine sum of all its control points:

$$\mathbf{b}^k(t) = \sum_{i=0}^{k} B_i^k(t) P_i \tag{6}$$

where the $B_i^k(t)$ are the Bernstein basis polynomials of degree k

$$B_i^k(t) = \binom{k}{i} t^i (1-t)^{n-i}.$$

This expression can be rewritten for the manifold case as

$$\mathfrak{b}^k(t) = RCM(P_0, \ldots, P_k; B_0^k(t), \ldots, B_k^k(t)) \tag{7}$$

where the Riemannian center of mass RCM has been defined in Eq. 3. Again, a curve is traced by computing Eq. 7 for t varying in $[0,1]$. This construction was addressed in [17], where an approximation of the RCM is proposed, which is based on an embedding in a higher dimension and Phong projection (see Sec. 6.3 for further details). A direct evaluation of the RCM is also possible through gradient descent on the energy of Eq. 3. A method has been proposed in [35], which requires computing a log map at each iteration, though.

If the control points are close enough to fulfill the Karcher condition, then the resulting curve is smooth, since both the RCM and the Bernstein polynomials are smooth. However, if the Karcher condition is not fulfilled, then the energy in Eq. 3 is no longer guaranteed to be convex, and it might even have infinitely many minima. In this case, the curve may be undetermined at some intervals. Fig. 6(b) provides an example of failure, where the RCM has been computed directly by gradient descent. Note that, the failure is independent of the method used to implement the RCM, being intrinsic to the non-convexity of the energy for some values of the weights. More examples of failure for the method of [35] and for the approximation of [17] are demonstrated in Section 6.3.

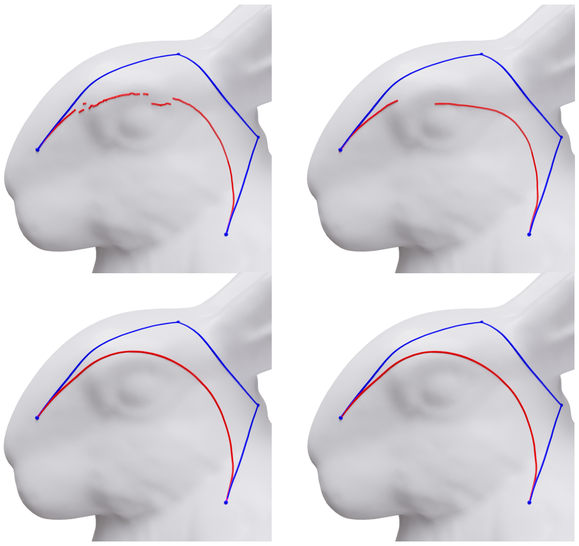

Fig. 6. An example of failure in tracing a curve with the direct de Casteljau (*top-left*), and the RCM evaluation (*top-right*). The same control polygon gives two smooth and nearly identical curves with the Recursive de Casteljau (*bottom-left*) and the Open-uniform Lane-Riesenfeld schemes (*bottom-right*) described in Sections 3.5 and 3.6, respectively.

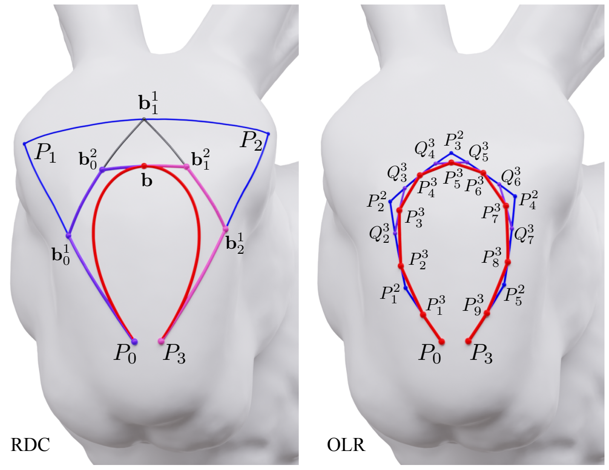

Fig. 7. The constructions at the basis of the RDC and OLR schemes for the same control polygon for the cubic case. *RDC (left)*: The control polygon (blue) is split into a chain of two control polygons (purple and pink) by computing three shortest geodesic paths. The limit curve is depicted in red. Here we show only the first subdivision. *OLR (right)*: One step of subdivision from Π^2 (blue) to Π^3 (red polygon). The even points P^3_{2j}, as well as the intermediate points Q^3_i lie on segments of Π^2 and are evaluated first. The evaluation of each odd point P^3_{2j+1} requires computing one shortest geodesic path (purple). This construction corresponds to one midpoint subdivision followed by two steps of smoothing by averaging consecutive points.

3.5 Recursive de Casteljau subdivision (RDC)

One step of the de Casteljau construction subdivides polygon Π into two control polygons Π_L and Π_R. See Fig. 7 (RDC) for an example. The junction point of Π_L and Π_R lies on the curve. The recursive application of this procedure for $t = 1/2$ defines a sequence of subdivision polygons Π^n_{DC}, which converges to the Bézier curve.

In the manifold setting, this algorithm produces a curve, which is *different* from the one obtained with the direct evaluation by varying the value of parameter t, as reviewed in Sec.3.3. This scheme in the manifold setting was studied first by Noakes [7], and implementations were proposed in [29], [32]. Noakes proved that this subdivision converges to a C^1 limit curve, provided that the initial control points lie in a convex set [7]. We extend this result to any set of control points.

Proposition 3.1. *For any given control polygon $\Pi = (P_0, P_1, P_2, P_3)$, the RDC subdivision implemented with the $\mathscr{A}$ operator converges to a limit curve that is C^1 continuous.*

Proof. As observed by Wallner [36] (Sec. 2.4), a result "in the small" can be generalized to any control polygon, if the control points in the sequence of subdivided polygons become sufficiently close after a finite number of subdivisions. This is straightforward from the following two facts:

1) the RDC scheme always satisfies the *contractivity property*, in particular, the greatest distance between two consecutive points of Π^{n+1}_{DC} is not greater than half the greatest distance between two consecutive points in Π^n_{DC}. This property depends only on the triangular inequality of the geodesic distance function, which holds everywhere, including at the cut locus.
2) On a compact manifold of bounded curvature there exists $\delta > 0$ s.t. every ball of radius $r \leq \delta$ is convex [44].

Therefore, in a finite number of subdivision steps, we obtain a sequence of control polygons such that each of them is contained in a convex set, hence undergoes the hypothesis of [7]. Moreover, by construction, every two consecutive segments have the same tangent at their junction point, thus their limit curves join with C^1 continuity. $\square$

Note that the constraints imposed by Noakes [7] have the purpose of warranting the uniqueness of a geodesic and its smooth dependence from its endpoints. In our case, if such constraints are violated, the limit curve is just one of the possible curves, which one obtains by the arbitrary, but deterministic, choices made by operator $\mathscr{A}$ at the cut locus. Once the choice is made, the resulting curve is guaranteed to be C^1. However, the result may not be continuous in the *space of curves* while varying the control points. The consequences of this fact will be discussed in Sec. 3.7.

Concerning curves of different order, Noakes [45] proved that the C^1 continuity holds also for quadratic curves. It remains an open question whether the RDC scheme produces curves with higher smoothness.

3.6 Open-uniform Lane-Riesenfeld Subdivision (OLR)

In [8], a uniform subdivision scheme has been proposed, which ports to the manifold setting the well known Lane-Riesenfeld (LR) scheme [46]. Such a scheme converges to B-splines and cannot be used directly to design curves with fixed endpoints. We generalize their result to a scheme with end conditions, which defines Bézier curves, and we show that, in the cubic case, it converges to segments that are C^2 everywhere, possibly except at the endpoints, where they are at least C^1.

We briefly review the Euclidean scheme [39], [47]. A cubic Bézier can be represented with an open-uniform B-spline* of order 4, having the same control polygon Π, and knot vector $(0,0,0,0,1,1,1,1)$. Repeated knot insertion at the midpoint of all non-zero intervals produces a sequence of open uniform B-splines,

∗. A B-spline is said to be open-uniform, or uniform with end conditions, if it is uniform, except at its endpoints, where repeated knots are inserted to make the curve interpolate the endpoints of its control polygon.

all describing the same curve; and the sequence of control polygons Π^n_{LR} converges to the curve itself.

The corresponding subdivision requires four special stencils at each end of the polygon, and it is defined as follows:

$$\begin{aligned}
P^{n+1}_{2j} &= \tfrac{1}{2}P^n_j + \tfrac{1}{2}P^n_{j+1} && j = 2...2^n-2\\
P^{n+1}_{2j+1} &= \tfrac{1}{8}P^n_j + \tfrac{3}{4}P^n_{j+1} + \tfrac{1}{8}P^n_{j+2} && j = 2...2^n-3\\
P^{n+1}_0 &= P_0\\
P^{n+1}_1 &= \tfrac{1}{2}P^n_0 + \tfrac{1}{2}P^n_1\\
P^{n+1}_2 &= \tfrac{3}{4}P^n_1 + \tfrac{1}{4}P^n_2\\
P^{n+1}_3 &= \tfrac{3}{16}P^n_1 + \tfrac{11}{16}P^n_2 + \tfrac{2}{16}P^n_3
\end{aligned} \tag{8}$$

where, for the sake of brevity, we have omitted the end conditions to the right end side, which are symmetric to the ones on the left. We also omit the special stencils that are needed at the first and second levels of subdivision, which can be derived easily, and treated analogously in the context of the following extension. Note that, the first two stencils in Eq. 8 give the uniform LR scheme with two smoothing steps, which is applied to all central points. Here, the stencils are written in compact form, instead of the usual sequence of one average step followed by two smoothing steps, because this leads to the same result in the linear scheme.

In order to port such a scheme to the manifold setting, we first observe that some of the stencils appearing in Eq. 8 involve more than two control points. Since, in general, we cannot rely on the RCM, we need to factorize such stencils with repeated averages, computed with operator $\mathscr{A}$. In the manifold setting, different factorizations may lead to different curves. We adopt a factorization that "in the middle" (i.e., for $j = 2\dots2^n-2$) gives the same scheme of [8]:

$$\begin{gathered}
\widetilde{P}^{n+1}_{2j} = P^n_j \quad \widetilde{P}^{n+1}_{2j+1} = \mathscr{A}(P^n_j, P^n_{j+1}, \tfrac{1}{2})\\
Q^{n+1}_{2j} = \mathscr{A}(\widetilde{P}^{n+1}_{2j}, \widetilde{P}^{n+1}_{2j+1}, \tfrac{1}{2}) \quad Q^{n+1}_{2j+1} = \mathscr{A}(\widetilde{P}^{n+1}_{2j+1}, \widetilde{P}^{n+1}_{2j+2}, \tfrac{1}{2})\\
P^{n+1}_{2j} = \mathscr{A}(Q^{n+1}_{2j}, Q^{n+1}_{2j+1}, \tfrac{1}{2}) \quad P^{n+1}_{2j+1} = \mathscr{A}(Q^{n+1}_{2j+1}, Q^{n+1}_{2j+2}, \tfrac{1}{2}).
\end{gathered}$$

This factorization indeed consists of one average step followed by two smoothing steps. Note, however, that Q^{n+1}_{2j} and Q^{n+1}_{2j+1} lie on the shortest geodesic path γ_j connecting P^n_j to P^n_{j+1}, and that averages between points lying on γ_j are in fact linear with respect to its arc length. Consequently, we have that $P^{n+1}_{2j} = \widetilde{P}^{n+1}_{2j+1}$, and we can rewrite the above formulas more compactly as follows:

$$\begin{aligned}
Q^{n+1}_{2j} &= \mathscr{A}(P^n_j, P^n_{j+1}, \tfrac{3}{4}), & Q^{n+1}_{2j+1} &= \mathscr{A}(P^n_{j+1} P^n_{j+2}, \tfrac{1}{4})\\
P^{n+1}_{2j} &= \mathscr{A}(P^n_j, P^n_{j+1}, \tfrac{1}{2}), & P^{n+1}_{2j+1} &= \mathscr{A}(Q^n_{2j}, Q^n_{2j+1}, \tfrac{1}{2}).
\end{aligned} \tag{9}$$

We factorize the end stencil for P^{n+1}_3 (and its symmetric point to the other end of the polygon) in a similar way. We require that this point is again obtained with one averaging step followed by two smoothing steps, and we apply the considerations above to express repeated averages along the same geodesic in a compact way. In order to accommodate for the other end conditions, the averaging step between P^n_1 and P^n_2 must be unbalanced, while all other steps can be maintained balanced. The only factorization fulfilling these constraints is given by the following equations:

$$\begin{gathered}
P^{n+1}_0 = P_0\\
P^{n+1}_1 = \mathscr{A}(P^n_0, P^n_1, \tfrac{1}{2})\\
P^{n+1}_2 = \mathscr{A}(P^n_1, P^n_2, \tfrac{1}{4})\\
Q^{n+1}_2 = \mathscr{A}(P^n_1, P^n_2, \tfrac{5}{8}), \quad Q^{n+1}_3 = \mathscr{A}(P^n_2, P^n_3, \tfrac{1}{4})\\
P^{n+1}_3 = \mathscr{A}(Q^{n+1}_2, Q^{n+1}_3, \tfrac{1}{2})
\end{gathered} \tag{10}$$

where, as above, the expressions of Q^{n+1}_2 and Q^{n+1}_3 incorporate the averaging step and the first smoothing step.

In summary, Equations 9 and 10 provide the stencils that generalize Eq. 8 to the manifold case. One step of subdivision for $n = 3$ is exemplified in Figure 7 (OLR).

We generalize the results of [8] to the above scheme, as follows:

Proposition 3.2. *For any given control polygon $\Pi = (P_0, P_1, P_2, P_3)$, the manifold OLR subdivision converges to a limit curve that is C^2 continuous, possibly except at its endpoints. The limit curve interpolates the endpoints of polygon Π and it is tangent to it.*

Proof. We show that everywhere, except at the endpoints, the results of [8] apply after a finite number of subdivision steps. To this aim, we recall that our scheme is the result of repeated knot insertion, which bisects all non-null intervals at each iteration. We exploit the relation between knots and points of the subdivision polygon to show that for every $t \in (0,1)$ the limit curve exists and is C^2. For any given value of t, after $\bar{n}$ iterations, we have that $t \in (2^{-\bar{n}}(j-1), 2^{-\bar{n}}(j+1))$ for some $j \in \mathbb{N}$. We can always choose $\bar{n}$ large enough that $j > 7$ and $j < 2^{-\bar{n}} - 3$. In this case, the five consecutive control points $P^{\bar{n}}_{j-4}, \dots, P^{\bar{n}}_j$, will undergo the uniform LR stencils at all subsequent levels of subdivision. Next we proceed as in the proof of Proposition 3.1. By triangular inequality, it is easy to show that the manifold OLR scheme is contractive. Thus, in a finite number of subdivision steps, say $\tilde{n}$, all 5-tuples of consecutive points in $\Pi^{\tilde{n}}_{LR}$, are contained in a totally normal neighborhood. If we take $n = \max(\tilde{n}, \bar{n})$ then, by [8], it follows that the polygon $P^n_{j-4}, P^n_{j-3}, P^n_{j-2}, P^n_{j-1}, P^n_j$ converges to a C^2 limit curve, corresponding to interval $(2^{-n}(j-1), 2^{-n}(j+1))$, which contains t. The end conditions in Eq. 10 trivially guarantee that the limit curve interpolates the initial control polygon Π at P_0, and it will be tangent at P_0 to the geodesic connecting P_0P_1.

□

It follows from the proposition above that a single cubic Bézier segment has C^2 continuity, while different segments can be joined to form splines with C^1 continuity. It remains an open problem how to build splines with C^2 continuity at junction points.

3.7 Limitations

Since the operator $\mathscr{A}$ is deterministic, the curve obtained with either the RDC or the OLR scheme is uniquely defined by its control points. However, the curve may jump to a different configuration for small displacements of control points, which make some of the paths in the construction cross a cut locus. In Fig. 8 (left, center), a tiny displacement of one control point takes one of the shortest paths in the control polygon to a drastically different route, resulting in a different curve; see the bottom part of Fig. 5 for another example. In the accompanying video we provide more dynamic examples of jumps. Note that jumps occur quite rarely, as the cut locus of each point covers a set of zero measure. This fact is intrinsic to the discontinuity of the manifold metrics and constitutes an essential limitation to the design of splines in the manifold setting, independently of the approach adopted.

This limitation can be circumvented easily, by means of splines containing more control points, instead of single Bézier segments. See Fig. 8 (right). Point insertion can be used to constrain the curve to a desired path, as is customarily done in curve design, and motivates the algorithms we present in Sections 4.1.3 and 4.2.3. Note that, in general, the spline obtained with point insertion is just an approximation of the original curve. This is also an intrinsic

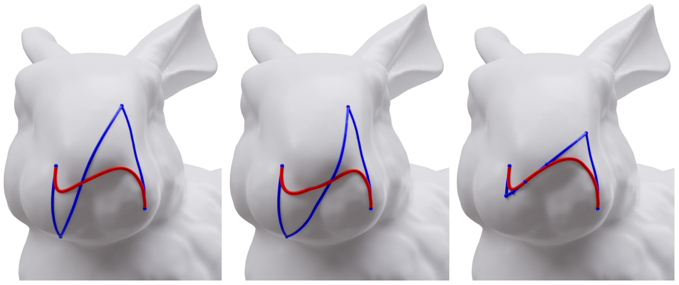

Fig. 8. Left, center: the geodesic line corresponding to the central segment of the control polygon can take two different routes at the cut locus of its endpoints, thus producing two different curves; in practice, the curve will jump between the two configurations when dragging a control point across a cut locus. Right: splitting the curve by point insertion makes the selected configuration stable upon dragging.

limitation of the manifold setting, where a sub-segment of a Bézier curve is not necessarily a Bézier curve itself. Automatic solutions would be possible. It is easy to check when a curve "jumps" while dragging a control point; in that case, the control polygon may be split, e.g., by adding a point on the curve before displacement as a new control point. In our user interface, we decided to avoid using automatic methods to warrant maximum flexibility to the user.

4 Practical Algorithms

We now focus on the RDC and OLR schemes. We provide algorithms for: approximating the curve with a geodesic polyline (curve tracing); evaluating a point on the curve for a given parameter value (point evaluation); and splitting a curve at a given point into a spline approximating it with two segments (point insertion). The algorithm for curve tracing with the RDC scheme is equivalent to the one proposed in [29], while the other five algorithms are novel.

To develop our algorithms, we assume to have procedures for (1) computing the point-to-point shortest path between pairs of points of M; (2) evaluating a point on a geodesic path at a given parameter value; and (3) casting a geodesic path from a point in a given direction. The computational details of such procedures, as well as additional algorithms to support interactive control, are provided in Section 5.

4.1 Algorithms for the RDC scheme

4.1.1 *Curve tracing*

The tracing algorithm recursively subdivides a geodesic polygon Π into two sub-polygons Π_L and Π_R. Recursion is initialized by computing the three shortest paths that constitute the polygon connecting the initial control points P_0, P_1, P_2, P_3. Referring to Fig. 7 (RDC), one step of subdivision entails computing three geodesic paths, and evaluating six midpoints of existing geodesics. The polygons Π_L and Π_R are built by collecting the sub-paths depicted in violet and in pink, respectively.

For uniform subdivision, a maximum level of recursion is either chosen by the user, or computed on the basis of the total length $L(\Pi)$ of the initial polygon Π, and a threshold δ. Since the paths in the subdivided polygon are shrinking through recursion, then after $\lceil \log_2(L(\Pi)/\delta) \rceil$ recursion levels, the length of a geodesic path in the output will be bounded by δ. For adaptive subdivision, we stop recursion as soon as the angles between tangents of consecutive segments of Π differ for less than a given threshold θ. For a small value of θ, this suggests that the curve can be approximated with a geodesic polyline connecting the points of Π. Note that angles are computed in tangent space, hence accounting just for the geodesic curvature of the curve while disregarding the normal curvature induced from the embedding. This approach works even when a curve crosses sharp creases on polyhedral objects. Like in the Euclidean case, cusps may appear at the transition between a simple and a self-intersecting configuration of a curve. We resolve cusps by stopping the recursion after a maximum number of levels.

4.1.2 *Point evaluation*

We support the evaluation of the point at a value $\bar{t}$ on the curve. This requires traversing the recursion tree with a bisection algorithm. We split the control polygon at each level as described above, but only compute the sub-polygon that contains $\bar{t}$. We stop recursion with the same criteria listed above.

The point at $\bar{t}$ is computed by direct de Casteljau evaluation on the leaf control polygon. Here we are assuming that the control polygon in the leaf node is short enough to support direct de Casteljau evaluation, and we use it to approximate the limit point on the subdivided curve. By using arguments of proximity, as in [36], [48], it can be shown that this approximation converges to the limit curve, as the subdivision polygon is subdivided further.

4.1.3 *Point insertion*

With point insertion, a user can split a curve at a given point $P_{\bar{t}}$, and obtain a spline consisting of two Bézier curves, from P_0 to $P_{\bar{t}}$ and from $P_{\bar{t}}$ to P_k, which substitutes the input curve. This is used to add detail during editing. While this computation is exact in the Euclidean setting, the identity of curves before and after point insertion cannot be guaranteed in the manifold setting. Here we provide a solution that approximates the input curve by interpolating its endpoints, as well as point $P_{\bar{t}}$, and preserving the tangents at such points. We place the unconstrained points by mimicking the algorithm in the Euclidean setting, trying to obtain a spline that closely approximates the input curve.

We refer to the construction illustrated in Fig. 9. We descend the recursion tree as in the previous algorithm, in order to find the leaf Π containing the splitting point $P_{\bar{t}}$. Assuming, as above, that Π is small enough, we split it at value $\bar{t}$ by direct de Casteljau evaluation, thus obtaining the two polygons Π_L and Π_R, in purple and magenta in the figure. We process the two halves independently. Here we show the algorithm for finding the polygon $\bar{\Pi}_L$ defining the curve between P_0 and $P_{\bar{t}}$. The construction of the other half is symmetric.

Let us denote

$$\Pi_L = (P', P_{L,1}, P_{L,2}, P_{\bar{t}})$$

the polygon in purple in Fig. 9, and let t' be the the parameter corresponding to P' on the input curve. In order to interpolate the endpoints and the related tangent directions, we must have

$$\bar{\Pi}_L = (P_0, \bar{P}_1, \bar{P}_2, P_{\bar{t}})$$

where $\bar{P}_1$ must lie along the geodesic line connecting P_0 and P_1, and $\bar{P}_2$ must lie along the extension of the geodesic line connecting $P_{L,2}$ and $P_{\bar{t}}$. The remaining degrees of freedom concern where to place $\bar{P}_1$ and $\bar{P}_2$ along such lines. We place such two points by maintaining the same proportions that would be used in the Euclidean case.

Since $P_{\bar{t}}$ is an endpoint of the curve defined by $\bar{\Pi}_L$, and it lies at parameter $\bar{t}$ on the input polygon Π, then $\bar{P}_1$ is found at distance

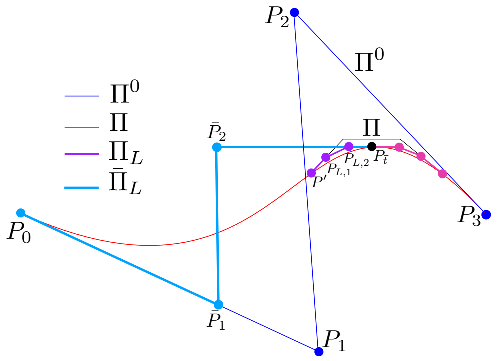

Fig. 9. The point insertion algorithm for a cubic curve (left side only). The control polygon $\bar{\Pi}_L$ (light blue) defining the left side of the curve upon split at $P_{\bar{t}}$ is built by shortening the first segment of Π^0 (dark blue) and extending the last segment of Π_L (purple).

$\bar{t} \cdot d(P_0, P_1)$ from P_0. Now we place $\bar{P}_2$ in such a way that, once $\bar{\Pi}_L$ is split at the parameter corresponding to P', the polygon Π_L is generated as its right sub-polygon. The parameter of P' with respect to the sub-curve from P_0 to $P_{\bar{t}}$ is $t'/\bar{t}$, thus we have

$$d(P_{\bar{t}}, P_{L.2}) = (1 - \frac{t'}{\bar{t}}) \cdot d(P_{\bar{t}}, \bar{P}_2).$$

Therefore, we conclude that $\bar{P}_2$ is obtained by extending the geodesic line from $P_{\bar{t}}$ to $P_{L,2}$ for a length $d(P_{L,2}, P_{\bar{t}}) \cdot \frac{t'}{(\bar{t}-t')}$.

While we do not provide any bound on the approximation of the input curve, we tested this algorithm on many curves and the results were mostly very close to the input. Except, as all other editing operations (e.g., dragging control points) a split may cause a jump of one sub-curve, discussed in Sec. 3.7. Jumps can be easily recovered either with further splits or by dragging the handle points that control tangents. Notice that, for practical applications, point insertion is aimed at adding more degrees of freedom to the spline: interpolation of pinned points and control of tangents at them are in fact all a designer requires.

4.2 Algorithms for the OLR scheme

4.2.1 Curve tracing

For uniform subdivision, our OLR scheme can be easily expanded up to a certain level $\bar{n}$, and the curve approximated with the geodesic polygon $\Pi^{\bar{n}}$. The maximum expansion level $\bar{n}$ is set as in the corresponding RDC algorithm. At each level of subdivision, we obtain the vertices of the refined polygon by applying the subdivision stencils of Equations 9 and 10. The construction of the third level of subdivision is shown in Fig. 7 (OLR).

Notice that the uniform subdivision, as described above, defines a (virtual and infinite) binary tree of intervals, that we call the *expansion tree*: the root of the expansion tree corresponds to the whole interval $[0,1]$, while a generic node $[t^i_j, t^i_{j+1}]$ at level i is split in the middle into two intervals at level $i+1$. The node $[t^i_j, t^i_{j+1}]$ encodes a segment of B-spline, defining the curve in the corresponding interval, with control points $(P^i_{j-3}, \dots, P^i_j)$. One more level of subdivision splits this interval into two sub-intervals $[t^{i+1}_{2j}, t^{i+1}_{2j+1}]$ and $[t^{i+1}_{2j+1}, t^{i+1}_{2j+2}]$ and generates 5 new control points, which depend just on $(P^i_{j-3}, \dots, P^i_j)$: the first 4 points are associated to the interval to the left, and the last 4 to the interval to the right, with an overlap of 3 control points between the two sets. The expansion tree is defined implicitly and it needs not being encoded.

We exploit the structure of the expansion tree to design an algorithm for adaptive subdivision, which is controlled by the same stopping criterion used for the RDC scheme, i.e., we stop the expansion of a node as soon as the angle between consecutive segments of the polygon is small enough. The algorithm corresponds to visiting a subtree of the expansion tree in depth-first order; a leaf of the subtree is a node of the expansion tree where we stop recursion. During the visit, at each internal node, we split the interval as described above, and generate the control points for its two children to continue the expansion; while at each leaf, we generate the nodes of the output polygon.

Depth-first traversal guarantees that leaves are visited left to right: the leftmost leaf in the expansion tree is the first one to produce an output, adding all its control points; all other leaves add just their rightmost control point to the output. The final approximation of the curve is obtained by connecting the output points pairwise with shortest geodesic paths.

Note that, it is not necessary to encode the subtree visited by the algorithm. It is just sufficient to encode the path in the expansion tree connecting the root to the current node, storing at each node its corresponding interval, and its control polygon.

4.2.2 Point evaluation

The point evaluation algorithm is analogous to the one for the RDC scheme, by descending a path in the expansion tree described above. Given interval $[t^i_j, t^i_{j+1}]$ containing $\bar{t}$ at subdivision level i, we only need to compute, with the proper stencils, the 4 points corresponding to its sub-interval containing $\bar{t}$ at the next level.

Once recursion stops, we assume that all pairs of consecutive control points in the current interval lie in a totally normal ball. Here we evaluate the curve directly with a manifold version of the de Boor algorithm [3], which works on repeated averages and can be obtained by substituting the affine averages with the manifold average $\mathscr{A}$, just like the direct de Casteljau evaluation.

The same remarks we made for the RDC scheme about approximation and convergence in the limit apply here, too.

4.2.3 Point insertion

This algorithm is analogous to the one described for the RDC scheme. We descend the recursion tree as in the point evaluation algorithm. When reaching the leaf containing the splitting point $P_{\bar{t}}$, we convert the control polygon of the uniform B-spline in that leaf into the corresponding control polygon of the Bézier curve, by applying the well known conversion formula [39](Sec.7.5), where affine averages are substituted with the manifold average $\mathscr{A}$. Then we proceed as described for the RDC scheme.

5 Implementation and User Interface

We implemented the algorithms described in the previous section for discrete surfaces represented as triangle meshes, targeting interactivity for long curves and meshes of millions of triangles. Our algorithms are implemented on top of a few geodesic operations that we describe in this section together with operations required to support the user interface. All operations are implemented in C++ and released as open source in [49].

5.1 Geodesic primitives

Data representation: We encode a triangle mesh with an indexed data structure consisting of three arrays for vertices, triangles, and triangles adjacencies, which also provide the dual graph having the triangles as nodes. We encode points lying on the mesh with their triangle index and barycentric coordinates. A geodesic path is encoded with an array of indices to the triangle strip crossed by the path, and an array of real values that encode the intercept of the path with the transversal edges in the strip.

Point-to-point shortest path: Our algorithm to compute locally shortest geodesic paths is derived by combining insights from the works of Lee and Preparata [50] and Xin and Wang [51]. The algorithm consists of three phases: (i) extraction of an initial strip; (ii) shortest path in a strip; and (iii) strip straightening.

Phase (i), which has been overlooked in several previous works, is critical as it can become the bottleneck on large meshes (see, e.g., the discussion in [33] 5.2.1). Given two mesh points P and Q, we compute a strip of triangles that connects them, performing a search on the dual graph. We experienced a relevant speedup over the classical Dijkstra search by using a shortest path algorithm based on the small-label-first (SLF) and large-label-last (LLL) heuristics [52], which do not require a priority queue, but just a double ended queue. The SLF heuristics adds a new node to either the front, or the back of the queue, according to the estimated distance of that node, compared to the distance of the first node in the queue. The LLL heuristics moves a node from the front to the back of the queue if its distance is larger than the average distance of nodes in the queue. Besides, we weight each node as in a classical A* search, with the sum of its current distance from the source plus its Euclidean 3D distance to the target. This heuristic prioritizes the exploration of triangles closer to the destination in terms of Euclidean distance, improving performance on most models.

In phase (ii), the strip is unfolded in the 2D plane and the shortest path within it is computed in linear time with the funnel algorithm [50]. See Fig. 10(a-b-c) for an example.

In phase (iii), in order to obtain the locally shortest path on the mesh, we remove reflex vertices from the strip where possible. To this aim, Xin and Wang find the reflex vertices that can be removed by computing angles about a vertex inside and outside the strip, respectively [51]. However, in our experiments, the computation of angles slows down the algorithm, because the star of each reflex vertex is retrieved from a data structure that is not in cache memory. Instead, we select the reflex vertex v that creates the largest turn in the polyline and, similarly to [51], we update the strip by substituting the current semi-star of v inside the strip with its other semi-star. We perform the unfolding and the funnel algorithm on the new strip: if v still remains on the path, then it is frozen; we repeat this procedure until all reflex vertices either are removed or become frozen. See Fig. 10(d-e-f) for an example.

In Sec. 6.3, we compare this algorithm with an algorithm at the state of the art [33], showing that we consistently beat it for about one order of magnitude in speed. The breakup of times presented in Fig. 23 suggests that the speed-up stems mostly from Phase (i).

Straightest geodesics and parallel trasport: We follow Polthier and Schmies [53] to trace a geodesic starting at a mesh point P and following one given direction $\mathbf{u}$. This entails unfolding the triangles of the strip crossed by the geodesic, and possibly mapping the star of vertices crossed by the geodesic to their tangent plane. We use an approach similar to Knöppel et al. [54] to parallel transport vectors along curves. Again, the strip of triangles traversed by the curve is unfolded, and local reference frames in the strip are used to transport the vector along it. The implementation of both techniques is straightforward.

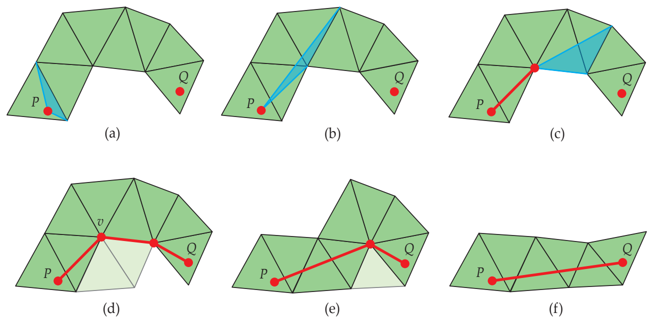

Fig. 10. Shortest path computation. Given a source P and a target Q an initial strip of triangles is found with a search on the dual graph of the mesh. (a) A shortest path within the strip is found by propagating a funnel, which is initialized with its apex at P and its front at the first edge crossing the strip. (b) The edges of the strip are processed one by one, to tighten the front of the funnel. (c) When the funnel collapses, a new vertex, called a pseudo-source, is added to the path and the apex of the funnel is moved to the pseudo-source. (d) When Q is reached, some reflex vertices may still lie on the path. (e) Reflex vertices are analyzed for possible removal, starting at the vertex v causing the sharpest turn. (f) The final path is found when no more reflex vertices can be removed.

5.2 User interface

Leveraging the proposed algorithms, we developed a graphical application, demonstrated in the supplemental video, which allow users to interactively edit cubic splines on meshes, with the same interaction methaphors used in 2D vector graphics. Our application supports moving, adding, and deleting control points, and by translating, scaling and rotating whole splines on the surface domain. Here we describe the main editing feature, referring the reader to the supplemental video for a demonstration.

Curve editing: Borrowing the editing semantic from 2D tools, control points are distinguished in *anchor points* and *handle points*. Anchors are those points where two Bézier curves are joined, while handle points are the ones preceding and following the anchor points. We connect each handle point with its corresponding anchor with a geodetic segment. A spline is tangent to those segments at the anchor points. In the 2D setting, when an anchor is dragged, the two tangent segments move with it and so do the associated handle points. To obtain the same behavior on the surface, when moving an anchor point from P to P', we find the two tangent directions of the tangent segments at P. Then, for each such segment, we trace a straightest geodesics starting at P' and for the same length of the segment, in the direction of its tangent, rotated by the parallel transport from P to P'. The endpoint of each segment is the new position of the corresponding handle point. In the 2D setting the user can impose an anchor to be "smooth", i.e. the two associated tangent segments are always colinear, which automatically ensure C^1 continuity at the anchor point. To provide the same functionality on the surface, whenever the handle point Q_1 is moved, the opposite handle point Q_2 is recomputed by tracing a straightest geodesic from the anchor P along the tangent direction defined from segment Q_1P to find the new position of handle Q_2.

Rotation, Scaling and Translation: We support translation, rotation and scaling of a whole spline. In the 2D settings these operations are obtained by just applying the same affine transform to all control points. In the surface setting, we define the transformation about the mesh point C under the mouse pointer. The normal coordinates of the control points are computed with respect to C, in

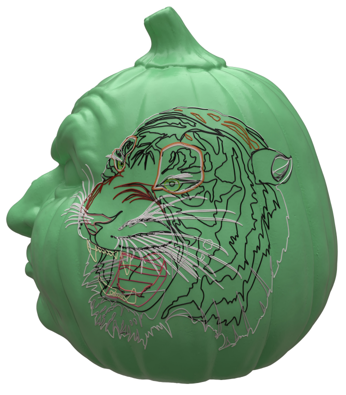

Fig. 11. Example of importing a large SVG, made of 2056 curves, onto the pumpkin model, consisting of 394k triangles. Our algorithm takes 289 milliseconds to trace all curves.

a sort of discrete exponential map. Then, the linear transformation is applied on these 2D coordinates, which are finally converted back into mesh points by tracing straightest geodesic paths outward from C. Translation needs special handling, as the center of the transformation C is dragged to a new position C'. To compensate for the change of reference frame, the normal coordinates are rotated by the opposite angle of the parallel transport given by the tangent vector from C to C'.

Importing SVG drawings: Sometimes, it is helpful to map a collection of 2D splines onto the surface. To this aim, we map the 2D control points onto surface, and then trace the splines on the manifold. See Figures 1 and 11 and the accompanying video for examples.We map control points with a method analogous to Biermann et al. [12] that is based on the conversion between polar coordinated in 2D and normal coordinates on the manifold. Each control point of the SVG drawing is converted into a mesh point by taking its polar coordinates, and tracing a geodesic from a center point in the given tangent direction, for the given distance.

Notice that this is intended just as a rough initialization. Since we map only the control points, the intersections between lines will be preserved only at the interpolated points of the splines, while they may differ elsewhere. The advantage here is that all distortions and artifacts that might arise in this phase can be adjusted by editing the result, which is provided in vector format directly on the surface. While this would be impossible with parametrization approaches that just map the discretized splines.

6 Results and Validation

We validate our work by tracing curves over a large number of meshes, by comparing it with state-of-the-art solutions, and by performing interactive editing sessions. Our algorithms always produce a valid output, in a time compatible with interactive usage in over 99% of the trials (Table 1). We overcome the limitations of state-of-the-art methods, producing valid results with any control polygon on any surface (Fig. 18); and our running times are comparable or faster than state-of-the-art methods (Table 2 and Fig. 22). Our system supports editing for meshes of the order of one million triangles on a laptop computer. Interaction is still supported on meshes with several millions of triangles, provided that single curves do not span too large a fraction of the model (see Figures 1 and 24, Table 3, and the accompanying video). Very long segments are rare in actual editing sessions, as real designs are usually made of many splines, each consisting of several small segments.

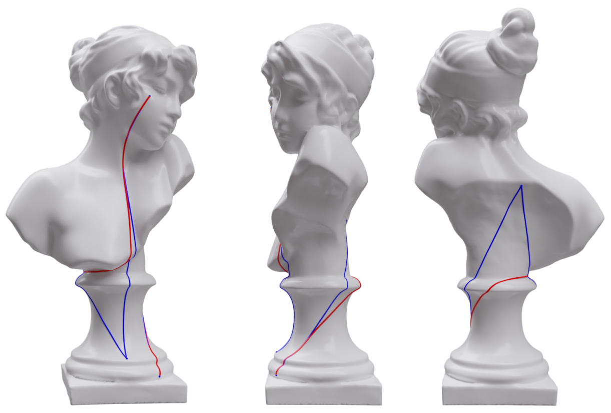

Fig. 12. Three views of a random curve generated during trials on a model from the Thingi10k repository: in all experiments the control points of each curve were randomly selected over the surface of the object.

6.1 Robustness and performance

We tested our algorithms for robustness by running a large experiment on the Thingi10k repository [11]. We extracted the subset of meshes that are manifold and watertight, for a total of 5567 models. The models are used as is, without any pre-processing. For each model, we consider 100 random cubic curves. For each curve, we take the model in its standard pose, and pick points on it by casting random rays orthogonal to the view plane, until we find four points that lie on the surface. These become the control points of the curve. We place no restriction on the arrangement of the control points. This gives us a total of more than half million control polygons. Fig. 12 shows a random curve generated on one of the objects during trials.

For each test, we run both the RDC and the OLR tracing algorithms, in their uniform and adaptive configurations. The uniform RDC algorithm is expanded to 4 levels of recursion, which generates a geodesic polyline consisting of 48 geodesic segments. The uniform OLR algorithm is expanded to 6 levels of recursion, which generates a geodesic polyline consisting of 66 geodesic segments. In fact, because of the different subdivision rules, we cannot generate the same number of segments for both schemes. For the adaptive variants, we set a threshold $\theta = 5^{\circ}$ for the maximum angle between consecutive geodesic segments along the polyline. In this case, the number of geodesic segments in output is variable, depending on the curve and on the method. Since all algorithms generate very similar curves, the final tessellated paths that approximate the curve on the mesh have about the same number of segments in all four cases.

Since we have no ground truth to compare with, we indirectly assess the correctness of the results with the following tests:

1) Termination: the algorithm must complete;
2) Continuity: all pairs of consecutive points along the output polyline must lie either inside or on the edges of the same triangle (notice that points on edges are forced by the algorithm to go across adjacent triangles).
3) Smoothness: the angle between consecutive segments of the polyline, measured in tangent space, must be lower than a given threshold (5 degrees).

All our algorithms passed all the tests in all experiments.

Trials were executed on a Linux PC with an AMD Ryzen 5 2600x and 32GB memory, running on a single core. In Table

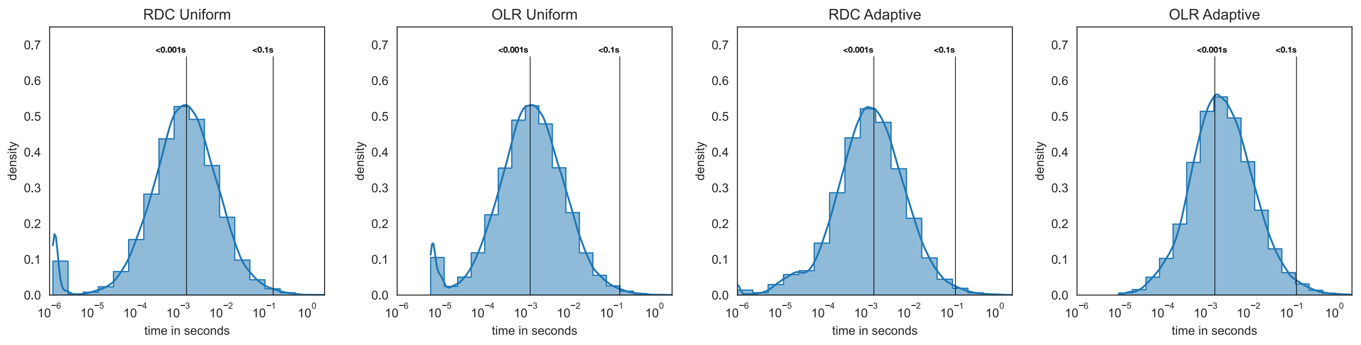

Fig. 13. The distributions of running times of our four algorithms for curve tracing in 556,700 trials on 5,567 models from the Thingi10k repository, tracing 100 random curves on each model. All algorithms provide a valid output in all trials. The different algorithms have similar behavior and are compliant with interaction (< 0.1 seconds/curve) in about 99% of the trials. For uniform subdivision, the OLR algorithm is slightly faster than the RDC algorithm; while for adaptive subdivision, the RDC algorithm performs slightly better than the OLR algorithm. Adaptive algorithms have slightly narrower distributions than uniform algorithms.

1 and Fig. 13, we compare the timing performance of the four algorithms. All algorithms perform quite similarly, and remain interactive in all cases, with roughly 40% of trials running at less than 1 millisecond per curve, and 99% of the trials running faster than 0.1 second/curve. The few trials in which they take more time are concerned, with very few exceptions, either with very long curves on large meshes (>1M triangles), or with meshes containing many topological holes, in which finding shortest paths between points is more expensive.

There are small differences in the performances of the different algorithms. For uniform subdivision, the OLR algorithm generates results as fast as the RDC algorithm, beside generating a more refined geodesic polyline. For adaptive subdivision, the RDC algorithm runs slightly faster than the OLR algorithm. These differences are probably due to the simpler structure of the OLR uniform algorithm in one case, and to the more involved structure of the OLR adaptive algorithm in the other. In fact, both variants of the RDC algorithm follow the same recursive pattern. On the contrary, the OLR uniform algorithm expands the curve level by level, following a simpler pattern; while the OLR adaptive algorithm requires a recursive pattern, with a slightly more involuted structure than the RDC algorithms.

In the previous experiments, the cost of computing a curve depends on both the length of the curve and the size of the mesh, with trends that are not linear. Roughly speaking, the cost of finding the initial path depends on both the length of the curve and the size of the mesh, while the subsequent cost of finding the shortest path depends just on the length of the curve. As the relative length of the curve grows, the cost of finding the initial path prevails, since it may requires exploring most of the mesh. Statistics on the relative costs of the two phases are shown in Fig. 23.

For the sake of brevity, we do not present here results on the algorithms for curve tracing and point insertion, which run much faster than the tracing algorithms.

6.2 Sensitivity to the input mesh

All the algorithms presented in Sec. 5.1 are driven by the connectivity of the underlying mesh. In particular, all intersections between the traced lines and the mesh are computed locally to each triangle and forced to lie on its edges, so that each traced line consistently crosses a strip of triangles. With this approach, we could process even meshes containing nearly degenerate triangles, with angles near to zero and edge lengths near to the machine precision, by relying just on floating point operations, without incurring in numerical issues. While this is usually not the case with models used in a production environment, such meshes are common in the Thingi10k repository and provide stress tests for the robustness of our algorithms.

algorithm	percent of trials		times at percentile	
	< 0.001s	< 0.1s	90%	99%
RDC Uniform	43.1%	99.0%	< 0.0122	< 0.097
OLR Uniform	44.7%	98.9%	< 0.0123	< 0.105
RDC Adaptive	43.9%	99.0%	< 0.0120	< 0.095
OLR Adaptive	30.0%	98.1%	< 0.0215	< 0.185

TABLE 1
Time performances of our algorithms in 556,700 trials. We report the percentage of trials in which tracing a curve takes less than 0.001 and 0.1 seconds, and the running times at the 90th and 99th percentiles.

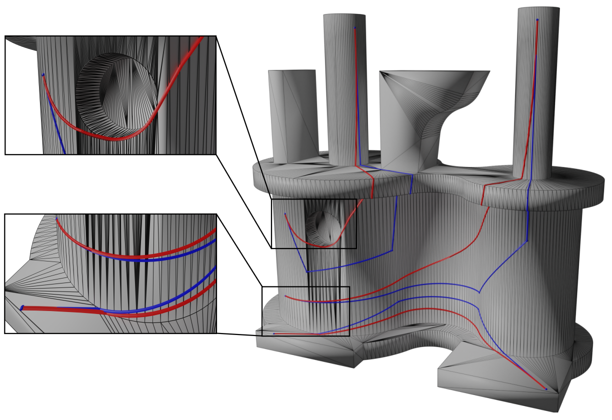

Fig. 14. The shortest path algorithm is driven by mesh connectivity and uses a refined dual graph, providing correct results even on highly anisotropic meshes containing long edges and narrow angles.

On the other hand, our algorithm for point-to-point shortest path assumes the initial guess obtained during Phase (i) to be homotopic to the result. This assumption is common to all algorithms for computing locally shortest paths, and it is reasonable as long as the mesh is sufficiently dense and uniform with respect to the underlying surface [33]. If, conversely, the mesh is too coarse and anisotropic, then Phase (i) may provide an initial guess, which

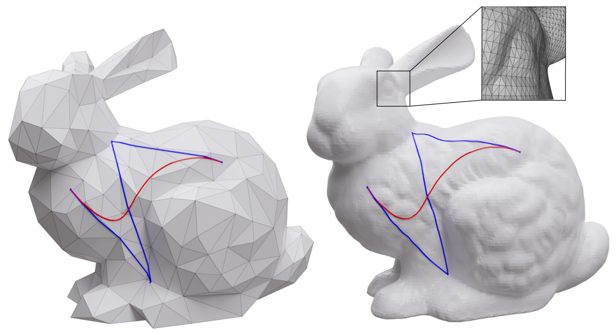

Fig. 15. Curves traced by positioning the control points in a similar way on two meshes representing the same object, one consisting of ~ 700 triangles (left) and the other consisting of ~ 70k triangles (right). Our method produces similar curves in both cases.

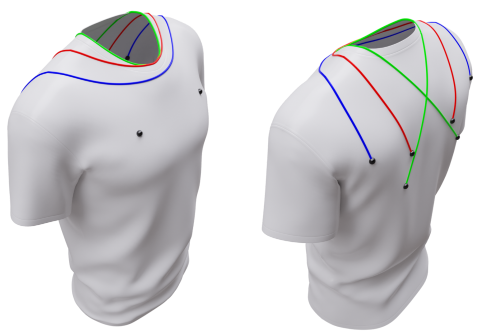

Fig. 16. Curves traced on a mesh with boundary. The three curves have different anchor points and the same handle points on the front of the shirt (black bullets). The green and the red curves are constrained to partially follow the boundary at the neck.

cannot be homotopically shortened to the correct solution. In this case, a naive application of the algorithm may get stuck in local minima of the space of shortest paths, leading to a wrong curve.

This limitation is quite rare in practice for meshes used in design applications, which is our target, but did happen for some meshes in the Thingi10k dataset. We overcome this limitation simply by creating a more accurate graph for computing the initial guess when dealing with meshes with long edges.

When we build the dual graph to be used in Phase (i), we split mesh edges that are too long at their midpoint, until all edges are shorter than a given threshold, and we symbolically subdivide their incident triangles accordingly. We chose the 5% of the diagonal of the bounding box of the model as threshold. Note that this subdivision is done just for the purpose of building the graph, without changing the underlying mesh. In this augmented graph, a single triangle may be represented by multiples nodes, giving us a more accurate approximation of paths. This approach has the effect of densifying the graph without changing the mesh upon which we run Phase (ii). Once the strip is computed on the augmented graph, we reconstruct the strip on the mesh using the graph's node provenance, i.e. the mesh triangle corresponding to each node, which we store during initialization. Fig.14 shows examples on highly anisotropic meshes from the Thingi10k repository.

An alternative approach to cope with the same problem would be to pre-compute an intrinsic Delaunay triangulation in the sense of [55] and do all computations by using intrinsic triangulations.

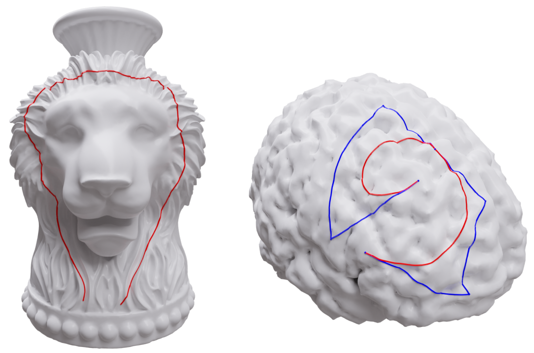

Fig. 17. Splines traced on meshes with many creases or bumps. Our algorithms work in the intrinsic metric of the surface and are oblivious of normal curvature caused from the embedding.

model		WA		ours (OLR)
name	triangles	pre-proc. (s)	tracing (ms)	tracing (ms)
cylinder	10k	54	2–2	1–1
kitten	37k	234	3–3	3–3
bunny	140k	665	2–2	10–12
lion	400k	2316	3–3	4–24
nefertiti	496k	2571	6–64	25–67

TABLE 2
Compared time performances of curve tracing with the WA method and our OLR, on the curves shown in Fig. 18 and Fig. 19. Each curve is sampled at 67 points; curve tracing times are averaged on each curve repeating tracing 1000 times per curve; we report minimum and maximum times over the different curves shown in the images.

Coarse meshes, boundaries, bumps and creases: Our algorithms are insensitive to the resolution of the mesh and work equally well on coarse as well as refined meshes. Fig. 15 shows similar curves obtained on two meshes representing the same shape at two very different resolutions. We can draw curves on meshes with boundaries, too, as shown in Fig. 16. In this case, some shortest paths, hence the curves they generate, may be constrained to follow convex portions of the boundary. Since our algorithms work in the intrinsic metric of the surface, they are insensitive to creases and bumps, as shown in Fig. 17.

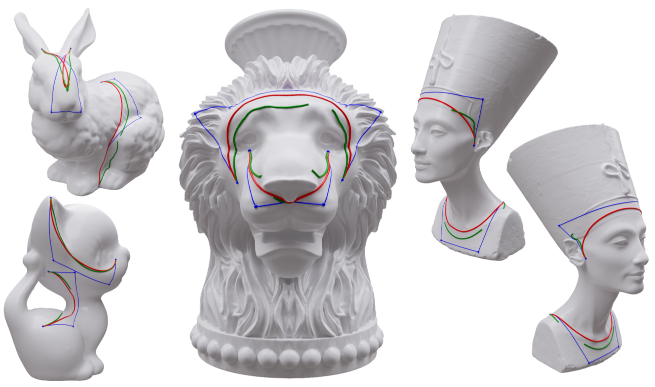

Fig. 18. Comparisons between the WA method [17] (green curves) on ours (red curves); control polygons in blue. The WA curves may contain heavy artifacts (bunny), lose tangency at the endpoints (bunny, nefertiti), or be broken (kitten, lion, nefertiti).

model		control	subdivided	time (ms)	
name	triangles	polygons	segments	total	per curve
veil	132k	2	402	2.3	1.1
arm	145k	2	856	35.6	17.8
boot	175k	2	755	21.1	10.5
deer	227k	4	1511	21.8	5.4
lady	281k	9	1917	11.4	1.2
car	282k	2	670	28.0	14.0
pumpkin	394k	5	1750	30.0	6.0
rhino	502k	7	2395	39.8	5.6
owls	641k	14	3224	20.8	1.4
alexander	699k	5	1560	20.5	4.1
vase	754k	8	1677	9.0	1.1
nike	5672k	7	4147	253.8	36.2
nefertiti	496k	463	64110	73.4	0.2
dragon	7218k	221	60656	761.7	3.4

TABLE 3
Time performances for curve tracing on the models in Fig. 24 and in the teaser, using the uniform OLR algorithm with 5 levels of subdivisions. We report the total time of computing all the curves and the average time of computing a single curve. For all the reported models, our algorithm achieves performance compatible with real-time editing, since the time per curve is at most in the order of tens of milliseconds.

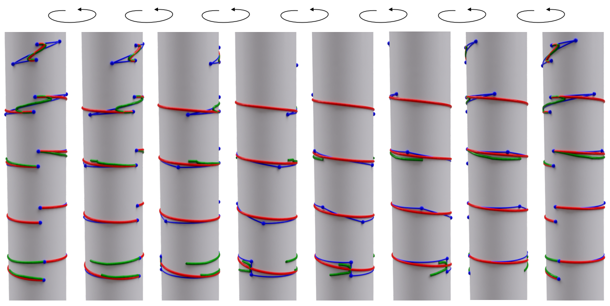
Fig. 19. Evolution of a curve while dragging handle points about a cylinder (top to bottom, rotated views left to right) with the WA method [17] (green curves) and ours (red curves). Our curve jumps from the “reversed S” configuration to the spire and remains stable throughout. The WA curve is stable only in the “reversed S” configuration, next it breaks, then it forms a spire, and eventually it breaks again.

6.3 Comparison with the state-of-the-art

Weighted Averages (WA) [17]: Panozzo et al. presented a method to estimate the RCM on a surface, by approximating the geodesic distance on the input mesh M with the Euclidean distance on a higher-dimensional embedding of M [17]. Given a set of control points and weights, instead of resolving our Eq. 3 on M, they compute the standard affine average of Eq. 1 in the embedding space. Then they use a special technique, called *Phong projection*, to bring the resulting space curve to the embedded mesh. Finally they recover the corresponding points on M. We compare with this technique by using the implementation provided by the authors, with the same sampling used in our experiments.

The embedding and the data structures to support Phong projection are computed in a pre-processing step, which is quite heavy in terms of both time and space, and can hardly scale to large datasets (see Table 2). We managed to pre-process datasets up to about 500K triangles, but we could not process some of the larger datasets we use in our work, because memory limits were exceeded. The embedding is built by sampling a small subset of the vertices first (fixed to 1000 by the authors), computing all-vs-all geodesic distances on M for such subset, and embedding such vertices in a 8D Euclidean space by keeping their mutual Euclidean distances as close as possible to their geodesic distances on M. The remaining vertices are embedded next, by using the positions of the first embedded vertices as constraints. The connectivity of M is preserved, and the positions of vertices are optimized, so that the distances between adjacent vertices remain as close as possible to their distances on M.

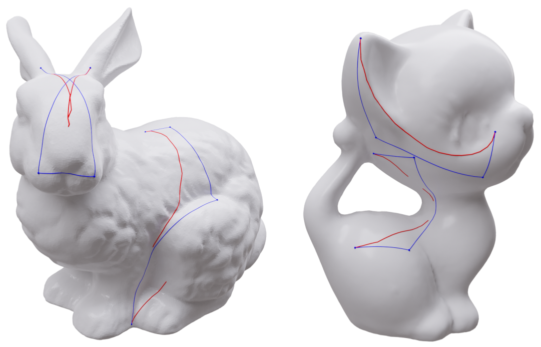
Fig. 20. The same curves of Fig. 18 on the kitten and bunny models have been traced with the RMC based on vector heat [35]. Some results are either discontinuous or highly perturbed because of non-convex configurations with multiple local minima.

The online phase of WA is very fast, and it is insensitive to the size of the input and the length of the curve (see Table 2). However, we experienced a case that took one order of magnitude more time than the others. We conjecture this is due to some unlucky configuration for the Phong projection, slowing its convergence. On the contrary, the performance of our method is dependent on both the size of the dataset and the length of the curve, being faster than WA on small datasets and shorter curves, and slower on large datasets and long curves. In terms of speed, both methods are equally compatible with interaction on the tested models.

Concerning the quality of the result, the smoothness of the WA embedding, which is necessary to guarantee the smoothness of the Phong projection, cannot be guaranteed, hence the WA method suffers of limitations similar to the RCM method analyzed in Sec. 3.4. As soon as the segments of the control polygon become long, relevant artifacts arise, and the curve may even break into several disconnected segments. Some results obtained with the WA method, compared with our results, are shown in Figures 18 and 19. In particular, Fig. 19 exemplifies the behaviors of the two methods as a control polygon becomes larger. While our curve remains smooth and stable throughout, except for the necessary jump between the “reversed S” and the spire, the WA curve becomes unstable and breaks in most configurations where the control points are far apart.

RMC based on vector heat [35]: Sharp et al. presented the vector heat method, which supports the efficient computation of the log map at an arbitrary point on the surface [35]. Algorithm 3 in the same paper uses such a log map to estimate the RCM by gradient descent, starting at an initial guess and iteratively converging to the point that minimizes the same energy of our Eq. 3. We have implemented the algorithm for tracing a Bézier curve defined with the RCM, by using the authors’ implementation of the vector heat, and plugging their Algorithm 3 into a loop, where parameter t of Eq. 7 varies between 0 and 1, and the point returned at each iteration is taken as initial guess to the next. Figures 20 and 21 show examples of failure, where curves are either discontinuous

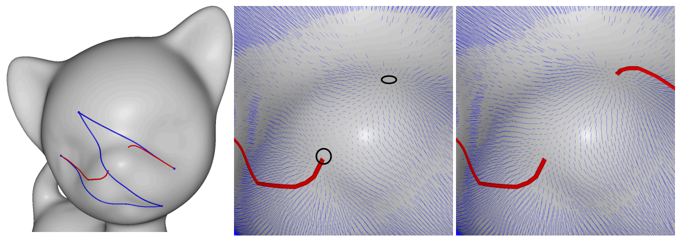

Fig. 21. Left: A curve traced with the RCM based on vector heat [35] has a big jump. Center (zoom-in): the gradient of the energy (blue needles) corresponding to the last point of the left branch, where two minima are present (black circles). Right (zoom-in): gradient field corresponding to the first point of the right branch, just one minimum has remained, which is found by gradient descent, causing the jump. Curve tracing occurs from left to right.

or highly perturbed, because the energy has more than one local minimum for certain values of t. The zoom-ins of Fig. 21 show the gradient fields before and after the jump. We remark again that such failures stem from an intrinsic limitation of the RCM and are independent of the method for computing it. Concerning efficiency, this algorithm requires computing a log map at each iteration during gradient descent, thus becoming rather slow when applied to curve tracing. We do not report the detailed time performance of this method, which takes minutes to evaluate a few tenths points on a curve in the reported examples.

RDC based on flipOut [32], [33]: Sharp and Crane proposed recently the *flipOut* algorithm as a fast solution to the computation of locally shortest geodesic paths [33]. On the basis of the *flipOut* algorithm, the same authors have implemented the algorithm of [29], which uses the same recursive scheme of our RDC algorithm for curve tracing [32].

While our algorithms have no limitations, and could provide a valid output in all 556,700 trials, the algorithm in [32] requires that the control polygon does not contain self-intersections, a case which is pretty common with cubic curves, and happens in 33% of the randomly generated polygons. This is due to an intrinsic limitation of the *flipOut* algorithm, which was discussed in [33].

We have used the authors' implementation [32] to run the same experiments of Sec. 6.1, with the same parameter used for our RDC algorithm with uniform expansion. Because of the above limitation, we excluded from the comparison all the trials for which their algorithm could not provide an output, keeping a total of 78,854 out of 556,700 trials. From a visual inspection of random samples of the results, it seems that both their algorithm and ours generate the same curves. In Fig. 22, we present a comparison between the performances of the two algorithms. Our RDC uniform algorithm exhibits a speedup of more than 10x on average.

Shortest paths: comparison with flipOut [33]: The speedup in the previous experiment is totally due to our shortest path algorithm described in Sec. 5.1. Note that the flipOut algorithm is one of the fastest available at the state of the art for computing locally shortest paths [33]. We compared the two algorithms by substituting the authors' implementation of flipOut [32] in our curve tracing algorithm in all the trials above, and measuring the times necessary just for the shortest path computations in the two cases. The comparison is shown in Fig. 23. Indeed, on average, our algorithm is one order of magnitude faster than flipOut. More precisely, while the times for path shortening (Phases (ii) and (iii) of our algorithm) are comparable with those of flipOut, our speedup is mostly due to the computation of the initial guess (Phase (i)),

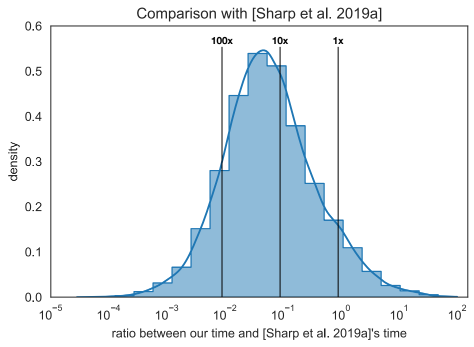

Fig. 22. The graph shows the distribution of the ratio of the running times between our RDC uniform algorithm and the implementation from [32], which is based on the *flipOut* method for computing geodesics [33]. Here we report only the 78,854 trials, out of 556,700, for which [32] could provide an output. On average, our RDC algorithm implementation provides more than a 10x speedup over [32].

which is a well known bottleneck for all this class of algorithms.

6.4 Interactive use

We have used extensively our system on a variety of models. All editing sessions where performed on a MacBook laptop with a 2.9GHz Quad-Core Intel Core i7 with 16GB memory, running on a single core.

Fig. 24 presents a gallery of curves drawn interactively on objects picked from the Thingi10k collection. Statistics for each example are summarized in Table 3. Interaction is quite intuitive, being supported with a GUI that mimics the drawing of spline curves in standard 2D systems, as described in Sec. 5.2. The most tricky aspects, with respect to the standard 2D case, are concerned with using tangents that consist of geodesic lines instead of straight lines. In our experience, the use of geodesic tangents, which is intrinsic to the manifold metric, becomes intuitive quickly.

We have stressed our system by working on very large meshes as shown in Fig. 1. Even on meshes of a few million triangles, our implementation remains interactive, as shown in Table 3.

7 Concluding Remarks

We propose methods for interactively drawing and editing of Bézier curves on meshes of millions of triangles, without any limitation on the curve shape and extension of control polygons. Our algorithms are robust, having been tested on over five thousands shapes with over half a million randomly generated control polygons, and they are compatible with interactive usage even on large meshes. Both subdivision schemes we have presented are simple to implement and produce C^1 splines. The Open-uniform Lane-Riesenfeld scheme provides the smoothest practical solution so far for Bézier segments in the manifold setting. It remains an open question how such segments could be chained to obtain C^2 Bézier splines.

The main limitation of these methods lie in the discontinuities of the space of curves with respect to their control points: curves are always smooth, but they may jump between different configurations during editing. Such a discontinuity is inherent of the geodesic metric, and it can be overcome by using a spline with shorter control polygons, instead of a single large polygon, to define the curve. Our algorithms for point insertion greatly help in this task.

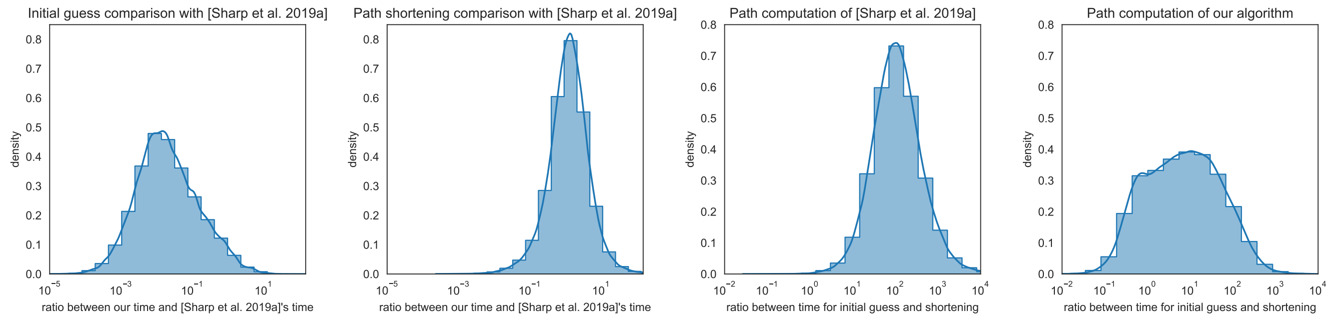

Fig. 23. Left (2 charts): On average, our algorithm for shortest paths beats by one order of magnitude the flipOut algorithm: the speedup is related to the computation of the initial guess, while the two algorithms have comparable speed for the path shortening stages. Right (2 charts): The initial guess is a bottleneck for both algorithms, but the ratio between the different stages is much more favorable for our algorithm.

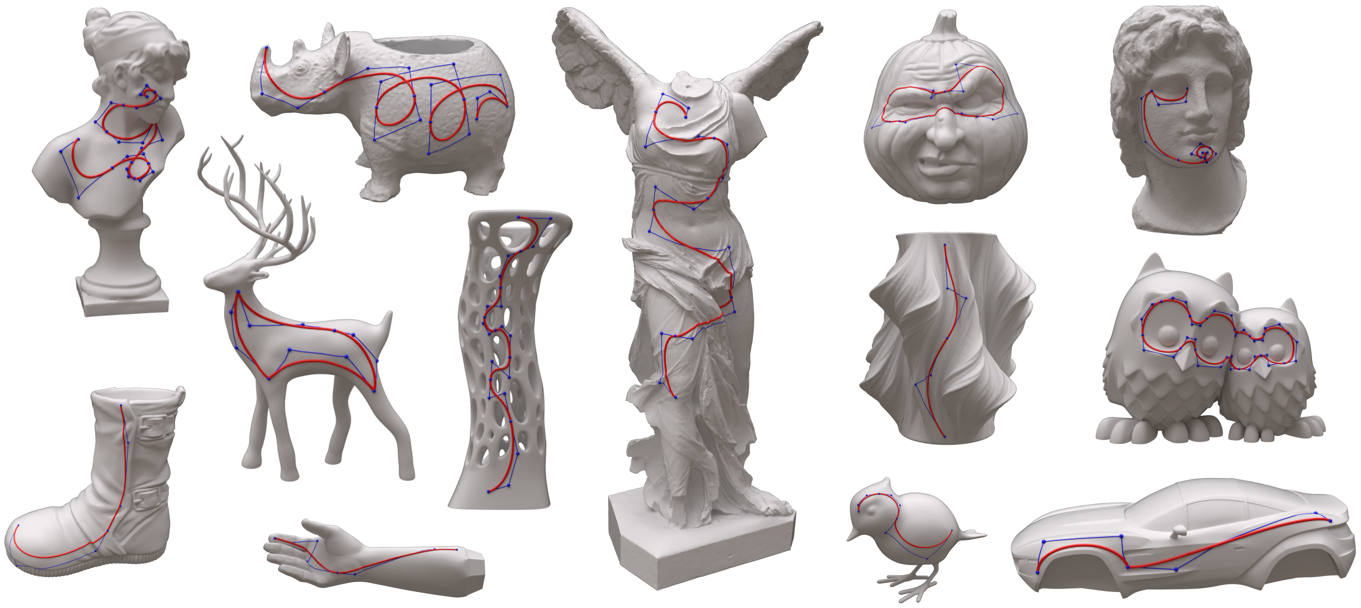

Fig. 24. A gallery of models and splines drawn with our method. Both smooth (C^1) and corner (C^0) continuity at junction points are exemplified. The selected models span a wide range of shapes and the sizes of meshes vary between about 130k and 5.7M triangles.

In the future, we want to consider other types of splines. An extension of our approach to B-splines is straightforward. An extension to interpolating splines seems easy, but it requires manifold extrapolation, which may become unstable. The most complex extension would be to handle NURBS, which at this point remains unclear how to do. More generally, the smoothness analysis in the non-uniform case needs a thorough investigation.

References

[1] "Adobe illustrator," Adobe inc., 2019. [Online]. Available: https://adobe.com/products/illustrator

[2] "Scalable vector graphics," W3C, 2010. [Online]. Available: https://www.w3.org/Graphics/SVG/

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

[4] G. Nazzaro, E. Puppo, and F. Pellacini, "geoTangle: Interactive design of geodesic tangle patterns on surfaces," *ACM Trans. Graph.*, 2021.

[5] M. Poerner, J. Suessmuth, D. Ohadi, and V. Amann, "adidas TAPE: 3-d footwear concept design ," in *ACM SIGGRAPH 2018 Talks*. New York: ACM Press, 2018, pp. 1–2.

[6] C. Mancinelli and E. Puppo, "Vector graphics on surfaces using straight-edge and compass constructions," *Computers & Graphics*, 2022.

[7] L. Noakes, "Nonlinear corner-cutting," *Adv. in Comp. Math.*, vol. 8, no. 3, pp. 165–177, 1998.

[8] T. Duchamp, G. Xie, and T. Yu, "Smoothing nonlinear subdivision schemes by averaging," *Numerical Algorithms*, vol. 77, no. 2, pp. 361–379, 2018.

[9] J. Wallner and H. Pottmann, "Intrinsic subdivision with smooth limits for graphics and animation," *ACM Trans. Graph.*, vol. 25, no. 2, pp. 356–374, 2006.

[10] F. De Goes, M. Desbrun, M. Meyer, and T. DeRose, "Subdivision exterior calculus for geometry processing." *ACM Trans. Graph.*, vol. 35, no. 4, 2016.

[11] Q. Zhou and A. Jacobson, "Thingi10k: A dataset of 10,000 3d-printing models," 2016.

[12] H. Biermann, I. Martin, F. Bernardini, and D. Zorin, "Cut-and-paste editing of multiresolution surfaces," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 312–321, 2002.

[13] P. Herholz and M. Alexa, "Efficient Computation of Smoothed Exponential Maps," *Comp. Graph. Forum*, vol. 38, pp. 79–90, 2019.

[14] R. Schmidt, "Stroke Parameterization," *Comp. Graph. Forum*, vol. 32, pp. 255–263, 2013.

[15] Q. Sun, L. Zhang, M. Zhang, X. Ying, S.-Q. Xin, J. Xia, and Y. He, "Texture brush: An interactive surface texturing interface," in *Proc. ACM SIGGRAPH Symp. Interactive 3D Graphics and Games*. New York, NY, USA: ACM, 2013, p. 153–160.

[16] R. Schmidt, C. Grimm, and B. Wyvill, "Interactive decal compositing

with discrete exponential maps," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 605–613, 2006.

[17] D. Panozzo, I. Baran, O. Diamanti, and O. Sorkine-Hornung, "Weighted averages on surfaces," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 60:1–12, 2013.

[18] L. Noakes, G. Heinzinger, and B. Paden, "Cubic splines on curved spaces," *IMA Jou. of Math. Control and Information*, vol. 6, pp. 465–473, 1989.

[19] M. Camarinha, F. Silva Leite, and P. Crouch, " Splines of class C^k on non-euclidean spaces ," *IMA Jou. of Math. Control and Information*, vol. 12, no. 4, pp. 399–410, 1995.

[20] A. Arnould, P.-Y. Gousenbourger, C. Samir, P.-A. Absil, and M. Canis, "Fitting Smooth Paths on Riemannian Manifolds: Endometrial Surface Reconstruction and Preoperative MRI-Based Navigation," in *Geometric Science of Information*. Cham: Springer, 2015, pp. 491–498.

[21] P.-Y. Gousenbourger, C. Samir, and P. Absil, "Piecewise-Bezier C1 Interpolation on Riemannian Manifolds with Application to 2D Shape Morphing," in *Proc. 22nd Int. Conf. on Pattern Recognition*, 2014, pp. 4086–4091.

[22] P.-Y. Gousenbourger, E. Massart, and P. Absil, "Data Fitting on Manifolds with Composite Bézier-Like Curves and Blended Cubic Splines," *Jou. of Math. Imaging and Vision*, vol. 61, pp. 1–27, 2018.

[23] M. Hofer and H. Pottmann, "Energy-minimizing splines in manifolds," *ACM Trans, Graph.*, vol. 23, pp. 284–293, 2004.

[24] Y. Jin, D. Song, T. Wang, J. Huang, Y. Song, and L. He, "A shell space constrained approach for curve design on surface meshes," *Computer-Aided Design*, vol. 113, pp. 24–34, 2019.

[25] H. Pottmann and M. Hofer, "A variational approach to spline curves on surfaces," *Computer aided geometric design*, vol. 22, no. 7, pp. 693–709, 2005.

[26] C. Samir, P. Absil, A. Srivastava, and E. Klassen, "A Gradient-Descent Method for Curve Fitting on Riemannian Manifolds," *Foundations of Comp. Math.*, vol. 12, no. 1, pp. 49–73, 2011.

[27] F. Park and B. Ravani, "Be ́ zier curves on Riemannian manifolds and Lie groups with kinematics applications," *Jou. of Mechanical Design*, vol. 117, no. 1, pp. 36–40, 1995.

[28] A. Lin and M. Walker, "CAGD techniques for differentiable manifolds," in *Algorithms for Approximation IV - Proc. Int. Symp. on Algorithms for Approximation*, J. Levesley, I. Anderson, and J. Mason, Eds. Huddlersfield University, UK, 2001, pp. 36–43.

[29] D. Morera, P. Carvalho, and L. Velho, "Modeling on triangulations with geodesic curves," *The Visual Computer*, vol. 24, pp. 1025–1037, 2008.

[30] E. Nava-Yazdani and K. Polthier, "De Casteljau's algorithm on manifolds," *Computer Aided Geometric Design*, vol. 30, no. 7, pp. 722–732, 2013.

[31] T. Popiel and L. Noakes, "Bézier curves and C2 interpolation in Riemannian manifolds," *Jou. of Approximation Theory*, vol. 148, no. 2, pp. 111–127, 2007.

[32] N. Sharp, K. Crane *et al.*, "geometry-central," 2019, www.geometry-central.net.

[33] N. Sharp and K. Crane, "You can find geodesic paths in triangle meshes by just flipping edges," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 249:1–15, 2020.

[34] P. Absil, P.-Y. Gousenbourger, P. Striewski, and B. Wirth, "Differentiable Piecewise-Bézier Surfaces on Riemannian Manifolds," *SIAM Jou. on Imaging Sciences*, vol. 9, no. 4, pp. 1788–1828, 2016.

[35] N. Sharp, Y. Soliman, and K. Crane, "The vector heat method," *ACM Trans. Graph.*, vol. 38, no. 3, pp. 1–19, 2019.

[36] J. Wallner, "Geometric subdivision and multiscale transforms," in *Handbook of Variational Methods for Nonlinear Geometric Data*, P. Grohs, M. Holler, and A. Weinmann, Eds. Springer, 2020, pp. 121–152. [Online]. Available: http://www.geometrie.tugraz.at/wallner/subdiv-survey.pdf

[37] N. Dyn and N. Sharon, "Manifold-valued subdivision schemes based on geodesic inductive averaging," *Journal of Computational and Applied Mathematics*, vol. 311, pp. 54–67, 2017.

[38] N. Dyn, R. Goldman, and D. Levin, "High order smoothness of non-linear Lane-Riesenfeld algorithms in the functional setting," *Computer Aided Geometric Design*, vol. 71, pp. 119–129, 2019.

[39] D. Salomon, *Curves and surfaces for computer graphics*. New York: Springer, 2006.

[40] M. do Carmo, *Riemannian Geometry*, ser. Mathematics. Boston, Massachusetts: Birkhäuser, 1992.

[41] K. Grove and H. Karcher, "How to conjugate c1-close group actions," *Mathematische Zeitschrift*, vol. 132, pp. 11–20, 1973.

[42] H. Karcher, "Riemannian center of mass and mollifier smoothing," *Communications on Pure and Applied Mathematics*, vol. 30, no. 5, pp. 509–541, 1977.

[43] B. Afsari, "Means and averaging on riemannian manifolds," Ph.D. dissertation, University of Maryland, 2009.

[44] T. Sakai, *Riemannian Geometry*. Providence: American Mathematical Society, 1997.

[45] L. Noakes, "Accelerations of Riemannian quadratics," *Proc. of the American Math. Soc.*, vol. 127, no. 6, pp. 1827–1836, 1999.

[46] J. Lane and R. Riesenfeld, "A theoretical development for the computer generation and display of piecewise polynomial surfaces," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 2, no. 1, pp. 35–46, 1980.

[47] T. Cashman, N. Dodgson, and M. Sabin, "Non-uniform B-spline subdivision using refine and smooth," in *Mathematics of Surfaces XII - LNCS*, R. Martin, M. Sabin, and J. Winkler, Eds. Berlin Heidelberg: Springer, 2007, vol. 4647.

[48] J. Wallner, "Smoothness Analysis of Subdivision Schemes by Proximity," *Constructive Approximation*, vol. 24, no. 3, pp. 289–318, Nov. 2006.

[49] F. Pellacini, G. Nazzaro, and E. Carra, "Yocto/GL: A Data-Oriented Library For Physically-Based Graphics," in *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, M. Agus, M. Corsini, and R. Pintus, Eds. The Eurographics Association, 2019.

[50] D. Lee and F. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers," *Networks*, vol. 14, no. 3, pp. 393–410, 1984.

[51] S.-Q. Xin and G.-J. Wang, "Efficiently determining a locally exact shortest path on polyhedral surfaces," *Computer Aided Design*, vol. 39, no. 12, pp. 1081–1090, 2007.

[52] D. Bertsekas, *Network optimization: continuous and discrete models*. Belmont, Massachusetts: Athena Scientific, 1998.

[53] K. Polthier and M. Schmies, "Straightest geodesics on polyhedral surfaces," in *Mathematical Visualization*. New York: Springer-Verlag, 1998, pp. 135–150.

[54] F. Knöppel, K. Crane, U. Pinkall, and P. Schröder, "Globally optimal direction fields," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 1–14, 2013.

[55] N. Sharp, Y. Soliman, and K. Crane, "Navigating intrinsic triangulations," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 55:1–16, 2019.

Claudio Mancinelli Claudio Mancinelli is a third year PhD student in Computer Science at the University of Genoa. He got both his BSc and his MSc in Applied Mathematics at the University of Genoa, respectively in 2015 and 2017. He started his PhD in 2018 under the supervision of Professor E. Puppo. His research interests are focused in Computer Graphics, Geometry Processing, and Discrete Differential Geometry. One of the purposes of his PhD thesis is to investigate how to interactively produce and edit vector graphics on highly tessellated meshes.

Giacomo Nazzaro Giacomo Nazzaro is a third year PhD student in Computer Science at the Sapienza University of Rome. He got his BSc in Applied Mathematics and his MSc in Computer Science, respectively in 2016 and 2018. In 2018 he started his PhD under the supervision of Professor F. Pellacini, after a 6-month collaboration with him as a research fellow. His research interests are in Computer Graphics, Geometry Processing, and Rendering, with a focus on porting interactive vector graphics design on dense meshes.

Fabio Pellacini Fabio Pellacini is a Professor of Computer Science at Sapienza University of Rome. Before joining Sapienza, Prof. Pellacini received his Laurea degree in Physics from the University of Parma, and his PhD in Computer Science from Cornell University, and worked at Pixar Animation Studios, Cornell University and Dartmouth College. For his research contributions, Fabio received an NSF CAREER award and an Alfred P. Sloan fellowship.

Enrico Puppo Enrico Puppo is a professor of Computer Science at the Department of Informatics, Bioengineering, Robotics and System Engineering of the University of Genova. Formerly, he has been a research scientist in the National Research Council of Italy. Enrico Puppo has co-authored about 150 peer-reviewed scientific publications in spatial data handling, computer graphics, geometric modeling, geometry processing, data visualization, shape analysis and understanding.