TriWild: Robust Triangulation with Curve Constraints

YIXIN HU, New York University, USA
TESEO SCHNEIDER, New York University, USA
XIFENG GAO, Florida State University, USA
QINGNAN ZHOU, Adobe Research, USA
ALEC JACOBSON, University of Toronto, Canada
DENIS ZORIN, New York University, USA
DANIELE PANOZZO, New York University, USA

We propose a robust 2D meshing algorithm, *TriWild*, to generate curved triangles reproducing smooth feature curves, leading to coarse meshes designed to match the simulation requirements necessary by applications and avoiding the geometrical errors introduced by linear meshes. The robustness and effectiveness of our technique are demonstrated by batch processing an SVG collection of 20k images, and by comparing our results against state of the art linear and curvilinear meshing algorithms. We demonstrate for our algorithm the practical utility of computing diffusion curves, fluid simulations, elastic deformations, and shape inflation on complex 2D geometries.

CCS Concepts: • **Mathematics of computing → Mesh generation**.

Additional Key Words and Phrases: Mesh Generation, Curved Triangulation, Robust Geometry Processing

ACM Reference Format:
Yixin Hu, Teseo Schneider, Xifeng Gao, Qingnan Zhou, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2019. TriWild: Robust Triangulation with Curve Constraints. *ACM Trans. Graph.* 38, 4, Article 52 (July 2019), 15 pages. https://doi.org/10.1145/3306346.3323011

1 INTRODUCTION

Triangle meshing is at the core of a large fraction of two-dimensional computer graphics and computer aided engineering applications, most commonly, used to solve PDEs or optimization problems on 2D domains, in the context of physical simulation, geometric modeling, animation and nonphotorealistic rendering. Major efforts have been invested in robustly generating meshes with linear edges with good geometric quality. However, the restriction to linear meshes makes precise reproduction of simple curved shapes, such as a Bézier curve, impossible independently of the resolution used, resulting in artifacts and/or excessive refinement in applications ranging from physical simulation to nonphotorealistic rendering. Curved meshes, i.e. meshes with curved edges, are an effective solution to this problem: the idea is to use curved triangles instead of linear ones, providing significantly superior geometric approximation of a shape using a mesh of a particular size. In most cases, the lower triangle count leads to an overall more efficient simulation for a given desired accuracy [Braess 2007; Ciarlet and Raviart 1972; Scott 1973, 1975]. A simple 2D example is shown in Figure 2, which has a geometric error of 2% of the overall area when 236 linear triangles are used, and the error can be reduced to numerical zero with the same number of curved triangles with a cubic Lagrangian geometric map (Figure 2). While the use of curved meshes is well established in the FEM literature (with a few applications in graphics [Boyé et al. 2012; Mezger et al. 2009]), the automatic generation of these meshes is rarely considered, and the few existing methods we tested have a high failure rate on real-world examples (Section 5).

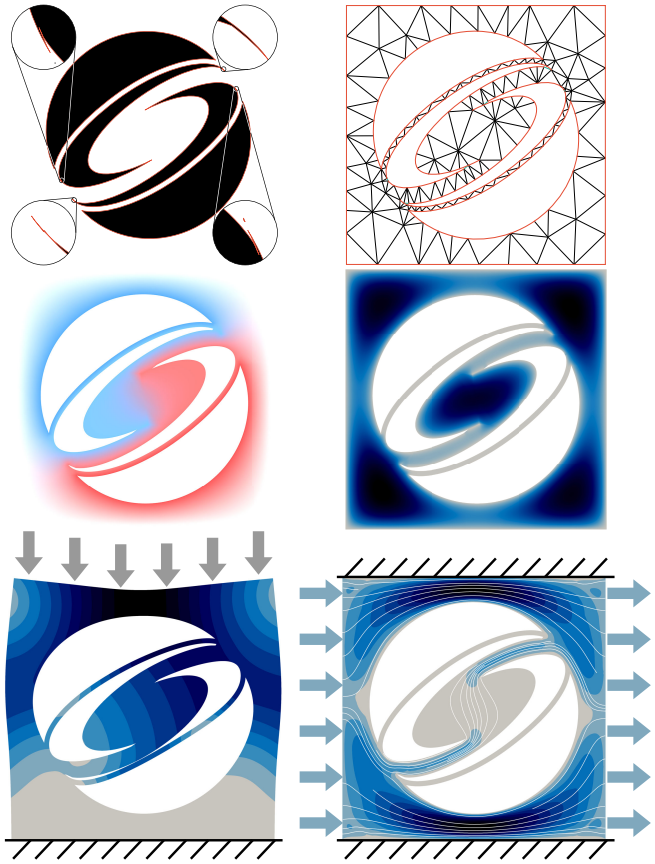


Fig. 1. The official ACM SIGGRAPH logo (www.siggraph.org/about/logos) is converted into a curved triangle mesh. We use the mesh to compute diffusion curves (Laplace), inflate surface (bilaplace), deform elastic bodies (Neo-Hooke), and simulate fluid flow (Stokes). Note that the imperfections in the input (shown in the closeups) are automatically healed by our method.

Authors' addresses: Yixin Hu, New York University, USA, yixin.hu@nyu.edu; Teseo Schneider, New York University, USA, teseo.schneider@nyu.edu; Xifeng Gao, Florida State University, USA, gao@cs.fsu.edu; Qingnan Zhou, Adobe Research, USA, qzhou@adobe.com; Alec Jacobson, University of Toronto, Canada, jacobson@cs.toronto.edu; Denis Zorin, New York University, USA, dzorin@cs.nyu.edu; Daniele Panozzo, New York University, USA, panozzo@nyu.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Association for Computing Machinery.
0730-0301/2019/7-ART52 $15.00
https://doi.org/10.1145/3306346.3323011

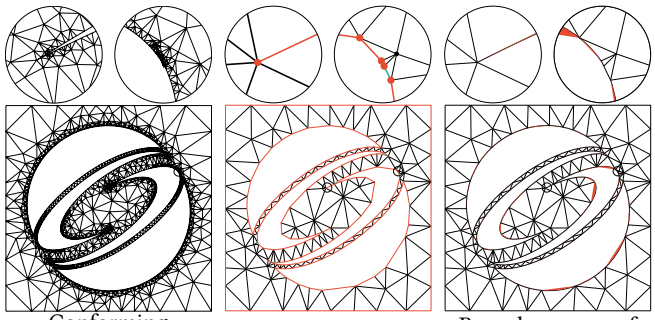


Fig. 2. Comparison of Conforming Delaunay Triangulation (left) and of our linear output (middle and right) corresponding to the model in Figure 1. The curved mesh in Figure 1 has only 236 triangles and it approximates the input exactly, without any geometric error, while the CDT linear mesh requires around 4 thousand triangles to have a comparable visual quality.

We propose a robust 2D meshing algorithm, *TriWild*, to generate high-quality curved meshes on complex 2D geometries. Our algorithm takes as input a 2D scene described as an SVG file (a soup of basic curved primitives, such as circles, ellipses, and Bézier curves), and automatically produces an analysis-ready, high-quality curved mesh. The algorithm starts by sanitizing the input curves and resampling them based on their curvature. This step is crucial, since "dirty" input is extremely common (see for a representative example the official SVG of the SIGGRAPH logo in Figure 1) and the preservation of degenerate features will inevitably lead to overrefined meshes not usable in downstream applications (Figure 2). The sanitized features are then meshed using a novel curved meshing algorithm that creates an initial linear mesh, curves its edges, and then maps each curved triangle to a reference domain (i.e., computes a *geometric map*). The algorithm internally uses rational coordinates for robustness and outputs a triangular mesh composed of cubic Bézier triangles with positive Jacobian in floating points coordinates. The created meshes are coarse, represent the input curves with high fidelity, and are directly usable to solve discrete PDEs.

We stress test TriWild on a large SVG collection, which is challenging even for existing robust linear triangular meshers. Our algorithm is able to handle even the most complex cases and produces meshes directly usable in FEM simulations. We also demonstrate the practical applicability of our algorithm in four common graphics applications: (1) color interpolation to create diffusion curves vector graphics [Boyé et al. 2012; Orzan et al. 2008], (2) viscous flow simulation in complex geometries [Stenberg 1984], (3) simulation of elastic deformations [Mezger et al. 2009], and (4) conversion of planar meshes into 3D models using surface inflation [Joshi and Carr 2008; Sýkora et al. 2014]. A reference implementation of our algorithm and a set of scripts to reproduce the results in the paper are provided at https://github.com/wildmeshing/TriWild.

2 RELATED WORK

Our method creates a linear triangulation and then bends the edges of the mesh to create a curved triangulation. Both types of triangulations have received significant attention in the meshing literatures.

2.1 Linear triangulations

Linear unstructured meshing in 2D is an old problem (e.g., [MacNeal 1949][1] or [Frederick et al. 1970]). There have been many papers, surveys and books written on this topic (e.g., [Cheng et al. 2012; Shewchuk 2012]). Existing works can be broadly categorized by their primary methodology and corresponding strengths and weaknesses.

Advancing front methods. generate a triangle mesh by growing a mesh in a flood-filling manner, typically growing inward from a given boundary [George 1971; Peraire et al. 1987; Sadek 1980]. While attractive because initial triangles placed near the starting regions can be high-quality and boundary-preservation is often trivial, the mesh quality typically gets progressively worse as the front continues. This culminates in a slew of issues when multiple fronts meet, where bad triangles are hard to avoid.

Grid/Quad-tree methods. are in some sense a complement to advancing front methods. These methods begin with a grid of high-quality triangles everywhere and then adjust triangles near the domain boundary [Baker et al. 1988; Bern et al. 1994; Yerry and Shephard 1983]. The methods are fast and can obtain good *average* quality since most triangles in the interior will have perfectly regular shape. However, these methods struggle to achieve accurate boundary preservation without sacrificing *worst-case* quality at boundary triangles.

Delaunay methods. are arguably the most widely used, in particular, the open source Triangle program by Shewchuk [1996]. Based on rigorous and well understood theory (e.g., [Aurenhammer 1991; Aurenhammer et al. 2013; Cheng et al. 2012; Shewchuk 1999]), triangulations boast good mathematical properties stemming from all or most triangles fulfilling the local Delaunay criteria. Categorizations of Delaunay methods generally split according to how they deal with one-dimensional line segment constraints. *Conforming* methods iteratively add points along constraints to pure Delaunay triangulation until each segment is covered by a union of Delaunay edges. While the number of necessary inserted points is bounded (e.g., Bishop [2016] proved by $O(n^{2.5})$ for an n-vertex input segment graph), the output meshes can be prohibitively over-dense near input features. To avoid over-refinement, a preprocessing guided by a user tolerance could be done to merge or re-align problematic features before applying conforming Delaunay [Busaryev et al. 2009]. In contrast, *constrained* methods relax the Delaunay requirement for input segments. This relaxation prevents an explosion in the vertex count, but introduces difficulty maintaining quality and robustness near features. We compare directly to the Delaunay triangulation libraries Triangle [Shewchuk 1996] and CGAL [Boissonnat et al. 2002], both of which implement conforming and constrained methods.

Improvement methods. attempt to increase the aggregate or worst-case quality of triangles in an existing mesh by local connectivity changes or vertex displacements [Canann et al. 1996, 1993; Lipman 2012]. The Optimal Delaunay Triangulation family choose a metric

[1] In his PhD thesis, MacNeal [1949] physically created a triangle mesh on drawing paper and, by measuring angles with a protractor, solved the 2D Poisson equation using the now famous cotangent formula.

that harmonizes with Delaunay methods and their duality with Voronoi diagrams [Chen and Xu 2004]. While strategies exist to encourage these methods to huge input domain boundaries [Alliez et al. 2005; Feng et al. 2018], they require a good, boundary-preserving initial starting point and generally do not support internal features. Our method follows the strategy of Hu et al. [2018] to create such an initial starting point for boundary and internal linear or curved features. We optimize the conformal AMIPS energy [Fu et al. 2015; Rabinovich et al. 2017] to measure and improve mesh quality.

2.2 Curved Triangulations

While linear meshes are predominant, curved meshes see frequent use in visual computing [Bargteil and Cohen 2014; Mezger et al. 2009] and engineering analysis [Bertrand et al. 2014a,b; Xue et al. 2005]. Meshes with curvilinear triangles offer a higher-order boundary approximation, enabling higher-accuracy simulations for smaller meshes [Babuška and Guo 1992, 1988; Bassi and Rebay 1997; Hughes et al. 2005; Luo et al. 2001; Oden 1994; Sevilla et al. 2011; Zulian et al. 2017]. To the best of our knowledge, all existing methods for constructing curved triangulations begin by creating a linear triangulation and then curve triangles to align with a curvilinear feature/boundary constraints. Our method is no exception.

Also to the best of our knowledge, all existing methods have been tested on small collections of comparatively simple models, and none of them can handle real-world, imperfect models (see Section 5.3 for our study on robustness of linear meshing methods, which are strictly simpler than high order methods). Our method is the first that has been tested on tens of thousands of real-world inputs.

Direct methods. split features into shorter curves and then create incident triangles on the interior of the domain (similarly to linear advancing front methods) [Dey et al. 1999] or directly fit or snap high-order nodes of a curved mesh (based on minimum distances) [Ghasemi et al. 2016]. The curved triangles are represented using Lagrange polynomials, [Dey et al. 1999], quadratic or cubic Bézier polynomials [George and Borouchaki 2012; Lu et al. 2013; Luo et al. 2002], or NURBS [Engvall and Evans 2017]. To capture the high-order curve or surface, while most techniques assume an isometric mapping between each element of the linear mesh and the corresponding high-order piece by evenly interpolating the parameters of linear vertices for high-order nodes, [Shephard et al. 2005; Sherwin and Peiró 2002] take into account the anisotropic property of the to-be-curved region to compute the high order node parametric positions accordingly.

Deformation methods. start with a linear mesh and elevate the degree of triangles to (so far, straight) high-order finite elements. By treating the triangulated domain as an elastic object, the mesh is deformed to curve triangles to match the input features. Different physical models have been employed, such as linear [Abgrall et al. 2012, 2014; Dobrzynski and El Jannoun 2017] and non-linear [Moxey et al. 2016; Persson and Peraire 2009; Poya et al. 2016] elasticity.

Distortion Metric, Inversion, and Intersections. Optimization methods are usually used as a post-processing step to attempt to untangle the inverted triangles created during the curving process and to improve their quality. Inverted triangles can be identified by extending the notion of area [Knupp 2000] to high-order function geometric maps [Engvall and Evans 2018; Johnen et al. 2013; Poya et al. 2016; Roca et al. 2012]. Various untangling strategies have been proposed, including geometric smoothing and connectivity modifications [Cardoze et al. 2004; Dey et al. 1999; Gargallo Peiró et al. 2013; George and Borouchaki 2012; Lu et al. 2013; Luo et al. 2002; Peiró et al. 2008; Shephard et al. 2005]. The mesh is then improved by optimizing various quality measures [Dobrzynski and El Jannoun 2017; Geuzaine et al. 2015; Karman et al. 2016; Roca et al. 2012; Ruiz-Gironés et al. 2017, 2016a,b; Stees and Shontz 2017; Toulorge et al. 2016; Ziel et al. 2017]. However, none of these methods can guarantee to produce an inversion-free curved mesh.

A different approach [Persson and Peraire 2009; Ruiz-Gironés et al. 2017] consists of initializing the optimization from a feasible inversion-free mesh, and preventing flips during deformation. Our method follows this approach, but unlike previous methods we do not sacrifice input feature preservation. Another issue is mesh overlaps due to intersections of curved boundary segments (not necessarily incurring flipped triangles). To the best of our knowledge, our method is the first to deal with this problem explicitly (Section 3.1).

Representations. of curved meshes vary: B-spline, implicit functions, and subdivision surfaces are typical high-order surface representations [Bruno and Pohlman 2003]. A few works assume the input being either an implicit function and fit B-spline patches [Peiró et al. 2008], or a linear boundary only and then try to obtain the high-order domain through the optimization of a linear mesh according to physical boundary conditions [Feng et al. 2018; Moxey et al. 2016; Poya et al. 2016; Ruiz-Gironés et al. 2017; Ziel et al. 2017]. The majority of the proposed works focus on polynomials, Bézier, and B-splines with a low degree (usually quadratic and cubic) [Dey et al. 1999; George and Borouchaki 2012; Geuzaine et al. 2015; Johnen et al. 2013; Lu et al. 2013; Luo et al. 2002; Peiró et al. 2008; Toulorge et al. 2013]. While Engvall and Evans [2017] show that exactly capturing NURBS patches is possible, their curved meshing algorithm is only applicable to clean CAD models, i.e. orientable, watertight, manifold, and without intersections, which are rare in practice. To the best of our knowledge, [Ruiz-Gironés et al. 2016b] is the first attempt on 2d meshing curved inputs with interior gaps. Since imperfect geometries are commonplace [Beall et al. 2003], all the existing curved meshing techniques are impractical in an automatic pipeline.

Curved Triangle Meshing Software. There are few 2D meshing software supporting curved triangles. To the best of our knowledge, the only ones available are in the Matlab Partial Differential Equation Toolbox [MATLAB Partial Differential Equation Toolbox 2018], GMSH [Geuzaine and Remacle 2009], and NekTar++ [Nek 2015; TUR 2018]. However, all of them have strict input requirements which are rarely met by vector drawings in the wild. Matlab only supports a CSG tree of circles, ellipses, and rectangles, greatly limiting its applicability. GMSH and NekTar++ both target the tessellation of domains specified in STEP format, using the OpenCASCADE engine to create initial linear triangle meshes on the interior of a parametric patch. Neither supports open or self-intersecting curves, which are

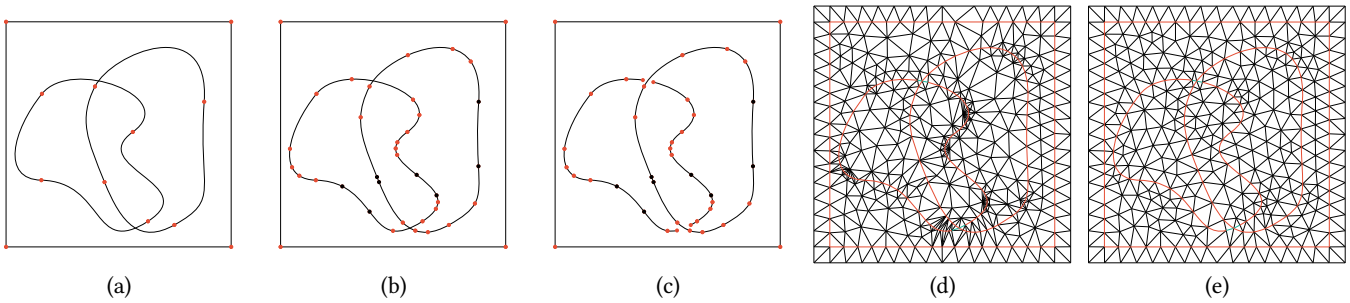


Fig. 3. Overview of the pipeline of our algorithm. The input piecewise-Bézier curves (a) are split at inflection points (black) and at optimal position to limit the total curvature (b), and finally all intersections are removed (c). This concludes the feature preprocessing and the features are first linearly meshed (d) and finally a curved mesh is obtained (e).

extremely common (in our dataset, this accounts for 99.95% of the inputs).

3 METHOD

Our curvilinear meshing algorithm (Figure 3) is divided into three stages: (1) analysis, filtering, and rounding of the input features (Section 3.1), (2) generation of a piece-wise linear initialization (Section 4.1), and (3) quality optimization and curving (4.2). The three stages are designed to work together, but can be used independently: for example, step (1) could be used as preprocessing for other linear triangle meshing pipelines to increase their robustness by sanitizing invalid inputs.

3.1 Input preprocessing and output

Commonly used 2D meshing algorithms and software make strong, often implicit assumptions about the quality of the input, usually requiring no self-intersection, no degeneracy, and no small angle between intersecting segments. However, these conditions are rarely met in real-world data, and their violation often results in either a meshing failure or over-refinement in the affected regions (Figure 2). In both cases, it might not be possible to solve PDEs in these domains, which then requires manual interaction to clean up the problematic regions. A possible solution is using a meshing tolerance [Hu et al. 2018; Mandad et al. 2015], i.e. allowing a controlled geometric error in the created mesh: if the tolerance is sufficiently large, these algorithms will automatically remove small features, preventing over-refinement and numerical problems due to imperfections in the input. However, this comes at the cost of approximating the input, which is particularly problematic around straight features, which might become jaggy (Figure 4).

We propose a different approach: we analyze the input curves to identify a subset (*primary feature curves*) that can be represented with a curved triangle mesh with a user-desired target edge-length l, and approximate the rest (*secondary feature curves*) with a piecewise linear mesh within an ϵ meshing tolerance.

Input Description. Our input is a feature soup $\mathcal{F} = \{\mathcal{P}, C\}$ of 2D isolated points ($\mathcal{P}$) and 2D cubic Bézier curves (C) representing the features of the scene. The parameters of the primitives are provided in double floating points.

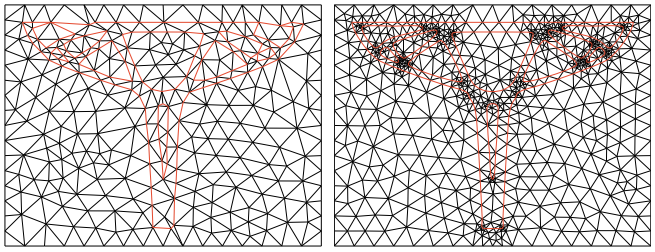


Fig. 4. Traditional meshing tolerance may lead to jaggy boundary even in simple configurations (left). Preserving features removes the problem (right).

We obtain the input feature soups by converting a subset of the SVG 2.0 standard (points, circles, ellipses, curves, and straight lines) into their cubic Bézier representation (replacing circles and ellipses with the Bézier approximation used by Adobe Illustrator) and we output our results in the gmsh format [Geuzaine and Remacle 2009].

Next, we define primary and secondary feature curve sets. The algorithms for obtaining these are described in Section 4. The primary feature curve set satisfies two conditions: bounded curvature and μ-separation.

Bounded Curvature. Let l_m be the user-controlled minimal edge length of the final mesh. Now consider a circle with curvature higher than $2/l_m$: since a triangle of this size cannot fit inside the circle it will be impossible to represent such a feature without refining more than what the user prescribed. We thus discard all the parts of curves whose local curvature is higher than $2/l_m$, and denote these new feature set as $\mathcal{F}_l$.

μ-separation. Ideally, we would like to preserve all features in $\mathcal{F}_l$. However, this might be impossible if we want to represent the output at a given resolution. A counterexample is simple to construct: take as input two parallel segments at a distance d. A triangle mesh that exactly represents both segments must contain at least one triangle between them, and the area of this triangle will tend to 0 as d tends to 0, leading to a possible inversion of the triangle due to floating point rounding errors. Similarly, two feature endpoints that are at a distance d between each other will force the insertion of two vertices in the final mesh, corresponding to the two endpoints, that

are also at a distance d. As d tends to zero, rounding errors might flip triangles that are in the neighbourhood. Even if we could somehow perturb the floating point coordinates to prevent inversions, the quality and size of these triangles will make the resulting mesh unusable for most downstream applications.

To avoid this problem, we formally identify these cases defining a local validity condition for sets of planar curves.

Definition 3.1. Let $\mathcal{F} \in \mathbb{R}^2$ be a collection of planar piecewise Bézier curves. The *μ-separated* set $\mathcal{F}_\mu$, is the subset of $\mathcal{F}_l$, such that for any point p on a feature f, the μ ball centered at p contains only a single connected component of f and no other feature curves or connected components of f.

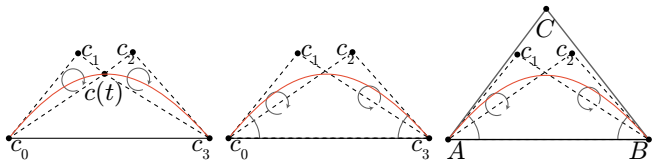
This definition ensures that no pair of points in the *μ-separated* set $\mathcal{F}_\mu$ can be closer than μ, except if they belong to the same curve and are connected by a part of the curve fully contained inside a μ ball. For convenience, we will refer to $\mathcal{F}_\mu$ as the *primary features*, and its complement $\mathcal{F} \setminus \mathcal{F}_\mu$, i.e. the parts of features that have been filtered out because of their high curvature or because of the μ-separation, as *secondary features*.

The parameters μ (feature envelope) and l_m (minimal edge length) control the desired level of approximation: a small μ will lead to denser meshes with more features tagged as primary and preserved more accurately in the final mesh (Figure 6). Similarly, a small ϵ (boundary envelope, Section 4.1) will produce secondary feature which better approximate the input secondary curves. We also use a numerical tolerance ϵ_m to account for rounding errors in the control points of the input curves.

Output Description. The output of our algorithm is a valid triangular mesh of a bounding box containing all input curves. Its edges are line segments or cubic Bézier curves, satisfying the following properties: (1) the edges do not intersect each other or the boundary of the bounding box, and (2) the edges in one-to-one correspondence with the primary features are within μ distance from their assigned feature (Section 4). While we do not have any formal guarantee on the preservation or approximation of the secondary features (Figure 3 (e) green edges), our algorithm strive to preserve them if the resolution allows and if they are not too close to primary features (which are given priority).

The coordinates of the output mesh vertices and control points for the curved edge are represented in double floating point representation. We use cubic Bézier curves for the edges since they can induce *volumetric* geometric map defined with cubic Lagrange triangles which, restricted to edges, correspond to the Bézier curve. We decide to use Lagrange bases for the map, since they are ubiquitously used in curved meshing applications and are supported by most FEM systems [Abaqus 2018; Ansys 2018; Geuzaine and Remacle 2009].

Note that, during our optimization we not only produce curved meshes but also try to produce a bijective geometric map expressed with Lagrange bases. Our algorithm could be adapted to produce curved meshes with NURBS edges, which would allow to reproduce ellipses and circles exactly; we leave this extension as a future work.



Lemma 3.2 Lemma 3.3 Theorem 3.4

Fig. 5. Notation for Lemma 3.2, Lemma 3.3, and Theorem 3.4.

3.2 Mesh Quality for Triangle Curving.

Our algorithm is based on a necessary and sufficient condition on the mesh quality (minimal angle) of a linear mesh, that ensures that its edges can be bent into Bézier curves without self-intersections. We will use this condition as a criteria to guide the discrete resampling of the input curves, which will be used to create an initial linear triangle mesh.

Lemma 3.2. *Let $c(t)$, $t \in [0,1]$, be a cubic Bézier curve with control points c_0, c_1, c_2, c_3, no self-intersection, and constant-sign curvature less than π, i.e. no inflection points (Figure 5, left). Then, for any $t \in [0,1]$, $c(t)$ is on the same side of line segments $[c_0, c_1]$ and $[c_2, c_3]$.*

Proof. Because of the constant-sign curvature, the rotation of the vector $c'(t)$ is always positive (or negative) prohibiting the curve to change direction. Additionally, since the curvature is less than π, the two segments $[c_0, c_1]$ and $[c_2, c_3]$ are in the same side of the line passing trough $[c_0, c_3]$. Therefore, for the curve to cross the edge $[c_0, c_1]$ and match the tangent at the end point it requires to a full turn which can be only achieved with a self intersection since degree 3 Bézier cannot spiral. $[c_2, c_3]$ follows by symmetry. □

Lemma 3.3. *Let $c(t)$, $t \in (0,1)$ be a cubic Bézier curve with control points c_0, c_1, c_2, c_3, without self-intersection, constant-sign curvature (no inflection points), and total curvature α strictly smaller than π (Figure 5, middle). Then the signed angles between the vectors $c_0\vec{c}_3$ and $c_0\vec{c}_1$ and $c_2\vec{c}_3$ and $c_0\vec{c}_3$ turns in opposite directions, and are smaller than α.*

Proof. The fact that two angles between the segments turn in opposite directions follows from Lemma 3.2: any point on the curve is on the right (and left side) of the tangent and in particular the intersection between the segments $[c_3, c_1]$ or $[c_0, c_2]$. Finally, since the total curvature is the integral of the curvature (which is constant-sign) is smaller than α, any tangent angles is smaller than α, and in particular the two at the endpoints which correspond to the angles $\angle c_1 c_0 c_3$ and $\angle c_0 c_3 c_2$. □

Theorem 3.4. *Let ABC be a triangle with minimal angle α, and $c(t)$, $t \in (0,1)$ be a cubic Bézier curve with control points A,c_1,c_2,B, no self-intersection, constant-sign curvature and total curvature α strictly smaller than π (Figure 5, right). Then $c(t)$ does not intersect the edges AC and CB, for any $t \in \{0..1\}$.*

Proof. Since that quadrilateral is contained in the triangle ABC, it follows that the curve will not intersect it. □

Theorem 3.4 provides a direct connection between linear mesh quality (measured as the minimal angle over the triangles of a linear

mesh) and the maximal turning angle of the feature curve assigned to one of linear mesh edges. We will fix a minimal angle $\alpha = 10$ degrees that the meshing algorithm will optimize for, and refine the input features to ensure that the angle between the tangents at the endpoints and corresponding edges does not exceed α. Note that this condition is only necessary but not sufficient to ensure a positive Jacobian of the Lagrangian geometric map assigned to the triangle, and it is thus only a very effective heuristic (Section 5).

4 FEATURE PROCESSING FOR CURVED MESHING

We introduce an algorithm to compute primary and secondary features from the set $\mathcal{F}$ of input features. This algorithm uses standard double floating point precision. The algorithm has six parameters: μ the feature envelope (distance between primary features), ϵ the boundary envelope (desired accuracy for secondary features, described below), ϵ_m (small number used to account for roundoff errors), α (desired minimal angle in the target mesh), l (desired edge length of the target mesh), and l_m (minimal edge length of the target mesh). In our experiments, we use μ =1e-3d, $\epsilon = 2\mu$, ϵ_m = 1e-8, $\alpha = 10$ degrees, $l = d/20$, and l_m =1e-4d with d the diagonal of the bounding box. It first splits the feature curves into elementary pieces (*discrete curve sections*) discarding some not satisfying primary feature constraints, and then classifies the remaining ones.

Removal of Degenerate Curves. We convert all degenerate curves which have all their control points too close together (i.e., contained in a circle of radius ϵ_m), into a feature point computed as the average of the control points. The rationale behind this choice is that any such curve will be too small to be represented with a mesh of target resolution and we thus opt to represent it as a single point in the output mesh.

We also identify all Bézier curves corresponding to straight segments by least-square fitting a line to the control points, and marking them as straight if the sum of distances of the control points to the line is smaller than ϵ_m. These features are replaced by line segments to avoid numerical problems in the next steps and to simplify the point to feature queries.

Inflection Point Split. We split each curve $c_i \in C$ at its inflection points, decomposing it into at most three curves with constant-sign curvature, which is a crucial property required by Theorem 3.4. The inflection points are computed explicitly by finding the parametric coordinates for which the curvature changes sign

$$\kappa \|c'(t)\|^3 = \det(c(t)', c(t)'') = 0, \tag{1}$$

and ensuring that $c(t)' \neq 0$.

Total Curvature Split. While each part of the curve has constant-sign curvature by construction, its total curvature is not bounded, potentially creating self-intersections in the curving step (Theorem 3.4). We thus recursively subdivide the curves until the turning number is smaller than 180 degrees to prevent self-intersections (Theorem 3.4). The optimal splitting point, which halves the total curvature, corresponds to the point in which the tangent is the average of the two endpoints tangents. This condition can be formulated as a *quadratic* equation in t

$$\det\left(\frac{c'(0) + c'(1)}{2}, c'(t)\right) = 0.$$

Note that when the angle between $c'(0)$ and $c'(1)$ is larger than π, we need to use minus the average tangent.

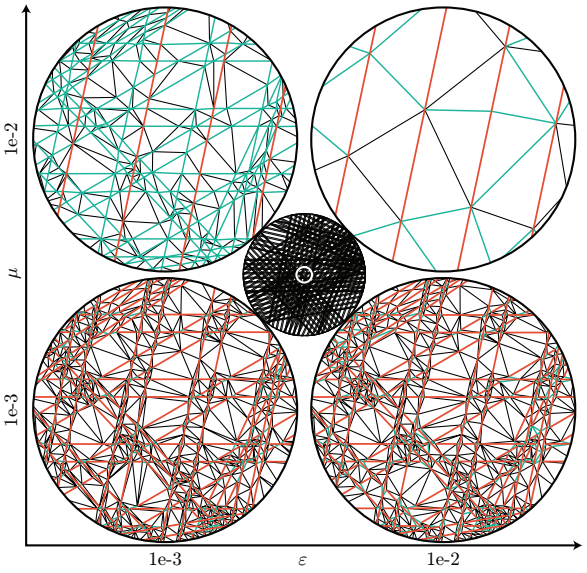


Fig. 6. Effect of the feature envelope μ and boundary envelope ϵ on the closeup (white circle in the middle) of an input mesh (center). The larger μ and ϵ (top right corner) are, the less primary (red) and secondary (green) features the final mesh will have. By enlarging μ (y-axis from bottom to top) more primary features will be present in the final result. A similar effect is obtained by enlarging ϵ (x-axis from left to right).

Curve Resampling. We discretize the resulting curves by sampling them recursively, splitting them in half at each step, until two conditions are satisfied: (1) the segments are shorter than the user-desired target edge length l and (2) the polyline approximation is within μ distance from the Bézier curve. From now on, each curve section will be denoted with the term *discrete curve section.*

Primary and Secondary Feature Tagging. We now compute a discrete version of a μ-separated soup of features, directly using the polyline approximating the curves. We greedily traverse all the discrete curve sections, and for each one we mark it as primary, check the μ-separated condition, and discard conservatively all the segments that violate it, marking them as secondary. The output of this stage is a valid μ-separated set features.

Secondary Feature Simplification. A second pass is used to prune close secondary features: every pair of feature endpoints closer than μ corresponding to secondary features are collapsed at their barycenter. This step is not strictly necessary, but it dramatically reduces the number of triangles generated by the BSP subdivision step, improving the running times in challenging models with thousands of self-intersections.

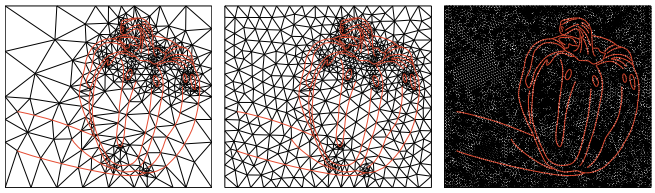


Fig. 7. Effect of the targeted edge length l on the output mesh.

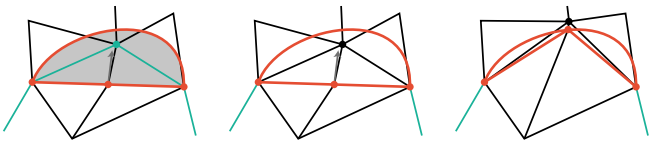


Fig. 8. The green vertex prevents the new orange point in the feature to move to the curve without creating inverted triangles (first image). We un-mark it so it is free to move (second feature) until is pushed away and the feature vertex is snapped to its feature (last image).

4.1 Linear Meshing

Similarly to many existing curved meshing algorithms (Section 2), we create an initial valid linear mesh and then curve its edges to match the feature curves. However, our pipeline differs from existing approaches: (1) we interleave mesh curving and quality mesh improvement and (2) we never allow the curved triangles to get inverted, i.e. we keep the Jacobian of their geometric map positive.

Generation of a Valid Linear Triangle Mesh. To construct an initial linear triangle mesh, we implement a 2D version of the TetWild algorithm proposed in [Hu et al. 2018] , trivially adapting all the steps to their 2D counterparts, which are both simpler and much more efficient than the volumetric version. We thus denote our algorithm as TriWild.

We use the same hybrid geometric kernel proposed in [Hu et al. 2018], using rational coordinates to avoid numerical problems in the first phase, and rounding them to floating point coordinates during quality optimization. The algorithm requires a parameter ϵ to control the size of boundary envelope. In Figure 6, we compared the results of different combinations of envelope ϵ and feature envelope μ. The influence of targeted edge lengths l and minimal edge length l_m on final output is shown in Figure 7. For the sake of brevity, we describe here the extensions required to handle the input curved features and we refer to [Hu et al. 2018] for the complete description of the algorithm.

Feature Invariants. Similarly to TetWild, TriWild preserves two invariants during the mesh generation and optimization: (1) secondary feature edges need to stay within the ϵ envelope, and (2) triangles cannot be inverted. We add two additional invariant for the primary feature edges: (3) the integrated curvature of the part of the parametric curve associated with every triangle edge must be smaller than α, and (4) primary feature vertices are allowed to only move on the curve. We simply discard all operations that violate any invariant, and we use our preprocessing (Section 3.1) to ensure that the invariants hold for the initial triangle mesh generated after BSP subdivision.

Feature Handling. TriWild iterates between four local operations for mesh improvement: (1) edge splitting, (2) edge collapsing, (3) edge swapping, and (4) vertex smoothing. Their behaviour and implementation are identical to their 3D counterpart in TetWild. The only exceptions are the vertices and edges corresponding to primary features: (1) when a feature edge is split, we place the inserted vertex in the middle of the parametric curve attached if it does not introduce inverted triangles, otherwise we place it in the middle of the linear edge, (2) when we collapse edges involving an endpoint of a curve, we always keep the endpoint in the same position, (3) we disallow swaps on primary and secondary feature edges, and (4) we restrict smoothing of vertices attached to a feature to lie on the feature itself (Appendix A). During all the operations, we explicitly keep track of the parametric position of all the vertices lying on primary features.

Vertex Projection. While unconditionally robust, triangle meshing using a BSP subdivision has the unfortunate side effect of potentially refining some of the edges corresponding to input features. This is problematic if the features are curved, since the inserted vertices will lie close, but not exactly on the feature. We thus add an additional step in the mesh improvement that moves every feature vertex as close as possible to its assigned feature (Figure 8) to enforce the fifth invariant. For each such vertex, we compute the closest point on the feature (Appendix B), and move as close as possible to it, while not violating any of the 3 invariants above. These vertices are not allowed to move further away from their closest point on the features. As the vertices move toward their target position, the rest of the mesh follows them since the smoothing operations strive to keep the quality high, eventually allowing these vertices to snap to the feature. While rare, it is possibly that some vertices in the region between the linear and the curved feature (Figure 8) cannot move due to Invariant (1). We thus delete the tagging for any secondary feature in these areas to allow the primary feature vertices to be snapped.

Termination Criteria. The quality optimization terminates when the minimal angle of the mesh is larger than α or the AMIPS energy is smaller than 10 (default value of TetWild), which is a good heuristic for curving the linear triangles (Theorem 3.4). Note that this condition does not guarantee that the elements will not be inverted during curving, but makes it less likely. We also stop the optimization if the maximum of iterations is reached. We have experimentally observed that for most models it is possible to stop the optimization prematurely, without affecting the quality in noticeable ways, and we thus used lower thresholds (10 degrees, AMIPS energy 30, max 80 iterations) for the large scale stress tests to reduce the overall running time at the cost of mesh quality.

4.2 Curvilinear Mesh Optimization

The result of the previous stage is a linear mesh which edges are assigned to a primary feature and which vertices have the corresponding parametric values. We now aim to construct a per-triangle

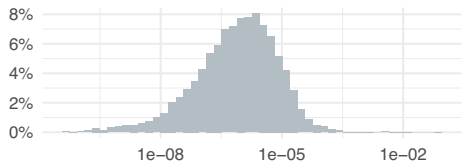


Fig. 9. Maximal least square fitting error with respect to d on a log scale.

bijective geometric map expressed in Lagrange form which, restricted to edges, corresponds to the curve. This curvilinear mesh optimization iteratively warps the edges of the linear mesh while optimizing its quality with local operations. Note that our optimization only bends the feature edges.

Splitting and Curving. To simplify all operations applied on a curved mesh, we split every triangle with more than one edge assigned to a curved feature until it has only one edge. We assign the cubic Lagrangian geometric map

$$g(x,y) = \sum_{i=0}^{9} v_i \ell_i(x,y)$$

(see Appendix C for the definition of $\ell_i(x,y)$) to each triangle with a feature edge (the first 3 v_i are the vertices of the triangle) and we compute its coefficients v_i by evaluating the curve $c(t)$ at $1/3$ and $2/3$ for the curved edge. For the remaining two linear edges, we sample them linearly, and use the average first 9 v_i for the position of the central node. This procedure is not guaranteed to produce a valid geometric map due to 2 reasons: (1) the solution might not exist (avoiding intersections between the boundary segments of a Lagrangian triangle is not a sufficient condition to ensure an overall positive Jacobian), and (2) even if it exists it might not be in the simple form we just described. We thus skip problematic triangles in this stage (assigning them a linear geometric map) and we improve the quality of the mesh further, increasing the probability that the geometric map fit will produce a bijective map.

Curved Mesh Optimization. The mesh quality is improved using the same local operations used in Section 4.1, but with less constraints, since it is not necessary anymore to enforce the total curvature invariant. Since the mesh is now using a non-linear geometric map, the Jacobian is no longer constant in each triangle, and checking for validity is more expensive [Geuzaine et al. 2015]. Note that each operation on a curved triangle potentially modifies the curved edges and requires to update the Lagrange coefficients v_i to *always* match the associated feature.

Termination Criteria and Least Squares Fitting. The optimization terminates when the user-controlled quality threshold is achieved (AMIPS Energy 20), or after a user-controlled number of iterations (default 10) is reached. If there are curved triangles which are still impossible to curve exactly using the simple Lagrange basis (i.e., the Jacobian of the geometric mapping is negative), we do a best-effort and fit them only in a least square sense by optimizing for the Lagrangian coefficients v_i that best approximate the boundary curve (minimize the distance between the two curves) under the constraint of having a positive Jacobian. This happens to 44.7% meshes of all our outputs and 0.18% faces per mesh (those with fitting) in average. The overall max error in our experiments is 6.99e-2d (mean 1.87e-5d, std 6.99e-4, Figure 9), indicating a faithful reproduction of the input features.

Fig. 10. Random selected examples of curved meshes obtained with our method on the Openclip dataset.

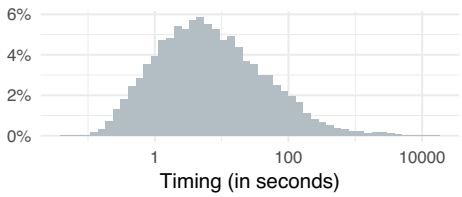


Fig. 11. Timing of our curved pipeline on a log scale.

5 RESULTS

We implemented our algorithm in C++, using Eigen [Guennebaud et al. 2010] for linear algebra routines. The source code of our reference implementation is available at https://github.com/wildmeshing/TriWild. The experiments were performed on cluster nodes with 2 Xeon E5-2690v4 2.6GHz CPUs and 250GB memory, each with 64GB of reserved memory, and allowed for a maximum running time of 6 hours. To test our curved mesh generation we crawled 19,686 SVG images from openclipart.org, a free SVG image repository. For each SVG image we extract the set of features $\mathcal{F}$ which are used in our algorithm. Figure 10 shows some examples of the created curved meshes. Within the time limit of 6 hours we successfully created 19,685 curved meshes with only one failure due to large input size. Figure 11 summarizes the running time of our curved pipeline. Since we always reject operations (included the fitting) introducing inverted elements, the only possible failure is inherited by TetWild: some vertices might fail to be rounded into floating points. However, this never happened in our experiments.

5.1 Applications

The main application of meshing is simulation, which aims to solve a partial differential equation (PDE) for an unknown function u defined over a domain Ω subject to some boundary constraints. The role of the function u depends on the application, for instance in elasticity it is displacement, while for fluids it is velocity. One of the typical numerical methods to solve PDEs is the finite element method (FEM). As the name suggests, the first essential step of a FEM consist of meshing the domain (thus creating the triangles) which can be done with both linear or curved meshes. In fact, for FEM simulations, one requires only a *bijective map* from the reference triangle (unit right-angle triangle) to the actual physical triangle, without any assumptions about the linearity of the map. Changing the *geometric* order of the triangles (e.g., from linear to cubic) only changes the local assembler but not affects the size and sparsity of the final linear system. In fact, it only requires a slightly higher quadrature order, leading to similar performance as linear triangles.

The main advantage of high order geometries is that we can use coarse meshes, which leads to faster simulations, without introducing any geometrical error on the boundary description. We now demonstrate the effectiveness of our curved meshes for different applications using different PDEs. We focus our applications to standard PDEs used in graphics, and remark that other more complex equations might benefit more from high-order geometries [Bassi and Rebay 1997]. All experiments are done on cubic meshes (the geometric map used is cubic Lagrange polynomials) with quadratic

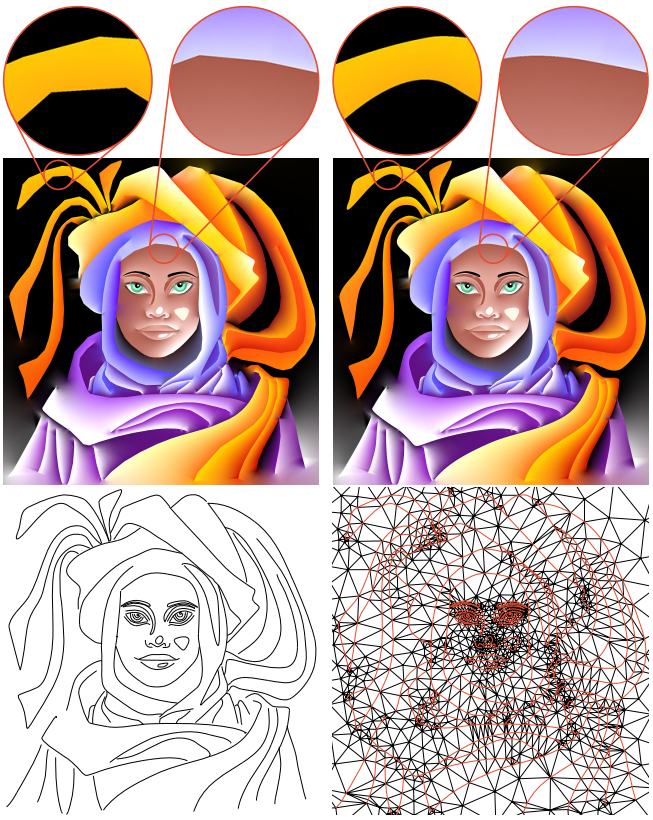


Fig. 12. Example of diffusion curves for a linear (top left) and curved (top right) mesh. The bottom left is the input and the bottom right is the our curved output.

Lagrange polynomials as basis functions to represent the solution. When we compare against meshes with straight edges, we replace the cubic geometric map with a linear one. We stress that replacing the linear geometric map with a cubic one produces systems with *exactly* the same size and sparsity, the only difference are the entries of the matrices, which are slightly more expensive to compute for the high-order map.

Laplacian. The simplest, and by far the most popular in graphics, PDE is the Laplace equations

$$\Delta u = f, \quad u = g \quad \text{for } x \in \partial\Omega,$$

where $\partial\Omega$ is the boundary of the domain. A popular graphics application of such equation is diffusion curves [Boyé et al. 2012; Orzan et al. 2008]. A set of curves is augmented with 2 colors, one on the left and one on the right. The final image is generated by diffusing the colors from the input colored curves. In other words, the mesh is cut open along the curves, the left and right colors play the role of boundary conditions, and the solution is the color at every pixel. We generate a curved mesh with our method from the annotated curves in [Orzan et al. 2008] and run the diffusion simulation with and without curved triangles, Figure 12. We clearly see that, at the same resolution, the curved mesh provides a superior result, avoiding

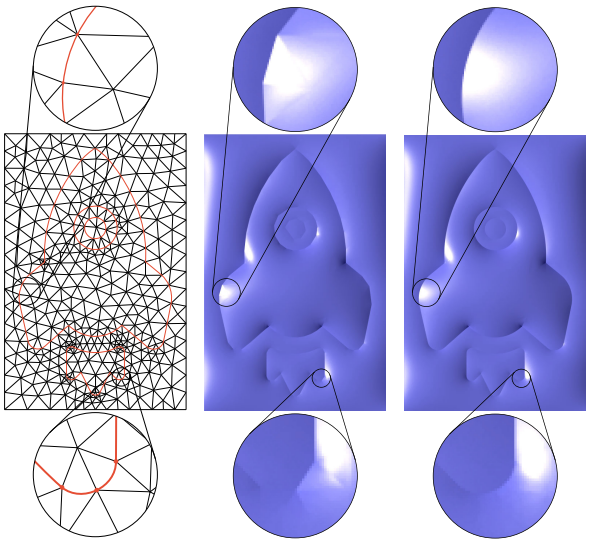


Fig. 13. Example of inflation on a linear (middle) and curved (right) mesh of the same resolution, the input mesh is shown in the (left).

approximation artifacts on the diffusion curves. The runtime for the linear mesh is 0.074s, and switching to cubic geometric map only adds 0.016 seconds.

Bi-Laplacian. For many graphics applications, the Bi-Laplacian equation is preferred since it produces smoother results around the boundary.

$$\Delta^2 u = 0, \quad u = g \quad \text{for } x \in \partial\Omega,$$

To solve this PDE we require mixed finite elements [Monk 1987] or C^1 basis functions [Boyé et al. 2012], we opted for the former since it is simpler to implement. Among many application of the bi-Laplacian, we picked surface inflation [Joshi and Carr 2008; Sýkora et al. 2014]. The idea is elegant: the mesh is fixed at the curves (zero boundary conditions) and a force is applied to the curves to "lift" the mesh. We follow the Rèpousse [Joshi and Carr 2008] construction but instead of using the curvature to inflate we use a constant value. Figure 13 shows the results for the same mesh with and without curved triangles: similarly as before, the curved mesh do not exhibit any visible artifacts, and the runtime is only slightly (0.2%) higher for the curved mesh.

Elasticity. In Figure 14, we study the difference of an elastic deformation using the Neo-Hookean elasticity model, where the stress σ relationship is not linear with respect to the displacement u:

$$-\text{div}(\sigma[u]) = f \quad \sigma[u] = \mu(F[u] - F[u]^{-T}) + \lambda \ln(\det F[u]) F[u]^{-T},$$

where $F[u] = \nabla u + I$ and λ, μ are material parameters. Note that due to the non-linearity of the PDE, the solution method requires a non-linear solver such as the Newton method. We use a simple setup, a square domain with a circular hole in the middle is hanged on the top (zero Dirichlet) and gravity is applied. Even in the context of non-linear PDE the overhead of curved triangles is negligible (curved 0.0975s, linear 0.0857s) and curved meshes produce results

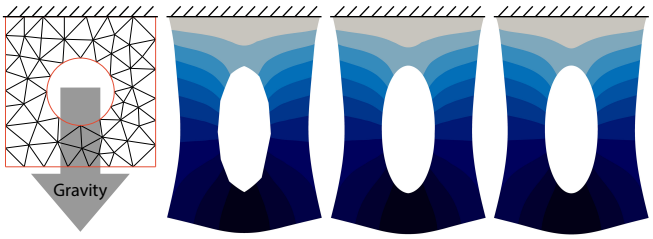


Fig. 14. Example of elastic deformation of linear (second figure) and curved (third figure) triangles. A solution on a dense mesh is also provided for reference (fourth figure). Note how the curved mesh provides a much more accurate solution than the linear mesh, especially around the hole. The coarse mesh used in both simulations is shown in the first figure. The color shows the y-displacement.

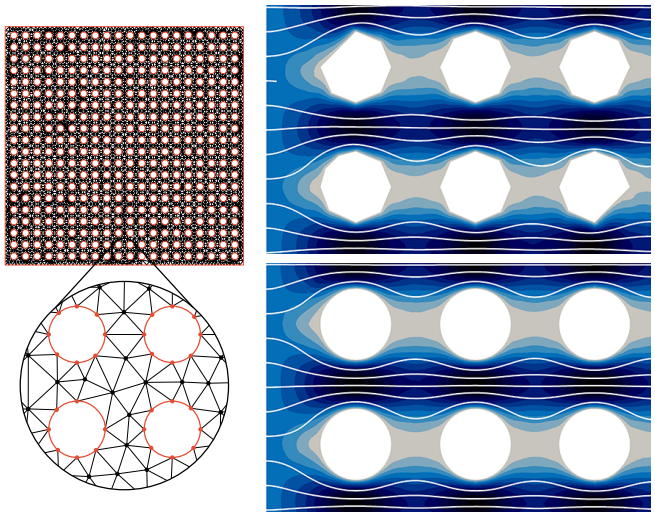


Fig. 15. Example of Stokes fluid simulation for small circular obstacles (left) on linear (right top) and curved (right bottom) triangles. The color shows the norm of the velocity of the fluid, and the lines are the stream lines.

closer to the ground truth solution computed on a dense mesh, Figure 14.

Stokes. The final application we consider is fluid simulation, where we are looking for the fluid velocity u of a fluid. We solve the Stokes equation

$$-\mu\Delta u + \nabla p = 0$$
$$-\text{div} u = 0,$$

where p is the pressure and μ is the viscosity of the fluid. The experiment consists of a pipe filled with small circular obstacles with a fluid passing through (Figure 15 left), a setup used for studying cancer cell migration within interstitial tissues [Panagiotakopoulou et al. 2016]. In other words, we have a constant non-zero boundary condition on the left and right side of the domain (in and out flow velocities) and zero velocity on the rest of the boundary (i.e., the top and bottom, and the obstacles). Figure 15 shows a close-up of the results of the simulation for curved and linear meshes. Because

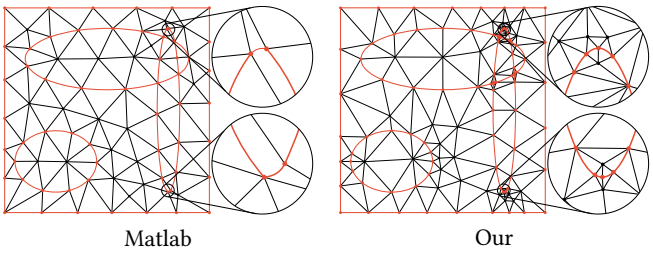


Fig. 16. Comparison between Matlab (left) and our (right) mesh for a simple set of ellipses.

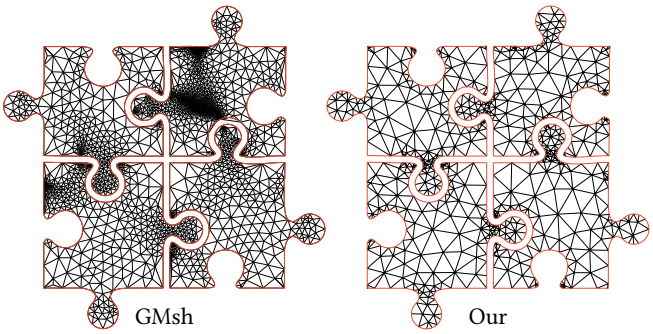


Fig. 17. Comparison of creating a curved cubic mesh for a puzzle piece with GMsh (left) and our method (right). GMsh mesh contains 19 inverted triangles, whereas ours has none and it can be directly used in downstream applications.

of the poor approximation of the linear mesh, the behaviour of the fluid is asymmetric and unnatural. While simulation on linear mesh takes 2.65s, it takes 2.70s on curved mesh where the difference is small.

5.2 Comparison with Curved Meshers

To the best of our knowledge only three existing available software allow to generate curved meshes that preserve input curve features: the Matlab PDE toolbox [MATLAB Partial Differential Equation Toolbox 2018], GMsh [Geuzaine and Remacle 2009], and NekTar++ [Nek 2015]. The main difference between these software and our solution is that they require a CSG tree (Matlab) or a boundary curve which clearly defines an interior domain (GMsh, NekTar++). With these solutions it is impossible to mesh a set of *open* input curves, thus limiting the applications, for instance diffusion curves or inflation (Section 5.1) would be impossible. We manually created 2 simple examples and provided a representative comparison to one method for each category.

Matlab. According to the Matlab documentation[2], the 2D meshing package exploits constructive solid geometry (CSG), which uses a set of solid blocks: square, rectangle, circle, ellipse, and polygon. This short list of primitives limits the applications considerably, for instance it is not possible to represent even closed polybezier curves. Another major limitation of the toolbox is that it supports only linear and quadratic triangles, leading, for instance, to poor approximations of circular arcs. To test Matlab we setup a simple example, a unit square with 3 ellipses curves added (Appendix D) and produced the SVG for our method, Figure 16. Note that our method constructs a dense high-quality mesh around the high-curvature tip of the ellipse while Matlab resorts to large low quality triangles.

While constructing this example we discover 2 problems in the Matlab mesher: if we shrink the last ellipse first semi-axis from 0.055 to 0.54 it produces an error and cannot generate any valid output; by shrinking it even more it goes in infinite loop, while our method is unaffected by these changes.

[2]https://www.mathworks.com/help/pde/geometry.html

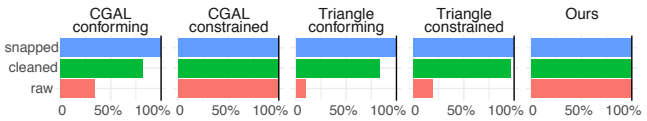


Fig. 18. The success rate of generating a triangulation over the raw, cleaned and snap rounded Openclipart data set.

GMsh. GMsh requires the curves to define a closed domain. Since GMsh exactly reproduces the input boundaries, it overrefines near the defects (Figure 17). Additionally, it relies on an "untangling" strategy: it first generates a possibly invalid curved mesh, and then tries to "un-invert" the triangles. For the example in Figure 17, the untangling fails and the mesh still contains 19 inverted triangles, making it unsuitable for FEM simulations.

5.3 Linear Meshing Comparison

Our algorithm can be used to robustly generate traditional linear triangles meshes, by simply loading a collection of points and line segments, and marking all the segments as secondary features (Section 4). We compare extensively with two popular open source 2D meshing libraries, implementing the current state of the art triangulation algorithms: Triangle [Shewchuk 1996] and CGAL's 2D Triangulation module [Boissonnat et al. 2002]. Both libraries are capable of taking a set of (potentially intersecting) segments as constraints and generating either a constrained Delaunay triangulation (CDT) or a conforming Delaunay triangulation (RDT) as output. We compare our results with both CDT and RDT results generated by these libraries on three sets of input: (1) raw input, (2) cleaned input, and (3) snapped input. Raw input contains piece-wise linear approximations of 19,686 SVG images crawled from openclipart.org The clean input is obtained by (1) removing duplicated vertices, (2) removing duplicated edges, and (3) removing degenerate edges. Note that intersecting segments are not fixed in the cleaning process, since all methods supports them. Lastly, the snapped input is the output of iteratively snap rounding [Goodrich et al. 1997] the raw input using ϵ as the pixel size. For linear comparison, we limit the maximum computing resources for each input to 1 hour running time and 16 GB memory.

Success Rate. The first, and simplest, metric is to check if the algorithm successfully generated a non-empty output. Figure 18 compares all methods on the three types of inputs. For snapped inputs (where there is no intersections), all methods succeeds nearly

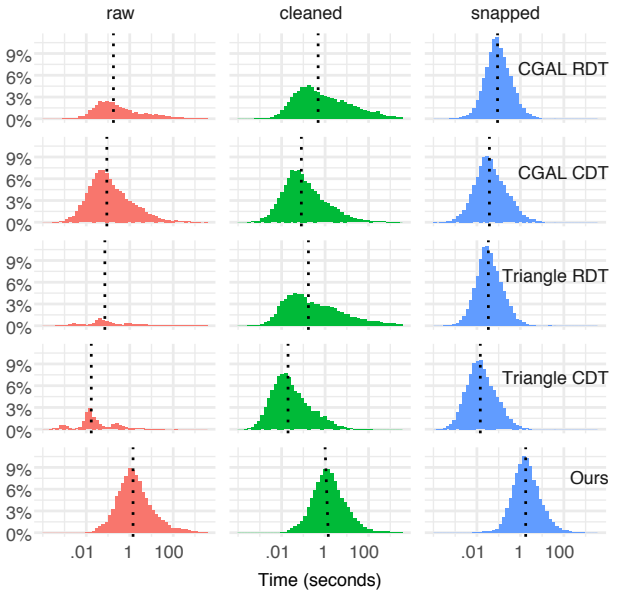


Fig. 19. The total running time of all triangulation algorithms.

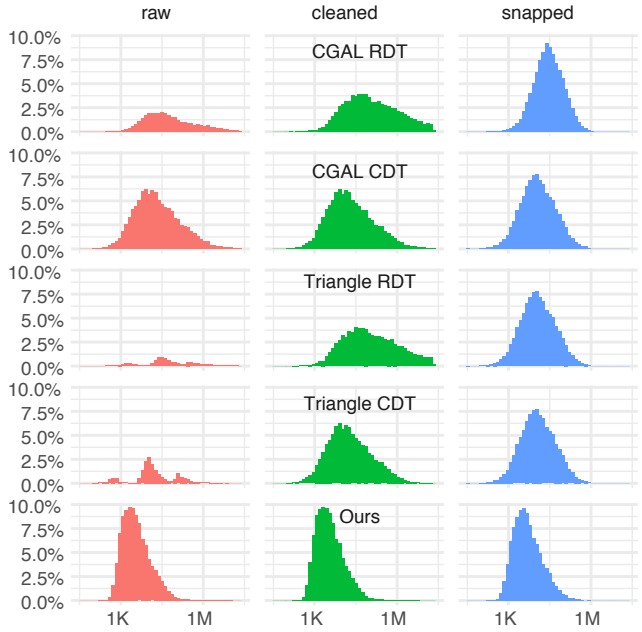


Fig. 20. Comparison of the number of triangles generated by each method on a log scale.

100%. However on cleaned inputs, RDT from both CGAL and Triangle fails due to the presence of intersecting segments. Only our approach and CGAL CDT are robust enough against the rampant presence of intersecting and degenerate segments from the raw input. It is interesting to observe that the single model that produces invalid output with Triangle RDT is not the same one that produces an invalid output for CGAL RDT.

Figures 19 and 20 shows the total running time and the total number of triangles generated by all methods on all three sets of inputs. Although slower, our algorithm along with CGAL's CDT are the only methods not requiring input preprocessing to be robust. Note that the preprocessing step can be very expensive (Figure 21) and will likely dominate the running time when paired with a triangle meshing algorithm that requires clean input. Note that all timing plots are using logarithmic scale for x-axis.

Correctness and Quality. Table 1 lists the number of invalid triangulations generated by each method, i.e. triangulation containing one or more inverted triangles. We use exact predicates to check for triangle inversions [Shewchuk 1997]. Our algorithm generates inversion-free triangulation for all inputs, while both Triangle and CGAL produce invalid outputs occasionally. Figure 22 illustrates one of the many triangulation quality measures [Shewchuk 2002]: min edge to max edge ratio. Our algorithm produces more well-shaped triangles than CDT, but slightly worse that RDT. The number (Figure 20) and quality (Figure 22) of the triangles generated by our method is mostly independent from the preprocessing done to the input, suggesting that our algorithm is stable to small perturbation in the input. This is not the case for other methods, which exhibit major differences in both number of elements and quality.

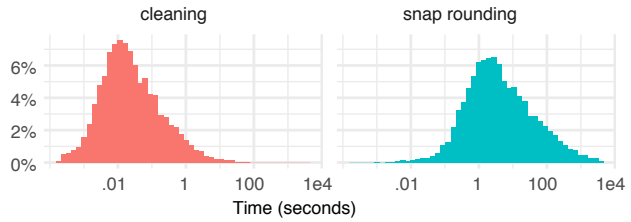


Fig. 21. The running time of two preprocessing algorithms we used. Note that snap rounding is much more expensive than triangulation in general.

Table 1. Total Number of Failed Results for All Methods.

	Triangle		CGAL		Ours*
	CDT	RDT	CDT	RDT	
raw	16010 (67)	17806 (1)	3743 (3585)	12945 (1)	13 (0)
cleaned	645 (24)	3329 (0)	524 (457)	3560 (0)	83 (0)
snapped	5 (0)	5 (0)	5 (0)	38 (0)	5 (0)

Note: The failure refers to no output or output with inverted triangles. Numbers in parenthesis represent the number of output triangulations that contains inversions.
*While our algorithm occasionally timed out due to limited computing resources, we have validated that it can always succeed with a larger epsilon and 64G memory.

6 LIMITATIONS AND CONCLUDING REMARKS

We introduce an algorithm to create curved triangular meshes preserving features from a large collection of real-world SVG drawings. We demonstrated that our algorithm supports many applications in computer graphics, and compares favorably against existing triangle meshers, producing coarser meshes due to its native ability to

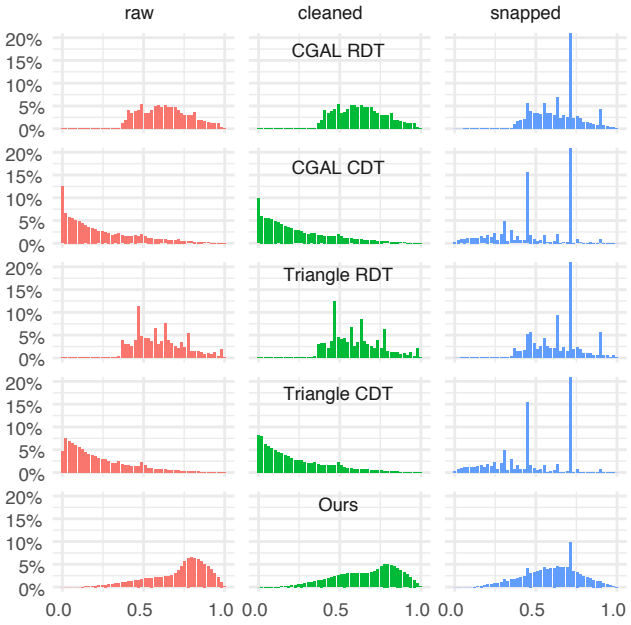


Fig. 22. Edge ratio distribution on 1,000 randomly sampled outputs for each method.

filter out small scale features that are smaller than a user-controlled epsilon.

The main limitation of our algorithm is that it does not guarantee to exactly preserve intersections between curves. While our current algorithm produces visually pleasing results in these cases, we would like to explore the use of an exact Bezier arrangement [Wein et al. 2018] of the input features, to address this issue. A second interesting direction for future work is the creation of meshes with geometric maps of arbitrary degree or with rational Bezier edges, able to exactly reproduce circular arcs. While extending our algorithm will require minor modifications, their use in FEM is unclear, since there are no standard elements that reproduce them. Compared to other linear triangle meshers, our algorithm is around 10 times slower. We believe that parallelization of the mesh optimization stage would provide a noticeable performance boost.

We expect our contribution and our reference implementation to have a large impact in computer graphics and mechanical engineering, by considerably lowering the efforts required to build curved (and linear) meshes and use them for FEM simulations.

ACKNOWLEDGMENTS

This work was supported in part through the NYU IT High Performance Computing resources, services, and staff expertise. This work was partially supported by the NSF CAREER award under Grant No. 1652515, the NSF grant IIS-1320635, the NSF grant DMS-1436591, the NSF grant 1835712, the SNSF grant P2TIP2_175859, NSERC Discovery Grants (RGPIN-2017-05235 & RGPAS-2017-507938), Canada Research Chair award, Connaught Fund, a gift from Adobe Research, and a gift from nTopology.

REFERENCES

2015. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications* 192 (2015), 205 – 219. https://doi.org/10.1016/j.cpc.2015.02.008

2018. Curvilinear mesh generation using a variational framework. *Computer-Aided Design* 103 (2018), 73 – 91. https://doi.org/10.1016/j.cad.2017.10.004 25th International Meshing Roundtable Special Issue: Advances in Mesh Generation.

Abaqus. 2018. Abaqus. http://www.feasol.com

Remi Abgrall, Cécile Dobrzynski, and Algiane Froehly. 2012. *A method for computing curved 2D and 3D meshes via the linear elasticity analogy: preliminary results.* Research Report RR-8061. INRIA. 15 pages. https://hal.inria.fr/hal-00728850

R. Abgrall, C. Dobrzynski, and A. Froehly. 2014. A method for computing curved meshes via the linear elasticity analogy, application to fluid dynamics problems. *International Journal for Numerical Methods in Fluids* 76, 4 (2014), 246–266. https://doi.org/10.1002/fld.3932 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.3932

Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. 2005. Variational Tetrahedral Meshing. *ACM Trans. Graph.* 24, 3 (July 2005), 617–625. https://doi.org/10.1145/1073204.1073238

Ansys. 2018. Ansys. https://www.ansys.com

Franz Aurenhammer. 1991. Voronoi Diagrams&Mdash;a Survey of a Fundamental Geometric Data Structure. *ACM Comput. Surv.* 23, 3 (Sept. 1991), 345–405. https://doi.org/10.1145/116873.116880

Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. 2013. *Voronoi Diagrams and Delaunay Triangulations.* WORLD SCIENTIFIC. https://doi.org/10.1142/8685 arXiv:https://www.worldscientific.com/doi/pdf/10.1142/8685

I. Babuška and B.Q. Guo. 1992. The h, p and h-p version of the finite element method; basis theory and applications. *Advances in Engineering Software* 15, 3 (1992), 159 – 174. https://doi.org/10.1016/0965-9978(92)90097-Y

I. Babuška and B. Q. Guo. 1988. The h-p Version of the Finite Element Method for Domains with Curved Boundaries. *SIAM J. Numer. Anal.* 25, 4 (1988), 837–861. http://www.jstor.org/stable/2157607

Brenda S. Baker, Eric Grosse, and Conor S. Rafferty. 1988. Nonobtuse triangulation of polygons. *Discrete & Computational Geometry* 3, 2 (01 Jun 1988), 147–168. https://doi.org/10.1007/BF02187904

Adam W. Bargteil and Elaine Cohen. 2014. Animation of Deformable Bodies with Quadratic Bézier Finite Elements. *ACM Trans. Graph.* 33, 3, Article 27 (June 2014), 10 pages. https://doi.org/10.1145/2567943

F. Bassi and S. Rebay. 1997. High-Order Accurate Discontinuous Finite Element Solution of the 2D Euler Equations. *J. Comput. Phys.* 138, 2 (1997), 251 – 285. https://doi.org/10.1006/jcph.1997.5454

Mark W Beall, Joe Walsh, and Mark S Shephard. 2003. Accessing CAD Geometry for Mesh Generation. In *Proceedings of the 12th International Meshing Roundtable.* 33–42.

Marshall Bern, David Eppstein, and John Gilbert. 1994. Provably good mesh generation. *J. Comput. System Sci.* 48, 3 (1994), 384 – 409. https://doi.org/10.1016/S0022-0000(05)80059-5

Fleurianne Bertrand, Steffen Muñzenmaier, and Gerhard Starke. 2014a. First-order System Least Squares on Curved Boundaries: Higher-order Raviart–Thomas Elements. *SIAM J. Numer. Anal.* 52, 6 (2014), 3165–3180.

Fleurianne Bertrand, Steffen Muñzenmaier, and Gerhard Starke. 2014b. First-Order System Least Squares on Curved Boundaries: Lowest-Order Raviart–Thomas Elements. *SIAM J. Numer. Anal.* 52, 2 (2014), 880–894.

Christopher J. Bishop. 2016. Nonobtuse Triangulations of PSLGs. *Discrete & Computational Geometry* 56, 1 (2016).

Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. 2002. Triangulations in CGAL. *Computational Geometry* 22 (2002), 5–19.

Simon Boyé, Pascal Barla, and Gaël Guennebaud. 2012. A Vectorial Solver for Free-form Vector Gradients. *ACM Trans. Graph.* 31, 6, Article 173 (Nov. 2012), 9 pages. https://doi.org/10.1145/2366145.2366192

Dietrich Braess. 2007. *Finite Elements* (third ed.). Cambridge University Press. Cambridge Books Online.

Oscar P Bruno and Matthew M Pohlman. 2003. High order surface representation. *Topics in Computational Wave Propagation, Direct and Inverse Problems* (2003).

Oleksiy Busaryev, Tamal K. Dey, and Joshua A. Levine. 2009. Repairing and Meshing Imperfect Shapes with Delaunay Refinement. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling (SPM '09).* ACM, New York, NY, USA, 25–33. https://doi.org/10.1145/1629255.1629259

S. A. Canann, S. N. Muthukrishnan, and R. K. Phillips. 1996. Topological refinement procedures for triangular finite element meshes. *Engineering with Computers* 12, 3 (01 Sep 1996), 243–255. https://doi.org/10.1007/BF01198738

Scott A. Canann, Michael B. Stephenson, and Ted Blacker. 1993. Optismoothing: An optimization-driven approach to mesh smoothing. *Finite Elements in Analysis and Design* 13, 2 (1993), 185 – 190. https://doi.org/10.1016/0168-874X(93)90056-V

David Cardoze, Alexandre Cunha, Gary L. Miller, Todd Phillips, and Noel Walkington. 2004. A Bézier-based Approach to Unstructured Moving Meshes. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry (SCG '04).* ACM, New York, NY, USA, 310–319. https://doi.org/10.1145/997817.997864

Long Chen and Jin-chao Xu. 2004. Optimal delaunay triangulations. *Journal of Computational Mathematics* (2004), 299–308.

Siu-Wing Cheng, Tamal K. Dey, and Jonathan Shewchuk. 2012. *Delaunay Mesh Generation.*

Philippe G Ciarlet and P-A Raviart. 1972. Interpolation theory over curved elements, with applications to finite element methods. *Comput. Meth. Appl. Mech. Eng.* 1, 2 (1972), 217–249.

Saikat Dey, Robert M. O'Bara, and Mark S. Shephard. 1999. Curvilinear Mesh Generation In 3D. In *IMR*. John Wiley & Sons, 407–417.

Cecile Dobrzynski and Ghina El Jannoun. 2017. *High order mesh untangling for complex curved geometries.* Research Report RR-9120. INRIA Bordeaux, équipe CARDAMOM. https://hal.inria.fr/hal-01632388

Luke Engvall and John A. Evans. 2017. Isogeometric unstructured tetrahedral and mixed-element Bernstein-Bézier discretizations. *Computer Methods in Applied Mechanics and Engineering* 319 (2017), 83 – 123. https://doi.org/10.1016/j.cma.2017.02.017

Luke Engvall and John A. Evans. 2018. Mesh Quality Metrics for Isogeometric Bernstein-Bézier Discretizations. *arXiv:1810.06975* (2018).

Leman Feng, Pierre Alliez, Laurent Busé, Hervé Delingette, and Mathieu Desbrun. 2018. Curved Optimal Delaunay Triangulation. *ACM Trans. Graph.* 37, 4 (2018).

C. O. Frederick, Y. C. Wong, and F. W. Edge. 1970. Two-dimensional automatic mesh generation for structural analysis. *Internat. J. Numer. Methods Engrg.* 2, 1 (1970), 133–144. https://doi.org/10.1002/nme.1620020112 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.1620020112

Xiao-Ming Fu, Yang Liu, and Baining Guo. 2015. Computing Locally Injective Mappings by Advanced MIPS. *ACM Trans. Graph.* 34, 4, Article 71 (July 2015), 12 pages. https://doi.org/10.1145/2766938

Abel Gargallo Peiró, Francisco Javier Roca Navarro, Jaume Peraire Guitart, and Josep Sarrate Ramos. 2013. High-order mesh generation on CAD geometries. In *Adaptive Modeling and Simulation 2013*. Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE), 301–312.

John Alan George. 1971. *Computer Implementation of the Finite Element Method.* Ph.D. Dissertation. Stanford, CA, USA. AAI7205916.

P.L. George and H. Borouchaki. 2012. Construction of tetrahedral meshes of degree two. *Internat. J. Numer. Methods Engrg.* 90, 9 (2012), 1156–1182. https://doi.org/10.1002/nme.3364 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.3364

Christophe Geuzaine, Amaury Johnen, Jonathan Lambrechts, Jean-François Remacle, and Thomas Toulorge. 2015. *The Generation of Valid Curvilinear Meshes.* Springer International Publishing, Cham, 15–39.

Christophe Geuzaine and Jean-François Remacle. 2009. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Internat. J. Numer. Methods Engrg.* 79, 11 (2009), 1309–1331. https://doi.org/10.1002/nme.2579 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2579

Arash Ghasemi, Lafayette K. Taylor, and James C. Newman, III. 2016. Massively Parallel Curved Spectral/Finite Element Mesh Generation of Industrial CAD Geometries in Two and Three Dimensions. *Fluids Engineering Division Summer Meeting* 50299 (2016). http://dx.doi.org/10.1115/FEDSM2016-7600

Michael T Goodrich, Leonidas J Guibas, John Hershberger, and Paul J Tanenbaum. 1997. Snap rounding line segments efficiently in two and three dimensions. In *Proceedings of the thirteenth annual symposium on Computational geometry*. ACM, 284–293.

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3.

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 37, 4, Article 60 (July 2018), 14 pages. https://doi.org/10.1145/3197517.3201353

Thomas JR Hughes, John A Cottrell, and Yuri Bazilevs. 2005. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Meth. Appl. Mech. Eng.* 194, 39 (2005), 4135–4195.

Amaury Johnen, Jean François Remacle, and Christophe A. Geuzaine. 2013. Geometrical validity of curvilinear finite elements. *J. Comput. Phys.* 233 (2013), 359 – 372. https://doi.org/10.1016/j.jcp.2012.08.051

Pushkar Joshi and Nathan A. Carr. 2008. Repoussé: Automatic Inflation of 2D Artwork. In *Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling (SBM'08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 49–55. https://doi.org/10.2312/SBM/SBM08/049-055

Steve L. Karman, J T. Erwin, Ryan S. Glasby, and Douglas Stefanski. 2016. High-Order Mesh Curving Using WCN Mesh Optimization. In *46th AIAA Fluid Dynamics Conference, AIAA AVIATION Forum.* https://arc.aiaa.org/doi/10.2514/6.2016-3178

Patrick M. Knupp. 2000. Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II, A framework for volume mesh optimization and the condition number of the Jacobian matrix. *Internat. J. Numer. Methods Engrg.* 48, 8 (2000), 1165–1185. https://doi.org/10.1002/(SICI)1097-0207(20000720)48:8<1165::AID-NME940>3.0.CO;2-Y

Yaron Lipman. 2012. Bounded Distortion Mapping Spaces for Triangular Meshes. *ACM Trans. Graph.* 31, 4 (2012).

Qiukai Lu, Mark S. Shephard, Saurabh Tendulkar, and Mark W. Beall. 2013. Parallel Curved Mesh Adaptation for Large Scale High-Order Finite Element Simulations. In *Proceedings of the 21st International Meshing Roundtable*, Xiangmin Jiao and Jean-Christophe Weill (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 419–436.

Xiaojuan Luo, Mark S Shephard, and Jean-Francois Remacle. 2001. The influence of geometric approximation on the accuracy of high order methods. *Rensselaer SCOREC report* 1 (2001).

Xiaojuan Luo, Mark S. Shephard, Jean-François Remacle, Robert M. O'Bara, Mark W. Beall, Barna A. Szabó, and Ricardo Actis. 2002. p-Version Mesh Generation Issues. In *IMR*.

Richard H. MacNeal. 1949. *The solution of partial differential equations by means of electrical networks.* Ph.D. Dissertation. CalTech.

Manish Mandad, David Cohen-Steiner, and Pierre Alliez. 2015. Isotopic Approximation Within a Tolerance Volume. *ACM Trans. Graph.* 34, 4, Article 64 (July 2015), 12 pages. https://doi.org/10.1145/2766950

MATLAB Partial Differential Equation Toolbox 2018. MATLAB Partial Differential Equation Toolbox. The MathWorks, Natick, MA, USA.

Johannes Mezger, Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straśer. 2009. Interactive physically-based shape editing. *Computer Aided Geometric Design* 26, 6 (2009), 680 – 694. https://doi.org/10.1016/j.cagd.2008.09.009 Solid and Physical Modeling 2008.

P. Monk. 1987. A Mixed Finite Element Method for the Biharmonic Equation. *SIAM J. Numer. Anal.* 24, 4 (1987), 737–749.

D. Moxey, D. Ekelschot, Ü. Keskin, S.J. Sherwin, and J. Peiró. 2016. High-order curvilinear meshing using a thermo-elastic analogy. *Computer-Aided Design* 72 (2016), 130 – 139. https://doi.org/10.1016/j.cad.2015.09.007 23rd International Meshing Roundtable Special Issue: Advances in Mesh Generation.

J.Tinsley Oden. 1994. Optimal h-p finite element methods. *Computer Methods in Applied Mechanics and Engineering* 112, 1 (1994), 309 – 331. https://doi.org/10.1016/0045-7825(94)90032-9

Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion Curves: A Vector Representation for Smooth-Shaded Images. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, Vol. 27. http://maverick.inria.fr/Publications/2008/OBWBTS08

Magdalini Panagiotakopoulou, Martin Bergert, Anna Taubenberger, Jochen Guck, Dimos Poulikakos, and Aldo Ferrari. 2016. A Nanoprinted Model of Interstitial Cancer Migration Reveals a Link between Cell Deformability and Proliferation. *ACS Nano* 10, 7 (2016), 6437–6448. https://doi.org/10.1021/acsnano.5b07406 PMID: 27268411.

Joaquim Peiró, Spencer J. Sherwin, and Sergio Giordana. 2008. Automatic reconstruction of a patient-specific high-order surface representation and its application to mesh generation for CFD calculations. *Medical & Biological Engineering & Computing* 46, 11 (01 Nov 2008), 1069–1083. https://doi.org/10.1007/s11517-008-0390-3

J. Peraire, M. Vahdati, K Morgan, and O. C. Zienkiewicz. 1987. Adaptive Remeshing for Compressible Flow Computations. *J. Comput. Phys.* 72, 2 (Oct. 1987), 449–466.

Per-Olof Persson and Jaime Peraire. 2009. Curved Mesh Generation and Mesh Refinement using Lagrangian Solid Mechanics. In *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition.*

Roman Poya, Ruben Sevilla, and Antonio J. Gil. 2016. A unified approach for a posteriori high-order curved mesh generation using solid mechanics. *Computational Mechanics* 58, 3 (01 Sep 2016), 457–490. https://doi.org/10.1007/s00466-016-1302-2

Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable Locally Injective Mappings. *ACM Trans. Graph.* 36, 2, Article 37a (April 2017). https://doi.org/10.1145/2983621

Xevi Roca, Abel Gargallo-Peiró, and Josep Sarrate. 2012. Defining Quality Measures for High-Order Planar Triangles and Curved Mesh Generation. In *Proceedings of the 20th International Meshing Roundtable*, William Roshan Quadros (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 365–383.

Eloi Ruiz-Gironés, Abel Gargallo-Peiró, Josep Sarrate, and Xevi Roca. 2017. An augmented Lagrangian formulation to impose boundary conditions for distortion based mesh moving and curving. *Procedia Engineering* 203 (2017), 362 – 374. https://doi.org/10.1016/j.proeng.2017.09.820 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain.

Eloi Ruiz-Gironés, Xevi Roca, and Jose Sarrate. 2016a. High-order mesh curving by distortion minimization with boundary nodes free to slide on a 3D CAD representation. *Computer-Aided Design* 72 (2016), 52 – 64. https://doi.org/10.1016/j.cad.2015.06.011 23rd International Meshing Roundtable Special Issue: Advances in Mesh Generation.

Eloi Ruiz-Gironés, Josep Sarrate, and Xevi Roca. 2016b. Generation of Curved High-order Meshes with Optimal Quality and Geometric Accuracy. *Procedia Engineering* 163 (2016), 315 – 327. https://doi.org/10.1016/j.proeng.2016.11.108 25th International Meshing Roundtable.

Edward A. Sadek. 1980. A scheme for the automatic generation of triangular finite elements. *Internat. J. Numer. Methods Engrg.* 15, 12 (1980), 1813–1822. https://doi.org/10.1002/nme.1620151206

L Ridgway Scott. 1973. *Finite element techniques for curved boundaries.* Ph.D. Dissertation. Massachusetts Institute of Technology.

Ridgway Scott. 1975. Interpolated boundary conditions in the finite element method. *SIAM J. Numer. Anal.* 12, 3 (1975), 404–427.

T.W. Sederberg and T. Nishita. 1990. Curve intersection using Bézier clipping. *Computer-Aided Design* 22, 9 (1990), 538 – 549. https://doi.org/10.1016/0010-4485(90)90039-F
Ruben Sevilla, Sonia Fernández-Méndez, and Antonio Huerta. 2011. NURBS-Enhanced Finite Element Method (NEFEM). *Arch. Comput. Methods Eng.* 18, 4 (2011), 441–484.
Mark S. Shephard, Joseph E. Flaherty, Kenneth E. Jansen, Xiangrong Li, Xiaojuan Luo, Nicolas Chevaugeon, Jean-François Remacle, Mark W. Beall, and Robert M. O'Bara. 2005. Adaptive mesh generation for curved domains. *Applied Numerical Mathematics* 52, 2 (2005), 251 – 271. https://doi.org/10.1016/j.apnum.2004.08.040 ADAPT '03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation.
SJ Sherwin and J Peiró. 2002. Mesh generation in curvilinear domains using high-order elements. *Internat. J. Numer. Methods Engrg.* 53, 1 (2002), 207–223.
J Shewchuk. 2012. Unstructured Mesh Generation. In *Combinatorial Scientific Computing*.
Jonathan Richard Shewchuk. 1996. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry Towards Geometric Engineering*, Ming C. Lin and Dinesh Manocha (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 203–222.
Jonathan Richard Shewchuk. 1997. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry* 18, 3 (Oct. 1997), 305–363.
Jonathan Richard Shewchuk. 1999. *Lecture Notes on Delaunay Mesh Generation.* Technical Report.
Jonathan Richard Shewchuk. 2002. What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In *Proceedings of the 11th International Meshing Roundtable, IMR 2002, Ithaca, New York, USA, September 15-18, 2002.* 115–126. http://imr.sandia.gov/papers/abstracts/Sh247.html
Mike Stees and Suzanne M. Shontz. 2017. A high-order log barrier-based mesh generation and warping method. *Procedia Engineering* 203 (2017), 180 – 192. https://doi.org/10.1016/j.proeng.2017.09.806 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain.
Rolf Stenberg. 1984. Analysis of Mixed Finite Element Methods for the Stokes Problem: A Unified Approach. *Math. Comp.* 42, 165 (1984), 9–23. http://www.jstor.org/stable/2007557
Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Brian Whited, Maryann Simmons, and Olga Sorkine-Hornung. 2014. Ink-and-ray: Bas-relief Meshes for Adding Global Illumination Effects to Hand-drawn Characters. *ACM Trans. Graph.* 33, 2, Article 16 (April 2014), 15 pages. https://doi.org/10.1145/2591011
Thomas Toulorge, Christophe Geuzaine, Jean-François Remacle, and Jonathan Lambrechts. 2013. Robust untangling of curvilinear meshes. *J. Comput. Phys.* 254 (2013), 8 – 26. https://doi.org/10.1016/j.jcp.2013.07.022
Thomas Toulorge, Jonathan Lambrechts, and Jean-François Remacle. 2016. Optimizing the geometrical accuracy of curvilinear meshes. *J. Comput. Phys.* 310 (2016), 361 – 380. https://doi.org/10.1016/j.jcp.2016.01.023
Ron Wein, Eric Berberich, Efi Fogel, Dan Halperin, Michael Hemmer, Oren Salzman, and Baruch Zukerman. 2018. 2D Arrangements. In *CGAL User and Reference Manual* (4.13 ed.). CGAL Editorial Board. https://doc.cgal.org/4.13/Manual/packages.html#PkgArrangement2Summary
Dong Xue, Leszek Demkowicz, et al. 2005. Control of geometry induced error in hp finite element (FE) simulations. I. Evaluation of FE error for curvilinear geometries. *Int. J. Numer. Anal. Model* 2, 3 (2005), 283–300.
M. A. Yerry and M. S. Shephard. 1983. A Modified Quadtree Approach To Finite Element Mesh Generation. *IEEE Computer Graphics and Applications* 3, 1 (Jan 1983), 39–46. https://doi.org/10.1109/MCG.1983.262997
V.S. Ziel, H. Bériot, O. Atak, and G. Gabard. 2017. Comparison of 2D boundary curving methods with modal shape functions and a piecewise linear target mesh. *Procedia Engineering* 203 (2017), 91 – 101. https://doi.org/10.1016/j.proeng.2017.09.791 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain.
Patrick Zulian, Teseo Schneider, Hormann Kai, and Krause Rolf. 2017. Parametric finite elements with bijective mappings. *BIT Numerical Mathematics* (2017), 1–19.

A CURVED AMIPS

Let $E(x, y)$ be the traditional AMIPS energy which is minimized using Newton method, thus we requires gradient and hessian of E. For the smoothing on the features the minimization becomes univariate:

$$\min_{t\in(0,1)} E(c(t)).$$

We now need gradient and hessian of $E(c(t))$ with respect to t which can be easily obtained by the chain rule:

$$E'(c(t)) = \langle(\nabla E)(c(t)), c'(t)\rangle,$$

and

$$E''(c(t)) = (c'(t))^T H_E((c(t))\, c'(t) + \langle(\nabla E)(c(t)), c''(t)\rangle.$$

Note that since $E(c(t))\colon \mathbb{R} \to \mathbb{R}$ the gradient and hessian are scalars (while for $E(x, y)\colon \mathbb{R}^2 \to \mathbb{R}$ they are tensorial).

B POINT CUBIC BEZIER DISTANCE

The parametric value $t^\star$ of the closest point to p in the curve c can be find as

$$t^\star = \arg\min_{t\in(0,1)} \|c(t) - p\|^2,$$

which, by the first order condition leads to the following equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\|c(t) - p\|^2 = 0.$$

Since $c(t)$ is a polynomial of order 3, the squared norm becomes of order 6, which implies that the first order condition equation is of order 5. Therefore it can be written as

$$\frac{\mathrm{d}}{\mathrm{d}t}\|c(t) - p\|^2 = at^5 + bt^4 + ct^3 + dt^2 + et + f = 0.$$

Fining roots of this polynomial is a challenging task, which we solve by computing the eigenvalues λ_i, $i = 0, \ldots, 4$ of the 5×5 companion matrix

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 & f/a \\ 1 & 0 & 0 & 0 & e/a \\ 0 & 1 & 0 & 0 & d/a \\ 0 & 0 & 1 & 0 & c/a \\ 0 & 0 & 0 & 1 & b/a \end{pmatrix}.$$

For each *real* eigenvalue λ_i we compute $\|c(\lambda_i) - p\|^2$ and select $t^\star$ as the λ_i with smallest distance in the interval 0, 1. If no λ_i are in interval, we know that $t^\star$ will be either zero or one, which can be decided by evaluating the distance from the endpoints. Note that we opted for this solution for simplicity, a more efficient alternative is Bezier clipping [Sederberg and Nishita 1990].

C CUBIC LAGRANGE BASES

The ten Lagrange bases are defined as

$$\ell_0 = -\frac{1}{2}(3y - 1 + 3x)(y - 1 + x)(3y - 2 + 3x) \qquad \ell_1 = \frac{1}{2}x(9x^2 - 9x + 2)$$
$$\ell_2 = \frac{1}{2}y(9y^2 - 9y + 2) \qquad \ell_3 = \frac{9}{2}x(x + y - 1)(3x + 3y - 2)$$
$$\ell_4 = -\frac{9}{2}x(3x^2 + 3xy - 4x - y + 1) \qquad \ell_5 = \frac{9}{2}xy(3x - 1)$$
$$\ell_6 = \frac{9}{2}xy(3y - 1) \qquad \ell_7 = -\frac{9}{2}y(3xy - x + 3y^2 - 4y + 1)$$
$$\ell_8 = \frac{9}{2}y(x + y - 1)(3x + 3y - 2) \qquad \ell_9 = -27xy(x + y - 1)$$

D MATLAB EXPERIMENT SETUP

The Matlab experiment consists of:

- a unit square at the origin
- an ellipse centered at $(0.25, 0.25)$ with semiaxes 0.2 and 0.15
- an ellipse centered at $(0.5, 0.75)$ with semiaxes 0.4 and 0.15
- an ellipse centered at $(0.8, 0.5)$ with semiaxes 0.055 and 0.4.

The four primitives are just added.