
Inserting a Curve into an Existing Two Dimensional Unstructured Mesh

Daniel W. Zaide and Carl F. Ollivier-Gooch

Department of Mechanical Engineering, University of British Columbia,
Vancouver, BC, Canada, V6T 1Z4
{cfd, cfog}@mech.ubc.ca

Abstract. In this work, a new method for inserting a curve as an internal boundary into an existing mesh is developed. The curve insertion is done with minimal adjustment to the original topology while maintaining the original sizing of the mesh. The curve is discretized by initially placing vertices, defining a length scale at every location on the curve based on the local underlying mesh, and equidistributing length scale along the curve between vertices. This results in the final discretization being spaced in a way that is consistent with the initial mesh. The new points are then inserted into the mesh and local refinement is performed, resulting in a final mesh containing a representation of the curve while preserving mesh quality. The advantage of this algorithm over generating a new mesh from scratch is in allowing for the majority of existing simulation data to be preserved, and not have to be interpolated onto the new mesh.

1 Introduction

In this paper, we examine the problem of taking a pre-existing mesh and inserting an internal boundary into its topology, based on an arbitrary curve defined within the domain. Internal boundaries are those within a mesh used to support discontinuous physical behavior, such as in the simulation of liquid-solid interactions, or other multi-material problems. Generally speaking, these are defined a priori to mesh generation and the mesh is generated from scratch with the internal boundary in mind. This work examines the challenge of obtaining a specified internal boundary in a general, pre-existing mesh. This work is motivated by remeshing to match the surface of a newly deposited layer of material in the simulation of the semi-conductor manufacturing process. As we build toward a solution of the three dimensional problem, we first consider the simpler, two dimensional problem for insight and understanding.

Previous work on unstructured mesh generation and adaptation in two dimensions is well developed, with an abundance of algorithms available [1, 2]. These algorithms perform well, generating guaranteed quality meshes of piece-wise linear boundaries. More recently work has been on the extension to domains with curved boundaries [3, 4, 5]. While current algorithms are adept at geometries with curved internal boundaries, we are interested in a different problem. Our goal is to insert a curve into a mesh with only local mesh adjustment, in the region near the curve. This is different than the boundary recovery problem at the beginning of mesh generation, as we begin with a pre-existing mesh. A similar idea is used in the simulation of shockwaves by the shock-fitting community [6, 7], with an ad-hoc approach to locally modifying the mesh and placing vertices along the curve. In our work, this is accomplished by borrowing an idea from the mesh adaptation community to determine the vertex locations on the curve: mesh movement.

The use of mesh movement for mesh adaptation (r -refinement) for improving the solution of PDEs has been around for several decades [8, 9], with the mesh vertices moved based on equidistribution of some quantity of interest, numerical, such as discretization error, or physical, such as density or entropy [10, 11]. The mesh movement problem is posed as a moving mesh partial differential equation (MMPDE), and mesh movement is treated as a stage within the numerical simulation. This principle is adapted to our problem, to properly space out new vertex locations on a curve by equidistributing the length scale function of the mesh computed on the curve. Once new vertex locations are determined and inserted into the mesh, the internal boundary is recovered, and existing techniques for improving meshes are performed to restore the original mesh quality.

2 Algorithm Overview

We begin with two initial inputs, an initial unstructured mesh and a curve described by control points. Here, all initial meshes are produced using a modified version of Shewchuk's 2D algorithm [12, 13], guaranteeing a minimum angle of 25.65 degrees. We interpolate the curve using a standard parametric cubic spline interpolation. The curve is then sampled on the mesh to determine length scales at various points on the curve, creating an initial distribution of vertices to insert in the mesh. We then move these vertices along the curve using the principle of equidistribution, resulting in a vertex distribution with spacing comparable to the underlying mesh.

With the new vertex locations determined, vertices near the original curve are removed, creating an open region in the mesh to insert these new points. For this work, vertices within a half length scale normal to the curve are removed; in practice, this leads to a low number of removed vertices since the length scale is based on the mesh and not the curve itself. The vertices are then safe to insert, and the mesh is reconnected appropriately. Local

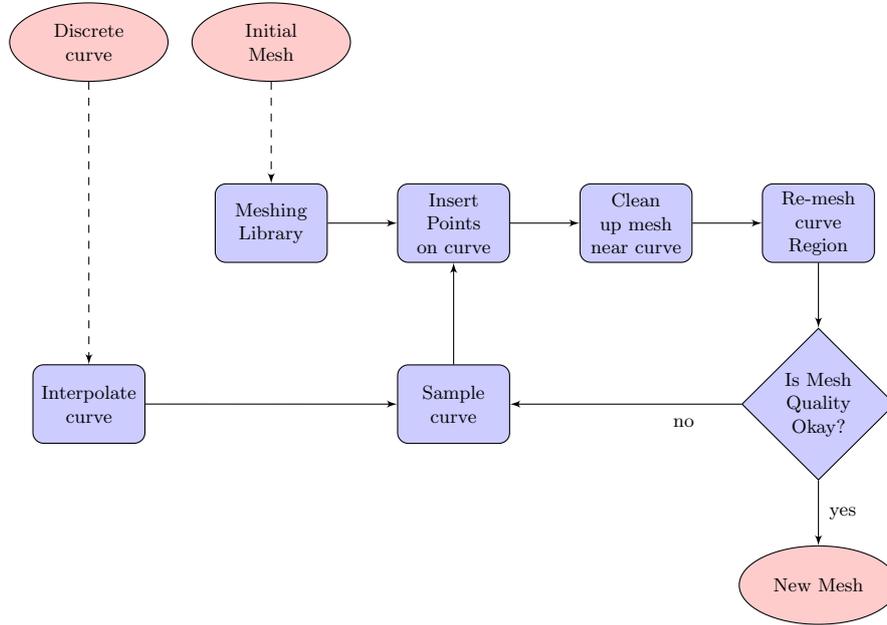


Fig. 1 Algorithm Flowchart

refinement is then used to clean up the mesh. A final check is performed to ensure an appropriate mesh quality is retained. The overall algorithm for curve insertion is shown in Figure 1 and is described in detail in the following sections.

3 Length Scale Equidistribution

We begin by defining the parametric cubic B-spline representation of the curve by parameter t and coordinates $(x(t), y(t))$, with arc length $\ell(t)$ such that $\ell(0) = 0$. A length scale function on the mesh, denoted $LS = LS(x, y)$, is defined as a measure of the distance between neighboring vertices on the mesh.

3.1 Length Scale

The length scale utilized on the curve is first defined from a vertex length scale function as $LS = LS(v)$. While there are many choices for vertex length scale on an existing mesh, in this work, we use the radius of a circle around the vertex based on neighboring vertices, calculated as

$$LS(v) = \left(4 \sum_{i=1}^N \frac{A_i}{N} \bigg/ \sum_{i=1}^N \theta_i \right)^{\frac{1}{2}} \quad (1)$$

where N is the number of neighboring cells, with area A_i and angle θ_i . With this definition, the length scale at any point on the curve can be determined by barycentric interpolation within a cell. Consider a location (x, y) in a cell defined by vertices v_1, v_2 and v_3 . The length scale is then

$$LS(\ell(t)) = LS(x(t), y(t)) = \lambda_1 LS(v_1) + \lambda_2 LS(v_2) + \lambda_3 LS(v_3) \quad (2)$$

for barycentric coordinates $\lambda_1, \lambda_2, \lambda_3$ corresponding to (x, y) . This leads to a continuous piece-wise linear description of the length scale on the curve, which we will use in the discretization of our curve.

3.2 Moving Mesh PDE

Define the computational domain by ξ running from 0 to N , and the length of the curve in this space as $\ell = \ell(\xi)$. The governing equation for equidistribution of length scale along a curve is the moving mesh PDE, written in one dimension as

$$\frac{d}{d\xi} \left(\frac{1}{LS(\ell)} \frac{d\ell}{d\xi} \right) = 0. \quad (3)$$

As we are only interested in getting a ‘good’ spacing and not in solving this exactly, we solve Equation 3 approximately using Gauss-Seidel iteration from an initial spacing of N points, defining discrete arc-lengths $i = 1, \dots, N$ and updating as

$$\ell_i^{n+1} = \frac{LS(\ell_{i-\frac{1}{2}}^{n+1}) \ell_{i+1}^n + LS(\ell_{i+\frac{1}{2}}^n) \ell_{i-1}^{n+1}}{LS(\ell_{i-\frac{1}{2}}^{n+1}) + LS(\ell_{i+\frac{1}{2}}^n)} \quad (4)$$

until the positions do not change.

To determine the initial spacing, a point is placed at one end of the curve and we work our way through along the curve, placing points based on the length scale of the previous point, until we get close to the end. Towards the end of the curve, points are placed until we are within the length scale at the end of the curve, where a final point is placed. This leads to a reasonably good initial spacing on the curve, much better than a uniform or geometric spacing would provide. This also determines N , the number of points. At this stage, we do not add or remove points as the MMPDE is solved, though recent research has explored and developed this idea [14].

3.3 Curve Insertion and Mesh Cleanup

Once the locations of the new vertices on the curve have been determined, the mesh needs to be prepared for their insertion. To ensure the new vertices will be inserted sufficiently far from existing vertices, all vertices within one half of a length scale normal to the curve are removed from the mesh. An example of this measure is shown in Figure 2. After each vertex is removed, the mesh is reconnected, maintaining a valid mesh structure throughout the process, leading to simpler subsequent steps.

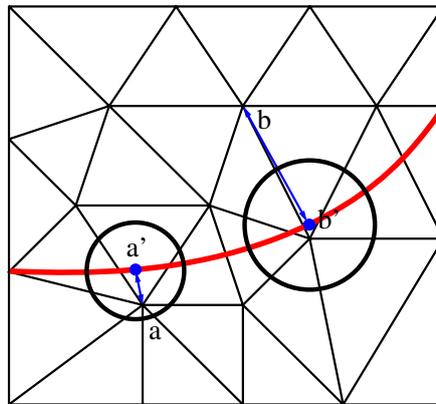


Fig. 2 Identifying vertices to remove. In this representative figure, the distance from the vertex at a to a' is less than half the length scale at a' , indicated by the circle, thus a is chosen to be removed. The vertex at b is further away from the curve than half the length scale at b' , and it remains in the mesh during the removal step.

With the vertices removed, the new vertices can be inserted. We choose to insert the new vertices independently of each other, simply by connecting them to vertices within the cell containing them, splitting the cell into three new cells (or if inserting on a face, two cells into four). Once all new vertices have been inserted, the curve can be recovered by swapping edges until adjacent vertices on the curve are connected by a single edge. During both the insertion and subsequent curve recovery stage, nearby cells to the curve have been created and modified with no consideration of cell quality. We are primarily focused on obtaining an internal representation of the curve inside the mesh. This leads to cells much worse than those in the original mesh, and a cleanup stage is needed to restore mesh quality.

With a wealth of two dimensional techniques for guaranteed mesh quality, we use the refinement techniques described in [15, 3]. These involve inserting vertices at the circumcenters of badly formed cells until the desired quality is met. As only cells near the curve have been modified, refinement can be

done locally. We also intentionally avoid boundary edge splitting to preserve our length scale based spacing. Prior to refinement, the point smoothing of Freitag and Ollivier-Gooch [16] is done on all vertices adjacent to the curve. Smoothing improves the mesh around the newly inserted surface, minimizing the amount of refinement needed. Following smoothing, local refinement is performed if needed, resulting in a final mesh that not only contains the inserted curve, but has maintained a reasonable quality while only a small number of vertices have been modified. While in two dimensions, an arbitrary initial spacing combined with quality-based smoothing could be used to generate a good spacing, this is impractical in three dimensions and we believe our presented algorithm extends better.

4 Numerical Results

In this section, numerical results for several representative examples are shown. In each example, the initial mesh is generated by GRUMMP [13]. While more complex boundary domains could be examined, we are more interested in the performance on the interior of the mesh and the utility of equidistribution of length scale. Gauss-Seidel iteration is considered complete when the total relative movement is less than 1%, $\sum_i |\ell_i^{n+1} - \ell_i^n| / \ell^n < 0.01$. Three examples are shown to verify the effectiveness of the algorithm.

4.1 Example 1 – Uniform Mesh, Cubic Spline

We begin with a uniform isotropic mesh consisting of 104 vertices into which we would like to insert a cubic function, described by the four coordinates $(x, y) = (0.0, 0.5), (0.3, 0.4), (0.7, 0.6), (1.0, 0.5)$, across the mesh, as shown in Figure 3a. This mesh is of good quality, with the minimum angle of any cell in the mesh greater than 34° . The first step is to determine the spacing, by starting from one end of the line and inserting points based on the length scale at the previous point. The initial placement is shown in white and leads to eleven points along the curve, a reasonable number for a mesh of ~ 10 vertices in each direction. Iterating using Equation 4 until the stopping criterion is met leads to the final point locations in black. As we start our initial spacing from the left side, $(0,0.5)$, the locations closer to the start are left relatively unchanged; points placed later in the initialization move further (Figure 3b). Had the initialization been started from the right side, the opposite behavior would be observed. This validates the utility of the moving mesh procedure, improving the spacing of vertices on the curve.

With the spacing determined, vertices near the line are checked and those that are too close are removed, leading to seven removed vertices. The mesh connectivity is retained within this process, resulting in the mesh in Figure 3c. After inserting the new vertices into the mesh and recovering the edge along the straight line (Figures 3d and 3e), we are left with a mesh containing

the initial curve. The quality of this mesh is lacking, with poorly shaped cells created within this process. This is easily resolved by local Delaunay swapping in the near line region, followed by smoothing. For this geometry, after smoothing, no additional refinement was needed to achieve minimum quality, demonstrating the efficacy of the placement of vertices on the curve.

The final result, illustrated in Figure 3f is a mesh that contains the line as a boundary and is of reasonable quality, with a minimum angle of 33.31° . In all, around 11% of the mesh vertices are modified through this algorithm, demonstrating the locality of our algorithm. These results are consistent on finer meshes, shown in Table 1. On the finer meshes, the locality of the algorithm is evident, with a smaller percentage of total vertices affected and a larger percentage of the original mesh unchanged. On the finest mesh, only 0.6% of vertices from the original mesh are modified to insert the curve.

4.2 Example 2 – Graded Mesh, Cubic Spline

With the algorithm demonstrating good performance on uniform meshes, a graded mesh is examined. The initial mesh in this example has 49 vertices, with the majority of them clustered around the right boundary edge. The curve we wish to insert is that of section 4.1, a cubic spline. In Figure 4b, the initial and final spacing is shown. As the mesh is graded, the initial placement of points on the curve is not as close to optimal as on the uniform mesh, and more movement and iterations of the moving mesh equation are needed before an optimal spacing is achieved. As in Example 1, after smoothing, no refinement is needed.

Of possible concern is the poor resolution of the curve in the final mesh (Figure 4f) due to the length scale being independent of the curve itself. This could be easily corrected in either the length scale definition or in the refinement process but we leave this for future investigation. As in the other examples, as the original mesh is refined, the locality of the algorithm is more evident, with less than one percent of the original mesh modified to insert the curve.

4.3 Example 3 – Uniform Mesh, Maple Leaf

In this example, we demonstrate the method on inserting a maple leaf shape into a uniform mesh. The maple leaf is defined by thirty-three lines of varying length, constraining the length scale based on the end points of the shortest lines. In Figure 5, the initial coarse mesh and final mesh containing the shape are shown. Unlike the previous examples, after smoothing, some additional refinement is needed, with 11, 7, and 3 vertices inserted through refinement on the coarse, medium and fine initial meshes respectively. The additional refinement is needed around small angles, where the cell quality from the initial vertex placement is often bad due to the angles in the maple leaf.

Comparing the initial and final meshes, Figures 5a and 5b, reveals that 78% of the initial mesh remains unchanged by the algorithm.

4.4 Example 4 – Bad Meshes, Cubic Spline

In the three examples shown, the initial mesh is of good quality. As a final test of the algorithm, poor quality uniform and graded meshes are generated and used as initial meshes for the cubic spline. These meshes are generated by randomly perturbing the good quality mesh. In Figures 6a and 6c, poor quality meshes generated from the meshes shown in Figures 3a and 4a. The cubic spline is inserted into the mesh, and final meshes are shown in Figures 6b and 6d after local smoothing and refinement. In both examples, the mesh quality is improved in the near curve region, with the remainder of the mesh unmodified. While the mesh is still of overall poor quality, this could be improved using the refinement and smoothing techniques over the whole mesh, as opposed to simply in the vicinity of the curve.

4.5 Summary

Results for each test example are summarized in Table 1. Additional results for medium and fine grid examples (not pictured) are also included. Comparing minimum angles before and after the process demonstrates our algorithm’s effectiveness in preserving mesh quality. For each mesh, the final quality is above 25.65° , the minimum angle guarantee provided by the refinement scheme [15]. In the third example, the mesh quality decreases slightly

Table 1 Summary of Results for Examples 1-4 on meshes of increasing size. In this table, “Remove” is the number of vertices in cleared out to make room for, “Insert”, the number of vertices inserted on the curve. “Smooth” is to the number of vertices moved in the smoothing process. “Min \angle ” is the minimum angle of any cell in the mesh, a measure of mesh quality. Lastly, “Unch. Verts” is the number of vertices left unchanged from the original mesh, a measure of the locality of the algorithm.

Example	Initial Mesh		Curve Recon. # Verts			Final Mesh		
	Verts	Min \angle	Remove	Insert	Smooth	Verts	Min \angle	Unch. Verts
1 - Coarse	104	34.02°	7	11	2	108	33.31°	95
1 - Medium	8315	30.24°	71	112	42	8356	30.01°	8202
1 - Fine	40315	30.24°	157	253	110	40411	30.04°	40048
2 - Coarse	49	30.40°	7	8	7	35	30.18°	35
2 - Medium	8254	30.17°	74	107	51	8287	29.04°	8129
2 - Fine	33952	30.06°	214	316	117	34054	30.06°	33621
3 - Coarse	614	30.01°	66	125	80	697	25.84°	468
3 - Medium	14601	30.07°	356	560	296	14843	26.63°	13949
3 - Fine	57077	30.30°	719	1105	640	57499	26.78°	55770

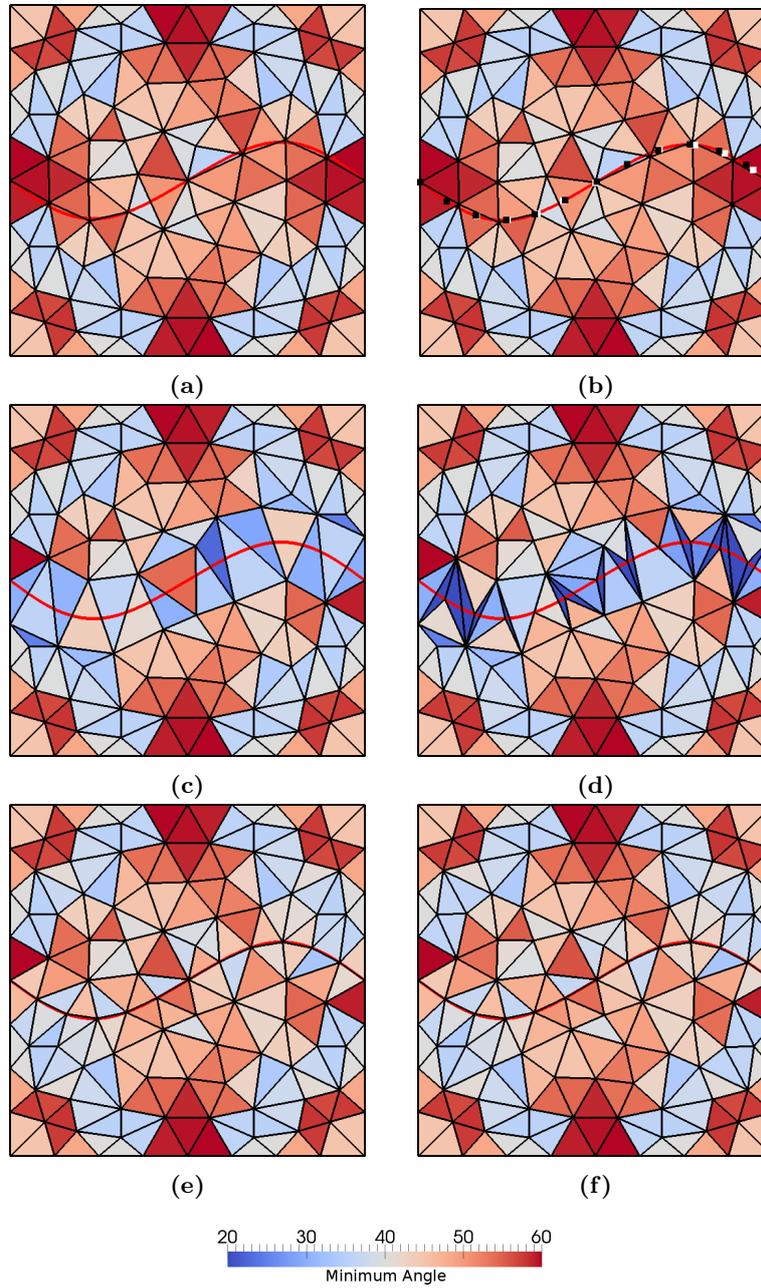


Fig. 3 Example 1 - Coarse Mesh. Each cell is colored by minimum angle. a) Initial mesh, with curve. b) Initial (white) and final (black) points to insert. c) Vertices too close to the curve have been removed. d) Points are now inserted as vertices. e) Boundary edges are recovered. f) Final mesh, after local swapping, refining, and smoothing.

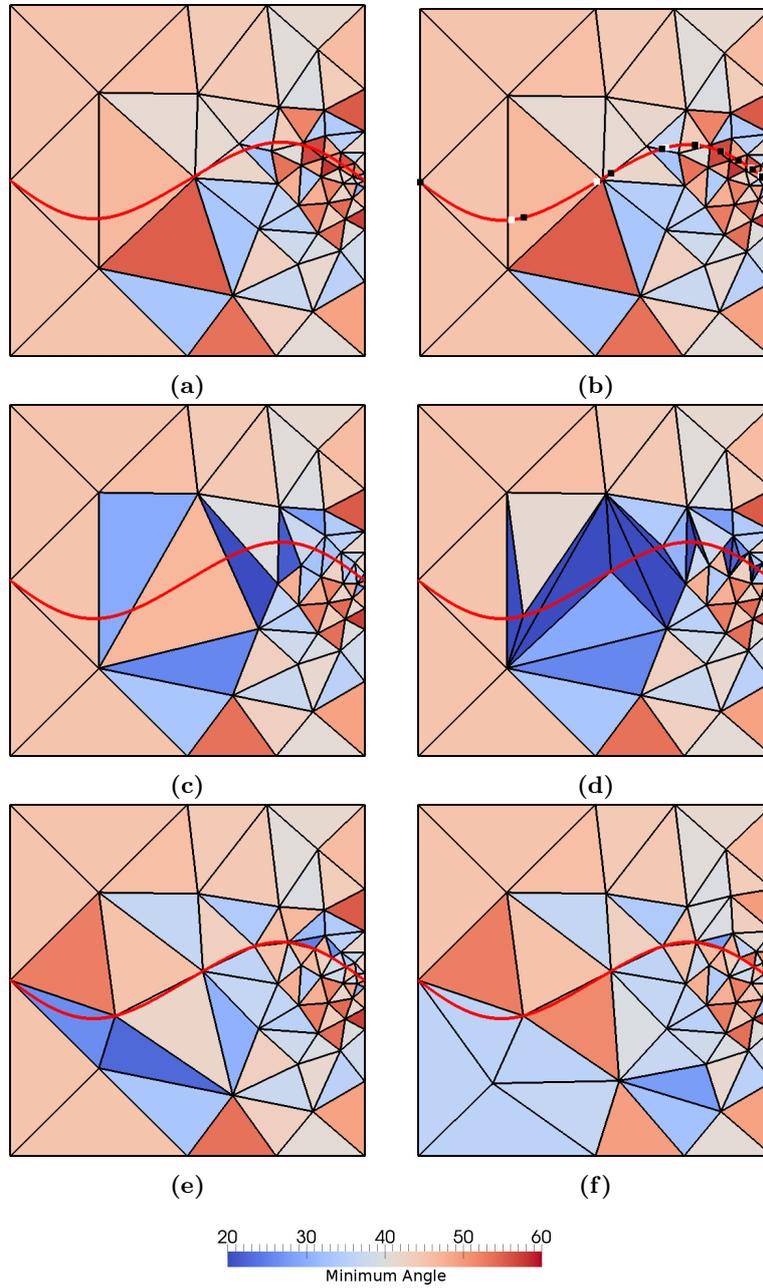
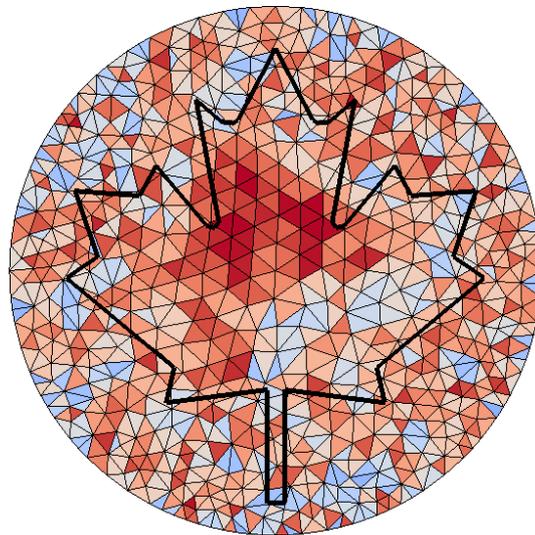
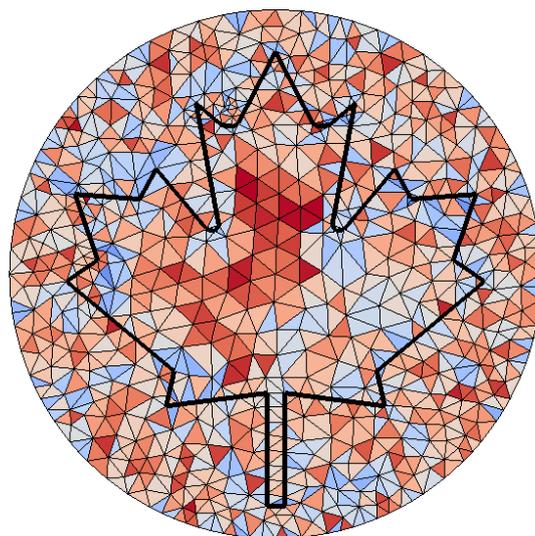


Fig. 4 Example 2 - Coarse Mesh. Each cell is colored by minimum angle. a) Initial mesh, with curve. b) Initial (white) and final (black) points to insert. c) Vertices too close to the curve have been removed. d) Points are now inserted as vertices. e) Boundary edges are recovered. f) Final mesh, after local swapping, refining, and smoothing.



(a) Initial Mesh



(b) Final Mesh

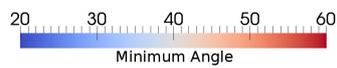


Fig. 5 Example 3 - Maple Leaf - Coarse Mesh. Each cell is colored by minimum angle. a) Initial mesh, with curve. b) Final mesh, after local swapping, refining, and smoothing. The mesh inside the maple leaf has been left unchanged, outside of a layer of vertices around the curve. The mesh inside the stem is also minimal, with a single layer of cells between bounding lines.

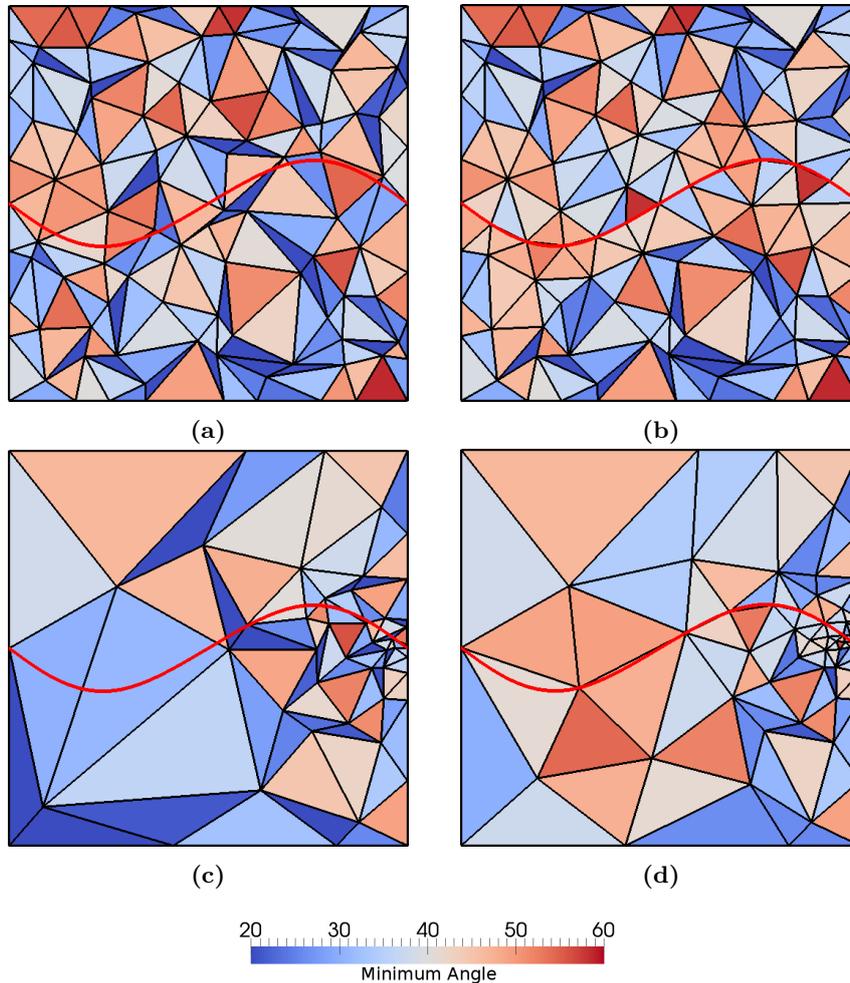
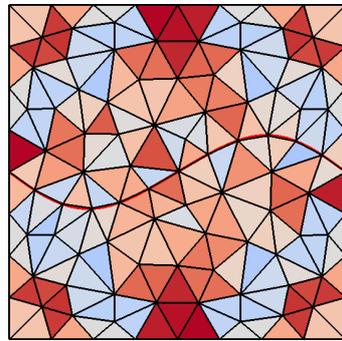


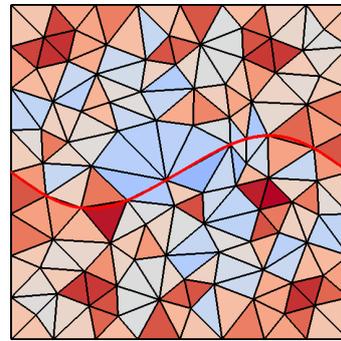
Fig. 6 Example - 4. Two initially poor quality meshes are shown on the left with the canonical cubic spline. Final meshes are shown on the right for the two initial meshes respectively. In both cases, the near curve region shows triangles of good quality, while away from the curve the algorithm leaves the mesh unmodified.

due to the new complex internal geometry, but a reasonable quality is still obtained. It important to observe that as the initial mesh size increases, a smaller percentage of the mesh is modified to insert the curve.

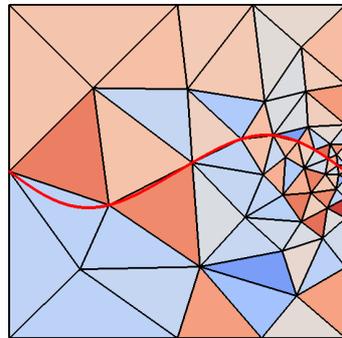
As a comparison for the first three examples, meshes generated from initial geometry including the curve are illustrated in Figure 7. For each pair of meshes, there are approximately the same number of vertices. Though generated in different ways, both meshes are similar in topology and quality. In Figure 7f, a mesh generated with the maple leaf as an internal boundary has additional vertices placed near smaller boundary angles, a result of the



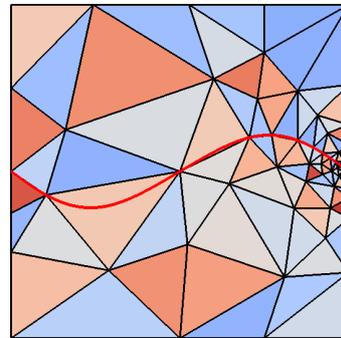
(a) Ex. 1, initial mesh generated, then curve inserted



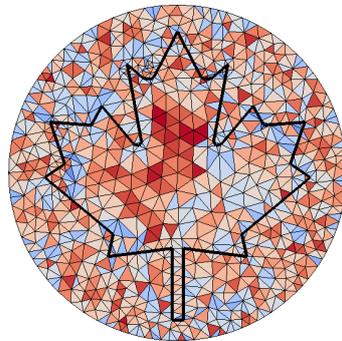
(b) Ex. 1, mesh initially generated with internal boundary



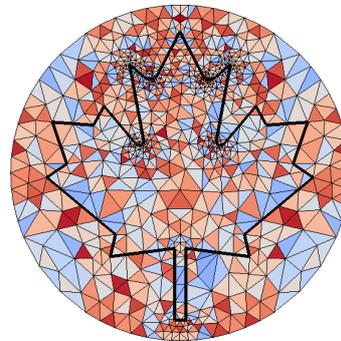
(c) Ex. 2, initial mesh generated, then curve inserted.



(d) Ex. 2, mesh initially generated with internal boundary.



(e) Ex. 3, initial mesh generated, then curve inserted.



(f) Ex. 3, mesh initially generated with internal boundary.

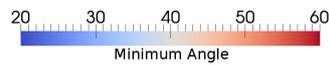


Fig. 7 Comparison between meshes produced by curve insertion into existing mesh (left) and by generation of mesh with the curve as an internal boundary in the initial boundary geometry (right). Comparable meshes for Example 1 (top), Example 2 (middle), and Example 3 (bottom).

initial Delaunay triangulation. For larger meshes, curve insertion is preferable and computationally less expensive than generating a new mesh. Again, the true value of our presented algorithm is in the simulation side, where minimal mesh adjustment allows for less interpolated of simulation data, such as cell-averaged data, onto the new mesh.

5 Conclusions

In this paper we have developed and implemented an algorithm for curve insertion into an existing mesh. This algorithm adopts the principle of equidistribution from the moving mesh community to determine the location of new vertices on the curve. This results in minimal additional work in the addition of these vertices to the mesh. With the new vertices inserted, existing mesh refinement and smoothing techniques are used to produce a final mesh with the curve as an internal boundary and acceptable mesh quality.

In three dimensions, the natural extension of the problem is inserting a surface into an existing tetrahedral mesh. The algorithm remains similar, with a length scale based surface mesh generated and the initial mesh cleaned up and reconnected around it, followed by smoothing and refinement. Mesh movement based on equidistribution for triangular meshes on a surface has been demonstrated effectively [17, 18] and is similar to two-dimensional metric-based mesh adaptation [19]. In our algorithm, the metric would be based on the length scale of the underlying initial mesh.

Acknowledgements. Both authors would like to acknowledge Peter Fleischmann, Stephen Cea, Patrick Keys, and Anil Sehgal of Intel Corporation for their support.

References

1. Weatherill, N.P., Thompson, J.F., Soni, B.K. (eds.): Handbook of Grid Generation. CRC Press (1999)
2. Cheng, S.W., Dey, T.K., Shewchuk, J.R.: Delaunay Mesh Generation. Chapman & Hall (2012)
3. Boivin, C., Ollivier-Gooch, C.F.: Guaranteed-quality triangular mesh generation for domains with curved boundaries 55(10), 1185–1213 (2002)
4. Li, X., Shephard, M., Beall, M.: Accounting for curved domains in mesh adaptation 58, 246–276 (2003)
5. Gosselin, S.: Delaunay Refinement Mesh Generation of Curve-bounded Domains. Ph. D. thesis (2009)
6. Trepanier, J.-Y., Paraschivoiu, M., Reggio, M., Camarero, R.: A conservative shock fitting method on unstructured grids. Journal of Computational Physics 126(2), 421–433 (1996)
7. Paciorri, R., Bonfiglioli, A.: A shock-fitting technique for 2d unstructured grids. Computers & Fluids 38(3), 715–726 (2009)

8. Huang, W., Ren, Y., Russell, R.D.: Moving mesh partial differential equations (mmpdes) based on the equidistribution principle. *SIAM Journal on Numerical Analysis* 31(3), 709–730 (1994)
9. Zegeling, P.: Moving grid techniques. In: *Handbook of Grid Generation*, pp. 37-1–37-18 (1999)
10. Stockie, J.M., Mackenzie, J.A., Russell, R.D.: A moving mesh method for one-dimensional hyperbolic conservation laws. *SIAM Journal on Scientific Computing* 22(5), 1791–1813 (2001)
11. Cao, W., Huang, W., Russell, R.D.: A moving mesh method based on the geometric conservation law. *SIAM Journal on Scientific Computing* 24(1), 118–142 (2002)
12. Shewchuk, J.R.: *Delaunay Refinement Mesh Generation*. Ph. D. thesis, School of Computer Science, Carnegie Mellon University (May 1997)
13. Ollivier-Gooch, C.: *Grump version 0.5.0 user’s guide*. Tech. rep., Department of Mechanical Engineering, The University of British Columbia (2010)
14. Ong, B., Russell, R., Ruuth, S.: An h - r moving mesh method for one-dimensional time-dependent PDEs. In: Jiao, X., Weill, J.-C. (eds.) *Proceedings of the 21st International Meshing Roundtable*, vol. 123, pp. 39–54. Springer, Heidelberg (2013)
15. Ollivier-Gooch, C.F., Boivin, C.: Guaranteed-quality simplicial mesh generation with cell size and grading control 17(3), 269–286 (2001)
16. Freitag, L.A., Ollivier-Gooch, C.F.: Tetrahedral mesh improvement using swapping and smoothing 40(21), 3979–4002 (1997)
17. Cao, W., Huang, W., Russell, R.D.: Anr-adaptive finite element method based upon moving mesh PDEs. *Journal of Computational Physics* 149(2), 221–244 (1999)
18. Crestel, B.: *Moving meshes on general surfaces*. Master’s thesis, Simon Fraser University (2011)
19. Pagnutti, D.: *Anisotropic adaptation: Metrics and meshes*. Master’s thesis (2008)