

A Direct High-order Curvilinear Triangular Mesh Generation Method Using an Advancing Front Technique^{*}

Fariba Mohammadi¹(✉), Shusil Dangi², Suzanne M. Shontz¹, and Cristian A. Linte²

¹ The University of Kansas, Lawrence, KS, 66045, USA
{fariba.m,shontz}@ku.edu

² Rochester Institute of Technology, Rochester, NY 14623, USA
{sxd7257,calbme}@rit.edu

Abstract. In this paper, we propose a novel method of generating high-order curvilinear triangular meshes using an advancing front approach. Our method relies on a direct approach to generate meshes on geometries with curved boundaries. Our advancing front method yields high-quality triangular elements in each iteration which omits the need for post-processing steps. We present several numerical examples of second-order curvilinear triangular meshes of patient-specific anatomical models generated using our technique on boundary meshes obtained from biomedical images.

Keywords: high-order mesh generation · advancing front · curvilinear triangular mesh.

1 Introduction

The use of high-order methods has attracted the interest of the scientific computing community, thanks to their ability to deliver highly-accurate solutions of partial differential equations (PDEs) at a low computational cost. However, while working with curved boundaries, the mesh used with high-order PDE solvers needs to be a high-order mesh that accurately captures the curvature of the geometries [19, 22]. A high-order mesh is composed of both straight-sided and curved elements, depending on the curvature of the geometric domain. One major challenge lies in generating high-order meshes that perfectly capture the curved boundaries; thus, to date, there are not many methods that can generate robust high-order meshes [22].

There are two categories of methods for generating a high-order mesh. The first category consists of direct methods, where a high-order mesh is generated

^{*} The work of the first author was funded by NSF OAC grant 1808553. The work of the second and fourth authors was supported by NIH grant NIGMS R35GM128877 and NSF grant OAC 1808530. The work of the third author was funded in part by NSF grants OAC 1808553 and CCF 1717894.

directly from the curved geometry. To the best of our knowledge, no direct methods are currently available.

The second category includes *a posteriori* methods, which are the most commonly used approaches for generating high-order meshes. Here additional nodes are first added to the low-order mesh, then the newly-added boundary nodes are moved to conform to the curved boundary. Finally, the interior nodes are moved to their new positions [10, 19–21]. These methods deform the linear mesh either using optimization [9, 11, 20, 21] or based on the solution of PDEs [8, 16, 19, 24], e.g., a linear elasticity approach [24], a nonlinear elasticity approach [19], or other strategies [8, 16]. The main challenge associated with this approach is to obtain a valid high-order mesh [22], since the boundary curving step can create tangled elements in the mesh. In this approach, the geometry of the desired high-order mesh is required to represent the curved boundary. This is often obtained from computer-aided design (CAD) files, but in the case of patient-specific anatomical models, such CAD files are not available.

Our proposed method uses a direct approach to generate high-order curvilinear triangular meshes. Our aim is to be able to generate meshes not only from CAD files, but also from other types of boundary representations, such as patient-specific 1D boundary meshes obtained from medical images. Several algorithms for generating unstructured triangular meshes have been developed over the years; among them, the Delaunay triangulation-based methods and the advancing front-based methods are most popular [17]. Here we use an advancing front approach [12–15] to generate high-order curvilinear triangular meshes.

The novelty of our work lies in our method’s ability to generate high-order meshes directly from curved boundaries. This is the first direct approach for high-order mesh generation. Our method does not require a post-processing step, such as mesh untangling, as it generates each element as a valid, high-quality element. Since our method uses a direct approach instead of an *a posteriori* approach, it can generate high-quality meshes on patient-specific models obtained from medical images where no CAD representation is available, serving as a basis for generating meshes for more complex geometries. Hence, our patient-specific meshes can accurately represent complex anatomies, and using these meshes, one will be able to deliver highly-accurate solutions when solving PDEs.

In Section 2, we describe our mesh generation method. Section 3 shows the numerical results of our method on several examples. Finally, in Section 4, we summarize our results and discuss limitations and future directions for this work.

2 High-order Curvilinear Triangular Mesh Generation

In this section, we describe a high-order curvilinear triangular mesh generation algorithm using an advancing front approach. Our method is currently designed to yield second-order curvilinear triangular meshes. In contrast to the traditional high-order mesh generation methods, where post-processing is often a required step, our direct high-order mesh generation method aims to generate high-quality elements in each iteration, so that post-processing is not required.

In the proposed algorithm, we start with a high-order 1D boundary mesh and use it to generate a curvilinear high-order triangular mesh. First, we assign the initial boundary mesh as the initial active front. As the method progresses and new elements are generated, we update the active front by deleting the edges that are already used to generate the triangles and adding the new edges that are created after generating the triangles. Next, we calculate the lengths of the boundary mesh edges. Since the edges of the boundary mesh are curved, we numerically approximate the lengths of the curved edges by dividing them into smaller sections. Then we use shape functions for a one-dimensional second-order Lagrange element to calculate the length of each section. Since the edges to be approximated are quite short, dividing them into smaller sections essentially yields linear segments, therefore rendering this approximation method more accurate and more efficient than calculating the edge's arc length.

To ensure better quality elements from the start, our goal is to generate triangles as close to equilateral triangles as possible. To this end, we first average the lengths of all the curved boundary edges and denote this average length by L_{avg} . We set an upper bound L_{max} on the average length and calculate the upper bound as $L_{max} = b L_{avg}$, where b is a constant. After calculating L_{max} , we use that as the side of an ideal equilateral triangle and calculate the height h of that triangle. This height can be changed by varying the b value in L_{max} . The higher this value, the longer the height of the triangle will be. For our meshes, we use b values ranging from 0.78-0.8.

Once we have calculated h , we start generating the triangular elements from the boundary edges. To this end, we select the first edge from the active front and insert a vertex a at a distance h from the midpoint of the selected edge. To ensure that the vertex is inserted on the correct side of the boundary, we calculate direction normals for the edges and insert the vertex in the direction of the inward normal vector of the boundary mesh. Next, we search for other suitable candidate vertices within a specific radius, $r = \alpha h$, of a . Here, α is a constant that can be varied according to the size of the geometry and mesh. Since a fixed h is used to generate all triangular elements, element size uniformity is ensured.

Figure 1(a) shows the vertex a , height h , and search radius r . The area shown by the gray circle represents the search area for more candidate vertices. Figure 1(b) shows the first curvilinear triangular element generated in the mesh. Figure 1(c) shows a candidate triangle that intersects with an existing triangular element. Note that only the low-order vertices can serve as candidates for the third vertex of the triangle. For each candidate triangle, we perform several validity checks and calculations to ensure we generate the best possible triangle from the candidate vertices available. Moreover, we also perform an intersection test to ensure that we do not have triangular elements that intersect with an existing edge or triangle. We then calculate the scaled Jacobian and equiangular skewness to measure distortion of the curved and straight-sided triangular elements. Considering the boundary edge as the base of our triangular element, we measure the lengths of the two sides of the triangle to make sure one side of

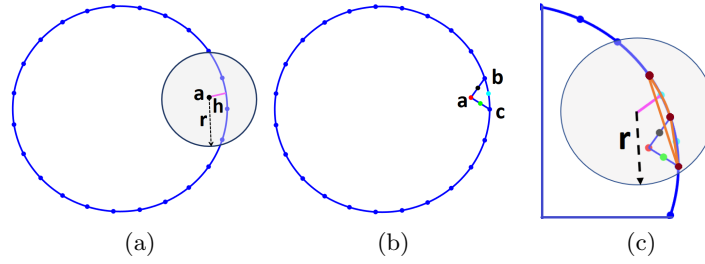


Fig. 1. Boundary mesh of a circle: (a) vertex a is inserted at a distance h , and a search for more candidate vertices is conducted within radius r ; (b) first triangular element bac , and (c) an intersecting candidate triangle shown in orange.

the selected triangle is not very long compared to the other side. These tests are described in Section 2.1 in detail.

If a candidate triangle passes the validity and quality checks, that vertex is inserted (as a new vertex) or selected (as a pre-existing vertex) as the third vertex of the triangle. For every triangle generated, if it has an equiangular skewness value greater than a specific value β , we then perform an edge swap on that triangle to ensure that there are no skinny triangles. Once the low-order vertex is finalized, we generate the high-order vertices for the newly-created triangle edges. For a second-order mesh, the high-order vertices will be the midpoints of each edge. Next, we update the active front by deleting the initially selected edge and adding the newly-created triangle edges. We repeat these steps until our active front is empty. Algorithm 1 gives the pseudocode of our second-order curvilinear triangular mesh generation method.

2.1 Triangle Validity and Quality Checks

Our method searches for suitable candidate vertices and uses several validity and quality checks to select the vertex that will generate the best quality triangle. We conduct the checks in the specified order, so that we can remove the unsuitable candidates one-by-one and preserve the best possible candidates. The unsuitable candidates are those which would generate triangles that intersect with an existing edge or triangle, or have a negative scaled Jacobian. Once we delete all the invalid candidate vertices, we use our triangle selection algorithm described in Algorithm 2 to select the best quality triangle from the remaining candidates. In this section, we summarize these checks.

Intersection Check: Once we identify the candidate vertices from our search, we conduct an intersection check on each triangle generated with those vertices. Our aim is to make sure we do not have candidate triangles that intersect with an existing edge or triangle. We use the `polyshape overlap` function of Matlab R2018a to perform this check. If we find a candidate vertex that, if selected as the third vertex of the triangle, would generate a triangular element that intersects with an existing edge or triangle in the mesh, we discard that candidate.

Algorithm 1: High-order curvilinear triangular mesh generation

Input: Boundary edges as active front
Output: Second-order curvilinear triangular mesh
 Calculate L , L_{avg} , L_{max} , and h
if active front is not empty **then**
 for each edge **do**
 if the edge exists in the active front **then**
 1. Calculate the direction normals
 2. Insert a point a at distance h inside the geometry (Fig. 1)
 3. Search for more candidate vertices within radius r of a (Fig. 1)
 4. Run intersection test
 5. Calculate scaled Jacobian
 6. Calculate equiangular skewness
 7. Considering the selected edge as the base of the triangle,
 calculate the two side lengths of each candidate triangle
 if triangle selection criteria are met as shown in Algorithm 2 **then**
 8. Generate triangle
 if triangle skewness $> \beta$ and edge swap keeps adjacent triangle
 skewness < 0.85 , **then**
 | 9. Perform edge swap
 end
 10. Insert high-order vertices
 11. Update active front
 end
 end
end
end

Scaled Jacobian Calculation: High-order meshes consist of both straight-sided and curved elements depending on the geometry. To measure the distortion or quality of our curvilinear triangular elements, we use the scaled Jacobian quality metric [19]. The scaled Jacobian is defined as:

$$\frac{\min J(\xi)}{\max J(\xi)}, \quad (1)$$

where $J(\xi) = \det(\partial \mathbf{x} / \partial \xi)$. This is the Jacobian of the mapping from the reference coordinate ξ to the physical coordinate \mathbf{x} . Figure 2 shows a second-order triangular element in both physical coordinates and reference coordinates. Scaled Jacobian values can range from $-\infty$ to 1. For a straight-sided element, the scaled Jacobian value is 1. A scaled Jacobian value of 1 does not necessarily indicate a good quality element, since a skinny, straight-sided triangle can also have a scaled Jacobian of 1. A negative scaled Jacobian indicates an inverted element. While the scaled Jacobian is constant for straight-sided elements, for curved high-order elements, a positive near-zero scaled Jacobian value would indicate significant distortion. We calculate the scaled Jacobian using the shape functions for a second-order Lagrange triangle and the high-degree Gaussian quadrature

Algorithm 2: Selection of the best triangular element

Input: Candidates that pass validity checks
Output: The most suitable candidate vertex

```

if scaled Jacobian  $\in (0,1)$  then
  1. Select the candidate that generates the shortest triangle side length
  if selected candidate's skewness  $> \beta$  then
    2. Select the candidate that generates the triangle with highest scaled
      Jacobian
  end
else
  if scaled Jacobian = 1 then
    1. Select the candidate that generates the shortest triangle side length
    if selected candidate's skewness  $> \beta$  then
      2. Select the candidate that generates the triangle with lowest
        skewness
    end
  end
end
3. A candidate vertex is selected
if there is another suitable candidate vertex within a distance,  $l$ , of the
  selected vertex then
  4. Check Delaunay empty circumcircle property
  if the selected vertex is inside the circumcircle of the triangle made with
    the newly found vertex then
    5. Discard the selected vertex and select the other candidate
  end
end

```

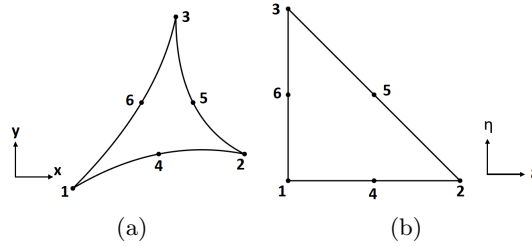


Fig. 2. Second-order triangular element: (a) in physical coordinates; (b) in reference coordinates.

rules developed by Dunavant [6] for triangles. We use a polynomial of degree 8 with 16 Gaussian points and weights. We perform the scaled Jacobian calculation on the updated candidates that we obtain after performing the intersection check. If we obtain a candidate with a negative scaled Jacobian, that candidate is no longer considered.

Equiangular Skewness Calculation: To detect skinny triangles and to measure the distortion of straight-sided elements, we use equiangular skewness. This

angular measure of element quality assesses how close a triangular element is to an equilateral triangle [2]. Since we have curvilinear triangles, we first measure the angles between the tangent lines of the curves using an analysis similar to that described in [22], then use those angles to measure the equiangular skewness, which is given by:

$$\max \left[\frac{\theta_{max} - \theta_e}{180 - \theta_e}, \frac{\theta_e - \theta_{min}}{\theta_e} \right], \quad (2)$$

where

- θ_{min} = smallest angle of the element,
- θ_{max} = largest angle of the element, and
- θ_e = angle for equiangular element, i.e., 60° for equilateral triangles.

For triangular elements, the equiangular skewness should not exceed 0.85.

Triangle Side Length Calculation: We consider the boundary edge selected from the active front as the base of the triangular element and measure the two side lengths of the candidate triangles. To ensure that the triangular elements maintain a uniform size throughout the mesh, we use the relationship $l_1 \leq \gamma l_2$, where l_1 and l_2 are the lengths of the two non-base sides of the triangle, and γ is a constant. The value of γ can be changed according to the geometry and mesh element size. If one side of a candidate triangle is longer than γ times the other side, that triangle is no longer considered.

2.2 Triangle Selection

Once we complete the validity and quality checks, we use the results to select the best triangular element based on the scaled Jacobian, skewness, and triangle side lengths. Since we have both straight-sided and curved elements, we cannot use only the scaled Jacobian to measure the quality of the elements. To avoid skinny triangles, we consider candidate triangles that have a skewness value less than β . The value of β can be changed according to the geometry. If performing an edge swap makes the skewness value of an adjacent triangle greater than 0.85, then we do not perform one. Our method selects the best quality triangle based on the following two cases.

Curvilinear Triangles: First, we select the triangle with the shortest side length that meets the skewness requirement. If that triangle does not meet the requirement, we select the triangle with the maximum scaled Jacobian. If there is another candidate vertex very close to the selected triangle that meets the skewness requirement and also has a scaled Jacobian higher than the previously selected one, we select this other candidate. To find such vertices, we search within a distance, l , of the currently selected vertex. This search distance can be varied according to the size and shape of the elements. We do this to avoid creating skinny triangles in the future.

Straight-sided Triangles: Here, we also first select the triangle that has the shortest side length, provided it meets the skewness requirement. If that triangle does not meet the requirement, we select the triangle with minimum skewness. Here, we cannot rely only on the scaled Jacobian, as for all straight-sided elements the scaled Jacobian will be 1. Again, if there is another candidate vertex very close to the selected triangle that meets the skewness requirement, we select this other candidate vertex instead.

For both cases, we check the edge lengths of the two non-base sides of the triangles to make sure one is not too long or short compared to the other side. We prioritize selecting vertices that already exist in the mesh over inserting new vertices if multiple vertices pass the validity and quality checks and if the vertices are very close to each other. If our selected vertex is a new vertex, then we search within a distance l of that vertex to determine whether there is another suitable candidate vertex that already exists in the mesh. If yes, then we construct the circumcircle of the triangle made with the pre-existing vertex and check to see if the Delaunay circumcircle for that triangle is empty. If the new vertex lies inside the circumcircle, we select the triangle made with the pre-existing vertex. Algorithm 2 gives the pseudocode for the triangle selection process.

3 Numerical Results

In this section, we demonstrate the results from applying our mesh generation algorithm to generate several second-order curvilinear triangular meshes on various patient-specific models. We show how the initial front advances to create the final mesh, as well as how performing edge swaps avoid the generation of skinny triangles in the mesh. We also report the wall-clock time required to generate the meshes. The method was run using Matlab R2018a, and the execution times were measured on a machine with 16GB of RAM and an Intel(R) Core(TM) i7-6700HQ CPU. All mesh visualizations were conducted using Gmsh [10].

For our examples, we use two different types of patient-specific geometries obtained from medical images. Our first set of examples consists of patient-specific cardiac geometries made available through several medical image segmentation challenges - the Left Ventricle Segmentation Challenge (LVSC) [7, 23] available through the Statistical Atlases and Computational Modeling of the Heart and the Automatic Cardiac Diagnostic Challenge (ACDC) [3]. The cardiac image dataset consisted of a stack of 2D image slices and their associated endocardial and epicardial contours at two cardiac phases - diastole and systole - extracted using the distance map regularized convolutional neural network formulation by Dangi et al. in [5]. We use 1D surface meshes of the patient myocardium obtained from magnetic resonance imaging (MRI). We show meshes of the myocardium both at the maximum contraction phase (systole) and maximum expansion phase (diastole) of the heart. Also, for both cases, we show results for a few different MRI slices.

For our second set of examples, we use boundary meshes of the brain ventricles of a patient with hydrocephalus [18] obtained from computed tomogra-

phy (CT) scan images. The final meshes show one pre-treatment and two post-treatment brain ventricles for a hydrocephalus patient who was treated by shunt insertion.

Finally, we use our method to generate triangular meshes of a pair of normal human lungs. The 1D boundary mesh for the lungs are generated from a chest CT scan image [1] using Seg3D [4].

Since our method takes a high-order curved surface mesh as input and the initial patient heart and brain meshes were straight-sided, low-order meshes, we use Gmsh [10] to generate the second-order 1D meshes from the low-order 1D meshes and to refine the meshes if necessary. Next, we use cubic spline interpolation to obtain a curved boundary mesh. We then determine the new positions of the high-order vertices on the curved boundary. For a second-order mesh, these are the midpoints of the newly-curved edges. We use this updated high-order curvilinear boundary mesh as the input for our method. Figure 3(a) shows a straight-sided, low-order mesh; Fig. 3(b) shows a second-order curvilinear coarse boundary mesh, and Fig. 3(c) shows a second-order curvilinear fine boundary mesh of the myocardium.

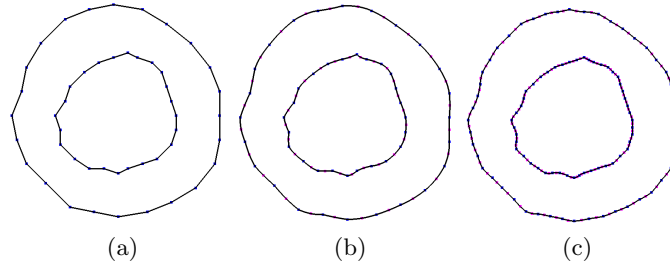


Fig. 3. 1D boundary mesh of patient myocardium: (a) low-order straight-sided coarse mesh; (b) second-order curvilinear coarse mesh; (c) second-order curvilinear fine mesh.

Figure 4 shows an example of how the active front is advancing in a counter-clockwise (CCW) direction to generate the triangular elements. The high-order 1D boundary mesh is used as the active front. As a new triangular element is generated, the active front advances, and this continues until all the vertices are connected in the mesh and the active front is empty. The red arrow in Fig. 4(a) represents the CCW direction in which the initial front is advancing. Figure 4(b) shows the first ring of triangular elements generated in the mesh. Figure 4(c) shows the two opposing directions of the fronts before they are merging.

Depending on the geometry of the input mesh, the active fronts can progress from different directions. When the fronts start to merge, the different-sized edges on the various fronts make it challenging to maintain good quality elements. This can create skinny triangles, like needles and caps. We use edge swaps to avoid creating such elements in our mesh, the success of which is il-

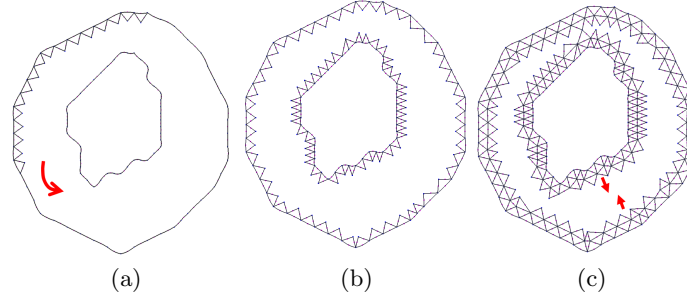


Fig. 4. Advancing front high-order mesh generation: (a) the active front is advancing in a CCW direction to create elements; (b) first ring of triangular elements; (c) the red arrows represent the directions of the two merging fronts, and the method ensures that the merging fronts do not cause element intersections.

illustrated in Figure 5. Figure 5(a,c) show two regions of the mesh before edge swapping is performed. The potential skinny triangles can be observed. Figure 5(b,d) show the same regions after edge swapping is performed to avoid skinny triangle generation.

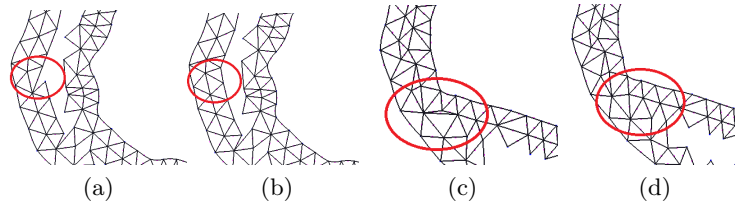


Fig. 5. Edge swapping to avoid skinny triangles: (a) before edge swap; (b) after edge swap; (c) before edge swap; (d) after edge swap.

Figure 6 shows the results of our mesh generation algorithm for various myocardia, at expansion (diastole) and contraction (systole) of the heart from four different MRI images. For diastole, we use a search radius of $1.5h$, $\gamma = 1.5$, and $\beta = 0.5$. For systole, we use a search radius of $1.4h$, $\gamma = 1.2$, and $\beta = 0.5$. Edge swapping is performed if a triangular element has a skewness greater than 0.5. Figure 6(a,b) show the meshes at diastole, and Fig. 6(c,d) show the meshes at systole. The meshes represent the patient-specific geometries accurately, and there are no inverted elements or skinny triangles in the meshes. The runtimes and element quality information are shown in Fig. 6(e).

Figure 7 shows our triangular meshes for the brain ventricles of a hydrocephalus patient before and after the shunt insertion treatment. Before the treatment was performed, the enlarged brain ventricles due to the build-up of cere-

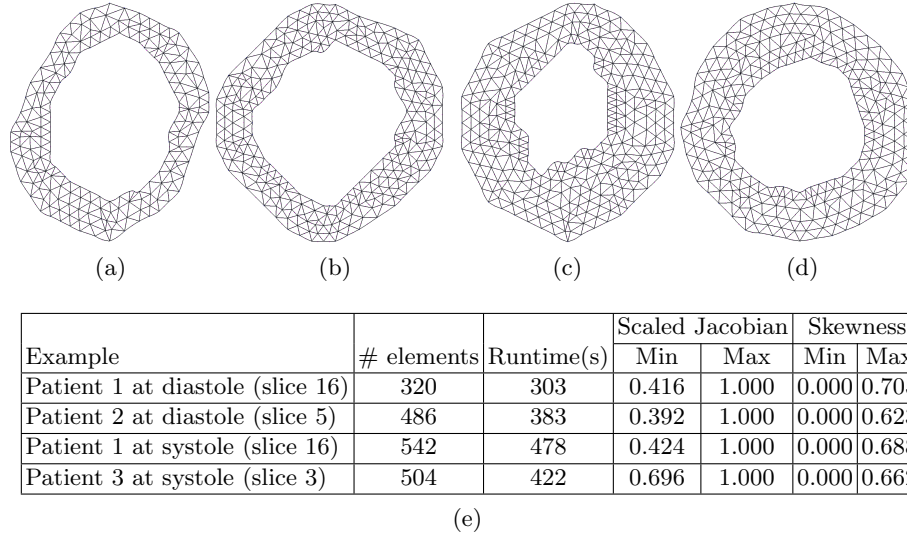


Fig. 6. Second-order triangular meshes of three patients' myocardia at various times in the heartbeat cycle: (a) patient 1 at diastole; (b) patient 2 at diastole; (c) patient 1 at systole; (d) patient 3 at systole; (e) mesh quality metrics and algorithm runtime statistics.

brospinal fluid (CSF) inside the ventricles (i.e., the white area) can be observed in Fig. 7(a). Post-treatment, the condition of the ventricles was observed at two different time points: six month and one year post-treatment. It is observed in Fig. 7(b,c) that the ventricle sizes are gradually reducing post-treatment. For these examples, we use a search radius of $1.5h$, $\gamma = 1.5$, and $\beta = 0.5$ to generate the meshes. There are no inverted elements in these meshes. For the two post-treatment meshes, there are a total of four triangular elements that are close to being skinny triangles, but none have a skewness value greater than 0.85. The runtimes and element quality information are shown in Fig. 7(d). Comparing our pre-treatment mesh with the mesh generated in [18], we observe that the low-order mesh had 8166 elements, whereas our high-order mesh has 1194 elements. This indicates that solving PDEs will require less computational time when employing our meshes. However, since each of our meshes has a different number of elements and different vertex connectivity, solving a PDE for the dynamic problem would require re-interpolation of the solution to go from one mesh to another. Hence, while our meshes will reduce the computational cost and deliver accurate results while solving PDEs on static meshes, their use is not designed for dynamic problems.

Figure 8 shows the triangular meshes of the right and left lungs. For these meshes, we use a search radius of $1.5h$, $\gamma = 1.2$, and $\beta = 0.5$. The lung meshes shown in Fig. 8(a,b) have no inverted elements or skinny triangles, and they accurately capture the patient-specific geometry of the lungs.

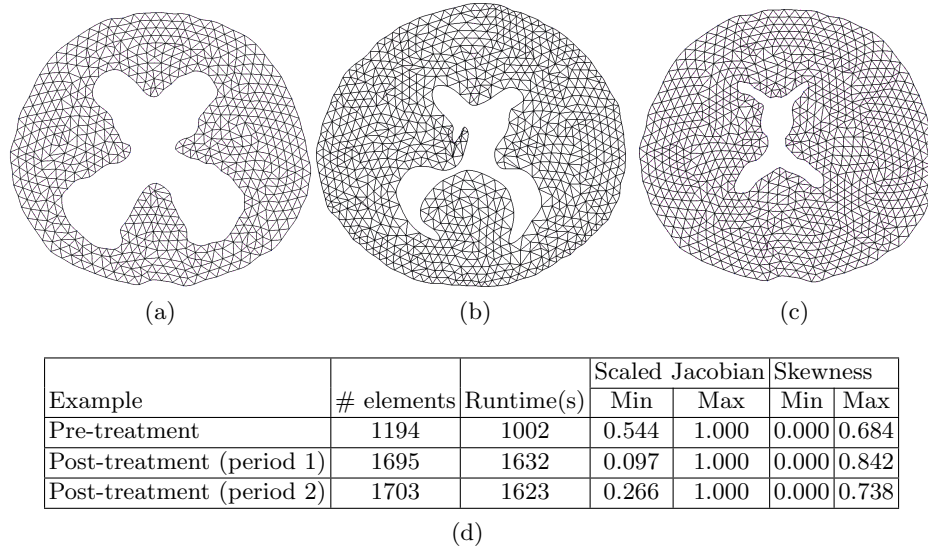


Fig. 7. Second-order triangular mesh of the brain ventricles of a hydrocephalus patient: (a) pre-treatment; (b) post-treatment period 1; (c) post-treatment period 2; (d) mesh quality metrics and algorithm runtime statistics.

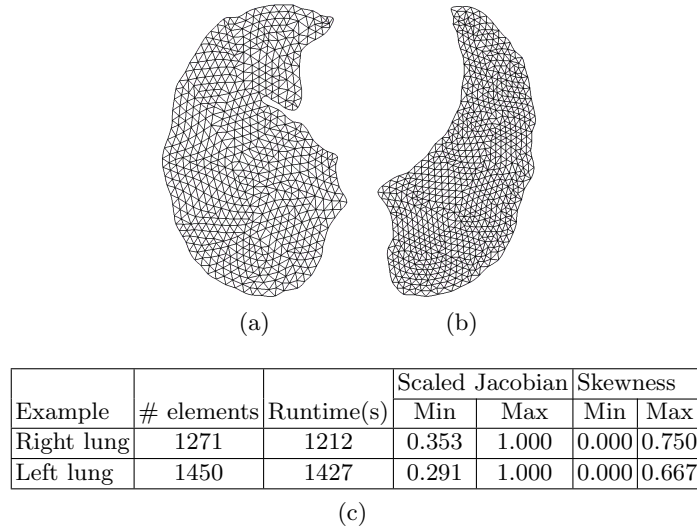


Fig. 8. Second-order triangular mesh of right and left lungs: (a) right lung; (b) left lung; (c) mesh quality metrics and algorithm runtime statistics.

The quality of our meshes can be controlled by changing the search radius r , skewness threshold β , and the γ value, which controls the lengths of the

triangle sides. This versatility is an important feature, since for patient-specific geometries obtained from medical images, the boundary mesh can have different-sized elements. We have observed that for finer meshes, a smaller search radius and a smaller γ value produce good results, whereas for coarser meshes, we need to search within a larger radius to obtain reasonable candidate vertices (and accordingly triangles). There is also the option of changing the height of the triangles by altering the L_{max} value, if necessary.

4 Concluding Remarks

In this paper, we present a new method of generating high-order curvilinear triangular meshes directly from curved geometries. Our method does not require a post-processing step, such as mesh untangling, since we generate each element as a high-quality valid element. Our proposed method can be used to generate curvilinear triangular meshes from patient-specific epicardial and endocardial contours, brain ventricular contours, and lung contours, among others, extracted via segmentation from medical images, such as MRI or CT images. Our mesh generation method can be used as a basis to generate more complex meshes on challenging geometries. To this end, we plan to extend this method to 3D to solve more real-world problems.

We note that, since we implemented our method in Matlab, our implementation has a larger than necessary runtime. Our future work will focus on implementing the method in C++ to reduce the runtime and to be able to work with larger meshes. Also, we only used edge swapping to remove possible skinny triangles from the mesh; an edge collapse operation can also be included to further avoid such triangles.

References

1. Radiologic Images of the Lungs, The Internet Pathology Laboratory for Medical Education, <http://ar.utmb.edu/webpath/radiol/pulmradi/pulm004.htm>, Accessed 14 Apr 2020
2. Skewness Calculation for 2D Elements, <https://www.engmorph.com/skewness-finite-elemnt>, Accessed 14 Apr 2020
3. Bernard, O., Lalande, A., Zotti, C., et al.: Deep learning techniques for automatic MRI cardiac multi-structures segmentation and diagnosis: Is the problem solved? *IEEE Transactions on Medical Imaging* **37**(11), 2514–2525 (2018)
4. CIBC: (2016), Seg3D: Volumetric Image Segmentation and Visualization. Scientific Computing and Imaging Institute (SCI), Download from: <http://www.seg3d.org>
5. Dangi, S., Yaniv, Z., Linte, C.: A distance map regularized CNN for cardiac cine MR image segmentation. *Medical Physics* **46**(12), 5637–5651 (2019)
6. Dunavant, D.: High degree efficient symmetrical Gaussian quadrature rules for the triangle. *International Journal for Numerical Methods in Engineering* **21**(6), 1129–1148 (1985)
7. Fonseca, C.G., Backhaus, M., Bluemke, D.A., et al.: The Cardiac Atlas Project—an imaging database for computational modeling and statistical atlases of the heart. *Bioinformatics* **27**(16), 2288–2295 (2011)

8. Fortunato, M., Persson, P.O.: High-order unstructured curved mesh generation using the Winslow equations. *Journal of Computational Physics* **307**, 1–14 (2016)
9. Gargallo-Peiró, A., Roca, X., Peraire, J., Sarrate, J.: Optimization of a regularized distortion measure to generate curved high-order unstructured tetrahedral meshes. *International Journal for Numerical Methods in Engineering* **103**(5), 342–363 (2015)
10. Geuzaine, C., Remacle, J.F.: Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering* **79**(11), 1309–1331 (2009)
11. Karman, S.L., Erwin, J.T., Glasby, R.S., Stefanski, D.: High-order mesh curving using WCN mesh optimization. In: *Proceedings of the 46th AIAA Fluid Dynamics Conference*. p. 3178 (2016)
12. Lo, S.H.: Volume discretization into tetrahedra—ii. 3D triangulation by advancing front approach. *Computers & Structures* **39**(5), 501–511 (1991)
13. Löhner, R., Parikh, P.: Generation of three-dimensional unstructured grids by the advancing-front method. *International Journal for Numerical Methods in Fluids* **8**(10), 1135–1149 (1988)
14. Mavriplis, D.J.: An advancing front Delaunay triangulation algorithm designed for robustness. *Journal of Computational Physics* **117**(1), 90–101 (1995)
15. Merriam, M.: An efficient advancing front algorithm for Delaunay triangulation. In: *Proceedings of the 29th Aerospace Sciences Meeting*. p. 792 (1991)
16. Moxey, D., Ekelschot, D., Keskin, Ü., Sherwin, S.J., Peiró, J.: High-order curvilinear meshing using a thermo-elastic analogy. *Computer-Aided Design* **72**, 130–139 (2016)
17. Owen, S.J.: A survey of unstructured mesh generation technology. In: *Proceedings of the 7th International Meshing Roundtable*. pp. 239–267 (1998)
18. Park, J., Shontz, S.M., Drapaca, C.S.: A combined level set/mesh warping algorithm for tracking brain and cerebrospinal fluid evolution in hydrocephalic patients. In: *Image-Based Geometric Modeling and Mesh Generation*, pp. 107–141. Springer (2013)
19. Persson, P.O., Peraire, J.: Curved mesh generation and mesh refinement using Lagrangian solid mechanics. In: *Proceedings of the 47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*. p. 949 (2009)
20. Roca, X., Gargallo-Peiró, A., Sarrate, J.: Defining quality measures for high-order planar triangles and curved mesh generation. In: *Proceedings of the 20th International Meshing Roundtable*, pp. 365–383. Springer (2011)
21. Ruiz-Gironés, E., Sarrate, J., Roca, X.: Generation of curved high-order meshes with optimal quality and geometric accuracy. In: *Proceedings of the 25th International Meshing Roundtable*. vol. 163, pp. 315–327. *Procedia Engineering* (2016)
22. Stees, M., Dotzel, M., Shontz, S.M.: Untangling high-order meshes based on signed angles. In: *Proceedings of the 28th International Meshing Roundtable*. pp. 267–282. Zenodo (2020)
23. Suinesiaputra, A., Cowan, B.R., Al-Agamy, A.O., et al.: A collaborative resource to build consensus for automated left ventricular segmentation of cardiac MR images. *Medical Image Analysis* **18**(1), 50–62 (2014)
24. Xie, Z.Q., Sevilla, R., Hassan, O., Morgan, K.: The generation of arbitrary order curved meshes for 3D finite element analysis. *Computational Mechanics* **51**(3), 361–374 (2013)