Quality Meshing of 2D Images with Guarantees Derived by a Computer-Assisted Proof

Jing Xu and Andrey N. Chernikov
Department of Computer Science,
Old Dominion University,
Norfolk, VA, USA,
{jxu,achernik}@cs.odu.edu

Keywords: mesh generation, angle bounds, computer-assisted proof, interval arithmetic

Abstract

This paper describes an algorithm for generating unstructured triangular meshes from a continuous two-dimensional object represented by an image. The algorithm uses squares as a background grid from which to build the quadrilateral elements that conform to the input contours. Then the triangulation is obtained by automatically choosing the diagonals that optimize the angles of the triangles. The extracted triangular meshes can be extensively used in the finite element method (FEM) since our triangulation provides a minimum angle bound of 18.4349°. The angle bound is verified by a computer-assisted proof using interval arithmetic.

1. INTRODUCTION

An important challenge of engineering decision-making is to establish procedures that can represent the physical reality with sufficient accuracy to make predictions. The main procedures are indicated schematically in Fig. 1.

Our work mainly concerns with the procedure of discretization, in other words, mesh generation. There are usually two kinds of meshes used in finite element methods and its applications. The first is surface mesh, which explicitly represents the surface of an object [2]. The second kind of mesh, called volumetric mesh, is distinct from surface mesh in that it explicitly represent both the surface and the volume of the structure [11]. We are developing a three-dimensional tetrahedral mesh generation algorithm, this paper is a preliminary result. We describe our two-dimensional algorithm with full details in this paper.

Finite element methods are now an important and frequently indispensable part of engineering analysis and simulation and modeling. Finite element computer programs are now widely used in practically all branches of engineering for the analysis of fluids, interfaces, and solids. The first step in the finite element computation is to discretize the problem domain into a union of elements, this step is often termed mesh generation. A common choice for an element in two dimensions is the triangle, in three dimensions is the tetrahedron.

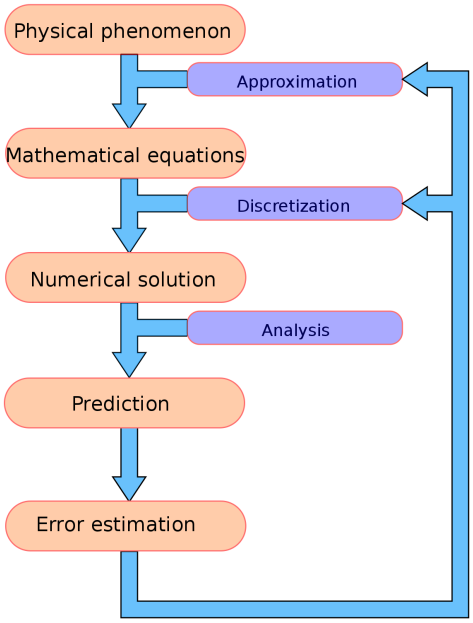

Figure 1. The main procedures of numerical simulation and error estimation.

Thus, a triangulation of the domain is required.

The quality of meshes as well as the number of mesh elements influences the accuracy of the finite element method and condition number of the stiffness matrix. Just a few bad elements can ruin the whole computation. On triangular meshes, the discretization errors usually occur when large angles approach 180°; and both large and small angles are deleterious to the stiffness matrix conditioning [10]. The size of mesh elements influences the computation time and memory storage; the memory requirement and computation time for the numerical process increases drastically when the number of elements increases.

There has been a significant amount of algorithms that theoretically guarantee a quality mesh. In principle, Delaunay triangulation, advancing front, and finite quadtree method (octree for three-dimensional) are all applicable to two or three-dimensional mesh generation.

Mesh generation by Delaunay refinement, whose input is

planar linear complexes, is one of the push-button algorithms used for constructing guaranteed quality triangular and tetrahedral meshes. Quality is traditionally defined in terms of the bounds on circumradius-to-shortest-edge ratio [3]. The use of this measure leads to the improvement of the minimum angle in two dimensions, which helps to improve the conditioning of the stiffness matrix used by a field solver. In three dimensions this measure does not yield such direct benefits.

The advancing front algorithm, taking the input models defined by level set function, generates elements one by one from an initial 'front' formed from the specified boundary of the domain, until the whole domain is completely covered by elements [7,9]. Usually this method accompanies with mesh quality enhancement techniques, such as mesh smoothing and mesh modification.

Quadtree method, whose input is also a level set function, was introduced for domain decomposition to generate non-uniform meshes by Yerry and Shephard [12]. Later, Bern, Eppstein, and Gilbert [1] studied several versions of generating triangular meshes of a planar point set or polygonally bounded domain which guarantee well-shaped elements and small total size simultaneously (Mitchell and Vavasis [8] extended Bern's work to three dimensions). Labelle and Shewchuk [6] adopted the idea of warping and proposed the Isosurface Stuffing tetrahedral meshing algorithm on geometric domains represented by a continuous cut function.

This paper develops a fast triangular mesh generation method. The algorithm generates a triangulation for smooth bounded domain with or without holes. In addition, our method offers two guarantees. First, all the elements in the meshes generated by our algorithm have high quality, meaning that all the angles of all the triangles are bounded between 18.4349° and 143.1302°. Second, the number of triangles is within a constant factor of the best possible for any triangulation with bounded angles. Besides, it is numerically robust and simple to implement. It is applicable in any numerical simulation of the partial differential equations such as fluid flow, mechanical deformation, and diffusion.

The angle bounds were obtained through a computer-assisted proof. These bounds hold for any continuous cut function without sharp edges or corners. Interval arithmetic is used in the proof to get the conservative bounds.

A second version of our algorithm creates meshes whose interior triangles are graded, but on the boundary, the triangles have uniform size. The algorithm relies on a balanced quadtree subdivision that offers interior grading. Since it ensures that the mesh elements are uniform on the objects' boundary, the angle bounds for the uniform meshes also apply to the graded ones.

The remainder of this paper is organized as follows. In Section 2 we give more details on our uniform meshing algorithm. In Section 3 we describe the boundary graded quality mesh generation algorithm. In Section 4 we provide the optimality proof. Section 5 concludes the paper.

2. UNIFORM MESHING ALGORITHM

Our algorithm starts with constructing a initial regular mesh from which to construct an output mesh. Then our algorithm identifies all the edges and points that across the boundaries of objects. During the third step, we deform the initial regular mesh so that a set of initial regular mesh edges respect objects' boundary. And finally we obtain the output mesh by chosing the best triangulation.

2.1. Physical Domain and Background Mesh

We first define a bounded domain $\Omega \subset R^2$, where R is the set of reals. Then we define $f : R^2 \to R$ be a continuous level set function that implicitly represents the geometric shape of the physical domain. The points in point set $\{p : f(p) = 0\}$ are on the boundaries of the domain. Points where f is negative are inside the domain; points where f is positive are outside the domain, and usually should not be meshed. Fig. 2 shows an example of the level set function, an ellipse.

The algorithm employs a space-tiling initial regular mesh to guide the creation of the output mesh. Let $L := Z^2$ be a square lattice, i.e., the set of points whose coordinates are integers. Let s be the lattice spacing, we denote the uniformly scaled square lattice by $L(s)$, where all two dimensions are scaled by $s > 0$.

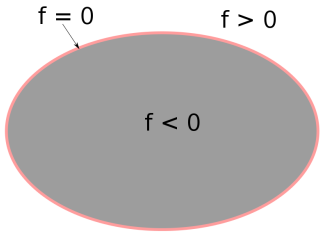

Figure 2. An example of level set function, an ellipse.

2.2. Identify the Cut Edges and Cut Points

For each lattice point of the initial regular mesh, our algorithm computes the sign of the value of the function. For any point inside the level set, the sign is assumed negative, and is assumed positive if the point lies outside the level set. If a lattice point just happen to lie exactly on the level set, the value is zero.

We define a *cutedge* E to be an edge in the initial regular mesh such that it has different signs at two end points. If an edge is a *cutedge*, there exists one point that exactly lies on both the edge and the level set. We define this kind of points a

cut point c such that $c \in E$, and $f(c) = 0$. Then the algorithm identifies all the edges that cross the level set and for each one of those edges a cut point is computed. In Fig. 3, cut edges are shown in red and cut points are shown in blue.

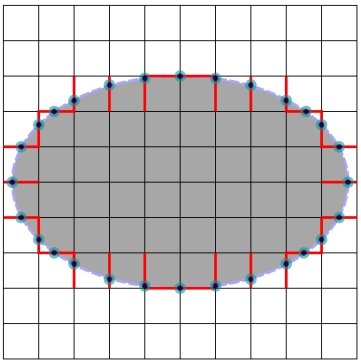

Figure 3. Cut edges and cut points.

2.3. Warping

The idea of deforming the initial regular mesh to conform the objects' boundary is to select a subset of mesh vertices that approximate the level set and to force these mesh vertices snap to the boundary. These vertices are chosen to prevent an interior mesh vertex from being connected to an exterior vertex through a mesh edge. The destination where these vertices be warped is the location of cut points. When one end point is warped to the boundary, the sign of the cut function no longer be positive or negative, it becomes exactly zero.

Let v be an end point of a cut edge E, let $D \in R$ be the distance portion function between v and the cut point c lying on this edge, and let l be the edge length function, then D can be calculated as:

$$D(v,c,E) = \frac{|v-c|}{l(E)} \tag{1}$$

We always choose the one of the two end points to warp whose Euclidean distance is shorter. That means,

$$D(v,c,E) \leq 0.5 \tag{2}$$

If one mesh vertex can be warped to several cut points, we choose any one of them. Fig. 4 shows the initial regular mesh after warping.

2.4. Triangulating the Background Mesh by Optimization

In general, after endpoints of all cut edges are warped to the destination cut points, there will be quadrilaterals and squares remaining in the initial regular mesh. We triangulate these quadrilaterals and squares by selecting different diagonals such that our choice optimizes the minimum angle in the two triangles we obtain. Fig. 5 illustrates the rule for choosing the diagonal.

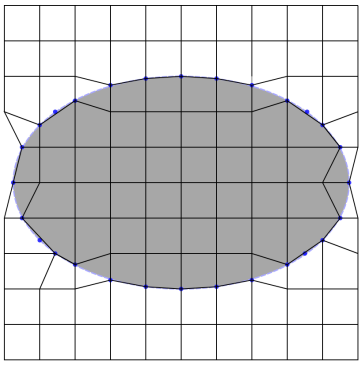

Figure 4. Initial lattice after warping.

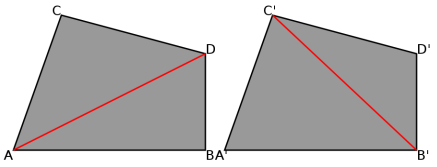

Figure 5. Rule of choice of the diagonal. The quadrilateral $ABCD$ and the quadrilateral $A'B'C'D'$ are the same quadrilateral. The minimum angle in quadrilateral $ABCD$ is $\angle BAD$, and the minimum angle in quadrilateral $A'B'C'D'$ is $\angle B'C'D'$. Because $\angle BAD > \angle B'C'D'$, we chose diagonal AD instead of diagonal $B'C'$. If the minimum angles in two quadrilaterals are same, choose anyone of them.

After choosing the diagonal, a set of triangles are obtained from the initial regular mesh. But only the ones inside the level set should be part of the final mesh. The following condition checks which triangle should be selected.

Let v_0, v_1 and v_2 be the three vertices of a candidate output triangle, let *iso* be the isovalue, the triangle is output ed if and only if

$$(f(v_0) \leq iso) \wedge (f(v_1) \leq iso) \wedge (f(v_2) \leq iso) \tag{3}$$

Fig. 6 shows the final mesh of the ellipse.

2.5. Pseudocode

Fig. 7 summarizes the process:

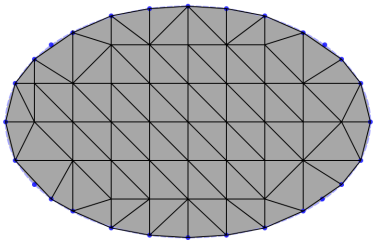

Figure 6. Final mesh.

UNIFORM MESH GENERATION(f, Iso)
Input: f is the level set function
iso is the isovalue specified by user
Output: triangular mesh $\mathcal{M}$ that represents the geometry feature of an object
1: Construct initial regular mesh B which is composed of a set of uniform squares. The size of squares is defined by user
2: Find all cut edges and their associated cut points
3: Warp, i.e., move endpoints of all cut edges to the destination cut points
4: Triangulate the warped initial regular mesh by selecting the best diagonals which optimize the minimum angle in each initial regular mesh elements
5: Output mesh

Figure 7. Pseudocode of uniform Mesh Generation.

3. GRADED MESHING ALGORITHM

Another version of our algorithm creates meshes that have graded elements in the interior but uniform fine elements on the boundary. The reason for this is for many applications, the need of accuracy on the boundary is greatest and most crucial, but the need in the interior does not have the same importance. Thus, the computational time for finite element method can be reduced by reducing the number of elements in the meshes. Fig. 8 illustrates an example of a graded interior mesh.

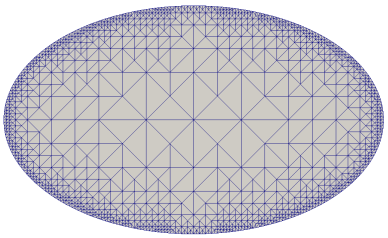

Figure 8. A graded interior mesh.

3.1. Pseudocode

We first present the pseudocode for our graded interior meshing algorithm in Fig. 9:

GRADED INTERIOR MESH GENERATION(f, Iso)
Input: f is the level set function
iso is the isovalue specified by user
Output: Graded mesh $\mathcal{M}$ that represents the geometry feature $\mathcal{M}$ has the fine uniform elements on its boundary
1: Construct initial regular mesh B by quadtree subdivision, the size of minimum leaves is defined by user
2: Balance it by requiring that any two neighboring squares differ at most by a factor of two in size
3: Find all cut edges and their associated cut points
4: Warp, i.e., move endpoints of all cut edges to the destination cut points
5: Triangulate the warped initial regular mesh by two rules. First, if at least one of the leaf side is split by a midpoint, introduce the center of the leaf and connect the center to the midpoint and the endpoints of the shared side; Second, select the best diagonals which optimize the minimum angle in each initial regular mesh elements
6: Output mesh

Figure 9. Pseudocode of graded interior mesh generation.

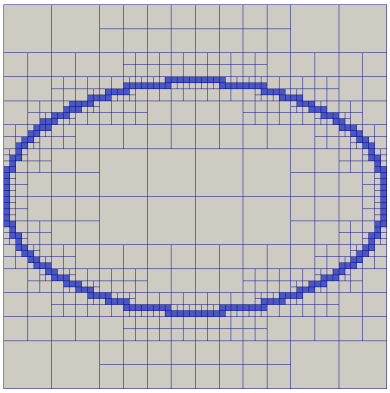

Figure 10. A balanced quadtree with marked leaves of an ellipse.

3.2. Quadtree Subdivision

The main difference between the uniform meshing algorithm and the graded interior meshing algorithm is the first step, i.e., constructing the initial regular mesh, although the implementation for the latter is more complicated. The main data structure we use for the graded meshing algorithm is a quadtree. The quadtree subdivision starts at subdividing a square, we call it a box. We commonly refer to each node of the quadtree as a sub-box. Later sub-boxes are warped and triangulated, changing their geometric structure. An quadtree node is either a leaf, or has four children. The process of generating the four children of a node is called *splitting*. A quadtree subdivision is a recursive function which needs a condition to terminate.

To make sure that small angles never be created in the mesh, after the quadtree subdivision, we balance it by requiring that any two neighboring squares differ at most by a fac-

tor of two in size. Fig. 10 shows the balanced quadtree with marked leaves that cross ellipse's boundary.

3.3. Optimizing the Triangulation

We triangulate quadtree leaves using the following two strategies: first, we say that the side of a sub-box is *split* if either of the neighboring sub-boxes sharing it is split. If at least one of its sides is split by a midpoint, introduce the center of the sub-box and connect the center to the midpoint and the endpoints of the shared side. Second, we select the mesh edge by optimizing the minimum angle in the triangulation as Fig. 11 illustrates. If none of the four sides was split, we select the best diagonal following the same rule in the uniform algorithm.

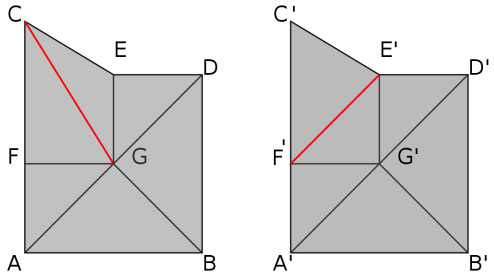

Figure 11. An example of the use of the rule for chosing mesh edges. The north side and the west side are splinted by midpoints. The sub-box $ABCD$ and the sub-box $A'B'C'D'$ are the same box. Vertex C and vertex C' are warped vertices. The triangulation of sub-box $ABCD$ is set $\{\triangle ABG, \triangle BDG, \triangle AFG, \triangle CFG, \triangle CEG, \triangle DEG\}$; and the triangulation of sub-box $A'B'C'D'$ is set $\{\triangle A'B'G', \triangle B'D'G', \triangle A'F'G', \triangle C'F'E', \triangle F'E'G', \triangle D'E'G'\}$. We compare the minimum angle in $\triangle CFG$, $\triangle CEG$ and the minimum angle in$\triangle C'E'F'$, $\triangle E'F'G'$, if the former is larger than the later, we choose edge CG as the remaining mesh edge; or choose edge $E'F'$ otherwise.

4. EXPERIMENTAL RESULTS

We applied the algorithm to a variety of shapes and images. All the steps in the previous two sections were implemented in C++. The input data is a scalar function $f(x,y)$ (i.e., a level set function). All tests were performed on a desktop with two Intel Core Xeon CPU with 3.06 GHz and 64 GB of main memory.

4.1. Geometric Shapes

Fig. 13 shows a breakdown of the total run time of graded interior algorithm applied to a sphere model into the major computational parts, as the diameter of the sphere grows from 100 to 400 pixels. These parts are the computation of a balanced quadtree, warping, and triangulating the initial regular mesh. For the simple shapes, $f(x,y)$ is implemented analytically. Fig. 12 shows a breakdown of the total run time of uniform algorithm applied to a sphere model into the major computational parts, as the diameter of the sphere grows from 100 to 400 pixels. These parts are the computation of the initial regular mesh, warping, and triangulating the initial regular mesh. We exclude the time taken by input and output.

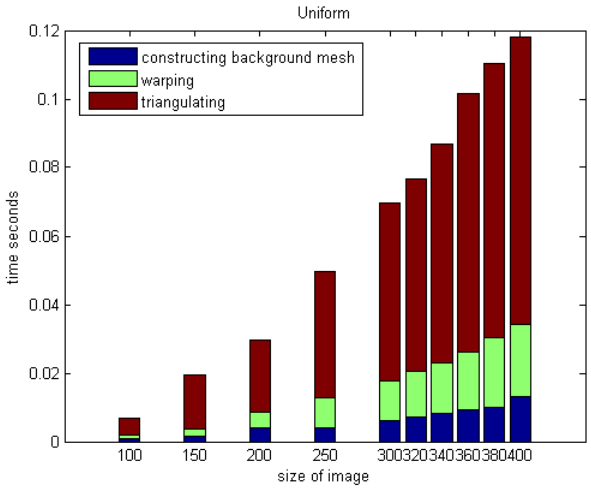

Figure 12. A breakdown of the total run time of uniform algorithm into the major computational parts, as the diameter of the sphere varies from 100 to 400 voxels.

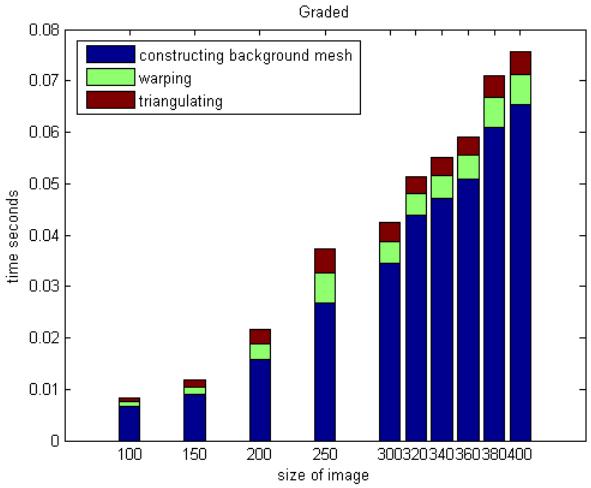

Figure 13. A breakdown of the total time of graded interior algorithm into the major computational parts, as the diameter of the sphere varies from 100 to 400 voxels.

4.2. Slices of Three-dimensional Images

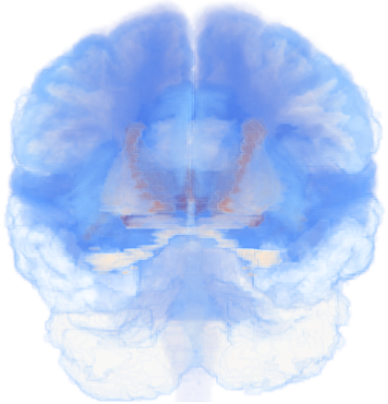

Figure 14. Original three-dimensional image of the brain atlas.

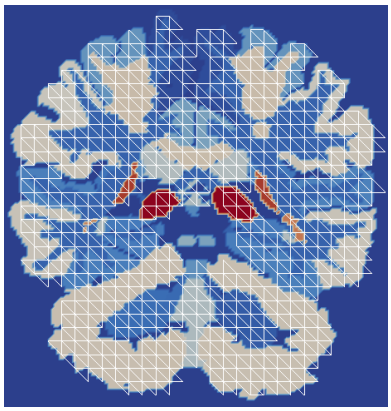

Figure 15. Uniform coarse mesh generated from background lattice with 2500 squares.

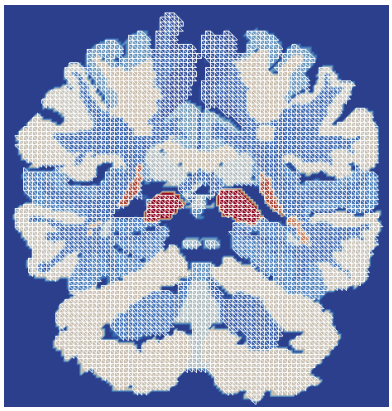

Figure 16. Uniform fine mesh generated from background lattice with 22500 squares.

We used two complex real-world medical images: slices from an abdominal atlas [5] and slices from a brain atlas [4]. The atlases come with a segmentation, such that each voxel is assigned a unique label. These slices are sampled by 256×256 rectilinear grids. The level set function $f(x,y)$ was defined by linear interpolation. The original images and output meshes are shown in Fig. 14 to Fig. 20. The running time and output mesh sizes are given in Table 1.

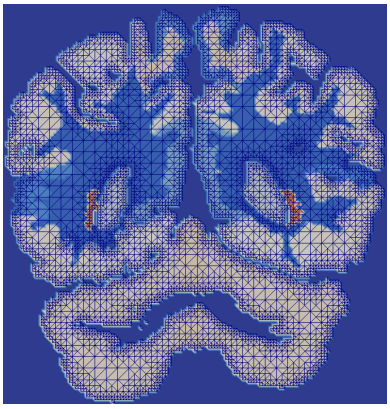

Figure 17. Graded mesh with fine boundary generated from background lattice with balanced quadtree of one slice of brain atlas.

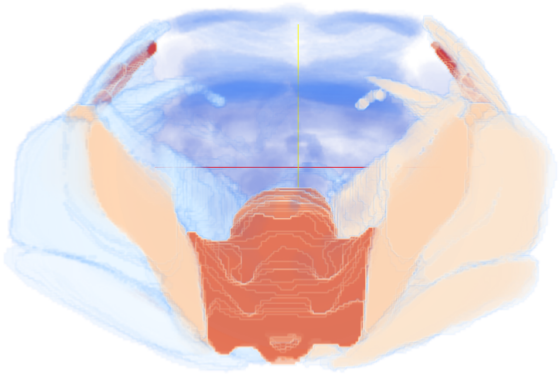

Figure 18. Original three-dimensional image of the abdominal atlas.

Table 1. Run time and size in output meshes

Output Mesh	Run Time	Number of Triangles
Mesh in Fig. 15	2.310391(s)	843
Mesh in Fig. 16	20.82288(s)	8336
Mesh in Fig. 17	46.22675(s)	14489
Mesh in Fig. 19	12.07399(s)	3132
Mesh in Fig. 20	44.50765(s)	12056

5. GUARANTEES FOR THE OUTPUT MESH

Our algorithm offers a guarantee on the output meshes that it never generates triangles with bad angles. Specifically, the

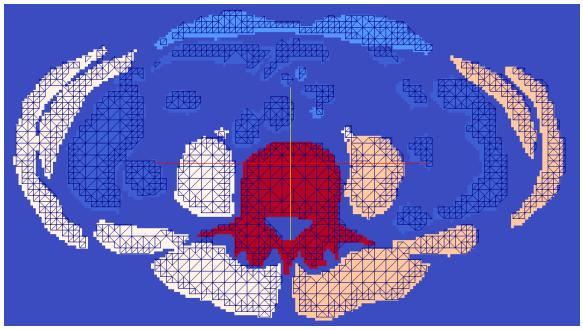

Figure 19. Graded mesh with coarse boundary generated from background lattice with balanced quadtree of one slice of abdominal atlas.

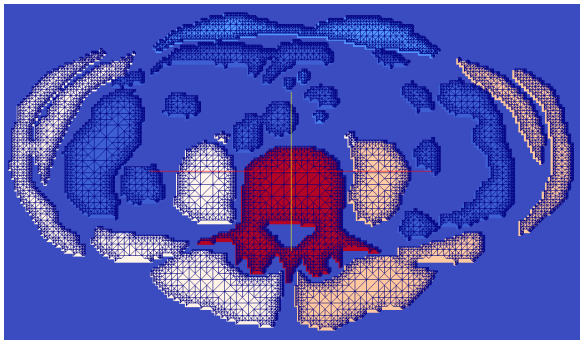

Figure 20. Graded mesh with fine boundary generated from background lattice with balanced quadtree of one slice of abdominal atlas.

angles in output mesh are bounded between 18.4349° and 143.1302°.

The main idea of our meshing algorithm is to ensure that the mesh elements on the objects' boundary are uniformly generated, so the angle bounds apply to our uniform meshes as well as graded ones. Our minimum angle bound was obtained through a computer-assisted proof. By placing a lower bound on the smallest angle of a triangulation, we are also bounding the largest angle, since in two dimensions, if no angle is smaller than θ , then no angle is larger than $180 - 2\theta$.

In our proof we work with a single square which represents any quadrilaterals in the initial regular mesh. We call it a generic square because our proofs are valid for any combination of locations the corners could be warped to. Since the number of locations where a cut point might be placed is infinite, we use interval arithmetic to verify the angle bounds. We divide the intervals of possible triangle configurations into a finite number of subintervals in that interval arithmetic calculates conservative bounds.

As illustrated in Fig. 21, each corner could lie on four segments, along x direction and y direction. We break each of the segment into n intervals, thus each corner could be located in $4n$ intervals. So the square requires the analysis of 4^{4n} cases.

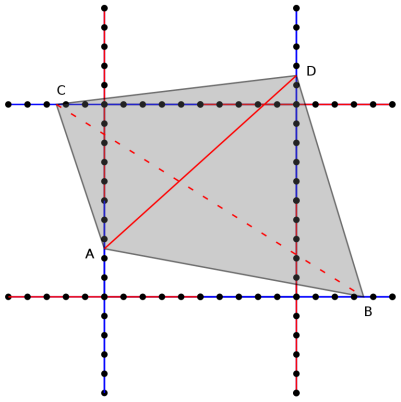

Figure 21. Gray quadrilateral is the generic square warped to position A, B, C and D. Because the minimum angle in $\triangle ABC$ and $\triangle BCD$ is smaller than the minimum angle in $\triangle ACD$ and $\triangle ABD$, we choose diagonal AD instead of diagonal BC.

In each of those cases, we choose the best diagonal optimizing the minimum angle in this square. Our minimum angle bound is obtained among all the minimum angles in those cases.

6. CONCLUSION

In this paper, we presented our new meshing algorithm both for uniform and graded mesh generation. We provide the angle bounds that make the resulting meshes suitable for FE simulation. Moreover, we proved that our mesh elements is optimal, and the meshes we created respect the geometric shapes of the input objects. For the future work, we are planning to extend it to three-dimensional tetrahedral mesh generation and its quality proof.

REFERENCES

[1] M. Bern and J. Gilbert. provably good mesh generation. In *Proceedings of the 31st Annual IEEE Symposium on the Foundations of Computer Science*, pages 231–241, New York, 1990.

[2] Andrey Chernikov and Jing Xu. A computer-assisted proof of correctness of a marching cubes algorithm. In *International Meshing Roundtable*, pages 505–523, Orlando, FL, October 2013. Springer.

[3] L. P. Chew. Guaranteed-Quality Triangular Meshes. Tech. Rep. pages TR–89–983, Department of Computer Science, Cornell University, 1989.

[4] I. Talos M. Jakab R. Kikinis and M. Shenton. Spl-pnl brain atlas, 2008.

[5] I. Talos M. Jakab R. Kikinis and M. Shenton. Spl abdominal atlas, 2010.

[6] F. Labelle and J. R. Shewchuk. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. In *ACM Transactions on Graphics, special issue on Proceedings of SIGGRAPH 2007*, volume 26(3), pages 57.1–57.10, August 2007.

[7] S.H. Lo. *A new mesh generation scheme for arbitrary planar domains.* Int. J. Numer. Meth.Eng., 1985.

[8] S. A. Mitchell and S. A. Vavasis. Quality mesh generation in three dimensions. In *the ACM Computational Geometry Conference*, pages 212–221, 1992.

[9] J Peraire and M Vahdati. *Adaptive remeshing for compressible flow computations.* J. Comp. Phys., 2002.

[10] Jonathan Richard Shewchuk. *What Is a Good Linear Finite Element? Interpolation, Conditioning, Anisotropy, and Quality Measures.* Preprint, 2002.

[11] Jing Xu and Andrey Chernikov. A guaranteed quality boundary graded triangular meshing algorithm backed by a computer-assisted proof. In *International Meshing Roundtable*, Orlando, FL, October 2013. Springer. 5-page research note.

[12] Mark A. Yerry and Mark S. Shephard. A modified quadtree approach to finite element mesh generation. *Computer Graphics and Applications*, 3:39–46, 1983.