Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 124 (2015) 96 – 108

Procedia Engineering

www.elsevier.com/locate/procedia

24th International Meshing Roundtable (IMR24)

Robust All-Quad Meshing of Domains with Connected Regions

Ahmad A. Rushdi[a,b], Scott A. Mitchell[b], Chandrajit L. Bajaj[a], Mohamed S. Ebeida[b,*]

[a] *Institute for Computational Engineering and Sciences, University of Texas, Austin TX 78712, U.S.A*
[b] *Sandia National Laboratories, Albuquerque NM 87185, U.S.A.*

Abstract

In this paper, we present a new algorithm for all-quad meshing of non-convex domains, with connected regions. Our method starts with a strongly balanced quadtree. In contrast to snapping the grid points onto the geometric boundaries, we move points a slight distance away from the common boundaries. Then we intersect the moved grid with the geometry. This allows us to avoid creating any flat quads, and we are able to handle two-sided regions and more complex topologies than prior methods. The algorithm is provably correct, robust, and cleanup-free; meaning we have angle and edge length bounds, without the use of any pillowing, swapping, or smoothing. Thus, our simple algorithm is also more predictable than prior art.

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24)

Keywords: All-Quadrilateral Meshing; Guaranteed Quality; Sharp Features

1. Introduction

Generating good quality meshes for point clouds, planar, or curved surfaces is a challenging problem for several engineering problems, e.g., Finite Element Analysis (FEA) [1–3] and Computer-Aided Design (CAD) [4,5]. Triangular meshes are robustly well-developed with good guaranteed qualities, but quadrilateral ones are not. Given a domain $\mathcal{B}$ of multiple regions, defined by a set of points $\{b_1, b_2, \ldots, b_N\}$ and connectivity edges, an all quad mesh $\mathcal{M}$ is one whose elements are all quadrilateral, accurately representing the domain $\mathcal{B}$. The quality of the mesh plays a significant role in the accuracy and stability of the numerical computation or engineering simulation. The use of algebraic mesh quality metrics is an essential component of automatic generation of unstructured meshes in both 2- and 3-d simulations [6]. It is difficult to determine if a mesh possesses even the minimal quality necessary to undertake a computational analysis without concrete metrics. For the most part, mesh quality metrics are based on geometric criteria. For example, angle minimum and maximum bounds are commonly used geometric quality metrics, along with element size, aspect ratio, skew, stretching, and orientation. Prior methods often have difficulty in achieving angles in $[45°, 135°]$. Indeed, sharp features of the input domain may make this impossible. However, even without sharp input features, many other methods create flat elements that require post-processing to achieve good angles.

* Corresponding author. Tel.: +1-505-845-7594
E-mail address: msebeid@sandia.gov

Peer-review under responsibility of organizing committee of the 24th International Meshing Roundtable (IMR24)
doi:10.1016/j.proeng.2015.10.125

1.1. Related Work

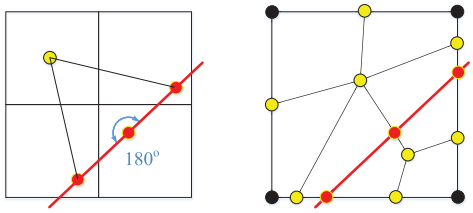
Unstructured all-quad meshing algorithms are usually categorized into two main categories: *indirect* and *direct*. A classical indirect approach starts with a triangular mesh, and then transforms the triangular elements into quadrilateral elements, via optimization [7,8], refinement and coarsening [9], or simplification [10]. A class of indirect methods start with a triangular mesh and apply the mid-point subdivision rule [11,12] to split a triangle into three quad elements. This can be achieved using local subdivision operations. For example, [13] implements a recursive subdivision algorithm based on a regular tiling composed of only diamonds and kites, but does not handle domain boundaries. Q-Morph [14] is a popular indirect approach that follows a sequence of systematic triangle transformations to create an all-quadrilateral mesh. However, Q-Morph requires topological cleanup and smoothing to guarantee the quality of the final all-quad mesh. Q-Tran [15] is another indirect algorithm that produces quadrilaterals with provably-good quality without a smoothing post-processing step, and manages to handle domain boundaries. Nevertheless, the class of indirect methods typically suffers from a large number of irregular nodes that are connected to more (or less) than four mesh elements, which is typically undesired in several numerical simulations. Direct approaches, on the other hand, construct quadrilaterals directly. The advancing front algorithms (e.g., the paving algorithm [16]) successfully generate all quad meshes with high quality, by placing mesh points on the boundaries of the input domain and form quad elements by recursively projecting edges on the front towards the interior of the domain until the whole domain is covered with quads [17,18]. However, they suffer from stability problems that require heuristic cleanup operations. Grid based methods construct a uniform Cartesian or quadtree background grid and aim at modifying that grid to conform to the domain boundaries [19,20]. These methods are easy to implement and can provide quality guarantees and angle bounds [21–23]. However, they often result in inverted elements. Square packing [24] and circle packing [25] are methods that generate all quads. Nevertheless, they bound the maximum angle to 120°, but does not bound the minimum angle. In [26], the minimum angle of the all quad mesh is bounded.

In this paper, we introduce a new direct grid-based algorithm that produces an all-quad mesh with provably-good quality without post-processing cleanup. Our method is stable, conforms to both the interior and exterior of the meshed domain boundary, and can easily adapt to changes in the input domain boundaries, via simple *local* operations.

1.2. Comparison to Dual Contouring

Dual contouring is a direct all quad meshing technique that has attracted a lot of attention recently [27–29]. It basically starts with a uniform mesh, and through intersection with the boundary, it constructs the dual of the uniform mesh. We highlight a few key comparative advantages for our algorithm over dual contouring:

1. **Element Quality: Angle Bounds.** Dual contouring adds points on the domain's boundary directly, creating a large number of flat or close to flat angles which degrades the quality of the resulting mesh. To mitigate the flat angle impact, dual contouring requires post-processing "pillowing" of the mesh in order to get rid of the flat



(a) Dual contouring. (b) Our algorithm.

Fig. 1: Starting with a uniform grid (black): comparing mesh element quality of our algorithm to that of dual contouring when an element is intersected by a domain boundary (red). Dual contouring generates a dual quad element whose vertices are preferably on the boundary (red) or the center of a non-intersected element (yellow), which often results in an undesired flat angle degrading the mesh quality. Our algorithm modifies the existing grid by adding points at the boundary intersections (red), splitting the intersected element, adding refinement points (yellow) at the middle of all edges and centers of split elements, and refining the split elements into well-shaped all-quad elements with good angles bounds.

angles. On the other hand, our algorithm creates one mesh with points that are far from the boundary before intersecting it. Therefore, it creates quad elements with balanced angles and aspect ratio. Furthermore, it is clean-up free, and avoids pillowing completely. A basic illustrative example is shown in Figure 1 to highlight how our algorithm would not create flat angles, as dual contouring does.

2. **Employing Quadtrees.** A desired mesh feature is to allocate more points around the boundary and less far from it. Our algorithm can achieve that since it starts with a balanced quadtree refinement, positioning more points "around" the domain's boundary. Dual contouring can not do that since it would create non-quad elements.
3. **Local versus Global Operations.** Modifying an existing mesh requires the creation of the "dual" mesh in the case of dual contouring. Our algorithm, on the other hand, can easily adapt to any input domain modification by applying the same steps locally in the neighborhood of the domain modification.

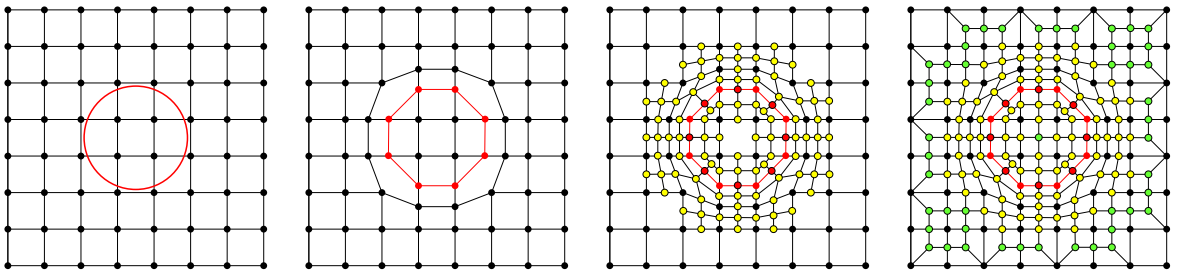
1.3. Terminology

To easily distinguish between the initial grid, the domain geometry, and the final mesh, we use the following terminology. The initial Cartesian or quadtree grid is composed of *squares*, each of which has four *sides* and *corners*. Corners of adjacent smaller squares appear as *hanging nodes*. The side length of a square is denoted s. The geometry is composed of *curves* (which are straight line segments, and two-sided) and *vertices*. The intersection of the quadtree and geometry results in *polygons* and *korner* points connected by *segments*. Polygons are meshed with midpoint subdivision; a *midpoint* subdivides each segment, and a *center* is placed interior to the polygon and connected to each midpoint. The final all-quad mesh is composed of *elements*, *edges*, and *nodes*.

2. Algorithm

Our all-quad meshing algorithm can be described in a rigorous set of repelling, splitting, and refinement steps to achieve the desired mesh properties. These steps are illustrated in Figure 2 for the simple case of a geometric circle and uniform grid. We explain the details of these steps in the following subsections. In summary, we:

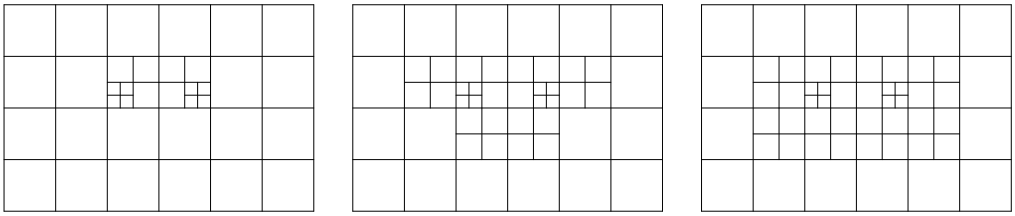
1. Start with any all-quad grid. We choose to start with uniformly-Cartesian grid or a strongly-balanced quadtree.
2. Perturb the quadtree by repelling mesh points *away* from the boundaries of the input domain. All elements intersected by the domain boundary are then split into hybrid elements conforming with the boundary.
3. Apply the mid-point subdivision rule to split and deformed elements to guarantee an all-quad mesh. Some elements of this mesh will possess hanging nodes.
4. Employ the two-refinement templates to get rid of all hanging nodes, generating the final conforming all-quad mesh with good qualities.



(a) Initial uniform grid. (b) Points repelled away, and elements split by boundary. (c) Intersected and deformed elements refined into all quad ones. (d) 2-ref template applied to get rid of any hanging nodes.

Fig. 2: An overview of the all-quad meshing algorithm, applied to a circle shape. Starting with a uniform Cartesian grid, we repel points that are too close to the boundary away from it with a ratio of the initial grid spacing, and split all elements that are intersected by the boundary. Each split or deformed element is then refined into four or more all-quad elements using the mid-point subdivision rule. Finally, we apply the two-refinement templates to guarantee conforming refinement with no hanging nodes.

2.1. Quadtree initialization, refinement, and balancing.



(a) An unblanced quadtree. (b) A balanced quadtree. (c) A strongly-balanced quadtree.

Fig. 3: (a) An unbalanced quadtree of 3 levels. In (b), a balanced tree is one where no edge is shared between two squares with more than a difference of 1 in levels. In (c), a strongly-balanced tree is one where no point is connected to two squares of more than a difference of 1 in levels. If either condition is violated, the larger square gets refined. We require a strongly-balanced quadtree to enable the 2-ref templates in the last step.

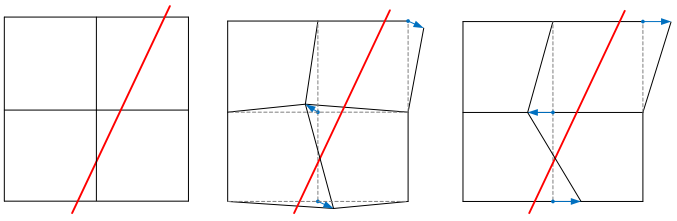
Our algorithm can start with any all-quad mesh. We chose to start with a Cartesian grid and refine it forming a quadtree that captures the fine details of the geometric domain boundaries. Our algorithm requires that the quadtree is strongly balanced, where corner-adjacent squares differ in size by at most one; see Figure 3 for an illustration. To simplify our analysis, we also chose uniform sizing function along the boundaries, as will be further explained in Section 3.

2.2. Perturbing the quadtree

The goal of this step is to prevent small mesh edges. We repel corners of squares *away* from nearby geometry, in coordinate with axis directions or normal to the domain. A raw geometry-quadtree intersection might be arbitrarily close to another point, or at a very small angle. To avoid these problems, we move all corners that are too close to the geometry away from it, horizontally and vertically, or normal to the boundary. In this paper, we present two repelling strategies:

- **Geometry-Normal Repelling.** We identify points that are within a certain distance δ from the boundary, and move them in the normal direction to the boundary until they are exactly δ-apart from the boundary.
- **Grid-Aligned Repelling.** *Horizontal* curves have absolute slope less than one; the others are *vertical*. We move a corner vertically away from a horizontal curve, and vice versa. Corners less than δ vertically (i.e. L_∞ norm) from a horizontal curve are moved vertically until that distance δ is achieved.

An example of the two repelling strategies is shown in Figure 4. We experimentally chose δ to be $\frac{s}{4}$; where s is the size of the square containing the corner.



(a) Grid points close to boundary. (b) Geometry-normal repelling. (c) Grid-aligned repelling.

Fig. 4: Repelling grid points away from boundary if they are too close to it as in (a). In (b), orthogonal repelling moved point in the direction normal to the boundary until the projection distance is exactly h. In (c), the boundary is closer to vertical than horizontal, with a slope larger than one. Therefore, grid aligned repelling moved points horizontally on their initial grid line away from the boundary until the grid line distance between the new point location and the boundary-grid intersection point is exactly h.

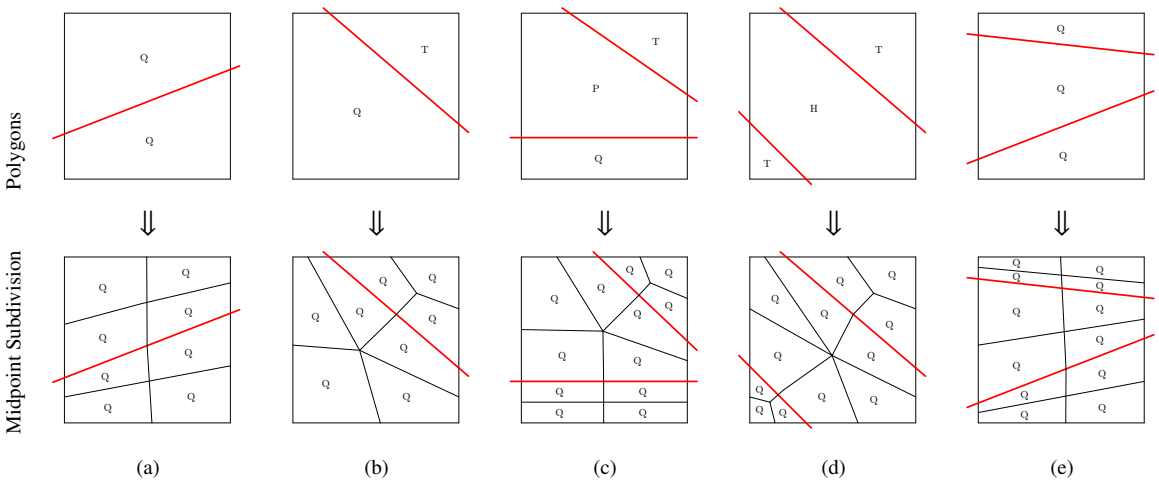


Fig. 5: Example grid-geometry intersection and the resulting midpoint subdivision mesh. Input geometry curves split squares into polygons: Triangles (T), Quads (Q), Pentagons (P), or Hexagons (H). Each polygon is divided into quad elements. The top row shows intersected elements, while the bottom shows the split and divided mesh. Columns show the cases when the boundary splits a quad element into (a) two quads, (b) a triangle and a pentagon, (c) a triangle, a quad, and a pentagon, (d) two triangles and a hexagon, and (e) three quads.

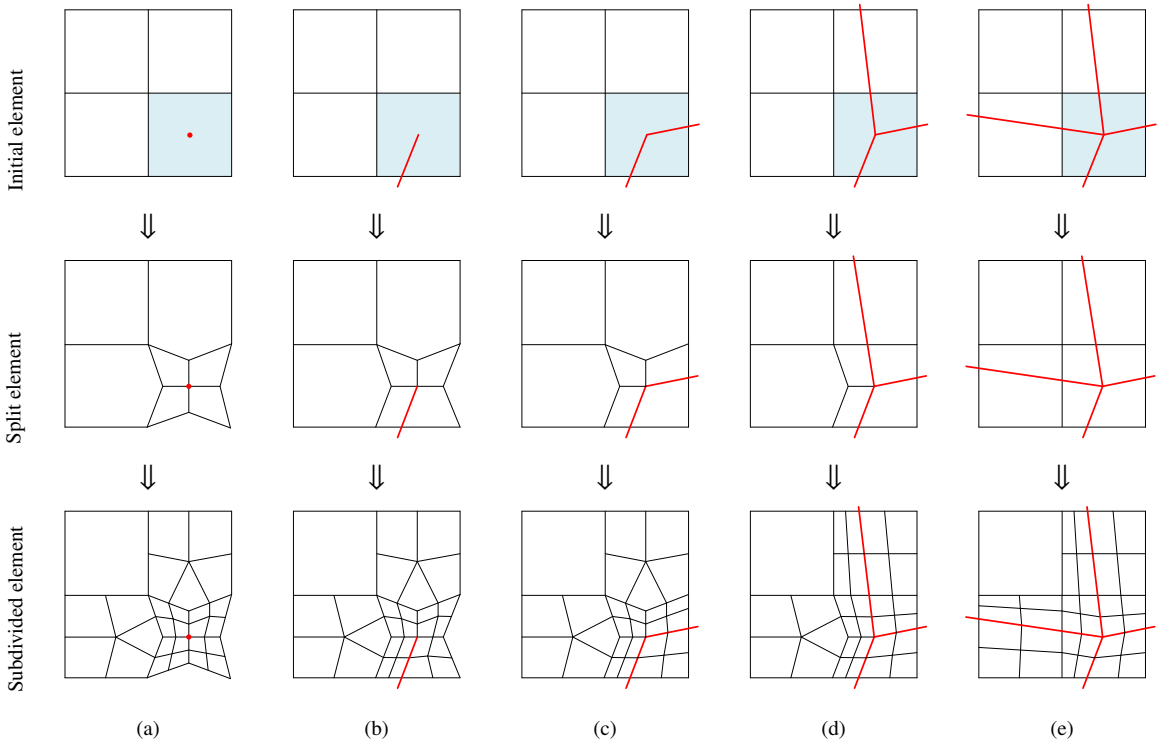


Fig. 6: Refining an element that includes a sharp boundary corner, (a) an isolated point, (b) endpoint of a hanging edge (one boundary intersection), (c) endpoint where two line segments meet (two boundary intersections), (d) endpoint where three line segments meet (three boundary intersections), (e) endpoint where four line segments meet (four boundary intersections). Other intersection scenarios can be decomposed into superpositions of these cases. The element is first split into four quad elements, pulling new edge intersections closer to it to avoid the creation of flat angle. Then, the split elements and the edge-neighbors of the initial element get split into quad elements using the mid-point subdivision rule. This procedure generates neighbor elements with hanging nodes, to be resolved in the last step of the algorithm.

2.3. Polygon Formation

Mesh elements that result from the split caused by intersecting boundaries could generally be triangles, quads, or other polygons. To recover an all-quad status, we refine all elements that were impacted by the domain intersections into smaller quads using the mid-point subdivision rule. Figure 5 shows an illustrative example of elements before and after refinement using basic refinement operations, when a quad element is intersected by one or more boundary segments on opposite or neighbor edges. As shown, the resulting polygons can be quads, pentagons, or hexagons. Each polygon type gets divided into a set of quads using that meet at its mid-point vertex.

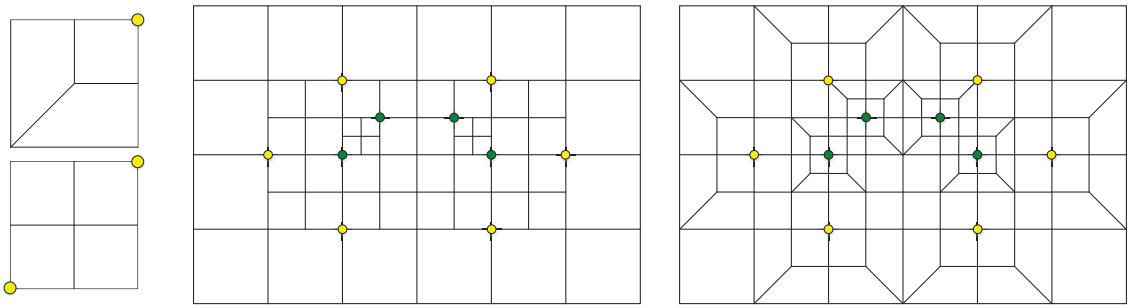
2.3.1. Handling Sharp Corners

On the other hand, to preserve sharp corner and features of the input boundary domain, our algorithm handles these features separately. We insert new mesh points right on top of the domains sharp corner points, and split any element that contains a sharp corner before we apply the mid-point subdivision rule to the element and its neighbors. We require that an element would contain at most one sharp domain corner.

Figure 6 shows an illustrative example of initial, split, and refined elements that include one isolated sharp corner (e.g., a point cloud), or a sharp corner associated with one or more boundary intersections. Elements are first split taking the boundary intersection into consideration as a new element edge. Mid-points that are not on the boundary are pulled closer to the sharp corner to avoid creating flat angles in the neighbor elements. Once the element with a sharp corner is split into four new elements, we further refine it and its neighbors using the mid-point subdivision rule. Hanging nodes will remain after this step is completed, and will be resolved in the last step of the algorithm. Note that while our algorithm is flexible enough to handle sharp corners, we only analyze smooth geometry curves in section 3.

2.4. Handling Hanging Nodes

Squares with geometry produced polygons and were meshed with midpoint subdivision. We now consider the remaining empty squares. Squares with no hanging nodes are immediately mesh quads. Squares with hanging nodes are meshed using the two-refinement (2-ref) templates with corner marking [30]. The two-refinement templates shown in figure 7(a) split an element with a hanging node into either three or four quads. Using a checker board pattern, we mark square corners for template application. In essence, the marking provides a local and globally consistent way to pair adjacent squares with one or three hanging nodes together, allowing all-quad mesh size transitions. For example, the strongly-balanced quadtree with hanging nodes in Figure 3 is marked for 2-ref template application in Figure 7(b) and consequently refined in Figure 7(c).



(a) Templates. (b) A quadtree with hanging nodes. (c) A balanced quadtree with templates applied.

Fig. 7: Applying the two-refinement templates to the strongly balanced quadtree with hanging nodes in Figure 3. The two-refinement templates are shown in (a), and applied to the marked mesh points in (b) to eliminate the hanging nodes as in (c).

3. Analysis

In this section, we analyze the theoretical bounds on the element quality in terms of angle bounds and edge lengths. We denote the maximum angle: ω, the minimum angle: α, the maximum edge length: $e_{\max}$, and the minimum edge length: $e_{\min}$. We take several cases into consideration, proceeding from simple to hard cases, to analytically study the impact of repelling and mid-point subdivision.

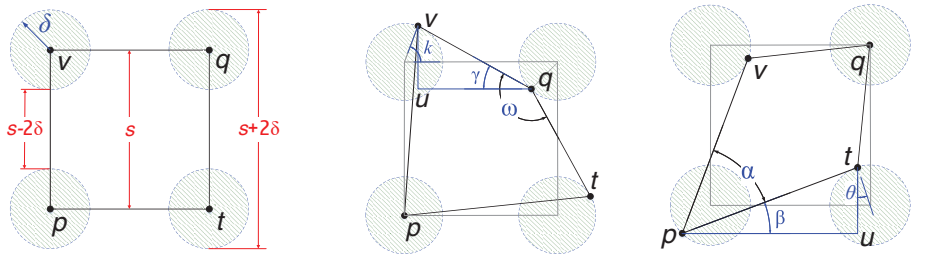
3.1. Squares without geometry or moved corners

Quadtree squares that do not interact with the geometry are meshed into quads using templates, depending on the hanging nodes. It is trivial to see that these results in edge lengths between s and $s/2$ (for a side with a hanging node) and angles between 45° and 135°, as can be seen in Figure 7(a).

3.2. Squares with moved corners, without geometry

Corners of a square with no hanging nodes would be moved in our algorithm using either boundary normal-repelling, or grid-aligned repelling. Each approach results in different angle bounds. Figures 8 and 9 shows the geometry of the largest and the smallest possible angles at a corner when moved within δ in an arbitrary directions, or within δ along the horizontal or vertical grid lines, respectively. The edge length $|e|$ varies in both repelling cases between $s - 2\delta$ and $s + 2\delta$. Next, we derive formulas for the expected minimum and maximum angles, and quantify their values for the experimental choice of $\delta = s/4$, when either repelling approach is used:

- *Boundary-Normal Repelling*. Starting at Figure 8(a), the largest angle would be formed when a corner, say q, moves inwards to the square's center a distance δ in the direction $\vec{qp}$, while its neighbor corners v and t move a distance δ outwards in directions such that qv and qt are orthogonal to the displacements of v and t, respectively. In this case, as shown in Figure 8(b), the horizontal distance between q and v is $|uq| = s - \frac{\delta}{\sqrt{2}} - \delta\cos\kappa$, while the vertical distance $|uv| = \frac{\delta}{\sqrt{2}} + \delta\sin\kappa$. This yields an angle γ between vq and qu, where $\tan\gamma = |uv|/|uq|$. The maximum angle ω at corner q is therefore equal to $90° + 2\gamma$. When $\delta = s/4$, the maximum angle is $\omega = 148.8°$.



(a) Initial square and possible boundary-normal corner movements.

(b) Corner movements resulting in the largest possible angle.

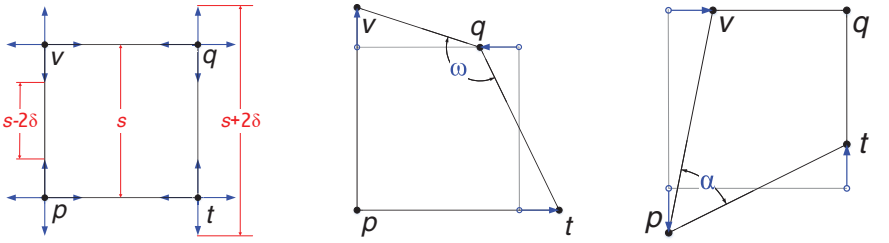
(c) Corner movements resulting in the smallest possible angle.

Fig. 8: Angles and edge lengths for boundary-normal moved corners of a square.

On the other hand, the smallest angle would be formed when a corner, say p, moves outwards away from the square's center a distance δ in the direction $\vec{qp}$, while its neighbor corners t and v move a distance δ towards the square's center in directions such that pt and pv are orthogonal to the displacements of t and v, respectively. In this case, as shown in Figure 8(c), the horizontal distance between p and t is $|up| = s + \frac{\delta}{\sqrt{2}} - \delta\cos\theta$, while the vertical distance $|ut| = \frac{\delta}{\sqrt{2}} + \delta\sin\theta$. This yields an angle β between pu and pt, where $\tan\beta = |ut|/|up|$. The minimum angle α at corner p is therefore equal to $90° - 2\beta$. When $\delta = s/4$, the minimum angle is $\alpha = 48.67°$.

- *Grid-Aligned Repelling*. This repelling approach can be perceived as a special case of the boundary normal repelling that limits the points' movements to the grid lines. Therefore, minimum and maximum angle bounds

here are expected to be tighter than what was achieved using boundary-normal repelling. Similar to the analysis above, we start with an initial square as shown in Figure 9(a). The largest angle would be formed when two opposite corners, say v and t move a distance δ outwards from the square on the vertical and horizontal grid lines, respectively, and the corner q moves a distance δ on either grid line. In this case, as shown in Figure 9(b), the largest angle can be found to be $\omega = 90° + \tan^{-1}(\frac{\delta}{s-\delta}) + \tan^{-1}(2\delta/s)$. When $\delta = s/4$, $\omega = 135°$.



(a) Initial square and possible grid-aligned corner movements.

(b) Corner movements resulting in the largest possible angle.

(c) Corner movements resulting in the smallest possible angle.

Fig. 9: Angles and edge lengths for grid-aligned moved corners of a square.

On the other hand, the smallest angle would be formed when two opposite corners, say v and t move a distance δ inwards, on the horizontal and vertical grid lines, respectively, and corner p moves a distance δ on either grid line. In this case, as shown Figure 9(c), the smallest angle can be found to be $\alpha = 90° - \tan^{-1}(\frac{\delta}{s+\delta}) - \tan^{-1}(2\delta/s)$. When $\delta = s/4$, $\alpha = 52.125°$.

3.3. Squares with geometry

Squares with geometry are divided into 3–6 sided polygons by the curves of the geometry. Some corners may be moved away from the geometry to ensure a δ clearance.

3.3.1. Edge Lengths Bounds

For a square with a vertex, a moved side length is at most $s + 2\delta$, bounded by the case of a square corner–corner edge with both nodes moved away from each other by δ. For a square cut only be curves, since curves are straight, and we consider moving nodes normal to the boundary (which provides upper-bounds for grid-aligned movements as well), two adjacent square corners are moved in the same direction. If a single corner is moved, then we have $|e|^2 < s^2 + \delta^2$. An edge along a curve may not be longer than square diagonal, $|e| < \sqrt{2}s$. Any corner–corner distance is at least δ, and hence corner–midpoint edge is at least $\delta/2$. Any vertex center to midpoint edge is at least δ, by construction. We now consider centers that are not vertices. Any polygon center is at least $\delta/3$ from a non-center edge by the following. The worst case is a triangle. Let one edge be the horizontal axis. The other corner must be at least δ from the edge. Since the center is placed at the geometric center, in particular its vertical coordinate is one-third of each of the corners' vertical coordinate, its height above the edge is at least $\delta/3$. Note for inputs containing a sharp input angle between two curves, the two cut nodes of a polygon may be closer than $\delta/3$ to one another. We do not parameterize our angle bounds for this case explicitly; choosing the actual distance between them as δ, one may carry that value through the following analysis for weaker angle bounds. To summarize, for $\delta = s/4$, we have corner–midpoint curve $|e| \in s[0.125, 0.71]$; and for sides $e \in s[0.125, 1.03]$. Center–midpoint edges are at least $\frac{s}{12}$.

3.3.2. Square corner angles

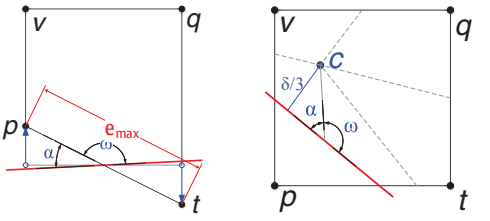
The square corner angles are bounded similar to the uncut squares case, where $\alpha = 48.6°$, and $\omega = 148.8°$.

3.3.3. Cut corner angles

The limiting case for angles at a node where a curve crosses a square side is shown in Figure 10a: each corner has been moved δ away from the curve, and the pre-cut side is at most $e_{\max}$ for sides, $1.03s$. We see that $\sin\alpha = 2\delta/e_{\max}$. Hence for $\delta = \frac{s}{4}$, the minimum angle is $\alpha = 29.04°$ and the maximum angle is $\omega = 150.96°$.

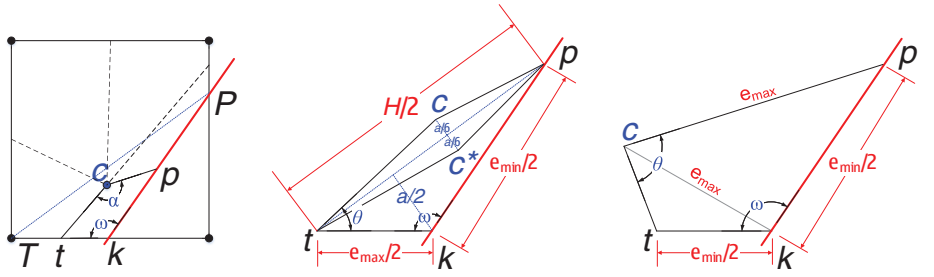
3.3.4. Midpoint angles

We now consider the angle at a midpoint. See Figure 10b. Recall the midpoint was placed along a straight edge (side or curve), and is connected to the center. This is nearly the same as the cut-corner-angle case. Recall that the center to straight-edge distance is at least $\delta/3$, and the midpoint to corner distance is less than $e_{\max}/2$, which is equal to $0.35s$ for curves. When the center to straight-edge distance is small, the center occurs above the corner-corner edge, not to either side. Choosing $\delta = \frac{s}{4}$ results in $\tan\alpha = 2\delta/(3e_{\max})$, at which the minimum angle bound is $\alpha = 13.4°$ and the maximum angle bound is $\omega = 166.6°$.



(a) Crossings. (b) Midpoints.

Fig. 10: Angles and edge lengths for crossings and midpoints.



(a) Large angle α at center point. (b) Large angle close-up. (c) Small angle at center point.

Fig. 11: Angles and edge lengths at center points.

3.3.5. Center angles

We first consider an upper bound. See Figure 11a. Suppose center c is opposite a large angle $\omega = \angle TkP$ with corner k. Let t and p be its connected midpoints. By the same center-is-average-coordinates argument as before, the height (taking $\overline{TP}$ as the horizontal axis) of c is at most one-third of k's height a. The midpoint-midpoint line $\overline{tp}$ is exactly half of k's height, $a/2$. That is, the height of c above $\overline{tp}$ is at least $a/6$. We first relate one of the small angles of $\triangle ctp$ to one of the small angles of $\triangle tkp$. For illustration, reflect c around $\overline{tp}$ as in Figure 11 right. Let c be to the T side of the altitude. Let f be the distance from t to a triangle altitude, and $H/2 = |\overline{tp}|$. Hence we see $\tan\theta = \frac{a}{2f}$ and $\tan\theta_c = \frac{a}{6f}$ so $3\tan\theta_c = \tan\theta$. We now turn our attention to showing that θ itself is large. A bounding case for θ is shown in Figure 11b, where $|tk|$ is large, $e_{\max}/2$; and $|kp|$ is small, $e_{\min}/2$; and ω is large. By the law of cosines, $H^2 = e_{\max}^2 + e_{\min}^2 - 2e_{\max}e_{\min}\cos\omega$. By the law of sines $\sin\theta = e_{\min}\sin(\omega)/H$. Here $e_{\min}$ could be as small as $\delta/2$ and $e_{\max}$ is less than $0.71s$. Recall $\omega < 150.96°$. Hence $H = 0.82s$ and therefore $\theta > 1.42°$. Hence $\theta_c > 0.47°$ and $\omega_c < 179.5°$. An extremely conservative but easy lower bound θ on a center angle is given by the prior bound on the corner angle, and recalling that the center lies inside a convex polygon; see Figure 11c. It can be shown that $e_{\min}\sin\omega = e_{\max}\sin\theta$ where $e_{\min} = \delta/2$ and $e_{\max} = s+\delta$ and $\omega < 150.9°$. Hence $\sin\theta > \sin(\omega)/10$ and then $\theta > 2.79°$. While the theoretical bounds on the center angle are not very strong, they are not tight and they do suggest that the algorithm provably creates convex quads, and in practice we achieve much better quality.

4. Experimental Results

To illustrate the capabilities of our algorithm, we apply it to some domains with smooth and sharp features. Figures 12 and 13 show initial uniform meshes and final all-quad meshes applied to a few smooth domains (flower, face, cat, lake, 2 circles). The lake is associated with high curvature while the intersection of the two circles introduces star-shaped boundary feature that is typically a challenge for grid based methods to mesh. Quality metrics of the resulting meshes are summarized in table 1, including minimum and maximum angles, and the min/avg ratios of minimum to maximum edge lengths. Our algorithm generates angles that are mostly 90° and are well-bounded between 45° and 135°.

Input Domain	v	E	$\theta_{\min}$	$\theta_{\max}$	$\alpha_{\min}$	α_{avg}
flower	4150	4130	45°	135°	0.107	0.705
face	3580	3496	45°	135°	0.332	0.714
cat	5466	5417	45°	135°	0.132	0.709
lake	153997	153964	45°	135°	0.064	0.718
2 circles	3220	3190	45°	135°	0.108	0.690

Table 1: All-quad mesh quality measures of a few domains: v is the number of mesh vertices, E is the number of mesh elements, $\theta_{\min}$ and $\theta_{\max}$ are the minimum and maximum angles, $\alpha_{\min}$ is the minimum ratio of minimum to maximum edge lengths over mesh elements, and α_{avg} is the average ratio minimum to maximum edge lengths over mesh elements.

5. Conclusions

We have introduced a new algorithm for all quad meshing of non-convex domains, with connected regions, conforming to both the interior and exterior of the domain. Our algorithm is robust and provably-correct. It does not require post-processing cleanup operations, such as pillowing, to control the angle bounds of the final mesh, and can easily handle smooth shapes as well as domains with sharp features and corners. Our next step is to adaptively vary the sizing function along the boundary. A natural extension of the presented algorithm is to apply the same steps to *3d all-hex meshing*. This is simply doable because the steps of our algorithm (repelling, splitting, mid-point subdivision, 2-ref template employment) can be employed in 3d in a similar fashion, especially for smooth surfaces. Some cases of sharp features would certainly be more challenging. The extent of the 3d analysis and implementation is outside the scope of this paper and will therefore be addressed in an upcoming publication.

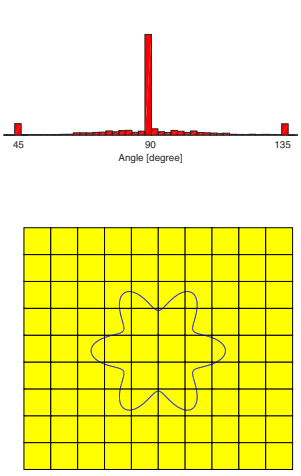
Acknowledgements

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

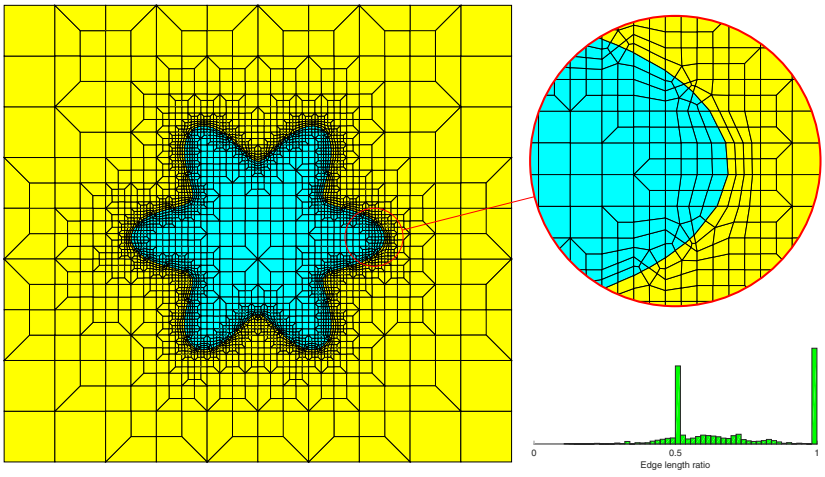
References

[1] K. Ho-Le, Finite element mesh generation methods: a review and classification, Computer-aided design 20 (1988) 27–38.
[2] V. D. Liseikin, Grid generation methods, Springer Science & Business Media, 2009.
[3] W. Hackbusch, Multi-grid methods and applications, volume 4, Springer-Verlag Berlin, 1985.
[4] K. Lee, Principles of CAD/CAM/CAE systems, Addison-Wesley Longman Publishing Co., Inc., 1999.
[5] A. Thakur, A. G. Banerjee, S. K. Gupta, A survey of CAD model simplification techniques for physics-based simulation applications, Computer-Aided Design 41 (2009) 65–80.
[6] P. M. Knupp, Algebraic mesh quality metrics for unstructured initial meshes, Finite Elements in Analysis and Design 39 (2003) 217–241.
[7] M. L. Brewer, L. F. Diachin, P. M. Knupp, T. Leurent, D. J. Melander, The mesquite mesh quality improvement toolkit., in: IMR, 2003.
[8] R. V. Garimella, M. J. Shashkov, P. M. Knupp, Triangular and quadrilateral surface mesh quality optimization using local parametrization, Computer Methods in Applied Mechanics and Engineering 193 (2004) 913–928.

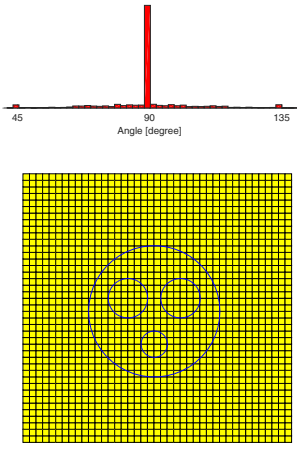
[9] B. D. Anderson, S. E. Benzley, S. J. Owen, Automatic all quadrilateral mesh adaption through refinement and coarsening, in: Proceedings of the 18th International Meshing Roundtable, Springer, 2009, pp. 557–574.
[10] J. Daniels, C. T. Silva, J. Shepherd, E. Cohen, Quadrilateral mesh simplification, in: ACM Transactions on Graphics (TOG), volume 27, ACM, 2008, p. 148.
[11] J. Peters, U. Reif, The simplest subdivision scheme for smoothing polyhedra, ACM Transactions on Graphics (TOG) 16 (1997) 420–431.
[12] H. Prautzsch, Q. Chen, Analyzing midpoint subdivision, Computer Aided Geometric Design 28 (2011) 407–419.
[13] D. Eppstein, Diamond-kite adaptive quadrilateral meshing, Engineering with Computers 30 (2014) 223–235.
[14] S. J. Owen, M. L. Staten, S. A. Canann, S. Saigal, Q-morph: an indirect approach to advancing front quad meshing, International Journal for Numerical Methods in Engineering 44 (1999) 1317–1340.
[15] M. S. Ebeida, K. Karamete, E. Mestreau, S. Dey, Q-tran: a new approach to transform triangular meshes into quadrilateral meshes locally, in: Proceedings of the 19th International Meshing Roundtable, Springer, 2010, pp. 23–34.
[16] T. D. Blacker, M. B. Stephenson, Paving: A new approach to automated quadrilateral mesh generation, International Journal for Numerical Methods in Engineering 32 (1991) 811–847.
[17] J. Zhu, O. Zienkiewicz, E. Hinton, J. Wu, A new approach to the development of automatic quadrilateral mesh generation, International Journal for Numerical Methods in Engineering 32 (1991) 849–866.
[18] D. R. White, P. Kinney, Redesign of the paving algorithm: Robustness enhancements through element by element meshing, in: 6th International Meshing Roundtable, Citeseer, 1997, pp. 323–335.
[19] P. L. Baehmann, S. L. Wittchen, M. S. Shephard, K. R. Grice, M. A. Yerry, Robust, geometrically based, automatic two-dimensional mesh generation, International Journal for Numerical Methods in Engineering 24 (1987) 1043–1078.
[20] R. Schneiders, R. Schindler, F. Weiler, Octree-based generation of hexahedral element meshes, in: Proceedings of the 5th International Meshing Roundtable, Citeseer, 1996.
[21] X. Liang, M. S. Ebeida, Y. Zhang, Guaranteed-quality all-quadrilateral mesh generation with feature preservation, Computer Methods in Applied Mechanics and Engineering 199 (2010) 2072–2083.
[22] X. Liang, Y. Zhang, Hexagon-based all-quadrilateral mesh generation with guaranteed angle bounds, Computer Methods in Applied Mechanics and Engineering 200 (2011) 2005–2020.
[23] X. Liang, Y. Zhang, Matching interior and exterior all-quadrilateral meshes with guaranteed angle bounds, Engineering with Computers 28 (2012) 375–389.
[24] K. Shimada, J.-H. Liao, T. Itoh, et al., Quadrilateral meshing with directionality control through the packing of square cells., in: IMR, 1998, pp. 61–75.
[25] M. Bern, D. Eppstein, Quadrilateral meshing by circle packing, International Journal of Computational Geometry & Applications 10 (2000) 347–360.
[26] F. B. Atalay, S. Ramaswami, D. Xu, Quadrilateral meshes with bounded minimum angle, in: Proceedings of the 17th International Meshing Roundtable, Springer, 2008, pp. 73–91.
[27] T. Ju, F. Losasso, S. Schaefer, J. Warren, Dual contouring of hermite data, ACM Transactions on Graphics (TOG) 21 (2002) 339–346.
[28] Y. Zhang, J. Qian, Dual contouring for domains with topology ambiguity, Computer Methods in Applied Mechanics and Engineering 217 (2012) 34–45.
[29] Y. Zhang, C. Bajaj, Finite element meshing for cardiac analysis, Univ. of Texas at Austin: ICES Technical Report (2004).
[30] M. S. Ebeida, A. Patney, J. D. Owens, E. Mestreau, Isotropic conforming refinement of quadrilateral and hexahedral meshes using two-refinement templates, International Journal for Numerical Methods in Engineering 88 (2011) 974–985.



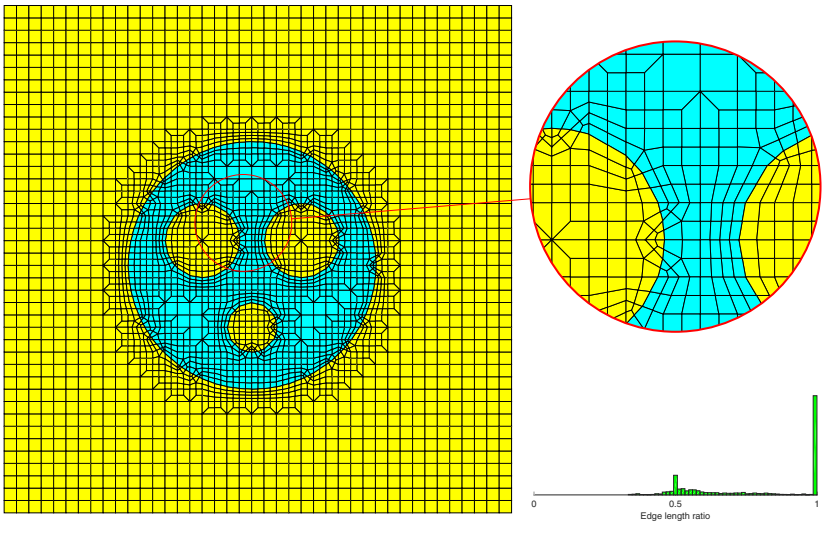
(a) Initial grid and domain.



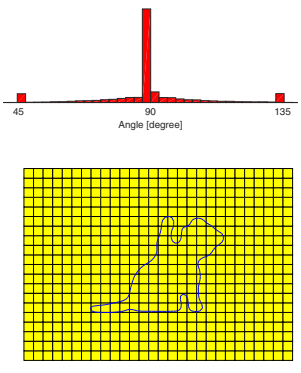
(b) Final all-quad mesh, a magnified region, and histograms of angles and edge lengths ratios.



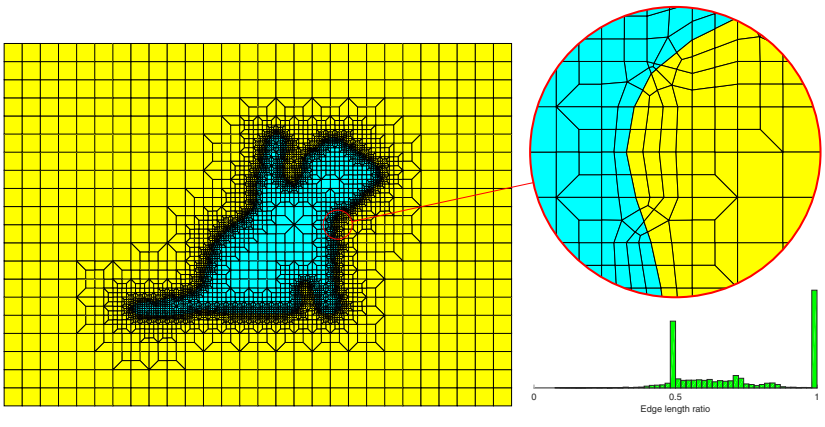
(c) Initial grid and domain.



(d) Final all-quad mesh, a magnified region, and histograms of angles and edge lengths ratios.

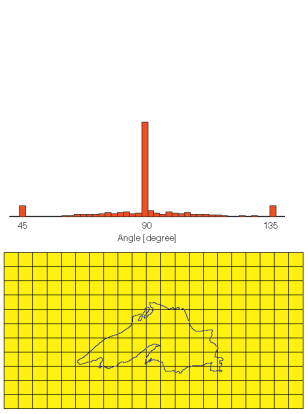


(e) Initial grid and domain.

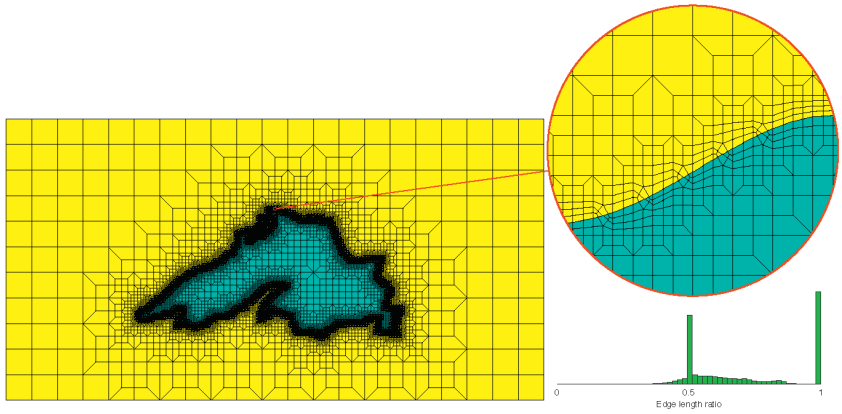


(f) Final all-quad mesh, a magnified region, and histograms of angles and edge lengths ratios.

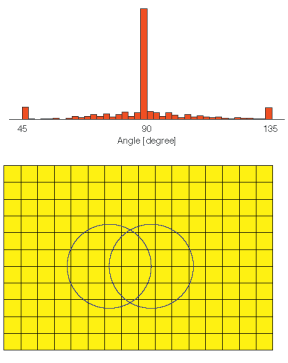
Fig. 12: Application of the all-quad meshing algorithm to a valentine flower, a face, and a cat-shaped domains.



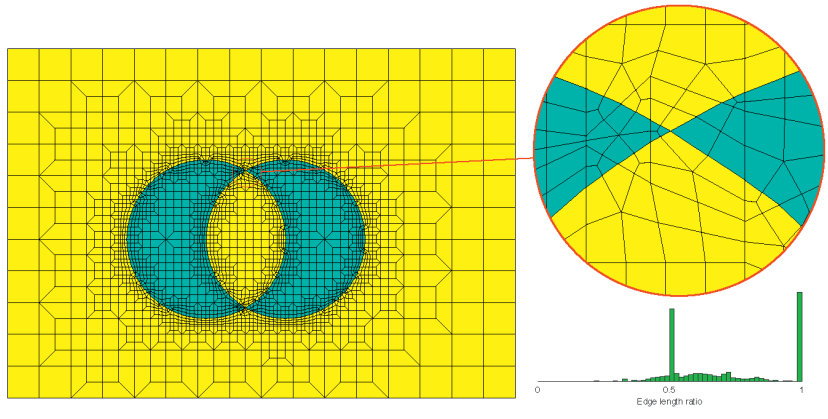
(a) Initial grid and domain.



(b) Final all-quad mesh, a magnified region, and histograms of angles and edge lengths ratios.



(c) Initial grid and domain.



(d) Final all-quad mesh, a magnified region, and histograms of angles and edge lengths ratios.

Fig. 13: Application of the all-quad meshing algorithm to a lake and intersecting circles domains.