The Impact of Mesh Regularity on Errors

by

Hongliang Fan

B.Eng., Beihang University, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Mechanical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

March 2017

Abstract

Mesh quality plays an important role in improving the accuracy of the numerical simulation. There are different quality metrics for specific numerical cases. A regular mesh consisting of the equilateral triangles is one of them and is expected to improve the error performance. In this study, Engwirda's frontal-Delaunay scheme and Marcum's advancing front local reconnection scheme are described along with the conventional Delaunay triangulation. They are shown to improve the mesh regularity effectively. Even though several numerical test cases show that more regular meshes barely improve the error performance, the time cost in the solver of regular meshes is smaller than the Delaunay mesh. The time cost decrease in the solver pays off the additional cost in the mesh generation stage. For simple test cases, more regular meshes obtain lower errors than conventional Delaunay meshes with similar time costs. For more complicated cases, the improvement in errors is small but regular meshes can save time, especially for a high order solver. Generally speaking, a regular mesh does not improve the error performance as much as we expect, but it is worth generating.

Preface

The research ideas and methods explored in this thesis are the fruits of a close working relationship between Dr. Carl Ollivier-Gooch and Hongliang Fan. The implementation of the methods, the data analysis, and the manuscript preparation were done by Hongliang Fan with invaluable guidance from Dr. Carl Ollivier-Gooch throughout the process.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Acknowledgements

I sincerely thank my research supervisor Dr. Carl Ollivier-Gooch for his patience, encouragement, and guidance in helping me to accomplish my graduate study. He is an awesome supervisor and mentor.

Also I would like to thank Gary Yan, Daniel Zaide, and the rest of my colleagues in the ANSLab research group for their suggestions and advice, which are inspiring and valuable to my study.

Dedication

I am grateful for my parents' unconditional love and support. Their encouragement and understanding make me stronger.

Chapter 1

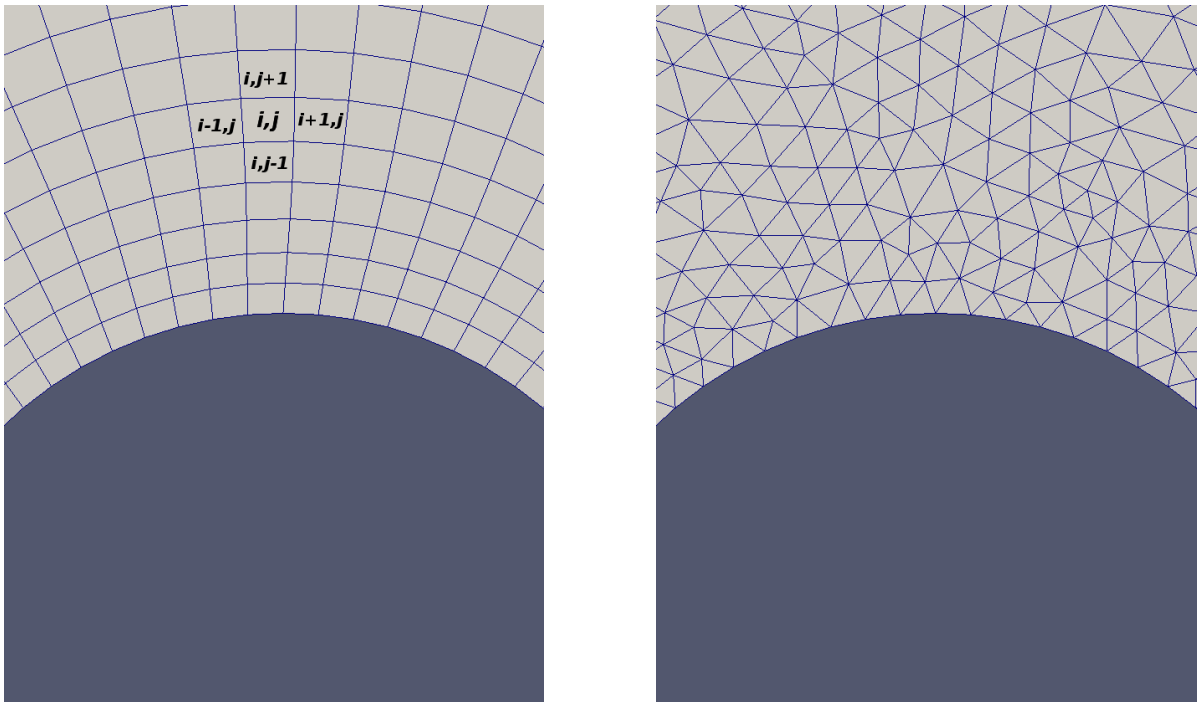
Introduction

Computational Fluid Dynamics, or CFD, is one of the three major methods in the study of fluids, along with theoretical and experimental analysis. CFD simulates the physical phenomena in the flow of fluids numerically [21]. These phenomena, such as shockwaves and the turbulence around a wing, are governed by the Navier-Stokes equations. In most cases, the models of interest are non-linear and have no analytical solutions. CFD allows an alternative for the understanding of complex flows via numerical simulations instead of physical experiments. Compared to experimental and theoretical fluid analysis, CFD is young and dramatically growing [2]. In the 1960s, digital computers began to be available, which made possible numerical simulations. In the following 50 years, CFD has grown with the development of more powerful computers and algorithms. Now, CFD has been applied to multiple disciplines such as fluid mechanics, electrical and electronic engineering, meteorology, biomedical engineering and more.

The complete CFD process contains three major steps: pre-processing, flow solution and post-processing.

Pre-Processing: This stage prepares all the information the flow solver needs. Given a particular problem, we choose a sufficient and representative model of the physics. Different models have different approximations. The approximation introduces physical modeling error, which means the flow of interest is not analyzed perfectly.

The geometry of the computational domain usually is defined by computer aided design (CAD) software. Then the domain is divided into finite non-overlapping cells or control volumes. This division process is known as mesh generation. It is also the focus



(a) Structured mesh

(b) Unstructured mesh

Figure 1.1: Meshes around a circular cylinder (Near View)

of this study. Generally speaking, the more cells there are in the domain, the more accurate the solution will be. However, finer meshes also require higher computational resources. Besides the size of the mesh, the connectivity and the shape of the cells also have impacts on the accuracy. Based on how the cells are connected, as illustrated in Fig. 1.1, meshes can be classified as structured and unstructured meshes. The connectivity of the structured mesh is predictable. As shown in Fig. 1.1a, the cell (i, j) always has adjacent cells marked as $(i \pm 1, j \pm 1)$. This regular connectivity makes structured meshes easier to deal with in the flow solver. It works well in simple geometries. However, when it comes to complicated geometries, it is not always possible to generate structured meshes automatically. Therefore, unstructured meshes are more widely used in CFD for complicated geometries. Though the connectivity must be stored explicitly in the flow solver, which makes the codes more complicated and challenging to write, current high-order accurate algorithms make it worthwhile to adopt unstructured meshes on complicated

geometries. The mesh generation software used in the study is GRUMMP (Generation and Refinement of Unstructured, Mixed-Element Meshes in Parallel) [27].

Flow Solution: In the pre-processing stage, we choose the mathematical governing equations which, in fluid dynamics, are partial differential equations with some algebraic constraints. To solve these equations numerically, we need to discretize the mathematical equations into algebraic equations within each cell. After decades of development, the current three major families of discretization schemes are finite-difference [21, 1], finite-element [30], and finite-volume schemes [37]. The finite-volume scheme is used in this study because of its advantages both in conservation of physics quantities and its easy application on unstructured meshes [21]. The algebraic equations will be solved numerically instead of the original mathematical equations. The approximation in the discretization both in space and time introduces truncation errors. The solver stops when the residual of the discrete equation is close to zero. The difference between the numerical solution and the exact solution is known as discretization error. In this study, we use our in-house code ANSLib (Advanced Numerical Simulation Library) [26] as the flow solver.

Post-Processing: This stage is the visualization and analysis of the numerical solution obtained from the flow solution stage, which includes geometry and mesh display, vector and contour plots, error estimation on the variables of interest and more. Visualization is usually completed with the help of plotting tools like Paraview [3], MATLAB [23], and gnuplot [38].

1.1 Motivation

As mentioned in the previous section, the accuracy of CFD is evaluated by how we control the physical modeling error and the numerical error. The latter is the target in this study. Usually, the more cells in the mesh, the more accurate we expect the solution to be. However, finer

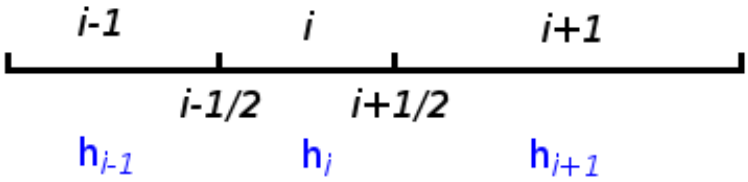


Figure 1.2: Simple 1-D example

meshes introduce larger costs. The cell shape also influences the solution accuracy. We take the one-dimensional case as a simple example to illustrate the concept. The governing equation for steady heat conduction is

$$\frac{d^2T}{dx^2} = 0. \tag{1.1}$$

Integrating the governing equation in cell i and applying Gauss's theorem, we get

$$\frac{1}{h_i}\left(\left.\frac{dT}{dx}\right|_{i+\frac{1}{2}} - \left.\frac{dT}{dx}\right|_{i-\frac{1}{2}}\right) = 0. \tag{1.2}$$

The solution in cell i is represented by a reconstruction polynomial

$$T(x) = T_i + (x-x_i)\left.\frac{dT}{dx}\right|_i + \frac{(x-x_i)^2}{2}\left.\frac{d^2T}{dx^2}\right|_i + \frac{(x-x_i)^3}{6}\left.\frac{d^3T}{dx^3}\right|_i + \cdots, \tag{1.3}$$

where x_i indicates the location of the cell reference point. As shown in Fig. 1.2, if we choose the cell center of the control volume (CV) i as the cell reference point and integrate the reconstruction over the cells, the cell averages are,

$$\overline{T}_i = \frac{1}{h_i}\int_{\mathrm{CV}_i} T(x)dx = T|_i + \frac{h_i^2}{24}\left.\frac{d^2T}{dx^2}\right|_i + O(h_i^4)$$

$$\begin{aligned}\overline{T}_{i+1} &= \frac{1}{h_{i+1}}\int_{\mathrm{CV}_{i+1}} T(x)dx \\ &= T|_i + \left(\frac{1}{2}h_i + \frac{1}{2}h_{i+1}\right)\left.\frac{dT}{dx}\right|_i + \left(\frac{1}{8}h_i^2 + \frac{1}{4}h_i h_{i+1} + \frac{1}{6}h_{i+1}^2\right)\left.\frac{d^2T}{dx^2}\right|_i + \\ &\quad \left(\frac{1}{48}h_i^3 + \frac{1}{16}h_i^2 h_{i+1} + \frac{1}{12}h_i h_{i+1}^2 + \frac{1}{24}h_{i+1}^3\right)\left.\frac{d^3T}{dx^3}\right|_i + O(h_i^4)\end{aligned}$$

$$\begin{aligned}\overline{T}_{i-1} &= \frac{1}{h_{i-1}}\int_{\mathrm{CV}_{i-1}} T(x)dx \\ &= T|_i + \left(-\frac{1}{2}h_i - \frac{1}{2}h_{i-1}\right)\left.\frac{dT}{dx}\right|_i + \left(\frac{1}{8}h_i^2 + \frac{1}{4}h_i h_{i-1} + \frac{1}{6}h_{i-1}^2\right)\left.\frac{d^2T}{dx^2}\right|_i + \\ &\quad \left(-\frac{1}{48}h_i^3 - \frac{1}{16}h_i^2 h_{i-1} - \frac{1}{12}h_i h_{i-1}^2 - \frac{1}{24}h_{i-1}^3\right)\left.\frac{d^3T}{dx^3}\right|_i + O(h_i^4).\end{aligned}$$

The first order surface gradients are,

$$\left.\frac{dT}{dx}\right|_{i+\frac{1}{2}} = \frac{\overline{T}_{i+1} - \overline{T}_i}{\frac{h_{i+1}+h_i}{2}} \tag{1.4}$$

$$\left.\frac{dT}{dx}\right|_{i-\frac{1}{2}} = \frac{\overline{T}_i - \overline{T}_{i-1}}{\frac{h_{i-1}+h_i}{2}}. \tag{1.5}$$

Substitute Eqn. 1.4 and Eqn. 1.5 into Eqn. 1.3,

$$\begin{aligned}\frac{1}{h_i}\left(\left.\frac{dT}{dx}\right|_{i+\frac{1}{2}} - \left.\frac{dT}{dx}\right|_{i-\frac{1}{2}}\right) &= \frac{d^2T}{dx^2} + \frac{h_{i+1} - 2h_i + h_{i-1}}{3h_i}\frac{d^2T}{dx^2} + \\ &\quad \frac{-h_i h_{i-1} + h_i h_{i+1} - h_{i-1}^2 + h_{i+1}^2}{12h_i}\frac{d^3T}{dx^3} + O(h^2).\end{aligned} \tag{1.6}$$

Comparing Eqn. 1.6 to Eqn. 1.1, we see that our discretization is not exact. This error, called the truncation error, is zeroth-order for arbitrary choices of cell sizes. On the other hand, if the mesh is regular, or $h_{i+1} = h_i = h_{i-1}$, the truncation error is second-order. Smooth variations of h also lead to second-order truncation error. Juretić and Gosman proved that the cell shape also affects the truncation error in 2-D and 3-D cases [19]. The prediction of error in the solution, or the discretization error, and the relations between truncation and discretization errors are

complex [32, 7].

Katz et al. [20] showed that mesh regularity affects truncation and discretization errors. The truncation error obtained from the regular mesh is higher order than those from perturbed meshes. As for the solution error, the advantage in the order of accuracy is only observed in inviscid fluxes. The viscous fluxes are not sensitive to mesh regularity in the sense of having an effect on solution error. Diskin et al. [8] also found that the relations between mesh characteristics and the solution accuracy are complicated. Mesh irregularities affect gradient errors and discretization errors in different ways. The viscosity affects the sensitivity to mesh regularity, as does the solution reconstruction method. For isotropic meshes, mesh regularity does not affect gradient error much if the unweighted least squares method is used. However, if using the Green-Gauss method, the regular quadrilateral mesh offers a higher order of accuracy than others in his test. In addition, Jalali et al [16, 18] and Yan et al [39] investigated the accuracy of different discretization schemes on a wide range of meshes from perfect to perturbed meshes for cell-centered and vertex-centered control volumes respectively. The degradation on the order of accuracy for truncation errors on perturbed meshes is observed in their research. They found that a particular discretization scheme may minimize the truncation error for a certain kind of mesh.

Most research on mesh regularity and errors are applied for perfect meshes and their perturbations for simple geometric domains. This study focuses on how regular meshes can be automatically generated for more complicated geometries, and how that affects the truncation and discretization errors. In particular, this thesis seeks to answer the following questions:

1. How regular a triangular mesh can we generate for realistic cases?

2. What is the additional cost to generate such a mesh?

3. Considering that the cell shape affects the accuracy greatly, can we expect small discretization error using unstructured regular meshes?

4. Can a coarse regular mesh offer the same accuracy as a finer irregular mesh with lower

costs in the flow solver?

1.2 Outline

In the next chapter, we will recall some background information on both mesh generation techniques and unstructured mesh finite volume schemes. The Delaunay mesh refinement algorithm is discussed. We will also introduce post-processing techniques for mesh generation such as smoothing and swapping.

In Chapter 3 and 4, the methodology to generate more regular meshes in this study is described. Two mesh generation techniques are used in the study: Engwirda's frontal-Delaunay algorithm and Marcum's advancing front local reconnection algorithm. Both are proven to improve the mesh regularity dramatically. Several changes are made in these two schemes to work on our current GRUMMP framework.

In Chapter 5, several numerical test cases are applied. In each case, the meshes generated by different mesh generation techniques are shown and discussed first. The comparison of the mesh regularity shows the advantage of new mesh generation techniques. The truncation error and discretization error are also visualized to illustrate the effects of the mesh regularity. The cost is shown at last to discuss whether it is worth it to generate more regular meshes.

In Chapter 6, the thesis concludes with the summary of this research.

Chapter 2

Background

Delaunay mesh refinement is chosen as a base-line mesh generation scheme because of its theoretical guarantees and its practical performance. It will be introduced in Section 2.1. Post-processing techniques such as edge swapping and vertex smoothing, which help improve the mesh quality, will be discussed in Section 2.2. Section 2.3 provides a brief review of the finite-volume method.

2.1 Delaunay mesh refinement

In two dimensions, a triangulation is a set of triangles that connect all given vertices. Triangles cannot intersect each other and all triangles collectively must fill the polygonal boundary of the domain. A Delaunay triangulation is the triangulation in which all triangles have empty circumcircles, which means that there are no vertices lying inside circumcircles[1] but the vertices are allowed to be located on circumcircles. Fig. 2.1 shows a simple example of a Delaunay triangulation. All the circumcircles (dashed lines) are empty. If no four vertices lie on the same circle, the Delaunay triangulation is guaranteed to be a unique triangulation. The proof can be found in Reference [6, 9]. However, a Delaunay triangulation may break the domain boundary in some cases. As shown in Fig. 2.2, the boundary is marked with bold lines. Because one boundary point fails to meet the empty circumcircle criterion (left), the Delaunay triangulation (right) of these vertices changes the domain boundary. To solve this problem, the constrained Delaunay triangulation was proposed [5]. Unlike the Delaunay triangulation which

[1]The circumcircle is a triangle's circumscribed circle. It is the unique circle that passes through all the triangle's three vertices. The center of the circumcircle is called the circumcenter, and its radius is called the circumradius.

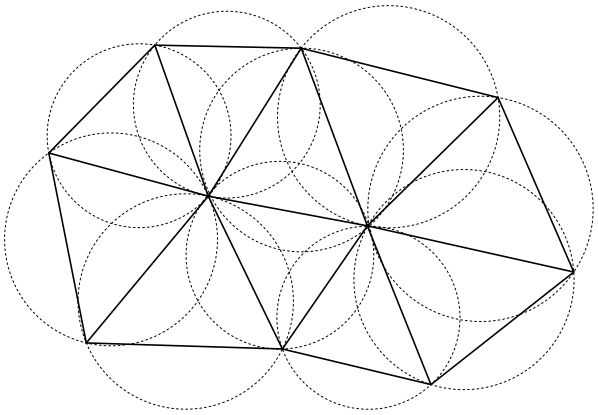


Figure 2.1: Delaunay Triangulation

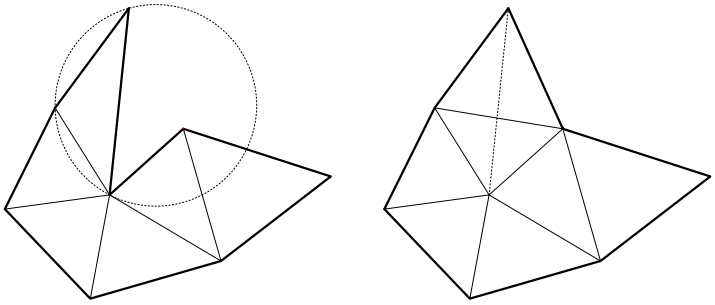
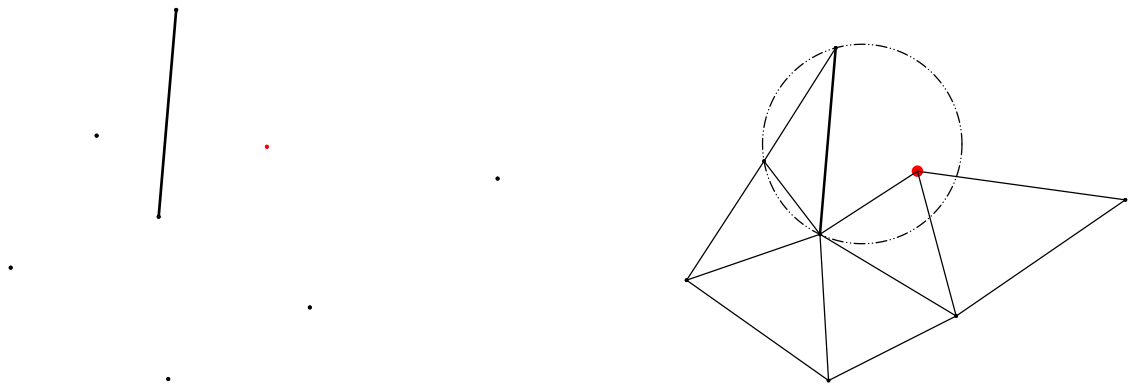


Figure 2.2: Delaunay Triangulation breaks the boundary



(a) PSLG (Eight points and one segment)

(b) Constrained Delaunay Triangulation (Boundary Protected)

Figure 2.3: Constrained Delaunay

starts with a set of vertices, the constrained Delaunay triangulation starts with a planar straight line graph (PSLG). The PSLG contains both vertices and segments, as shown in Fig. 2.3a. Every endpoint of the segments must be among the vertices in this PSLG. The segments are only permitted to intersect at endpoints. In a constrained Delaunay triangulation, the segments in PSLG serve as edges of the triangles. Only vertices that are visible to a triangle are considered when checking empty circumcircles. Visibility means that no input segment stands between the vertex and the triangle. By setting boundary edges as segments in the PSLG, we can protect them from breaking by the Delaunay triangulation. Fig. 2.3b illustrates how a constrained Delaunay triangulation conserves boundaries. The red vertex is located inside the circumcircle of a triangle. However, the segment in the PSLG (marked as bold line) separates the vertex from that triangle. Therefore the red vertex is not visible to the triangle whose circumcircle contains it. This is not regarded as a violation of the empty circumcircle rule so the boundary is conserved.

The algorithm used in GRUMMP to generate guaranteed-quality triangular meshes [4, 28] is based on Ruppert's scheme [33]. Ruppert's scheme starts with the constrained Delaunay triangulation. The Delaunay triangulation maximizes the minimum angles in meshes but usually, bad quality triangles still exist. The quality of the triangulation is improved by inserting points into the constrained Delaunay triangulation. Ruppert's scheme inserts circumcenters of the bad

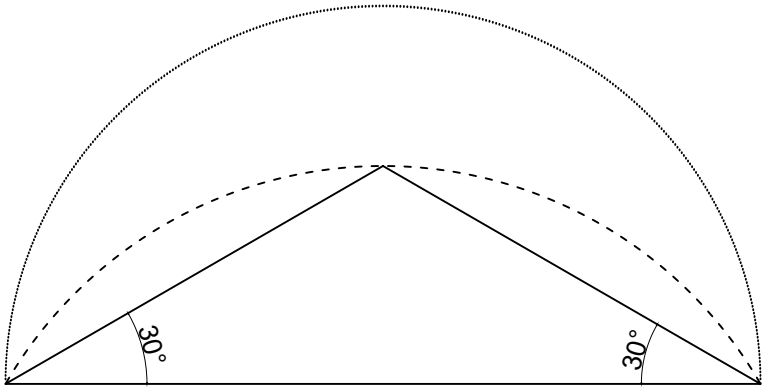


Figure 2.4: Diametral circle (dotted line) and diametral lens (dashed line)

quality triangles into the mesh unless the circumcenters encroach boundaries. Encroachment happens when a point is located inside a circle with a boundary edge as its diameter. Ruppert showed that the algorithm terminates with the minimum angle in the mesh larger than 20.7°. Shewchuk showed that when using diametral lenses instead of diametral circles to determine encroachment, the minimum angle in the mesh is greater than 25.7° [34]. The difference between diametral lenses and diametral circles is illustrated in Fig. 2.4.

To generate a guaranteed-quality triangular mesh, both element size and shape need to be considered. The size is controlled by length scales. If the circumradius of a triangle is less than the average length scales of its three vertices, that triangle satisfies the size constraint. GRUMMP uses a user controlled refinement parameter R to determine how small the elements are. Small triangles are usually not required everywhere in the mesh. In the far field or the areas where variables of interest tend to be constant, triangles are permitted to be much larger. Therefore, a grading parameter G is introduced to control how fast the edge length grows over distance. The length scale (LS) for point p is defined with these two parameters,

$$\mathrm{LS}(p) = \min\left(\frac{\mathrm{lfs}(p)}{R}, \min_{\text{neighbours } q_i} \mathrm{LS}(q_i) + \frac{1}{G}|q_i - p|\right). \tag{2.1}$$

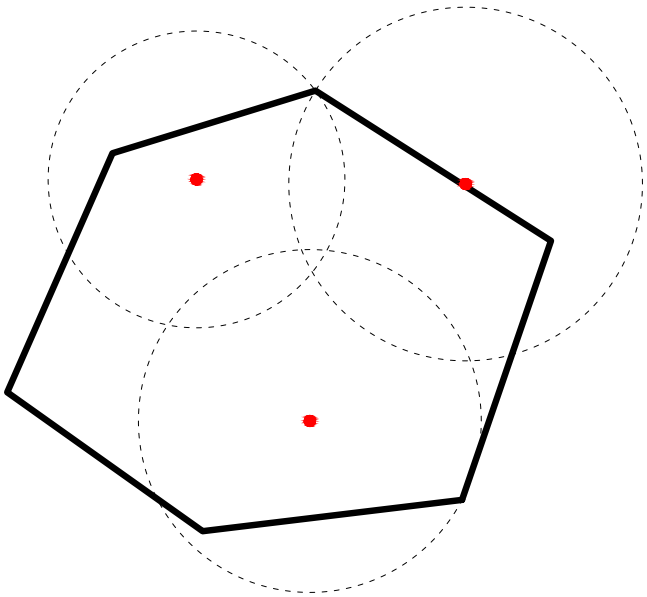


Figure 2.5: Local feature size

q_i denotes the neighbour points to point p. In Fig. 2.5, local feature sizes (lfs) for points (red points) are the radii of those circles. The lfs is defined as the radius of the smallest circle centered at point p that touches two disjoint parts (vertices or edges) of the domain boundary. A large value of R reduces the target edge length while a larger value of G leads to a slower increase in element size over the same distance. As mentioned before, the minimum angle in constrained Delaunay triangulation is guaranteed to be larger than 25.7° if the diametral lens is used for encroachment. A triangle is regarded as good in shape when the minimum angle of it is larger than 25.7°. In GRUMMP, we use the normalized shortest edge length to circumradius ratio to evaluate the shape. The shape quality ρ is

$$\rho = \frac{\ell_m}{\sqrt{3}r}.$$

The quality varies from 0 to 1. An equilateral triangle enjoy the maximum shape quality 1 and a degenerate triangle has minimum shape quality 0. According to the law of sines, the minimum edge length ℓ_m, the minimum angle α_m, and the circumradius r have the relationship

below,

$$\frac{\ell_m}{\sin(\alpha_m)} = 2r.$$

Therefore the shape quality can also be expressed as

$$\rho = 2\sin(\alpha_m)/\sqrt{3}. \tag{2.2}$$

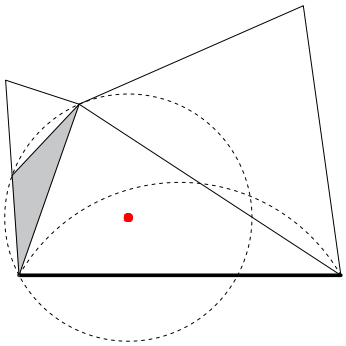
When the diametrial lens is used for encroachment, the minimum shape quality allowed in the mesh is:

$$\rho = \frac{2\sin(25.7^\circ)}{\sqrt{3}} \approx 0.5.$$

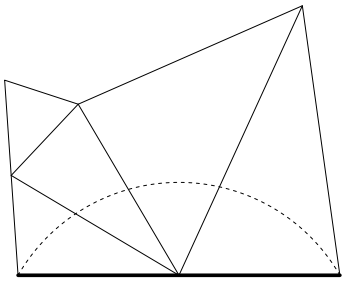
If the diametral circle is used instead, the minimum shape quality is about 0.4. Only those triangles which satisfy both size and shape constraints are considered as good quality elements.

The process of inserting vertices into an initial constrained Delaunay triangulation to improve the mesh quality is called mesh refinement. In GRUMMP, all bad triangles in the mesh are queued initially. The triangle with the worst quality comes at the top of the queue. The insertion procedure grabs the top triangle in the queue and calculates the circumcenter of that bad triangle as the location for a new vertex to insert. If the newly inserted vertex encroaches boundaries, the vertex is rejected and instead, the encroached boundary edge is divided in half, as shown in Fig. 2.6. If not, we use Watson's method [31] to insert the vertex into the mesh. All triangles whose circumcircles contain that newly inserted vertex are deleted, leaving a hull with that new vertex in it. Connecting faces of the hull with the newly inserted vertex forms several new triangles. New triangles with bad quality need to be queued for later splitting while good ones do not. The refinement procedure terminates when the queue is empty. Fig. 2.7 illustrates Watson insertion when a new vertex does not encroach boundaries. The gray triangle in Fig. 2.7a is the bad one to be refined, whose circumcenter is marked as red. The triangles with bold lines are those whose circumcircles contain that red vertex. The hull and the triangulation after Watson insertion are shown in Fig. 2.7b and Fig. 2.7c.

The priority queue often plays an important role in the mesh quality. In GRUMMP, badly

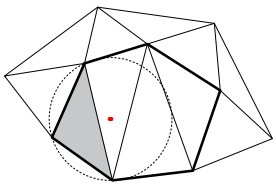


(a) The newly inserted vertex encroaches the boundary (bold line)

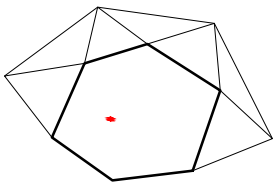


(b) The boundary is split in half

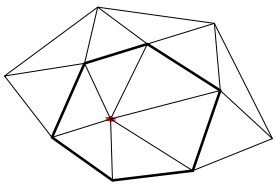
Figure 2.6: Boundary Encroachment



(a) The circumcenter of the bad triangle is calculated



(b) Deleting the triangles leaving the hull



(c) New triangles generated

Figure 2.7: Watson Insertion

Algorithm 2.1 Initial Queuing algorithm

```
i = 0
while i <number of triangles in the mesh do
    if Triangle T_i is still in the mesh and fails quality test then
        Input T_i with priority from quality test into the queue
    end if
    i = i + 1
end while
```

Algorithm 2.2 Quality Test

```
Calculate the circumradius R
Calculate the average length scale l̄ of three vertices in T
Find the shortest edge length l_min
if R > l̄ then
    Quality = −R/l̄
else
    Quality = l_min/(√3 R)
    if Quality > WorstShapeAllowable then
        Quality = NeverQueue
    end if
end if
```

sized elements, the ones with the large circumradius to averaged length scale ratio, come before badly shaped elements, largest first. If triangles satisfy the size constraint, the shape quality is considered. The triangle with the smallest shape quality comes at the top of the queue. Algorithm 2.1 and Algorithm 2.2 illustrate how we deal with the priority queue in GRUMMP. Triangles with "NeverQueue" quality will not added to the queue. Other details such as how to handle small angles in the domain can be found in Reference [4]. The flow chart for this scheme is shown in Fig. 2.8.

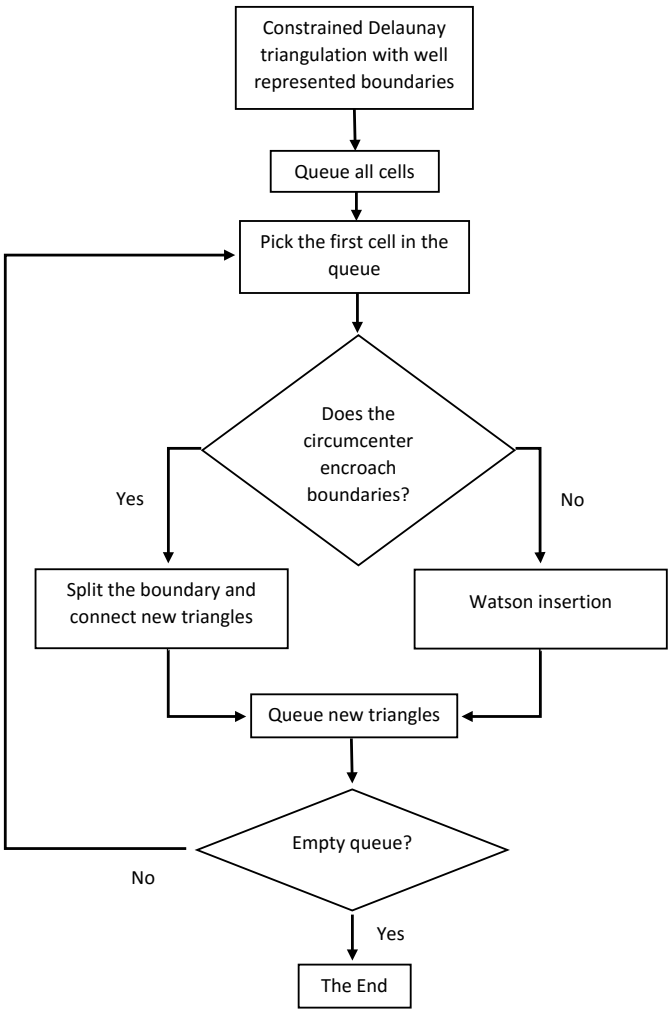


Figure 2.8: Flow chart for refinement algorithm

Mesh generation for domains whose boundaries are straight lines is straightforward. Boundaries can be simply split into edges with the user-controlled size. The constrained Delaunay triangulation with the boundary discretized serves as the starting triangulation of the refinement algorithm discussed before. However, the discretization of curved boundaries is more challenging. As illustrated in Fig. 2.9, for high curvature curves, they should be discretized into more edges locally to capture local curvature features. However, the fine discretization is not required for the low curvature parts on curves. The algorithm for discretizing both parts in different ways automatically is not straightforward. In GRUMMP, the technique is based on a total variation of tangent angles (TVT). The details can be found in Reference [4]. The TVT boundary discretization scheme starts with a very coarse discretization. Fig. 2.10 shows the starting boundary edges in blue lines while the actual geometry curve is marked as red. After queueing all the starting boundary edges, pick the first one in the queue and split it if the edge length is larger than the local length scale. If the local curvature is small, the edge is split at the geometric average location. Otherwise the split location is calculated based on TVT so that the change in normal direction is divided equally between the two new segments. The length scale for a point p on a curved boundary is calculated by Eqn. 2.1, where local feature size should be modified with curvature,

$$\mathrm{lfs_c}(p) = \min(\rho(p), \mathrm{lfs}(p)).$$

ρ denotes to the radius of curvature. The boundary discretization based on this approach is shown in Fig. 2.11. The TVT split scheme successfully uses more edges around the leading edge to capture high curvature curve. The local feature size at the trailing edge is large in the starting triangulation. Therefore there is not much for the TVT split scheme to do. The very coarse discretization around the trailing edge is not good news because it may fail to capture rapidly changing solution features. This can be fixed by manually setting a small length scale at the trailing edge but there is still a problem. Usually, the curvature is nearly constant around the trailing edge so boundary edges are simply split in half. In this scheme, there are some transition areas where the length of one edge is twice as large as the adjacent one. This makes

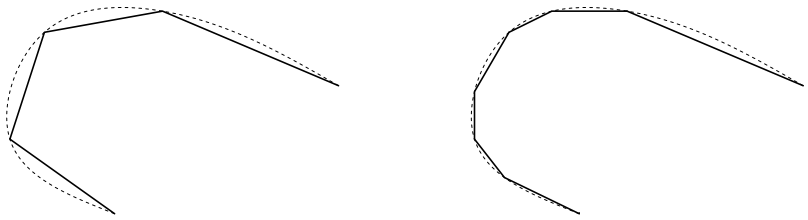


Figure 2.9: Boundary discretization

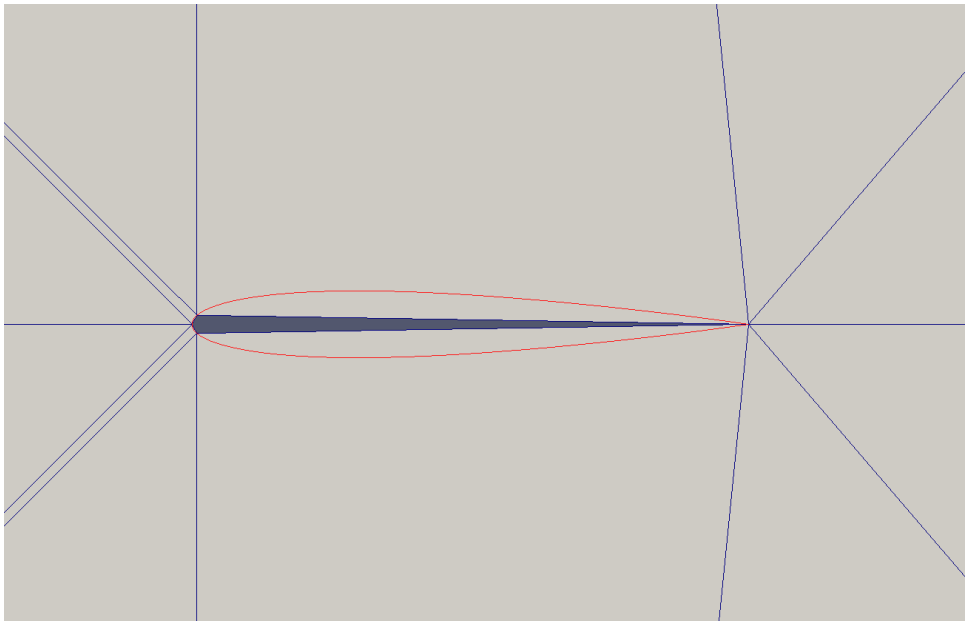


Figure 2.10: Constrained Delaunay trianglulation before boundary recovery

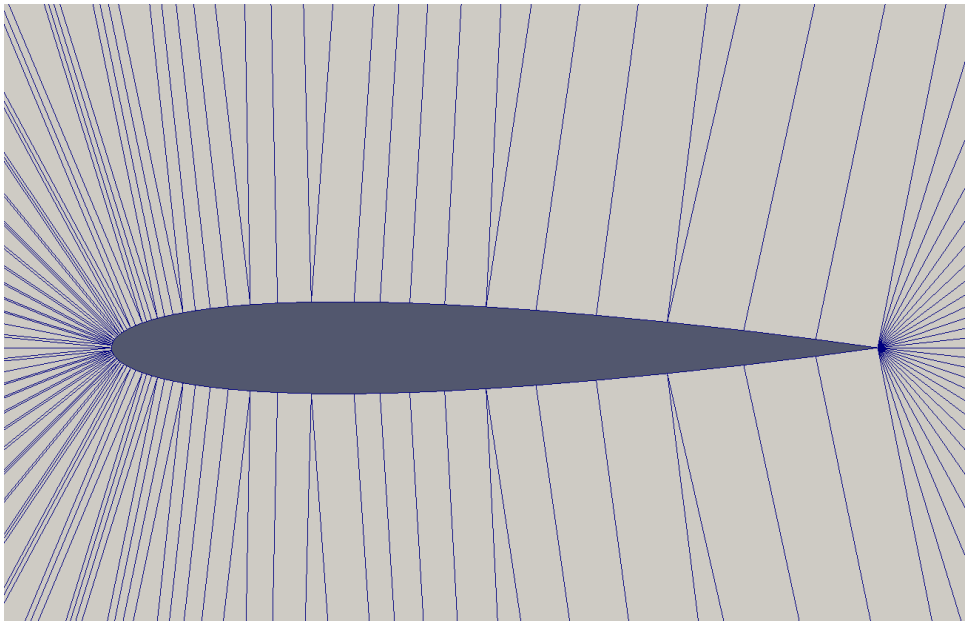


Figure 2.11: Boundary discretization based on TVT split

it difficult to generate equilateral triangles.

The improved boundary discretization algorithm is introduced to deal with trailing edges of airfoils [40]. The new scheme starts at end point of an airfoil, which is usually the trailing edge. Given a parametric curve, $C(t)$, the length at parameter t is marked as $\ell(t)$. We denote total curve length as $\bar{\ell}$. The average length scale is easily calculated as

$$LS(t) = \frac{\bar{\ell}}{R_b},$$

where R_b is the refinement parameter to decide how fine the boundary would be discretized. Approximately, it denotes the number of edges along the curve. Boundary grading parameter G_b is also needed to determine the boundary edge length growth rate. G_b is the ratio of the largest boundary edge length to the smallest one. A large value of G_b results in a small length scale at trailing edge. Based on the average length scale, we would expect length scales are smaller near end points or parts with high curvature. For airfoils, these would be trailing edges and leading edges. Within a certain fraction (In this study, $\alpha = 0.25$) of total length distance to

end points, which denotes the region of trailing edges,

$$\min(\ell(t), \bar{\ell} - \ell(t)) < \ell_{\text{end}} = \alpha\bar{\ell} \;\; \alpha \in (0, 0.5),$$

smaller length scales are applied,

$$LS(t) = \frac{1}{2}\left(\frac{\bar{\ell}}{R_b} - \frac{1}{G_b}\frac{\bar{\ell}}{R_b}\right)\left(\sin\left(\pi\frac{\ell(t)}{\ell_{\text{end}}} - \pi/2\right) + 1\right) + \frac{1}{G_b}\frac{\bar{\ell}}{R_b}.$$

This also makes the edge length growth near the trailing edge more smooth to fix the problem that one edge length is twice large as the adjacent one in the previous scheme. Near the leading edge, if $\rho(t) < \frac{\bar{\ell}}{R_b}$, it is regarded as a high curvature area. The smaller length scale for this area is

$$LS(t) = \max\left(\rho(t), \frac{\bar{\ell}}{R_b}\frac{1}{G_b}\right).$$

Instead of dividing current edges recursively as used in the TVT boundary discretization scheme, the new algorithm discretizes the boundary by marching from one end to another. The starting parameter is 0. Insert the point P_1 with a distance $LS(0) = \bar{\ell}/(R_b \cdot G_b)$ from the end point P_0 on a curve. The parameter t_1 of this distance is obtained from the parametric curve. Point P_2 is inserted with a distance $LS(t_1)$ from P_1. The first two insertions are shown in Fig. 2.12. Keep inserting points until the point to be inserted is too close to the other end point. In my study, distance less than $0.5LS(t_{\max})$ is regarded as the termination of the insertion. Fig. 2.13 shows a smooth boundary discretization with this new scheme. Both the leading edge and trailing edge are treated properly.

2.2 Post-processing

After the refinement, when all triangles have good shapes and sizes, post-processing procedures such as swapping and smoothing are applied to improve mesh quality.

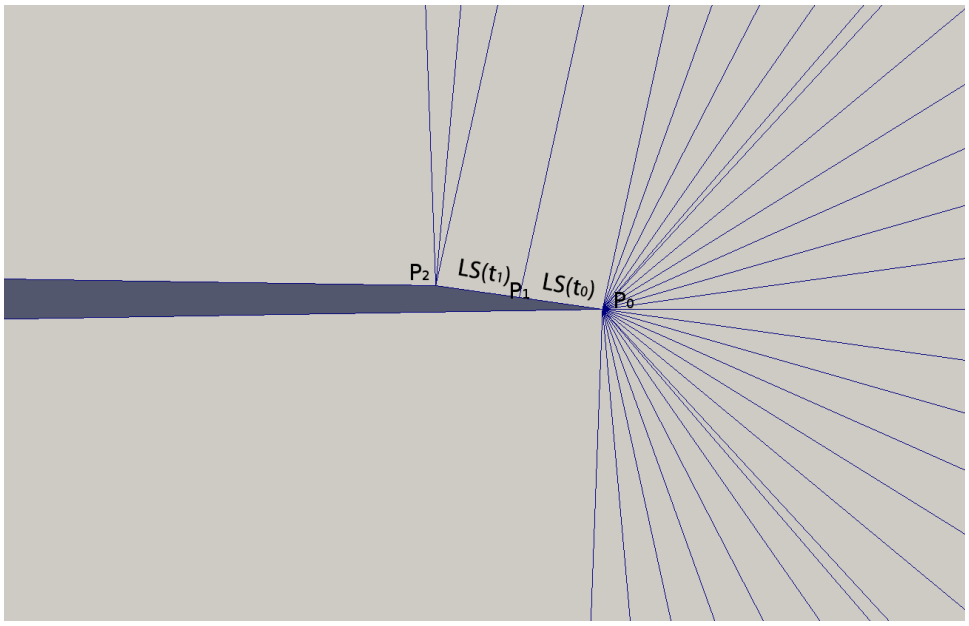


Figure 2.12: New boundary discretization (after second insertion)

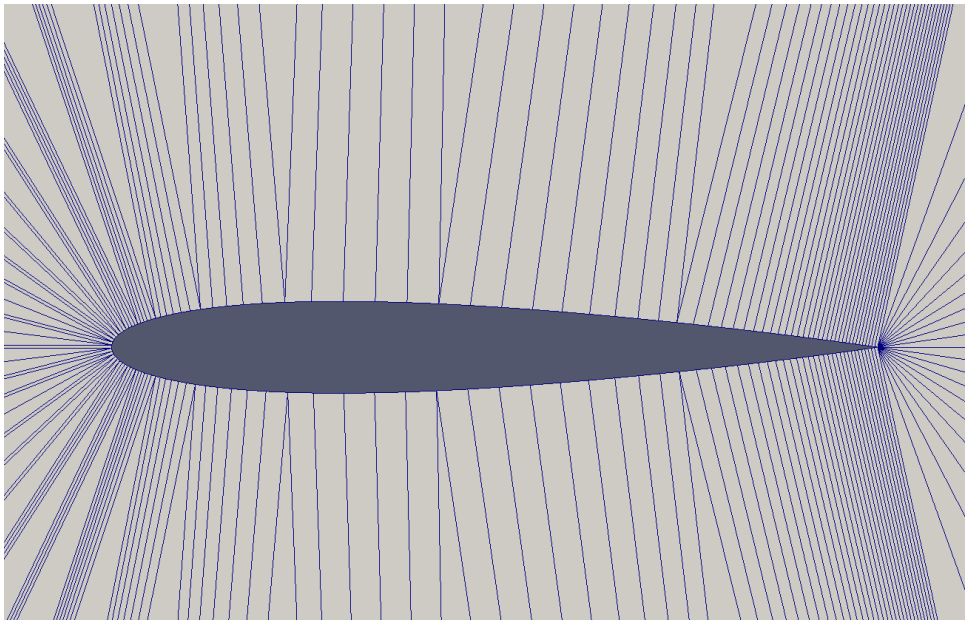


Figure 2.13: New boundary discretization

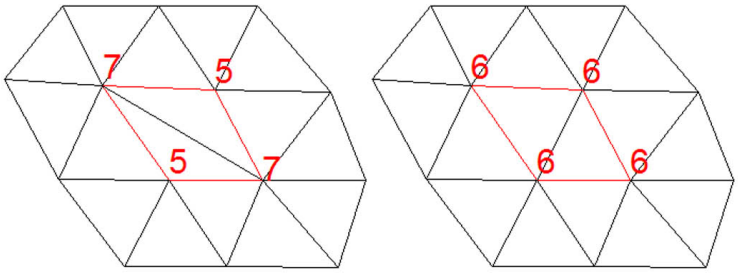


Figure 2.14: Uniform-degree edge swapping

Algorithm 2.3 Uniform degree swapping

```
Queue all faces
while The queue is not empty do
    Grab the first face in the queue and its two adjacent triangles
    Find degrees of four vertices and calculate their variance from six
    Calculate the variance after swapping
    Remove the face from the queue
    if The variance after swapping is smaller then
        Do swapping
        Add the new diagonal to the queue
    end if
end while
```

2.2.1 Swapping

In the triangular mesh, two adjacent triangles have four different vertices. If they form a convex quadrilateral, there are two possible configurations. Based on a certain criterion, if the other configuration is better than the current one, it can be easily acquired by changing the diagonal.

A regular mesh consists of equilateral triangles, where all vertices have six incident cells. Therefore in this study, we use uniform-degree edge swapping where the degree of a vertex is the number of incident cells as shown in Fig. 2.14. The total variance of degree for six is used as the criterion. The optimal variance is zero when all four vertices have six neighbours. The pseudo code for uniform degree swapping is shown in Algorithm 2.3. In Fig. 2.14, the variance before swapping is 4. After the swapping, all vertex degrees are six and the variance is 0, which is smaller than before swapping. Therefore the swap is performed.

2.2.2 Smoothing

A number of smoothing methods have been used to improve mesh quality. These approaches can be classified as local and global smoothing techniques. A local method adjusts the geometric location of one vertex at a time to achieve the optimal shape quality in a neighborhood. The overall mesh quality improves by applying the local smoothing for every adjustable vertex in the mesh. It is efficient if each local adjustment is inexpensive. The global smoothing method adjusts all vertex locations in the same time. It needs to solve an optimization problem as large as the number of vertices in a mesh, which is computationally expensive.

One of the commonly-used local smoothing techniques is Laplacian smoothing [11]. It relocates the free vertex to the arithmetic mean location of its incident vertices,

$$x_{\text{free}} = \frac{\sum_{i=0}^{i=N} x_i}{N} \qquad y_{\text{free}} = \frac{\sum_{i=0}^{i=N} y_i}{N}.$$

N refers to the number of incident vertices to the free vertex and x, y are the spatial coordinates. This method is computationally inexpensive, but it does not guarantee any improvement in the element quality. Actually, it is possible to produce inverted elements which have negative volume.

In GRUMMP, we use the optimization-based local smoothing technique proposed by Freitag et al. [12]. Optimization techniques use functions and their gradients to find the position where the functions obtain optimal values. Only a brief review of this method is shown here and the details can be found in Reference [13, 14]. To get a regular triangular mesh, we consider the optimal location for a free vertex to be where the minimum sine value is maximized. The challenge is that the function to be optimized as shown in Fig. 2.15,

$$\phi(x,y) = \min_{i \in S} f_i(x,y) = \min_{i \in S} \sin(\theta_i(x,y)),$$

is only piecewise smooth and therefore is not differentiable everywhere. In this case, a simple

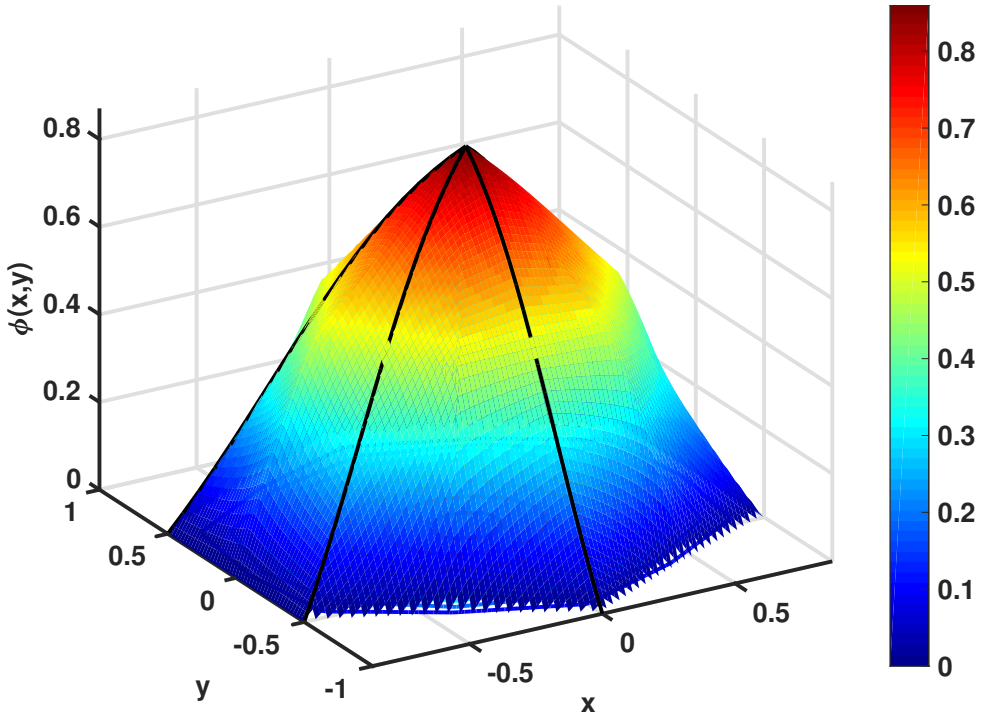


Figure 2.15: The minimum sine value $\phi(x,y)$

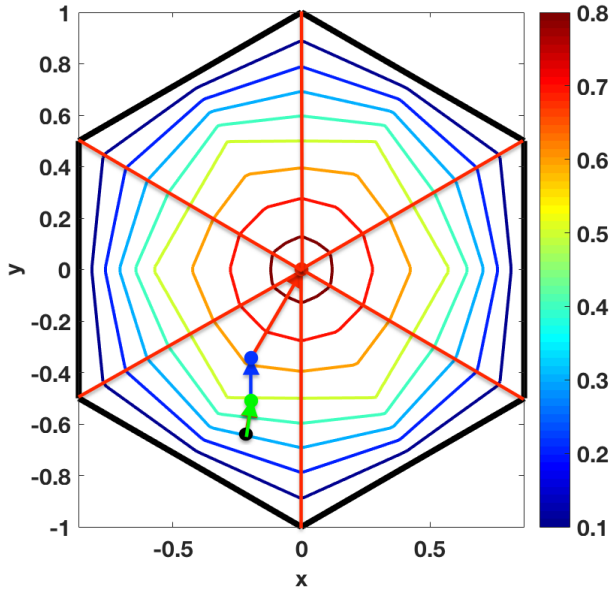
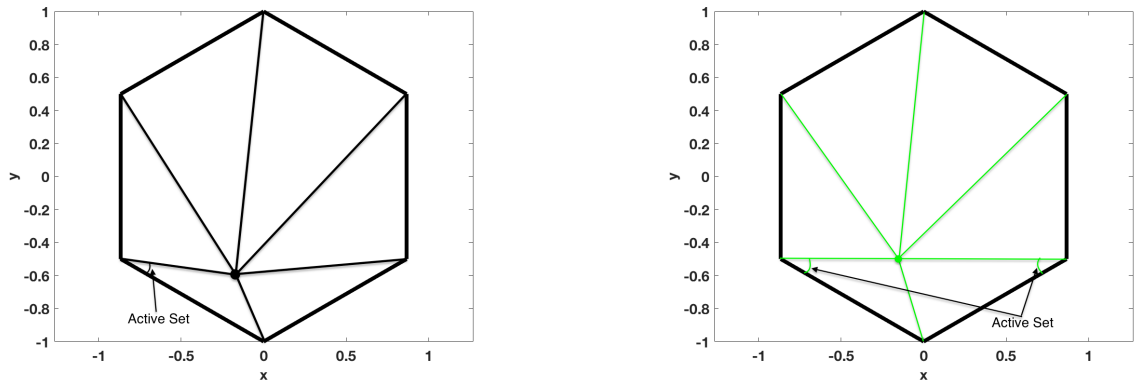


Figure 2.16: Contours of $\phi(x,y)$



(a) Starting location

(b) Two angles in the active set

Figure 2.17: The active set

regular hexagon topology as shown in Fig. 2.16 is used to facilitate the illustration. We denote the active set $\mathscr{A}$ as the set of minimum angles when the point is located at (x, y). For example, in Fig. 2.17a, only one angle is in the active set, in other words, there is only one angle has the smallest sine value in this case. The first search step attempts to find the location where more than one angles are in the active set as shown in Fig. 2.17b. For this simple case, the two active angles set the trajectory straight up. The next step stops where the active set changes. The approximate route is shown in Fig. 2.16.

2.3 Finite-volume solver

The finite-volume solver starts with the conservative form of the governing equations, which can be cast into a generic form [2]

$$\frac{\partial U}{\partial t} + \nabla \cdot F = S \tag{2.3}$$

where U denotes the solution vector, F the flux vector, and S the source term.

Integrating Eqn. 2.3 over an arbitrary control volume (CV) i and applying Gauss's theorem, we obtain

$$\iint_{CV_i} \frac{\partial U}{\partial t} dA + \oint_{\partial CV_i} F \cdot \hat{n} ds = \iint_{CV_i} S dA. \tag{2.4}$$

Assuming the discretized physical domain to be constant, Eqn. 2.4 can be further simplified as

$$\frac{d\overline{U_i}}{dt} + \frac{1}{A_i}\oint_{\partial CV_i} F \cdot \hat{n} ds = \frac{1}{A_i}\iint_{CV_i} S dA \tag{2.5}$$

where A_i denotes the area of the control volume, $\overline{U_i} = \frac{1}{A_i}\iint_{CV_i} U dA$ the control volume average, and $\hat{n}$ the outward unit normal vector.

The general finite-volume numerical method can be summarized in the following stages [21]:

1. Approximate $U(x,y,z)$ in each control volume with a polynomial $U_i(x,y,z) = U_i + \frac{\partial U_i}{\partial x}(x - x_i) + \frac{\partial U_i}{\partial y}(y - y_i) + \frac{\partial U_i}{\partial z}(z - z_i) + \cdots$. Given the value of $\overline{U_i}$ for each control volume, perform solution reconstruction to obtain the polynomial coefficients $(U_i, \frac{\partial U_i}{\partial x}, \frac{\partial U_i}{\partial y}, \frac{\partial U_i}{\partial z}, \cdots)$. Using this polynomial approximation of U, find U at the control volume boundary, and evaluate flux F at the boundary.

2. Since there is a distinct approximation in each control volume, two distinct values of the flux are generally obtained at the boundary between two control volumes. Apply some strategy for resolving the discontinuity in the flux at the control volume boundary to produce a single value of the flux.

3. Integrate the flux along the control volume boundary.

4. Advance the solution in time to obtain new values of $\overline{U}$.

In this section, the solution reconstruction is further discussed. The details of other algorithms such as flux evaluation and flux integration can be found in Reference [2, 29].

A two-dimensional solution reconstruction is used as an illustration. The reconstructed solution $\tilde{U}_i(x,y)$ is obtained by a Taylor expansion

$$\begin{aligned}\tilde{U}_i(x,y) = U|_i &+ \left.\frac{\partial U}{\partial x}\right|_i (x - x_i) + \left.\frac{\partial U}{\partial y}\right|_i (y - y_i) \\ &+ \left.\frac{\partial^2 U}{\partial x^2}\right|_i \frac{(x-x_i)^2}{2} + \left.\frac{\partial^2 U}{\partial x \partial y}\right|_i (x - x_i)(y - y_i) + \left.\frac{\partial^2 U}{\partial y^2}\right|_i \frac{(y-y_i)^2}{2} + \cdots\end{aligned} \tag{2.6}$$

where x_i and y_i denote the locations of the reference point of the control volume i.

Conservation of the mean within the control volume i requires that

$$\overline{U_i} = \iint_{CV_i} \tilde{U}_i(x,y)dA. \tag{2.7}$$

Substituting Eqn. 2.6 into Eqn. 2.7, we get that

$$\overline{U}_i = U|_i + \left.\frac{\partial U}{\partial x}\right|_i \bar{x}_i + \left.\frac{\partial U}{\partial y}\right|_i \bar{y}_i + \left.\frac{\partial^2 U}{\partial x^2}\right|_i \frac{\overline{x^2}_i}{2} + \left.\frac{\partial^2 U}{\partial x \partial y}\right|_i \overline{xy} + \left.\frac{\partial^2 U}{\partial y^2}\right|_i \frac{\overline{y^2}_i}{2} + \cdots \tag{2.8}$$

where the geometric moments are represented as

$$\overline{x^m y^n}_i = \frac{1}{A_i} \iint_{V_i} (x - x_i)^m (y - y_i)^n dA \tag{2.9}$$

Accuracy of the reconstruction is determined by the number of derivatives evaluated in the polynomial. To make it k-exact, or $(k+1)$-order accurate, the polynomial has to be degree k. For instance, a second-order accurate reconstruction,

$$\tilde{U}_i(x,y) = U|_i + \left.\frac{\partial U}{\partial x}\right|_i (x - x_i) + \left.\frac{\partial U}{\partial y}\right|_i (y - y_i) + O(\Delta x^2, \Delta y^2),$$

requires $U|_i$, $\left.\frac{\partial U}{\partial x}\right|_i$, and $\left.\frac{\partial U}{\partial y}\right|_i$ to be evaluated.

To compute these derivatives, we seek to minimize the error in predicting the mean value of the function for neighbouring control volumes. The mean value for the control volume CV_j is,

$$\frac{1}{A_j} \iint_{CV_j} \tilde{U}_i dA = U|_i + \left.\frac{\partial U}{\partial x}\right|_i \frac{1}{A_j} \int_{CV_j} (x - x_i) dA + \left.\frac{\partial U}{\partial y}\right|_i \frac{1}{A_j} \int_{CV_j} (y - y_i) dA + \cdots. \tag{2.10}$$

To avoid computing moments of each control volume, we replace $x - x_i$ with $x - x_j + x_j - x_i$

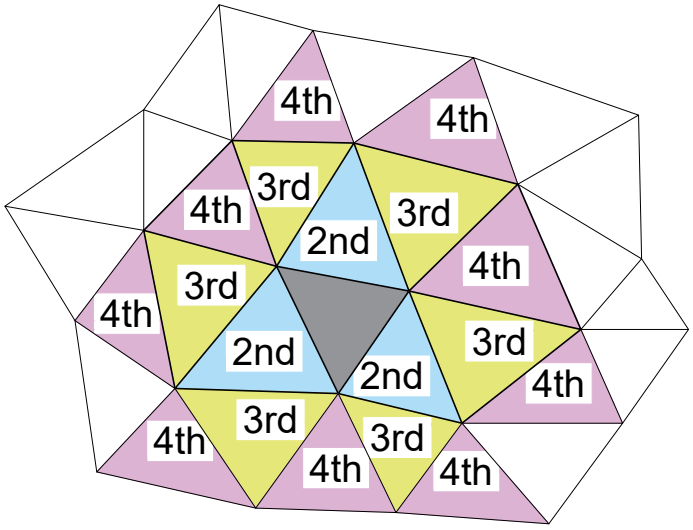


Figure 2.18: Neighboring CV stencil

and $y - y_i$ with $y - y_j + y_j - y_i$. Eqn. 2.10 can be represented as

$$\frac{1}{A_j}\iint_{CV_j} \tilde{U}_i dA = U|_i + \left.\frac{\partial U}{\partial x}\right|_i \hat{x}_{ij} + \left.\frac{\partial U}{\partial y}\right|_i \hat{y}_{ij} + \cdots \tag{2.11}$$

with

$$\begin{aligned}\widehat{x^m y^n}_{ij} &= \frac{1}{A_j}\iint_{CV_j} ((x - x_j) + (x_j - x_i))^m \cdot ((y - y_j) + (y_j - y_i))^n dA \\ &= \sum_{l=0}^{n}\sum_{k=0}^{m} \frac{n!}{l!(n-l)!}\frac{m!}{k!(m-k)!}(x_j - x_i)^k \cdot (y_j - y_i)^l \cdot \overline{x^{m-k}y^{n-l}}_j\end{aligned}$$

The minimum number of neighbouring control volumes in the stencil is equal to the number of derivatives to be evaluated. In practice, we choose three neighbors for the second-order accuracy and nine for the third-order, as shown in Fig. 2.18. The least-squares system for

evaluating derivatives is,

$$\left[\begin{array}{cccccc|c} 1 & \bar{x}_i & \bar{y}_i & \overline{x^2}_i & \overline{xy}_i & \overline{y^2}_i & \cdots \\ \hline w_{i1} & w_{i1}\widehat{x}_{i1} & w_{i1}\widehat{y}_{i1} & w_{i1}\widehat{x^2}_{i1} & w_{i1}\widehat{xy}_{i1} & w_{i1}\widehat{y^2}_{i1} & \cdots \\ w_{i2} & w_{i2}\widehat{x}_{i2} & w_{i2}\widehat{y}_{i2} & w_{i2}\widehat{x^2}_{i2} & w_{i2}\widehat{xy}_{i2} & w_{i2}\widehat{y^2}_{i2} & \cdots \\ w_{i3} & w_{i3}\widehat{x}_{i3} & w_{i3}\widehat{y}_{i3} & w_{i3}\widehat{x^2}_{i3} & w_{i3}\widehat{xy}_{i3} & w_{i3}\widehat{y^2}_{i3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ w_{iN} & w_{iN}\widehat{x}_{iN} & w_{iN}\widehat{y}_{iN} & w_{iN}\widehat{x^2}_{iN} & w_{iN}\widehat{xy}_{iN} & w_{iN}\widehat{y^2}_{iN} & \cdots \end{array}\right] \left(\begin{array}{c} U \\ \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \\ \frac{\partial^2 U}{2\partial x^2} \\ \frac{\partial^2 U}{\partial x \partial y} \\ \frac{\partial^2 U}{2\partial y^2} \\ \vdots \end{array}\right)_i = \left(\begin{array}{c|c} \bar{U}_i & \\ \hline w_{i1}\bar{U}_1 & \\ w_{i2}\bar{U}_2 & \\ w_{i2}\bar{U}_2 & \\ \vdots & \\ w_{iN}\bar{U}_N & \end{array}\right)$$

where geometric weights w_{ij} are used to specify the relative importance of good prediction for neighboring control volumes. The weights are based on the distance between control volume reference points, $w_{ij} = \frac{1}{|x_i - x_j|}$. The first row of this least-squares system is the mean constraint as in Eqn. 2.8. It is eliminated by Gaussian elimination resulting in an unconstrained least-squares system which is then solved for every control volume using the Singular-Value-Decomposition (SVD) method. After the derivatives are evaluated, the flux evaluation and integral [29, 36] are performed.

Chapter 3

Frontal-Delaunay triangulation

The regularity of the mesh obtained from the Delaunay triangulation is not satisfying enough. Even with the help of swapping and smoothing, there are still lots of triangles that are far from equilateral in the mesh, as shown in Fig. 3.1.

Another competing two-dimensional unstructured mesh generation scheme, the advancing-front method, begins with a discretization of boundaries. The boundary edges form the initial front. A particular edge from the front is selected as the base edge, and a new triangle is created with this base edge and a newly created vertex or an existing vertex. The new vertex is inserted to make an optimal triangle. After insertion, the base edge is removed from the front since it is obscured by the new triangle, and the newly generated edges are either added to or removed from the front based on their visibility. This procedure terminates when the front is closed. Fig. 3.2 gives a simple illustration of the advancing-front scheme. The front is marked by the solid lines in the figures. As shown in Fig. 3.2a, the first triangle is generated. Two newly generated edges are added to the front while the boundary edge on that triangle is removed from the front because it is no longer visible for other edges in the front. Fig. 3.2b shows the first case when the insertion is not valid. The optimal triangle to be generated is marked as red but the new edges are intersected with the existing ones. Therefore the scheme connects the red edge in Fig. 3.2a with the nearby existing vertex, resulting in the blue triangle. With the insertion continuing, the optimal triangle may require insertion of a vertex too close to an existing vertex as shown in Fig. 3.2c. If the optimal triangle is generated nevertheless, a very short edge will be the side-effect. Therefore, the nearby existing vertex rather than the optimal vertex is connected with the base edge, forming the blue triangle in Fig. 3.2d. The advancing-

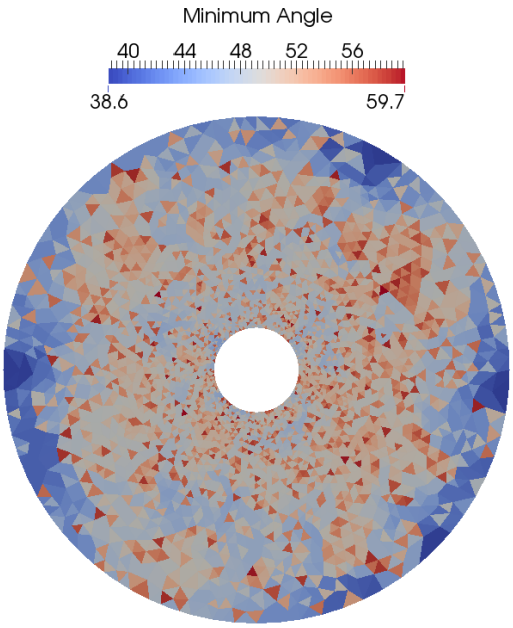
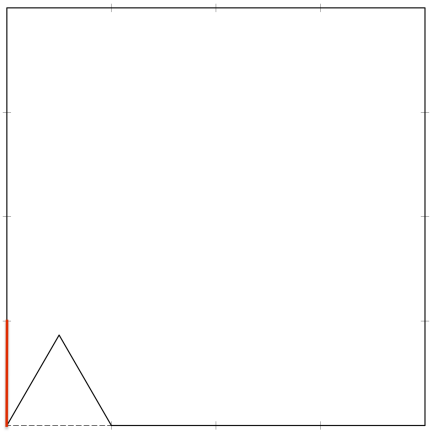


Figure 3.1: Delaunay triangulation

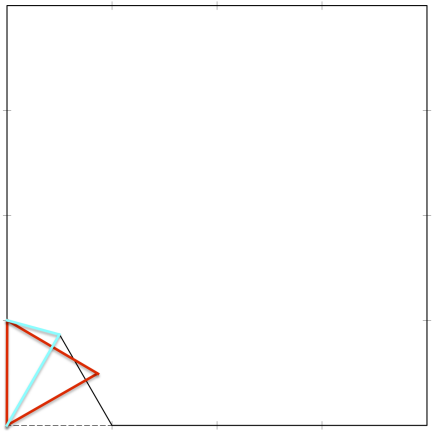
front scheme terminates when the front is empty. Fig. 3.2e shows what the front (solid lines) looks like after the final insertion. If we choose any edge from the front, the new optimal triangle will either intersect with or be too close to the existing triangles. Therefore, the blue edge in Fig. 3.2f is applied to close the front.

Frontal-Delaunay algorithms [31, 24, 25] are the hybridization of Delaunay-refinement and advancing-front techniques, in which the Delaunay triangulation is used to define the topology of a mesh while new points are inserted in a manner consistent with the advancing-front method. Engwirda's frontal-Delaunay method [10, 9] is applied in this study because it improves the mesh quality compared to the Delaunay triangulation, as shown in Fig. 3.3.

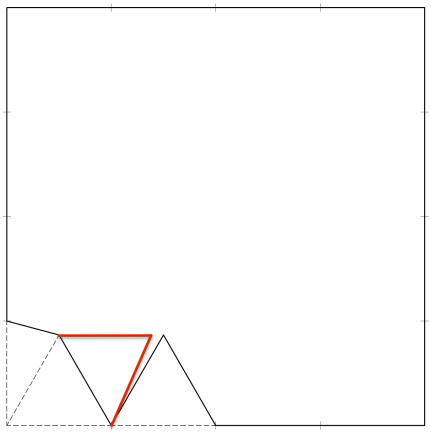
Engwirda's frontal-Delaunay method is similar in execution to the classic Delaunay refinement algorithm, but uses different queuing and point placement strategies. The priority queue for bad triangles is sorted by radius-edge ratio, the shape quality (Eqn. 2.2) introduced in Delaunay-refinement triangulation. Only bad triangles whose shortest edges are smaller than 1.5 times the local feature sizes are added to the queue. This constraint ensures refine-



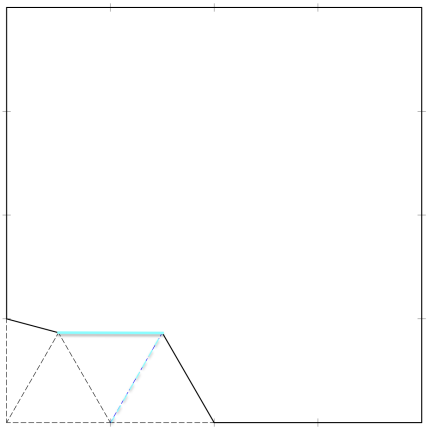
(a) After the first insertion



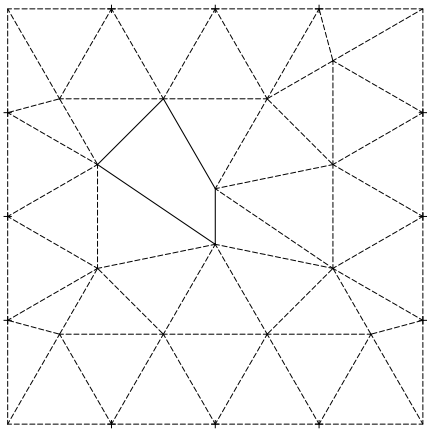
(b) Rejection due to intersections



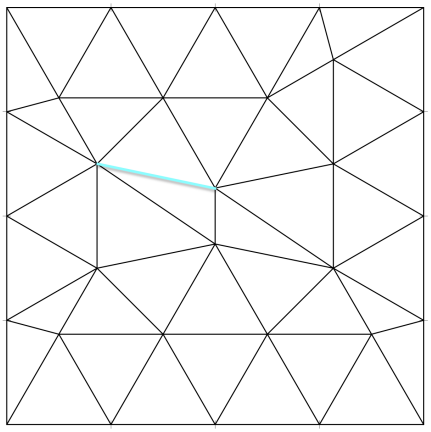
(c) Rejection due to being too close to existing vertex



(d) Connecting existing vertex



(e) The front after the last insertion



(f) Final mesh

Figure 3.2: Advancing-front illustration

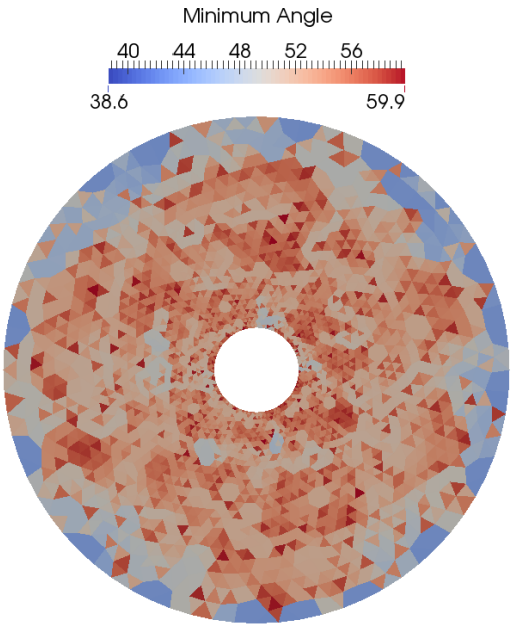


Figure 3.3: Engwirda's frontal-Delaunay triangulation

ment occurs in a frontal fashion, because the triangle to be refined is guaranteed to be adjacent to a boundary or to a triangle that meets the quality criterion. Two new types of vertices to be inserted are used by Engwirda: the shape-optimal vertex and the size-optimal vertex. Both are off-centers proposed by Üngor [35], which are located on the Voronoi diagram of the corresponding Delaunay triangulation.

The shape-optimal vertex c', as shown in Fig. 3.4, is placed so that $\triangle ABc'$ is the largest isosceles triangle that satisfies the bound on the minimum allowable angle $\alpha_{\min}$. c denotes the circumcenter and the gray line denotes the Voronoi edge associated with the base edge (the shortest edge of the bad triangle). The altitude of the new vertex to the base edge is calculated as

$$h' = \frac{||e_{\min}||}{2} \frac{1}{\tan(\alpha_{\min}/2)},$$

where $||e_{\min}||$ denotes the minimum edge length. By positioning the new vertex at c', the newly generated triangle just satisfies the minimum shape constraint.

The size-optimal vertex c'' generates a triangle satisfying the local size constraint. The

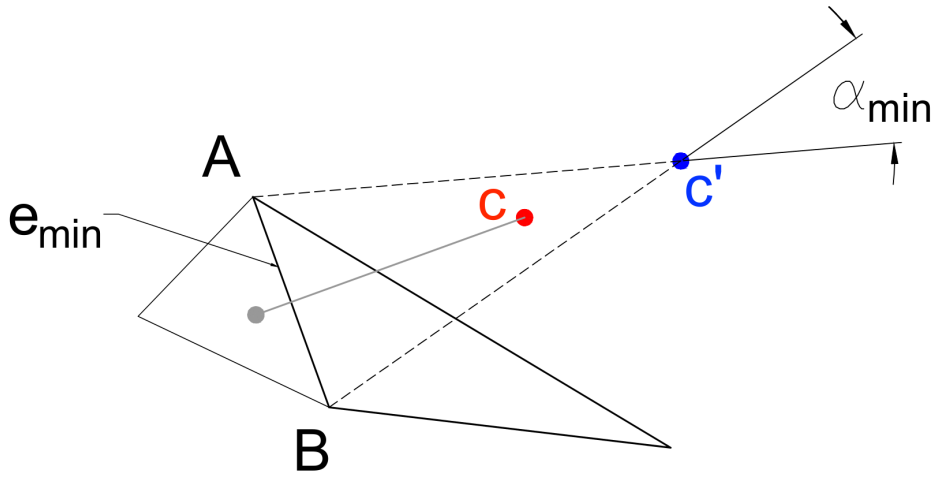


Figure 3.4: shape-optimal vertex

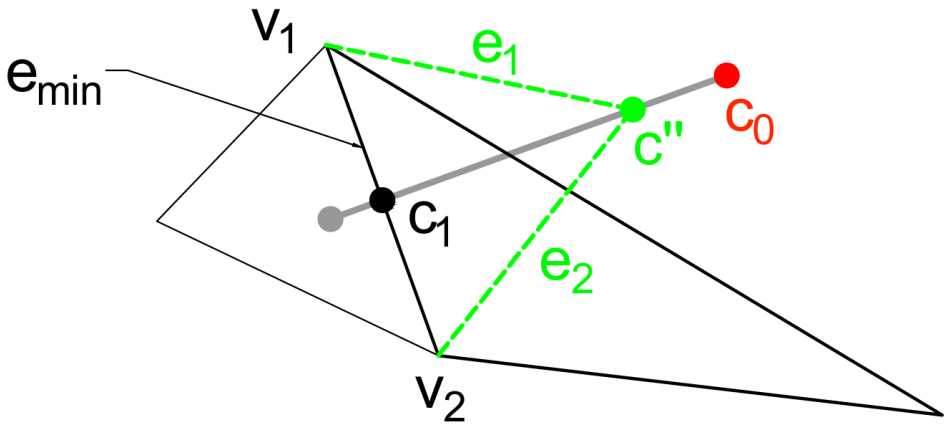


Figure 3.5: Size-optimal vertex

queuing algorithm of Engwirda's frontal-Delaunay scheme guarantees that only the triangles with their shortest edges satisfying size constraints are considered. Therefore, when calculating the position of c'', only the two newly created edges need to be compared with the local feature sizes. An iterative bisection method is used to find the position of the new vertex. Take edge e_1 in Fig. 3.5 as an example. Pick the circumcenter and the middle point of the base edge as the starting points c_0, c_1. Then calculate the local length scales of these starting points by Eqn. 2.1. Define

$$S_i = ||e_1|| - \frac{1}{2}(\text{lfs}(v_1) - \text{lfs}(c_i)).$$

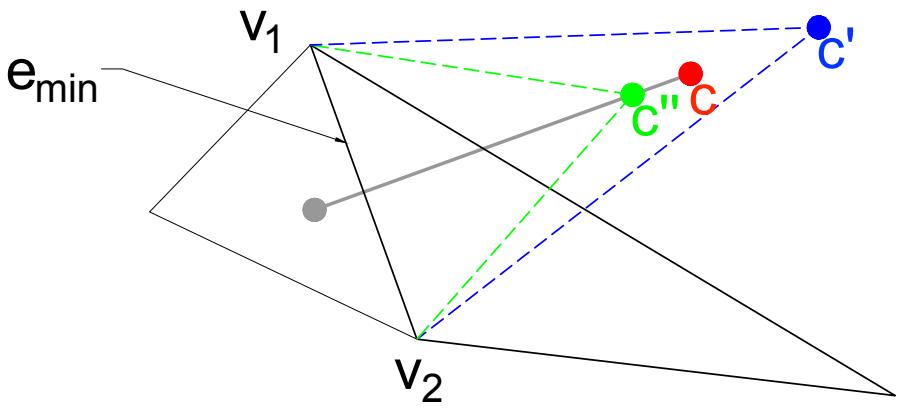


Figure 3.6: Choosing between the three kinds of vertices

If S_0 and S_1 have the same sign, the size-optimal point is farther than the circumcenter, and its location is irrelevant as we will see. Otherwise choose the middle point of edge $\overline{c_0c_1}$ as c_2. Determine S_2 and choose the interval $[c_0, c_2]$ or $[c_2, c_1]$ so that there is a zero inside the interval. Iterate this procedure until $S_n < 10^{-4}||e_{\min}||$. Next, repeat this algorithm for the edge e_2. Choose the point closer to the base edge between this two vertices as the size-optimal vertex c''. All three edges in the newly generated triangle $\triangle v_1 v_2 c''$ will satisfy the size constraint.

Overall, we compare three candidate vertices: the circumcenter c, the shape-optimal vertex c', and the size-optimal vertex c''. Each of them will form a new triangle with the same shortest edge $e_{\min}$ as the bad triangle. Of c, c', and c'', the one closest to the shortest edge is chosen for insertion. Fig. 3.6 shows an example of which vertex should be inserted. In this case, size-optimal vertex c'' is closer to the shortest edge than shape-optimal c'. Therefore, by choosing c'' as the off-center, both the shape and size constraints are satisfied. Since the off-center has to be on the Voronoi edge, the circumcenter is chosen if it is closer to the base edge than the off-center. Otherwise, one of the off-centers is chosen, which is the case in Fig. 3.6.

With the Steiner vertex calculation method in mind, the basic algorithm of Engwirda's frontal-Delaunay method is shown in Algorithm 3.1. Rejection for a vertex too close to existing ones in original advancing-front scheme is not required here. A proper quality bound is sufficient to deal with this problem. In this techniques, if an unwanted small triangle is generated, it will be regared as good sized triangle as in Algorithm 3.3. It will not be added to the queue. Even if it is a badly shaped triangle, usually the next splitting will make it pass the quality test with the help of a carefully chosen Steiner vertex. Therefore a vertex inserted close to existing ones is permitted and will not blow up the algorithm globally.

Algorithm 3.1 Engwirda's frontal-Delaunay scheme

```
1:  Starting with a constrained Delaunay triangulation
2:  for Every triangle in the mesh do
3:      Queue triangle
4:  end for
5:  while The queue is not empty do
6:      Take the top triangle in the queue
7:      if Triangle is not in the mesh then
8:          Continue
9:      else
10:         Calculate the location of the new vertex
11:         Use Watson insertion to insert the new vertex
12:         Queue the newly generated triangles
13:     end if
14: end while
```

Algorithm 3.2 Engwirda's frontal-Delaunay Queuing scheme

```
1:  procedure QUEUE(Triangle)
2:      if QualityTest(Triangle) == BAD then
3:          Find the minimum edge length of that triangle
4:          if Minimum edge length < 1.5× local feature size then
5:              Calculate the normalized shape quality
6:              Add the triangle into the queue with shape quality as priority
7:          end if
8:      end if
9:  end procedure
```

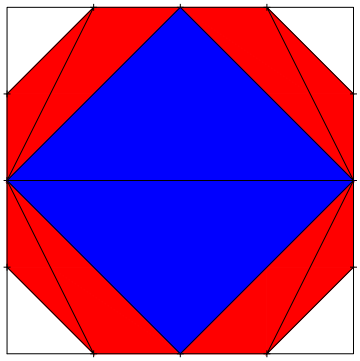
A simple example is used to illustrate the procedure of Engwirda's frontal-Delaunay method. The geometry is a 20×20 square, we set the local feature size to 5 everywhere at the begining.

Algorithm 3.3 Quality Test

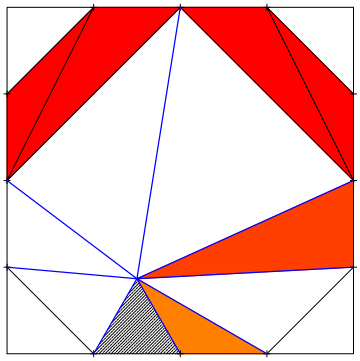
```
1:  procedure QUALITY TEST(Triangle)
2:      if Average length of three edges on that triangle > 1.5× local feature size then
3:          BadSizeTriangle
4:      else
5:          GoodSizeTriangle
6:      end if
7:      if Shape quality of the triangle < minimum allowable value then
8:          BadShapeTriangle
9:      else
10:         GoodShapeTriangle
11:     end if
12:     if BadSizeTriangle or BadShapeTriangle then
13:         return BAD
14:     else
15:         return GOOD
16:     end if
17: end procedure
```

The minimum angle allowed in the mesh is 33°. Therefore, in this mesh, any triangle with minimum angle smaller than 33° or average edge length greater than 7.5 would be regarded as a bad triangle. Also, only the triangles whose minimum edge lengths are smaller than 7.5 are allowed to be added to the queue. To simplify the illustration, the size-optimal vertex is always chosen as the insertion point, which means the newly generated triangles are equilateral triangles. The refinement scheme starts with a constrained Delaunay triangulation as shown in Fig. 2.10. All the boundary edge lengths are 5. Several triangles in the mesh are colored red or blue. Red triangles denote that they are badly shaped triangles while blue triangles denote they are badly sized triangles. At this stage, only the red triangles are added into the queue because the minimum edge lengths of the blue triangles are too large to be queued. All the red triangles have the same shape quality. They are all on the top of the queue. In the next step, any triangle could be the one to be refined. In Fig. 3.7b, the hatched triangle is the newly generated one. Watson insertion is applied. After connecting the new vertex to all the edges on the empty hull, several new triangles are generated. Only triangles in the updated queue are shown in the figure. Darker red indicates it has worse shape quality, therefore it has higher priority to be

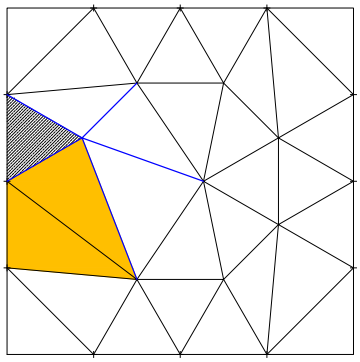
refined. After several insertion, we come to a stage where there are only a few triangles in the queue, as shown in Fig. 3.7c. After the last insertion, as shown in Fig. 3.7d, all the triangles in the mesh satisfy both the size constraint and shape constraint. The queue is empty and the scheme terminates. There are still several triangles that are far from equilateral. Increasing the quality bound will help (Fig, 3.7e). However, it comes with the risk of generating unnecessary small cells or even termination problems, as shown in Fig, 3.7f.



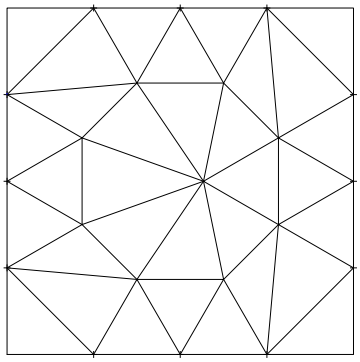
(a) Constrained Delaunay triangulation



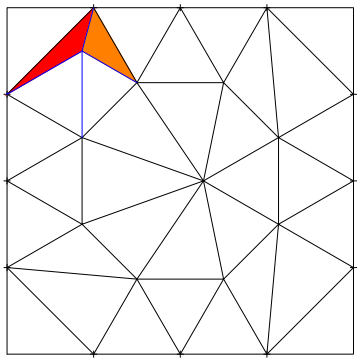
(b) After the first insertion



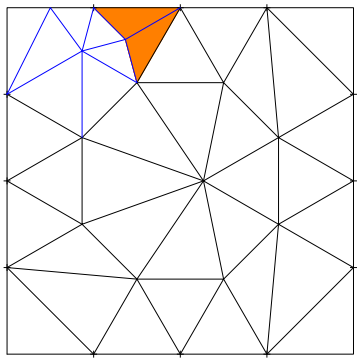
(c) The mesh just before final insertion



(d) Final mesh



(e) Increase the quality bound



(f) Generating unnecessary small triangles

Figure 3.7: Illustration of Engwirda's frontal-Delauany method

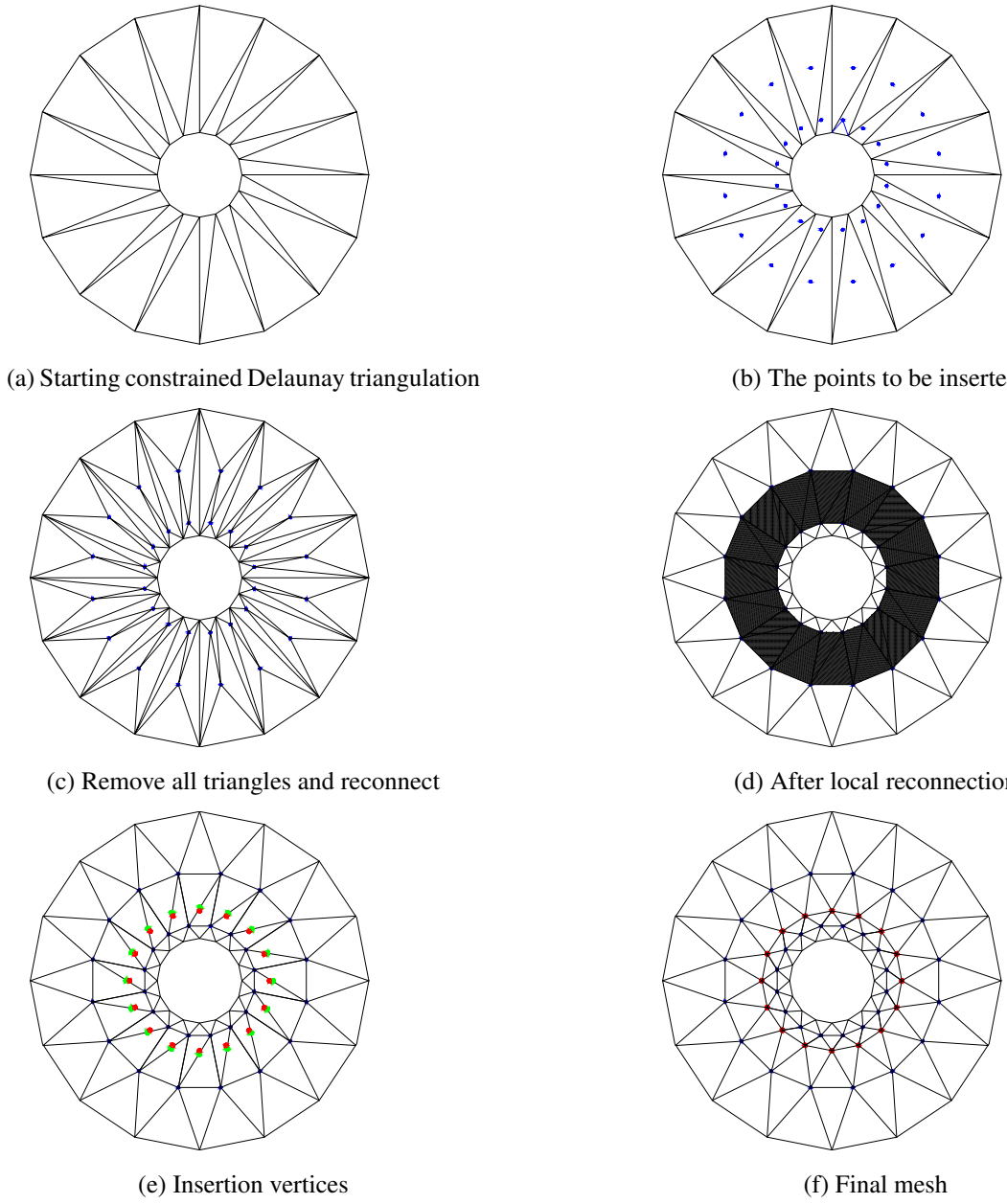
Chapter 4

Advancing-front local reconnection method

Marcum's advancing-front local reconnection (AFLR) method is also proven to generate high quality meshes [22]. In this chapter, a similar annulus domain as the one in Marcum's paper is used as an example to illustrate this AFLR scheme. The inner circle is radius 1 while the outer circle is radius 4. The circles are divided into 16 edges, as shown in Fig. 4.1a. First, the local feature sizes (*lfs*) are calculated. To simplify the illustration, all the vertices on the inner circle have the same *lfs* $2\pi/16 \approx 0.4$. The *lfs* on the outer circle is $8\pi/16 \approx 1.6$. The gradient parameter g is set to 2.5 to make the *lfs* equation (Eqn. 2.1) valid. Set the center of both circles at $(0,0)$, then the *lfs* in this domain can be simplified as

$$lfs(r,\theta) = 0.4 + \tfrac{r-1}{2.5}, \quad 1 \le r \le 4 \ .$$

In Marcum's scheme, the *active*/*off* flags are applied to all the triangles to indicate whether they are good or bad in a size criterion. If the length of each edge on a triangle is less than the 1.5 times average length scale of the two end vertices of that edge, the triangle is regarded as good triangle and applied the *off* flag. The *off* flag means that this triangle is turned off and currently not to be refined. Otherwise, the *active* flag is applied. As shown in Fig, 4.1a, all the boundary edges are exactly equal to the average length scales. As for the edges connecting two boundaries, the allowable lengths are $1.5 \times \frac{(1.6+0.4)}{2} = 1.5$. Since the radius difference of the two circles is 3, all these edges fail the size quality test. Therefore all the triangles in constrained Delaunay triangulation are marked as *active*. The insertion points are calculated



(a) Starting constrained Delaunay triangulation

(b) The points to be inserted

(c) Remove all triangles and reconnect

(d) After local reconnection

(e) Insertion vertices

(f) Final mesh

Figure 4.1: Marcum's original scheme

for all the *active* triangles. In this scheme, the new point will form a equilateral triangle with the shortest edge of the *active* triangle. In Fig. 4.1b, all the new vertices are shown in blue dots. Then, the triangles who contains the new vertices need to be removed and reconnected with the new vertices, which means every triangle for this case. Fig. 4.1c shows the mesh after the first insertion. A local reconnection scheme is applied to maximize the minimum angles in the mesh, as shown in Fig. 4.1d. The *active/off* flags are applied to the new triangles. All the *active* triangles are shadowed in Fig. 4.1d. All the vertices to be inserted are calculated once again. In Fig. 4.1e, the green dots are generated from edges closer to the center while red dots are from the outer circle. Some of these are too close to each other. Therefore, a rejection test is applied. In Marcum's scheme, if the distance between two vertices is smaller than 0.7 times the larger *lfs* between the two, the vertex with larger *lfs* is rejected. In Fig. 4.1e, all the green vertices are rejected from the mesh. After the local reconnection, the final mesh is shown in Fig. 4.1f.

Because of the difficulty of implementing Marcum's method into GRUMMP's framework, a few necessary modifications are made. It gives comparable results as Marcum's algorithm, as shown in Fig. 4.2 . Instead of only considering the size constraint in Marcum's scheme, I also take the shape quality (Eqn. 2.2) into consideration. A triangle is set to be *off*, or in other words, marked as a good triangle only if it satisfies both constraints. To be consistent with the previous two algorithms, the modified algorithm still uses Watson's insertion to improve the local topology. The front and *active/off* flags, or quality tests, are updated after each insertion, rather than updating after all the points are inserted into the active elements as introduced in Marcum's scheme. A new denotation, layer index, is introduced to make sure the refinement is marching layer-wise as in Marcum's method. The layer index indicates which layer the edge is on. Starting with the constrained Delaunay mesh, the inner boundary edges are marked as layer index 0 and all other edges are marked as layer index ∞, as shown in Fig. 4.3a. When using Watson insertion, an empty hull (Fig. 4.3b) is obtained. The minimum layer index is found among all the edges on the hull. In this circumstance, it is 0. After the insertion, the layer

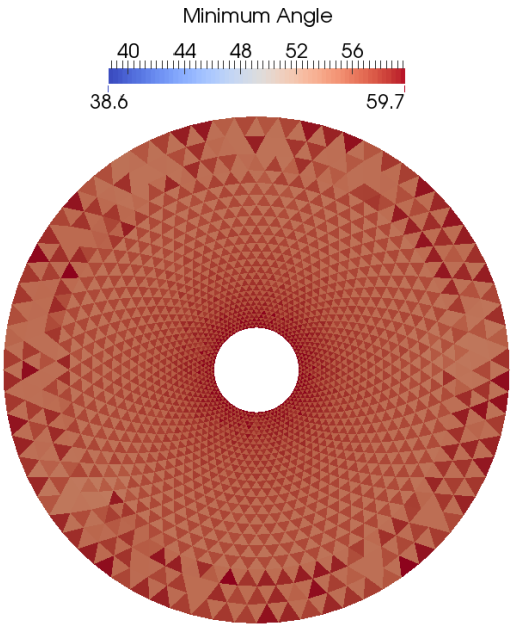
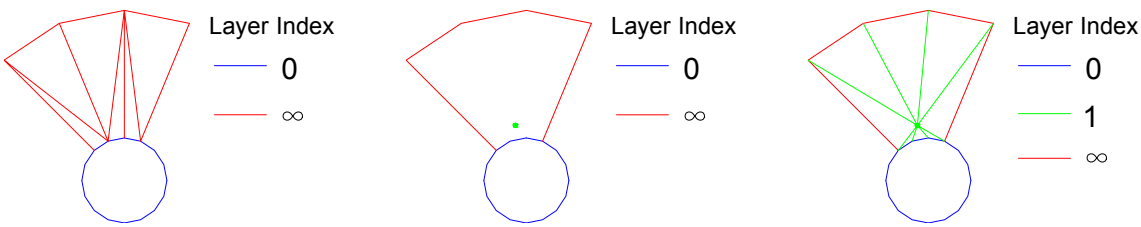


Figure 4.2: AFLR mesh

index of all newly created edges is *the minimum layer index* $+1$. As shown in Fig. 4.3c, the newly inserted edges are all marked as layer index 1. The local reconnection is omitted in our method because we are using Watson's insertion, which guarantees a local max-min (maximize the minimum angles) criterion. The elements in the queue are edges instead of triangles as in the other schemes. The queue priority scheme will be discussed later.

Another big difference from Marcum's scheme is the calculator for vertices to be inserted. In Fig. 4.4, c, c', and c'' are the circumcentre, the shape-optimal vertex and the size-optimal



(a) Layer index before the insertion

(b) Empty hull

(c) Layer index after the insertion

Figure 4.3: Layer index

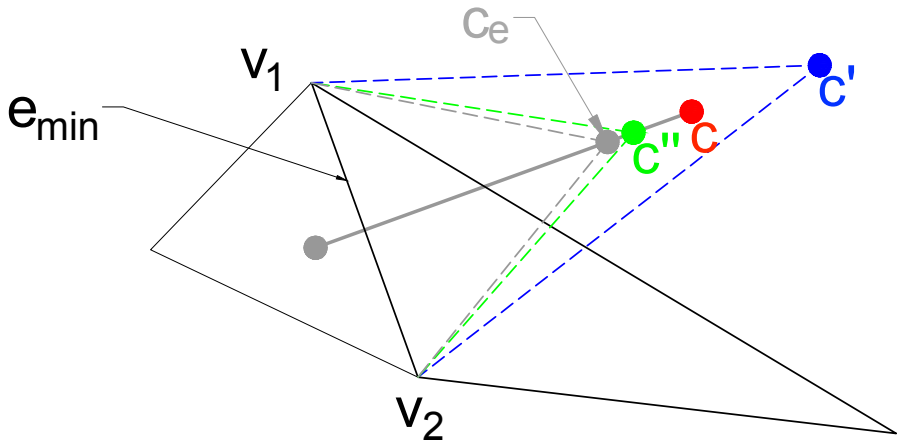


Figure 4.4: Four potential vertices

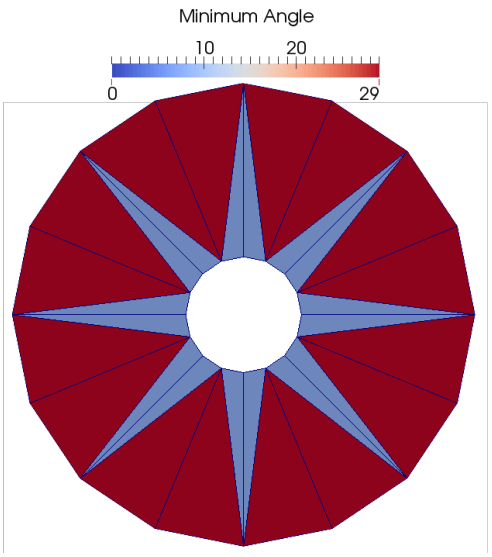
vertex respectively, as discussed in Chapter 3. c_e denotes the vertex generating an equilateral triangle with shortest edge $e_{\min}$. The vertex nearest to $e_{\min}$ among the four vertices is chosen for insertion.

The brief procedure of the modified AFLR scheme is illustrated by the same annulus example. In this case, the inner circle is set as layer index 0 at the beginning. Other edges in the mesh are set as layer index ∞. The triangles in Fig. 4.5 are colored by the layer index. Since the layer index is only applied to edges, the cell layer index is introduced as:

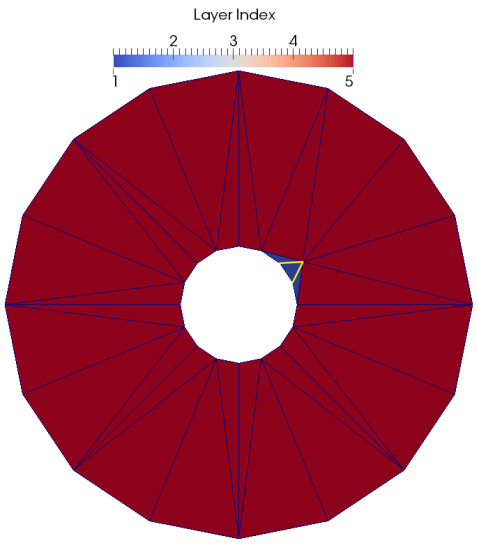
$$\text{Cell Layer Index} = \max_{i=1}^{3}\left(\mathit{LayerIndex}(e_i)\right) + 0.1 \times \min_{i=1}^{3}\left(\mathit{LayerIndex}(e_i)\right). \tag{4.1}$$

This is shown in Fig. 4.5b. After the first insertion, the newly generated triangle has two new edges with layer index 1 and a layer index 0 boundary edge. Therefore, its cell layer index is $1+0.1\times 0 = 1$.

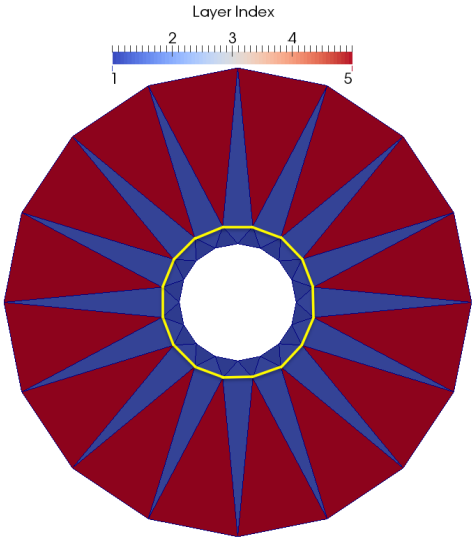
The general procedure is outlined as follows:



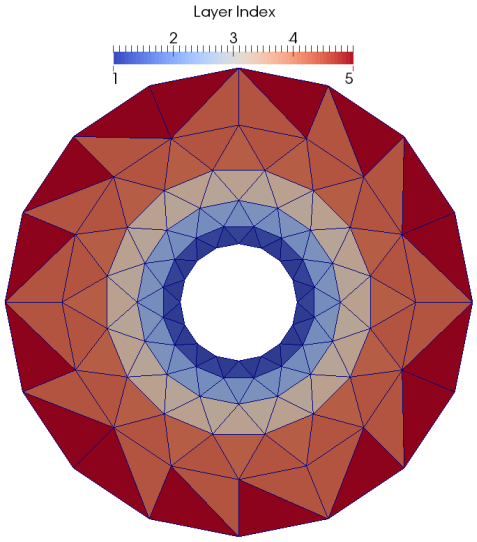
(a) Constrained Delaunay triangulation



(b) The first insertion



(c) First layer finished



(d) Final mesh

Figure 4.5: Illustration of modified AFLR scheme

1. Start with the constrained Delaunay triangulation (Fig. 4.5a).

2. Initialize the data structure. For each triangle, if its each edge length is smaller than 1.5 times the averaged lengthscale and its radius-edge ratio is larger than the threshold value, this triangle is treated as a good triangle. Otherwise, it is treated as bad.

   Example In the annulus case, all the triangles are bad. In Fig. 4.5a, the blue triangles have minimum angles less than 29°, which is the threshold value in this case. Besides that, all the triangles fail the size constraint test as discussed in Marcum's original scheme.

3. Queue boundary edges with bad adjacent triangles and interior edges which have a good adjacent cell on one side and a bad adjacent cell on the other side.

   Example In Fig. 4.5a, only the boundary edges are queued because they all have bad adjacent triangles. All the interior edges are shared by bad triangles, therefore they are not queued. In Fig. 4.5b, after the first insertion, a good triangle is generated. The boundary edge on it is removed from the queue. Then the two newly generated interior edges are queued because now they have a good triangle on one side and a bad one on the other side.

4. The queue priority is calculated by the sum of layer index, edge length, and shape quality of the adjacent bad triangle. The smaller the sum is, the higher priority the edge has.

   Example

$$\text{Priority} = \text{Layer Index} + ||e|| + \frac{||e_{\min}||}{\sqrt{3}r}$$

   r denotes the circumradius of the adjacent bad triangle. The layer index is an integer ranging in $[0,\infty]$. The shape quality $\frac{||e_{\min}||}{\sqrt{3}r}$ ranges in $[0,1]$ and $||e||$ is usually much smaller than 1 (not in this example). The layer index is dominant in priority. Therefore when comparing two similar edges, the one with a lower layer index has

a higher priority. This ensures the scheme works in a layer-wise marching pattern. When the layer index is the same, the shape quality is dominant. This means that when comparing two edges on the same layer, the one adjacent to a worse triangle has a higher priority, which is consistent to Engwirda's frontal-Delaunay scheme that the worst shaped triangle is refined first. If these two values are exactly the same, the shorter edge jumps ahead. The small edge or triangle may locate in the area of interest, where we want regular triangles.

5. Choose the first edge from the queue and find the bad triangle adjacent to it. Calculate four kinds of points: the size-optimal vertex and shape-optimal vertex mentioned in the previous algorithm, the point generating an exact equilateral triangle and the circumcenter. Choose the point that is nearest to the base edge.

   Example This is illustrated in Fig. 4.4.

6. Reject the candidate point if the insertion generates an edge shorter than 0.7 times the base edge length.

   Example This rejection is inherited from Marcum's original scheme, preventing unnecessary small triangles being generated.

7. Use Watson insertion to insert the new point. The new edges created by the insertion are added to the queue if they satisfy the criteria in step 3.

8. Remove the edge from the queue regardless of whether the point is rejected or accepted.

9. Repeat steps 5-8 until the queue is empty.

   Example Take the same annulus case, thanks to the layer index dominant priority, all the boundary edges on the inner circle has the highest priority. The first layer is generated based on boundaries as shown in yellow segments in Fig. 4.5c. In this

circumstance, these edges form a slightly larger circle, or front. They are queued because sharing by good triangles and bad ones. Those edges between the inner circle and these yellow fronts are removed from the queue because they have good triangles on both sides. After layer-wise insertions, the scheme terminates under conditions like those in Fig. 4.5d. All triangles in the mesh are good ones, which also means all the edges are adjacent to good triangles. In other words, no more edges are queued.

Chapter 5

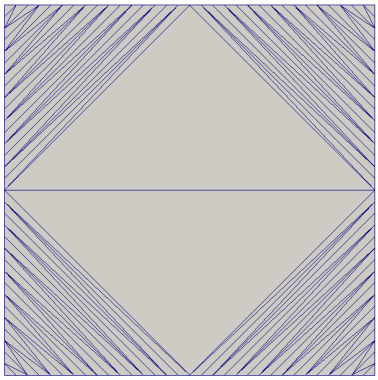
Numerical results

The mesh regularity, truncation error, discretization error, and time costs for several numerical test cases are discussed in this chapter to answer the questions posed in Section 1.1.

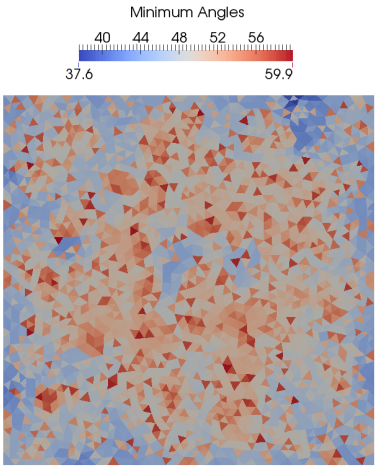
5.1 Poisson equation on a square domain

5.1.1 Mesh regularity

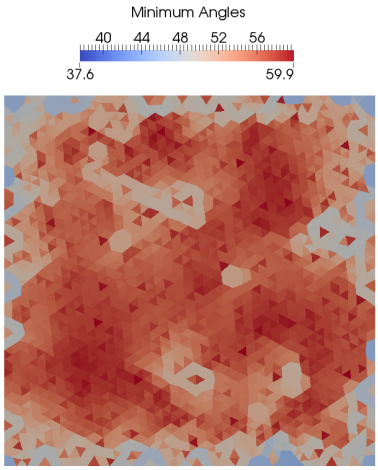
Three different mesh refinement schemes are applied to a square domain, of which the edge length is 1. Starting from the same constrained Delaunay triangulation, shown in Fig. 5.1a, the three schemes generate different meshes. Delauany refinement, as shown in Fig. 5.1b, does not generate any obvious chunks of equilateral triangles. In Fig. 5.1c, Engwirda's algorithm uses a different priority queue and different insertion points from Delaunay refinement. Only the triangles whose shortest edges satisfy the local length scales can be added to the queue. This ensures the newly created triangles are based on the existing good triangles. As shown in Fig. 5.2, Engwirda's scheme introduces a "snake" pattern. It turns into the area where the worst triangles exist. It generates several large chunks of equilateral triangles in the middle. However, badly shaped triangles emerge when the scheme tries to fill the gaps between these chunks because they are not perfectly aligned. Marcum's AFLR scheme refines the mesh in a different way. It starts from the boundary and marches into the interior layer by layer. As shown in Fig. 5.3, each layer consists of nearly equilateral triangles along the boundaries and a few badly shaped ones at the corners. In the end, the mesh is almost perfect except the cross shown in Fig. 5.1d.



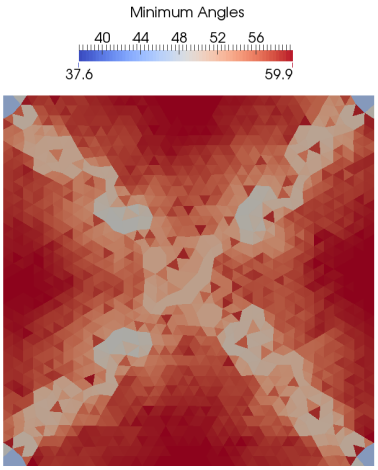
(a) Starting Delaunay Triangulation



(b) Delaunay Refinement

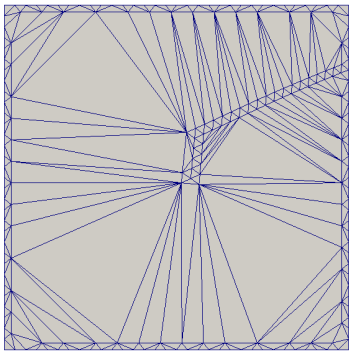


(c) Engwirda frontal-Delaunay

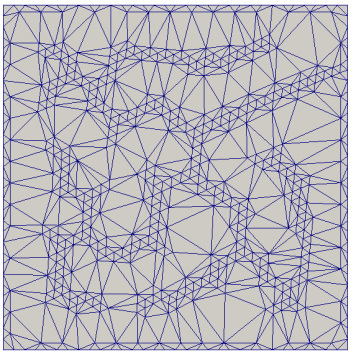


(d) Marcum's AFLR

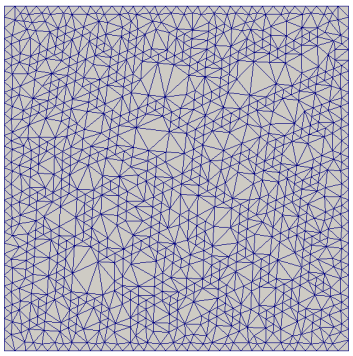
Figure 5.1: Meshes generated on a square domain



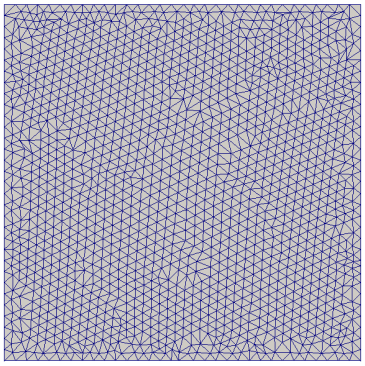
(a) After 100 insertions



(b) After 500 insertions

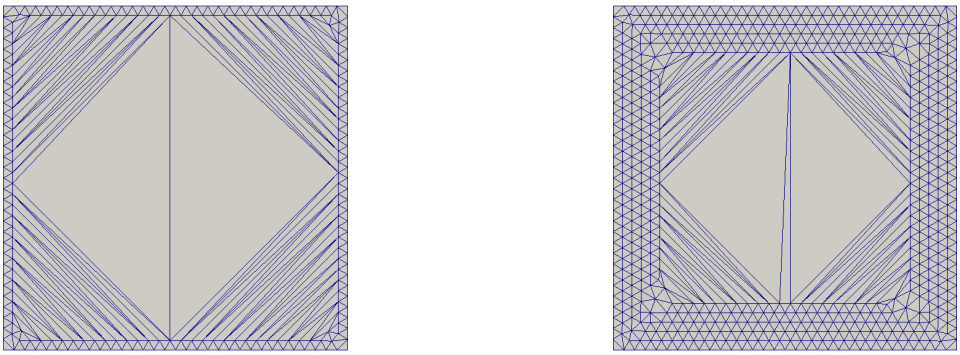


(c) After 1000 insertions



(d) After 1500 insertions

Figure 5.2: The process of Engwirda's scheme



(a) After the first layer (b) After the fifth layer

Figure 5.3: The process of AFLR scheme

Delaunay refinement uses circumcenters as Steiner points. The badly sized triangles are refined first. Inserting a circumcenter into a badly sized triangle does not guarantee the new triangle satisfies the shape constraint. Therefore, a bad triangle may be refined several times to satisfy the size and shape criterions. Because both the AFLR scheme and Engwirda's frontal-Delaunay scheme select Steiner points more cleverly, the newly created triangle will satisfy the size and shape constraints. Generally speaking, the sizes of meshes obtained from AFLR scheme and frontal-Delaunay scheme are smaller than those from Delaunay refinement under the same constraints. They both improve the regularity of the meshes, but which scheme generates the most equilateral triangles for this case? As illustrated in Fig. 5.4, the AFLR mesh has about 70 percent of all the angles between 55° and 65°, while Engwirda's frontal-Delaunay mesh is only 5 percent behind. The traditional Delauany method generates the worst mesh for this regularity comparison.

5.1.2 Truncation error and discretization error

The first test case for the relationship between errors and mesh regularity is solving the Poisson equation,

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = S, \tag{5.1}$$

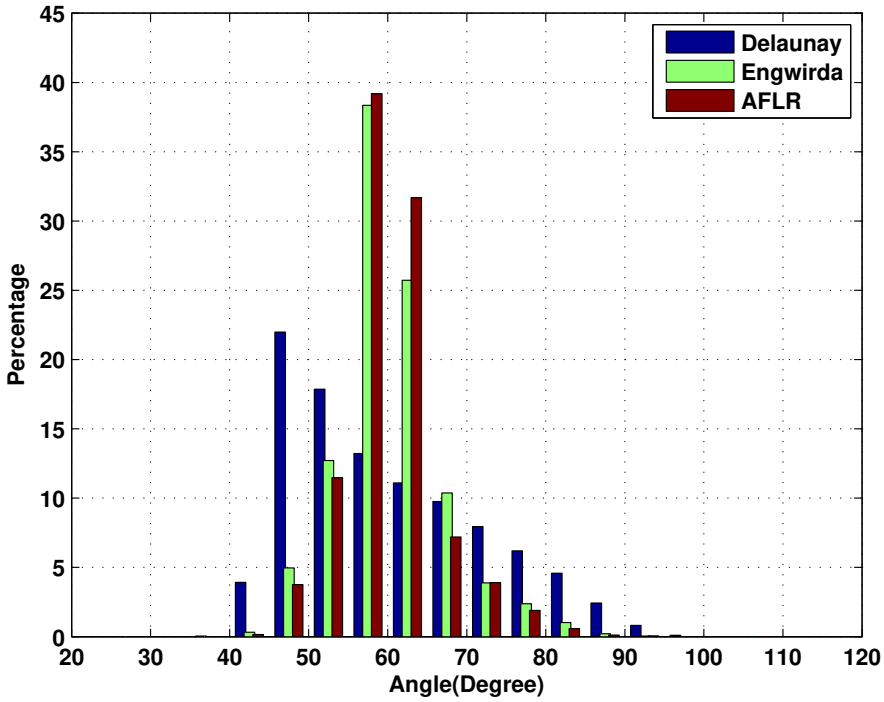


Figure 5.4: Angle distribution for meshes on square domain

on a square domain. Substitute the manufactured solution shown in Fig. 5.5,

$$T = \pi \frac{\sin(\pi x)\sin(\pi y)}{8},$$

into the Poisson equation 5.1, then the source term S can be obtained easily,

$$S = -\pi^3 \frac{\sin(\pi x)\sin(\pi y)}{4}.$$

A second order finite-volume discretization and least-squares reconstruction are used[36, 15]. Setting the control volume averages of the manufactured solution as the initial condition, the flux integral at the first step,

$$\frac{1}{A_{\mathrm{CV}}}\left[-\int_{\mathrm{CV}} S dA + \oint_{\partial \mathrm{CV}} (\nabla T)\cdot \vec{n} ds\right],$$

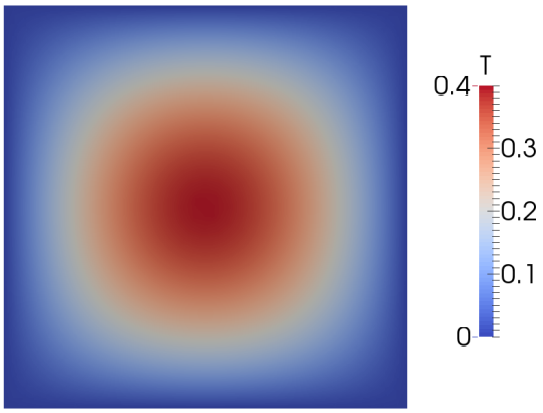
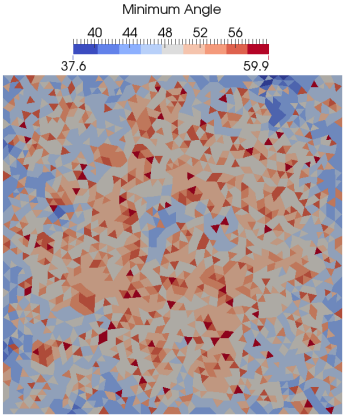
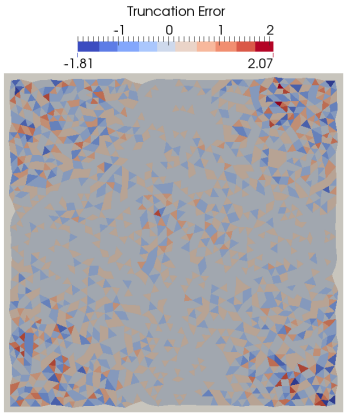


Figure 5.5: Sinusoidal manufactured solution of the Poisson equation

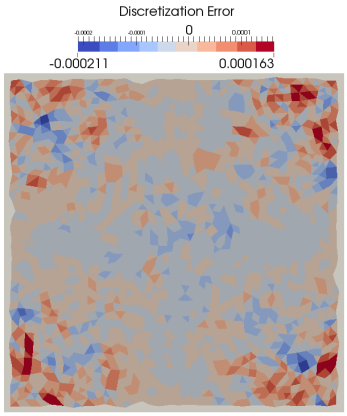
is the truncation error. The converged numerical solution is used to compute the discretization error with the control volume averages of the manufactured solution. In Fig. 5.6, we can observe that in the areas consisting of nearly equilateral triangles, the magnitudes of truncation error and discretization error tend to be very near zero, relative to the error where the mesh is less regular. Even in the Delaunay mesh, where most triangles are far from equilateral ones, we can still find a few well shaped control volumes which have very small error magnitudes. In the meshes obtained from Engwirda's frontal-Delaunay and Marcum's AFLR schemes, this pattern is much more obvious. The areas marked as red in the left-most figures in Fig. 5.6 indicate chunks of nearly equilateral triangles. In the same areas, we can expect nearly zero truncation errors. However, the discretization error seems more sensitive to the mesh regularity. As we can see, nearly zero discretization error only occurs at the perfect equilateral triangles. Therefore when we compare the L_2 norm of the errors, shown in Fig. 5.7, a large decrease in the magnitude of the truncation error can be found in more regular meshes, along with an increase in the order of accuracy. For the discretization error, on the other hand, the order of accuracy for all three kinds of meshes tends to be the same. A small magnitude decrease can still be observed in the two more regular meshes.



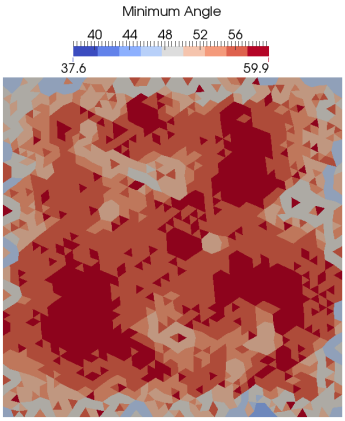
(a) Delaunay Mesh Regularity



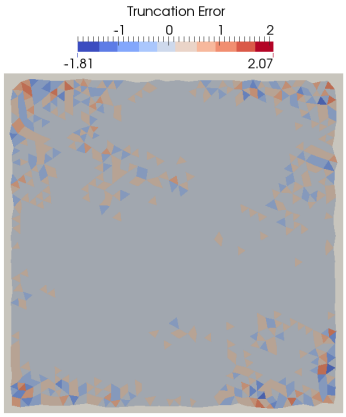
(b) Delaunay Truncation Error



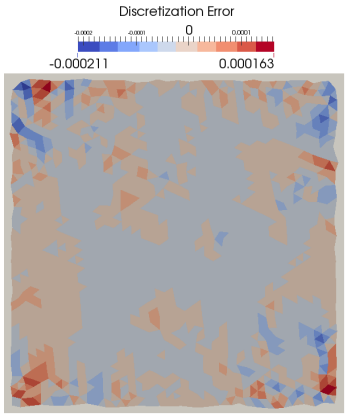
(c) Delaunay Discretization Error



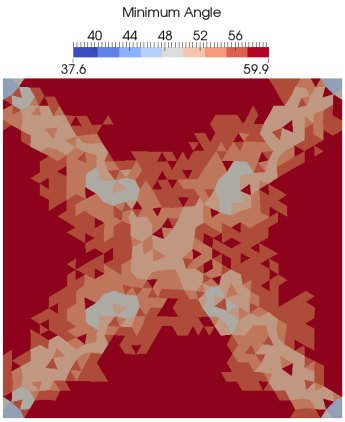
(d) Frontal-Delaunay Mesh Regularity



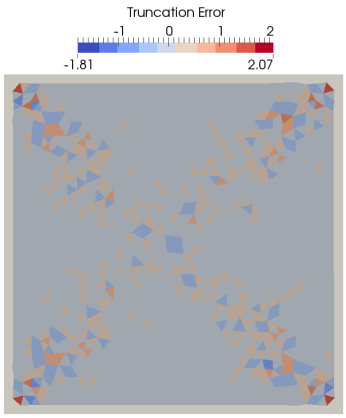
(e) Frontal-Delaunay Truncation Error



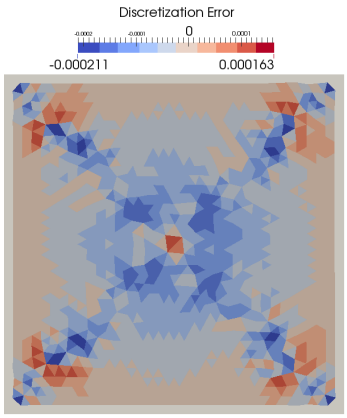
(f) Frontal-Delaunay Discretization Error



(g) AFLR Mesh Regularity

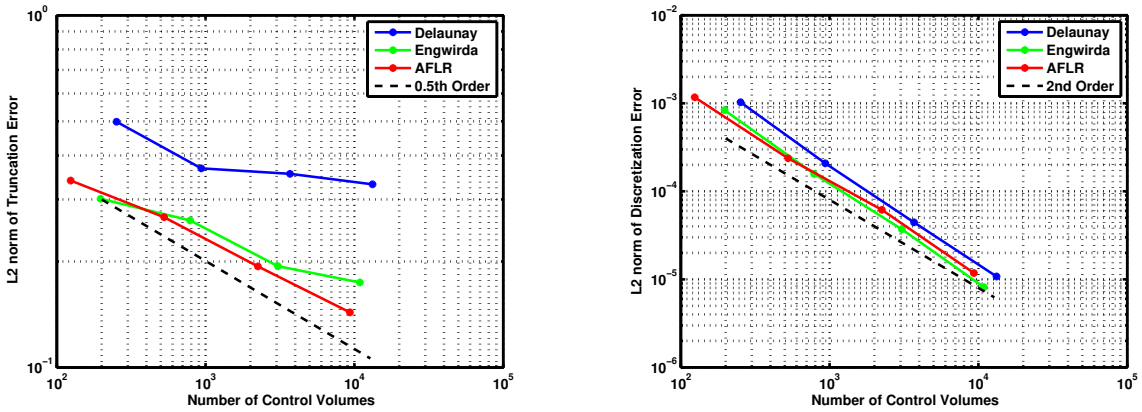


(h) AFLR Truncation Error



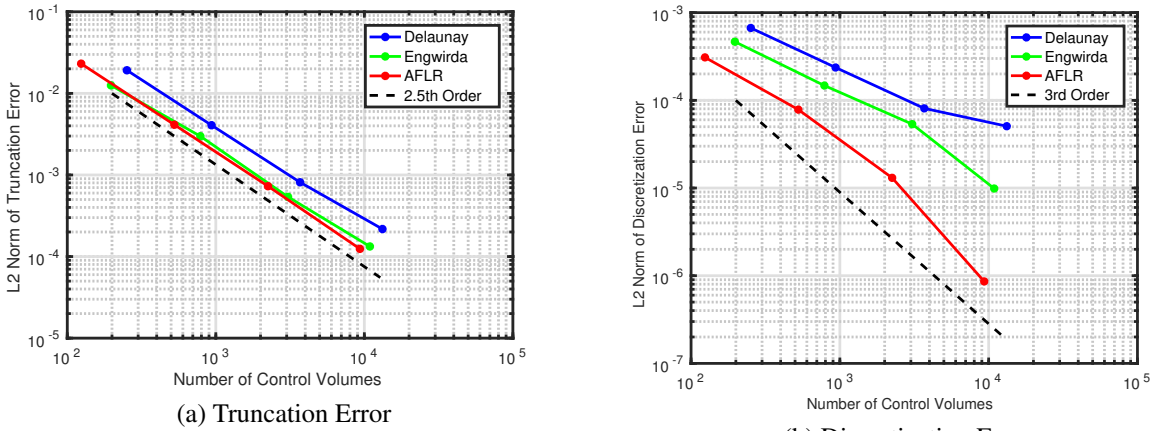
(i) AFLR Discretization Error

Figure 5.6: Mesh regularity, truncation error, and discretization error (second order)



(a) Truncation Error (b) Discretization Error

Figure 5.7: Order of Accuracy for Second Order Discretization



(a) Truncation Error (b) Discretization Error

Figure 5.8: Order of Accuracy for the Fourth Order Discretization

The fourth order discretization scheme [17, 29] is employed to show whether high order reconstruction and flux integral will affect the relationship between errors and the mesh regularity. Fig. 5.8 shows the convergence when performing the fourth order discretization. For the truncation error, the advantage of regular meshes for the order of accuracy disappears. They all converge at 2.5th order. But a regular mesh still gives lower magnitude of the error. When it comes to the discretization error, the AFLR mesh, which has most equilateral triangles, reduces the error magnitude dramatically, and increases the order of accuracy. The size of Marcum's AFLR mesh can be $1/16$ of the Delaunay mesh while the error magnitude is about the same.

Method	Number of Vertices in mesh	Total Time(s)	Time per insertion(s)
Delaunay	1915	0.957	4.997×10^{-4}
Engwirda	1597	0.965	6.043×10^{-4}
AFLR	1188	1.507	1.269×10^{-3}

Table 5.1: Mesh Generation Time Cost

5.1.3 Time cost

The two mesh generation schemes which generate more regular meshes take more time for each insertion because they calculate multiple possible Steiner points, and then reject some of them. However, they may require fewer cells to fill up the domain than the Delaunay triangulation. In Table 5.1, we see that Engwirda's frontal-Delaunay mesh generation scheme only takes a little longer than Delaunay triangulation in each insertion. With the help of fewer vertices in the mesh, the time cost in mesh generation is nearly the same. Marcum's AFLR scheme is the best scheme in terms of regularity, but checking whether a candidate vertex is too close to the existing ones takes a lot of time for each insertion. Even though it has the smallest mesh size, the time cost for AFLR is around twice that of the other two schemes. The difference in the time cost between the two new schemes and Delauany triangulation grows larger as the size of mesh increases, as shown in Fig. 5.9.

Considering that the regularity of mesh can reduce the error magnitude, the two new schemes have smaller time costs for a given error when combining both the mesh generation cost and the solver cost together. Fig. 5.10 shows the combined time cost. For the second order discretization, it is not worth waiting for the most regular AFLR mesh. In that case, Engwirda's frontal-Delaunay scheme is the optimal one, in that it reaches a certain error magnitude fastest. For the fourth order discretization, the advantage of Marcum's AFLR scheme is obvious. Even though it takes much longer time in mesh generation, the smaller mesh size saves a lot of time in the solver and produces much lower error for a given target edge length in the mesh. This pays off the large difference in mesh generation time cost when the mesh gets larger. In practice, usually the mesh generation code is run once and the mesh can be solved several times. In

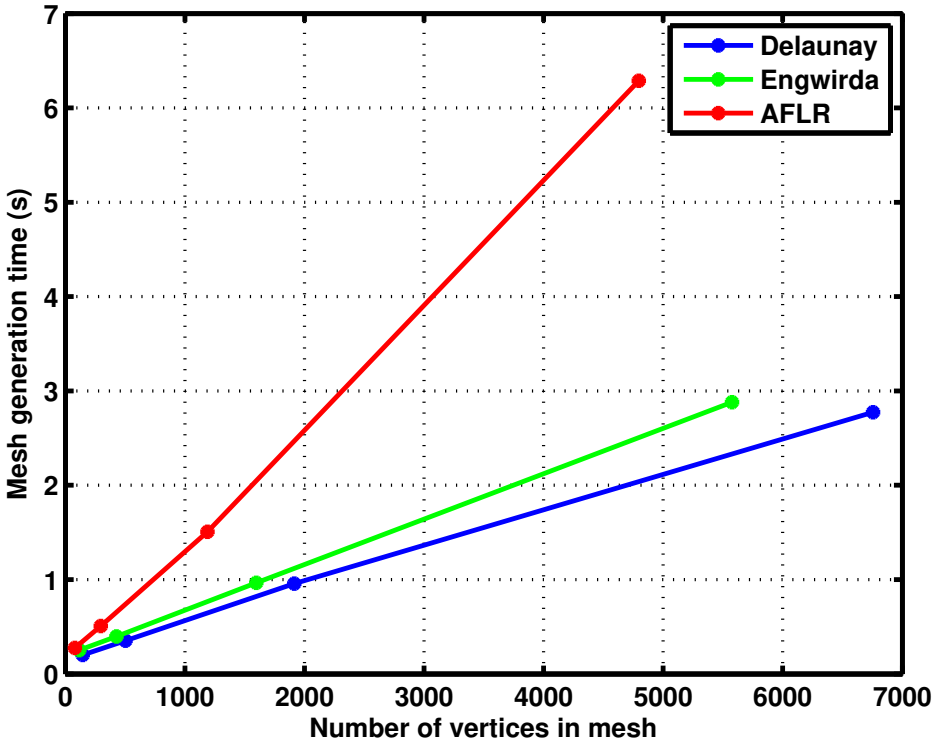
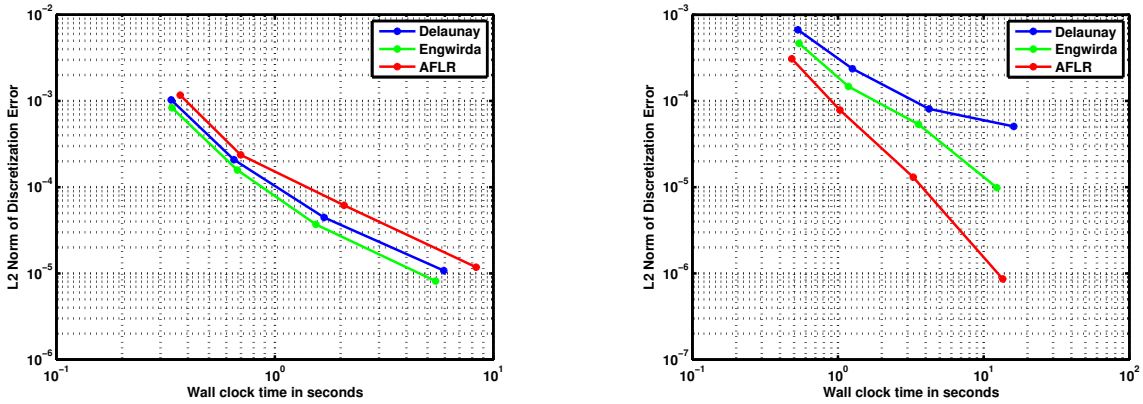


Figure 5.9: Mesh generation time on a square domain



(a) The 2nd Order

(b) The 4th Order

Figure 5.10: Time cost of the mesh generation and the solver

that case, the regular mesh is definitely worth waiting for.

5.2 Advection-diffusion equation in a channel

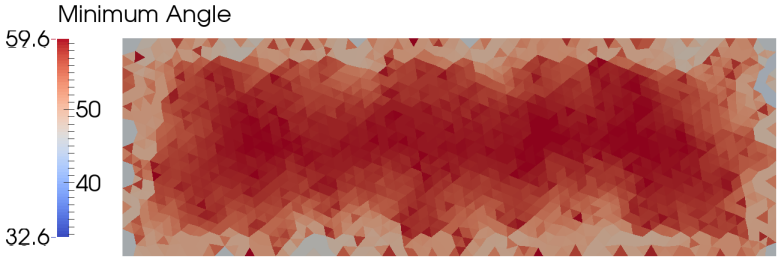
The second test case is solving the advection-diffusion equation in a channel.

5.2.1 Mesh regularity

The geometry is a rectangular channel with height $H = 1$ and length $L = 3$. As shown in Fig. 5.11, similar results as meshes on a square domain can be observed. The mesh from AFLR scheme has a strikingly large area of equilateral triangles. The only exceptions are the inlet, outlet, and a few cells in the middle. In this case, only the upper and lower boundaries are set to be layer index 0 at the beginning. The inlet and outlet have ∞ layer index. The refinement is marched from upper and lower boundaries into the middle. Similar to the square case, the right angles make this scheme fail to generate perfect layers. This is why the inlet and outlet have bad cells adjacent to them. Engwirda's frontal-Delaunay scheme does not improve the triangles near the boundaries. However it has several large chunks of perfect triangles in the interior domain. The Delaunay refinement still does not guarantee any equilateral triangles. The angle distribution shown in Fig. 5.12 also confirms the mesh qualities are similar to the previous square domain case.

The AFLR scheme has its limit. The finest mesh obtained from Marcum's AFLR scheme is discussed here to show where the AFLR scheme fails. The equilateral triangles layers have the limit. In Fig. 5.13, the triangles are colored by the cell layer index (Eqn. 4.1). In this figure, the maximum layer index is 59.3. There are about 50 layers in the mesh. The first 20 layers are almost perfect. However, at the transient layer where the perfect pattern breaks, there are several triangles that barely passed the local quality test. They make the edge lengths in the next layer vary a lot and result in the perfect layers terminating. If the minimum angle allowed in the mesh is set higher than 28°, the hatched triangle shown in Fig. 5.14a is

(a) Delaunay triangulation (1652 vertices)



(b) Engwirda's frontal-Delaunay (1401 vertices)

(c) Marcum's AFLR (1689 vertices)

Figure 5.11: Meshes in a channel

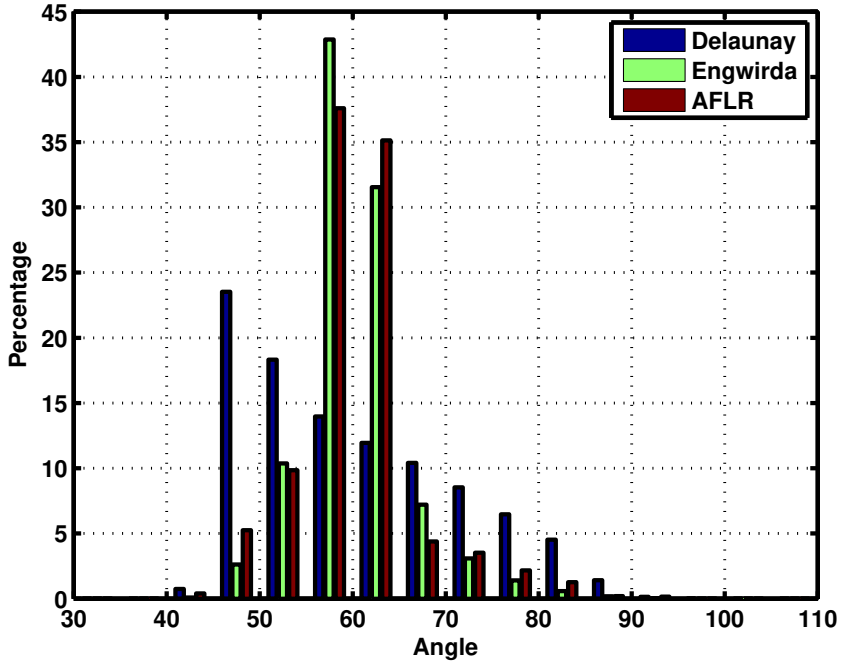


Figure 5.12: Angle distribution of meshes in a channel

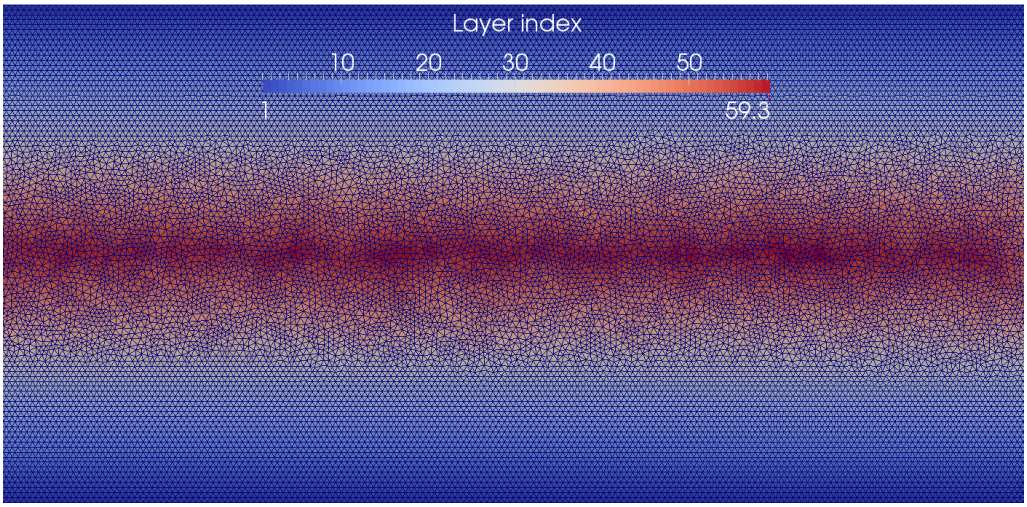


Figure 5.13: Bad AFLR mesh

(a) The normal pattern

(b) AFLR fails circumstance

Figure 5.14: AFLR scheme failure

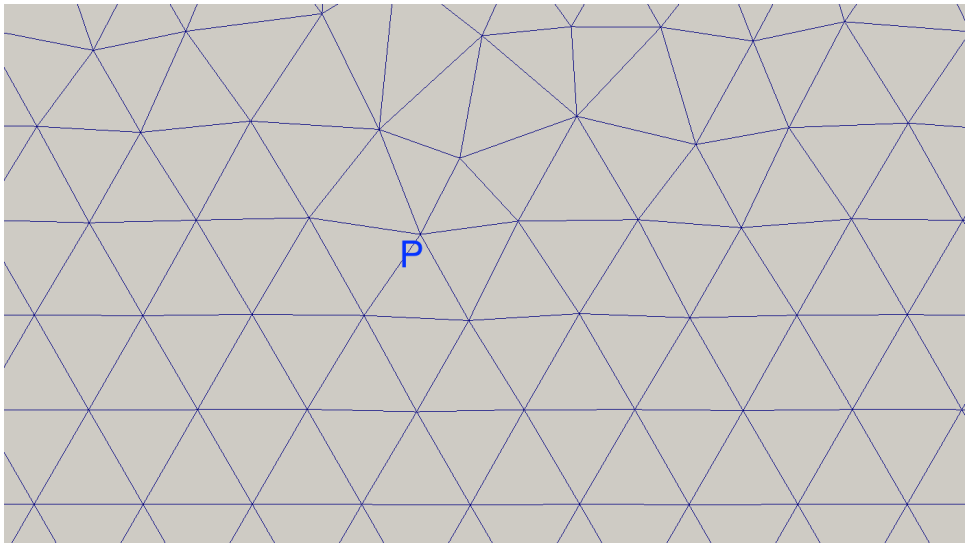


Figure 5.15: Near view of the transition area

refined. This generates another three almost equilateral triangles, keeping this layer complete and regular. However, if angles less than or equal to 28° are allowed in the mesh, the triangle to be refined would be the hatched one in Fig. 5.14b. This leads to two adjacent edges with very different edge lengths. The triangles generated are also irregular. Because there is no smoothing technique performed in the middle of refinement, in the next layer, the triangles around this area will be worse, as shown in Fig. 5.15. The point P is generated in the pattern from Fig. 5.14b. Slight misallignment is made on the layer and after two more layers, the perfect pattern blows up. One obvious amendment for this case is to increase the quality bound to make each layer perfect. But not all geometries can be meshed perfectly. Increasing the quality bound is highly likely to introduce problems with robustness.

5.2.2 Truncation error and discretization error

The 2D advection-diffusion equation

$$\vec{u} \cdot \nabla \phi = \alpha \nabla^2 \phi$$

is solved in this case. A manufactured exact solution (shown in Fig. 5.16) is provided:

$$\phi = \left(\frac{e^{R_1 x} R_2 e^{R_2 L} - e^{R_2 x} R_1 e^{R_1 L}}{R_2 e^{R_2 L} - R_1 e^{R_1 L}} \right) \sin(\pi y),$$

where $R_1 = \frac{H}{2\alpha} + \sqrt{\frac{H^2}{4\alpha^2} + \pi^2}$ and $R_2 = \frac{H}{2\alpha} - \sqrt{\frac{H^2}{4\alpha^2} + \pi^2}$. The coefficient of diffusion α is set as 0.01. The boundary conditions for both upper and lower boundaries are enforced by $\phi = 0$, whereas at the outlet, the gradient is set as zero. For the inlet boundary, the boundary condition is set as the exact solution $\phi = \sin(\pi y)$. Notice that ϕ changes in the y-direction more than the x-direction. The AFLR mesh is more regular than Engwirda's frontal-Delaunay mesh in the y-direction, so we expect the truncation error on the AFLR mesh will be much less than Engwirda's mesh. Both more regular meshes can be expected to give smaller truncation error

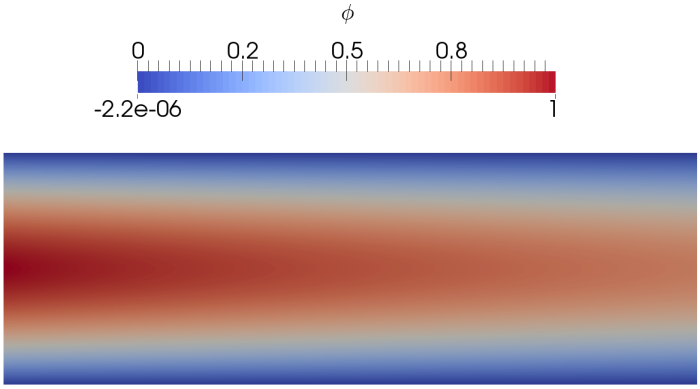
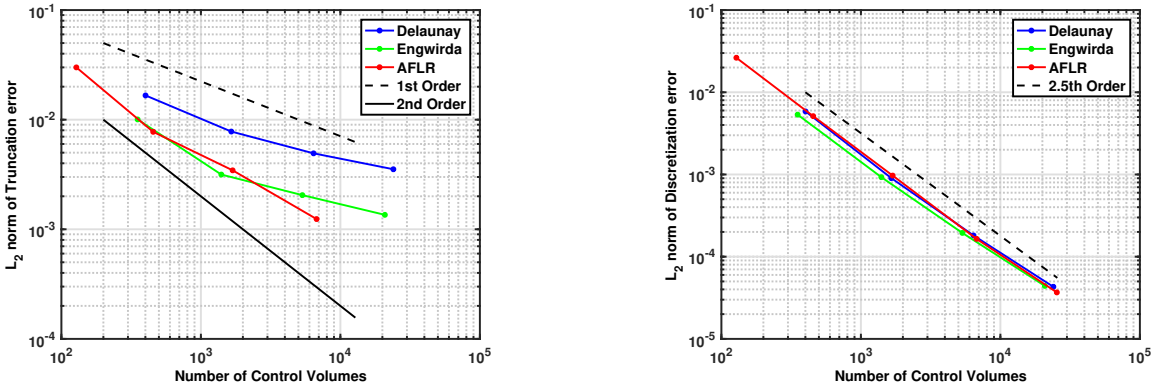


Figure 5.16: Exact solution for advection-diffusion equation



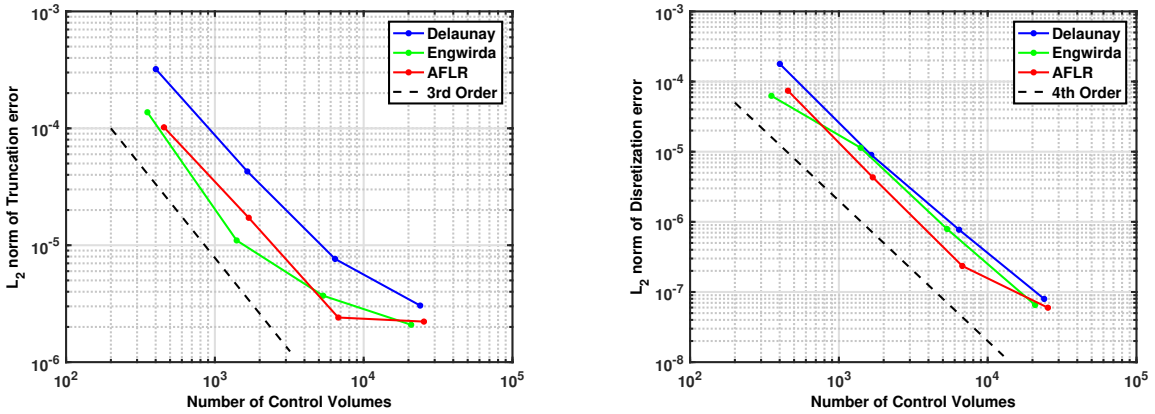
(a) Truncation error

(b) Discretization error

Figure 5.17: Order of Accuracy for second order discretization

than traditional Delaunay refinement.

The convergence of truncation errors is shown in Fig. 5.17a. The Delaunay mesh converges at first order, while the other two more regular meshes can achieve second-order for coarse meshes. As the mesh becomes finer, they become about first order. The large truncation error occurs at the area where irregular triangles exist. When we refine the mesh, the truncation error for the perfect control volumes decreases at second order whereas the bad control volumes are only about 0.25th order. On the very coarse mesh, the magnitudes of truncation error for regular and irregular control volumes are comparable, so they tend to get a higher order of accuracy globally. However, when the mesh is refined, the truncation errors for badly shaped control



(a) Truncation error

(b) Discretization error

Figure 5.18: Order of Accuracy for fourth order discretization

volumes can be hundreds of times larger than the global average. This large error magnitude is dominant, which lowers the global order of accuracy. For the discretization error, shown in Fig. 5.17b, the improvement is not obvious. Similar to the previous Poisson equation test case, the relation between truncation error and discretization error is complicated. Small truncation error does not guarantee small discretization error.

Fourth order discretization is also applied for this case. Again, a decrease in the magnitude of truncation error can be observed but for the discretization error, the improvement is small.

5.2.3 Time cost

Since the discretization error is not improved as we expected, the time cost is compared to see if it is worthwhile to generate regular meshes. In Table 5.2, the mesh generation time cost is comparable to the second order solving time cost. The AFLR mesh has very long mesh generation time and it is not worth waiting since it does not decrease time cost in solver and discretization errors. It takes nearly the same time to generate the Delaunay mesh and Engwirda's frontal-Delaunay mesh. With the help of the smaller frontal-Delaunay mesh size, it takes a shorter time for Engwirda's frontal-Delaunay mesh to converge. The decrease in time cost is not large but it does no harm to generate Engwirda's frontal-Delaunay mesh. As for the fourth order solver shown in Table 5.3, the time cost decrease for Engwirda's mesh is larger.

Mesh Size	Delaunay		Engwirda		AFLR	
	Mesh	Solver	Mesh	Solver	Mesh	Solver
~400	0.28	0.29	0.28	0.31	0.55	0.31
~1600	0.75	0.81	0.71	0.63	2.03	0.87
~6400	2.65	3.01	2.60	2.42	8.52	3.16
~25000	10.28	13.59	10.86	11.20	33.00	14.68

Table 5.2: Time cost in mesh generation and second order solver

Mesh Size	Delaunay		Engwirda		AFLR	
	Mesh	Solver	Mesh	Solver	Mesh	Solver
~400	0.28	0.60	0.28	0.59	0.55	0.82
~1600	0.75	2.60	0.71	1.89	2.03	2.54
~6400	2.65	9.52	2.60	8.76	8.52	10.98
~25000	10.28	43.02	10.86	32.18	33.00	39.79

Table 5.3: Time cost in mesh generation and fourth order solver

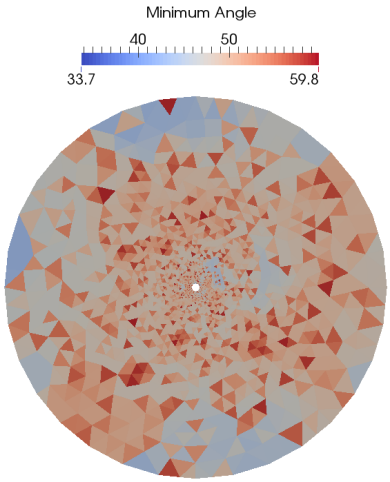
Considering the slightly better error performance, Engwirda's mesh is definitely recommended for high order solver in this test case.

5.3 Potential flow around a circular cylinder

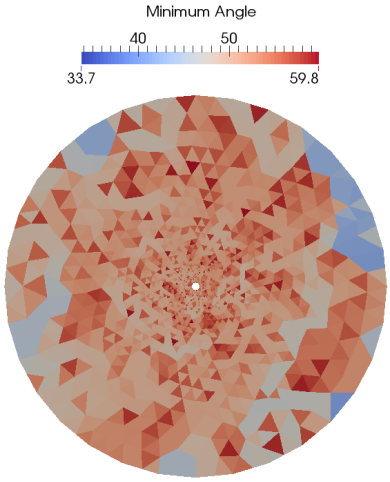
The incompressible flow is solved in ANSLib's compressible flow solver with some modifications.

5.3.1 Mesh regularity

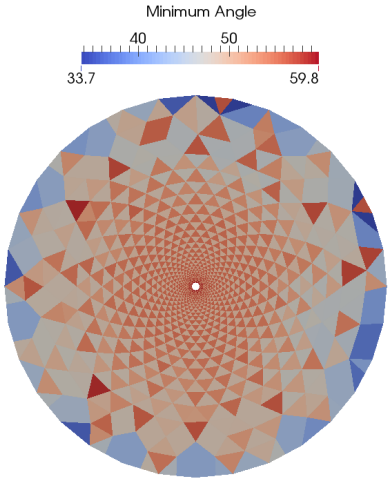
The domain is bounded by two circles. The inner circle denotes a circular cylinder whose radius is 1. The wall boundary condition is set for the circular cylinder. The outer circle denotes the far field boundary. The radius is 50. Three different mesh generation schemes are applied to generate meshes in which cell sizes increase gradually with the distance from the circular cylinder. Fig. 5.19 shows that in the mesh obtained from the AFLR scheme, most triangles have minimum angles about 50°. The circle is a good geometry for a layer-wise marching method. Even though they are not nearly equilateral triangles, they grow in a smooth



(a) Delaunay mesh



(b) Engwirda's frontal-Delauany



(c) Marcum's AFLR

Figure 5.19: Meshes around a circular cylinder

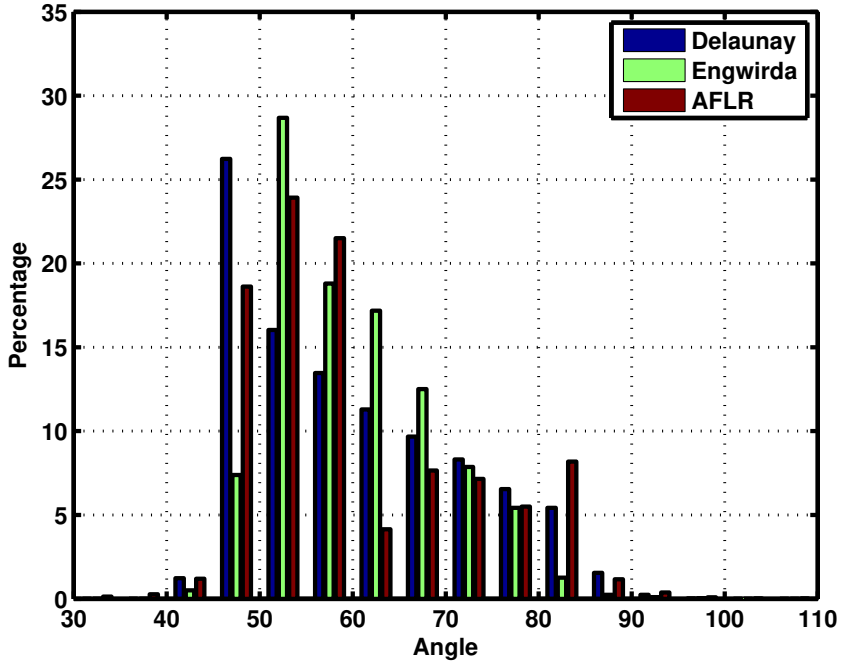


Figure 5.20: Angle distribution of meshes around a circular cylinder

variation. The AFLR scheme usually fails to generate beautiful triangles everywhere because bad triangles are kept in the mesh as discussed in the previous section. In this case it only happens in the far field. It will not affect the overall error improvement since the flow varies little in that area. Engwirda's frontal-Delaunay scheme also generates many triangles whose minimum angles are above 50°. The Delaunay mesh still is the worst in terms of the regularity, as shown in Fig. 5.20.

5.3.2 Truncation error and discretization error

In this case, the Euler equations are solved

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ E\end{pmatrix}+\frac{\partial}{\partial x}\begin{pmatrix}\rho u\\ \rho u^2+P\\ \rho uv\\ u(E+P)\end{pmatrix}+\frac{\partial}{\partial y}\begin{pmatrix}\rho v\\ \rho uv\\ \rho v^2+P\\ v(E+P)\end{pmatrix}=0, \tag{5.2}$$

where the energy $E = P/(\gamma-1) + \rho(u^2+v^2)/2$. The analytic solution is available for the incompressible potential flow around a circular cylinder. We consider the case where the center of the cylinder is $(0,0)$. There is no circulation and the far field flow is in the x-direction. The complex conjugate velocity $W = u - iv$ is :

$$W = V_\infty - \frac{V_\infty R^2}{z^2}$$

where $z = x + iy$ and V_∞ is the free stream velocity. In our in-house code, the equations are solved in non-dimensional form, so the non-dimensional Mach number $M_\infty = 0.3$ is used as V_∞. The radius $R = 1$. For this test case, the incompressible solution can be simplified to

$$\rho = 1$$

$$W = M_\infty - M_\infty \cdot \frac{1}{(x+iy)^2},$$

$u = real(W)$ and $v = -imag(W)$. The pressure is calculated by Bernoulli's equation

$$p = p^* - \frac{1}{2}\rho\left(u^2+v^2\right),$$

where $p^* = \frac{1}{\gamma}\cdot\left(1+\frac{\gamma-1}{2}M_\infty^2\right)^{\frac{\gamma}{\gamma-1}}$.

The analytic solution shown in Fig. 5.21 is for an incompressible solver. To use this as a manufactured solution for compressible Euler equation 5.2, the source term calculated by substituting the analytic solution into Eqn. 5.2 needs to be added.

$$\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ E\end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix}\rho u\\ \rho u^2+P\\ \rho uv\\ u(E+P)\end{pmatrix} + \frac{\partial}{\partial y}\begin{pmatrix}\rho v\\ \rho uv\\ \rho v^2+P\\ v(E+P)\end{pmatrix} = \begin{pmatrix}0\\ 0\\ 0\\ S\end{pmatrix},$$

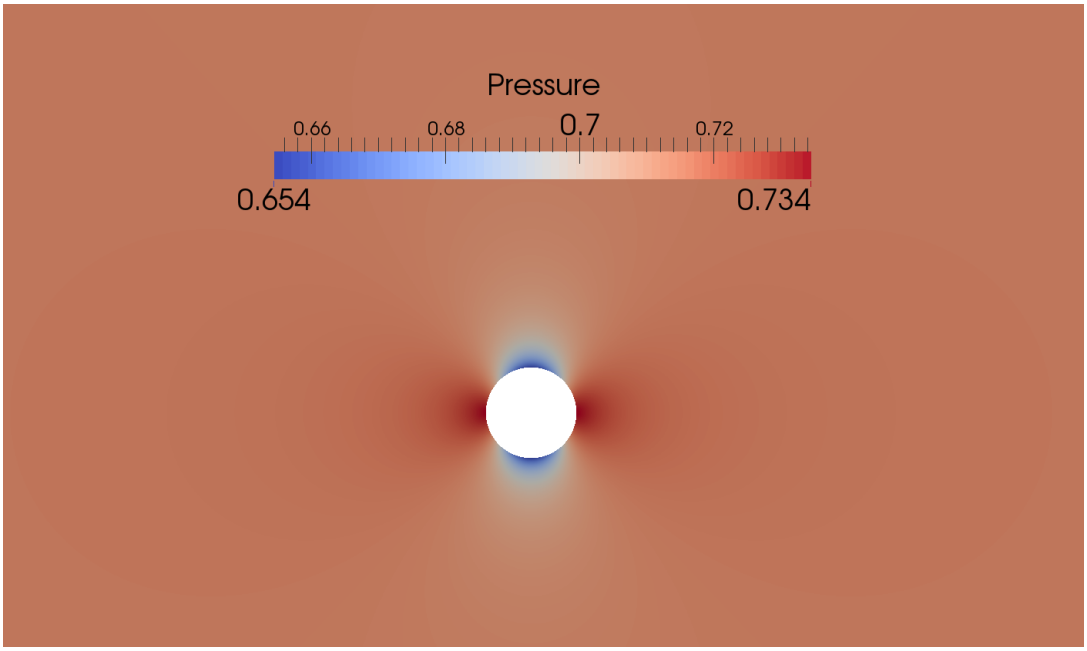


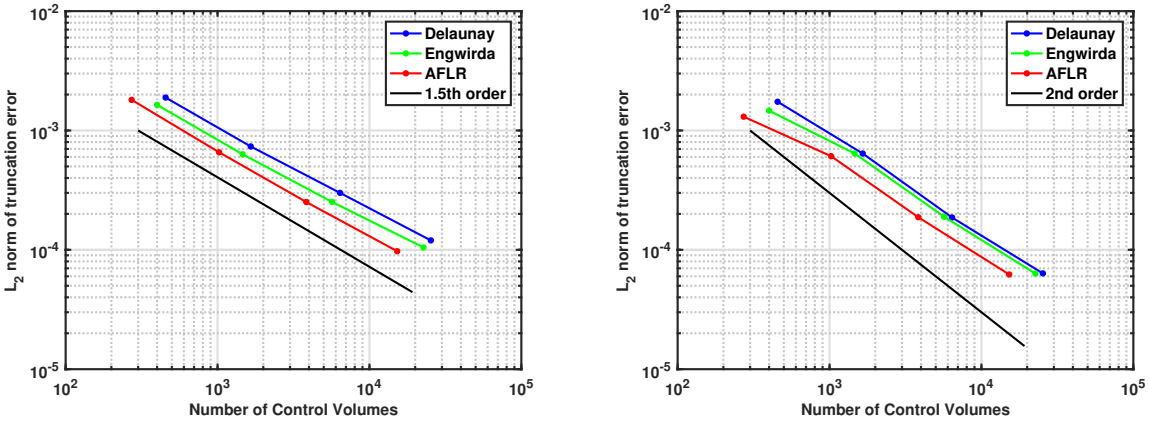
Figure 5.21: Exact solution for the non-dimensional pressure around a circular cylinder

Only the energy equation has a source term:

$$S = \frac{\partial}{\partial x}\left(u \cdot \left(\frac{\gamma p}{\gamma - 1} + \frac{u^2 + v^2}{2}\right)\right) + \frac{\partial}{\partial y}\left(v \cdot \left(\frac{\gamma p}{\gamma - 1} + \frac{u^2 + v^2}{2}\right)\right).$$

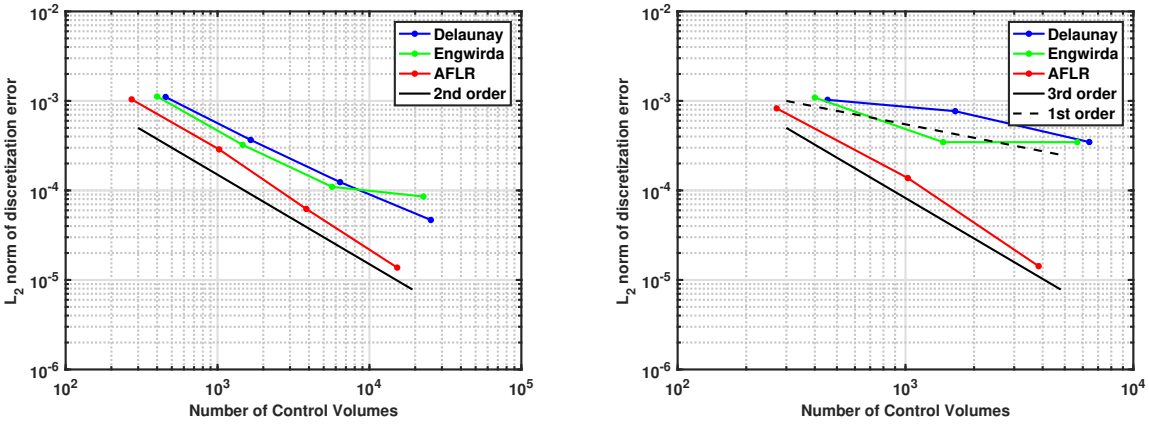
With the help of the complex conjugate velocity W, the source term can be evaluated in $S(x, y)$.

The truncation error is estimated by the flux integral when setting the manufactured solution as the initial solution. As shown in Fig. 5.22a, the magnitude of truncation error in the AFLR mesh is reduced compared to the other two meshes. The Engwirda's frontal-Delaunay mesh also decreases the truncation error magnitude. As for the discretization error, shown in Fig. 5.23a, Marcum's AFLR method improves the magnitude of the error. All the three meshes achieve second-order accuracy on discretization error, which is consistent to the discretization scheme in use. When the fourth order discretization scheme is applied, similar results can be observed in Fig. 5.22b for the truncation errors. The discretization error shown in Fig. 5.23b illustrate that the AFLR mesh improves both error magnitude and the order of accuracy. In the fourth order solver, the finest Delaunay mesh and Engwirda's frontal-Delaunay mesh fail to converge. This is because that we are solving the incompressible flow in a compressible



(a) The second order solver (b) The fourth order solver

Figure 5.22: Truncation error for the energy equation



(a) The second order solver (b) The fourth order solver

Figure 5.23: Discretization error for the pressure

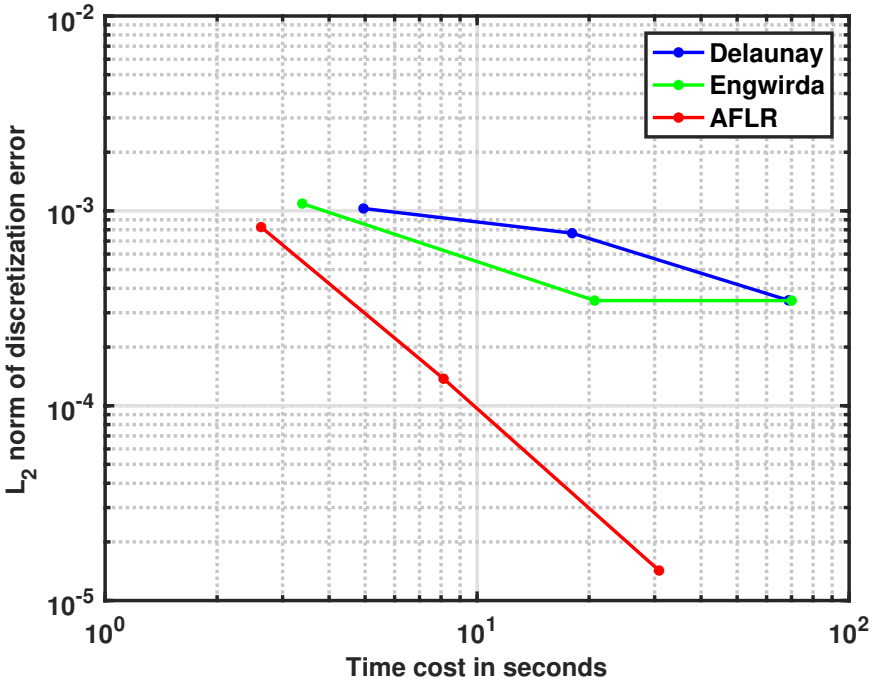


Figure 5.24: Time cost for both mesh generator and solver (fourth order)

flow solver. The solver tries to work against the source term. The lack of dissipation in the high order solver makes it more sensible to the errors. The AFLR mesh is the best in error magnitude but the time cost must be checked since the Euler equations are harder to solve than the Poisson equation and the advection-diffusion equation.

5.3.3 Time cost

The advantage of the AFLR scheme in the mesh regularity also contributes to time savings. As shown in Fig. 5.24, AFLR is definitely the best mesh generation scheme for this potential flow around a cylinder case as measured by combined time to a given solution error. Note that the mesh generation time cost for AFLR scheme is about twice as high as the other two (Table 5.4). But the smaller size and the mesh regularity help to reduce the time cost in the high order solver. The number of non-linear iterations required by the AFLR mesh is the smallest, which contains the time demanding tasks such as the Jacobian calculation and factorization.

For solving the potential flow around a circular cylinder, Marcum's AFLR scheme is obvi-

Mesh Size	Delaunay-refinement	Engwirda's frontal-Delaunay	Marcum's AFLR
~300	0.2687	0.2433	0.3647
~1,000	0.7430	0.7026	1.0411
~4,000	2.8018	2.9261	4.6033
~20,000	12.5435	11.5693	20.3652

Table 5.4: Time cost in mesh generation (seconds)

	Delaunay	Engwirda	AFLR
Mesh size	6405	5675	3840
Non-linear iteration (NLI)	9	10	6
Linear iteration (LI)	78	234	37
Time cost (s)	65.90	67.17	26.24

Table 5.5: Fourth order solver time cost

ous the best choice in both reducing errors and saving time.

5.4 NACA-0012 airfoil

In this test case, the compressible inviscid flow around the NACA-0012 airfoil is solved. The Mach number is 0.5 and different angles of attack (α) are applied.

5.4.1 Mesh regularity

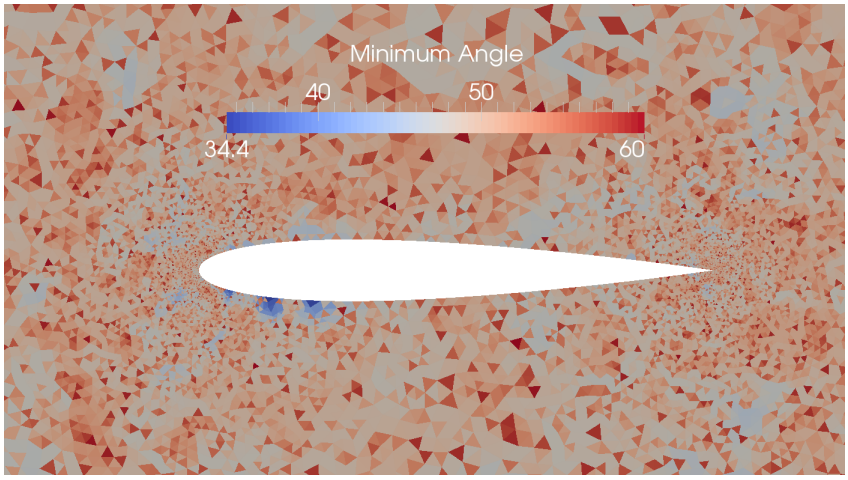
Similar to the cylinder case, Engwirda's frontal-Delaunay scheme and Marcum's AFLR scheme improve the mesh regularity in different ways. As shown in Fig. 5.25b, in Engwirda's mesh, large areas of dark red are always expected but at the same time, they lack continuity. A perfect equilateral triangle is not necessarily adjacent to another equilateral triangle or nearly equilateral triangle. On the other hand, in the mesh obtained from Marcum's AFLR scheme, the first few layers around the boundary usually contain lots of equilateral triangles. As shown in Fig. 5.25c, the areas around the leading edge and trailing edge are filled with nearly equilateral triangles. They also show a continuous pattern thanks to the layer-wise marching pattern. In this case, the triangles around the airfoil in the AFLR meshes are not all nearly equilateral

triangles. Several of them have the minimum angle of 45°. As shown in Fig. 5.26, the AFLR scheme generates fewer nearly 60° angles than Engwirda's frontal-Delaunay scheme. But both of them are still quite good when compared with the Delaunay mesh.

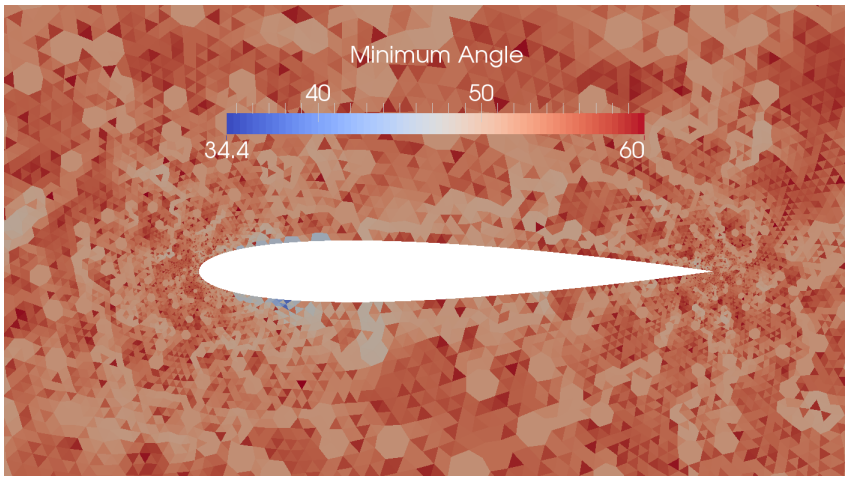
5.4.2 Errors

The exact solution of this case is not available, therefore the truncation error is omitted for this case. In Fig. 5.27, the lift coefficient and drag coefficient are estimated when the angle of attack is 2°. Fig. 5.27a shows the convergence of the lift coefficient. The solid lines are for the second order discretization and dashed lines denote the fourth order discretization. For the second order discretization, the AFLR mesh, which has the best mesh quality around the airfoil, converges slightly faster than Engwirda's mesh. The Delaunay mesh has a different convergence pattern, with a slightly different grid-converged value. For higher order discretization, the AFLR mesh and Engwirda's frontal-Delaunay mesh perform consistently with second order scheme. As for the Delaunay mesh, the convergence pattern is similar to the other two meshes. But it still gets a different value. As shown in Fig. 5.27b, the convergence of the drag coefficient is almost exactly the same for all three meshes. Generally speaking, mesh regularity does not lead to significant improvement on lift coefficients and drag coefficients. For the second order scheme, mesh regularity helps to converge faster to an acceptable solution. But when it comes to higher order, the improvement is not significant.

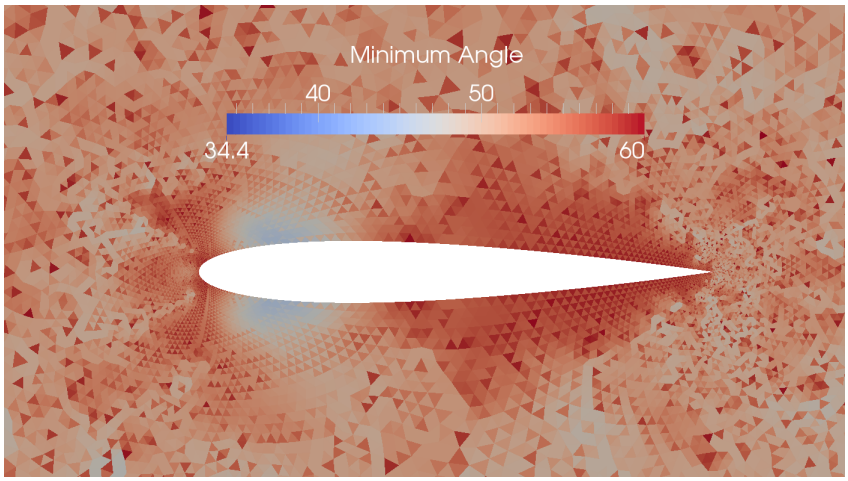
The numerical tests are re-run for the same meshes when setting the angle of attack as 4°. Similar results can be observed in Fig. 5.28. Again, mesh regularity does not help in this case.



(a) Delaunay mesh



(b) Engwirda mesh



(c) AFLR mesh

Figure 5.25: Regularity for meshes around NACA-0012 airfoil

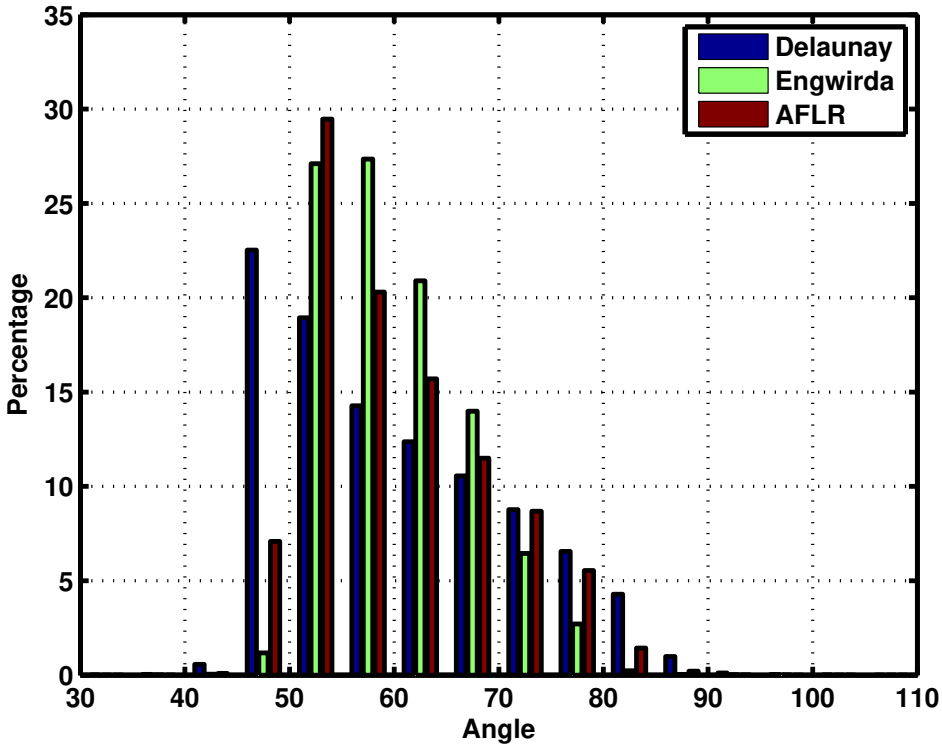
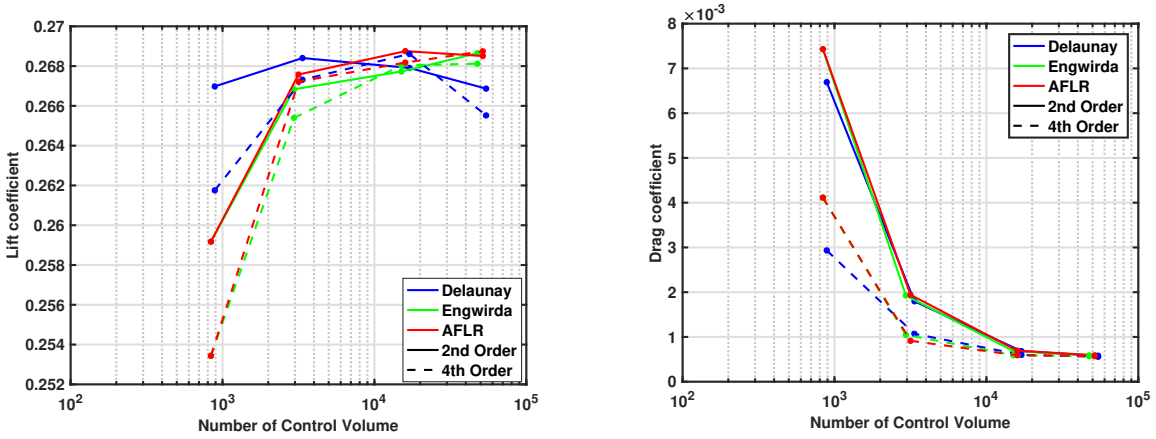


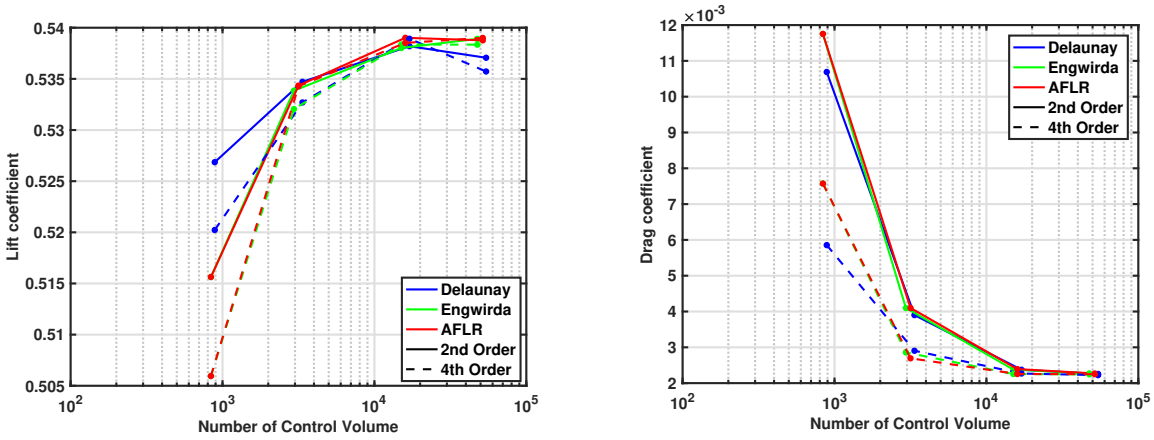
Figure 5.26: Angle distribution of meshes around NACA-0012 airfoil



(a) Lift coefficient

(b) Drag coefficient

Figure 5.27: NACA-0012 airfoil with angle of attack 2°



(a) Lift coefficient

(b) Drag coefficient

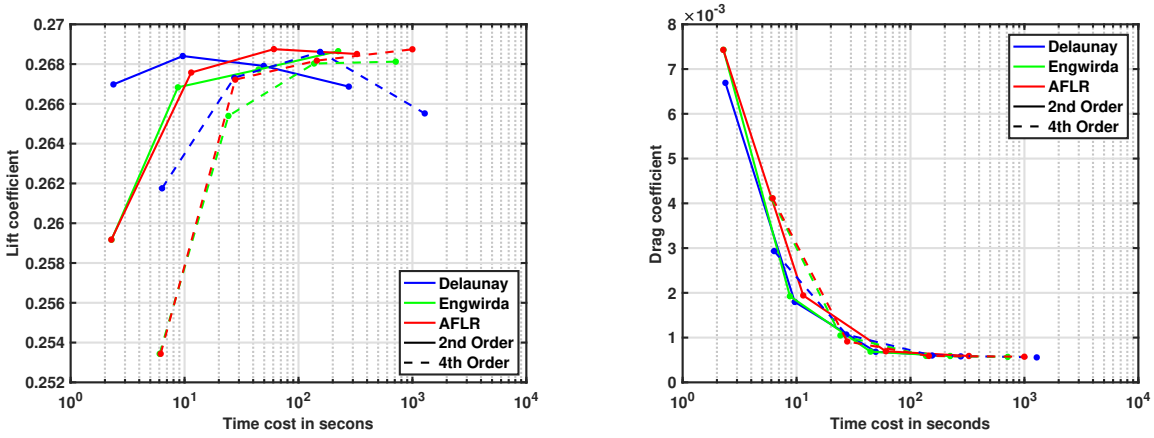
Figure 5.28: NACA-0012 airfoil with angle of attack 4°

5.4.3 Time cost

Since mesh regularity does not help to get an accurate solution on a coarser mesh as expected, it is doubtful whether it reduces the total time cost. As shown in Fig. 5.29, the improvements in time cost for both second order and fourth order scheme are not obvious. Table 5.6 shows the time cost for the total of mesh generation and solving with 2° angle of attack and the fourth order scheme. We can observe that the time cost in the solver is much higher than that in the mesh generator. Therefore in this case, the additional time cost of Marcum's AFLR scheme can be neglected. The advantage of the mesh regularity shortening the solving process is only observed for fine meshes. The time cost of Engwirda's mesh is about half of the one of Delaunay mesh. Marcum's mesh also decreases time cost by about 25 percent. When running the cases several times, more regular meshes are recommended to reduce the time

Mesh Size	Delaunay		Engwirda		AFLR	
	Mesh	Solver	Mesh	Solver	Mesh	Solver
~1000	0.48	5.85	0.55	5.49	0.52	5.64
~4000	1.77	25.54	1.85	22.40	4.01	23.83
~16000	8.42	146.52	8.41	126.68	22.60	122.24
~64000	30.66	1251.67	27.14	685.97	79.90	919.58

Table 5.6: Comparison of time cost in mesh generation and the fourth order solver ($\alpha = 2°$)



(a) Time cost vs lift coefficient

(b) Time cost vs drag coefficient

Figure 5.29: Time cost of mesh generation and solver in total

	Delaunay	Engwirda	AFLR
Mesh Size	54341	47613	51683
NLI	22	15	16
LI	156	134	177
Time Cost	1251.67	686.97	919.58

(a) AoA 2°

	Delaunay	Engwirda	AFLR
Mesh Size	54341	47613	51683
NLI	24	16	13
LI	198	140	176
Time Cost	1245.98	733.67	671.33

(b) AoA 4°

Table 5.7: Time cost in the fourth order solver

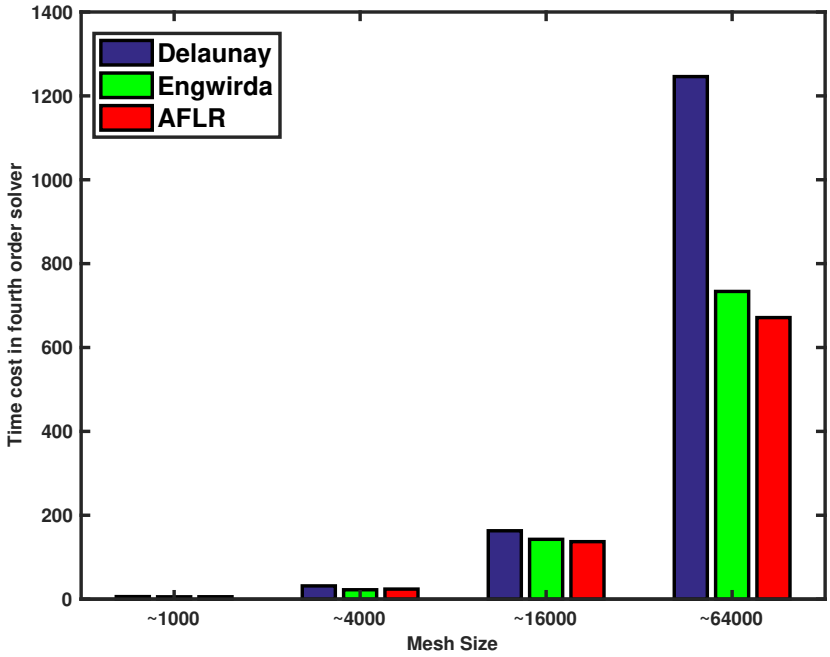


Figure 5.30: Time cost in the fourth order solver for AoA 4°

cost. For 4° angle of attack, similar results (Fig. 5.30) can be obtained: the AFLR mesh and Engwirda's frontal-Delaunay mesh helps to decrease the time cost in the solver. Similar to the circular cylinder case, mesh regularity helps reduce the mesh size and the number of non-linear iteration required. As shown in Table 5.7b, even though the AFLR mesh is a bit larger than Engwirda's frontal-Delaunay mesh, the smaller number of non-linear iterations required helps to reduce the time cost. Again, the Delaunay mesh is worse than the other two in terms of the number of non-linear iterations. Notice that this improvement is only observed with a high order discretization scheme. The decrease in time cost for the second order scheme is very limited as shown in Fig. 5.31.

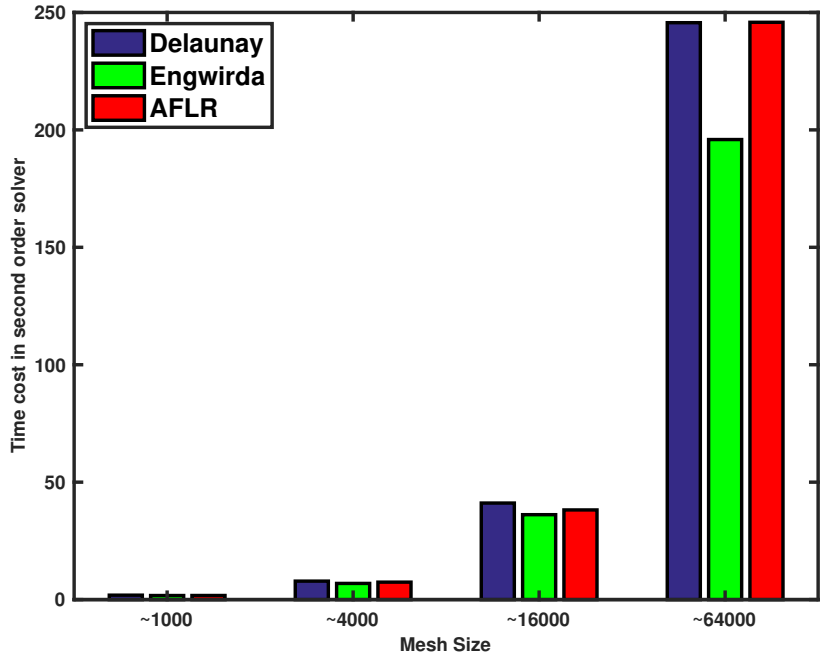


Figure 5.31: Time cost in the second order solver for AoA 2°

Chapter 6

Conclusions

The previous research on mesh regularity is completed on structured triangular meshes [8, 20]. However, the impact of unstructured mesh regularity on errors is missing. In my thesis, I increased the unstructured mesh regularity with Engwirda's frontal-Delaunay algorithm and Marcum's AFLR algorithm. The impact of that on errors is a supplement to the previous research. Besides, the impact of mesh regularity on time costs in both mesh generation and the flow solver is also considered in this thesis, which is another missing part from the previous research. The unstructured mesh regularity reduces the time cost in the solver, especially in the high order solver, which is not expected before this study.

Marcum's AFLR algorithm works well in terms of the unstructured mesh regularity. The original scheme is not consistent with the Delaunay refinement framework, which brings difficulties to implement this algorithm. In my thesis, I made several necessary modifications to implement this algorithm in the Delaunay refinement framework. The layer index introduced in Chapter 4 works well to achieve the layer-wise pattern in the Delaunay refinement framework. The point selection method in Marcum's algorithm is supplemented with the off-centres. The improved boundary discretization technique is introduced in Section 2.1. It is specially designed for airfoils. It discretizes the boundary in a smooth way. This smoothness of the boundary edge length helps the modified AFLR algorithm perform better in that the abrupt change of edge lengths will break the layer-wise pattern. With the help of these modifications, the modified AFLR algorithm works well and I can reproduce the similar results as those from Marcum's algorithm.

After introducing the three different mesh generation schemes and several numerical test

cases, the questions posed in Section 1.1 can now be answered.

1. How regular a triangular mesh can we generate for realistic cases?

   Depends on geometries. For simple geometries such as a square or annulus, almost 70% of triangles in the most regular mesh we generated have minimum angles above 55°. For more realistic cases like airfoils, the ratio decreases to 30%.

2. What is the additional cost to generate such a mesh?

   Point calculation and rejection. In my implementation, both Engwirda's frontal-Delaunay and Marcum's AFLR scheme use similar vertex location calculators, which calculate two or three more possible vertex locations than Delaunay triangulation. The size-optimal vertices even require iterative bisection method to complete. Marcum's AFLR method also adopts a rejection scheme. When it rejects a vertex to be inserted, all the calculation time cost for this vertex is in vain. Both the point location calculator and the rejection scheme increase the time costs at every step. The rejection scheme is dominant. Considering that the regular mesh size is usually smaller than the Delaunay mesh, Engwirda's frontal-Delaunay scheme, which only costs a little more because of the location calculator every step, can get similar time cost as Delauany mesh. The AFLR mesh costs much more than the other two, especially for very fine meshes.

3. Considering that cell shape affects the accuracy greatly, can we expect small discretization error using unstructured regular meshes?

   Not really. In my numerical tests, regular meshes did not consistently produce lower error.

4. Can a coarse regular mesh offer the same accuracy as a finer irregular mesh with lower costs in the flow solver?

Kind of. As answered in the previous question, the solution accuracy is not much improved by the regularity. It is not likely that a coarse regular mesh achieves the same error magnitude as the double-sized irregular mesh. Although regularity fails to reduce discretization error, it still helps to decrease time costs in the solver. The smaller mesh sizes and faster convergence in the solver both contribute to this. However, for the second order solver, this improvement is not significant. For higher order solvers, the smaller time cost on the regular mesh is an advantage.

Engwirda's frontal-Deluany scheme and Marcum's AFLR scheme both generate more regular meshes than Delaunay meshes. For some simple geometries like an annulus or a square, AFLR scheme is able to generate almost perfect equilateral triangles everywhere. It is a good choice since it gives smallest truncation error and discretization error with reasonable additional time cost in mesh generation. For more complicated numerical cases, especially for the high-order solver, Engwirda's frontal-Delaunay is optimal because of its good regularity, good error performance, and decrease in time cost.

Bibliography

[1] J. D. Anderson. *Computational Fluid Dynamics: The Basics with Applications*. McGraw-Hill Education, 1995.

[2] John David Anderson and J Wendt. *Computational Fluid Dynamics*, volume 206. Springer, 1995.

[3] Utkarsh Ayachit. The paraview guide: a parallel visualization application. 2015.

[4] Charles Boivin and Carl Ollivier-Gooch. Guaranteed-quality triangular mesh generation for domains with curved boundaries. *International Journal for Numerical Methods in Engineering*, 55(10):1185–1213, 2002.

[5] L Paul Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1-4):97–108, 1989.

[6] Boris Delaunay. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[7] Boris Diskin and James L Thomas. Notes on accuracy of finite-volume discretization schemes on irregular grids. *Applied Numerical Mathematics*, 60(3):224–226, 2010.

[8] Boris Diskin and James L Thomas. Effects of mesh regularity on accuracy of finite-volume schemes. In *50th AIAA Aerospace Sciences Meeting*, pages 2012–0609, 2012.

[9] Darren Engwirda. *Unstructured Tessellation & Mesh Generation*. PhD thesis, University of Sydney, 2014.

[10] Darren Engwirda and David Ivers. Face-centred Voronoi refinement for surface mesh generation. *Procedia Engineering*, 82:8–20, 2014.

[11] David A Field. Laplacian smoothing and Delaunay triangulations. *Communications in Applied Numerical Methods*, 4(6):709–712, 1988.

[12] Lori Freitag, Mark Jones, and Paul Plassmann. A parallel algorithm for mesh smoothing. *SIAM Journal on Scientific Computing*, 20(6):2023–2040, 1999.

[13] Lori A Freitag, Mark Jones, and Paul Plassmann. An efficient parallel algorithm for mesh smoothing. In *Proceedings of the Fourth International Meshing Roundtable*, pages 47–58, 1995.

[14] Lori A Freitag and Carl Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1997.

[15] Alireza Jalali. Truncation error analysis of unstructured finite volume discretization schemes. Master's thesis, University of British Columbia, 2012.

[16] Alireza Jalali and Carl Ollivier-Gooch. Accuracy assessment of finite volume discretizations of convective fluxes on unstructured meshes. In *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, number 2013-0705.

[17] Alireza Jalali and Carl Ollivier-Gooch. Accuracy assessment of finite volume discretizations of diffusive fluxes on unstructured meshes. In *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, number 2012-0608.

[18] Alireza Jalali, Mahkame Sharbatdar, and Carl Ollivier-Gooch. Accuracy analysis of unstructured finite volume discretization schemes for diffusive fluxes. *Computers & Fluids*, 101:220–232, 2014.

[19] F Juretić and AD Gosman. Error analysis of the finite-volume method with respect to mesh type. *Numerical Heat Transfer, Part B: Fundamentals*, 57(6):414–439, 2010.

[20] Aaron Katz and Venkateswaran Sankaran. Mesh quality effects on the accuracy of CFD solutions on unstructured meshes. *Journal of Computational Physics*, 230(20):7670–7686, 2011.

[21] Harvard Lomax, Thomas H Pulliam, and David W Zingg. *Fundamentals of Computational Fluid Dynamics*. Springer Science & Business Media, 2013.

[22] David L Marcum and Nigel P Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. *AIAA journal*, 33(9):1619–1625, 1995.

[23] MATLAB. *R2013b*. The MathWorks Inc., Natick, Massachusetts, 2013.

[24] Dimitri J Mavriplis. An advancing front Delaunay triangulation algorithm designed for robustness. *Journal of Computational Physics*, 117(1):90–101, 1995.

[25] J-D Müller, PL Roe, and H Deconinck. A frontal approach for internal node generation in Delaunay triangulations. *International Journal for Numerical Methods in Fluids*, 17(3):241–255, 1993.

[26] Carl Ollivier-Gooch. A toolkit for numerical simulation of PDEs: I. Fundamentals of generic finite-volume simulation. *Computer Methods in Applied Mechanics and Engineering*, 192(9):1147–1175, 2003.

[27] Carl Ollivier-Gooch. GRUMMP version 0.7.0 user's guide. *University of British Columbia*, 2016.

[28] Carl Ollivier-Gooch and Charles Boivin. Guaranteed-quality simplical mesh generation with cell size and grading control. *Engineering with Computers*, 17(3):269–286, 2001.

[29] Carl Ollivier-Gooch and Michael Van Altena. A high-order-accurate unstructured mesh finite-volume scheme for the advection–diffusion equation. *Journal of Computational Physics*, 181(2):729–752, 2002.

[30] Olivier Pironneau. *Finite Element Methods for Fluids*. Wiley Chichester, 1989.

[31] S Rebay. Efficient unstructured mesh generation by means of Delaunay triangulation and Bowyer-Watson algorithm. *Journal of Computational Physics*, 106(1):125–138, 1993.

[32] Christopher J Roy. Review of discretization error estimators in scientific computing. *AIAA Paper*, (2010-126).

[33] Jim Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[34] Jonathan R Shewchuk. Delaunay refinement mesh generation. Technical report, DTIC Document, 1997.

[35] Alper Üngör. Off-centers: A new type of Steiner points for computing size-optimal quality-guaranteed Delaunay triangulations. In *Latin American Symposium on Theoretical Informatics*, pages 152–161. Springer, 2004.

[36] Michael Van Altena. *High-order finite-volume discretisations for solving a modified advection-diffusion problem on unstructured triangular meshes*. PhD thesis, University of British Columbia, 1999.

[37] H K Versteeg and W Malalasekera. *An Introduction to Computational Fluid Dynamics, the Finite Volume Method*. Longman Scientific & Technical, 1995.

[38] Thomas Williams, Colin Kelley, and many others. Gnuplot 4.6: an interactive plotting program. `http://gnuplot.sourceforge.net/`, April 2013.

[39] Gary Yan, Varun Prakash Puneria, Alireza Jalali, and Carl Ollivier-Gooch. Truncation and discretization error for diffusion schemes on unstructured meshes. *AIAA SciTech. American Institute of Aeronautics and Astronautics*, (2014-0478).

[40] Daniel Zaide. Personal Email Communication.