Entire Domain Advancing Layer Surface Mesh (EDAM-S) Generation

by

Jasmeet Singh

B. Tech, Indian Institute of Technology (BHU), Varanasi, 2015

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

Masters in Applied Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Mechanical Engineering)

The University of British Columbia
(Vancouver)

December 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Entire Domain Advancing Layer Surface Mesh (EDAM-S) Generation

submitted by **Jasmeet Singh** in partial fulfillment of the requirements for the degree of **Masters in Applied Science** in **Mechanical Engineering**.

Examining Committee:

Carl Ollivier-Gooch, Mechanical Engineering
Supervisor

XYZ, Mechanincal Engineering
Supervisory Committee Member

PQR, LMN Department
Supervisory Committee Member

Table of Contents

List of Tables

List of Figures

Glossary

****

To be updated

****

Chapter 2

Methodology Part 1: Geometry Representation and Point Placement

In this section, we will talk about the initial import of surface triangulation and storing the surface as a collection of bezier surface patches. Then, the point placement subroutine is explained which decides the mesh element structure. Lastly, a small discussion on the local mesh element quality improvement is added to explain the face swapping algorithm.

2.1 Surface Import

2.1.1 Surface Representation - A brief overview

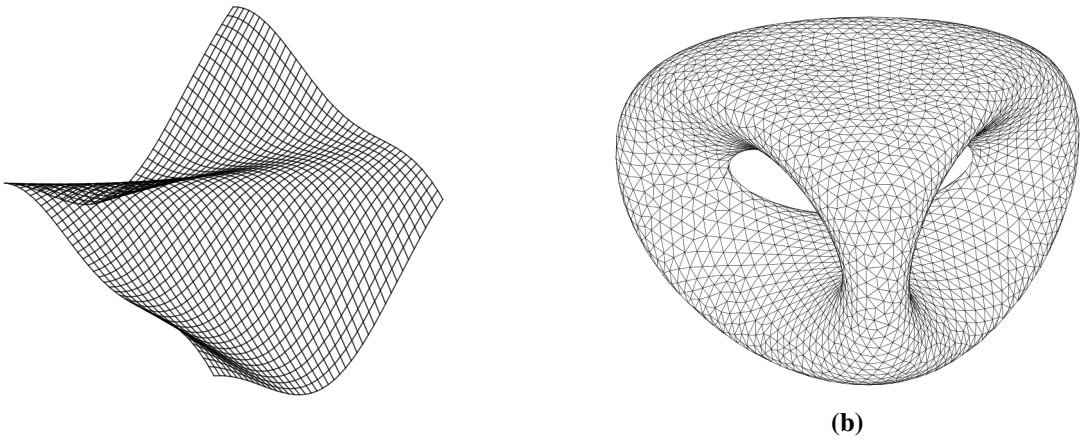
Surfaces can be represented in various forms. A surface could be represented by an explicit equation such as the one shown below.

$$z = F(x, y) \tag{2.1}$$

Where the coordinate z can be found by solving the aforementioned explicit equation, given the remaining two coordinates x and y. Explicit form of surfaces are easy to trace. However, it is not very versatile. Surfaces could also be represented with their implicit form, given by an implicit equation such as the one shown below.

$$F(x, y, z) = 0 \tag{2.2}$$

Here, solutions to the implicit equation represent points on the three dimensional surface. Figure 2.1



(a) **(b)**

Figure 2.1: (a) An explicit surface, given by $z = \cos((x+y)) + \frac{x^2}{6} - \frac{y^2}{6}$. (b) An implicit surface, given by $2y\left(y^2-3x^2\right)\left(1-z^2\right)+\left(x^2+y^2\right)^2-\left(9z^2-1\right)\left(1-z^2\right)=0$.

shows an implicit and an explicit surface together with their mathematical formulations.

Apart from the explicit and implicit forms, surfaces can also be represented in their parametric form. The coordinates of a point (x,y,z) of the surface patch are expressed as functions of parameters u and v in a closed rectangle:

$$x = x(u,v), \quad y = y(u,v), \quad z = z(u,v), \quad u_1 \leq u \leq u_2, \quad v_1 \leq v \leq v_2. \tag{2.3}$$

In vector notation, the parametric surface can be specified by a vector-valued function

$$\mathbf{r} = \mathbf{r}(u,v) \tag{2.4}$$

The parametric representation of surfaces is the most versatile out of the three. It is axis independent and is highly flexible in terms of defining complex intersections and point classification. It is generally easier to manipulate free-form shapes in parametric form than implicit or explicit forms [24]. Hence, most of the CAD packages use parametric form of the surfaces to manipulate them. Bezier surface patch is one example of parametric form of a surface.

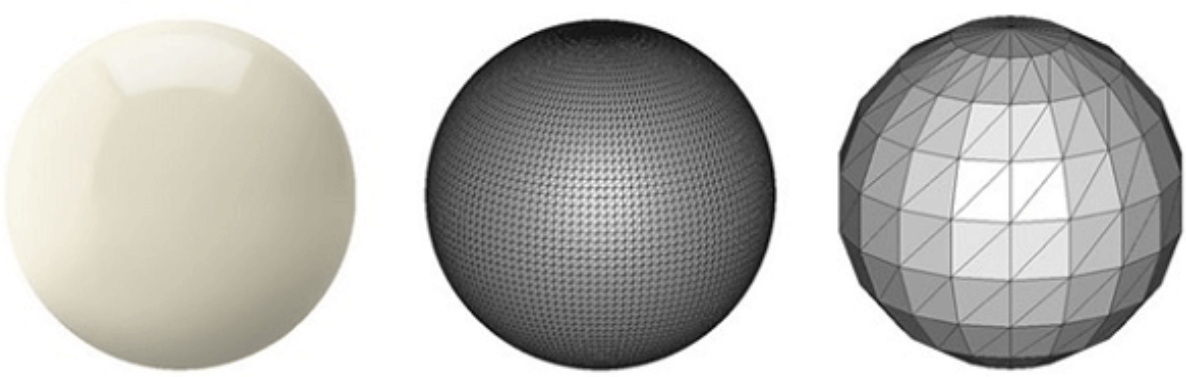


Figure 2.2: The perfect spherical surface on the left is approximated by tessellations. The figure on the right uses big triangles, resulting in a coarse model. The figure on the center uses smaller triangles and achieves a smoother approximation [1]

2.1.2 Surface File Format

Given a free-form 3D surface geometry, various CAD packages could be used to encode it and store it in a file. Encoding the geometry using an approximate mesh is one of the most common methods adopted to store a surface geometry. An approximate mesh is created by covering the surface geometry with a series of tiny imaginary polygons. Trianlges are the most common polygon used for mesh generation. The encoded surface mesh can be stored in a file for sharing and future reference purposes. These files store the vertices of the triangles as well as the outward normal directions to the triangles. This process of tiling a surface with non-overlapping geometric shapes is also known as tessellation. Hence, the file formats used for storing the surface representation are called tessellated formats.

Due to the independent development of various CAD packages, a plethora of surface file formats are present in the mesh generation ecosystem. Many of these formats, such as DWG file format by AutoCAD and BLEND file format by Blender are proprietary. Hence, many of these cannot be shared between people working on different CAD packages. Native file formats are used to solve this problem. These formats can be shared easily among people working on different meshing softwares. One of the most common neutral surface file format is the STL (STereoLithography) file format. This format is compatible with most of the CAD and visualization softwares. Hence, we use the STL file format to import the surface geometry into our mesh generation algorithm.

An STL file stores the surface as a triangulated mesh. The following information is stored for all the triangles in teh STL file format:

1. The coordinates of the vertices
2. The components of the unit normal vector to the triangle pointing outwards with respect to the 3D model

Innumerable software packages are available online which can be used to triangulate a surface (see [2] for a list of such softwares). A fine triangular mesh can be considered as an approximate encoding of a given surface geometry. The approximation could be improved by increasing the number of triangles or decreasing their size. However, using smaller triangles results in larger number of triangles needed to tile the surface. This increases the mesh file size. Hence, a user should define the mesh element size according to the kind of refinement needed.

2.2 Surface Import and Segmentation using Common Geometry Module (CGM)

As explained earlier, we import the surface geometry as a triangulation as an STL file. The Common Geometry Module (CGM) package is used to read the surface file and store the triangulation for further processing. The triangulated surface is stored as a collection of segmented sub-surfaces. The segmentation in CGM is done by identifying features in the surface. The only input parameter for surface segmentation accepted by CGM is the feature angle. We keep this feature angle value to be 135°. This value helps us to identify sharp corners and edges on the surface. Figure 2.3 shows a surface triangulation of an arbitrary mechanical part. The surface triangulation is segmented into 10 sub-surfaces, which are identified by the sharp features on the surface. Four of these are shown in outline in the figure.

Each triangle imported in CGM is stored as a quartic-bezier patch. We will refer to the imported triangulation as T and the underlying bezier surface representation as S. The underlying bezier surface representation S is considered to be the ground truth for the surface mesh and the mesh points are placed directly over S. Initial imported triangulation T is taken as the initial mesh.

Using T as our initial mesh means that we use a closed advancing layer mesh generation methodology. There are advantages and disadvantages of using such a methodology. Open advancing layer method requires less work overall to generate the mesh as there are no points to delete or reconnect to, ahead of the front. On the other hand, handling mesh layer collisions is more tricky in open advancing layer method as no connectivity information is available ahead of the front in such methods. This leads to abrupt layer closures, which are undesirable in an anisotropic mesh.

Closed advancing layer mesh generation method generates a valid surface mesh at any given point in time during the mesh generation process. Hence, there is more flexibility in terms of when to stop the marching layers and output the mesh in its current state. Additionally, connectivity information is known ahead of the front. This information is helpful while tackling front collisions. This will be explained in more detail when we talk about handling front collisions in *ref*.

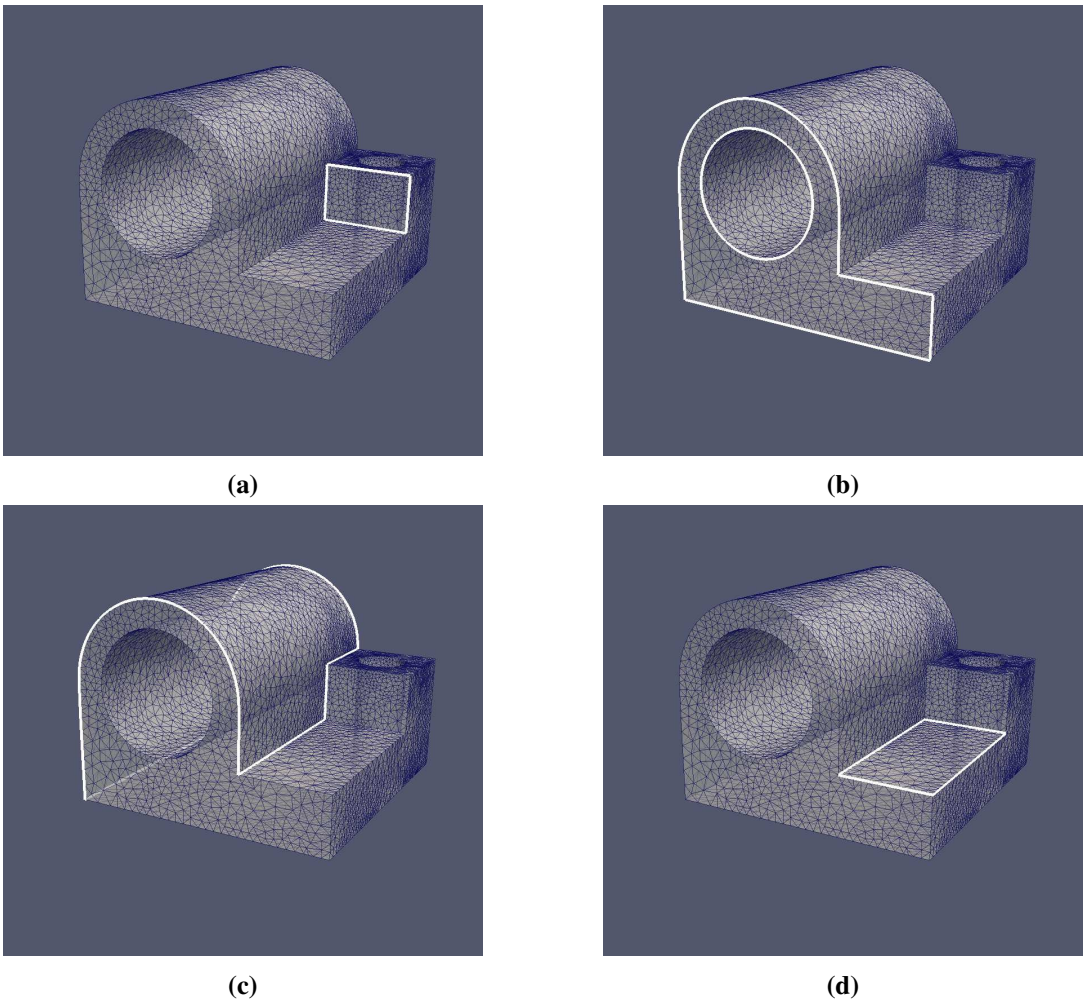


Figure 2.3: An example input triangulation of an arbitrary mechanical part. Four of the segmented surfaces are outlined.

2.2.1 Advancing Layer Initialization

Sharp features of the imported surface are used by CGM to segment the surface. The boundary curves of the segmented surface denote these sharp features. For the purpose of this thesis, we consider these identified boundary curves as the initial front of the mesh. In other words, the boundary curves of the segmented surfaces (or sub-surfaces) serve as the zeroth layer of the advancing layer surface mesh.

Picking the boundary curves of the sub-surfaces to serve as the initial front helps us create anisotropy normal to these boundary curves. This way, desired refinement can be obtained normal to these boundary curves, which is a desirable feature as these boundary curves define the surface features.

The discretization of the boundary curves imported along with the surface triangulation defines the refinement and gradation along the boundary curves (or along the zeroth front). Hence, the refinement and gradation along the boundary curves of the surface is defined by the input triangulation and is up

to the user to vary. The surface mesh generated by EDAM-S hence provides with anisotropy along the normal direction (on the surface) to these boundary curves of the sub-surfaces.

We chose the boundary curves identified by CGM to serve as the zeroth layer of EDAM-S. However, the initial front could be chosen to be something else without affecting the rest of the mesh generation procedure. For eg. curves along high principal curvature directions on the surface could be added to the zeroth layer of the mesh to get the required anisotropy along highly curved regions of the surface. This is a work in progress and might be added to EDAM-S in the future.

2.3 Point Placement

After importing the surface triangulation, we have a valid underlying surface representation with us. Also, segmented sub-surfaces and their boundaries curves provide us with the boundaries we need to march off of. Each vertex on these boundary curves is extruded in two directions, one each for the sub-surfaces which share the boundary point. To make things simpler, two copies of each boundary vertex (a vertex on a boundary curve) are created and associated with the two sub-surfaces which share the vertex. This untangles the process of generating the surface mesh of the two sub-surfaces, which can now be meshed independently.

Meshing sub-surfaces independently has several advantages. If one sub-surface mesh fails to generate, other sub-surfaces would still continue to generate the advancing layer mesh. Different layer on layer *growth ratios* can be fed for different sub-surfaces, hence providing the flexibility to choose different anisotropy for each sub-surface. Also, parallelisation of the surface mesh generation subroutine would be simpler. From here onwards, the discussion would focus on generating the mesh for a single sub-surface, which is a segment of the complete surface.

The mesh generation routine starts by initializing the front as the boundary curves of each sub-surface. All the boundary points are marked as candidate marching points and form the starting layer in the mesh. The points at the boundary curves of the sub-surfaces serve as the parent points for the first layer inserted into the mesh.

The data structure created to store a vertex on the advancing front of the mesh also stores the edges adjacent to that vertex so as to identify the marching directions. Hence, the extrusion direction or marching direction of a point is obtained from the location of the vertex, its adjacent edges, and the underlying sub-surface which is being meshed. The extrusion direction calculation procedure is explained in detail in the next subsection.

Similar to the vertices on the boundary curves, edges on the boundary curves are also stored in two copies, one for each sub-surface associated with that edge. Each edge on the advancing layer stores the direction into the interior of the sub-surface it bounds with respect to the surface normal. In other words, the edge datastructure stores its orientation relative to the sub-surface associated with it. As the sub-

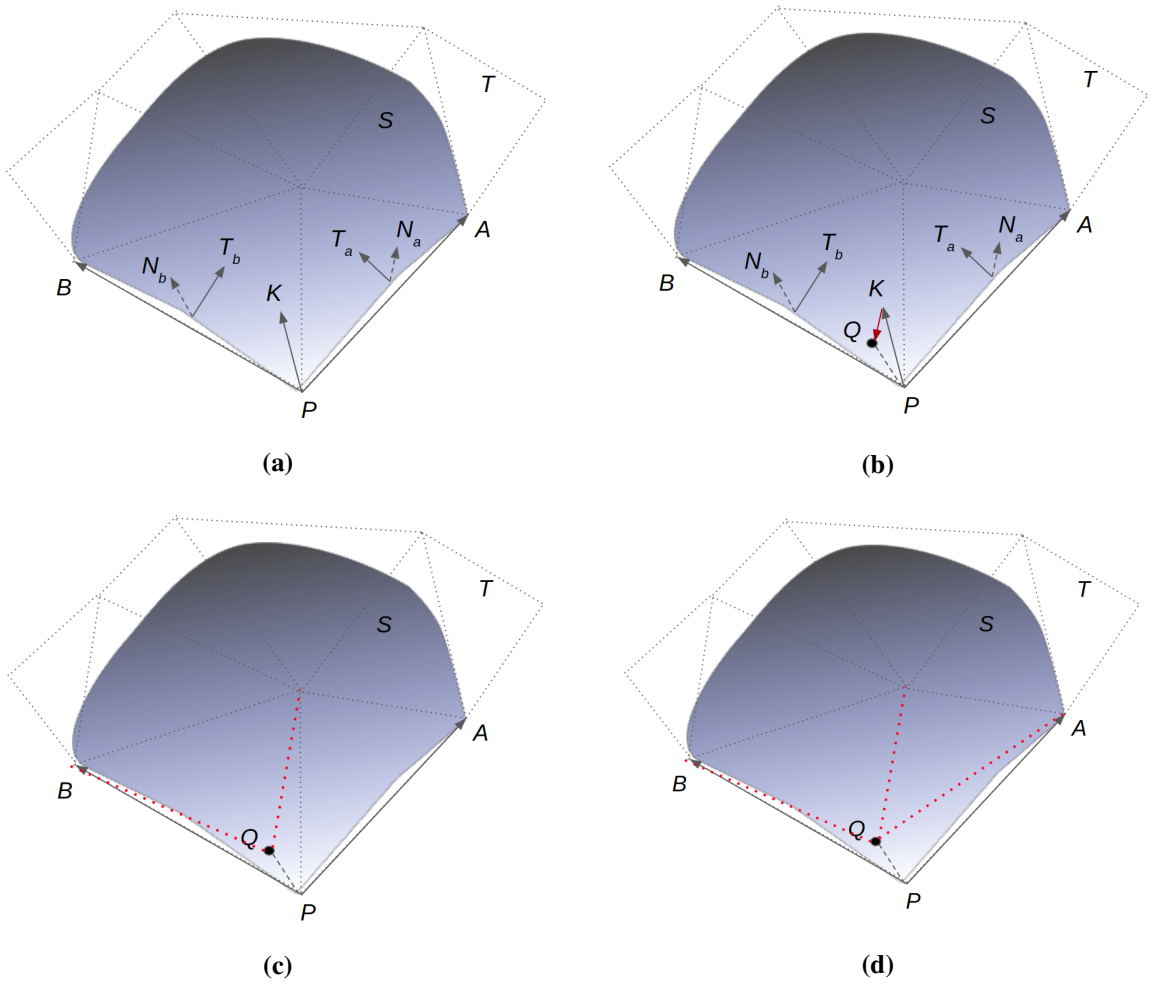


Figure 2.4: (a) Calculation of the extrusion direction of vertex P at the boundary curve of the sub-surface. (b) Projection of the extruded vertex K onto the underlying surface. The projected vertex is marked Q. (c) Insertion of the new vertex Q in the triangulation T. Red dotted lines denote the new edges created in the mesh. (d) One of the edges next to Q is swapped so as to improve the quality of the mesh.

surfaces sharing a common boundary curve are meshed independently, it is easy to identify the normal to an edge along the sub-surface for advancing the front in the mesh generation algorithm.

2.3.1 Extrusion Direction and Length

For each of the sub-surfaces of the geometry, the advancing layer routine iteratively picks a point from its boundary and extrudes it in a given direction. After evaluating the extruded point, we project the point onto the underlying surface. This process is set up to be of two steps for simplicity, accuracy and computational efficiency as will be explained later.

In the first step, we extrude the parent point to get the extruded point. We would interchangeably call the

extruded points as the kid points as they represent the successors of their parent points from the previous layer. The direction of this extrusion is set up to be the average of the normals of the parent vertex's adjacent edges in the tangential plane of the sub-surface. In other words, the normals of the adjacent two edges of a vertex on the underlying sub-surface are averaged to get the extrusion direction.

Consider Figure 2.4a. Initial input triangulation is marked T and the bezier surface representation is marked S. Point P is at the boundary curve of the surface. We need to find the extrusion direction of P to know where its kid would be located. In the underlying triangulation T, PB and PA are the edges adjacent to P on the boundary of the surface S. We first find the mid-points of quartic-Bezier curves PB and PA which are constructed by CGM as a part of constructing the quartic-Bezier triangular surface patches from the underlying triangulation T. Next, we find the normal directions on the surface at these mid-points. The normal directions are carefully queried from the sub-surface which is being meshed currently. The normal directions are labeled as N_b and N_a in the figure. We cross product the vector PB with N_b to get the direction T_b. The vector T_b is normal to the edge PB as well as tangential to the surface S. Similarly, we find the vector T_a, which is normal to the edge PA and tangential to surface S. The direction of extrusion $\overrightarrow{PK}$ is chosen to be the average of the direction of T_a and T_b.

The *initial extrusion length* is an input parameter provided by the user. This length would be taken as the extrusion length when the boundary points are extruded for the first time to the interior of the surface. The initial extrusion length can vary with boundary vertices as the points are extruded independently. Hence, this extrusion length can either be supplied by the user for all the points of the boundary separately or as a single value for all the boundary points. To obtain best quality quad elements, we scale the extrusion length at a given vertex on the advancing layer with respect to the interior angle between the direction vectors T_a and T_b. If the vertex is a concave corner vertex, the extrusion length is increased so as to create good quality quad elements in the next layer. Scaling extrusion length for concave vertices also helps in avoiding immediate front collapse. This process is described in detail in section *ref*.

2.3.2 Vertex Projection and Insertion

After we have extruded the point, we project it on to the underlying sub-surface S, which is considered as the ground truth for the geometry in our mesh generation process. This operation ensures that all the points we insert in the mesh are on the underlying geometry. Errors here would compound in subsequent layers. Hence, exact projection on S is done to avoid accumulation of errors. CGM supports projection of a point in space to the sub-surface. If the projection is outside the sub-surface, the projected point is moved to the closest location on the sub-surface. This is highly unlikely to occur as the extrusion direction is always towards the interior of the sub-surface. Refer to Figure 2.4b for an illustration. Extruded point K is projected onto the sub-surface S so as to get point Q. After obtaining the projected point, a check is done to make sure that the newly obtained point Q is on the surface S by calculating the shortest distance between Q and S. The distance is asserted to be zero for all projections.

Points are inserted in the mesh and the mesh elements are subdivided to include the new point. The candidate point for insertion Q can subdivide an existing triangle to replace a previous triangle with three new ones, or it can subdivide an edge to replace existing two triangles with four new ones. To find the best triangle or edge for subdivision, we first make a guess for the triangle to insert the point. Any triangle in the surface interior adjacent to the point being extruded is chosen as the first guess. Starting from this triangle, we iteratively jump to the best edge or triangle by comparing the barycentric coordinates of the new point with respect to the triangle in consideration. This technique suffers from two disadvantages. First, we need to compare double precision values of barycentric coordinates for making a decision on which triangle to choose for insertion. If the values are too close, the point might be inserted in the wrong triangle and would eventually lead to deviation of the mesh from the underlying surface. Second, the process of iteratively finding the right triangle for insertion might end up being in an infinite loop.

Both of these problems are substantially reduced with a good isotropic initial triangulation. However, we add several validity tests to avoid these problems even for a coarse initial triangulation. These checks include orientation checks of the triangles formed with respect to the surface, thresholding the maximum deviation of the newly formed triangle from the surface and thresholding the dihedral angles between two triangles on the surface. We use an epsilon value of 10^{-5} while comparing the values of barycentric coordinates to zero. Also, we insert the point on a face rather than in a triangle when the ratio of the second-smallest barycentric coordinate to the smallest one is more than a set threshold (10^2). This helps us avoid very skinny triangles with large obtuse angles and also helps in avoiding several unnecessary face swapping in the mesh.

After advancing one layer to the surface interior, we increase the extrusion length at each point by a factor. This factor, called the *growth ratio*, specifies the anisotropic layer-on-layer extrusion length growth as we march on the surface. A value of growth ratio between 1.1 and 1.4 generally gives us satisfactory anisotropy at the boundary curves of the surface.

2.4 Local Reconnection for quality

A mesh observer object is initialized for the mesh data structure. The observer monitors all the changes happening to the mesh. These include the creation and deletion of vertices, edges and cells in the mesh. An edge swapping manager and quality criterion for deciding a swap are also initialized for the mesh. The primary quality criterion we chose for an edge swap is maximization of the minimum angle in the pair of triangles adjacent to the edge before and after the swap. However, only this criterion is not enough for improving surface mesh quality. This is because a better quality pair of triangles may result by edge swapping according to this criterion, however, they might deviate quite a bit from the underlying surface. Hence, we threshold the maximum deviation of the resultant triangles from the surface. This deviation is calculated as the angle between the normal to the triangle and the normal at the centriod of the triangle projected onto the surface.

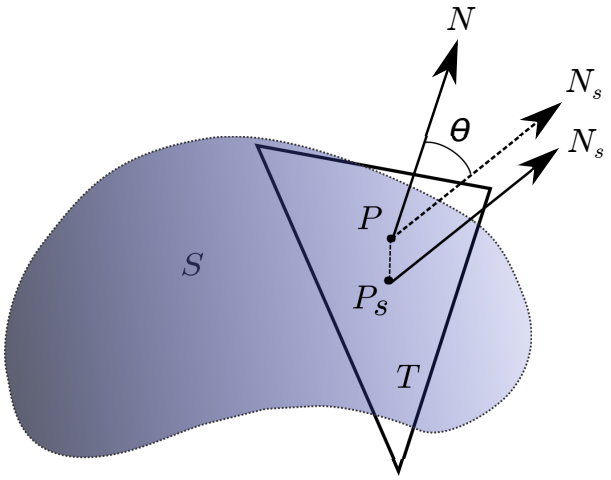


Figure 2.5: Triangle T deviates from the underlying surface S. P is the centroid of T and P_s is the projection of P on S. The deviation is calculated as the interior angle between the normal to the triangle PN and the normal to the surface at P_s, which is P_sN_s. The angle θ represents the deviation here. The deviation is exaggerated for illustration purposes.

Figure 2.5 illustrates the deviation of a triangle T formed as a result of vertex insertion from the underlying surface. Deviation of a triangle T with respect to a surface S is shown. P is the centroid of the triangle and P_s is the projection of the centroid onto the surface. PN is the normal to the triangle and P_sN_s is the normal to the surface at P_s. The interior angle θ is considered as the deviation of the triangle from the surface. An upper bound is set for θ to limit the deviation of resultant triangles from the surface. The bound is set to $\angle 30^\circ$ for the surfaces meshes we are dealing with. This bound is sensitive to the refinement in the initial triangulation. A higher value of θ would be appropriate for a coarse initial triangulation while the value can be set low for a fine initial triangulation.

We queue up the edges of the triangles affected by the insertion of the new point and do edge swapping for these edges to improve on the existing mesh. See Figure 2.4c and 2.4d for an illustration. One of the edges in the mesh is swapped to improve the quality of the triangles surrounding point Q.

Figure 2.6 shows the process of point insertion and local reconnection in two steps in an actual mesh generation process. First, a point is inserted into the mesh at the desired location. This results in subdivision of a triangle as can be seen in figure 2.6b. Subsequently, edge swaps occur to improve the mesh quality. Two iterations of edge swap can be seen in figure 2.6c and 2.6d. We do not swap the edges which comprise the front or the edge between parent and child points. Doing so would invalidate the advancing front. Hence, only interior edges of the surface or the diagonals of the quad elements generated are swapped by the face swapping algorithm. After edge swapping, the algorithm moves on to the next point in the current layer to repeat the same process.

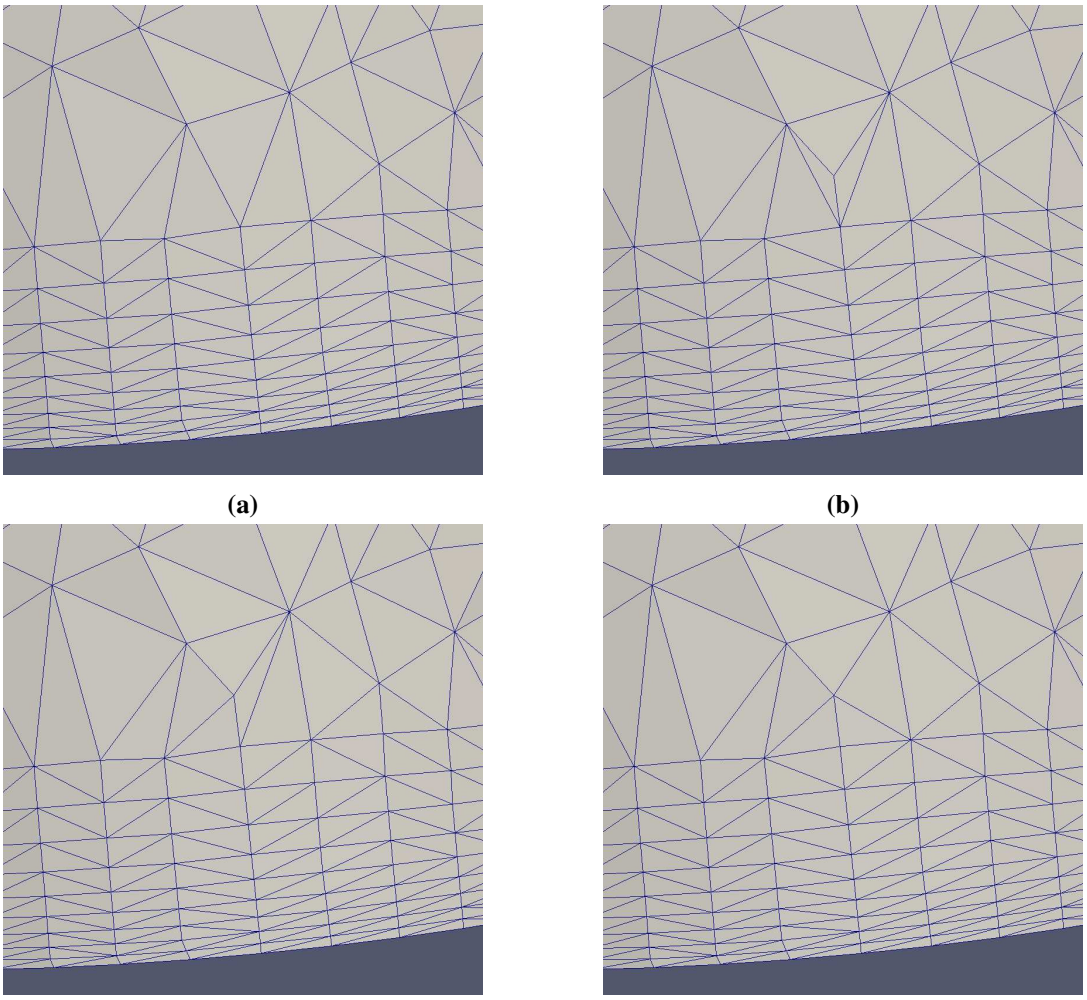


Figure 2.6: Point Insertion and local reconnection for quality shown in four steps. (a) is the initial state of the mesh. A point is inserted in the mesh after extrusion from the parent point, projection onto the surface and finding the right triangle to insert. The triangle is subdivided into three new triangles as shown in (b). To improve the mesh quality after point insertion, swapping is done based on maximization of the minimum angle in a triangle. Two swaps occur as shown in figure (c) and (d) to improve mesh quality.

2.5 Extrusion Length Scaling

The initial extrusion length x_i for the mesh is an input parameter provided by the user. This length, along with the growth ratio g defines the refinement in the direction normal to the boundary curves in the surface mesh. Keeping x_i constant along the boundary curve is one way to proceed with the mesh generation procedure. However, at the concave corners of the mesh, using a uniform extrusion length would result in immediate front collapse as the neighbours of the corner would race towards each other while the corner vertex lags behind in the advancing layers. Hence, some sort of extrusion length scaling is required at the front which takes into account the angle included between the edges adjacent to the

vertex on the front.

Let us consider a concave corner vertex P in the mesh located at an advancing layer. The adjacent edges to P on the layer are PR and PQ. The included angle between these two edges is denoted as 2θ. Let's assume that the length of the edge PQ is l (see Figure 2.7) and the extrusion length for the layer at which vertex P is located as h. We need to find a reasonable extrusion direction and extrusion length for the corner point P.

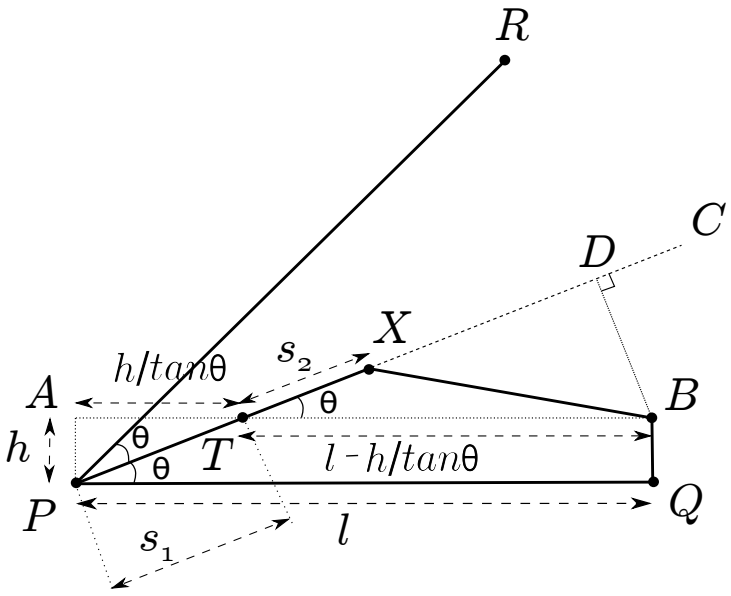


Figure 2.7: Extrusion

For good quad cells in the surface mesh, i.e. quad cells with angles as close to right angles as possible, we need the point P to 'race' a bit initially towards the interior to catch up with its adjacent points. This would be helpful in avoiding mesh collisions. Hence, we formulate a method to find the scaling factor for the extrusion length.

The line PC is constructed at equal angles from edge PR and PQ. Let us place the point X extruded from point P on the line PC. In an orthogonal structured mesh scenario, if the angle between PR and PQ was 180 degrees, the extrusion direction should have been along the edge PA and the extruded point would be at a distance h from the point P, which coincides with point A in the figure. The quad thus formed would have been $PABQ$. However, this is not the case. We have a skewed interior angle between PQ and PR. If we place the extruded point at a distance h from P along PC, the mesh is going to fold onto itself in the next few layers. To avoid this, we take an alternative approach. We place the point X further away from distance h. This distance is calculated using the following strategy.

The area of the quad $PABQ$ would have been $h \times l$ in the orthogonal mesh case. Our strategy is to put the point X on the line PC such that the area of quad $PXBQ$ is also $h \times l$.

Let us assume that the line PC intersects AB at point T. The area of the quad $PTBQ$ is common to both

quads $PABQ$ and $PXBQ$. Hence, to make the area of the quads $PABQ$ and $PXBQ$ the same, we just have to equate the area of $\triangle APT$ and $\triangle XTB$.

$$Area(\triangle ATP) = Area(\triangle TXB) \tag{2.5}$$

Using this equation, we can find the length PX. Let us assume that line segment PT is of length s_1 and TX is of length s_2. Hence, $PX = s_1 + s_2$. s_1 can be easily computed as $h/\sin\theta$. To find s_2, we use the area constraint from equation 2.5. Area of triangles $\triangle ATP$ and $\triangle TXB$ can be easily computed in terms of variables h, l and θ.

$$Area(\triangle ATP) = \frac{1}{2}.h.\frac{h}{\tan\theta} \tag{2.6}$$

$$Area(\triangle TXB) = \frac{1}{2}s_2\left(l - \frac{h}{\tan\theta}\right)\sin\theta \tag{2.7}$$

Solving for s_2 using equations 2.5, 2.6 and 2.7

$$\begin{aligned} \frac{1}{2}\frac{h^2}{\tan\theta} &= \frac{1}{2}s_2\left(l - \frac{h}{\tan\theta}\right)\sin\theta \\ \implies s_2 &= \frac{h}{\sin\theta}.\frac{1}{(l/h)\tan\theta - 1} \end{aligned} \tag{2.8}$$

Finally PX is computed as

$$\begin{aligned} PX &= s_1 + s_2 \\ &= \frac{h}{\sin\theta}\left(1 + \frac{1}{(l/h)\tan\theta - 1}\right) \end{aligned} \tag{2.9}$$

Let l/h be denoted as the aspect ratio AR.

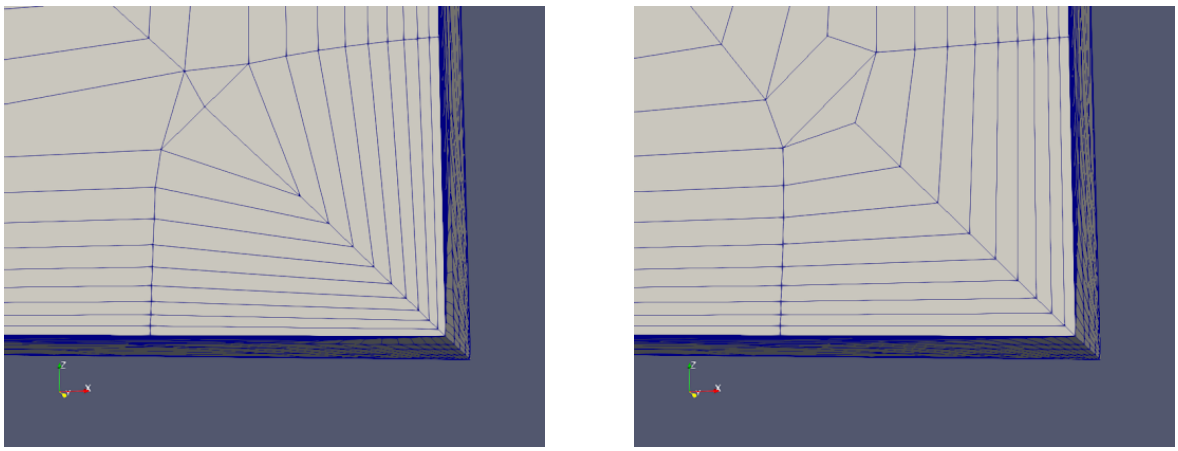
$$PX = \frac{h}{\sin\theta}\left(1 + \frac{1}{AR\tan\theta - 1}\right) \tag{2.10}$$

If the lengths of the adjacent edges PQ and PR are different, we would want to consider both the aspect ratios to calculate the length PX and then average it. This gives us

$$PX = \frac{h}{2\sin\theta}\left[2 + \frac{1}{AR_1\tan\theta - 1} + \frac{1}{AR_2\tan\theta - 1}\right] \tag{2.11}$$

And the extrusion factor f is simply PX/h

$$f = \frac{1}{2\sin\theta}\left[2 + \frac{1}{AR_1\tan\theta - 1} + \frac{1}{AR_2\tan\theta - 1}\right] \tag{2.12}$$



(a) **(b)**

Figure 2.8: Extrusion length scaling is illustrated using a concave corner. In (a), the layers are about to fold onto one another because of constant extrusion length at each layer. (b) shows the vertex at the corner having a larger scaled extrusion length so as to maintain higher quad quality and prevent layer collision.

Hence, the extrusion factor f depends on the half included angle θ and the aspect ratios AR_1 and AR_2 at the vertex. We use a couple of constraints while using this extrusion factor. First, we ensure that $A > cot\theta$, so that we never get negative values for f. Secondly, we limit the extrusion length at any vertex to be less than the minimum of the length of adjacent edges on the boundary. This is done because the extrusion factor can be really large for small angles and small aspect ratios. Hence, to avoid generating skinny elements at concave corners of the mesh, we limit the extrusion length at any vertex such that the resultant aspect ratio at that vertex would always be more than 1. Figure 2.8 shows an example of scaling the extrusion length at the concave corner. Along with preventing layers to fold onto themselves, this technique also helps in improving the quality of quad cells generated in the mesh.

Extrusion length scaling works similar to the principle of hyperbolic meshing, where corners are raced towards the interior of the domain to make the successive layers more rounded. Immediate front collapses are avoided to a great extent for majority of the cases by using such scaling. However, if the corners are quite sharp ($< 10^\circ$) or the refinement on the boundary curves is not sufficient (low Aspect Ratios), front collapses are inevitable. Hence, we implement a separate subroutine to handle such corner collisions in section *ref*.

2.6 Front Recovery

In an advancing layer mesh generation algorithm, an accurate representation of the boundary by anisotropic layers generated from it is very important. Also, generation of quad-elements from triangles with the right aspect ratios is far easier if we retain the boundary representation as we advance multiple layers. Hence, after all points of the parent layer are extruded, we implement a front recovery routine which

recovers all edges between kid vertices.

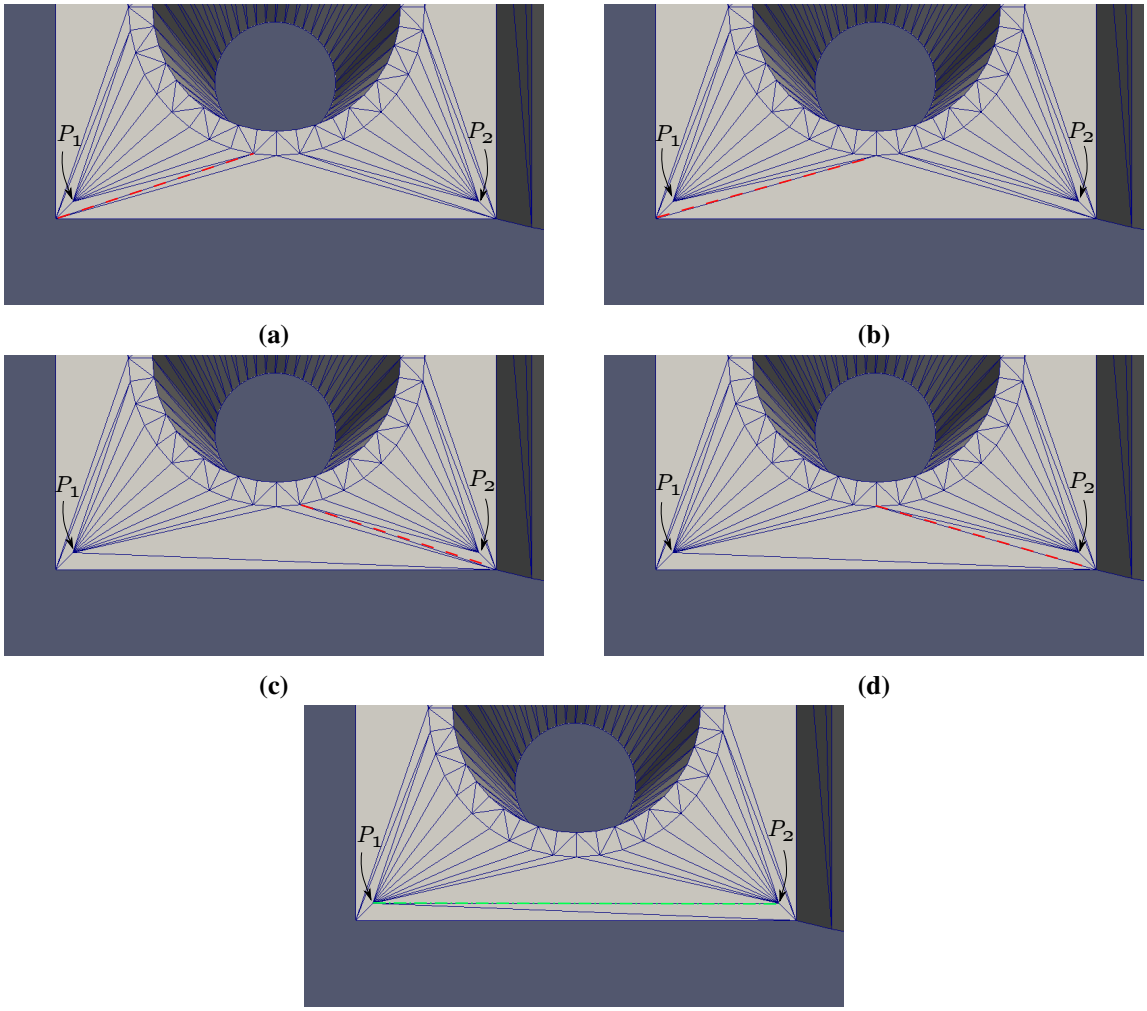


Figure 2.9: Front Recovery: Point P_1 and point P_2 here represent the two kid points generated from the boundary of the surface. We try to force a connection between the two points by iteratively swapping edges which topologically obstruct their connection. Red dashed lines represents the edge chosen to be swapped next. In (e), the green edge is the edge recovered and would serve as a part of the next front in the mesh. Note that this example is for front recovery illustration purposes and the initial boundary discretization is too coarse to get a good-quality advancing layer mesh.

Front recovery is done by selecting a pair of points from the parent layer and then forcing an edge between the points extruded from these parent points; that is by forcing an edge between the kid vertices in the extruded layer. An edge is forced between the kid vertices by iteratively swapping any edge that lies topologically in between the two kid points. This kind of forced swapping in the mesh might seem to decrease mesh quality. However, with the right boundary discretization, it helps in giving us the high-aspect ratio elements in the mesh that we desire.

After forcing an edge between all the adjacent kid vertices, validity checks are run to ensure that the extruded layer is well defined and there are no discrepancies in this layer as it will serve as the parent layer for the next one. This incorporates checking the connectivity of each vertex with its adjacent edges and the connectivity of each edge with its end points. Figure 2.9 shows how exactly we recover an edge in the front by iteratively swapping the edges which obstruct the edge creation between two kid vertices, P_1 and P_2 in the figure. The edges which are topologically obstructing kid vertices P_1 and P_2 are swapped iteratively. A test is setup for each swap to avoid the flipping of triangles on the surface. After all the obstructing edges are swapped, a front edge between the kid vertices P_1 and P_2 would be recovered as shown in green in Figure 2.9e.

Bibliography

[1] 2019 most common 3d file formats. https://all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/. Accessed: 2020-01-12. → pages x, 24

[2] Mesh generation software list. http://www.robertschneiders.de/meshgeneration/software.html/. Accessed: 2010-01-12. → page 25

[3] M. Aftosmis, D. Gaitonde, and T. S. Tavares. On the accuracy, stability, and monotonicity of various reconstruction algorithms for unstructured meshes. 1994. → pages 7, 17

[4] T. D. Blacker and M. B. Stephenson. Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 32(4):811–847, 1991. → pages ix, 7, 8

[5] M. Castro-Díaz, F. Hecht, B. Mohammadi, and O. Pironneau. Anisotropic unstructured mesh adaption for flow simulations. *International Journal for Numerical Methods in Fluids*, 25(4): 475–491, 1997. → pages ix, 11, 12

[6] H. Chen and J. Bishop. Delaunay triangulation for curved surfaces. *Meshing Roundtable*, pages 115–127, 1997. → page 16

[7] J.-C. Cuillière. An adaptive method for the automatic triangulation of 3d parametric surfaces. *Computer-aided design*, 30(2):139–149, 1998. → page 16

[8] E. F. D'Azevedo and R. B. Simpson. On optimal triangular meshes for minimizing the gradient error. *Numerische Mathematik*, 59(1):321–348, 1991. → page 4

[9] P.-J. Frey and F. Alauzet. Anisotropic mesh adaptation for cfd computations. *Computer methods in applied mechanics and engineering*, 194(48-49):5068–5082, 2005. → page 9

[10] R. V. Garimella and M. S. Shephard. Boundary layer mesh generation for viscous flow simulations. *International Journal for Numerical Methods in Engineering*, 49(1-2):193–218, 2000. → pages ix, 12, 14

[11] P.-L. George and H. Borouchaki. Delaunay triangulation and meshing. 1998. → page 16

[12] Y. Ito and K. Nakahashi. Unstructured mesh generation for viscous flow computations. In *IMR*, pages 367–377, 2002. → pages ix, 13, 14

[13] Y. Ito, A. M. Shih, B. K. Soni, and K. Nakahashi. Multiple marching direction approach to generate high quality hybrid meshes. *AIAA journal*, 45(1):162–167, 2007. → pages ix, 15

[14] G. Kunert. Toward anisotropic mesh construction and error estimation in the finite element method. *Numerical Methods for Partial Differential Equations: An International Journal*, 18(5): 625–648, 2002. → page 11

[15] T. Lan and S. Lo. Finite element mesh generation over analytical curved surfaces. *Computers & Structures*, 59(2):301–309, 1996. → pages x, 16, 18

[16] Y. Lee and C. K. Lee. A new indirect anisotropic quadrilateral mesh generation scheme with enhanced local mesh smoothing procedures. *International journal for numerical methods in engineering*, 58(2):277–300, 2003. → page 12

[17] X. Li and W. Huang. An anisotropic mesh adaptation method for the finite element solution of heterogeneous anisotropic diffusion problems. *Journal of Computational Physics*, 229(21): 8072–8094, 2010. → pages ix, 11, 12

[18] R. Löhner. Matching semi-structured and unstructured grids for navier-stokes calculations. In *11th Computational Fluid Dynamics Conference*, page 3348, 1993. → page 12

[19] A. Loseille and R. Löhner. On 3d anisotropic local remeshing for surface, volume and boundary layers. In *Proceedings of the 18th International Meshing Roundtable*, pages 611–630. Springer, 2009. → page 15

[20] D. Mavriplis. Unstructured grid techniques. *Annual Review of Fluid Mechanics*, 29(1):473–514, 1997. → pages 7, 17

[21] D. J. Mavriplis. Adaptive mesh generation for viscous flows using triangulation. *Journal of computational Physics*, 90(2):271–291, 1990. → pages ix, 11

[22] K. Nakahashi. Fdm-fem zonal approach for viscous flow computations over multiple-bodies. In *25th AIAA Aerospace Sciences Meeting*, page 604, 1987. → pages 11, 12

[23] S. J. Owen. A survey of unstructured mesh generation technology. In *IMR*, pages 239–267, 1998. → page 16

[24] N. M. Patrikalakis and T. Maekawa. *Shape interrogation for computer aided design and manufacturing*. Springer Science & Business Media, 2009. → page 23

[25] S. Pirzadeh. Unstructured viscous grid generation by the advancing-layers method. *AIAA journal*, 32(8):1735–1737, 1994. → page 12

[26] O. Sahni, K. E. Jansen, M. S. Shephard, C. A. Taylor, and M. W. Beall. Adaptive boundary layer meshing for viscous flow simulations. *Engineering with Computers*, 24(3):267, 2008. → pages ix, 13

[27] K. Shimada, A. Yamada, T. Itoh, et al. Anisotropic triangular meshing of parametric surfaces via close packing of ellipsoidal bubbles. In *6th International Meshing Roundtable*, pages 375–390, 1997. → page 11

[28] J. R. Tristano, S. J. Owen, and S. A. Canann. Advancing front surface mesh generation in parametric space using a riemannian surface definition. In *IMR*, pages 429–445, 1998. → page 16

[29] J. Tu, G.-H. Yeoh, and C. Liu. Chapter 6 - practical guidelines for cfd simulation and analysis. In J. Tu, G.-H. Yeoh, and C. Liu, editors, *Computational Fluid Dynamics (Second Edition)*, pages

219 – 273. Butterworth-Heinemann, second edition edition, 2013. ISBN 978-0-08-098243-4. doi:https://doi.org/10.1016/B978-0-08-098243-4.00006-8. URL http://www.sciencedirect.com/science/article/pii/B9780080982434000068. → pages 4, 10

[30] N. Viswanath, K. Shimada, and T. Itoh. Quadrilateral meshing with anisotropy and directionality control via close packing of rectangular cells. *world wide web*, 10:12, 2000. → pages ix, 12

[31] S. Yamakawa and K. Shimada. High quality anisotropic tetrahedral mesh generation via ellipsoidal bubble packing. In *IMR*, pages 263–274. Citeseer, 2000. → pages ix, 13, 14

[32] J. Zhu, O. Zienkiewicz, E. Hinton, and J. Wu. A new approach to the development of automatic quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 32 (4):849–866, 1991. → page 7