UNIVERSITY OF CALIFORNIA SAN DIEGO

Automatic generation of global shell-element meshes for large-scale structural design optimization

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Engineering Sciences (Mechanical Engineering)

by

Ning Li

2020

EPIGRAPH

There is only one success – to be able to spend your life in your own way.

—Christopher Morley

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my committee chair Professor John T. Hwang, who is also my Master's advisor. I wish to thank him for encouraging me to think independently on this project and being patient whenever I was stuck at some point. Without his guidance and persistent help this thesis would not have been possible.

I would like to thank the other members of my thesis committee—Professor David Kamensky, and Professor Michael T. Tolley—for sparing time on my Master's defense from their busy schedule. Their invaluable advice improved my thesis and expanded it in different areas.

Next, I would also like to acknowledge Jiayao Yan, Han Zhao, Ru Xiang, Aobo Yang and other members from Professor Hwang's research group for their extensive advice and Juefan Xie and Cheng Qian for their help on initialization of the ideas.

I would like to thank my loved ones, who have supported me throughout entire process, both by keeping me harmonious and helping me putting pieces together. I will be grateful forever for your love.

The thesis, in full, is an adaption of the material as it appears in Li, Ning, and John T. Hwang. "Automatic generation of global shell-element meshes for large-scale structural design optimization." AIAA AVIATION 2020 FORUM. 2020. This thesis is currently being prepared for submission for publication of the material. Li, Ning; Hwang, John T. The thesis author was the primary investigator and author of this material.

VITA

2013-2017	B. E. in Mechanical Engineering, Nanjing Forestry University
2018-2020	M. S. in Mechanical and Aerospace Engineering, University of California San Diego

PUBLICATIONS

Li, Ning, and John T. Hwang."Automatic generation of global shell-element meshes for large-scale structural design optimization." In AIAA AVIATION 2020 FORUM, p. 3135. 2020.

Yan, Jiayao, Ning Li, Tongji Luo, Michael T. Tolley, and John T. Hwang. "Optimal control and design of an underactuated ball-pitching robotic arm using large-scale multidisciplinary optimization." In AIAA Aviation 2019 Forum, p. 3450. 2019.

ABSTRACT OF THE THESIS

Automatic generation of global shell-element meshes for large-scale structural design optimization

by

Ning Li

Master of Science in Engineering Sciences (Mechanical Engineering)

University of California San Diego, 2020

Professor John T. Hwang, Chair

In aircraft design, unconventional configurations have gained significant recent interest due to increasing demands for greater efficiency as well as emerging applications such as urban air mobility. Since traditional empirical models are not applicable for unconventional aircraft, high-fidelity computational design methods like large-scale design optimization can be employed. However, building up models for such unconventional aircraft is time-consuming and requires a large amount of manual work during the iterative design process. Generating a mesh of a structure with complicated geometry is one of the difficult steps. We present a fully automatic approach for constructing quadrilateral meshes for shell-element analysis of aircraft structures. The method is designed to produce highly isotropic meshes while guaranteeing the coincidence of the nodes of the meshes of internal members and the structural skin. This paper also presents the applications of the algorithm on the OWN-06c Transonic Airliner from OpenVSP and the eVTOL eCRM-002 from Uber. The results show that our algorithm is simple, efficient, and versatile, which basically fulfills all the requirements of generating shell element meshes for aircraft.

Chapter 1

Introduction

Large-scale design optimization (LSDO) is the solution of numerical optimization problems in the context of engineering design, where the optimization problem has a large number of design variables, at least in the hundreds. Such an approach is useful because optimization in high-dimensional design spaces yields results that are unintuitive and can provide insights into complex design problems. The high dimensionality can come from a problem that involves multiple coupled disciplines, as is frequently the case in aircraft design. It can also come from a problem in which the quantity being optimized is spatially distributed, such as the sizing of the different parts of a load-carrying structure.

In the field of aircraft design, there is currently a large amount of interest in unconventional aircraft configurations for: commercial aircraft to achieve radical improvements in fuel efficiency and environmental impact; supersonic aircraft to enable their re-introduction into the market; and electric vertical takeoff and landing (eVTOL) in the emerging area of urban air mobility (UAM). With unconventional configurations, traditional empirical and semi-empirical models are inadequate; therefore, high-fidelity computational design methods such as LSDO can play an important role. However, the challenge in applying high-fidelity tools to a conceptual design setting is satisfying the conflicting criteria that: the tool is versatile in that it can consider a wide range of designs with large design changes; there is a small turnaround time so that the engineer can quickly set up, execute, and visualize the computational results; and mesh quality is maintained so that the expected accuracy of the simulation is achieved.

The difficulty of satisfying all of these criteria simultaneously has thus far limited the use of high-fidelity tools early in the design process such as in conceptual design where unconventional configurations are considered. If these criteria can be satisfied, large-scale design optimization has the potential to significantly accelerate the investigation of unconventional configurations for which there is a lack of prior knowledge and empirical models.

The focus of this thesis is on the structural analysis in large-scale design optimization (which can be multidisciplinary). Structures is an important discipline to consider even at the earlier stages of aircraft design because of the importance of reasonably accurate weight prediction. For conventional aircraft, there are many empirical weight models [1] broken down by part (e.g., wing, fuselage, horizontal tail, landing gear) which are functions of various parameters (e.g, area, aspect ratio, sweep, fuselage length, gross weight). These empirical models are very useful because they are simple and easy to use, but they provide enough accuracy to obtain reasonable early estimates of empty weight and mass distribution. For unconventional configurations, we desire structural analysis methods and tools that can provide a similar ability to estimate empty weight and mass distribution while satisfying the aforementioned criteria: versatility with respect to large design changes; small turnaround time; and reasonable mesh quality. High-fidelity structural simulations of airframes are performed using shell-element finite-element analysis.

The generation of a shell-element mesh of an aircraft can be challenging since the internal members of the aircraft affect the mesh of the aircraft's outer skin because the nodes of the meshes of internal members and structural skin must coincide. Jakob et al. [2] introduce a set of interactive brush tools that can be used to control the alignment of the edges in the final mesh, their exact position on the surface, e.g. the location of an imprinted member of an aircraft can not be easily added with brush tools exactly. Hwang et al. [3] develop the GeoMACH tool suite to generate geometry and meshes while allowing the users to determine the layout of the internal structures and keeping the coincidence of intersection nodes between the members and skin. However, GeoMACH has several limitations: it is very time-consuming; its algorithm requires the geometry to be developed within GeoMACH and cannot work with externally created geometries such as CAD files; and it requires a considerable amount of manual work since it subdivides the entire geometry surface into components and generates meshes on these components respectively to combine them into the mesh of the surface [4]. Thus, a simple approach is desired that can generate a high-quality mesh of an airframe without losing automation and robustness. OpenVSP (Open Vehicle Sketch Pad) is NASA's parametrically driven, open-source aircraft geometry design tool, which can be used to generate triangle meshes and quadrilateral wireframes [5]. However, OpenVSP can only generate triangular meshes of thin-walled aircraft structures, and its capability for generating quadrilateral meshes is limited to structured wireframes. Here, we desire a method that can generate fully unstructured quadrilateral meshes of the entire airframe including the skin and internal members of wings, fuselages, and other aircraft components.

In this thesis, we present a novel approach that satisfies all previous requirements using an algorithm whose key steps are a series of simple and robust solutions of linear and quadratic programming problems. Compared to nonlinear programming, linear and quadratic programming are always more robust, simple, and efficient. Our fully automatic mesh algorithm starts with the model of a commercial CAD software, e.g., SolidWorks. For each member, it first defines four intersection curves that become boundaries of the internal member by projecting four line segments out of the aircraft surface to the aircraft model. Then, we apply transfinite interpolation (TFI) to create a continuous parametric surface based on the four boundary curves and generate the initial mesh nodes in the parametric domain [6]. The mesh nodes are later used to generate an initial mesh using Delaunay triangulation, which always has a unique solution in the 2D case. After that, the intersection points between the projected curve and the original triangulation is found and used to reconstruct the triangulation. At last, several mesh quality improvement steps are performed on the meshes of the imprinted members and aircraft skin separately. The mesh quality improvement methods involve splitting, merging, and smoothing optimizations, which are formulated as linear or quadratic programming problems to make topological and geometric adjustments.

The thesis proceeds as follows. In Sec. 2, we survey existing related work in mesh generation of aircraft structures, and in Sec. 3, we demonstrate the details of our algorithm. In Sec. 4, we present the results of applying this algorithm to a model from Vehicle Sketch Pad (OpenVSP). In Sec. 5, we summarize the accomplishments of this thesis and future plans.

Chapter 2

Prior work

A wealth of methods and algorithms have been developed to generate shell-element meshes. For maximizing mesh quality, much of the prior work focuses on minimizing energy to capture smoothness. Some previous studies are able to generate high-quality quadrilateral meshes by minimizing the smoothness energy [2, 7, 8]. Knoppel et al. [9] further generate an algorithm for computing the stripe patterns on a surface which can be applied to create a mesh on the surface. By extending the idea of smooth orientation and position field [2], Huang et al. [10] remove most of the singularities in the output mesh by formulating a minimum-cost network flow problem. However, they are all complicated, and some of them involve nonlinear programming which does not guarantee convergence or global minimum of the objective function. The mesh generated by the algorithm proposed in this thesis may not always be better, but it is more efficient and robust.

In addition to the prior work that focuses on minimizing the smoothness energy, other methods have been developed to generate the mesh or smooth the existing mesh. Parthasarathy et al. [11] develop a constrained optimization method to generate a smooth mesh. Canann et al. [12] use a local optimization approach to improve the topology of unstructured quadrilateral finite element meshes. Some prior work has also been done to use the parameterization of surface patches to generate quadrilateral meshes by initializing the mesh in the parametric domain, then do the smoothing and project the mesh back to physical domain. [13–15]. After the projection, some post-processing steps may be implemented to improve the mesh quality. One drawback of using parameterization to generate meshes is that if the optimization and smoothing are done in the parametric domain, the projected meshes in physical domain may be distorted. Our optimizations are done in the physical domain, thus the mesh quality is preserved. Verma et al. [16] provide a method to produce an isotropic fully-quad mesh using Suneeta Ramaswamy's tree matching algorithm and Guy Bunin's one-defect remeshing. Several studies focus on generating mesh from geometries with gaps or overlaps, or fixing inter-domain boundaries [17–19].

Recent papers focus more on the adoption of existing algorithms and incorporation of the existing algorithms and techniques from the other fields. Some papers adapt the advancing front techniques to generate quadrilateral meshes [20–25]. The advancing front techniques are robust and simple, but it is usually more time-consuming compared to other global mesh generation methods. Rouxel et al. [26] define a discrete Riemannian Voronoi diagram for which its dual is an embedded triangulation. Engwirda et al. [27] offer a robust frontal-Delaunay approach in which new vertices are constrained by element size and the shape of the model.

Otoguro et al. [28] present a method based on multiblock-structured mesh generation and projection of the structured mesh to a NURBS mesh, and recovery of the original model surfaces. Liu et al. [29] combine an octree-based polyhedral mesh generation with the scaled boundary finite element method to perform stress analysis of standard tessellation language (STL) models. The octree structure is a commonly used data structure for 3D models, which can help improve the efficiency of searching desired elements. Some researchers introduce more metrics to the existing algorithms to improve the mesh quality [30, 31]. Soni et al. [32] propose an efficient algorithm by applying centroidal voronoi tessellation on the voxelized Surface. Altenhofen et al. [33] model 3D objects using a volumetric subdivision representation to encode the volumetric information in the design mesh making the mesh generation process much faster. CDT is an extension of Delaunay triangulation by constraining the desired connections of vertices.

What is lacking in the prior work is an efficient way to add internal members to a model, which is one of the motivations of the thesis. The algorithm that we propose as a solution to this is based partly on the algorithm in the aforementioned GeoMACH library [3]. This algorithm divides the aircraft skin into faces and generates a smooth mesh on each face based on a six-step algorithm. However, the generated mesh is considered smooth locally in the parametric domain of each face. The algorithm proposed here can generate smoother meshes since all the optimizations including smoothing are done globally in the physical domain. Another weakness of GeoMACH is that it requires the user to generate the geometry within GeoMACH which is time-consuming. Also, the given CAD geometries cannot be reproduced exactly in GeoMACH. The current algorithm works with input geometries specified by CAD files.

Chapter 3

Methodology

The algorithm begins with a triangulation of the geometry, which would in most cases be generated by commercial CAD software. The triangular mesh can be represented as $\eth = (\nu, \varepsilon, \delta)$, where ν is the set of all vertices associated with a position $\boldsymbol{v}_i \in \mathbb{R}^3$. In addition, ε is the set of edges of the input mesh $(i, j) \in \varepsilon$ if $\boldsymbol{v}_i$ is the neighboring vertex of $\boldsymbol{v}_i$. Similarly, $(i, j, k) \in \delta$ if $\boldsymbol{v}_i$, $\boldsymbol{v}_i$, and $\boldsymbol{v}_k$ forms a triangle on the surface mesh. The neighborhood of vertex i is defined by $\boldsymbol{N}(i) = \{j \in \nu | (i, j) \in \varepsilon\}$.

In our algorithm, each time we define a structural member, four projection points are needed with also the projection directions if necessary (the details of the projection methods are presented in Sec. 3.2.1). The layout of the members is decided by the projection of points, and the input of the projection algorithm are $\boldsymbol{P}$, the set in which all the projection points for each member are stored, and $\boldsymbol{D}$, the set containing all the projection directions of the projection points. For instance, $\boldsymbol{P}_i$ is the set of four projection points for the ith member, and $\boldsymbol{D}_i$ is the set of four corresponding projection directions.

Finally, the output of our algorithm is the connectivity of the fully quad mesh η and updated set of point coordinates ν. If $\boldsymbol{v}_i$, $\boldsymbol{v}_j$, $\boldsymbol{v}_k$, and $\boldsymbol{v}_l$ forms a quad on the quadrilateral mesh, $(i, j, k, l) \in \eta$.

3.1 Algorithm overview

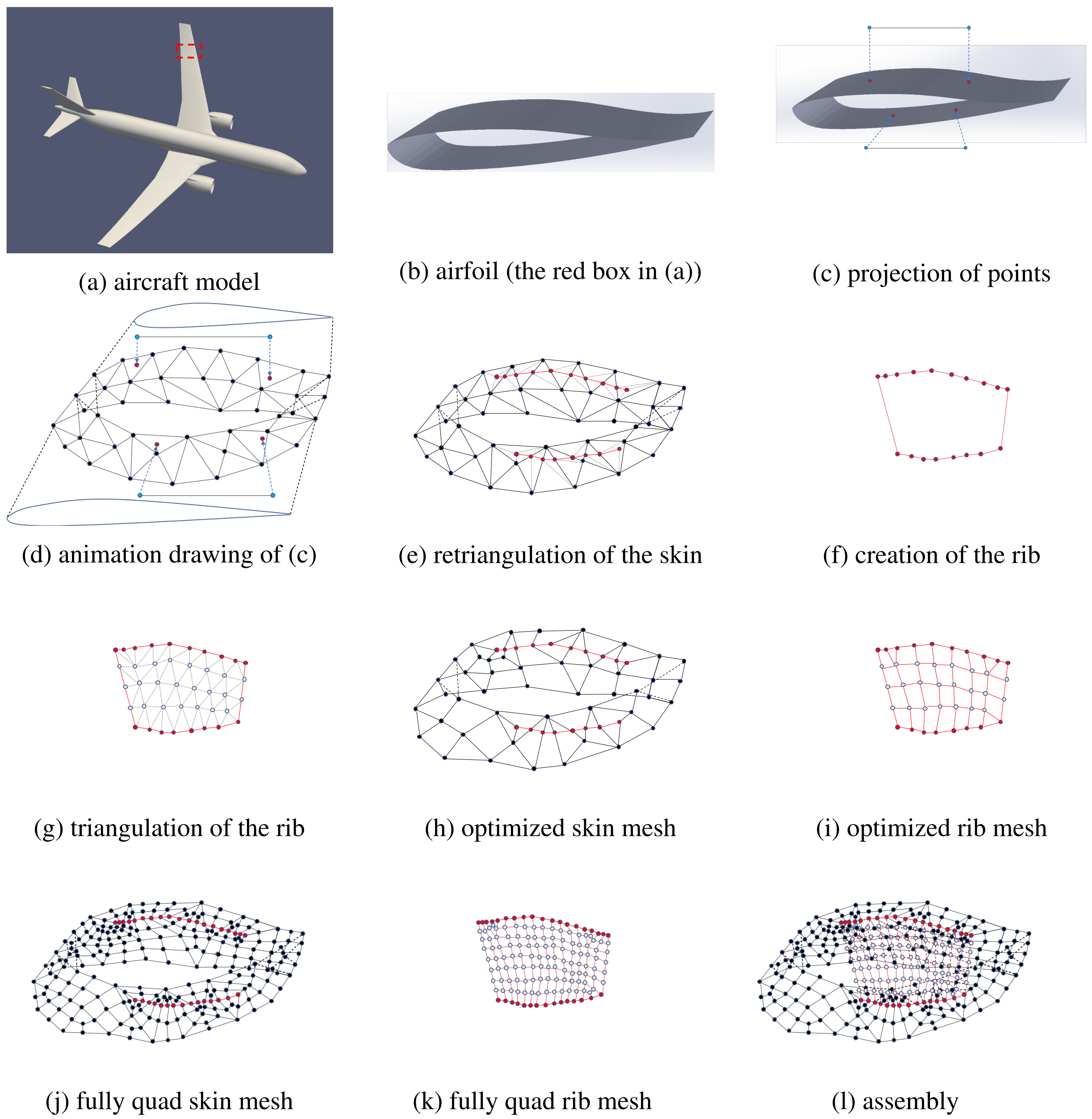


Figure 3.1: An example of applying our algorithm to generate of shell-element meshes for an aircraft structure, where a rib is added to the wing component of the aircraft.

The general flow of our algorithm is in Alg. 1.

Algorithm 1: flow of the overall algorithm

```
Input: $\boldsymbol{P}$, $\boldsymbol{D}$, ν, ε,  δ
Output: η, ν, $\eta_i$, $\nu_i$ ($i$ from 1 to the number of members)
Data: $\boldsymbol{P}$     =  the set in which all the projection points for each member are stored
      $\boldsymbol{D}$     =  the set containing all the projection directions of the projection points
      $\boldsymbol{P}_i$   =  the set of four projection points for the ith member
      $\boldsymbol{D}_i$   =  the set of four projection directions for the ith member
      ν       =  the set of all vertices
      ε       =  the set of all edges
      δ       =  the set of all triangles
      $\nu_i$    =  the set of all vertices for the ith member
      $\delta_i$    =  the set of all triangles for the ith member
      $\boldsymbol{Pd}_i$  =  the projected points for the ith member
      $\boldsymbol{I}_i$   =  the intersection points between ε and the ith member
      $\boldsymbol{S}_i$   =  the four sides of the ith member
      $\boldsymbol{C}$     =  the intersection points between the ith and jth member
      $\mu$     =  the quad dominant mesh after mesh optimizations for the skin
      $\mu_i$    =  the uad dominant mesh after mesh optimizations for the ith member
      η       =  the optimized fully quad mesh for the skin
      $\eta_i$    =  the optimized fully quad mesh for the ith member
1  for $\boldsymbol{P}_i$ in $\boldsymbol{P}$ do
2      $\boldsymbol{Pd}_i \leftarrow$ projection_algorithm($\boldsymbol{P}_i$, $\boldsymbol{D}_i$, ε) ;      // (Fig. 3.1c, 3.1d and Sec. 3.2.1)
3      $\boldsymbol{I}_i \leftarrow$ intersection_detection($\boldsymbol{Pd}_i$, ε);      // (Fig. 3.1e and Sec. 3.2.2)
4      δ and ν $\leftarrow$ retriangulation($\boldsymbol{I}_i$, ν, δ);      // (Fig. 3.1e and Sec. 3.2.2)
5      $\boldsymbol{S}_i \leftarrow$ member_creation($\boldsymbol{I}_i$);      // (Fig. 3.1f and Sec. 3.2.2)
6      $\nu_i$ and $\delta_i \leftarrow$ TFIandDT($\boldsymbol{S}_i$);      // (Fig. 3.1g and Sec. 3.3)
7  end
8  for i from 1 to the number of members do
9      for j from i to the number of members do
10         if intersection between the ith and jth members exist then
11             $\boldsymbol{C} \leftarrow$ intersection_detection($\nu_i$, $\delta_i$, $\nu_j$, $\delta_j$);      // (Sec. 3.4)
12             $\nu_i$ and $\delta_i \leftarrow$ retriangulation($\nu_i$, $\delta_i$, $\boldsymbol{C}$);      // (Sec. 3.4)
13             $\nu_j$ and $\delta_j \leftarrow$ retriangulation($\nu_j$, $\delta_j$, $\boldsymbol{C}$);      // (Sec. 3.4)
14     end
15 end
16 for i from 1 to the number of members do
17     $\nu_i$, $\mu_i \leftarrow$ mesh_optimization($\nu_i$, $\delta_i$);      // (Fig. 3.1i and Sec. 3.4.3)
18     $\nu_i$, $\eta_i \leftarrow$ Catmull($\nu_i$, $\mu_i$);      // (Fig. 3.1k and Sec. 3.4.3)
19 end
20 $\mu$, η $\leftarrow$mesh_optimization(ν, δ);      // (Fig 3.1h and Sec. 3.5)
21 ν, η $\leftarrow$ Catmull(ν, $\mu$);      // (Fig 3.1j and Sec. 3.6)
```

First, to determine the location, shape, and layout of the structural members, we create a geometric class that consists of points and edges for each imprinted member created by the projection of points and retriangulation of the skin of the structure (Fig. 3.1c, 3.1d, and 3.1e).

Then, based on the edges defined in the geometric class, we can determine four boundaries of a member by finding out the edges on the four boundaries (Fig. 3.1f). By interpolating the discrete points on the each boundary curve of the member with B-splines and applying the transfinite interpolation (TFI) technique, we can create evenly distributed points on the interior of the implicit surface of the member [6]. The initial meshes of the internal members are created by utilizing the Delaunay triangulation (DT) method on the discrete points generated by TFI and boundary points in the parametric domain (Fig. 3.1g). After mapping the point coordinates in the parametric domain $(u,\ v)$ back to the physical domain $(x,\ y,\ z)$ and reserving the connectivities of the triangulation, the meshes are ready for further operations. A general and robust intersection involvement method is then applied to the initial mesh to obtain the intersection curves between all the internal members. If not additionally declared, the intersection of all members will be detected. With the intersection curves detected, the triangulation in the initial meshes is modified, and the discrete intersection points are shared and fixed in the following steps to guarantee the coincidence. Afterward, the mesh quality improvement optimizations, including splitting, merging, and smoothing optimizations, are implemented on the meshes of the skin and the internal members separately (Fig. 3.1h and 3.1i). Finally, the quad-dominant mesh is transformed into a fully quad mesh with one step of the Catmull–Clark subdivision (Fig. 3.1j and 3.1k). Therefore, the algorithm is divided into five steps:

1. Creation of surfaces of internal members (Sec. 3.2)
2. Generation of the initial triangulation (Sec. 3.3)
3. Detection and addition of intersection curves (Sec. 3.4)
4. Mesh quality improvement (Sec. 3.5)

5. Transformation to a fully quad mesh (Sec. 3.6)

3.2 Creation of surfaces of internal members

3.2.1 Creation of edges

Employing of the octree data structure

The accuracy and convergence rate of the finite element analysis depends greatly on the mesh quality, mesh resolution, and element order. Thus, with a smaller element size, for the same model, there needs to be more elements which results in more edges and vertices. Especially for structures like aircraft, high-fidelity physics-based structural analyses are vital for accurate results in, for instance, prediction of the weight of each component of an aircraft. To handle the curse of the dimensionality of the number of elements, we use the octree data structure to store all the triangles in the triangulation based on their positions to increase the speed of search, as shown in Fig. 3.2. The octree data structure is widely used to partition a 3D space by recursively subdividing it into eight octants. To put all the triangles in of the initial mesh in the octree data structure, we first generate a cube in space that contains every triangle in the triangulation. The first cube created is called the root node. Then, we evenly subdivide the root node into eight octants by cutting it in half along each axis. Afterward, the triangles are stored in different subnodes based on the locations of their centroids. For each subnode, we subdivide them again recursively until the number of triangles in the node at the maximum level of the octree has less than 200 triangles. These nodes at the maximum level of the octree are called the leaf nodes. As shown in Fig. 3.2, we store the triangulation of a conventional aircraft in an octree. The octree nodes are coarser at the region far away from the aircraft, while as they are closer to the aircraft, the octree nodes are finer.

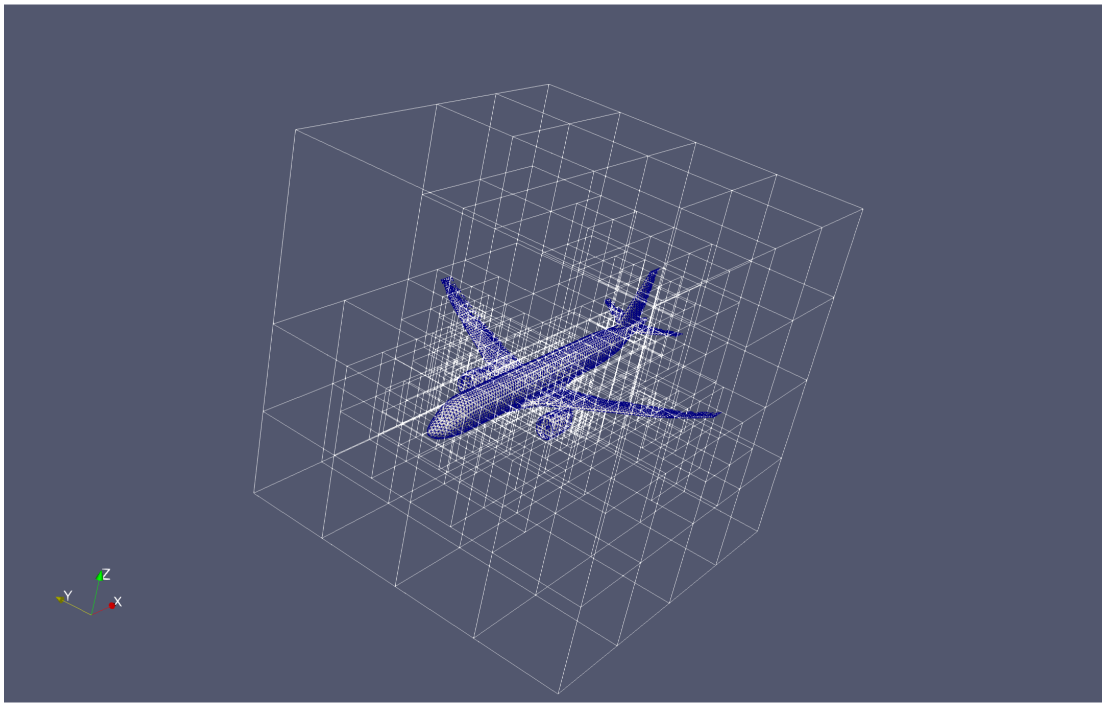


Figure 3.2: The generated octree structure for an aircraft contains the aircraft in it.

Projection algorithm

First, we generate four projection points in space for each member. For instance, to create a rib in the aircraft wing, we first define two points right above the aircraft wing and two points right below the aircraft wing, as shown in Fig. 3.3, which are later connected to make two line segments, one above the wing and one below the wing.

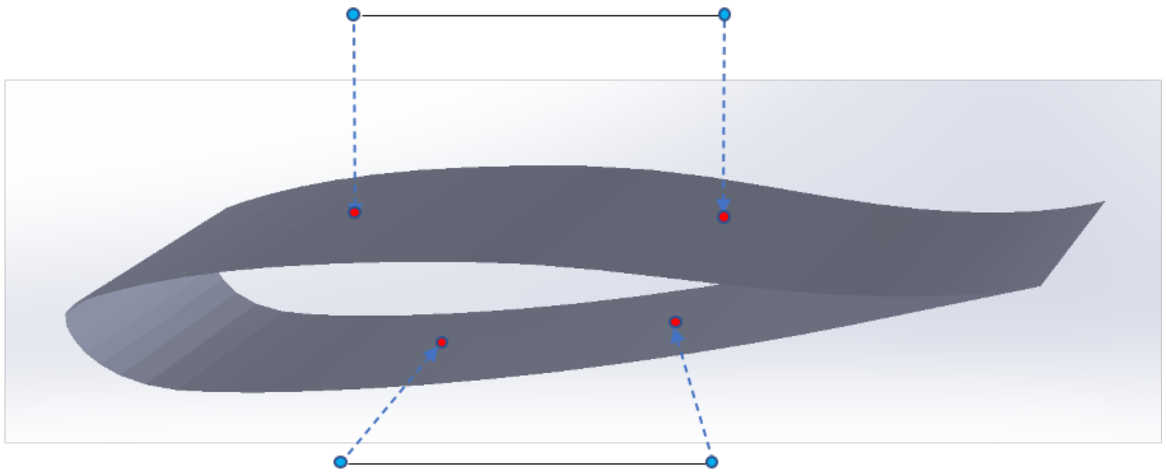


Figure 3.3: Two line segments with four projection points are defined to determine the location and shape of the rib member in the airfoil. The blue points are the projection points, while the red points are the projected points.

Given the line segments that decide the layout of the internal members, we discretize the line segment into several even pieces or maintain the original line segment, depending on the targeted mesh resolution. If the element size of the final output mesh is much smaller than that of the initial mesh, then the line segment must be split into more pieces to fit the finer resolution. Then, for each discretization point $\boldsymbol{l}_i \in \mathbb{R}^3$ with $i \in l$, l is the set of all the points on the line segment. We find the corresponding projected point $\boldsymbol{p}_i \in \mathbb{R}^3$ with i being a member of the set storing all the projected points p. All elements in l and p are ordered so that every two adjacent members in each set are neighboring points in the line segment or projected curve. Two algorithms can be employed to project the discretization points onto the surface of the model.

If the projection directions are given, we can define a ray starting from the point on the line pointing along the projection direction and find the intersection point with the model that gives the shortest distance between the intersection and projection point. Starting from the root node in the octree, [1] we can first determine the probability of the intersection of the ray and

[1] `https://github.com/mhogg/pyoctree`

a node, which is a cubic box in the octree. If the intersection is possible, we loop through all the subnodes and check the occurrence of the intersection. By doing this operation recursively, we end up with a leaf node. Looping through all the triangles in the leaf node and finding the triangles that intersect with the ray, the distances between the triangles and the projection point can be found. Among all the projected points, the one that yields the shortest distance is the point we want. The projection of the line segment below the wing is found with the first projection algorithm and a given projection direction (Fig. 3.3).

If the projection direction is not given, we can then find the point on the triangulation that gives us the shortest distance to the projection point. The algorithm is not too different from the previous one with projection directions. The first step is to start from the root node and iterate over all eight subnodes and then find the one that has the smallest distance to the projection point. If it is a leaf node, we loop through all the triangles in this node and find the projected point and corresponding shortest distance. If it is not a leaf node, we perform the first step recursively until we end up with a leaf node. Then, we need to check the validity of this point from the leaf node level to the root node level. At each level, we need to check if the shortest distance found is shorter than the distance from the projection point to the other nodes. If the found distance is shorter than that between the projection point and the other nodes, go to the upper level and check again until we reach the root node level. If not, we need to find the shortest distance between the triangles in the other node and compare it to the shortest distance found in case that the projected point is in the node that does not yield the shortest distance between it and the projection point among all its counterparts. The line segment above the wing is projected with the second projection algorithm by simply finding the closest points on the triangulation (Fig. 3.3).

After projecting all the designated discretized line segments onto the aircraft, a group of projected edges and a projected curve can be created by connecting the neighbouring projected points, where the projected edges are pairs of neighboring projected points $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$.

3.2.2 Creation of the four-sided domains

Once all the projected points of the discretized points on the line segment are found, we can connect them sequentially and reconstruct the triangulation. For each pair of neighboring projected points $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$, we first calculate the middle point $\boldsymbol{m}_i$ of their corresponding projection points $\boldsymbol{l}_i$ and $\boldsymbol{l}_{i+1}$. Based on $\boldsymbol{p}_i$, $\boldsymbol{p}_{i+1}$, and $\boldsymbol{m}_i$, we can calculate the reverse direction of the projection vector $\boldsymbol{rev}$:

$$\boldsymbol{rev} = \boldsymbol{m}_i - \frac{1}{2}(\boldsymbol{p}_i + \boldsymbol{p}_{i+1}). \tag{3.1}$$

If the projection is done with directions, while the reverse of it can be calculated easily without using Eqn. 3.1, we still calculate the reverse vector in this way because here, the projection may be done without a direction, in which case Eqn. 3.1 cannot be avoided. Once we have the reverse projection direction, to find out all the edges in the triangulation we want to have intersection with, we calculate the normal vector of each triangle. If the dot product of the normal vector and the projection direction is positive, we mark the corresponding triangle and add all its edges to the pool of the possible intersection edges $\varepsilon_{intersect}$. Then, the intersection points of all the edges in the pool and each projected edge can be found easily and inserted between $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$, with their distances to $\boldsymbol{p}_i$ in an increasing order. After inserting all the intersection points between every pair of neighboring projected points, p is updated. To construct the new triangulation, we iterate over all the members in p. If a point is located on the edge of a triangle, we connect it and the opposite vertex in the triangle, and thus two new triangles are added to δ, while the split triangle is deleted from δ. If it is located on the interior of a triangle, we connect it to the three vertices of the triangle, and therefore, in this case, three triangles are added to δ and the split triangle is deleted from δ. A detailed algorithm of how to reconstruct the triangulation is presented in Alg. 2.

After the projected lists are updated with Alg. 2, for a rib, the upper and lower boundary curves can be defined as p projected from the line segment above the wing and that projected from the line segment below the wing. Connecting the starting and ending vertices of the upper

Algorithm 2: reconstruction of the triangulation

```
Data: $p$, $l$, $\nu$, $\varepsilon$, $\delta$
Result: $p$, $\delta$
for $i$ in $p$ except the last element do
    $\boldsymbol{m}_i \leftarrow \frac{1}{2}\boldsymbol{l}_i + \frac{1}{2}\boldsymbol{l}_{i+1}$;
    $\boldsymbol{rev} \leftarrow \boldsymbol{m}_i - \frac{1}{2}(\boldsymbol{p}_i + \boldsymbol{p}_{i+1})$
    for $(x,y,z)$ in $\delta$ do
        $\boldsymbol{n}_{tri} \leftarrow (\boldsymbol{v}_y - \boldsymbol{v}_x) \times (\boldsymbol{v}_z - \boldsymbol{v}_y)$;      // normal vector of the triangle
        if $\boldsymbol{n}_{tri} \cdot \boldsymbol{rev} > 0$ then
            add $(x,y)$, $(y,z)$, and $(z,x)$ to the intersection edges pool $\varepsilon_i ntersect$
    end
    for $(x,y)$ in $\varepsilon_{intersection}$ do
        if $(x,y)$ intersects with the projected edge connecting $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$ then
            $\boldsymbol{coord} \leftarrow$ the intersection point;
            $\boldsymbol{dist} \leftarrow \sqrt{(\boldsymbol{p}_i - \boldsymbol{coord})^2}$;
            save $(\boldsymbol{coord}, \boldsymbol{dist})$ in $\boldsymbol{int}$;
    end
    sort $\boldsymbol{int}$ with $\boldsymbol{dist}$ in the increasing order;
    insert $\boldsymbol{coord}$ from $\boldsymbol{int}$ between $i$ and $i+1$ in $p$;      // update $\boldsymbol{p}$ with
     intersection points
end
add $p$ to $\nu$;
for $i$ in $p$ do
    for $(x, y, z)$ in $\delta$ do
        if $\boldsymbol{v}_i$ locates on the edge of triangle $(x, y, z)$ then
            delete $(x, y, z)$ from $\delta$;
            if $\boldsymbol{v}_i$ lies between $\boldsymbol{v}_x$, $\boldsymbol{v}_y$ then add $(y, z, i)$ and $(i, z, x)$ to $\delta$;
            if $\boldsymbol{v}_i$ lies between $\boldsymbol{v}_y$, $\boldsymbol{v}_z$ then add $(z, x, i)$ and $(i, x, y)$ to $\delta$;
            if $\boldsymbol{v}_i$ lies between $\boldsymbol{v}_z$, $\boldsymbol{v}_x$ then add $(x, y, i)$ and $(i, y, z)$ to $\delta$;
        if $\boldsymbol{v}_i$ locates on the interior of triangle $(x, y, z$ then
            delete $(x, y, z)$ from $\delta$;
            add $(x, y, i)$, $(y, z, i)$, and $(z, x, i)$ to $\delta$
    end
end
```

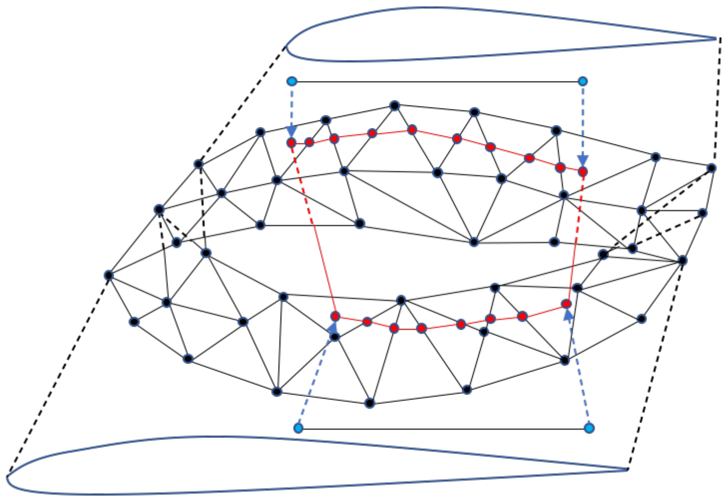


Figure 3.4: The projected points and a four-sided domain (rib) is created. The intersection points between the triangulation and projected edges are denoted as the red points. The black edges are the triangulation.

and lower boundaries, we get the other two boundary curves. With these four boundary curves, the four sides are determined and a surface with four sides can be defined. This is the method of creating a structural member that has intersections with the skin of the model, while for those members that are defined inside an aircraft between other members, like the wingbox and the ribs and spars inside the wingbox, the procedure is exactly the same except that the projection of points is no longer necessary because we can define the four sides by picking points from the other structural members. For instance, the four points of the spar inside the wingbox are directly from the front and rear spars.

3.3 Generation of the initial triangulation

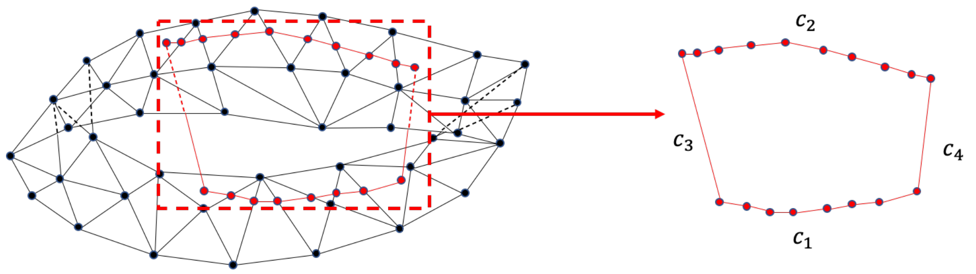


Figure 3.5: The four-sided surface is separated from the triangulation. Starting from the four-sided domain on the right hand side, we generate the triangulation on this member. The four parametric functions of the surface boundaries are denoted as $\boldsymbol{c}_1$, $\boldsymbol{c}_2$, $\boldsymbol{c}_3$, and $\boldsymbol{c}_4$.

As presented in Fig. 3.5, for each member created, we have four boundary curves created by the projection of points. To generate the triangulation of the members, we first need to produce the parametric equations for each boundary. A curve in 3D space is topologically of one dimension; thus, only one parametric coordinate is needed. Take the upper boundary of the rib in Fig. 3.5 as an example. It is represented by a group of points. Taking the point on the left end as the starting point and the point on the right end as the ending point, we assign 0, 1 as the parametric coordinates for the starting and ending points. For the rest of the points between the two end points, the parametric coordinates are computed as

$$u_i = \frac{i-1}{n_p}, \tag{3.2}$$

where i is the index for the point, u_i is the corresponding parametric coordinates, and n_p is the total number of the points on the boundary. Then we can interpolate the physical coordinates of these points with respect to their parametric coordinate using a second-order B-spline curve, and we get the parametric functions of the upper boundary. Repeating the previous steps for the other three boundaries, we end up with the parametric representation of the four sides and can apply the transfinite interpolation to get a continuous parametric surface based on the four boundary curves and generate the initial mesh nodes in the parametric domain. Delaunay triangulation is employed on the mesh nodes in the parametric domain to create the initial triangulation. At last, we map the mesh nodes back to the physical domain to achieve the initial mesh.

3.3.1 Transfinite interpolation

Transfinite interpolation is a technique that constructs a parametric function of a surface such that it matches the curve functions on the boundary of the surface. A surface in 3D space is topologically two-dimensional. Given that the parametrized curves $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are functions of the parametric coordinate u describing one pair of opposite boundaries of the surface, while $\boldsymbol{c}_3$ and $\boldsymbol{c}_4$, as functions with respect to the other parametric coordinate v, describes the other pair of opposite boundaries. Plugging $\boldsymbol{c}_1$, $\boldsymbol{c}_2$, $\boldsymbol{c}_3$, and $\boldsymbol{c}_4$ into the following equation, we get the parametric function of the implicit surface,

$$\begin{aligned}\boldsymbol{S}(u,v) = &(1-v)\boldsymbol{c}_1 u + v\boldsymbol{c}_2 u + (1-u)\boldsymbol{c}_3 v + u\boldsymbol{c}_4 v \\ &- \Big[(1-u)(1-v)\boldsymbol{P}_{(1,3)} + uv\boldsymbol{P}_{(2,4)} + u(1-v)\boldsymbol{P}_{(1,4)} + (1-u)v\boldsymbol{P}_{(2,3)}\Big],\end{aligned} \tag{3.3}$$

where $\boldsymbol{S}$ is the function of the surface with respect to u and v, and $\boldsymbol{P}_{(a,b)}$ is the intersection of two boundary functions. For instance $\boldsymbol{P}_{(1,3)}$ is the intersection point between $\boldsymbol{c}_1$ and $\boldsymbol{c}_3$, which is the lower left corner shown in Fig. 3.5. With the parametric surface defined, we can generate mesh nodes in the parametric domain that are evenly spaced on the interior and boundaries of the surface, for example, the left and right boundary curves in Fig. 3.5. The number of the points on the boundary is decided by rounding the length of the boundary curve divided by the average element size of the triangulation, while the number of the points on the interior of the surface is decided by the number of discretized points on the boundaries. For instance, if there are four points on each side of the surface, the number of the points on the interior will be four, which constructs a two by two frame. The generated mesh nodes for the rib in Fig. 3.5 is here:

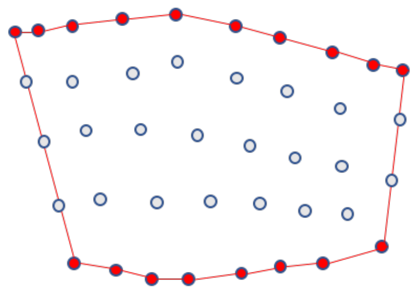
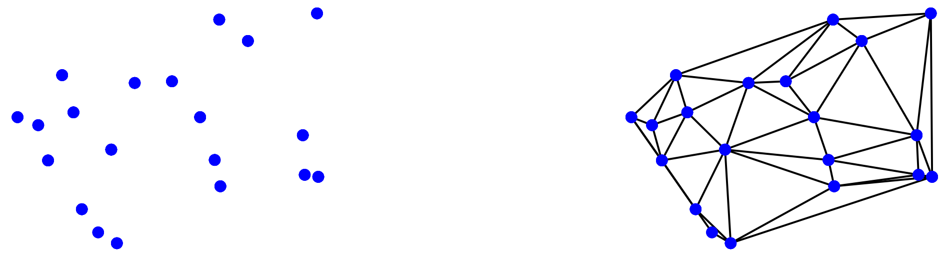


Figure 3.6: The generated mesh nodes are denoted in grey. Since the right boundary curve is shorter than the left boundary curve. There are less discretized points on the right boundary curve.

3.3.2 Delaunay triangulation

Delaunay triangulation is ubiquitously used in in mathematics and computational geometry because its useful geometric properties.



(a) input (b) Delaunay triangulation

Figure 3.7: Given a set of discrete points, Delaunay triangulation generates triangulation that the circumcircle of each triangle is empty.

The fundamental property is the empty circumcircle criterion in the case of 2D triangulation. For a set of points in 2-D, a Delaunay triangulation of these points ensures the circumcircle associated with each triangle contains no other point in its interior. It also maximizes the minimum angle of all the angles of the triangles in the triangulation to avoid a triangle with one or two extremely acute angles, hence a long/thin shape, increasing the quality of the resultant triangular mesh. Since our surface is in 3D space and not flat enough to be considered a plane, we decide to do Delaunay triangulation in the parametric domain.

Because any 3D surface in the parametric domain is a square and a square is convex, there is no need to do constrained Delaunay triangulation (CDT) to ensure the proper connection of the boundary curves. Once the it is done, we map the vertices back to the physical domain, and that is the initial mesh. Although the triangulation after mapping may not still satisfy all the criteria of Delaunay triangulation, in most cases, as the members in the aircraft are mostly flat, the quality of the mesh is preserved. The results of the rib shown in Fig. 3.6 is below:

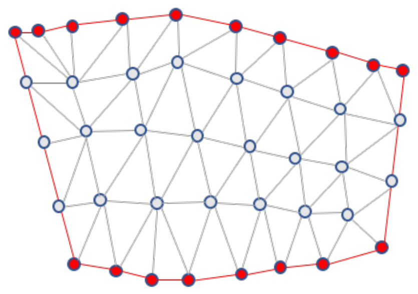


Figure 3.8: The initial triangulation of the rib shows good quality. However, the triangles at the left top corner of the surface have acute angles, Delaunay triangulation cannot deal with that. We will fix this problem in the following steps.

3.4 Detection of intersections

As it should be and because we create many members in the aircraft, the intersection of different members is inevitable. We must make sure that for two members that intersect, the intersection curve must be added to their meshes and shared by both. However, there are some cases where we do not want to have intersections between specific members, and then the intersections between these members are ignored. For instance, to transfer the bending loads from the wings to the ribs and spars, we usually have stringers in the wingbox. To make sure these stringers have no intersection with the ribs in the wingbox, engineers add elliptical holes on the ribs to let the stringers go through. Instead of making cutouts on the ribs, the intersections between the ribs and the stringers are ignored and are not detected. Thus, the general procedure for updating the triangulation is that we first loop through every pair of members and find the intersection curves, omitting the specific members we do not want intersecting. Then, we add the detected intersection curves to the corresponding members iteratively. And last, we will fix these points on the intersection curves in the following steps to make sure the intersection curves still coincide after all the operations.

3.4.1 Detection of intersections between triangles

Because the meshes of the members are all triangular, we need to detect intersections between two triangulations. Essentially, the intersection between triangulations is the intersection between triangles. There are two ways that triangles intersect as shown in Fig. 3.9. In either case, only one intersection line can be found there has only two end points on the edges of these two triangles. Thus the mission of finding the intersection curve between two triangulations is simplified to finding all the intersections between a group of triangle pairs, in which the two triangles come from two different triangulations. Because the end points are on the edges, the problem now becomes finding whether one edge intersects with a triangle and, if so, where the intersection point is. Let $\boldsymbol{t}_1$, $\boldsymbol{t}_2$, and $\boldsymbol{t}_3$ denote the triangle, and the two end points of the edge are $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$. The signed volume of a tetrahedron can be computed as

$$SV(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c},\boldsymbol{d}) = \frac{1}{6}\cdot[(\boldsymbol{b}-\boldsymbol{a})\times(\boldsymbol{c}-\boldsymbol{a})]\cdot(\boldsymbol{d}-\boldsymbol{a}), \tag{3.4}$$

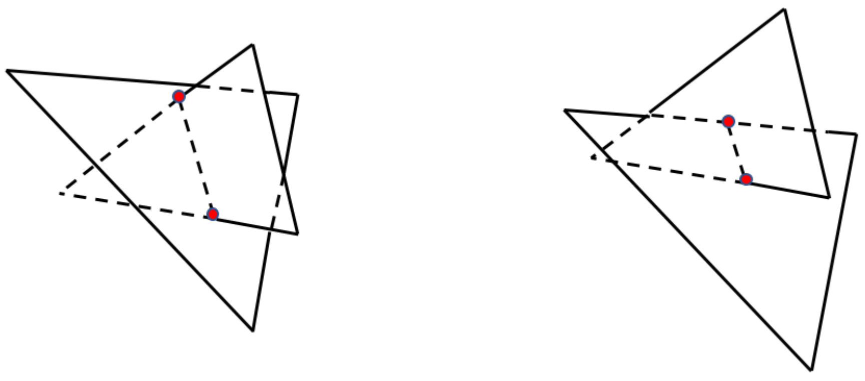

(a) (b)

Figure 3.9: In subfigure (a), both the endpoints of the intersection line are on the edges of the same triangle, while in subfigure (b), the endpoints of the intersection line locate on the edges of different triangles. The red points are the endpoints of the intersection line.

where $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{d}$ are four points of a tetrahedral. Two criteria must be satisfied to make sure the edge intersects the triangle:

$$\begin{aligned} &\textbf{The 1st criterion: } SV(\boldsymbol{q}_1,\, \boldsymbol{t}_1,\, \boldsymbol{t}_2,\, \boldsymbol{t}_3)\cdot SV(\boldsymbol{q}_2,\, \boldsymbol{t}_1,\, \boldsymbol{t}_2,\, \boldsymbol{t}_3) \geqslant 0 \\ &\textbf{the 2nd criterion: } SV(\boldsymbol{q}_1,\, \boldsymbol{q}_2,\, \boldsymbol{t}_1,\, \boldsymbol{t}_2)\textbf{, } SV(\boldsymbol{q}_1,\, \boldsymbol{q}_2,\, \boldsymbol{t}_2,\, \boldsymbol{t}_3)\textbf{, and } SV(\boldsymbol{q}_1,\, \boldsymbol{q}_2,\, \boldsymbol{t}_3,\, \boldsymbol{t}_1) \\ &\qquad\qquad\qquad\quad \textbf{must have the same sign.} \end{aligned} \tag{3.5}$$

If only the first criterion is satisfied, the edge does not intersect with the plane where the triangle is located, as shown in Fig. 3.10a. If only the second criterion is satisfied, the edge intersects with the plane but does not intersect with the triangle, which can be seen in Fig. 3.10b. If and only if both criteria are met, the edge has intersections with the triangle.

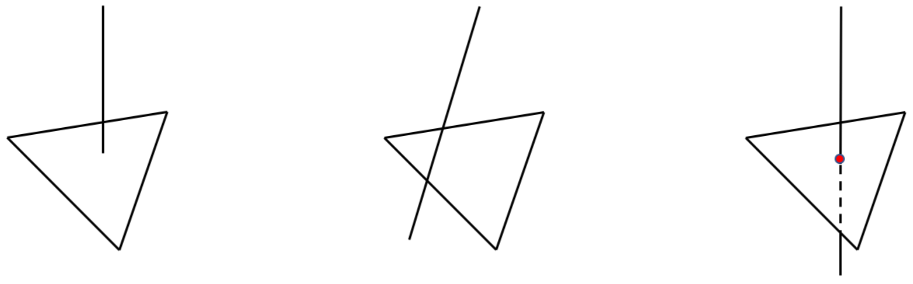

(a) only the first criteria (b) only the second criteria (c) both criterion

Figure 3.10: There are three different cases of a line segment intersecting a triangle. In cases (a) and (b), the triangle and the line has no intersection. If only both the criteria are met, there is intersection between the line segment and the triangle.

3.4.2 Algorithm for finding the intersection points

Once we figure out that the intersection point exits, we can figure out the location of the intersection point. First of all, the normal vector $\boldsymbol{N}$ of the triangle is computed as

$$\boldsymbol{N} = (\boldsymbol{t}_2 - \boldsymbol{t}_1) \times (\boldsymbol{t}_3 - \boldsymbol{t}_2). \tag{3.6}$$

The parametric form $\boldsymbol{l}(u)$of the line segment is

$$\boldsymbol{l}(u) = \boldsymbol{q}_1 + u \cdot (\boldsymbol{q}_2 - \boldsymbol{q}_1), \tag{3.7}$$

where u is the parametric coordinate, and the parametric and physical coordinates $(\hat{u})$ and $\hat{\boldsymbol{q}}$ of the intersection point can be calculated using

$$\begin{aligned} \hat{u} &= -\frac{(\boldsymbol{q}_1 - \boldsymbol{t}_1) \cdot \boldsymbol{N}}{(\boldsymbol{q}_2 - \boldsymbol{q}_1) \cdot \boldsymbol{N}}, \\ \hat{\boldsymbol{q}} &= \boldsymbol{q}_1 + \hat{u} \cdot (\boldsymbol{q}_2 - \boldsymbol{q}_1). \end{aligned} \tag{3.8}$$

After we find all the intersection points between two triangulations, we can group them into a list of points and update the triangulations to include this list of points.

3.4.3 Retriangulation

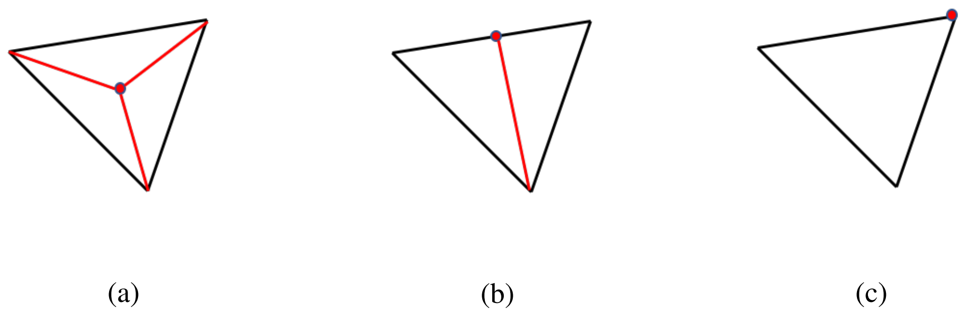
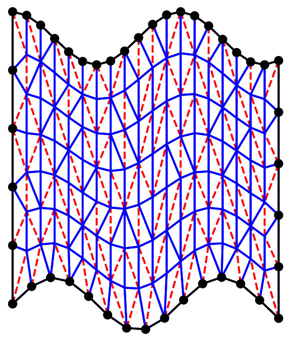


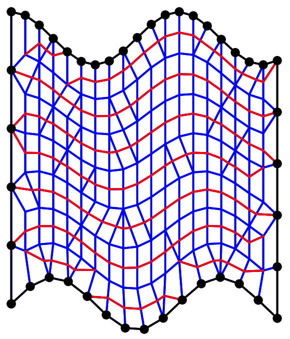
Figure 3.11: There are three cases where the intersection points locate, on the interior of the triangle, on one edge of the triangle and on the vertex of the triangle. The red point is the intersection point and the red edge is created to divide the triangle.

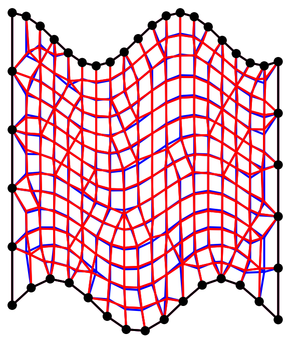
To update the triangulation, we first find out which triangle the intersection points are located in by looping through all the triangles in the triangulation. As shown in Fig. 3.11, if the intersection point stays on the interior of the triangle, we split the triangle into three subtriangles by connecting the intersection point and all three vertices in the triangle. If the intersection point stays on the edge of the triangle, we connect the point and the opposite vertex. If the intersection point stays exactly on a vertex of the triangle, nothing needs to be done.

3.5 Mesh quality improvement

In this section, three optimization problems are formulated to improve the mesh quality, in which the splitting and merging optimization are the topological optimization, and the smoothing optimization is the geometric optimization, as shown in Fig. 3.12. By applying the splitting and merging optimization to the mesh, the number of vertices in the mesh may increase, and the total number of polygons in the mesh also changes, which is caused by splitting a quad or a triangle and merging two adjacent triangles. On the other hand, the smoothing optimization does not change the number of vertices or connectivities of polygons but moves the positions of the vertices. After each smoothing optimization, the updated locations of all vertices are projected back to the original triangulation to maintain the geometry of the structure. All the optimization methods aim to decrease the aspect ratio of each polygon in the mesh or make the polygons more regular.


(a) merging optimization


(b) splitting optimization


(c) smoothing

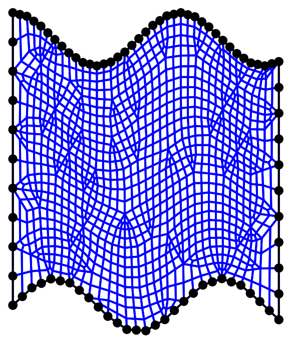

(d) fully quad transformation

Figure 3.12: The black line represents the fixed edges. In (a), the red dashed lines are removed edges from the blue mesh. In (b), the red lines are splitting lines for the blue original mesh. In (c), the blue edges are the input edges, red edges are the smoothed edges. In (d), it is transformed to a fully quad mesh.

3.5.1 Splitting optimization

The meshes created by commercial CAD software are usually structured. For structures with complex geometry, like aircraft, engineers often split the entire model into several components, for instance, fuselage, wings, and tail, and build these components individually. This is efficient when it comes to building up the model. However, the extracted shell-element meshes may have bad quality because the CAD software tends to generate meshes for each component and merge them, which sometimes results in triangles with large aspect ratios at the regions where different components intersect. Similarly, in the triangulation and detection of intersection steps, when we involve the intersection curve to these corresponding members, we, somewhat in a brutal-forced way, connect discretized points on the intersection curve to the existing triangle in which these projected points are located, which also results in some poor quality triangles. To deal with these irregular triangles and also expand the idea to a quad element, the splitting optimization is derived to allow triangles and quads to be split into a triangle and a quad or two triangles or two quads, thus making the polygons in the mesh more regular. For instance, if we split an obtuse triangle into two acute triangles along the median of its obtuse angle, the resultant two triangles clearly have a lower aspect ratio and more regular shape, and thus, they are higher quality. This also works for a quad. If we split a 60° rhombus into two equilateral triangles, the resulting triangles triangles obviously have higher quality because, in the initial rhombus, the angles of the corners are 30° off to 90°, but equilateral triangles are the most regular triangles.

Objective functions and design variables

We form the splitting optimization problem as a linear programming problem for robustness and efficiency. The detailed expression is below:

$$\begin{aligned} \textbf{minimize} \quad & f_s = \boldsymbol{c}_s^{\intercal} \boldsymbol{x}_s \\ \textbf{with respect to} \quad & \boldsymbol{lb}_s < \boldsymbol{x}_s < \boldsymbol{ub}_s \\ \textbf{subject to} \quad & \boldsymbol{A}_s \boldsymbol{x}_s = \boldsymbol{b}_s, \end{aligned} \tag{3.9}$$

where the design variable $\boldsymbol{x}_s$ has the size of eleven times the number of polygons in the mesh, eleven being the number of splitting options. The lower and upper bounds $\boldsymbol{lb}_s$ and $\boldsymbol{ub}_s$ are usually 0 and 1; however, in some cases the $\boldsymbol{ub}_s$ may be constrained to 0. The equality constraints are built up with two criteria, agreement on splitting the shared edge and only one option chosen for one polygon. More details are presented in the following paragraphs.

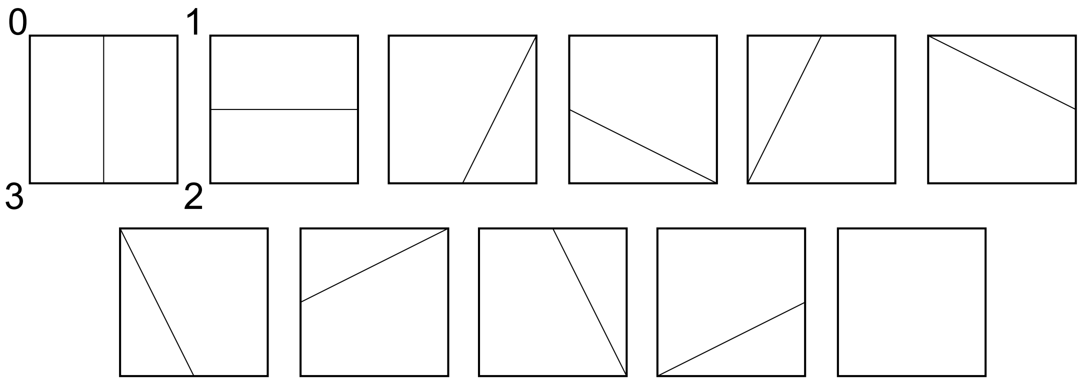


Figure 3.13: Details of the quad splitting: There are eleven options to split a quad, the last one being the original configuration, not splitting at all. The number surrounding the top left quad (option 0) is the local indexing of the quad. From the top left to the bottom right, the options are indexed from 0 to 10.

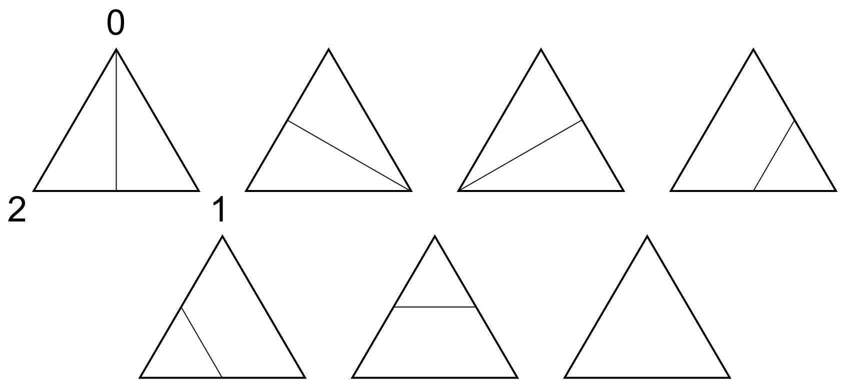


Figure 3.14: Details of the triangle splitting: There are seven options to split a triangle, the last one being the original configuration, not splitting at all. The number surrounding the upper left triangle (option 0) is the local indexing of the triangle. The options are indexed from 0 to 6.

As shown in Fig. 3.13, the eleven options to split a quad can be divided into three types, splitting by a line connecting middle points of two opposite edges, splitting by a line connecting a vertex and the middle point of one opposite edge, and not splitting. Similarly, as shown in Fig. 3.14, the triangles can be split in the same ways, but there are fewer options. Because we want to make our polygons as regular as possible, we define energy describing the regularity of the mesh based on the aspect ratio of each polygon and the difference between the angle in each triangle or quad and 60 or 90 degrees. The expression is

$$e_s = c_1 \sum_{i=0}^{n} \frac{l_i}{l_{avg}} + c_2 \sum_{i=0}^{n} \frac{\theta_i}{\theta}, \tag{3.10}$$

where e_s is the energy, n is the number of edges of a polygon, l_i is the length of each edge, l_{avg} is the average length of all the edges in this polygon, θ_i is the angle of each corner, c_1 and c_2 are the coefficients for energy regarding edge lengths and angles, and θ is the reference angle, 60 degrees for triangles and 90 degrees for quads. The weight, which reflects how much we prefer an given option, can be calculated as the total energy after the splitting operation. For instance, if we split a quad into two new quads, the weight is calculated as the sum of the energy of two new quads. After calculating the weights of all the options, we group them as a vector and assemble it to $\boldsymbol{c}_s$

in Eqn. 3.9. There are fewer options for splitting a triangle, but we still assign eleven options to it because we want to make the number of options the same for a triangle and a quad to make the assembly of $\boldsymbol{c}_s$ easier. Thus, we need to change the upper bounds of the last four triangle splitting options to zero for validity. The computation of the weight of each option of a triangle is the same as that of a quad. The design variable $\boldsymbol{x}_s$ is the decision of each option for all polygons, and thus, its length is eleven times the number of polygons.

The first equality constraint

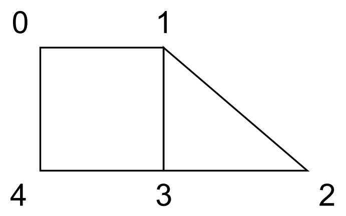


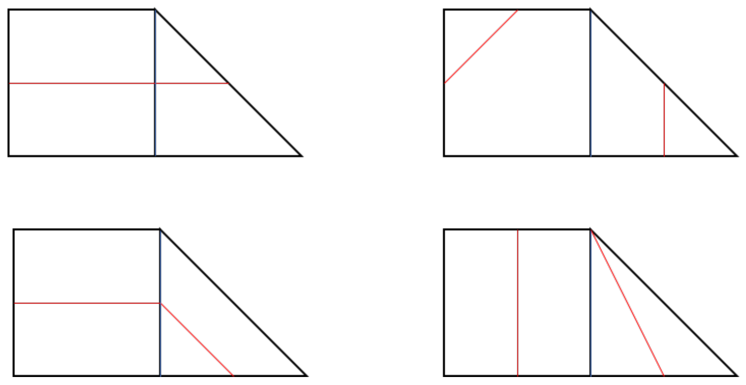
Figure 3.15: An example showing the formulation of the equality constraint: The numbers are the indices for each vertex in the mesh. The quad can be presented as $(0,1,3,4)$ and the triangle can be expressed as $(1,2,3)$. They have one shared edge.

Because every option is in contrast to all other options and the polygon is either split or not split, the equality constraint that can be formed as the sum of all options must be one. As shown in Fig. 3.15, if we take $\boldsymbol{x}_t$ and $\boldsymbol{x}_q$ as the decisions of splitting the quad and triangle, the first linear constraint for this specific case can be formulated as

$$\begin{bmatrix} 1 & \dots & 1 & 1 & 1 & 1 & 1 & \dots & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{x}_q \end{bmatrix} = \begin{bmatrix} 2 \end{bmatrix}, \tag{3.11}$$

where the total length of the first vector is 22, which is the sum of the number of options for both the triangle and quad. The sum of all the options should be 2, one for each polygon.

The second equality constraint


(a) valid splitting

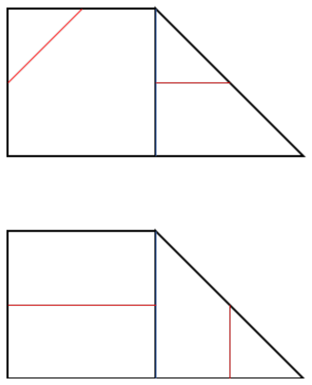

(b) invalid splitting

Figure 3.16: The adjacent polygons must agree on whether to split the shared edge. In subfigure (a), all the splittings are valid, while in subfigure (b), the splittings are invalid since the triangle and the quad do not agree on splitting the shared edge.

Up to this point, we only consider the situation of each polygon individually and overlook their adjacency, so another equality constraint for establishing agreement on whether to split the shared edges of adjacent polygons must be constructed. Take the simple mesh in Fig. 3.15 as an example, where the quad $(0,1,3,4)$ and triangle $(1,2,3)$ have one shared edge $(1,3)$. Because they have the shared edge, they must agree on whether to split the shared edge. For the triangle, the local indices of the shared edge are $(2,0)$ from Fig. 3.14. Options $(2,3,5)$ split the shared edge, and options $(0,1,4,6)$ do not split it. For the quad, the local indices of the shared edge are $(1,2)$, and options $(0,2,6)$ tend to split the shared edge, while options $(1,3,4,5,7,8,9,10)$ do not split it. Since our algorithm can only handle triangular, quad-dominant, and fully quad mesh, if the adjacent polygons do not have an agreement on the decision of splitting the shared edge, a polygon with more than four edges may be created. Thus, the splitting decisions must agree between the quad and triangle, and we can formulate a linear equality constraint for them:

$$\begin{bmatrix} 0\,0\,1\,1\,0\,1\,0\,0\,0\,0\,0 & -1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1\,1\,0\,0\,1\,0\,1\,0\,0\,0\,0 & 0 & -1 & 0 & -1 & -1 & -1 & 0 & -1 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{x}_q \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{3.12}$$

where $\boldsymbol{x}_t$ is the group of decisions for the triangle, and $\boldsymbol{x}_q$ is the group of decisions for the quad, the first row of the first matrix means the triangle and the quad agree to split the shared edge, while the second row means that they agree on keeping the shared edge.

Sometimes, we also have a list of edges that we want to fix during the splitting optimization stemming from the intersection curve of the internal members and model skin or vertices at the high-curvature region. To fix these edges, we find the corresponding edges in the triangulation and set the upper bound of every option that tends to split the fixed edge to zero. Thus, the fixed edge cannot be split during the optimization.

3.5.2 Merging optimization

Merging optimization is designed to merge two adjacent triangles to a high-quality quad to create a quad-dominant mesh. The best part of it is that if the given mesh is a structured mesh, although each triangle may be irregular, the merging optimization can help to transform it into a quad-dominant mesh with quads that are each close to square. The merging optimization can also help decrease the number of edges, with polygons that helps to decrease the computational cost of the following optimization steps. To control the resolution of the final mesh, merging optimization is critical. As like the splitting optimization, the merging optimization problem can also be formulated as a linear optimization problem.

Objective functions and design variables

The detailed expression is presented as

$$\begin{aligned} \textbf{minimize} \quad & f_m = \boldsymbol{c}_m^\intercal \boldsymbol{x}_m \\ \textbf{with respect to} \quad & \boldsymbol{lb}_m < \boldsymbol{x}_m < \boldsymbol{ub}_m \\ \textbf{subject to} \quad & \boldsymbol{A}_m \boldsymbol{x}_m \leq \boldsymbol{b}_m, \end{aligned} \tag{3.13}$$

where the design variable $\boldsymbol{x}_m$ is based on each edge, and its length is the number of edges with the lower $\boldsymbol{lb}_m$ and upper $\boldsymbol{ub}_m$ bounds set at 0 and 1, where 1 is removing the edge, and 0 is not removing it. The inequality constraint stems from the criteria that, at most, one edge per triangle can be removed because allowing multiple edges of a triangle to be merged may result in a polygon with more than four edges. Because our algorithm can only handle triangle, quad-dominant, and fully quad mesh, we do not want that to happen.

As with the splitting optimization, energy is defined as describing the quality of the merged quad. It has three terms, where two are exactly the same as the energy defined for the splitting algorithm, and the third one comes from the penalty of the dihedral angle of the resultant quad, which is

$$e_m = c_1 \sum_{i=0}^{n} \frac{l_i}{l_{avg}} + c_2 \sum_{i=0}^{n} \frac{\theta_i}{\theta} + c_3 \sum_{i=0}^{n} \boldsymbol{e}_i^{\intercal} \boldsymbol{n}_q, \tag{3.14}$$

where c_3 is the coefficient for the third energy regarding the dihedral angle, $\boldsymbol{e}_i$ is the unit vector pointing along each edge of the quad, and $\boldsymbol{n}_q$ is the normal vector of the quad calculated as the cross-product of the unit vectors in the two diagonal directions. The weight is computed by subtracting the energy before merging from the energy after merging. The lower the energy after merging and the higher the energy before merging, the shared edge is more encouraged to be merged. Assembling the weight, $\boldsymbol{c}_m$ can be constructed. The upper bound for an edge shared by two triangles is set at 1, however, if the edge is found to be shared by two quads or a quad and a triangle, there is no way that we can merge them, and thus the upper bound is set to be zero in those two cases. Also, if an edge is fixed, the upper bound for this decision of this edge is zero.

Inequality constraint

To build the matrix $\boldsymbol{A}_m$ and $\boldsymbol{b}_m$ in the inequality constraint, we first loop through all the edges to find the indices for edges in each triangle. Then we end up with a matrix that has a size of the number of triangles by three. By assigning corresponding elements in $\boldsymbol{A}_m$ as one with respect to the generated matrix and setting all the elements in $\boldsymbol{b}_m$, the inequality constraint is constructed.

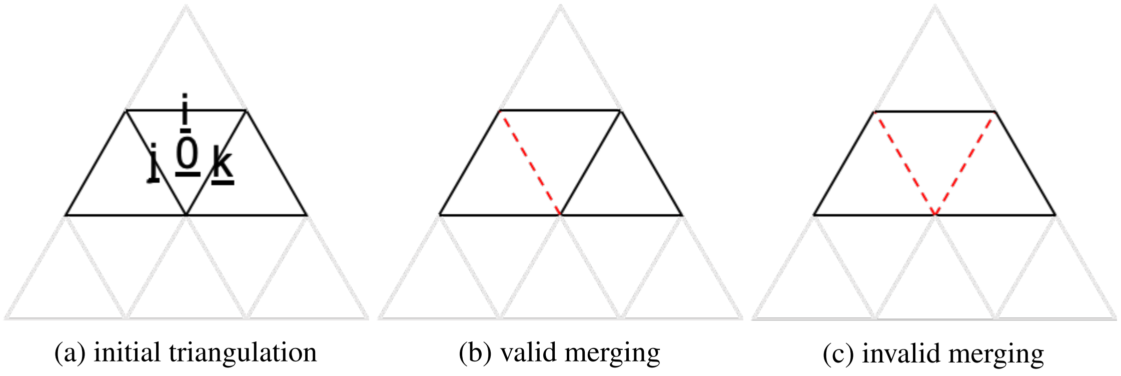


Figure 3.17: Starting with the triangle in the initial triangulation, subfigure (a) shows a valid merging and subfigure (b) shows an invalid merging.

Take triangle 0 from Fig. 3.17 as an example, where i, j, and k are the indices of three triangle edges. The constraint can be formulated as

$$\begin{bmatrix} 0 & \dots & 0 & 1 & 1 & 1 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \vdots \\ \boldsymbol{x}_{i-1} \\ \boldsymbol{x}_i \\ \boldsymbol{x}_j \\ \boldsymbol{x}_k \\ \boldsymbol{x}_{k+1} \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix} \leq \begin{bmatrix} 1 \end{bmatrix}, \tag{3.15}$$

where n is the number of all the edges in the triangulation, and the sum of $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, and $\boldsymbol{x}_k$ must be no greater than one, which means that, at most, one edge per triangle can be removed.

3.5.3 Smoothing optimization

As described before, the splitting and merging optimizations tend to optimize the mesh quality locally. The splitting optimization, although constrained by the criteria that the adjacent polygons must agree on the shared operation, still tends to focus on the quality of each element. Hence, the resultant mesh may have different element sizes in different regions, and the global aspect ratio may be even more significant than that before the splitting optimization. Similarly, in the merging optimization, if the mesh is formed at some regions with large triangles, and at other regions, it consists of small quads, then the global aspect ratio may remain large after the merging optimization. To deal with the problems derived from the splitting and merging optimizations, the smoothing optimization step aims to change the location of each vertex in the mesh to decrease the aspect ratio, thus improving the overall quality. We allow all the vertices to move in its tangential plane within a small range, then minimize the sum of the lengths of all edges with respect to the displacement vector. Finally, we project all the vertices back to the model to maintain the geometry.

The smoothing optimization method can be formulated as a quadratically constrained quadratic programming problem:

$$\begin{aligned} \textbf{minimize} \quad & f_{sm} = \frac{1}{2}[(\boldsymbol{V}_0^{\mathsf{T}} + \boldsymbol{D}^{\mathsf{T}}[\boldsymbol{E}_0]) - (\boldsymbol{V}_1^{\mathsf{T}} + \boldsymbol{D}^{\mathsf{T}}[\boldsymbol{E}_1])] \cdot [(\boldsymbol{V}_0 + \boldsymbol{D}[\boldsymbol{E}_0]) - (\boldsymbol{V}_1 + \boldsymbol{D}[\boldsymbol{E}_1])] \\ \textbf{with respect to} \quad & \boldsymbol{D} \\ \textbf{subject to} \quad & \boldsymbol{m}\boldsymbol{D} \leq \boldsymbol{r} \\ & \boldsymbol{n}\boldsymbol{D} = 0, \end{aligned} \tag{3.16}$$

where $\boldsymbol{D}$ is the displacement for all vertices, $\boldsymbol{V}_0$ and $\boldsymbol{V}_1$ are the initial position of the end points for all edges, $\boldsymbol{E}_0$ and $\boldsymbol{E}_1$ are the indices for the end points of all edges, and $\boldsymbol{V}_0 + \boldsymbol{D}[\boldsymbol{E}_0]$ and $\boldsymbol{V}_1 + \boldsymbol{D}[\boldsymbol{E}_1]$ are the updated position of the end points of all edges. The design variable $[(\boldsymbol{V}_0 + \boldsymbol{D}[\boldsymbol{E}_0]) - (\boldsymbol{V}_1 + \boldsymbol{D}[\boldsymbol{E}_1])]$ is calculated by subtracting the updated ending points from the updated starting points representing the vector along the edges, and the objective function measures the lengths of edges. In the inequality constraint, $\boldsymbol{m}$, as the matrix form of D, is found iteratively assigning three coordinates of each vertex to the corresponding positions in each row. Multiplying $\boldsymbol{m}$ and $\boldsymbol{D}$, we can measure the squared distance that each vertex moves a distance smaller than $\boldsymbol{r}$, and the maximum radius each vertex is allowed to move, which is computed by the distance between the closest neighboring vertex and itself, times a coefficient. In the equality constraint, $\boldsymbol{n}$ is the normal matrix created by assigning the normal vector of each vertex to the associated positions of each row. The equality constraint helps to keep each vertex moving on its own tangential plane by setting the displacement vector vertical to the normal vector, which is calculated as the area-weighted normal of adjacent triangles for each vertex.

If we take the inequality constraint as a regularization term, the QP problem can be simplified to:

$$\begin{aligned}
\textbf{minimize} \quad & f_{sm} = \frac{1}{2}[(\boldsymbol{V}_0^{\mathsf{T}} + \boldsymbol{D}^{\mathsf{T}}[\boldsymbol{E}_0]) - (\boldsymbol{V}_1^{\mathsf{T}} + \boldsymbol{D}^{\mathsf{T}}[\boldsymbol{E}_1])] \cdot [(\boldsymbol{V}_0 + \boldsymbol{D}[\boldsymbol{E}_0]) - (\boldsymbol{V}_1 + \boldsymbol{D}[\boldsymbol{E}_1])] \\
& + \frac{1}{2}\boldsymbol{D}^{\mathsf{T}}\boldsymbol{w}\boldsymbol{D} \\
\textbf{with respect to} \quad & \boldsymbol{D} \\
\textbf{subject to} \quad & \boldsymbol{n}\boldsymbol{D} = 0,
\end{aligned} \tag{3.17}$$

where $\boldsymbol{w}$ is the regularization parameter decided by the local curvature for each vertex, and if some vertices are fixed or the curvature is found to be very high, we set the weight to be an extremely large number to prevent any movement. Eqn. 3.17 can be solved by applying the KKT conditions:

$$\begin{bmatrix} \boldsymbol{B} & \boldsymbol{A}^{\mathsf{T}} \\ \boldsymbol{A} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{D}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{c} \end{bmatrix}, \tag{3.18}$$

where $\boldsymbol{B}$ is the partial derivative of the first term in the objective function with respect to $\boldsymbol{D}$,

$\boldsymbol{A} = \boldsymbol{n}$, $\boldsymbol{D}^*$ and $\boldsymbol{\lambda}^*$ are the optimal displacements and associated Lagrange multiplier, respectively, $\boldsymbol{b}$ is the coefficient of the linear term, and $\boldsymbol{c} = \boldsymbol{0}$.

After updating the vertex positions with the optimal solution, we project the vertices down to the outer surface of the model by iterating over all the triangles and quads in the mesh to find the closest point to the given updated vertex point. The point that yields the shortest distance from a triangle to a point is not hard to find analytically [34], but we have to apply a Newton search to find that for a quad. Since we assume that vertex movement along the tangential plane within a reasonably small range can still be considered 'close' to the real geometry, the regularization term $\boldsymbol{w}$ is set to be large and the vertices are not allowed to go far during each position iteration. Therefore, we do multiple iterations of the smoothing optimization steps at a time, which helps us to preserve the geometry while moving the vertices a measurable amount.

3.6 Transformation to a fully quad mesh

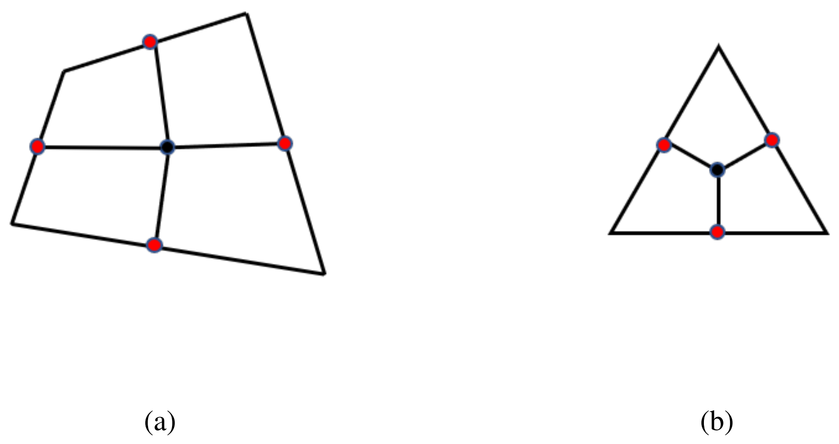

(a) (b)

Figure 3.18: The black points are the face points while the red points are the edge points. Connecting the face points to the edge points, a polygon is divided into quads.

After a series of optimizations, the initial triangulation now becomes a quad-dominant mesh. To transform a quad-dominant mesh to a fully quad mesh, we only need to take one step of the Catmull-Clark subdivision, which first adds a point on each polygon that is the average of all original points for the polygon. Then, we add a point on each edge that is the midpoint of the two end points of the edge. Finally, for each polygon, add edges between the face point and all the edge points of the polygon. For subdivision of members in the structure, the detailed algorithm is in Alg. 3, the same algorithm can be applied on the skin mesh, too. This is the end of one step of the Catmull-Clark subdivision, and given that we only need one step, the rest of the algorithm is not covered here. Shown in Fig. 3.18 are the subdivisions on the structural members.

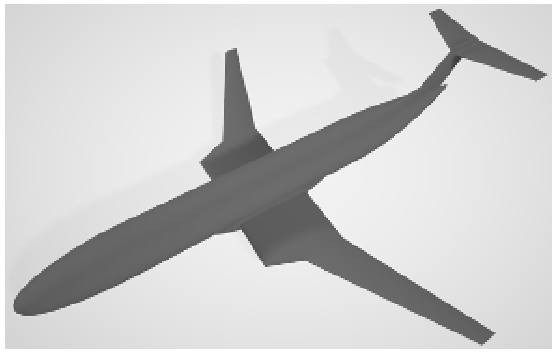
Algorithm 3: one-step of Catmull˙Clark subdivision on the members

```
Data: $\nu_i$ ($i$ from 1 to the number of members)
Result: $\eta_i$, $\nu_i$
for i from 1 to the number of members do
    for j from 1 to the number of subsets in $\eta_i$ do
        if the jth subset $(x, y, z)$ is a triangle then
            insert the face point $f$ and edge points $e_1$, $e_2$, $e_3$ into $\nu_i$;
            insert $(x, e_1, f, e_3)$, $(y, e_2, f, e_1)$, and $(z, e_3, f, e_2)$ into $\eta_i$;
            delete the $j$th set form $\eta_i$;
        if the jth subset $(w, x, y, z)$ is a quad then
            insert the face point $f$ and edge points $e_1$, $e_2$, $e_3$, $e_4$ into $\nu_i$;
            insert $(w, e_1, f, e_4)$, $(x, e_2, f, e_1)$, $(y, e_3, f, e_2)$, and $(z, e_4, f, e_3)$ into
              $\eta_i$;
            delete the $j$th set form $\eta_i$;
    end
end
```

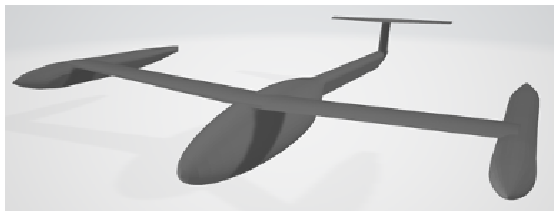
Chapter 4

Results and discussions

To test the robustness of our algorithm, we do several tests on the model OWN-06c Transonic Airliner provided by NASA from OpenVSP,[1] and the eVTOL eCRM-002.[2] For convenience in adding members on the aircraft wing, we remove the turbine engines of the OWN-06c aircraft and all the internal members, landing gears, and the two inner lift nacelles from the eCRM-002. Thus, with no obstacles, the projection algorithm works better with no obstacles.



(a)



(b)

Figure 4.1: Some unnecessary components are removed from the input CAD models of OWN-06c Transonic Airliner (a) and eVTOL eCRM-002 (b).

[1]`http://hangar.openvsp.org/vspfiles/75OWN-06cTransonicAirliner`

[2]`https://www.uber.com/us/en/elevate/uberair/`

4.1 Results with the OWN-06c Transonic Airliner

Skin mesh

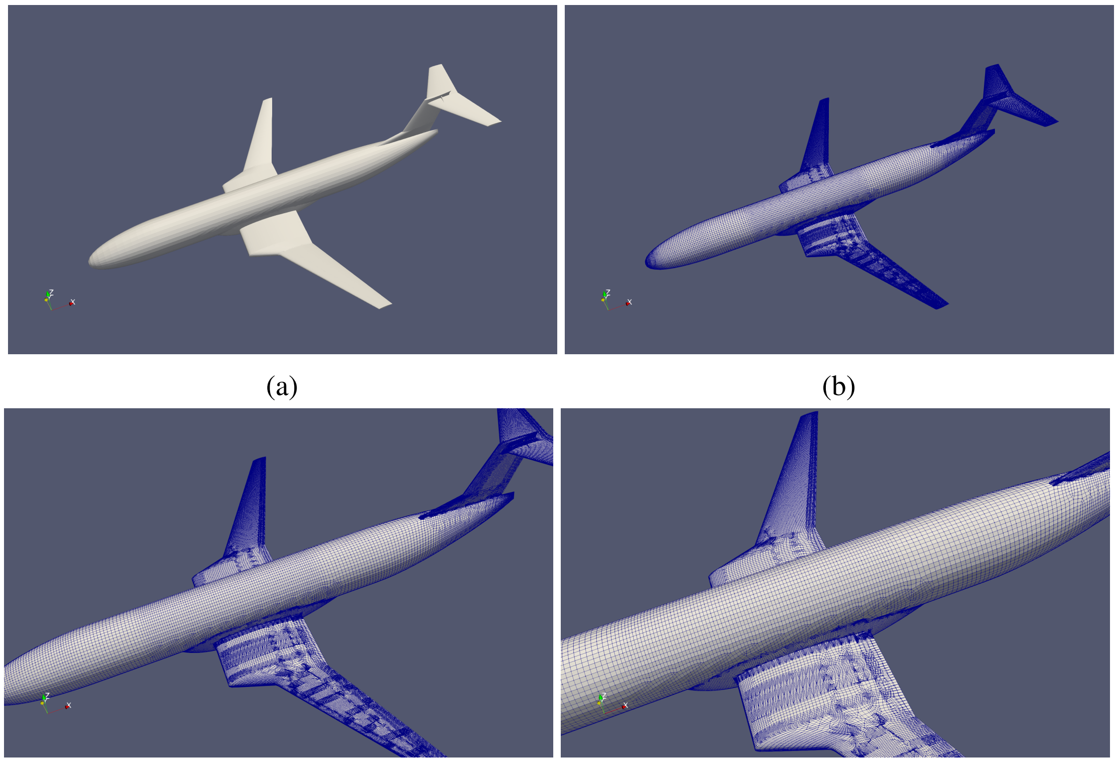


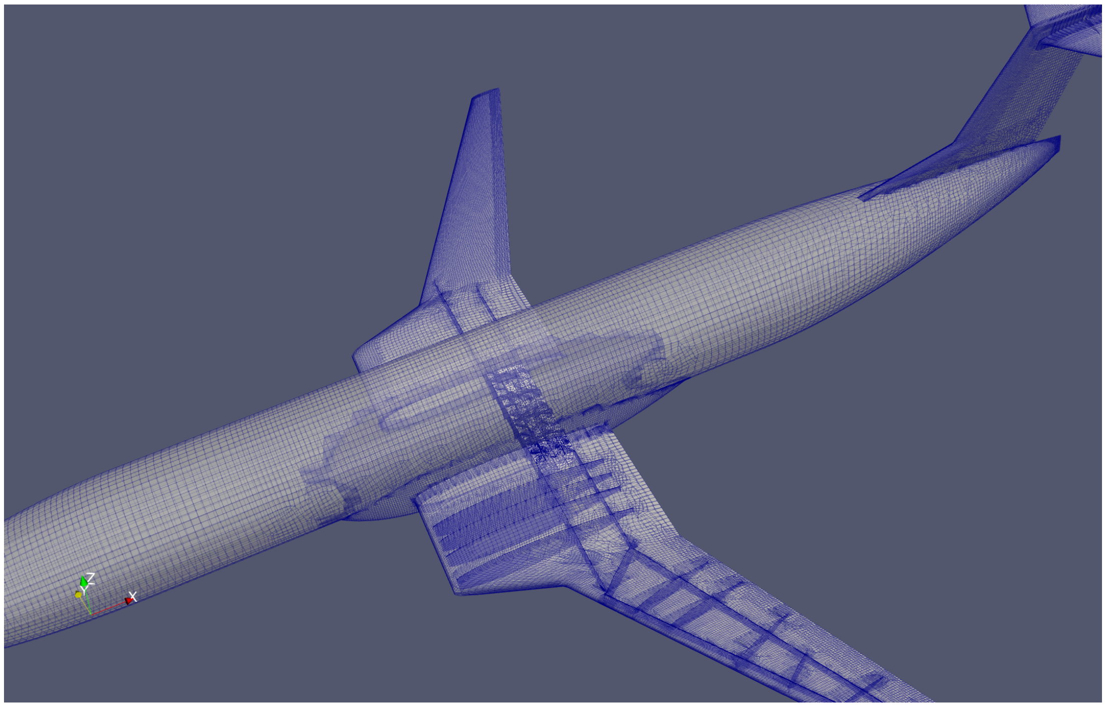
Figure 4.2: Subfigures (a) and (b) are the overview of the skin mesh. Subfigures (c) and (d) zoomed figures of a portion of the skin.

First, we test the projection algorithm, and both ways work well on the OWN-06c aircraft, except that finding the closest point does not behave ideally when we create multiple internal members on the aircraft wing. Some members are very close to the fuselage, if the given projection points are not close enough to the wing surface such that the projected points are located on the fuselage instead of the wing. The reconstruction of the triangulation also behaves nicely on the OWN-06c aircraft. The input for our algorithm is the surface triangular mesh of the aircraft. Although only a small number of splitting, merging, and smoothing optimizations are done on the OWN-06c aircraft skin mesh, the mesh is still transformed into a high-quality unstructured quadrilateral mesh. Also, the intersections of all internal members are successfully detected. As shown in Fig. 4.2, we created 4 spars and 12 ribs on the aircraft wing, and the intersection curves between them and the aircraft skin are clearly shown in the region where the mesh element size is a little smaller.

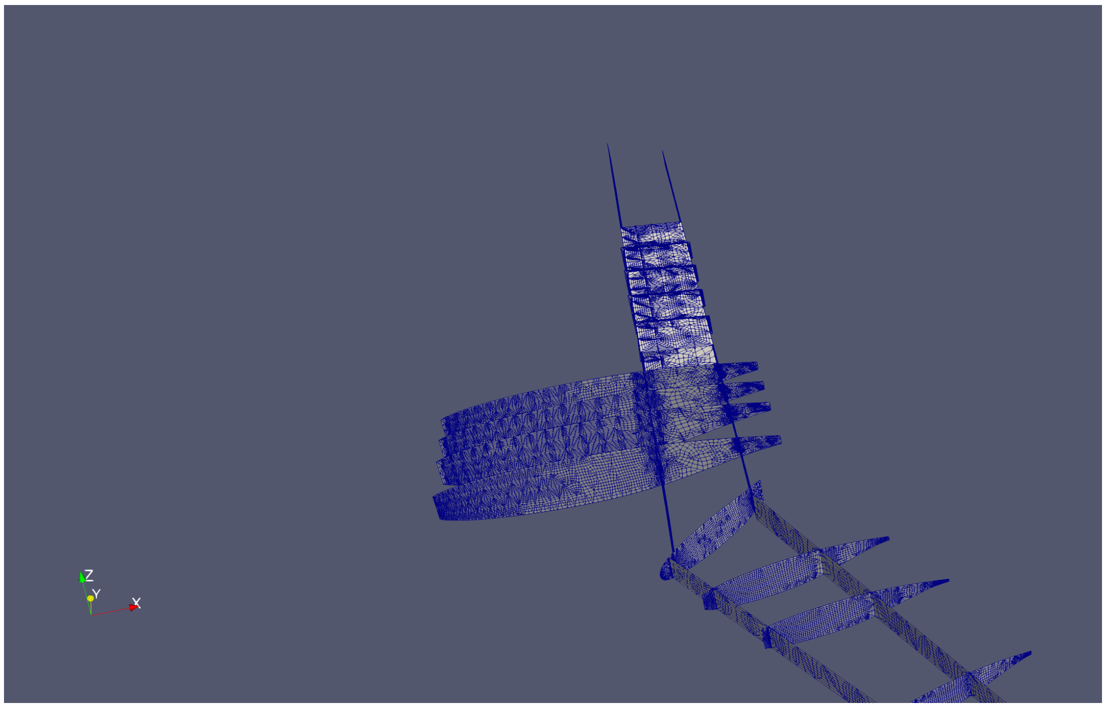
The reason the aircraft skin mesh near the intersection curve between the imprinted members and the skin seems to have a lower quality is that after the reconstruction of triangulation, the region near the intersection tend to have smaller and less regular triangles. The retriangulation algorithm simply connects the projected points to the three vertices of the triangle and creates three new triangles if the projected points are located on the interior of a triangle. Assuming that all the triangles in the original mesh are perfectly equallateral, the created triangles can only be less regular and smaller than the triangle before splitting. In addition, given that some of members are close to each other, if the average size of the initial mesh is not small enough, the retriangulation algorithm will only make the new triangulation worse. In the extreme case, if the initial triangles are so big that all the projected points of two different members are projected onto the same triangle, the mesh quality improvement methods cannot help much when intersection curves are fixed.

Mesh of structural members

The intersection curves between the internal members and the aircraft skin appear to match up perfectly, as shown in Fig.4.3a. As for the mesh of the internal members, we implement a sequence of steps in the mesh quality improvement methods. It is clear from Fig. 4.3b that, for the ribs, the mesh quality improvement methods tend to build structured-like elements in the center, while at the boundary of these imprinted members, even though the elements are less regular, the overall quality is considered good. One possible reason for this phenomenon is that points at the boundary of each imprinted member are created by the projection algorithm, which



(a)



(b)

Figure 4.3: Subfigure (a) shows the intersection is properly detected. Subfigure (b) shows the meshes of the internal members.

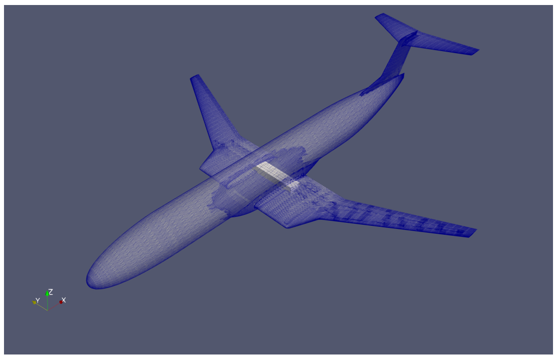
heavily relays on the quality of the initial mesh from the CAD software. Also, the boundary points are fixed throughout the algorithm, which means they cannot be moved to improve the quality of the elements containing them. Conversely, the points at the center are evenly spaced on the parametric domain of a smooth surface created by the transfinite interpolation algorithm and are not fixed throughout the optimization. With high-quality initial mesh and the ability to move, it is easier for the center regions to end up with structured mesh.

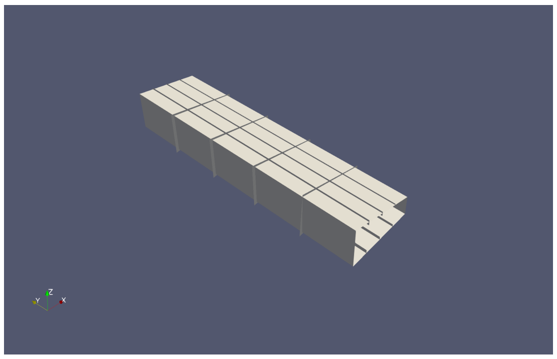
The best thing about this algorithm is that it is fairly efficient. For the mesh generation of the OWN-06c aircraft model, with around 9,000 vertices and 17,000 triangles, it only takes less than 1 hour to finish even though we add more than 20 structural members, which shows great potential for structural analysis in the conceptual design phase.

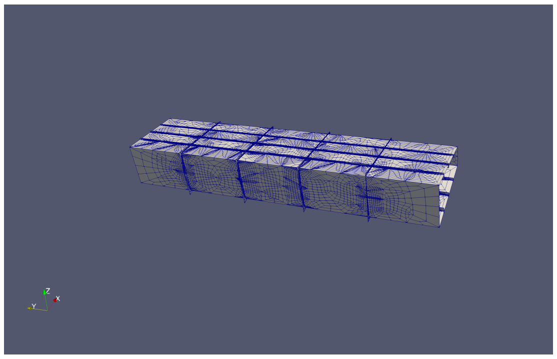
Mesh of the wingbox

To create a wingbox, we create the front and rear spars, ribs, and stringers. One thing that is different about the wingbox is that, unlike the other members, the stringers in the wingbox are only considered to intersect with the surfaces of the wingbox and the ribs should have no intersection with them. As shown in Fig. 4.4b and 4.4c, the members on the upper surface of the wingbox are the stringers, and as can be seen in the resulting mesh, the intersection between the stringers and the wingbox is detected properly.

The successful creation and mesh of the wingbox demonstrates great potential of our algorithm for constructing different kinds of members in different structures, such as the sensors in a robotic arm.


(a)


(b)


(c)

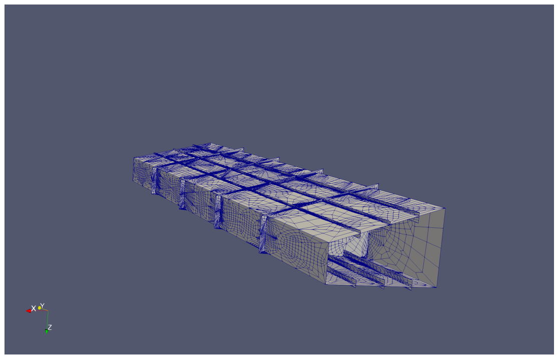
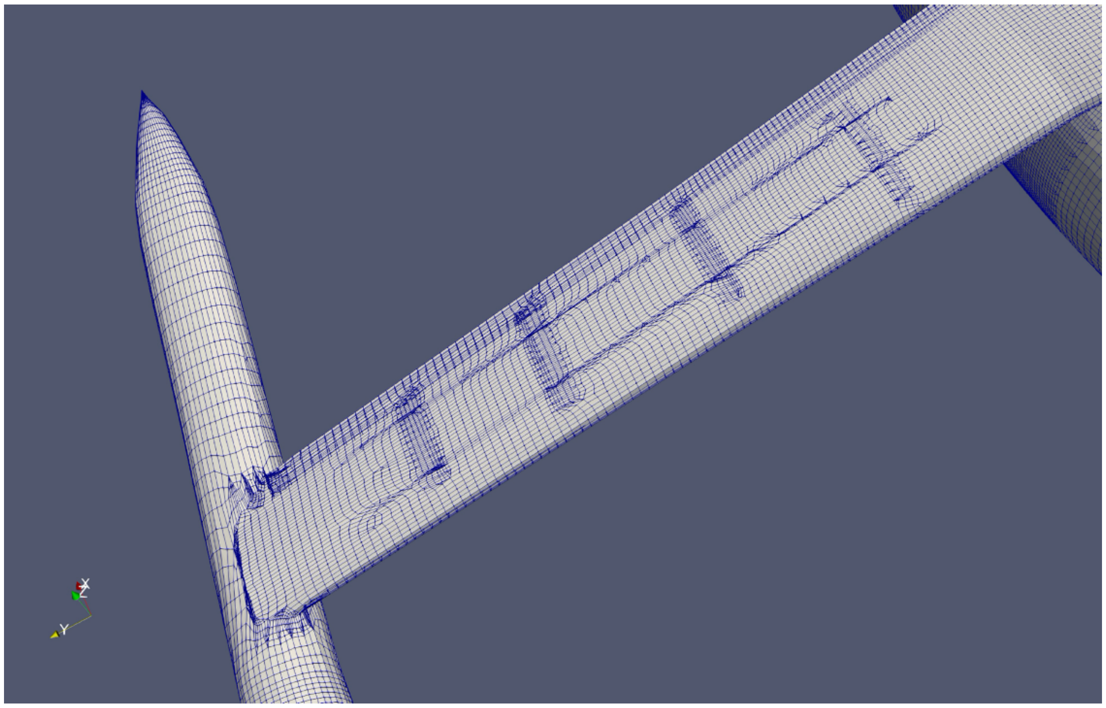
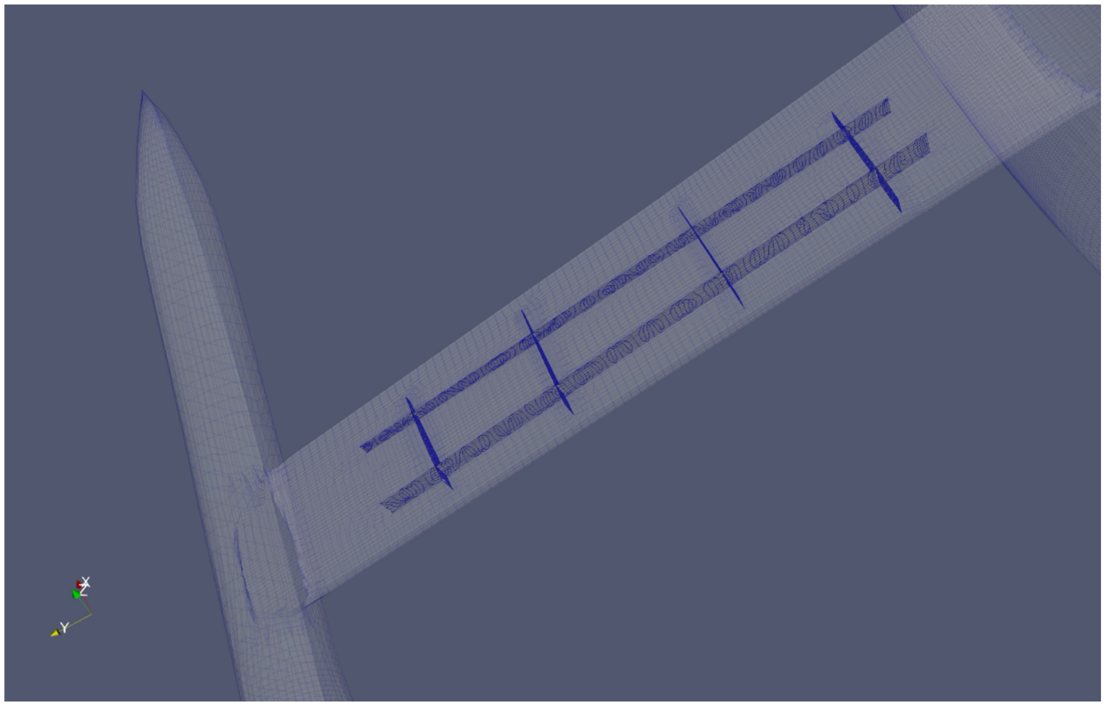

(d)

Figure 4.4: Subfigure (a) shows the location of the wingbox in the aircraft. Subfigure (b) shows the created wingbox component. Subfigures (c) and (d) show views of the mesh of the wingbox from different perspectives.



(a)



(b)

Figure 4.5: We create 2 ribs and 4 spars in the wing of the eVTOL to test the robustness of ouralgorithm

4.2 Results with the eVTOL eCRM-002

The results with the eVTOL have the similar features as the results for the transonic airliner. One thing that requiring some attention is that, with this eVTOL, as we moved in a the span-wise direction, the edges in the chord-wise direction were sometimes more numerous and sometimes less numerous. This kind of pattern is not necessary and is one drawback of our algorithm. We plan to use topology cleanup to fix this problem [12]. Overall, the smoothing method has taken place with local improvements made to the mesh quality.

Our preliminary results are not mature, and we do not yet have a realistic tool for a fully global finite element mesh because we will need additional implementation of features like specifying a circle in space to define the fulselage frame.

Chapter 5

Conclusion

In this thesis, we presented a novel algorithm to generate shell-element meshes for large-scale structural design optimization. Taking an arbitrary triangulation from commercial CAD software, to project the discretized points on the line segments out of the model onto the model skin, we first define two projection algorithms: finding a point on the mesh that gives the smallest distance to the projection point; finding the closest intersection point on the mesh given the projection point and direction. Then, we reconstruct the triangulation by connecting the projected points and finding the intersection between the projected points and the initial triangulation. Based on the projected points, we can fit curves and create a smooth four-sided surface using transfinite interpolation. The surface can then be transformed into a triangular mesh by evenly interpolating points on the parametric domain and applying a Delaunay triangulation algorithm. Finally, the mesh quality of the imprinted members and the skin of the model is improved after a series of splitting, merging, and smoothing optimizations.

The primary contribution is the overall algorithm, which is established to generate smooth quadrilateral meshes for an aircraft, taking any arbitrary mesh as input. The algorithm formulates the key mesh improvement steps as linear and quadratic programming problems, which have the advantages of being solvable in a computationally efficient and robust way. An important part of this algorithm is the trust region approach applied in the mesh smoothing step, where this approach turns the inequality constraints into equality constraints and turns the nonlinear programming problem into a quadratic programming problem. This eliminates the possibility of non-convergence. The algorithm also works for geometries other than aircraft. For instance, we can apply it to the shell-element mesh generation of other engineering systems where structures and materials are important such as vehicles, robots, and medical devices.

In summary, the overall approach is potentially a useful way to generate shell-element meshes and add imprinted members to the structures with complicated geometry. Starting from our current algorithm, our future work will be adapting the retriangulation algorithm and in the current framework searching for any possibility to make it more general, user-friendly and robust. We will also test the algorithm on other structures from the other fields other than the aerospace field and structures with more complicated geometry. In addition, we want to extend the ideas in this algorithm to the generation of high-quality solid element meshes.

The thesis, in full, is an adaption of the material as it appears in Li, Ning, and John T. Hwang.“Automatic generation of global shell-element meshes for large-scale structural design optimization” AIAA AVIATION 2020 FORUM. 2020. This thesis is currently being prepared for submission for publication of the material. Li, Ning; Hwang, John T. The thesis author was the primary investigator and author of this material.

Bibliography

[1] Daniel Raymer. *Aircraft design: a conceptual approach*. American Institute of Aeronautics and Astronautics, Inc., 2012.

[2] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Instant field-aligned meshes. *ACM Trans. Graph.*, 34(6):189–1, 2015.

[3] John T Hwang and Joaquim RRA Martins. An unstructured quadrilateral mesh generation algorithm for aircraft structures. *Aerospace Science and Technology*, 59:172–182, 2016.

[4] John Hwang and Joaquim Martins. Geomach: geometry-centric mdao of aircraft configurations with high fidelity. In *12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 5605, 2012.

[5] Andrew Hahn. Vehicle sketch pad: a parametric geometry modeler for conceptual aircraft design. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, page 657, 2010.

[6] Lars-Erik Eriksson. Practical three-dimensional mesh generation using transfinite interpolation. *SIAM journal on scientific and statistical computing*, 6(3):712–741, 1985.

[7] Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. N-symmetry direction field design. *ACM Transactions on Graphics (TOG)*, 27(2):10, 2008.

[8] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Transactions on Graphics (ToG)*, 32(4):59, 2013.

[9] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Stripe patterns on surfaces. *ACM Transactions on Graphics (TOG)*, 34(4):39, 2015.

[10] Jingwei Huang, Yichao Zhou, Matthias Niessner, Jonathan Richard Shewchuk, and Leonidas J Guibas. Quadriflow: A scalable and robust method for quadrangulation. In *Computer Graphics Forum*, volume 37, pages 147–160. Wiley Online Library, 2018.

[11] VN Parthasarathy and Srinivas Kodiyalam. A constrained optimization approach to finite element mesh smoothing. *Finite Elements in Analysis and Design*, 9(4):309–320, 1991.

[12] Scott A Canann, SN Muthukrishnan, and RK Phillips. Topological improvement procedures for quadrilateral finite element meshes. *Engineering with Computers*, 14(2):168–177, 1998.

[13] Felix Kälberer, Matthias Nieser, and Konrad Polthier. Quadcover-surface parameterization using branched coverings. In *Computer graphics forum*, volume 26, pages 375–384. Wiley Online Library, 2007.

[14] Pierre-Alexandre Beaufort, Christophe Geuzaine, and Jean-François Remacle. Automatic surface mesh generation for discrete models: A complete and automatic pipeline based on reparameterization. *arXiv preprint arXiv:2001.02542*, 2020.

[15] Jianwei Guo, Fan Ding, Xiaohong Jia, and Dong-Ming Yan. Automatic and high-quality surface mesh generation for cad models. *Computer-Aided Design*, 109:49–59, 2019.

[16] Chaman Singh Verma and Tim Tautges. Jaal: Engineering a high quality all-quadrilateral mesh generator. In *Proceedings of the 20th International Meshing Roundtable*, pages 511–530. Springer, 2011.

[17] YK Lee, Chin K Lim, Hamid Ghazialam, Harsh Vardhan, and Erling Eklund. Surface mesh generation for dirty geometries by the cartesian shrink-wrapping technique. *Engineering with Computers*, 26(4):377–390, 2010.

[18] Benjamin Villard, Vicente Grau, and Ernesto Zacur. Surface mesh reconstruction from cardiac mri contours. *Journal of Imaging*, 4(1):16, 2018.

[19] Dawei Zhao, Jianjun Chen, Yao Zheng, Zhengge Huang, and Jianjing Zheng. Fine-grained parallel algorithm for unstructured surface mesh generation. *Computers & Structures*, 154:177–191, 2015.

[20] Antonio CO Miranda and Luiz F Martha. Mesh generation on high-curvature surfaces based on a background quadtree structure. *space*, 500:N2, 2002.

[21] Changhyup Park, Jae-Seung Noh, Il-Sik Jang, and Joe M Kang. A new automated scheme of quadrilateral mesh generation for randomly distributed line constraints. *Computer-Aided Design*, 39(4):258–267, 2007.

[22] Carlos A Recarey Morfa, Irvin Pablo Pérez Morales, Márcio Muniz de Farias, Eugenio Oñate Ibañez de Navarra, Roberto Roselló Valera, and Harold Díaz-Guzmán Casañas. General advancing front packing algorithm for the discrete element method. *Computational Particle Mechanics*, 5(1):13–33, 2018.

[23] Huibo Dang, Hongjie Zhang, and Huixue Dang. Automatic generation of cfd mesh for mountainous regions. 2017.

[24] Alexander B Costenoble, Bharath Govindarajan, Yong Su Jung, and James D Baeder. Automated mesh generation and solution analysis of arbitrary airfoil geometries. In *23rd AIAA Computational Fluid Dynamics Conference*, page 3452, 2017.

[25] Jasmeet Singh and Carl F Ollivier Gooch. Advancing layer surface mesh generation. In *AIAA Scitech 2020 Forum*, page 0902, 2020.

[26] Mael Rouxel-Labbé, Mathijs Wintraecken, and J-D Boissonnat. Discretized riemannian delaunay triangulations. *Procedia engineering*, 163:97–109, 2016.

[27] Darren Engwirda and David Ivers. Off-centre steiner points for delaunay-refinement on curved surfaces. *Computer-Aided Design*, 72:157–171, 2016.

[28] Yuto Otoguro, Kenji Takizawa, and Tayfun E Tezduyar. A general-purpose nurbs mesh generation method for complex geometries. In *Frontiers in Computational Fluid-Structure Interaction and Flow Simulation*, pages 399–434. Springer, 2018.

[29] Yan Liu, Albert A Saputra, Junchao Wang, Francis Tin-Loi, and Chongmin Song. Automatic polyhedral mesh generation and scaled boundary finite element analysis of stl models. *Computer Methods in Applied Mechanics and Engineering*, 313:106–132, 2017.

[30] Christos Tsolakis, Fotis Drakopoulos, and Nikos P Chrisochoides. Sequential metric-based adaptive mesh generation. 2018.

[31] Wei Chen, Xiaopeng Zheng, Jingyao Ke, Na Lei, Zhongxuan Luo, and Xianfeng Gu. Quadrilateral mesh generation i: Metric based method. *Computer Methods in Applied Mechanics and Engineering*, 356:652–668, 2019.

[32] Ashutosh Soni and Partha Bhowmick. Quadrangular mesh generation using centroidal voronoi tessellation on voxelized surface. In *International Workshop on Combinatorial Image Analysis*, pages 97–111. Springer, 2018.

[33] Christian Altenhofen, Felix Schuwirth, André Stork, and Dieter W Fellner. Implicit mesh generation using volumetric subdivision. In *VRIPHYS*, pages 9–19, 2017.

[34] David Eberly. Distance between point and triangle in 3d. *Magic Software, http://www. magic-software. com/Documentation/pt3tri3. pdf*, 1999.