ABSTRACT

Title of thesis: HAMSTRAN, AN INDIRECT METHOD TO CREATE ALL- QUADRILATERAL GRIDS FOR THE HAMSTR FLOW SOLVER

Zhu Zihao, Master of Science, 2017.

Thesis directed by Professor, James Baeder,
Department of Aerospace Engineering

HAMSTR is a newly developed flow solver that utilizes Hamiltonian paths and strand grids for three-dimensional flows on overset and hybrid meshes. In order to use HAMSTR, it is required that one has a decent all-quad surface mesh; this is the motivation for the development of HAMSTRAN. It is an indirect method to create an unstructured all-quadrilateral 3D surface mesh and strand templates for each vertex. It transforms triangular as well as quad-dominant surface meshes into all-quad meshes, without any smoothing steps. The proposed method is fast and can work on highly complicated surfaces with lots of sharp features while producing a minimum number of irregular grids. HAMSTRAN is mostly based on the Q-Tran algorithm, but it has many advantages over Q-Tran. For example, HAMSTRAN is not only able to utilize an all-triangular

mesh, but can also use quad-dominant and hybrid meshes as input and generate a decent all-quad mesh. After creating the surface mesh, HAMSTRAN proceeds to create the strand templates for each vertex. The strand templates are vectors extruding from the surface in the wall-normal direction without crossing each other. The direction of the vectors can be adjusted according to the 3D volume mesh required for the future flow solving process. Several examples will be presented to demonstrate the efficiency of this method, such as for airfoils, wings, rotor blades and fuselage.

HAMSTRAN: AN INDIRECT METHOD TO CREATE ALL-QUADRILATERAL GRIDS FOR THE HAMSTR FLOW SOLVER

By

Zhu Zihao

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2017

Advisory Committee:
Professor James Baeder, Chair
Assistant Professor Stuart Laurence
Assistant Professor Bharath Govindarajan

Dedication

To my family and friends who have

always loved and supported me

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. James Baeder, for providing me the opportunity to work on this project and providing me with help whenever I need it.

Thank you to the United States Army for financially supporting this project.

Thank you to Bharath Govadajaran, Alexander Costenoble and Yong Su Jung for providing me with some of the intial background mesh.

Thank you to the developers of the openvsp software for providing me with some of the initial background mesh.

I would also like to thank all my lab mates for providing me with a comfortable working environment and lots of fun during this year.

List of tables

List of figures

Nomenclature

ρ ……density (kg/m^3)

u…….velocity (m/s)

p…….pressure (Pa)

τ……..stress tensor

E…….kinetic energy (J)

T……temperature (K)

k…….heat conductivity (W/(m.K))

a,b,c…….three edges of a triangle

θ……..intersection angle between two edges (°)

CFD……...….computational fluid dynamics

mesh……..the collection of all grid cells in a domain

structured mesh……..grid cells have a particular order

unstructured mesh……..grid cells have no particular order

flow solver…….a computational software to simulate flows in real life

valence……the number of edges sharing a vertex

BDF1……..first order backwards differential formula

BDF2……..second order backwards differential formula

MUSCL……..a third order discretization scheme in space

airfoil……...cross section of a wing

chord……..an imaginary straight line joining the leading and trailing edges of an airfoil

fuselage……..….central body portion of an aircraft

skewness…….an indicator of the mesh quality, the smaller the better

vortex……. a region in a fluid in which the flow rotates around an axis

Mach number……..the ratio of flow velocity to the local speed of sound

inviscid flow……..an ideal flow that has no viscosity

laminar flow……...a type of flow that travels in smooth paths or layers

turbulent flow…….a type of flow where the fluid undergoes irregular fluctuations or mixing, in contrast to laminar flow

hexagon…….a simple 2D geometry that has 6 sides

quadrilateral……..a geometry that has four sides

shock……a propagating disturbance that only occurs when the speed of the flow is higher than the local speed of sound

transonic…….. the condition of flight in which a range of velocities of airflow exist flowing past an air vehicle that are below, at, and above the speed of sound

boundary layer……..the layer of fluid in the immediate vicinity of a bounding surface where the effects of viscosity are significant

Reynolds number……..the ratio of inertial forces to viscous forces within a fluid, used to determine whether the flow is laminar or turbulent

Table of contents

Chapter 1: Motivation

1.1. Introduction to computational fluid dynamics

With the advancement of aviation technology, new and better components are developed every day, ranging from airfoils, fuselage and rotor blades. Traditionally, whenever a new component is designed, its performance must be tested by experiments. However, some experiments can be very expensive or dangerous to carry out in the real world. As a result, combined with the development in computational technology, $computational\ fluid\ dynamics\ (CFD)$[1] is becoming an increasingly important design tool in engineering. This tool not only decreases the cost of development, but also allows the designer to simulate conditions that would be dangerous in the physical world.

Computational fluid dynamics is concerned with the motion of the fluid as well as the interaction of the fluid with solid bodies. This method is usually based on the *Navier-Stokes* $Equation$[2]. These equations consist of the conservation of mass, momentum and energy and they are able to describe how the velocity, pressure, temperature, and density of a moving fluid are related.

The conservation of mass:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_i)}{\partial x_i} = 0 \qquad (1)$$

The conservation of momentum:

$$\frac{\partial(\rho u_i)}{\partial t} + \frac{\partial(u_j \rho u_i)}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ij}}{\partial x_j} \tag{2}$$

The conservation of energy:

$$\frac{\partial(\rho E)}{\partial t} + \frac{\partial(u_j \rho E)}{\partial x_j} - \frac{\partial}{\partial x_i}\left(k \frac{\partial T}{\partial x_i}\right) + \frac{\partial(u_j p)}{\partial x_j} - \frac{\partial(\tau_{ij} u_j)}{\partial x_j} = 0 \tag{3}$$

The greatest challenge of computational fluid dynamics is the fidelity. Since the CFD method is created to simulate flows in the real world, its correspondence with experimental data is very important. So a crucial part of developing a CFD method is to verify its accuracy with canonical experimental data. Only when the simulation results show good agreement with the experimental data can this simulation be considered successful. Another challenge is to reduce the time required for a simulation. Since one of the purposes of CFD is to decrease the cost of development, the time required for the simulation should also be as short as possible. The purpose of this research is to find ways to increase the accuracy of computational fluid dynamics and reduce the time required to run a simulation.

The first step of CFD is the discretization of physical space, also known as the grid generation. In order to improve the fidelity of a computational simulation, grid generation schemes are becoming more and more complicated. The development started first with relatively simple *structured* $mesh$[3] systems constructed either by algebraic methods or by using partial differential equations. But with increasing geometrical complexity of the configurations, the grids had to be broken into

a number of topologically simpler blocks (multiblock approach). This approach allows a structured mesh system to be built around a complicated geometry. However, the generation of a structured mesh around a complicated geometry may take weeks to complete. Therefore, this led to the development of *unstructured mesh system*[3], especially for complex geometries.

An *unstructured grid* system[3] is mainly comprised of triangular grids for surface mesh and tetrahedral grids for volume mesh. Since the triangular girds and tetrahedral grids can be generated automatically regardless of the complexity of the domain, this scheme is able to discretize a complex domain with minimum effort. However, traditional unstructured flow solvers have limited convergence and spatial accuracy of the solution primarily because of the difficulty in identifying "line-structures" in a purely unstructured grid system. As a result, unstructured solvers generally have a much higher computational cost and lower accuracy comparing to structured solver. Despite this shortcoming, they are still widely used for its fast and simple mesh generation schemes.

As can be seen from the discussion above, the unstructured mesh generation scheme is faster than the structured mesh generation scheme while the structured flow solver is faster than unstructured flow solver. So in order to decrease the computational cost to its minimum, the current work decided to utilize a scheme that could employ unstructured mesh in a structure-based flow solver, known as the HAMSTR[4] flow solver.

1.2. The HAMSTR flow solver

One way to exploit structure in an unstructured mesh is to utilize *Hamiltonian paths*[4]. This is achieved by extracting 'linelets' in an unstructured grid system. Then stencil-based discretization schemes can be carried out along these 'linelets', similar to that of a structured flow solver. However, on a surface mesh, it is very hard to extract these lines in triangular grids while most unstructured mesh generators create triangular grids. As a result, there needs to be a way to transform these triangular grids into quadrilateral grids. Traditionally, this is done by *Catmull-Clark subdivision*[5]. During this process, the midpoint of each edge in a polygon is connected to the centroid of said polygon, diving each triangle into three quadrilaterals and each quadrilateral into 4 smaller quadrilaterals, as is shown in Fig 1. Then 'linelets' can be created by linking the midpoint of opposite edges in a grid and the centroids of adjacent grids, creating structure in an unstructured surface mesh.

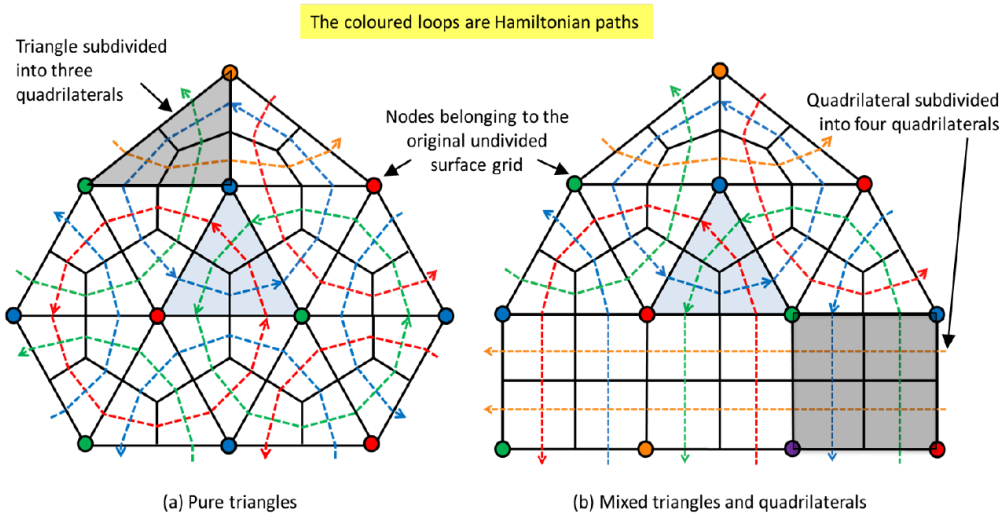


Fig1. Simple sketch of the *HAMILTONAIN loops*[4]

In order to extend this formulation to 3D, strand grids have been employed. The role of strand grids is to provide structure in the wall normal direction, thus allowing the line-implicit methods to be carried out along these strands. In order to create *strand girds*[4] from a surface mesh, one must first create strand templates. A strand template is a wall-normal direction vector originated from a vertex on the surface and a one-dimensional grid point distribution along this vector. As a result, layers of hexahedra are generated by extruding the surface grids into the 3D volume, as can be seen in Figure 2. The strand grids not only creates structure in the wall normal direction, but also preserves the connectivity of Hamiltonian paths in the surface direction.

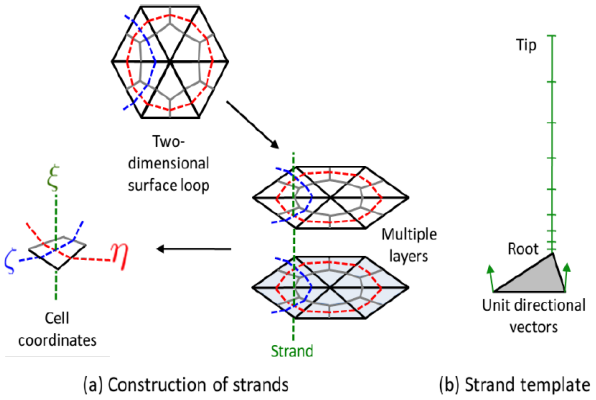


Fig 2. A sketch of the *strand template*[4]

However, the Catmull-Clark subdivision algorithm has several problems. The biggest one is that this method creates a large number of irregular vertices, vertices with a valence not equal to 4. This is because the centroid of each triangle is connected to 3 midpoints on the edge, giving it a valence of 3. This nature of the Catmull-Clark subdivision will led to the creation of a huge number of irregular vertices. Apart from this, the method also dramatically increases the number of cells, leading to a higher computational cost. Although we can make the original background mesh coarser through a loss of resolution, this might change the feature of the mesh system and lead to

a loss of accuracy in the future flow solving process. So unless the original background mesh is comprised of very regular grids and the increased computational time is within an acceptable range, usually a more complicated method is preferred. Many methods have been developed for this purpose, this paper would like to introduce four of them in the next chapter.

1.3. Previous work

In order to use the HAMSTR flow solver, an all-quad surface mesh is required. The mesh should be able to represent the surface features of a geometry while having a minimum number of irregular vertices and most grid cells should resemble a rectangle. The schemes to generate an all-quad mesh system can be grouped into 2 categories, the indirect approach and the direct approach. The direct approach generates quad grids directly while an indirect approach transforms a background mesh into an all-quad mesh. For an indirect approach, the number of grid cells after the transformation should stay approximately the same as the input tessellation. The transformation process should be fast and automatic regardless of the complexity of the original input tessellation.

1.3.1. Hypermesh commercial mesh generator

$Hypermesh$[6] is a mesh generator that can generate unstructured all-quad mesh for complicated geometries, something a lot of commercial software cannot do. The biggest advantage of this software is its ability to generate quad grids directly with no background mesh system needed. However, there are several disadvantages regarding this method. First, the size of the grid cells generated by this software has to be approximately the same, making it unsuitable for a lot geometries. For example, in a wing mesh, the grid size at the leading edge should be very small in order to capture the high pressure gradient caused by its sharp curvature. However, larger grid

cells are required on the main body where the pressure gradient is low due to the smooth geometry, or else the number of grids will be overly high, increasing the computational cost. Second, in order to generate a mesh of good quality, one has to divide the surface into different domains. This subdivision process can be very time consuming for a complicated geometry.

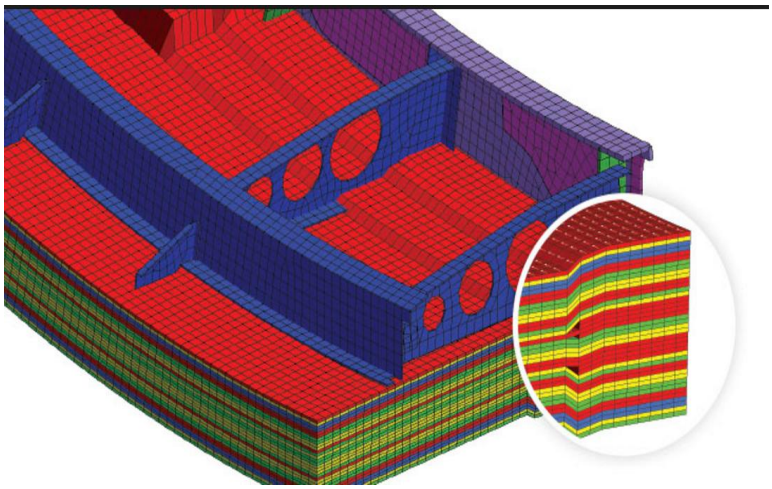
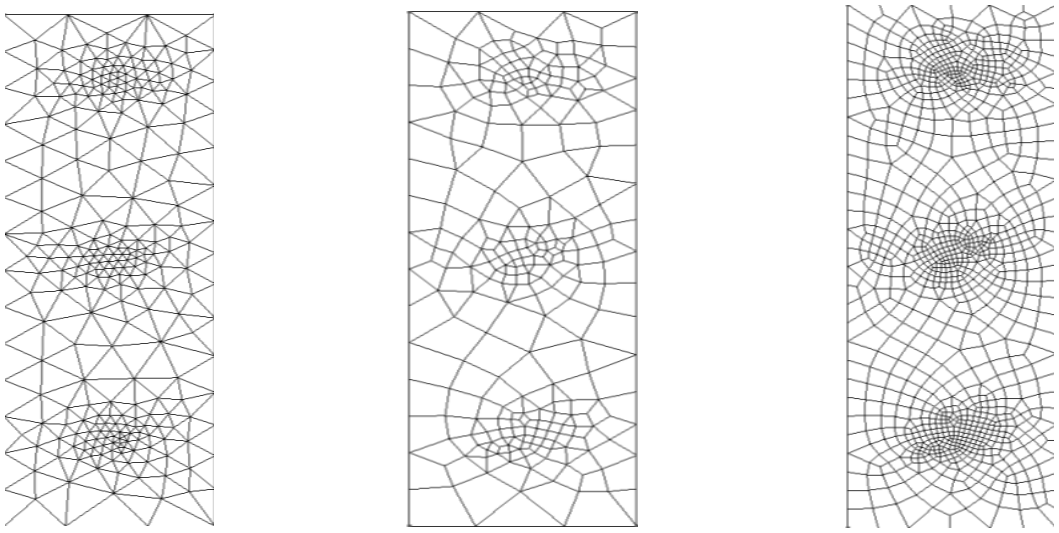


Fig 3. Example of mesh generated by the *Hypermesh* $software$[6]

1.3.2. The Blossom method

An indirect all-quad mesh generation method is the *Blossom* $algorithm$[7], which creates quad grids by fusing two adjacent triangular cells into one quad cell. However, this method requires a smoothing process and a subdivision process, without which there will be many leftover triangles and poor quality quad grids. The smoothing process will change the coordinates of the original vertices on the background mesh. In a 2D surface, this is not a big problem, however, in a complicated 3D surface, this might lead to the loss of surface feature. The sequence of the merging process will also heavily affect the quality of the quad mesh, so a lot of work is required to determine the optimum sequence. The more complicated the mesh is, the harder it is to calculate

the optimum sequence. Since one may need an algorithm that is fast and automatic regardless of the complexity of the original input mesh, this algorithm is far from satisfactory.



The original triangular mesh | The merging process | The subdivision process

Fig 4. An illustrative picture of the *blossom method*[7]

1.3.3. The Q-Morph method

$Q-Morph$[8] utilizes an advancing front algorithm to determine the sequence of triangle transformations. The process starts at the boundary and after each row of triangles are transformed into quadrilaterals, a smoothing process is performed before advancing the boundary. Although this method can make quad grids of decent quality, it has 2 major disadvantages. First of all, it requires a smoothing process that can change the coordinates of the vertices on the background mesh, which might destroy the surface feature. Second, the advancing front algorithm is very hard to implement in complicated meshes without clear boundaries. Third, the amount of work required

increases dramatically with the number of grids and the complexity of the geometry, making it unsuitable for complicated geometries, similar to the Blossom algorithm.

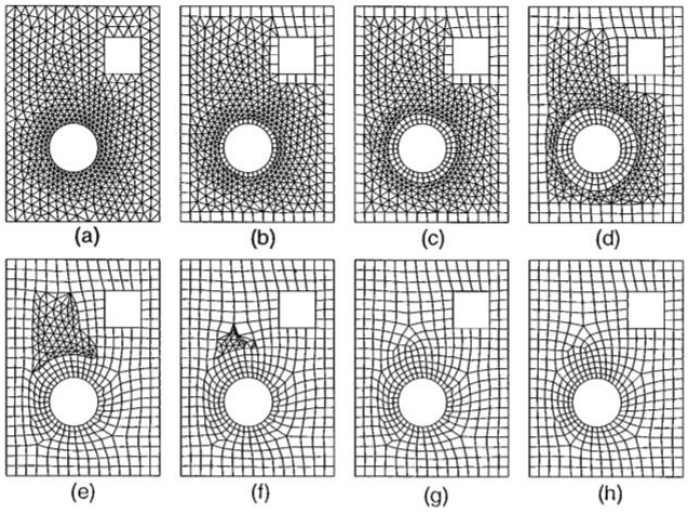
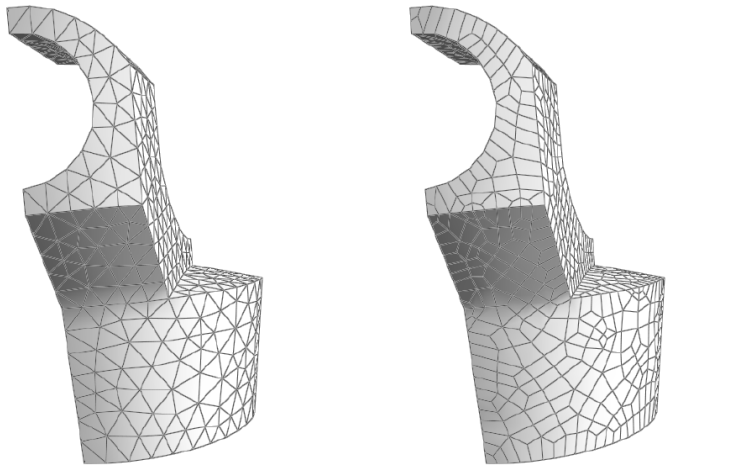


Fig 5. A plot to illustrate the advancing front process in *Q-Morph*[8]

1.3.4. The Q-Tran method

$Q-Tran$[9] is an indirect method to create an all-quad mesh system from a triangular input mesh. This method preserves the coordinates of original vertices on the background mesh, thus preserving the surface feature of the geometry. It can be carried out automatically regardless of the complexity of the domain. These two features make Q-Tran superior to Q-Morph and Blossom. Since the feature of the original mesh is well kept during the transformation process, as long as the input triangular mesh is able to cluster where it needs to be, so will the output mesh. Since most unstructured triangular mesh generators have the ability to cluster in places where the curvature is high, Q-Tran is also superior to Hypermesh in this regard. Additionally, the number of elements stays approximately the same after implementing the algorithm. However, it also has some

limitations, it only deals with all triangular background mesh but doesn't deal with unstructured quad-dominant mesh, or hybrid mesh comprised of unstructured triangular grids and structured quadrilateral grids. Some steps in the Q-Tran algorithm should also be modified to make the method easier for users to carry out.



(a) Input: 1,014 triangles (b) Output: 1,212 quadrilaterals

Fig 6. Example of the Q-Tran algorithm[9]

1.4. The purpose of this research

Although the Q-Tran algorithm has many limitations, it is still more suitable for generating an all-quad surface mesh comparing to all the other methods explained in this work. So the purpose of the current research is to create a new and better algorithm based on Q-Tran, but has more explicit steps to follow and can be implemented on a wider range of background meshes, called the HAMSTRAN. This method not only has all the advantages of the Q-Tran algorithm, but also has some unique advantages which will be explained in the next chapter. After implementing this mesh generation scheme in the HAMSTR flow solver, it should be tested to validate the improvement in the accuracy of results and savings in computational time. Thus this algorithm is implemented on several geometries, such as the *NACA0012* $airfoil$[10] and the triangular wedge in 2D, the *OneraM6* $wing$[10], the *Caradonna Tung lifting* $rotor$[11] and the *robin* $fuselage$[12] in 3D. The results are compared with that of the traditional subdivided mesh as well as the experimental results when available.

Chapter 2: Methodology

2.1. An introduction to HAMSTRAN

The HAMSTRAN algorithm has many unique advantages. For example, the output tessellation has a minimum number of irregular vertices. In this way, the HAMSTR flow solver will be able to extract long, smooth Hamiltonian paths from this mesh instead of short, curved ones from the subdivided mesh, leading to improved accuracy of computational results. The number of grids in the output mesh is approximately the same as that of the original mesh, which could save the computational cost considerably comparing to the subdivided mesh. Unlike Blossom and Q-Morph, this method also preserves all the coordinates on the background mesh, thus preserving the surface feature of the geometry.

HAMSTRAN can be carried out in 5 steps, the initial clean-up process, the edge classification, the quad generation, the final merging process and the generation of strand templates for each vertex. Except for the first and last step, all the major steps share a lot of similarities with Q-Tran although there are many differences as well. Comparing to Q-Tran, this algorithm is able to deal with a wider variety of background mesh. Everything will be explained in detail in the current chapter.

HAMSTRAN needs to be carried out according to the input tessellation. The input tessellation can be classified as all-triangular mesh, unstructured quad-dominant mesh and hybrid mesh comprised of unstructured triangles and structured quadrilaterals. First, this paper will introduce its implementation on an all-triangular background mesh.

2.1.1. Implementation on all triangular background mesh

A. The initial clean-up process: Destroy the highly irregular gird cells

Sometimes the original mesh may have some highly irregular grid cells that would affect the quality of the mesh, so a clean-up process is required to improve the quality of the background mesh, something that is absent from Q-Tran. A grid cell is classified into a highly irregular grid when its aspect ratio exceeds 4.0, the aspect ratio can be calculated by Equation 4.

$$S = (a + b + c)/2 \qquad (4)$$

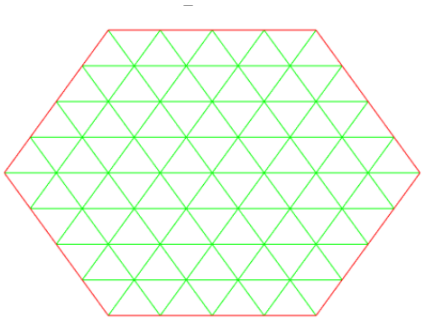
$$\text{Aspect Ratio} = \frac{abc}{8(S-a)(S-b)(S-c)}$$

The details of this process is as follows.

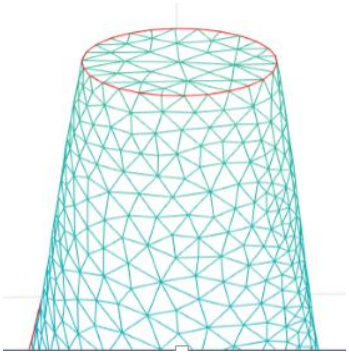
1. Build the normal vectors of all the original triangles, make sure that all the normal vectors point outwards away from the geometry.
2. A type 1 boundary edge is an edge adjacent to a single triangular face, as shown in Fig 7a.
3. A type 2 boundary edge is an edge adjacent to 2 triangular faces whose normal vectors has an intersection angle bigger than a certain value, usually 30 to 60 degrees is recommended. It can also be an edge associated with important features of the geometry, such as the camber line of an airfoil, as can be seen in Figure 7b,c.
4. In areas associated with high curvature, any triangular face with all 3 vertices belonging to a boundary edge is also classified as a type 3 boundary edge, as can be seen in Figure 7d.
5. Select the highly irregular grids. For each of them, select the vertex with the biggest angle,

called the primary vertex. The primary vertex must not belong to a boundary edge. Among the other two vertices in the triangle, the one closest to the primary vertex is classified as a secondary vertex.

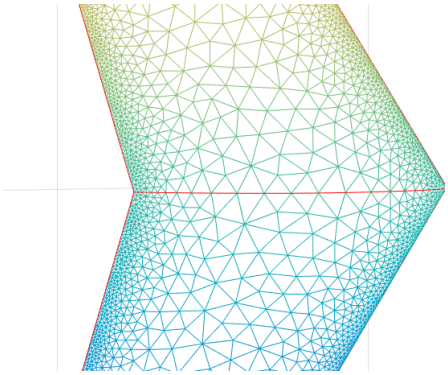
6. Any edge that is connected to the primary vertex is reconnected to the secondary vertex, thus destroying the irregular grids, as shown in Fig 8.



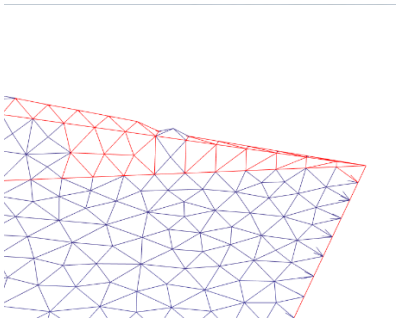
a. Type 1 boundary edge



b. Type 2 boundary edge



c. Type 2 boundary edge



d. Type 3 boundary edge

Fig 7. Examples of boundary edge

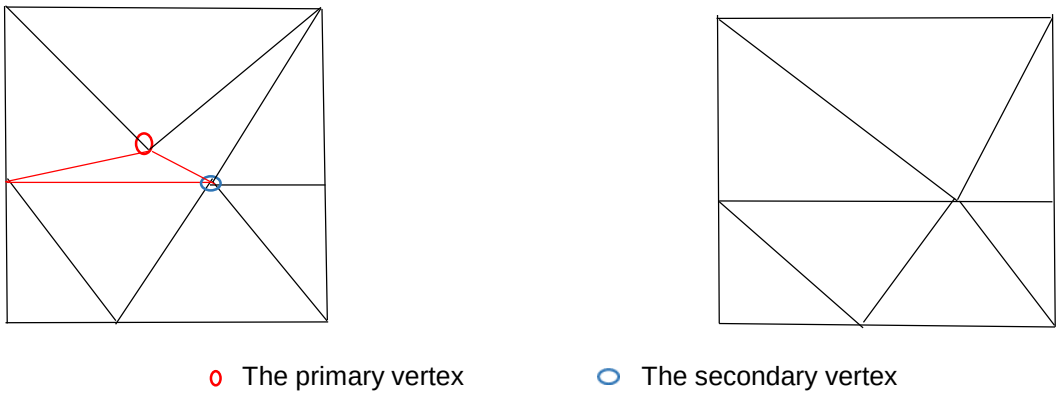


Fig 8. An illustrative picture to show the clean-up process (The red triangle is the highly irregular grid)

Under this method, the quality of the background mesh will improve. However, if there are too many highly irregular grids in the background mesh, then it is best to choose another background mesh with better quality. This process provides us with a better input tessellation, something that is absent from Q-Tran. Although this process takes some primary vertices off the mesh, it doesn't change the coordinates of any remaining vertices. Not to mention highly irregular triangular grids are rare, thus the surface feature would usually be preserved after this process.

B. The initial edge classification

In order to transform triangular grids into quadrilateral grids, each triangle must be treated differently according to the types of edges it possesses. So the first step is to classify the sides of each triangle into different types of edges. The details are shown in both the following paragraphs and Figure 9.

1. The boundary edge, BE, all the type 1, type 2 and type 3 boundary edges in the previous section are all classified into boundary edges.

2. An anisotropic diagonal edge, ADE, is a non-boundary edge adjacent to two triangular. faces with its length greater than the length of the remaining four edges and at least one of the two triangles is anisotropic. A triangular face is considered anisotropic if its aspect ratio exceeds 3.0. The aspect ratio can be calculated according to Equation 4.

3. An isotropic diagonal edge, IDE, is a non-boundary edge adjacent to two isotropic triangular faces such that its length is greater than the length of the remaining four edges and none of these edges is a boundary edge.

4. A regular edge, RE, is an edge that doesn’t belong to any of the above categories.

5. A special regular edge, SRE, is a regular edge with two adjacent triangles that contains two aligned boundary edges (Two edges are considered to be aligned if their intersection angle is bigger than a certain value, for example 150 degrees).

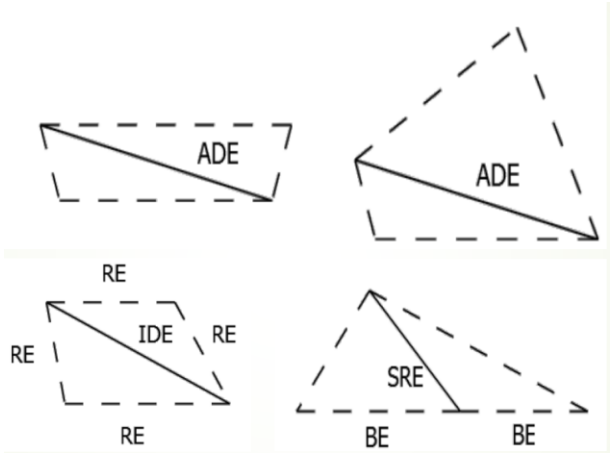


Fig 9. *An illustrative plot of the edge classification* process[9]

In this way, all the triangular edges are grouped into different types. However, if the quad grids are created according to these edges, a lot of irregular grids will occur. Here a grid is considered to be an irregular grid if it deviates too far from a rectangle, as shown in Fig 10.

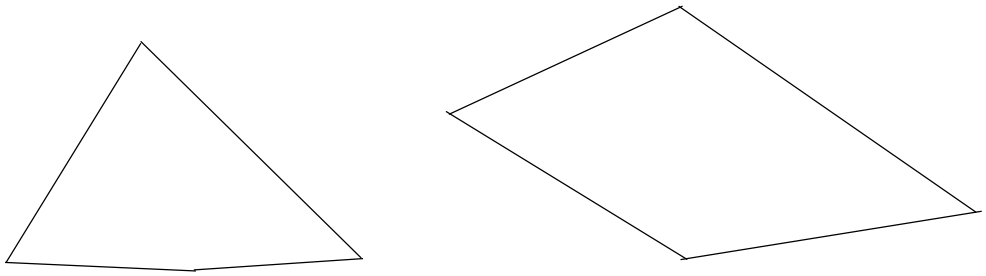


Fig 10. Examples of irregular grids

C. The edge reclassification

The purpose of this step is to regroup some edges into others, in this way, HAMSTRAN will be able to generate more regular grids.

1. A special regular edge is reclassified into a boundary edge.
2. A regular edge that shares a triangle with an anisotropic diagonal edge is reclassified as a boundary edge.

3. Any isotropic diagonal edge that shares a triangle with the newly generated boundary edge is reclassified into a regular edge.

4. A regular edge is reclassified into a corner edge if one of its 2 adjacent triangles has 2 boundary edges.

5. A regular edge is reclassified into an anisotropic diagonal edge if both adjacent triangles have 2 boundary edges and the length of the regular edge is greater than all 4 surrounding edges, If one of the 4 adjacent boundary edges is longer than the regular edge, then this edge is reclassified into a boundary edge.

6. A corner edge with 2 adjacent angles whose sum is greater than 180 degrees is reclassified into a boundary edge.

As can be seen from above, the edge classification process is very similar to that of Q-Tran, however there are many differences as well. For example, in Q-Tran, the classification of corner edge is done during the initial edge classification, but here it is done in the edge reclassification process. The reason for this is that during the edge reclassification process, the special regular edges are transformed into boundary edges, so some triangles that originally have only 1 boundary edge will end up having 2. Thus it is better to start generating the corner edges after turning all the special regular edges into boundary edges.

D. Generation of new vertices

In order to create quadrilateral grids, the vertices on the original background mesh aren't enough, so one needs to generate additional vertices. The vertices can be generated according to the following steps.

1. Create a vertex, the edge vertex, at the midpoint of each non-regular edge. The non-regular edges include the boundary edge, the isotropic diagonal edge and the anisotropic diagonal edge, but it doesn't include the corner edge.
2. Create a vertex, the face vertex, at the center of each triangular face. The coordinates of this vertex is shown in Figure 11 and can be calculated by equation 5.

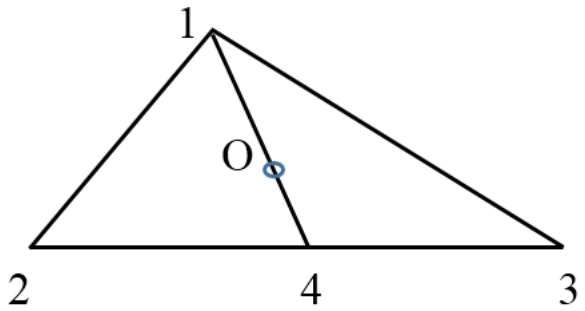


Fig 11. A sketch of the face vertex

$$x_4 = \frac{x_2 + x_3}{2} \; ; \quad y_4 = \frac{y_2 + y_3}{2} \; ; \quad z_4 = \frac{z_2 + z_3}{2}$$

$$\mathrm{x(O)} = x_4 + \frac{1}{3}(x_1 - x_4) \; ; \; \mathrm{y(O)} = y_4 + \frac{1}{3}(y_1 - y_4) \; ; \; \mathrm{z(O)} = z_4 + \frac{1}{3}(z_1 - z_4) \qquad (5)$$

The coordinate of face vertex O is (x(O), y(O), z(O))

This step is also different from Q-Tran. In Q-Tran, the face vertices are classified into regular face vertices and boundary face vertices. However, for this work it seems to be unnecessary, simply having face vertices is enough for the quad generation process.

E. Generating the quadrilateral grids

During this process, the quadrilateral grids are created according to the types of edges in the local triangles, the details are described in the following steps.

1. For each corner edge, generate quads as shown in Fig.12.

2. For each triangular face that has 3 boundary edges, split the triangle into 3 quadrilaterals.

3. For each anisotropic diagonal edge, transform the adjacent two triangular grids into 4 quadrilateral grids.

4. For each regular edge, select the 2 triangles adjacent to this edge. If a triangle contains a non-regular edge, then its corresponding edge vertex is considered as a side vertex. If a triangle doesn't contain any non-regular edge, then the face vertex of that triangle is classified as a side vertex. Use the two vertices of the regular edge as well as the corresponding side vertices to create a new quadrilateral grid.

The steps to generate quad grids can be summarized in Fig 12. After this process, an all-quad mesh system is finally created. However, this system contains many irregular vertices and irregular grids. A vertex is defined as a regular vertex if it has a valance of 4 (the valence should be 3 if it is on a type 1 boundary), or else it is defined as an irregular vertex. In order to improve the quality of the mesh system, a topological clean-up process, the merging process is introduced. This process is able to decrease the number of irregular vertices significantly.

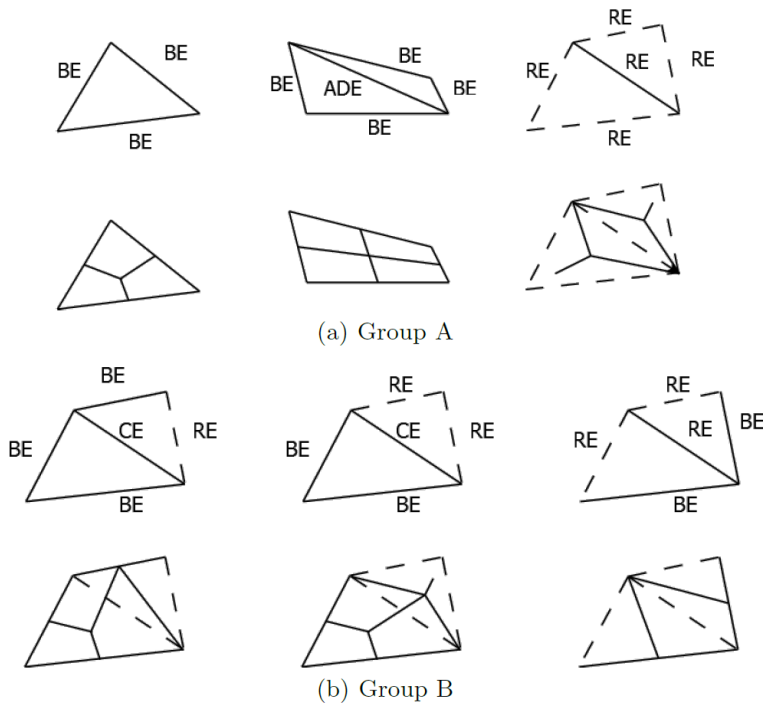


Fig 12. *A simple sketch showing the generation of quad grids*[9]

F. The merging process

The objective of this procedure is to reduce the number of irregular vertices as well as irregular grids. Different from other topological clean-up methods, this step does not change the coordinates of original vertices in the input tessellation. The merging process can be carried out by the following steps.

1. Select the vertices with a valence of 3, called the tri-valence vertices, a tri-valence vertex must not be on the boundary.

2. Select the quad grids that contains 2 tri-valence vertices.

3. Select the vertices that are adjacent to only 2 quads as well as the quad grids connected to them.

4. The grids selected above are called the initial irregular grids.

5. Use the face collapse technique to destroy the selected irregular grids, the face collapse technique is demonstrated in Figure 13. Keep repeating the above processes until there are no initial irregular grids left.

6. After all the initial irregular grids are destroyed, select the grids that contain only one tri-valence vertex, called the secondary irregular grids. The secondary irregular grids can also be destroyed using the face collapse method. Don't carry out this step if it makes the quality of the mesh even worse.

The quality of the mesh system is improved dramatically after this process. The number of irregular grids and irregular vertices are reduced, as can be seen in Fig 13.

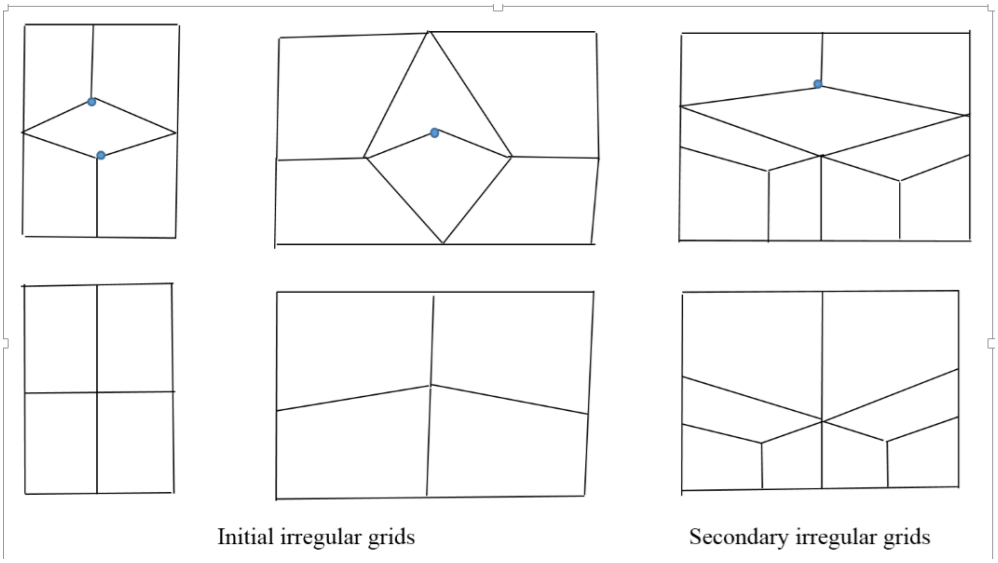


Fig 13. Illustration of the face collapse process

2.1.2. Implementation on structure-unstructured hybrid mesh

Sometimes the input tessellation consists of both unstructured triangular grids as well as structured quadrilateral grids. The unstructured part of the mesh can be dealt with according to the steps of the previous chapter. However, if we simply implement HAMSTRAN on the unstructured part, it would leave a lot of hanging vertices inside the mesh, as can be seen in Fig 14a. So there needs to be a way to make sure all these hanging vertices are also connected to an edge. Of course one can simply subdivide the structured part of the mesh, but that would dramatically increase the number of grid cells and sometimes modifying the structured mesh might lead to inaccuracies of the solution, so a new and better method is required, called the node capturing.

The steps of the node capturing process:

1. Select all the edges adjacent to both an unstructured grid and a structured grid; usually these edges would form a smooth long loop, called the boundary loop, otherwise there is no choice but to subdivide the structured mesh.
2. Select all the structured quad grids that share an edge with the boundary loop.
3. Split each surface grid into 3 new quadrilateral grids, just like in Fig 14b.
4. In this way, all the hanging vertices on the boundary would also be connected to a quad grid. However, this process will only eliminate all hanging nodes if there are even number of edges on each boundary loop.
5. If there are odd number of edges on the boundary layer, one needs to slightly change the algorithm in order to get rid of all the hanging nodes, the specific method depends on the geometry being implemented.

After the node capturing process, all the hanging vertices are connected to the newly generated grids and only a small number of structured grids are modified. So after this process, the structured part of the mesh remains mostly untouched and the number of grids also stay approximately the same. Fig 15 provides an example of the node capturing process on a 3D hybrid mesh.

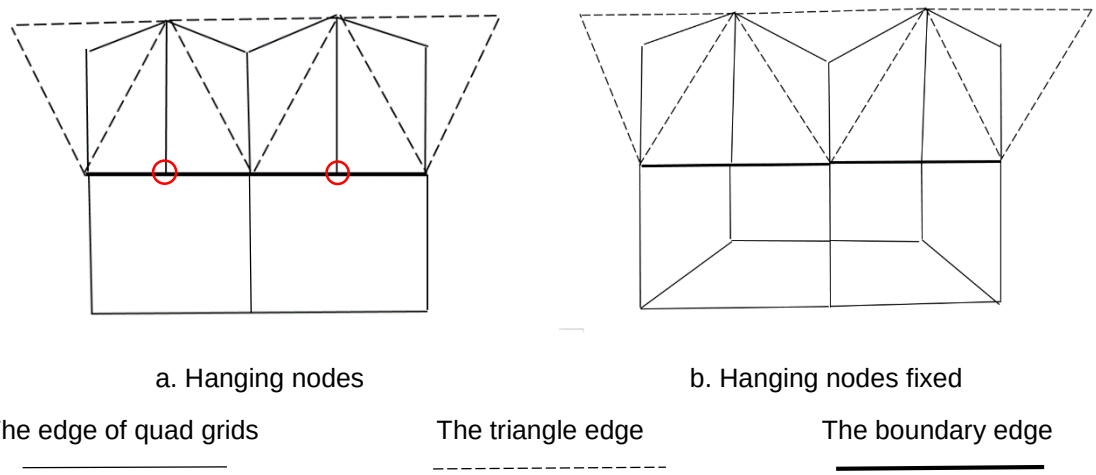


Fig 14. A sketch of the node capturing process

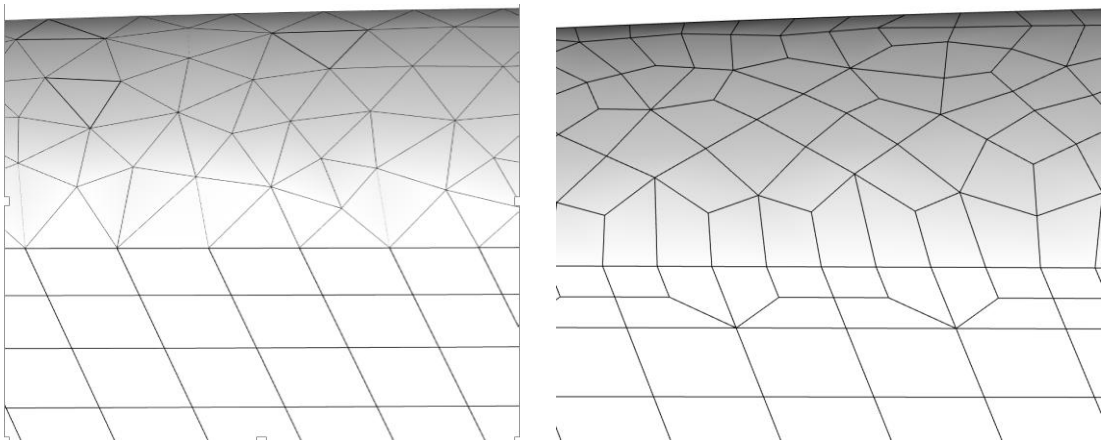


Fig 15. An example of the node capturing process

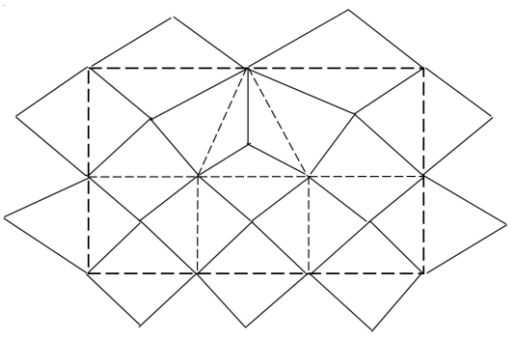
2.1.3. Implementation on unstructured quad dominant mesh

In this kind of input tessellation, the entire mesh is unstructured and the quadrilaterals and triangles are mixed together. The easiest way to transform it into an all-quad mesh, is of course to subdivide every quadrilateral into 2 triangles, and then implement the steps directed towards all-triangular input tessellation, also known as the multi-step method. However, this method doesn't always generate good quality grid cells, so another faster and better algorithm is introduced.

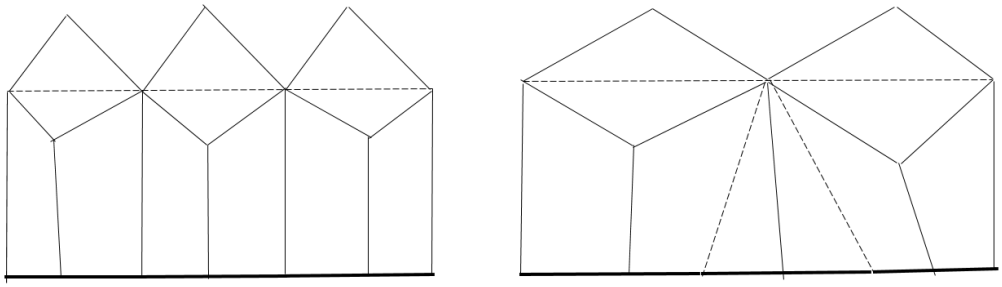
The steps to transform a quad-dominant mesh into an all-quad mesh:

1. An edge that is only adjacent to one grid face is classified as a type 1 boundary edge.
2. An edge adjacent to 2 grid faces and the intersection angle of their normal vectors exceeds a certain value, this edge is classified into a type 2 boundary edge.
3. Any edge that isn't a boundary edge is classified as a regular edge.
4. Generate a new vertex at the center of each grid face, the face vertex. Generate a new vertex at the midpoint of each boundary edge, the edge vertex.
5. Generate the quad grids according to Fig 16.

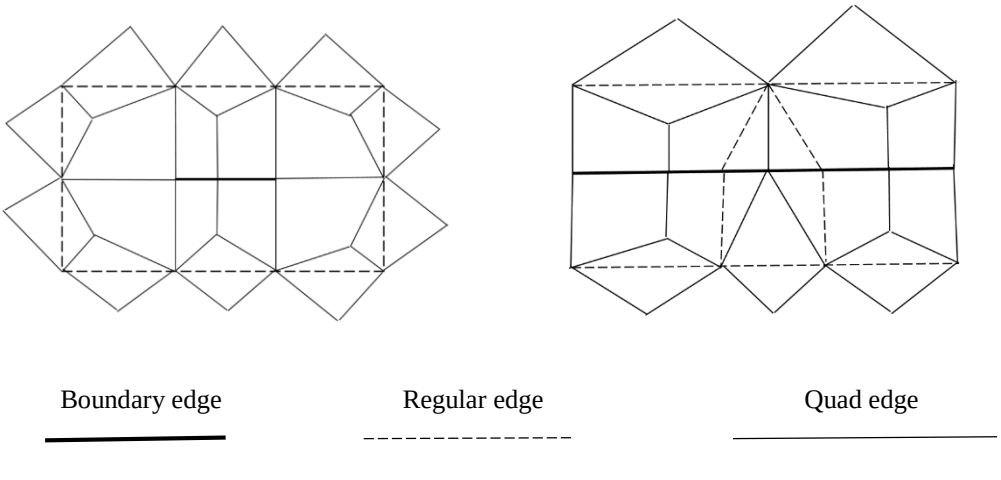
In this way, a quad-dominant mesh can be easily transformed into an all-quad mesh. Since most vertices generated by this method are regular vertices, a merging-process isn't necessary. The downside of this process is that it will increase the number of grids by roughly twice, but it is still better than the Catmull-Clark subdivision process, which would increase the number of grids by approximately 4 times.



a. Quad transformation around regular edge



b. Quad transformation around type 1 boundary edges



c. Quad transformation around type 2 boundary edges

Fig16. The implementation of HAMSTRAN on quad-dominant mesh

2.1.4. Generate the strand templates

In order to create 3D strand grids, simply having a surface mesh isn't enough, strand templates extruding from the surface of the geometry are also required. The templates should be approximately normal to the geometry surface. They shouldn't cross each other either, or else the resulting strand grids would overlap each other. Unfortunately, Q-Tran doesn't provide us with any methods for the generation of strand templates. So HAMSTRAN developed the weighted average method.

Steps for the weighted average method：

1. For each vertex, select all the edges connected to the said vertex.

2. Calculate the angles between the adjacent edges.

3. Calculate the normal vector of each angle by calculating the cross product of its two corresponding edges, normalize the vectors, making sure its length equal to one.

4. Calculate the weighted average of all the angle vectors, using equation 6, the resulting vector is the strand template of this vertex, normalize this vector as well. This step can be visualized in Fig 17, which uses the example of a regular vertex. The strand templates of the irregular vertices can be calculated in the same way.

$$\theta_1 = \operatorname{acos}\left(\frac{\overrightarrow{OB}.\overrightarrow{OC}}{|\overrightarrow{OB}||\overrightarrow{OC)}|}\right), \theta_2 = \operatorname{acos}\left(\frac{\overrightarrow{OA}.\overrightarrow{OC}}{|\overrightarrow{OB}||\overrightarrow{OC)}|}\right), \theta_3 = \operatorname{acos}\left(\frac{\overrightarrow{OA}.\overrightarrow{OD}}{|\overrightarrow{OB}||\overrightarrow{OC)}|}\right), \theta_4 = \operatorname{acos}\left(\frac{\overrightarrow{OB}.\overrightarrow{OD}}{|\overrightarrow{OB}||\overrightarrow{OC)}|}\right)$$

$$\theta = \theta_1 + \theta_2 + \theta_3 + \theta_4 \tag{6}$$

$$\text{Normal(O)} = \left(\overrightarrow{OB} \times \overrightarrow{OC}\right) \times \left(\frac{\theta_1}{\theta}\right) + \left(\overrightarrow{OC} \times \overrightarrow{OA}\right) \times \left(\frac{\theta_2}{\theta}\right) + \left(\overrightarrow{OA} \times \overrightarrow{OD}\right) \times \left(\frac{\theta_3}{\theta}\right) + \left(\overrightarrow{OD} \times \overrightarrow{OB}\right) \times \left(\frac{\theta_4}{\theta}\right)$$

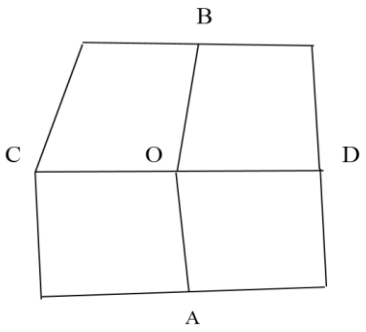


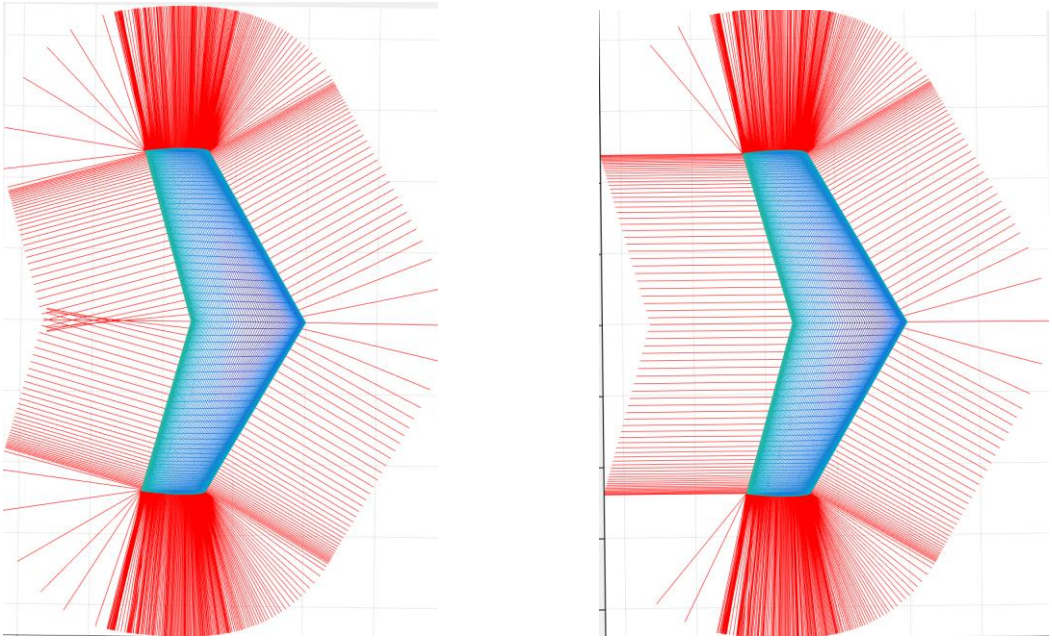
Fig 17. Generating the strand templates

5. For each vertex, select all the surrounding vertices which share a grid with the said vertex.

6. Calculate the average of all the templates of the selected vertices, using this newly calculated vector to replace the original template. In this way, each template becomes more parallel with its surrounding templates.

7. Repeat steps 5 and 6 until the strands no longer cross each other. If this is impossible, then repeat until a minimum number of strands cross each other.

Of course, the weighted average method is just a general method, in order to generate strand templates of decent quality, some adjustments must be made according to each individual geometry. For example, in the concave regions of a wing mesh, all the strand templates should be made parallel to the template at the peak, as shown in Fig 18. This process is called the strand smoothing.



a. Strands generated by the weighted average method b. Strands after the smoothing process

Fig 18. Examples of the strand smoothing process

As is shown in Fig 18, the strands generated by the weighted average method would cross each other in a very short distance at the trailing edge. After the strand smoothing process, those strands became parallel with each other, thus able to generate 3D strand grids in the future. Of course this is just one example, this smoothing process must be carried out according to each individual geometry.

2.2. Examples of the HAMSTRAN algorithm

In order to test the efficiency of this method, the HAMSTRAN algorithm is implemented on several geometries. The criteria used to test the quality of the mesh includes the number of grids, the average skewness and the number of irregular vertices. The skewness is used to measure the shape of a quad cell. Ideally speaking, all cells should be rectangles and the skewness should be 0, however, in an unstructured mesh system, this is nearly impossible. Generally speaking, the smaller the skewness, the better the quality of the mesh. The skewness is calculated by equation 7. In order to make a comparison, the Catmull-Clark subdivision algorithm is also carried out on these geometries. The details will be explained in the next section.

$$\text{Skewness}=\max\left[\frac{\theta_{max}-\theta_e}{180-\theta_e}, \frac{90-\theta_{min}}{90}\right] \quad (7)$$

θ_{max}: *The biggest angle in the grid* θ_{min}: *The smallest angle in the grid*

θ_e: *For a quadrilateral grid, this value is* 90

2.2.1. The hexagon mesh

Before moving to more complicated geometries, HAMSTRAN is first implemented on a simple 2D mesh, the hexagon mesh. This mesh is comprised of uniform triangles representing internal 2D geometry. The details of this mesh can be seen in Figure 19. As is shown in this figure, the HAMSTRAN algorithm works well on meshes of various sizes.

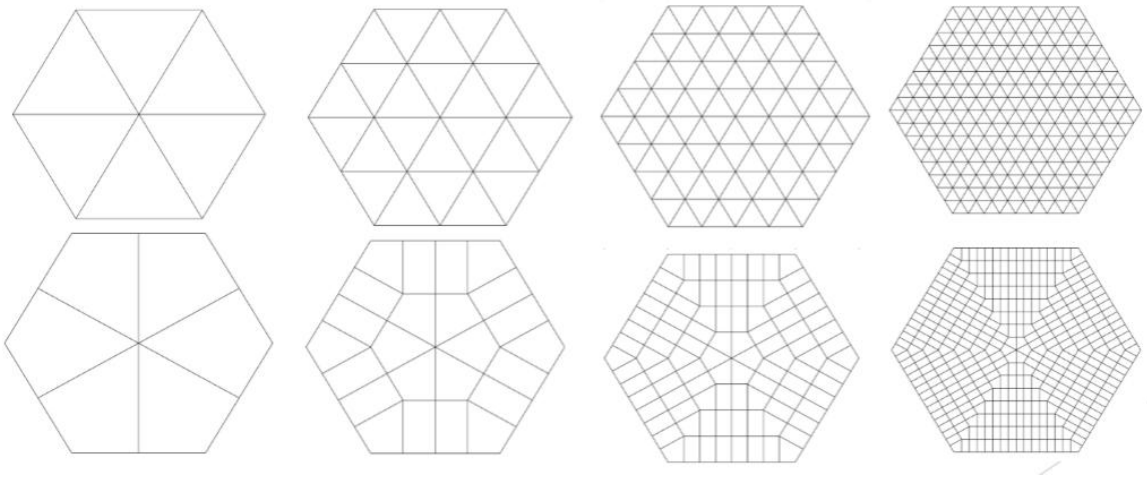
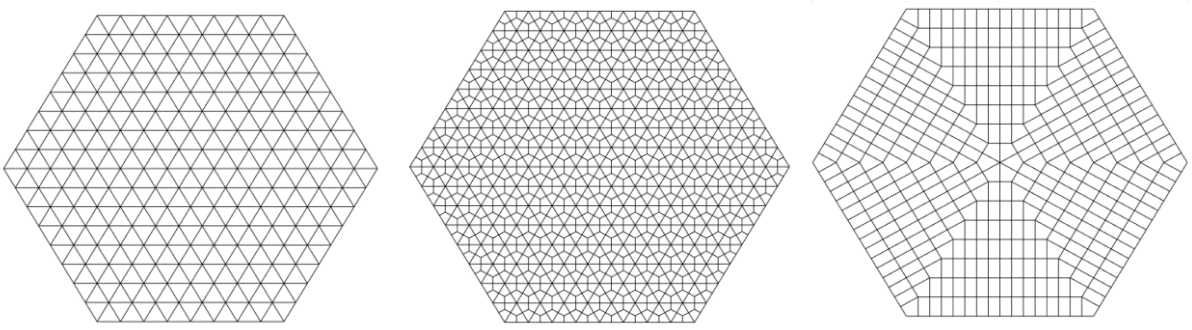


Fig 19. The implementation of HAMSTRAN on 2D hexagon mesh of varying size



a. The original background mesh b. Subdivision c. HAMSTRAN

Fig 20. Comparison between the subdivision algorithm and the HAMSTRAN algorithm

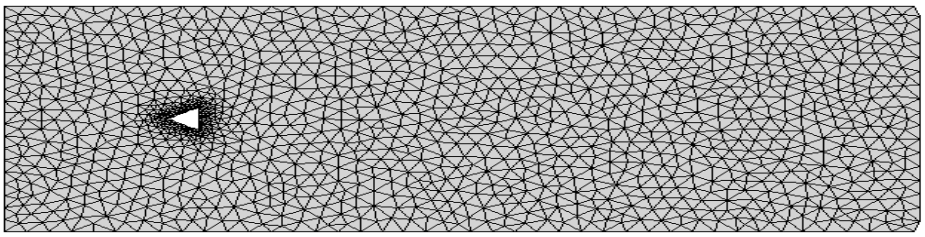
	Number of original triangles	Number of original quads	Number of final quad grids	Average skewness	Regular Vertices	Irregular vertices	Percentage of regular vertices
Subdivision	384	0	1152	0.333	606	595	50.46%
HAMSTRAN	384	0	384	0.042	426	7	98.38%

Table 1. Quality comparison of the hexagon mesh

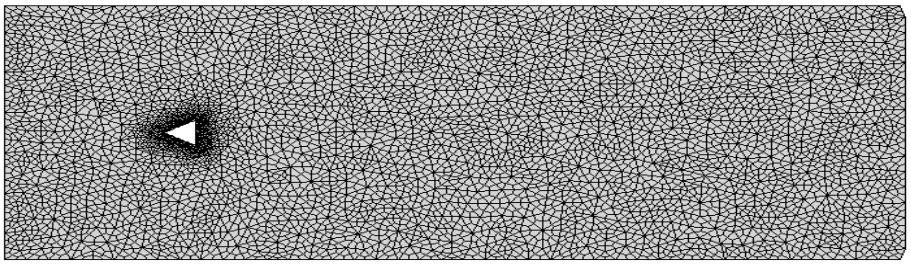
As can be seen from Table 1, the mesh created by HAMSTRAN generates a far higher percentage of regular vertices comparing to the subdivision process. The average skewness is also much lower, indicating there are more regular grids in the mesh. The number of grids stayed approximately the same before and after the transformation process when using HAMSTRAN, but it increased by about 3 times when using the subdivision method. So from what one can see in this case, HAMSTRAN is overall superior to the Catmull-Clark subdivision algorithm. However, this mesh is too simple, one needs to implement it on more complicated mesh systems in order to test the efficiency of HAMSTRAN.

2.2.2. The 2D inviscid wedge mesh

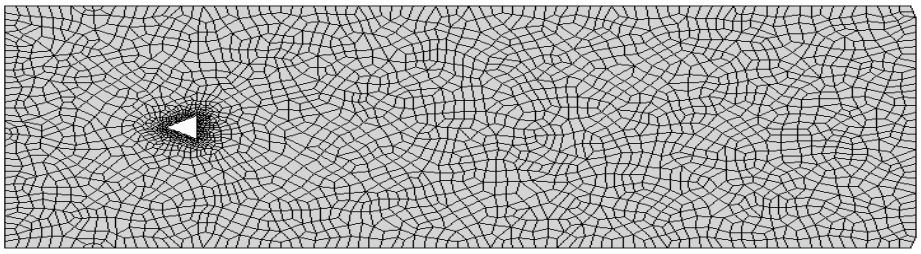
This mesh is an external all-triangular mesh around a *triangular wedge*[10[. As we know, the flow near the wedge is much more complicated than the flow in the far field, thus the velocity gradient is much higher in the near body region. As a result, smaller grid cells are required to capture the higher gradient, as can be seen in Fig 21.



a. The original background mesh

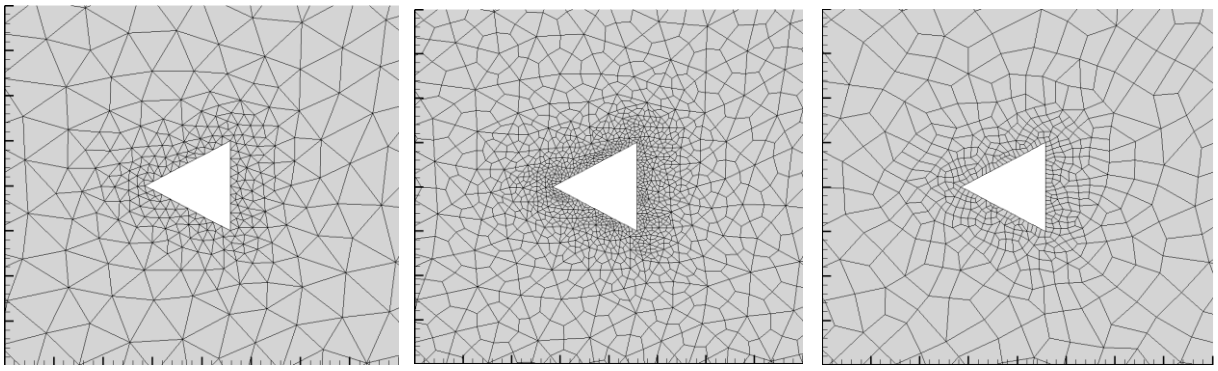


b. The mesh created by the subdivision algorithm



C, The mesh created by HAMSTRAN

Fig 21. The 2D inviscid wedge mesh



a. The background mesh b. Subdivision c. HAMSTRAN

Fig 22. A blow up near the triangular wedge

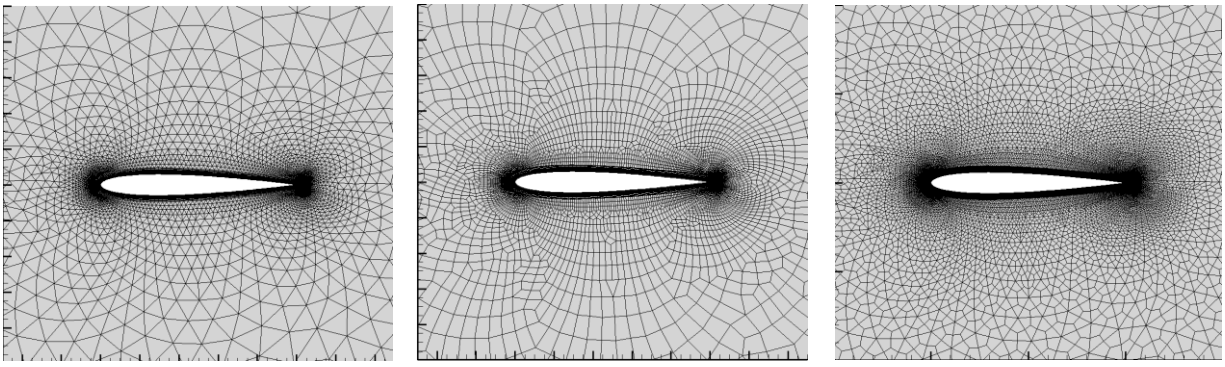
	Number of original triangles	Number of original quads	Number of final quad grids	Average skewness	Regular Vertices	Irregular vertices	Percentage of regular vertices
Subdivision	2520	0	7560	0.4463	3936	3772	51.06%
HAMSTRAN	2520	0	2583	0.3020	1939	792	71.00%

Table 2. Quality comparison of the inviscid wedge mesh

As can be seen from Table 2, the mesh created by HAMSTRAN generates a higher percentage of regular vertices comparing to the subdivision process. Although due to the complexity of this mesh, the percentage of regular vertices isn't as high as in the first case, but overall, this is still a huge improvement. Some irregular vertices are required in order to allow for the varying size of the resulting quads. The average skewness is also lower, indicating there are more regular grids in this mesh comparing to the subdivided mesh. The reason the skewness is not as small as the first case is due to the complexity of this mesh. The number of grids stays approximately the same before and after the transformation process when using HAMSTRAN, but it increased by about 3 times when using the subdivision method. Generally speaking, HAMSTRAN works well for this more complicated mesh.

2.2.3. The 2D NACA0012 hybrid mesh

The *NACA0012 airfoil*[10] is an airfoil shape applied in many wings and also used in a variety of experiments. Since this case studies a turbulent flow around the airfoil, very thin layers of structured grids are required around the airfoil to capture the complexity of the boundary layer, making this mesh a hybrid mesh.



a. The background mesh b. HAMSTRAN c. Subdivision

Fig 23. The 2D NACA0012 hybrid mesh

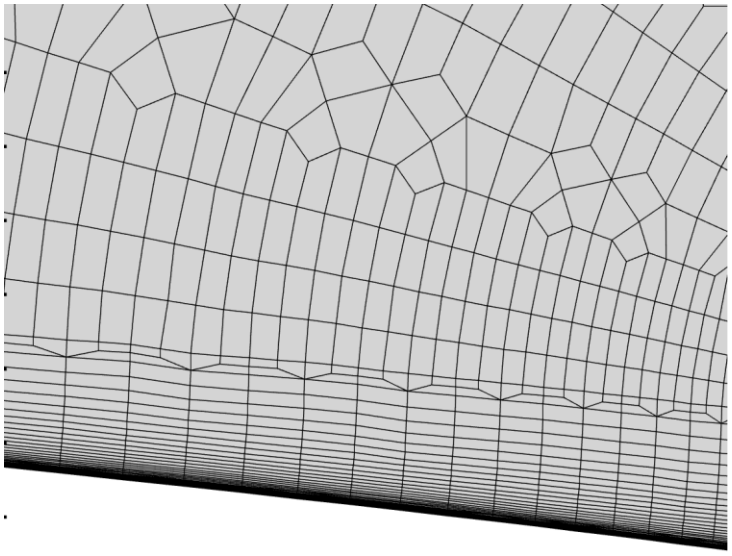


Fig 24. A blow up to show the node capturing process

	Number of original triangles	Number of original quads	Number of final quad grids	Average skewness	Regular Vertices	Irregular vertices	Percentage of regular vertices
Subdivision	10407	20740	114181	0.1397	98714	15802	86.20%
HAMSTRAN	10407	20740	32559	0.1041	30046	2698	91.76%

Table 3. Quality comparison of the NACA0012 mesh

As can be seen from Table 3, the skewness and the percentage of regular vertices aren't very different for both methods. This is because the original mesh is comprised of mainly structured quad grids, so in order to make the advantages of HAMSTRAN more clear, another table is made only for the unstructured part of the mesh.

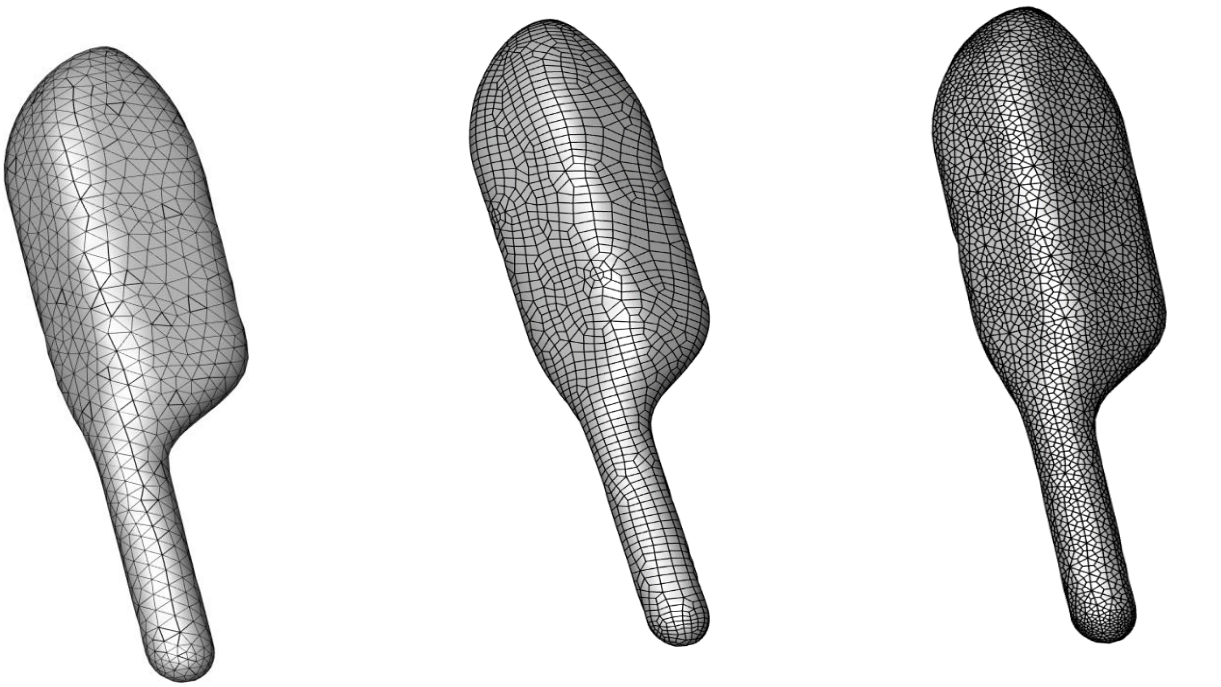
	Number of original triangles	Number of original quads	Number of final quad grids	Average skewness	Regular Vertices	Irregular vertices	Percentage of regular vertices
Subdivision	10407	0	31221	0.4238	15918	15800	50.19%
HAMSTRAN	10407	0	10897	0.2087	9385	2009	82.37%

Table 4. Quality comparison of the unstructured part of the mesh

In this table, the advantage of HAMSTRAN becomes much more obvious. The average skewness is much lower, the percentage of regular vertices also becomes much higher. So as we can see, HAMSTRAN works well even for a hybrid mesh. Of course, up until now, all the test cases are in 2D. In order to test the full capacity of HAMSTRAN, one would like to test some cases in 3D as well.

2.2.4. The robin fuselage surface mesh

The *robin fuselage*[12] is a fuselage widely used by helicopters. It is also widely applied in both experimental and computational tests. This is an all-triangular mesh that covers the entire surface of the geometry.



a. The original background mesh b. The HAMSTRAN mesh c. the subdivdided mesh

Fig 25. The robin fuselage surface mesh

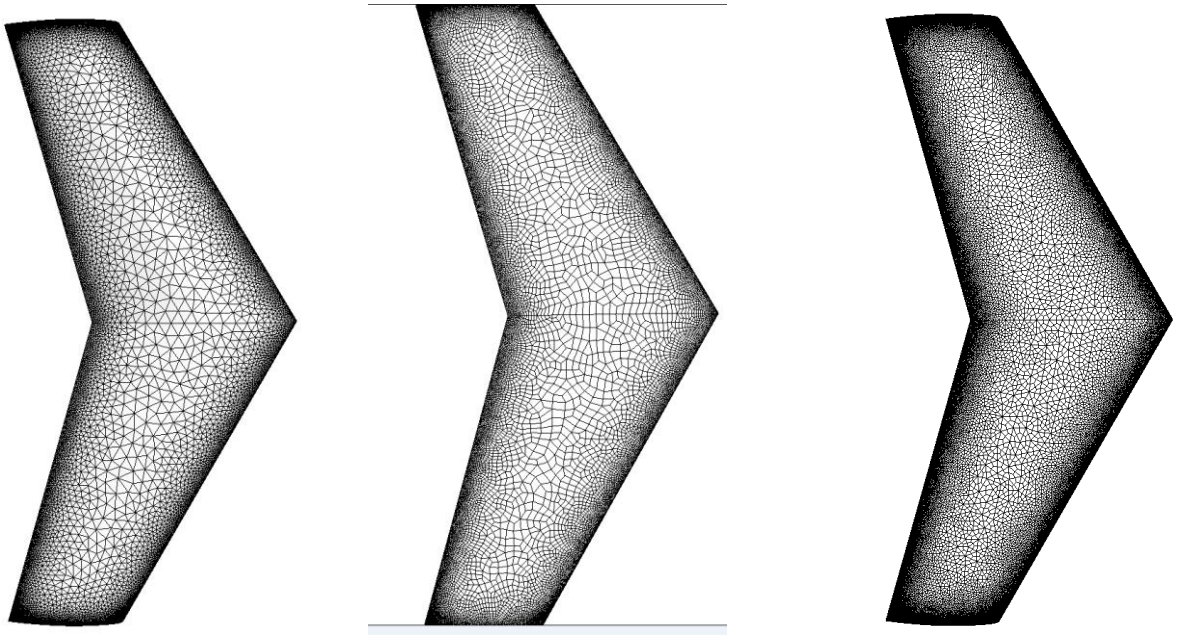
	Number of original triangles	Number of original quads	Number of final quad grids	Average skewness	Regular Vertices	Irregular vertices	Percentage of regular vertices
Subdivision	2096	0	6288	0.3962	3147	3143	50.03%
HAMSTRAN	2096	0	2180	0.1950	1754	428	80.38%

Table 5. Quality comparison of the robin fuselage surface mesh

As can be seen from Table 5, HAMSTRAN also works well on 3D surface meshes. It has a much lower skewness and the percentage of regular vertices is much higher. Of course, this is a relatively simple mesh and all grids have approximately equal sizes. In order to test the full capability of HAMSTRAN, it should be implemented on more complicated meshes.

2.2.5. The OneraM6 wing unstructured mesh

Figure 26 looks at surface meshes of the *OnearM6 wing*[10], which is a classic transonic test case for CFD. Different from the robin fuselage surface mesh, this mesh has clustering near the leading and trailing edge. This is to capture the high pressure gradient in those areas, making it a more complicated mesh.



a. The original background mesh b. The HAMSTRAN mesh c. The subdivided mesh

Fig 26. The OneraM6 wing surface mesh

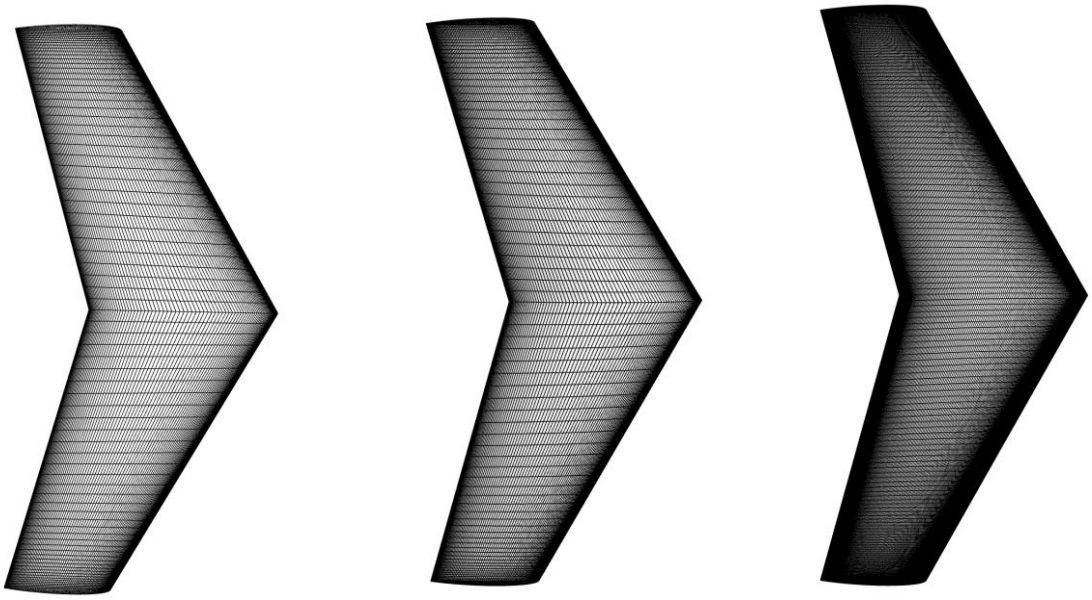
	Number of original triangles	Number of original quads	Number of final quad grids	Average skewness	Regular Vertices	Irregular vertices	Percentage of regular vertices
Subdivision	45310	0	135930	0.4361	68484	67448	50.38%
HAMSTRAN	45310	0	49597	0.2876	34076	15523	68.70%

Table 6. Quality comparison of the OneraM6 wing surface mesh

Although this mesh is more complicated than the previous one, HAMSTRAN still performed quite well, as can be seen from Table 6. Comparing to the subdivision algorithm, HAMSTRAN is able to generate a mesh with much lower average skewness and a higher percentage of regular vertices, making it a better method.

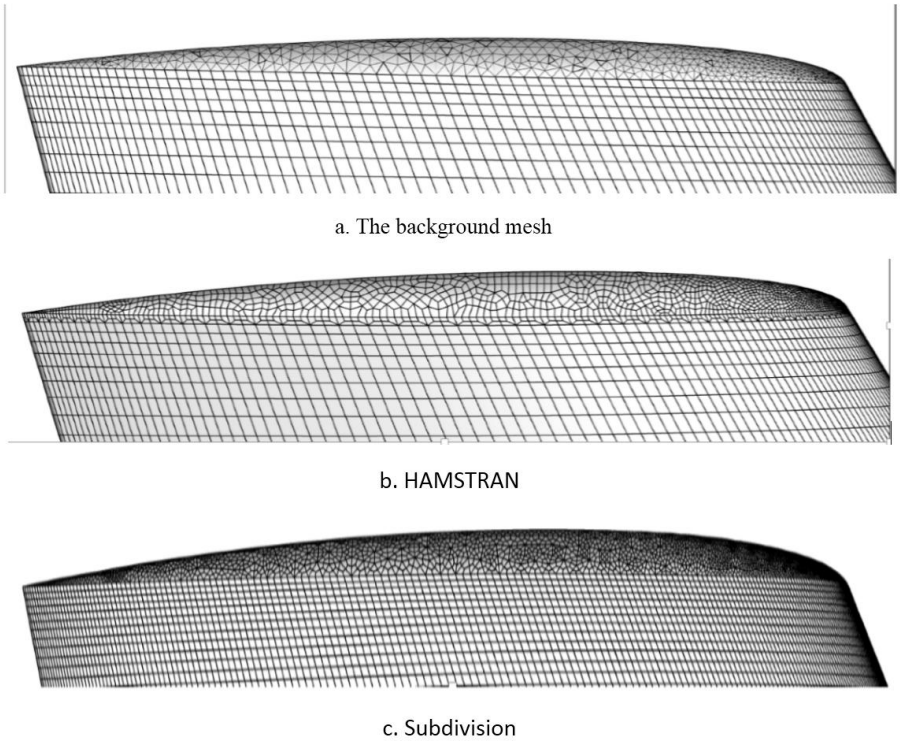
2.2.6. The OneraM6 wing structure-unstructured hybrid mesh

This is another mesh for the OneraM6 wing. Since there is little change in flow across the span wise direction, except for near the tip, one could make a structured mesh for the main body of the wing while only the tip is comprised of unstructured grids. The tip of the wing is comprised of triangular grids while the main body is comprised of structured quadrilateral grids. Since HAMSTRAN is mainly implemented on the tip of the wing, Fig 27 doesn't really show much information, thus another figure, Fig 28 is made only for the tip of the wing. Just like the NACA0012 mesh, two tables are also made here, one for the entire mesh system and one only for the unstructured part of the mesh.



a. The original background mesh b. HAMSTRAN c. Subdivision

Fig 27. The OneraM6 wing hybrid mesh



a. The background mesh

b. HAMSTRAN

c. Subdivision

Fig 28. A blow up around the tip of the wing

	Number of original triangles	Number of original quads	Number of final quad grids	Average skewness	Regular Vertices	Irregular vertices	Percentage of regular vertices
Subdivision	4728	14508	72216	0.2877	64998	7220	90%
HAMSTRAN	4728	14508	20400	0.2715	18148	2254	88.95%

Table 7. Quality comparison of the OneraM6 hybrid wing mesh

	Number of original triangles	Number of original quads	Number of final quad grids	Average skewness	Regular Vertices	Irregular vertices	Percentage of regular vertices
Subdivision	4728	0	5148	0.4513	7406	7152	50.87%
HAMSTRAN	4728	0	14184	0.2997	3826	1696	69.29%

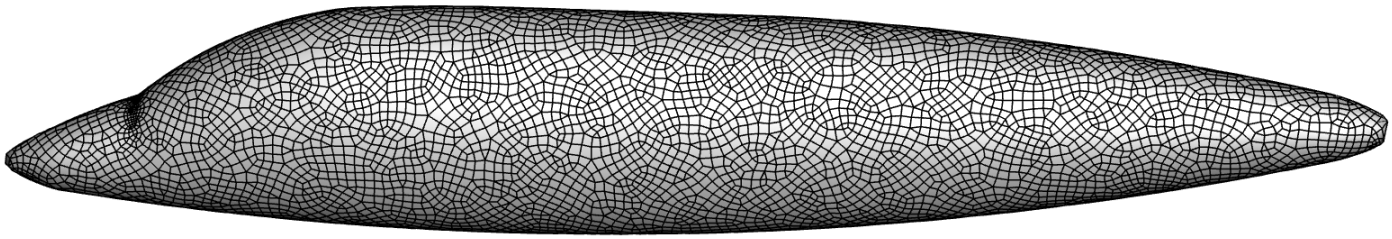
Table 8. Quality comparison at the tip of the wing

As we can see from Tables 7 and 8, HAMSTRAN generates far fewer grids comparing to Catmull-Clark subdivision, and the quality of the mesh improved, especially when one only compares the unstructured part of the mesh. Using HAMSTRAN, the average skewness is lower and the percentage of regular grids is much higher.

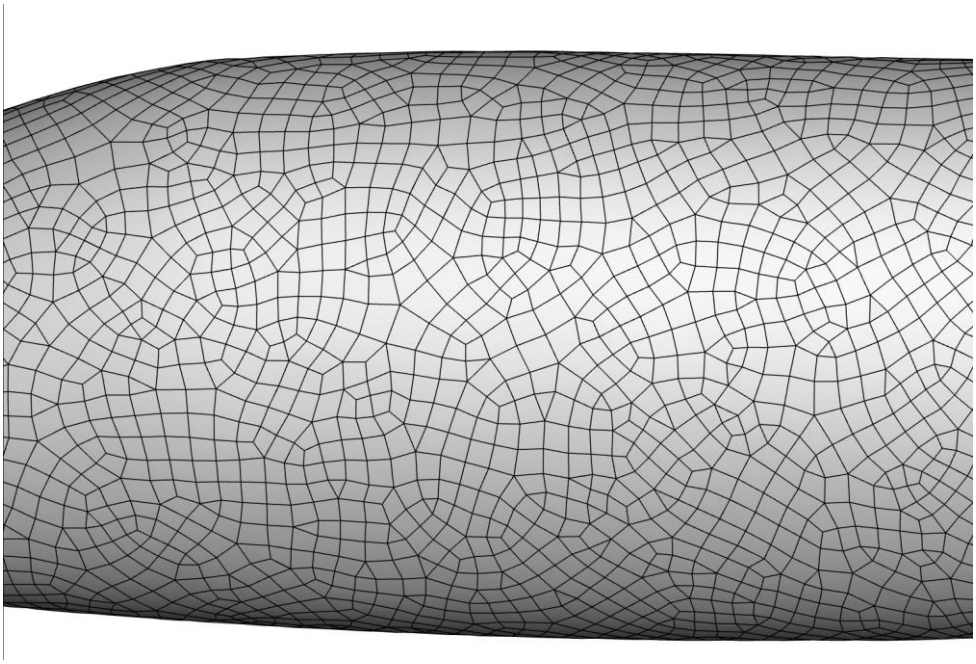
2.2.7. The quad dominant x2 fuselage mesh

The x2 fuselage is another fuselage used by helicopters. Different from the previous meshes, this input tessellation has a mixture of unstructured quadrilateral and triangular grids, as can be seen in Figure 29. In order to make a comparison, this work introduces the multi-step algorithm, which subdivides every quad grid into 2 triangles, and then implements the triangular HAMSTRAN

algorithm. The results from the quad-dominant HAMSTRAN (Figure 32) are compared with results from both the multi-step method (Figure 30) and the subdivision method (Figure 31).

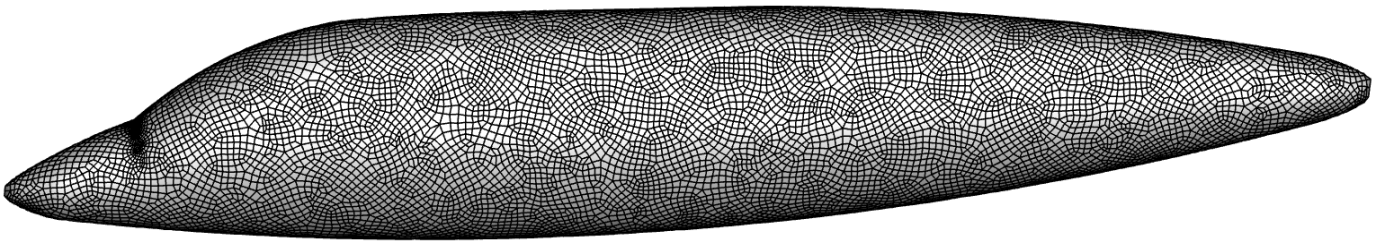


a. An overview of the entire mesh system

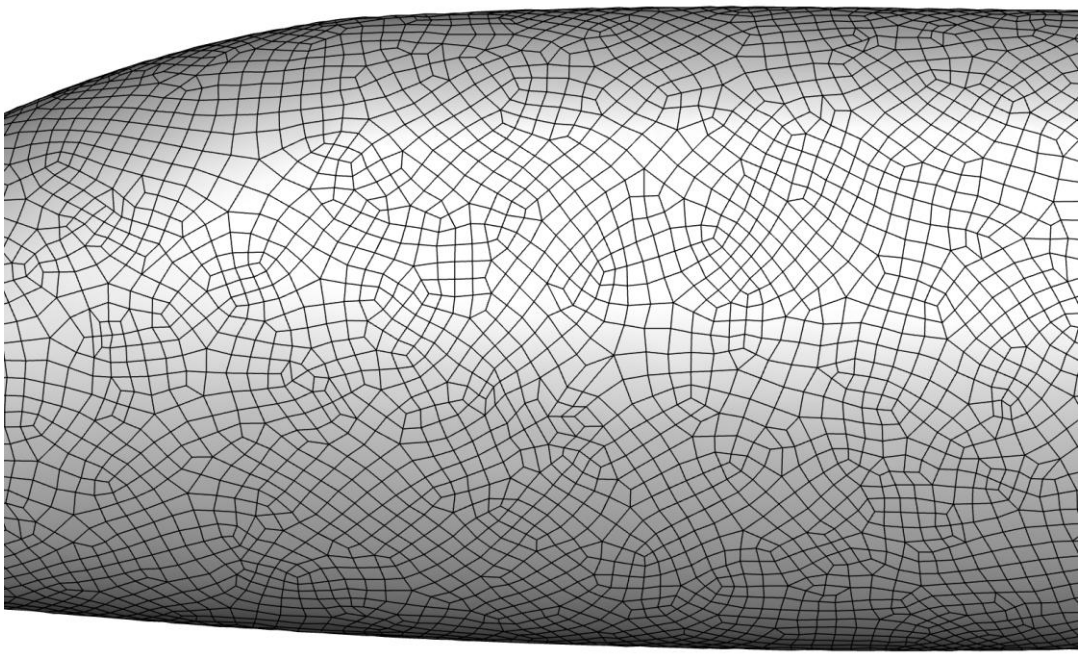


b. A blow up of the details

Fig 29. The original quad-dominant mesh for x2 fuselage

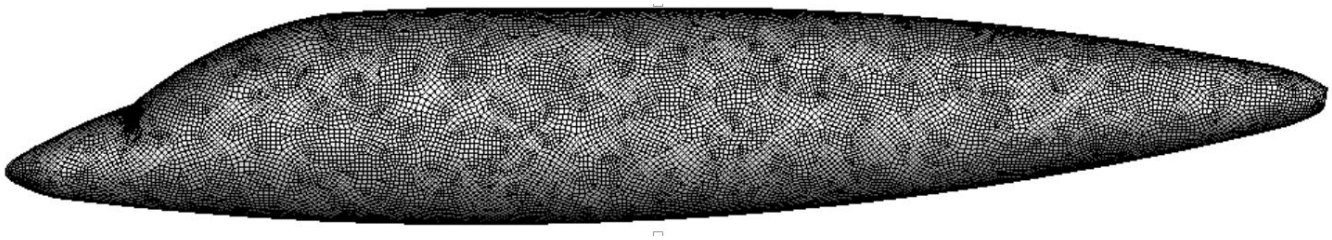


a. An overview of the entire mesh

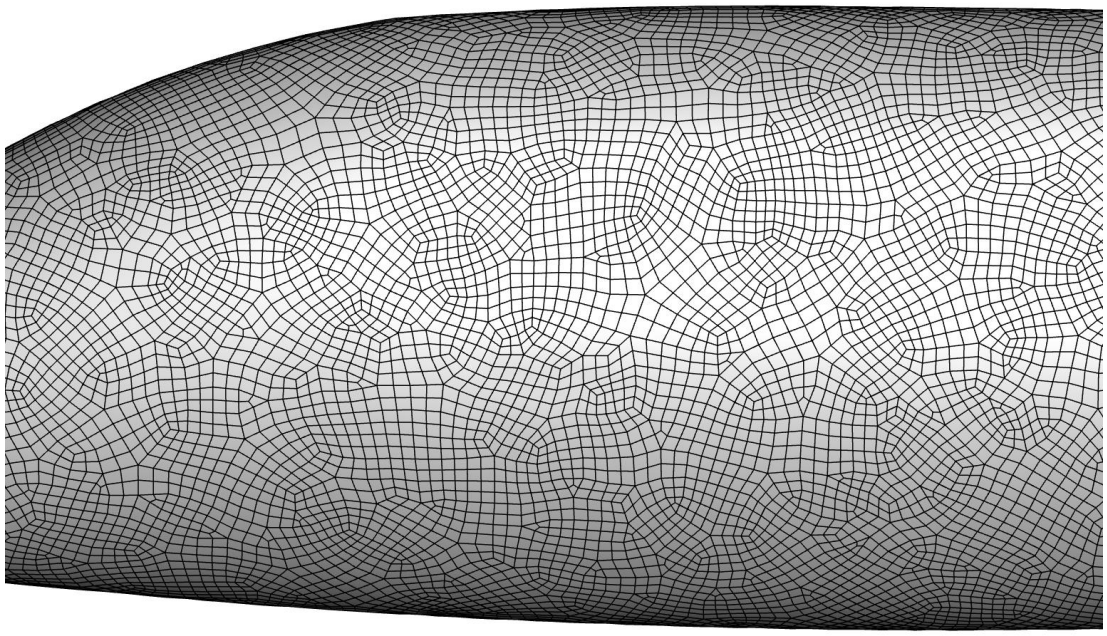


b. A blow up to show the details

Fig 30. All-quad X2 fuselage mesh created by the multi-step method

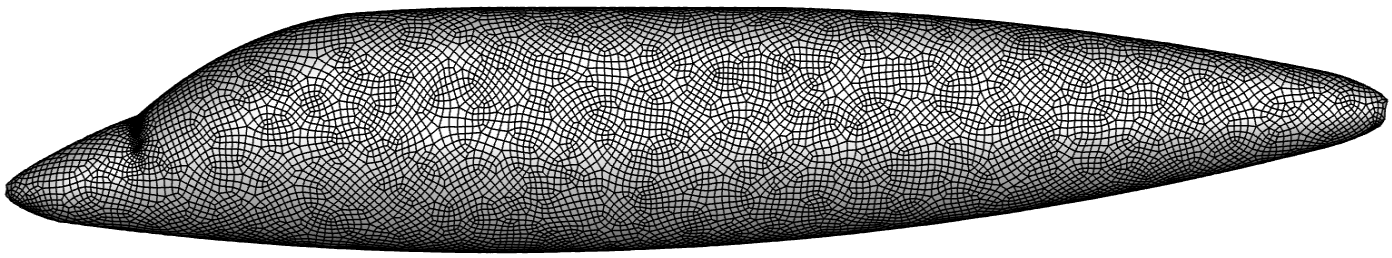


a. Overview of the entire mesh

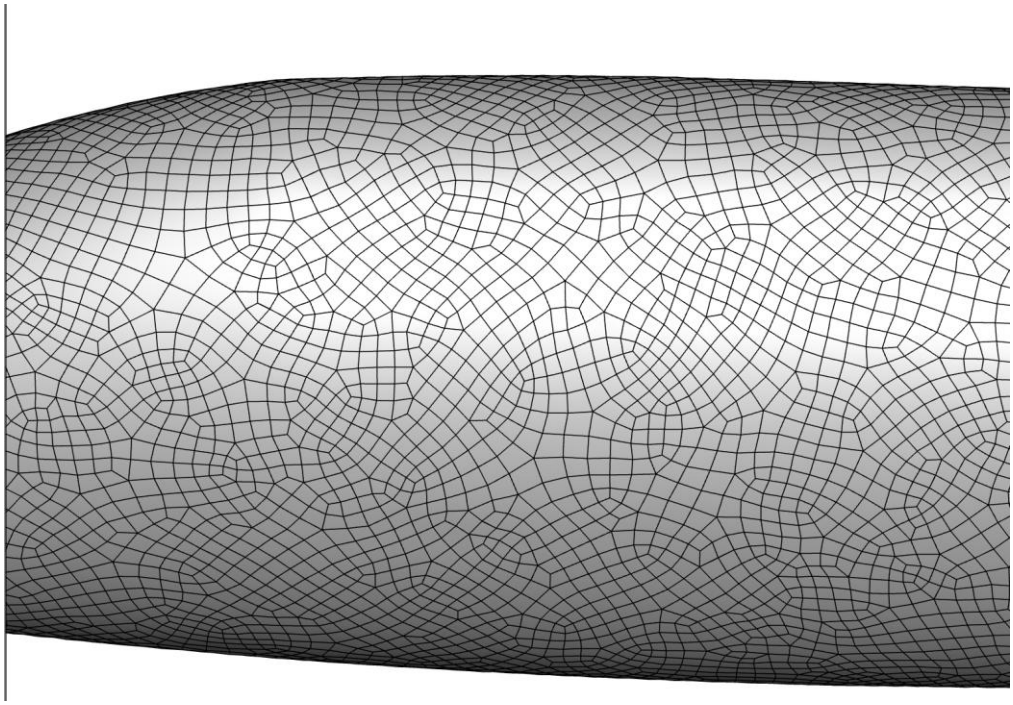


b. A blow up to show the details of the mesh

Fig 31. All-quad x2 fuselage mesh created by the Catmull-Clark subdivision algorithm



a. Overview of the entire mesh



b. A blow-up to show the details of the mesh

Fig 32. All-quad x2 fuselage mesh created by quad-dominant HAMSTRAN

	Number of original triangles	Number of original quads	Number of final quad grids	Average skewness	Regular Vertices	Irregular vertices	Percentage of regular vertices
Multi-step	764	8250	17758	0.2130	14783	2977	83.24%
HAMSTRAN	764	8250	17646	0.1890	15080	2568	85.46%
Subdivision	764	8250	35292	0.1665	32726	2568	92.72%

Table 9. Quality comparison of the quad-dominant x2 fuselage mesh

As can be seen from Table 9, the results for the quad-dominant background mesh aren't as good as those from the all-triangular meshes. This time the subdivision algorithm actually produces grids with the lowest skewness, but at the same time, the number of cells increased almost four times. Both the multi-step method and HAMSTRAN increase the number of grids by two times, but HAMSTRAN generates more regular vertices and has a lower skewness. So here the only advantage of the HAMSTRAN algorithm is producing fewer number of grids comparing to the subdivision method.

2.3. HAMSTRAN algorithm on more complicated geometries

This is a helicopter hub used to connect the fuselage to the rotor blades. It is much more complicated than the all previous geometries in this paper. Figure 33 shows the original all-triangular background mesh while Figure 34 shows the all-quad mesh created by HAMSTRAN.

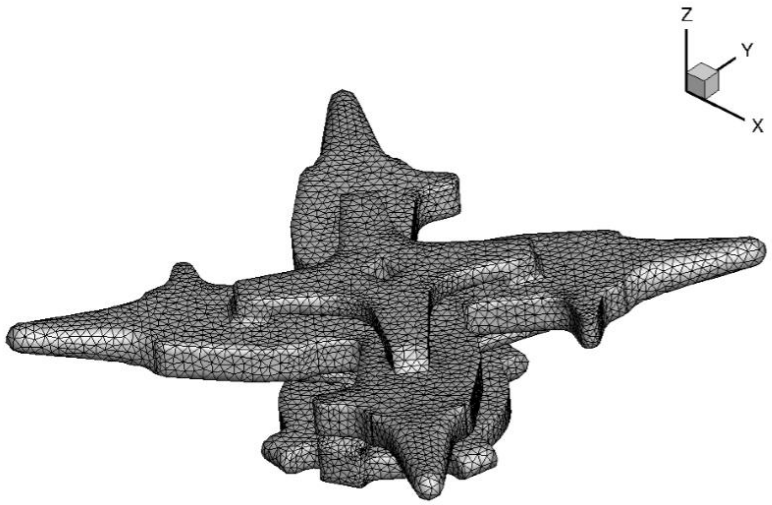


Fig 33. The original background mesh of the helicopter hub

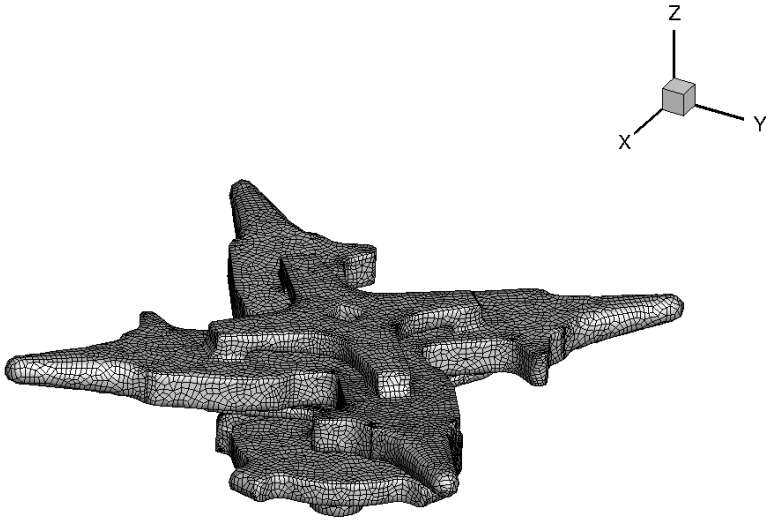


Fig 34. The HAMSTRAN mesh for the helicopter hub

This is the wing of fierce fighter jet Silent Viper, it has a very sharp point in the front, as is shown in Figure 35, but HAMSTRAN is still able to generate an all-quad mesh of decent quality, as can be seen in Figure 36.

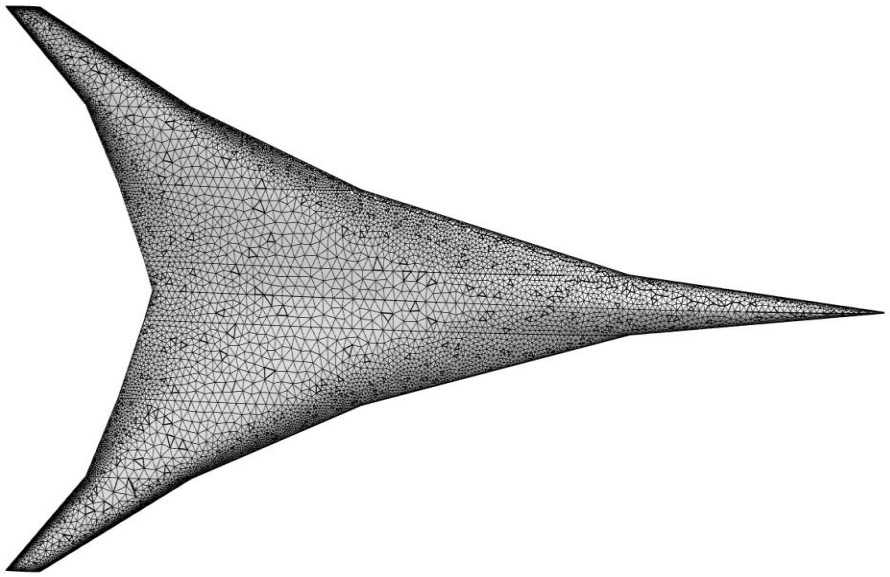


Fig 35. The all triangular background mesh of a fighter wing

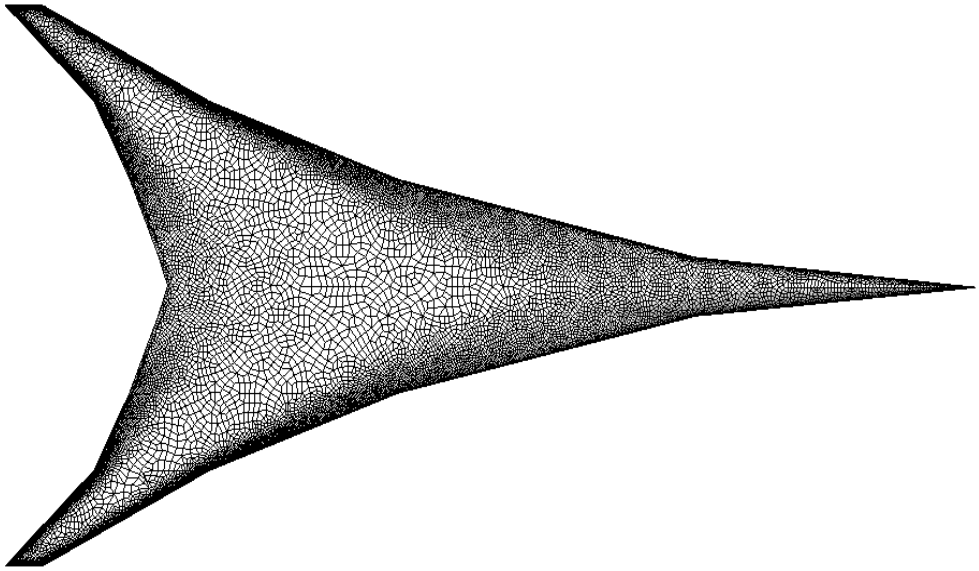


Fig 36. The HAMSTRAN mesh for the fighter wing

This is a mesh of a propeller, it is the only geometry in this paper that is comprised of 3 separate geometries, as is shown in Figure 37, but the performance of HAMSTRAN is just fine, as can be seen in Figure 38.

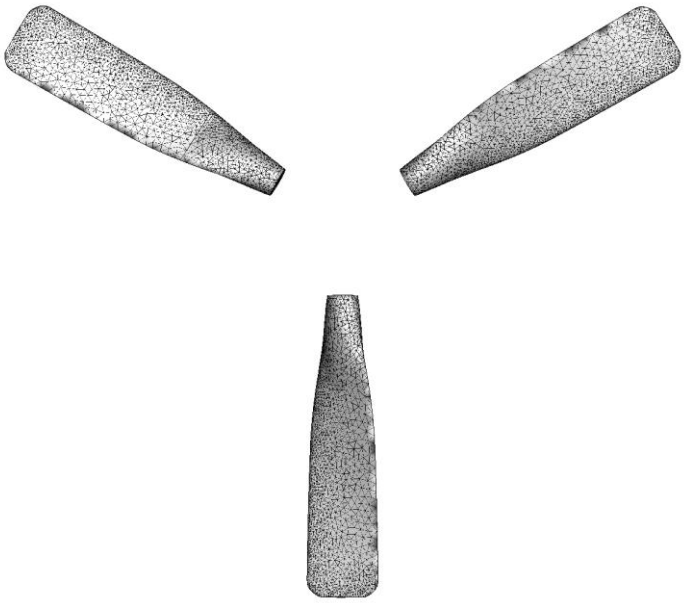


Fig 37. The triangular background mesh of a propeller

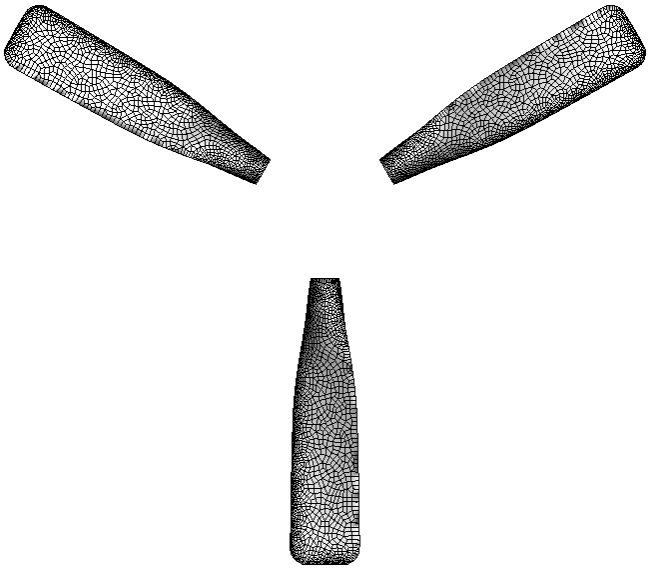


Fig 38. The HAMSTRAN mesh of a propeller

Figure 39 shows the all-triangular surface mesh of the Lockheed blended wing body geometry, an airplane still under development. HAMSTRAN performed just fine despite its complicated geometry, as can be seen in Figure 40.

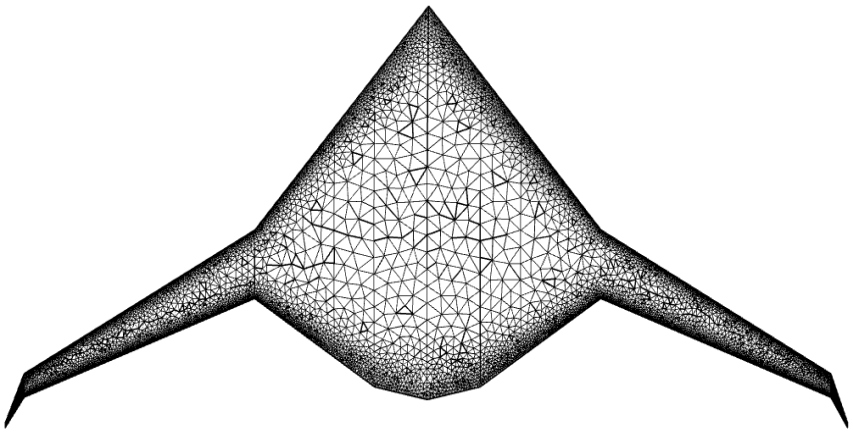


Fig 39. Triangular background mesh of a Lockheed blended wing geometry

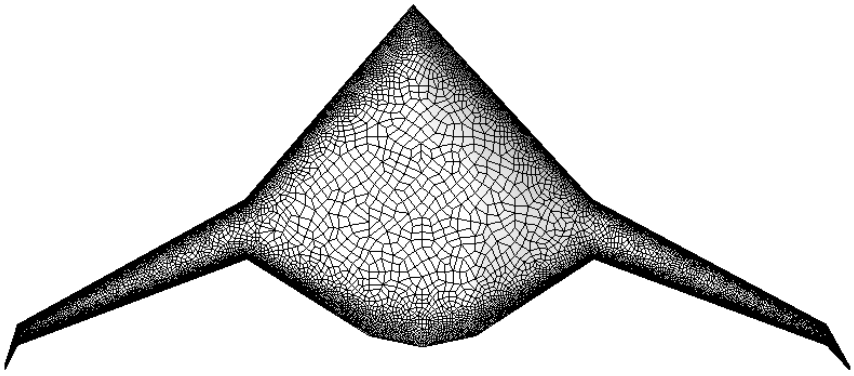


Fig 40. The HAMSTRAN mesh for the Lockheed blended wing mesh

All of the geometries in this section show that HAMSTRAN can work on almost any complicated geometries. However, since flow fields resulting from these meshes will not be discussed in this work, tables highlighting the quality of these meshes are not shown, rather the figures provide a visual overview of the HAMSTRAN process.

Chapter 3: Results and discussion

The HAMSTRAN algorithm for all-quad mesh generation has been validated for various flow conditions on different types of geometries in both 2D and 3D. For 2D, this method is implemented and tested for both the NACA0012 airfoil in turbulent flow and a triangular wedge in inviscid flow. As for 3D cases, the HAMSTRAN algorithm is tested on the OneraM6 wing, the robin fuselage and the lifting rotor. The results are compared with those from simple Catmull-Clark subdivision method and verified by experimental results.

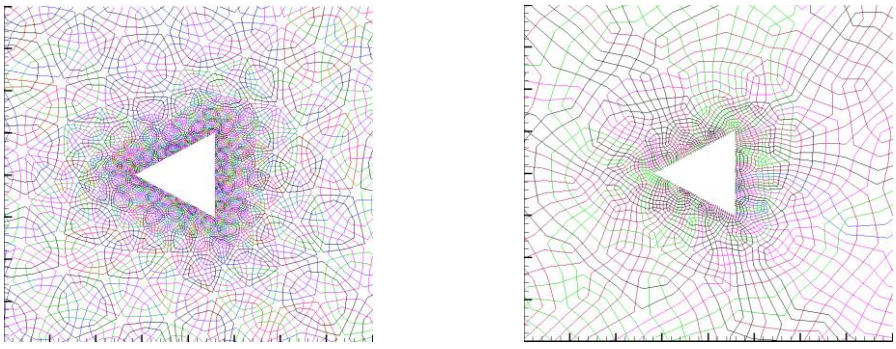
In summary, the first step of this research is generating an unstructured surface mesh, all triangle, quad dominant or hybrid. Then in order to acquire an all-quad mesh, the HAMSTRAN algorithm is implemented. After that, for 3D cases, strand templates extruding from the surface of the mesh are generated as well before creating 3D volume grids. Finally, the volume mesh is implemented into the HAMSTR flow solver to run various simulations. In order to test the efficiency of this new algorithm, several parameters are calculated. They include the distribution of the pressure coefficient on the geometry, the cpu time required to reach convergence and the lift and drag coefficient. Due to the decreased grid number and higher quality Hamiltonian strands, the flow solver should be able to reach convergence in a much shorter time and the accuracy of results should improve. This will be verified in the following sections.

3.1. The triangular wedge in inviscid flow

In this section, this work is going to validate the unsteady method by computing the inviscid flow past a triangular edge. The free stream Mach number is 0.5 and the angle of attack is 0°. The background mesh consists of 2583 triangular grids. Since this is an inviscid case and there is no boundary layer, near body strand grids aren't required. Due to the original mesh being too coarse, this mesh is subdivided again after implementing HAMSTRAN, creating 10,332 grids in total. In order to make a comparison, the original mesh is subdivided twice, resulting in 30,240 grids. This case is computed using 2 different schemes, scheme 1 uses BDF1 in time and 3rd order MUSCL in space, scheme 2 uses BDF2 in time and 3rd order MUSCL in space. Sadly, there is no reference for the inviscid case, but one can judge the accuracy of the simulation through the vortices.

1. The Hamiltonian paths

As can be seen in Figure 41, long and smooth Hamiltonian paths are extracted from the HAMSTRAN mesh, comparing to the short circles extracted from the subdivided mesh. This makes it easier for line-implicit methods to be carried out along these strands.



a. Subdivision b. HAMSTRAN

Fig 41. Hamiltonian loops around the wedge

2. The density contour around the wedge

As can be seen from Figures 42 and 43, the mesh system successfully captured the vortices convecting downstream for both schemes. It showed that the mesh corresponds well with both time discretization methods. The higher order solution from BDF2 maintains the strength of vortices for longer distances comparing to lower order BDF1, as is shown in Figure 42 and 43.

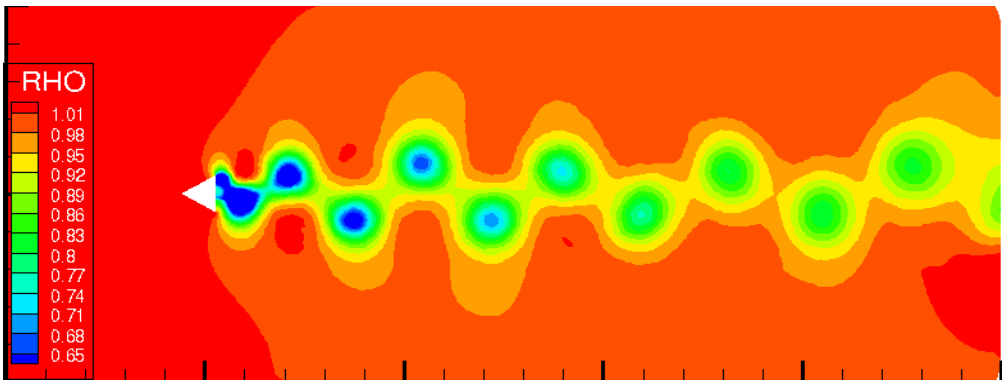


Fig 42. Density contour around the wedge for scheme 1

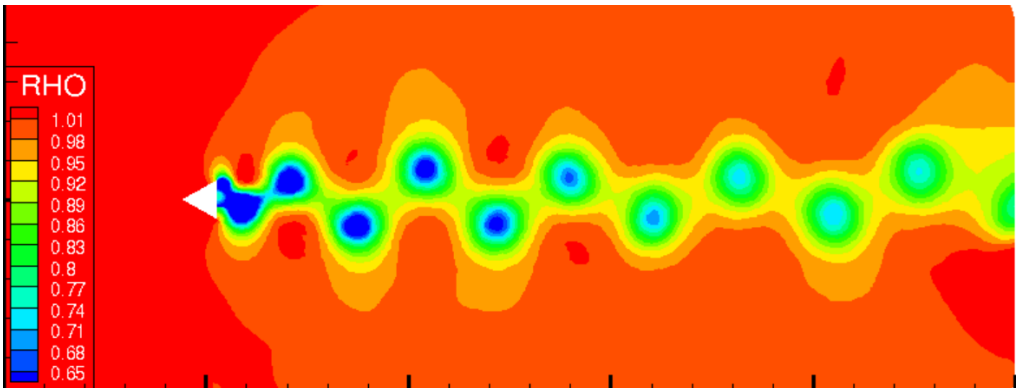


Fig 43. Density contour around the wedge for scheme 2

3. The convergence history

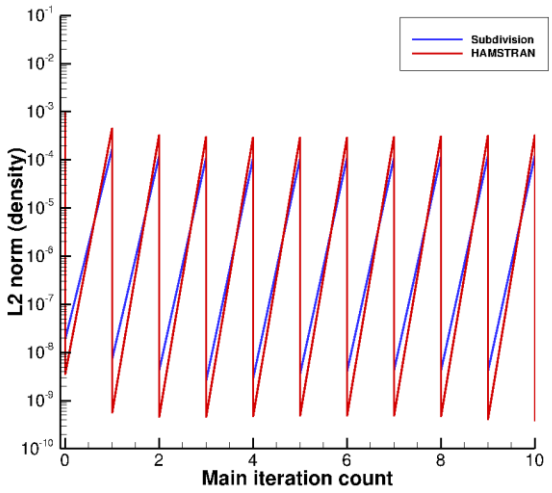


Fig 44. Convergence of density residual for scheme 1

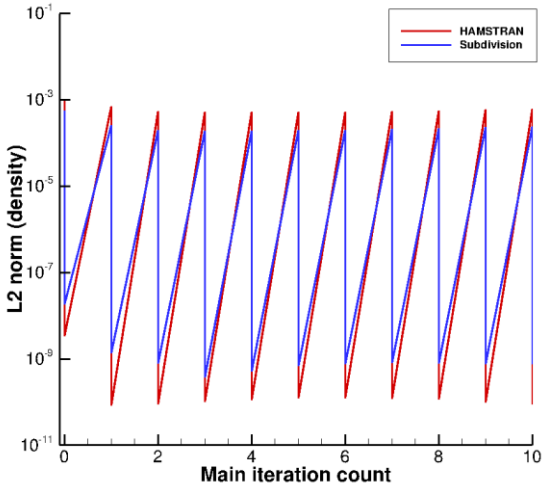


Fig 45. Convergence of density residual for scheme 2

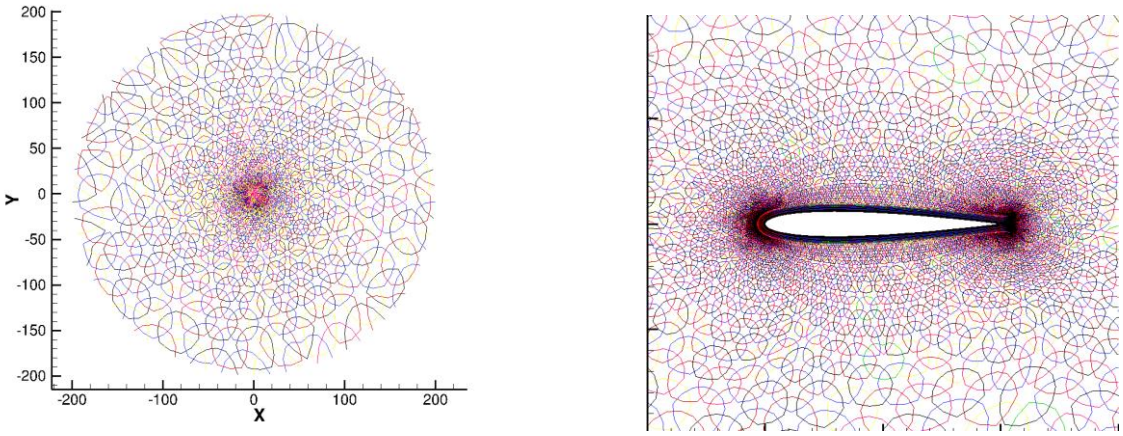
As can be seen from Figures 44 and 45, the density residual converged much more during each main iteration after implementing HAMSTRAN in the flow solver and it is able to successfully track the vortices in the flow. However, in the real world inviscid flow doesn't really exist; so in order to test the fidelity of HAMSTRAN, one needs to use cases of turbulent flow, only in this way can the results be compared with real-life experimental results. One such case is the turbulent flow past a NACA0012 airfoil.

3.2. The NACA0012 airfoil in turbulent flow

In order to test the fidelity of the HAMSTRAN algorithm, turbulent flow past a NACA0012 airfoil is computed at a freestream Mach number of 0.15, Reynolds number of 6 million and three angles of attack (0 ,10 and 15 degrees). The original mesh consists of 10,407 triangular grids and 20,740 quad grids. Due to the original mesh being too coarse, a subdivision algorithm is carried out after implementing the HAMSTRAN algorithm, resulting in 128,394 grids in the mesh. The results are compared with the subdivided mesh of 114,181 grids and some reference data. The reference data include the Gregory data[13] and the results from the OVERTURNS[14] flow solver.

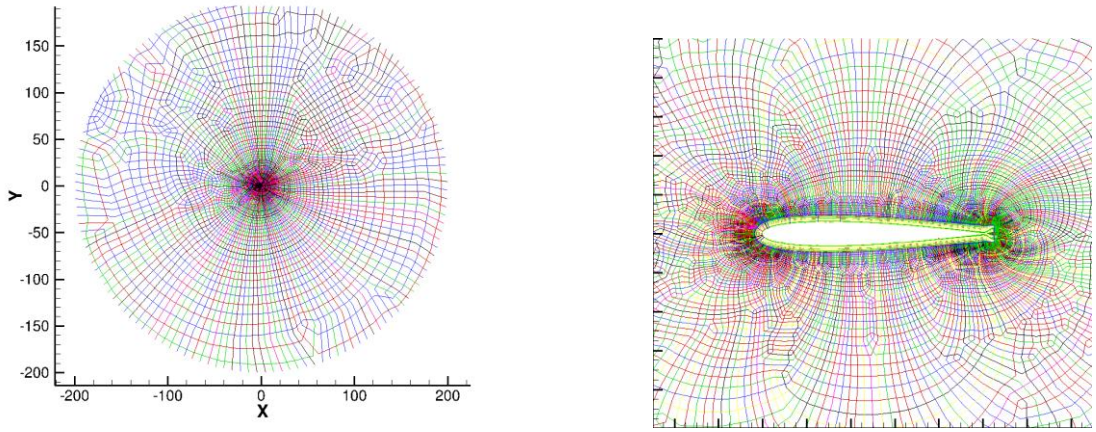
1. The Hamiltonian paths

As can be seen from Figures 46 and 47, the Hamiltonian paths generated from the HAMSTRAN algorithm are generally speaking much longer and smoother, instead of the small circles generated using the traditional subdivision algorithm. Since the stencil-based discretization is easier to carry out on smooth paths, it would increase the accuracy of the results. The following results can be used to verify this.



A. Overview of the entire mesh b. A blow up near the airfoil

Fig 46. The Hamiltonian loops extracted from the subdivided mesh (NACA0012)



A. Overview of the entire mesh b. A blow up near the airfoil

Fig 47. The Hamiltonian loops extracted from HAMSTRAN mesh (NACA0012)

2. The properties of the flow field

Figures 48-53 show the Mach number contours of the flow field and details of the boundary layer, indicating that this mesh is able to successfully capture the boundary layer of the flow.

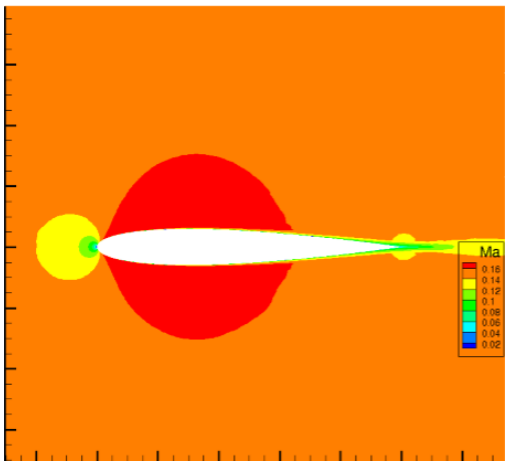


Fig 48. The Mach number contour ($\alpha = 0$)

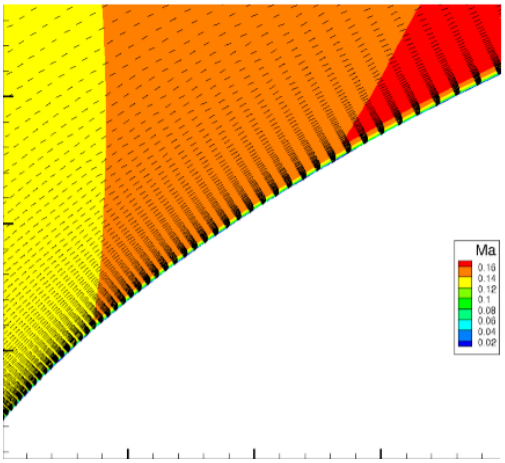


Fig 49. Sketch of the boundary layer ($\alpha = 0$)

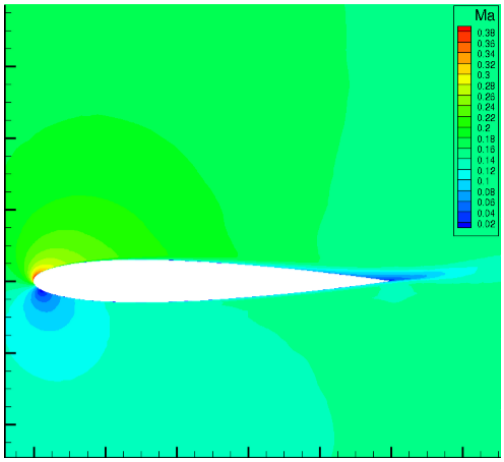


Fig 50. The Mach number contour ($\alpha = 10$)

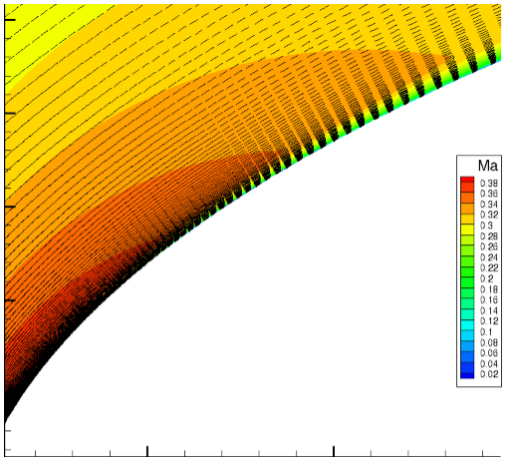


Fig 51. Sketch of the boundary layer ($\alpha = 10$)

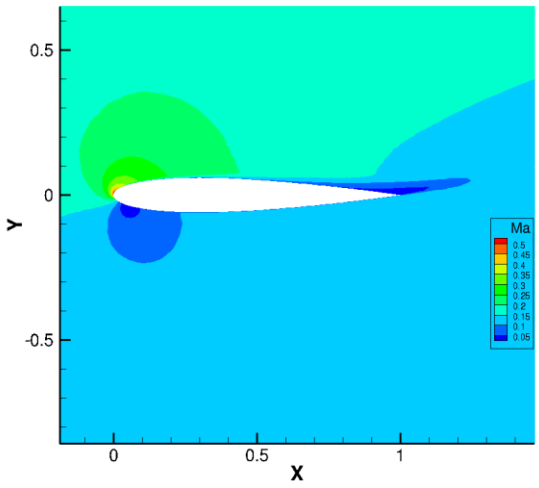


Fig 52. The Mach number contour ($\alpha = 15$)

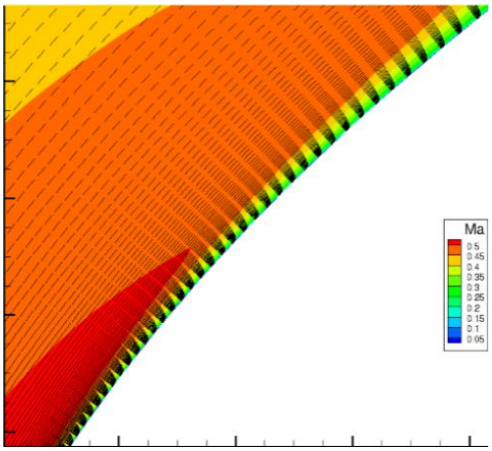


Fig 53. Sketch of the boundary layer ($\alpha = 15$)

3. The distribution of pressure coefficient along the airfoil

Figures 54-56 show the pressure distribution results from using HAMSTRAN as compared with those from using the traditional subdivision algorithm as well as both the Gregory data and the results from the OVERTURNS flow solver. As can be seen from the plots, the results of HAMSTRAN correspond well with the Gregory data and the results from the OVERTURNS flow solver. Although the results from the subdivision algorithm is accurate as well, it has a lot of fluctuations, as can be seen in the following plots. The results from HAMSTRAN are much smoother.

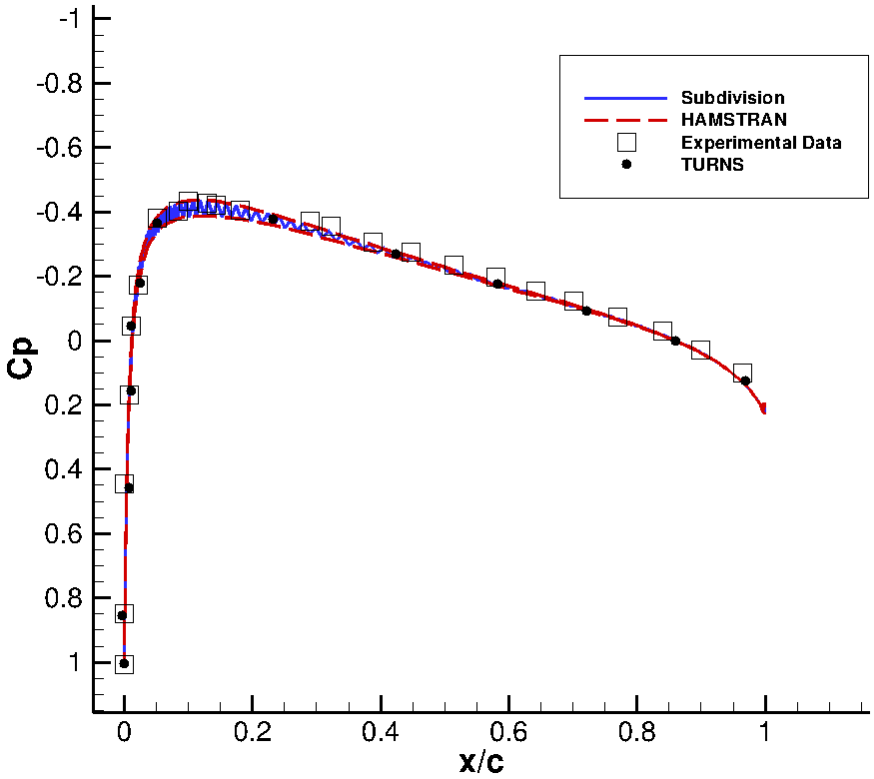


Fig 54. The pressure coefficient over NACA0012 airfoil ($\alpha = 0$)

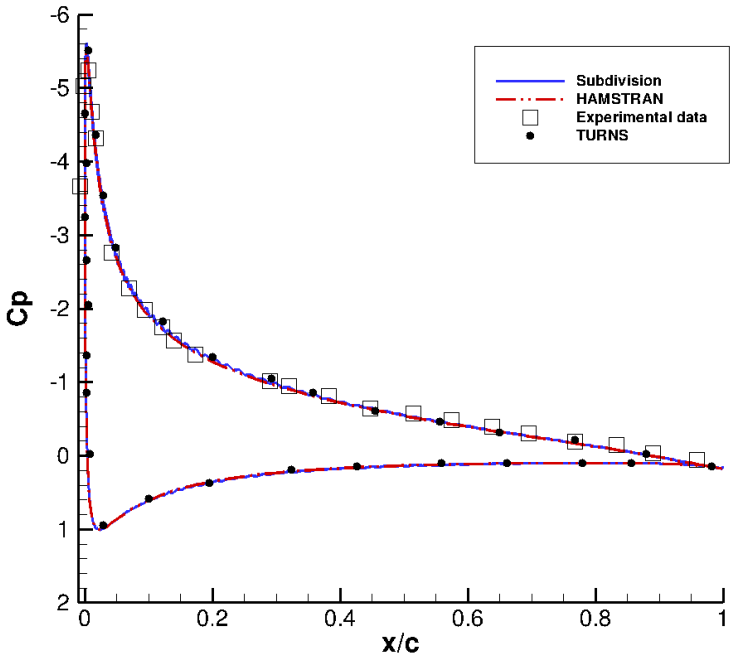


Fig 55. The pressure coefficient over NACA0012 airfoil ($\alpha = 10$)

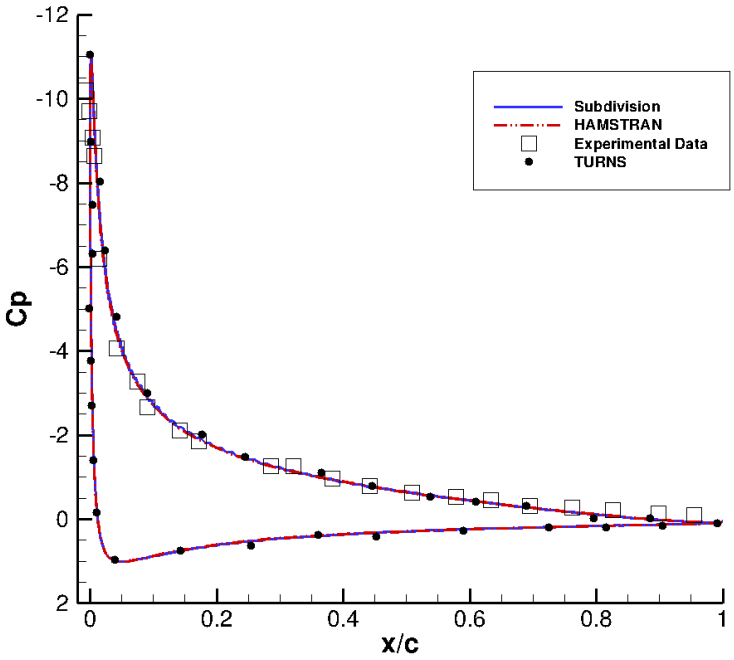
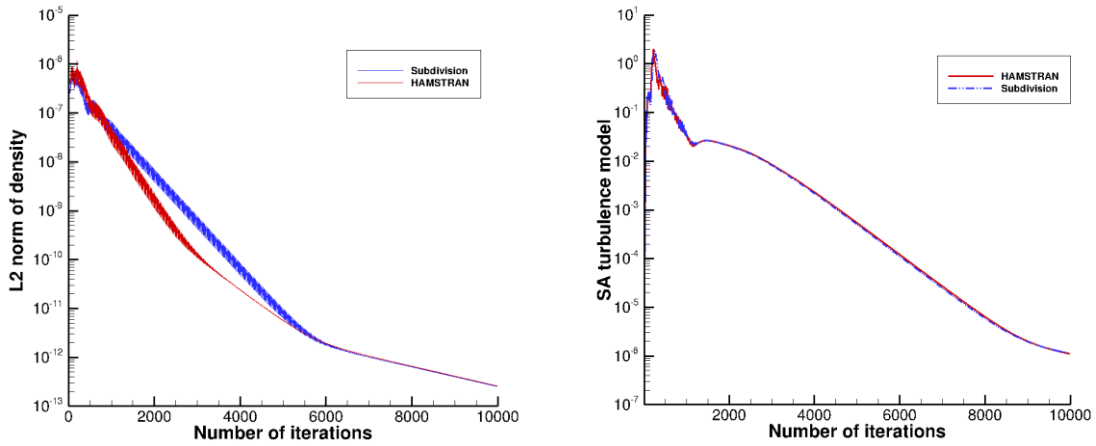


Fig 56. The pressure coefficient over NACA0012 airfoil ($\alpha = 15$)

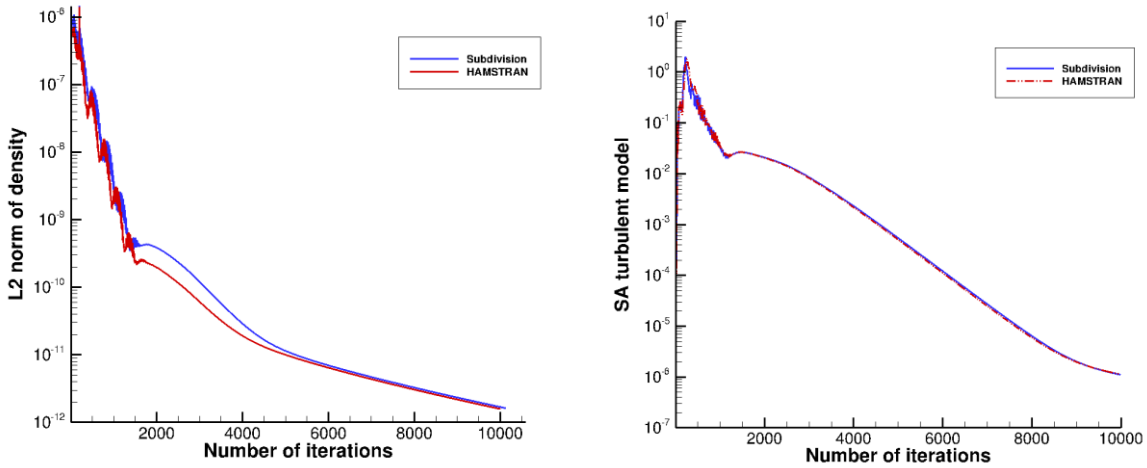
4. The convergence history

As can be seen in Figure 57-59, the residual usually converges a little faster under HAMSTRAN comparing to the mesh created by Catmull-Clark subdivision, although not by a lot.



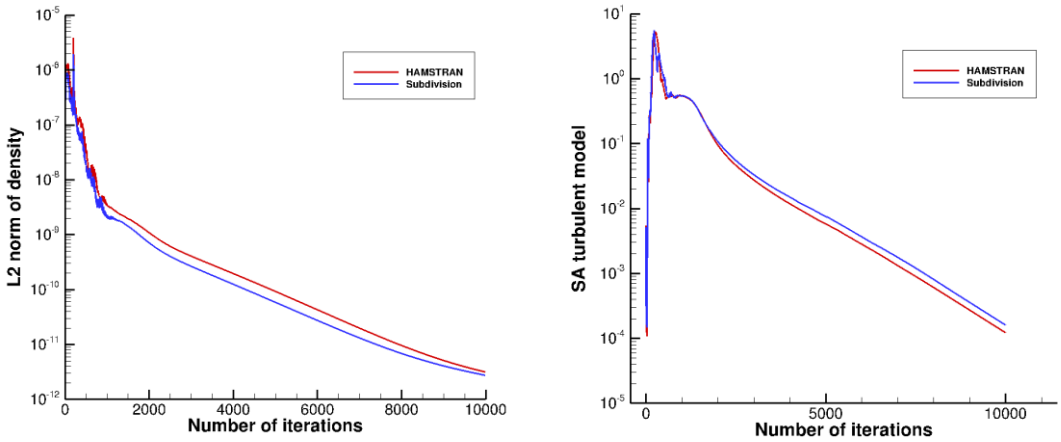
a. The density residual b. The turbulent residual

Fig 57. The convergence history of density and SA turbulence model at $\alpha = 0$



a. The density residual b. The turbulent residual

Fig 58. The convergence history of density and SA turbulence model at $\alpha = 10$

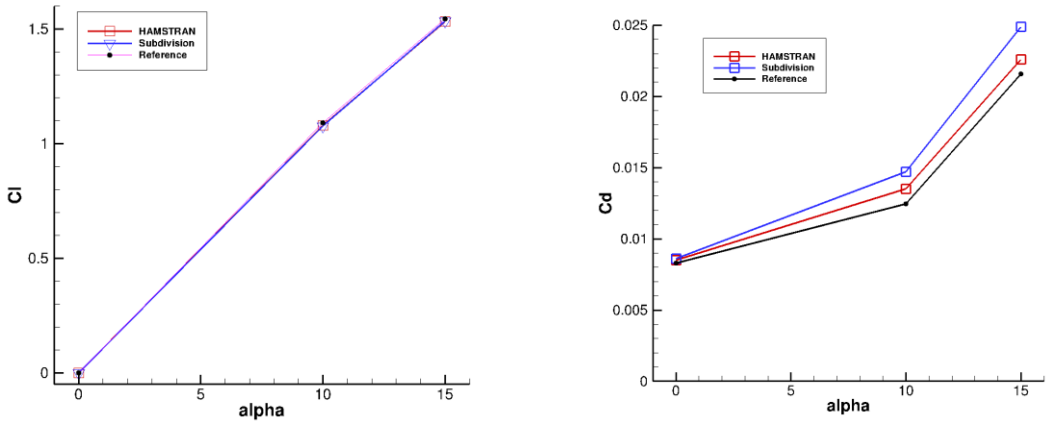


a. The density residual b. The turbulent residual

Fig 59. The convergence history of density and SA turbulence model at $\alpha = 15$

5. The Simulation results

In this section, the lift and drag coefficients are examined as well as the computational time required for both HAMSTRAN and the Catmull-Clark subdivision algorithm. The results are compared with that of OVERTURNS. Under this method, the lift and drag coefficient is 0 and 0.0083 for 0 degrees angle of attack, 1.1000 and 0.0123 for 10 degrees angle of attack, 1.5642 and 0.0214 for 15 degrees angle of attack.



a. The lift coefficient b. The drag coefficient

Fig 60. The lift and drag coefficient for different angles of attack

	Angle of attack	Iteration	Density residual	Turbulence residual	Lift coefficient	Drag coefficient	cpu time
Subdivision	0	10000	2.59×10^{-13}	2.16×10^{-7}	0.0001	0.0086	11597
HAMSTRAN	0	10000	2.53×10^{-13}	2.32×10^{-7}	0.0008	0.0085	13109

Table 10. The simulation results of NACA0012 airfoil at 0 degrees angle of attack

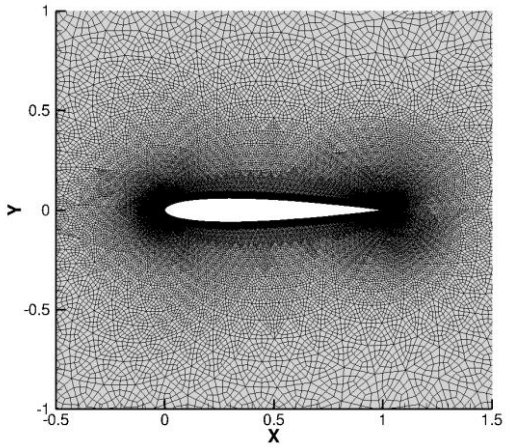
	Angle of attack	Iteration	Density residual	Turbulence residual	Lift coefficient	Drag coefficient	cpu time
Subdivision	10	10000	1.70×10^{-12}	1.13×10^{-6}	1.0770	0.0147	11615
HAMSTRAN	10	10000	1.58×10^{-12}	1.11×10^{-6}	1.0800	0.0135	13259

Table 11. The simulation results of NACA0012 airfoil at 10 degrees angle of attack

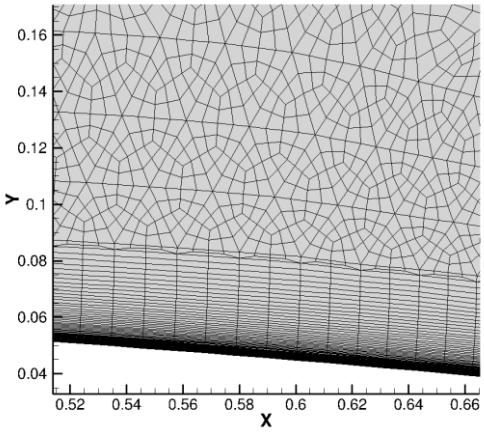
	Angle of attack	Iteration	Density residual	Turbulence residual	Lift coefficient	Drag coefficient	cpu time
Subdivision	15	10000	3.15×10^{-12}	1.23×10^{-4}	1.5346	0.0249	10395
HAMSTRAN	15	10000	2.75×10^{-12}	1.61×10^{-4}	1.5329	0.0226	12399

Table 12. The simulation results of NACA0012 airfoil at 15 degrees angle of attack

As can be seen from Table 10-12, the lift and drag coefficients calculated by HAMSTRAN showed much better agreement with the reference data comparing to that of the subdivision algorithm. However, in order to further illustrate the advantage of HAMSTRAN, one can create a finer mesh using the subdivision process. After the initial subdivision, the unstructured part of the mesh is subdivided again, then a node capturing process is implemented to get rid of the hanging nodes, as shown in Fig 61. The reason for not subdividing the structured mesh twice is because it would make the surface grids overly thin, causing stiffness in the flow solving process.



a. Overview of the mesh around the airfoil



b. A blow up to show the node capturing process

Fig 61. The finer subdivided mesh

The finer subdivided mesh has a total of about 209,690 grids. This extra case was run with an angle of attack of 10 degrees, a Reynolds number of 6 million and a free stream Mach number of 0.15. Although this mesh is finer than the HAMSTRAN mesh, the results are still slightly worse, as can be seen in Table 13.

	Angle of attack	**Iteration**	**Density residual**	**Turbulence residual**	**Lift coefficient**	**Drag coefficient**	**cpu time**
Subdivision	10	10000	1.70×10^{-12}	1.13×10^{-6}	1.0770	0.0147	11615
HAMSTRAN	10	10000	1.58×10^{-12}	1.11×10^{-6}	1.0800	0.0135	13259
The finer subdivded mesh	10	10000	1.28×10^{-12}	1.06×10^{-6}	1.0763	0.0139	28895

Table 13. Results of the finer subdivided mesh at 10 degrees angle of attack

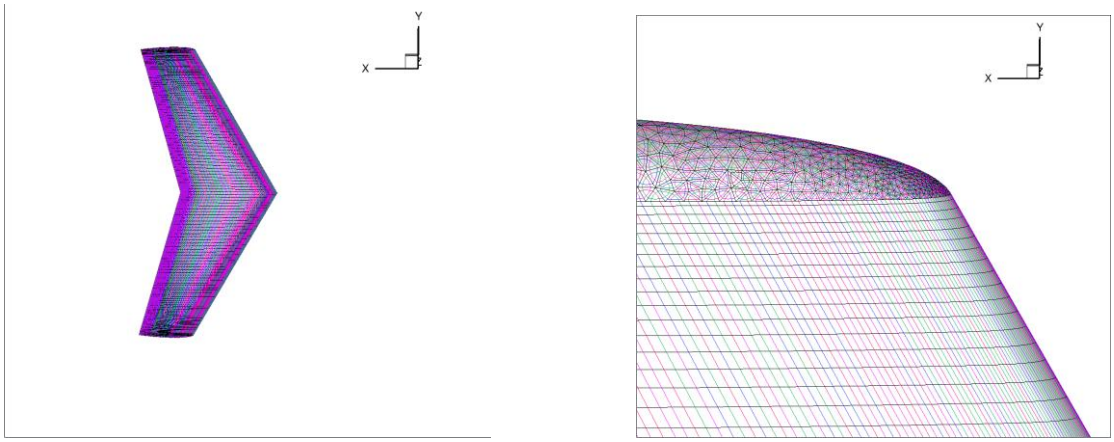
From this table we can see that the drag coefficient became more accurate than the original subdivision process, but it is still not as good as from HAMSTRAN. The lift coefficient became even worse, not to mention it takes twice as long to reach the same level of convergence. These results showed that HAMSTRAN is superior to the Catmull-Clark subdivision algorithm in 2D mesh transformation.

3.3. The OneraM6 wing hybrid mesh case

All the previous cases study 2D meshes, in order to test the full capability of this method, one should implement HANSTRAN in 3D HAMSTR flow solver. The first case is to compute transonic flow past an OneraM6 wing at a Mach number of 0.8395, an angle of attack of 3.06 degrees and a Reynolds number of 11.72 million. The input tessellation is a structure-unstructured hybrid mesh, with structured grids filling the main body and only the tip of the wing is comprised of unstructured triangular grids. After implementing HAMSTRAN, there are 20,380 grids on the surface. Then starting from an initial wall spacing of 1×10^{-5} chord length, 67 layers of strands extrude from the surface of the geometry, creating a volume mesh consisting of approximately 1.4 million grids. The details of the mesh can be seen in Figure 62. To make a comparison, a subdivided mesh consisting of about 4 million grids is also created. Of course, the results are also compared with that of the $WIND$[15] solver and experimental data from *Experimental data base for computer program assessment*[16].

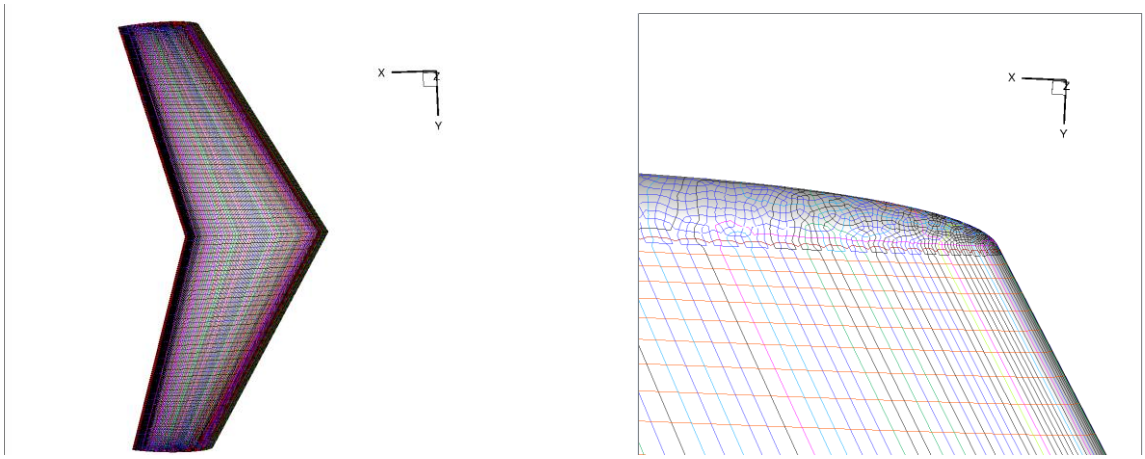
1. The Hamiltonian loops on the surface of the OneraM6 wing

Figures 62 and 63 visualize the Hamiltonian paths. Since 3D Hamiltonian loops are too messy to be seen, one might want to look at only the loops on the surface of the geometry. As can be seen from the plots, the Hamiltonian loops are about the same on the main body of the wing except there are fewer loops for the HAMSTRAN mesh. On the tip of the wing, the strands extracted from the HAMSTRAN mesh are much longer and smoother in general.



a. An overview of the entire mesh b. A blow up at the tip of the wing

Fig 62. The Hamiltonian loops extracted from the subdivided mesh (OneraM6 hybrid)

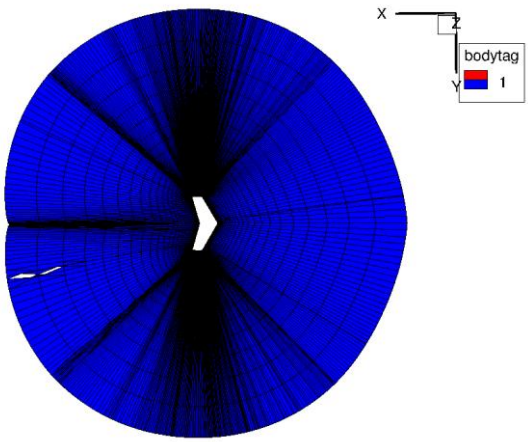


a. An overview of the entire mesh b. A blow up at the tip of the wing

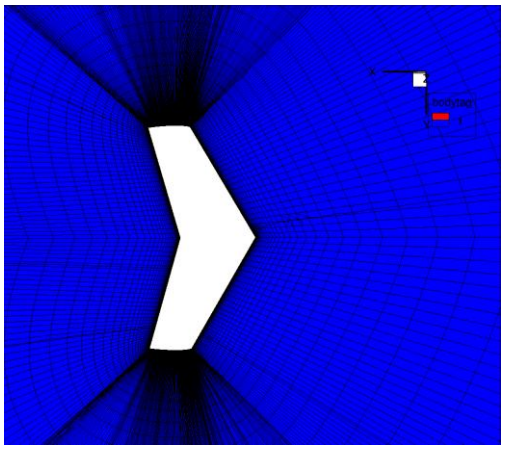
Fig 63. The Hamiltonian loops extracted from the HAMSTRAN mesh (OneraM6 hybrid)

2. The strand grids around the wing

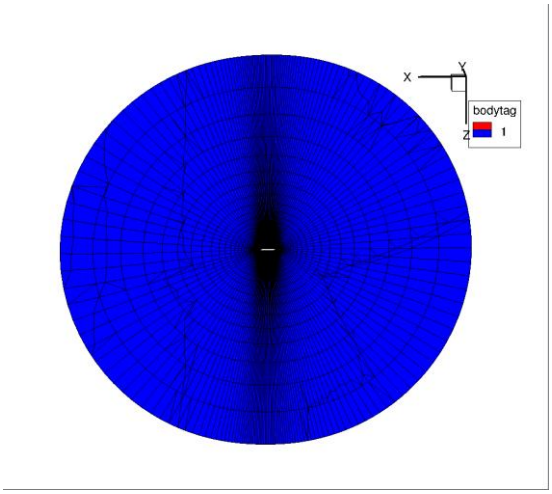
As is mentioned in the first chapter, the Hamiltonian paths only provide structure in the surface direction, in order to find structure in the wall normal direction, one needs to utilize strand grids. The strands cannot cross each other nor can there be any sudden, abrupt changes in the volume of adjacent grids. The details of the strand grids can be seen in Figure 64.



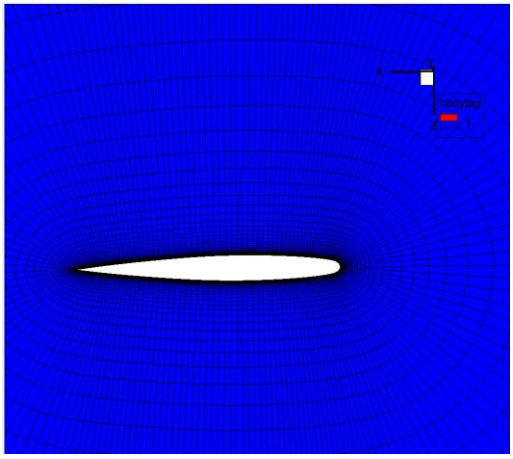
a. The strand grids in the main plane



b. A Blow up near the wing



c. The strand grids at the wing tip

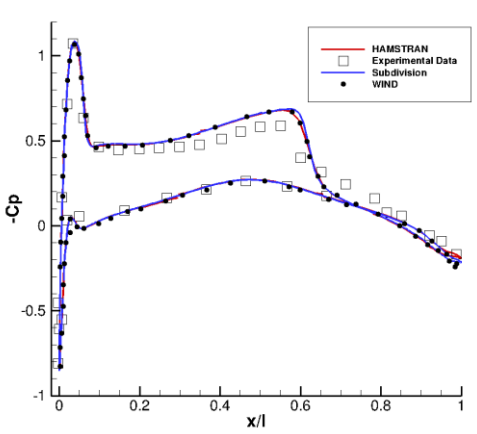


d. A Blow up around the tip of the wing

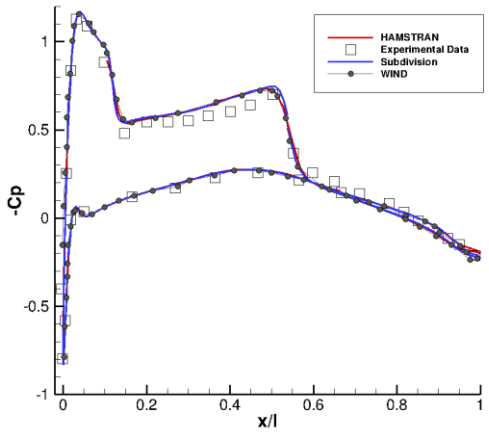
Fig 64. An overview of the strand grids around the OneraM6 wing

3. The pressure distribution at different areas of the surface

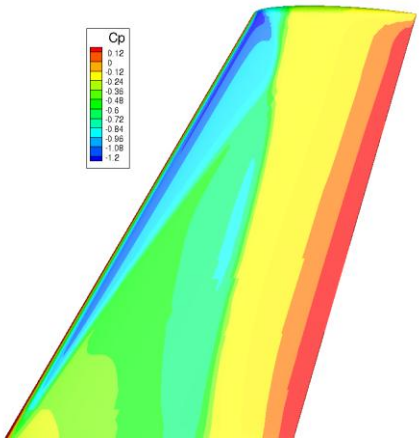
As can be seen from Figure 65, the pressure distribution shows good agreement with both experimental data and the WIND solver. In Figure 65 a, b, there are clear pressure jumps on the plot, indicating a shock. This mesh successfully captured the local shocks. This shows that HAMSTRAN not only works well in 2D, but also works well combined with 3D strand grids.

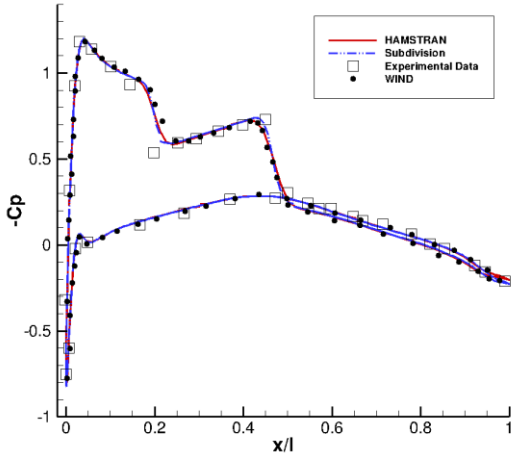


a. y/l=0.20

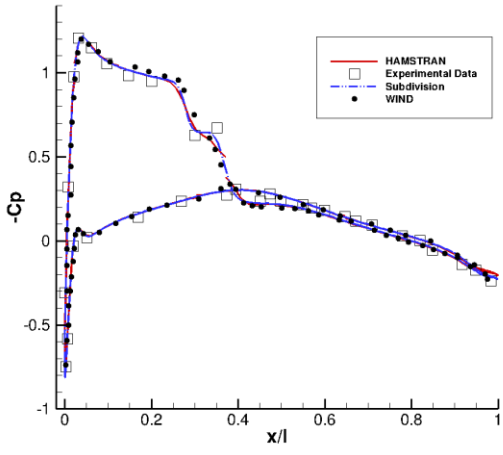


b. y/l=0.44





c. y/l=0.65



d. y/l=0.80

Fig 65. The pressure distribution on different areas of the wing (OneraM6 hybrid)

4. The simulation results

As can be seen from plot 66, the density residual converged faster after adding the HAMSTRAN algorithm to the flow solver. Due to having less number of grids, the time required for each time step also decreased. As can be seen in Table 14, it only requires 16216 cpu seconds to complete the calculation, about 1/3rd of the time it used to take. The best thing is that the accuracy of results didn't change due to having less number of grids, as can be seen in the time history of lift and drag coefficients in plots 67 and 68, HAMSTRAN converged a little faster but eventually they both reached the same results.

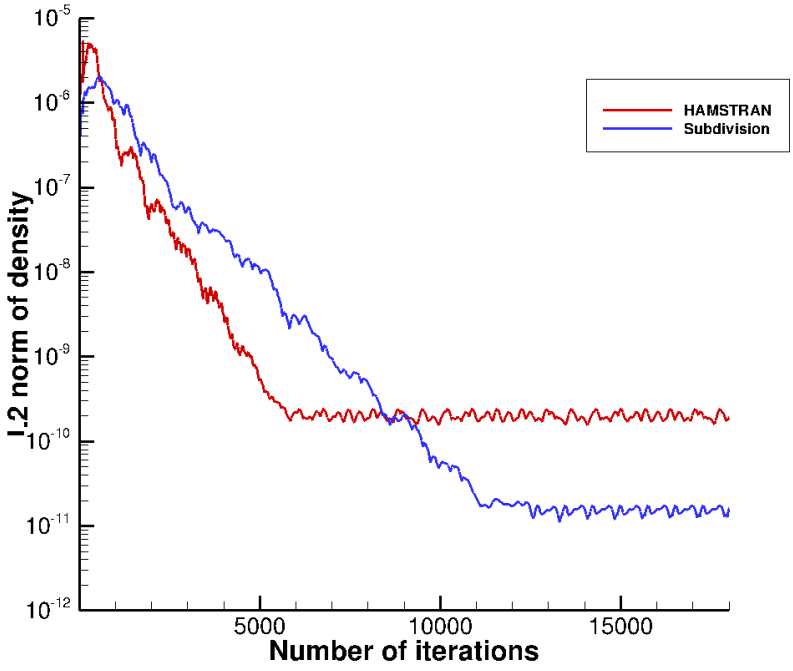


Fig 66. The convergence history of density residual (OneraM6 hybrid)

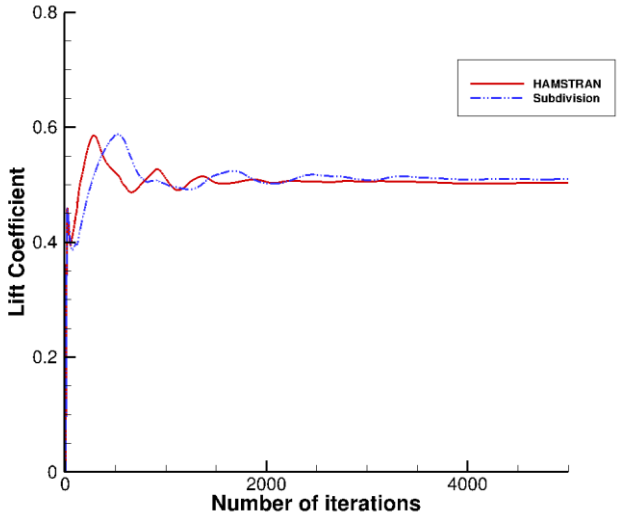


Fig 67. The convergence history of lift coefficient (OneraM6 hybrid)

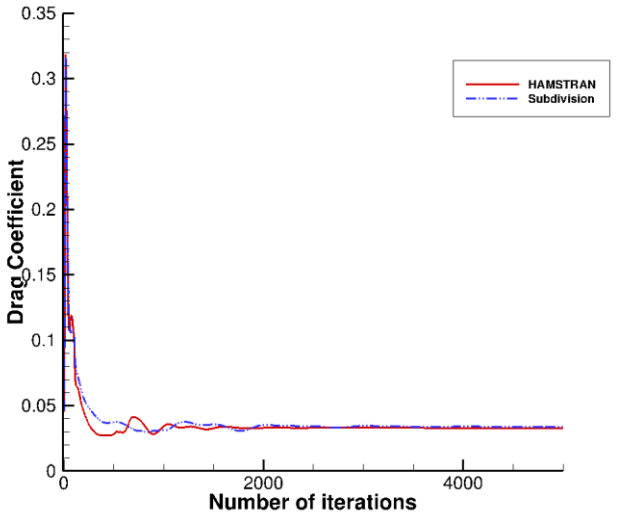


Fig 68. The convergence history of drag coefficient (OneraM6 hybrid)

	Number of iterations	**Lift coefficient**	**Drag coefficient**	**cputime**
subdivision method	18000	0.5098	0.0339	52958
HAMSTRAN	18000	0.5037	0.0330	16216

Table 14. Simulation results of OneraM6 wing hybrid mesh case

3.4. Low Mach number flow around a robin fuselage

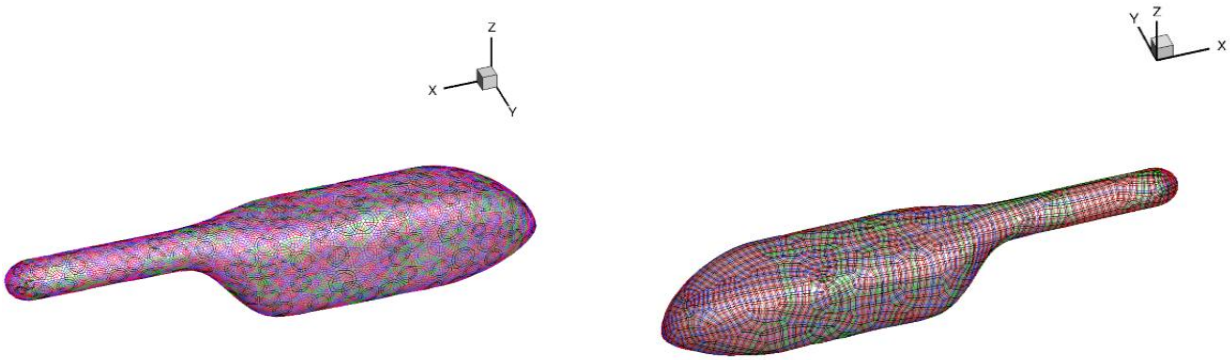
This case computes a low Mach number flow past a robin fuselage at 0 degrees angle of attack. The original surface mesh consists of 2096 triangles, making it a coarse mesh. In order to improve the accuracy of results, another subdivision process is carried out after implementing HAMSTRAN. In the end, the surface mesh consists of 8720 grids. In order to test the performance of the mesh in various flow conditions, laminar flow, inviscid flow and turbulent flow are simulated in this case.

The mesh for inviscid and laminar flow has an initial wall spacing of 0.1% of the fuselage length. 54 layers of strands extrude from the surface of the robin fuselage, giving the 3D volume mesh 0.5 million grids. For the laminar case, the Reynolds number is 5000 and the Mach number is 0.3. Due to the complexity of the boundary layer in a turbulent mesh, finer grids are required. So the initial wall spacing is 1×10^{-5} of the fuselage length, 75 layers of strands extrude from the surface, creating about 0.6 million grids. The Reynolds number for the turbulent case is 1.6 million with a Mach number of 0.1. The results are compared with experimental data from *Fundamental aeronautics* $program$[12] and $OVERFLOW$[12] results.

1. The Hamiltonian paths on the surface of the robin fuselage

As one can see in Figure 69, long and smooth Hamiltonian loops are extracted from the HAMSTRAN mesh, unlike the subdivided mesh where the loops are small circles. This makes it

much easier for line-implicit methods to be implemented on these paths, increasing the accuracy and convergence of the results.

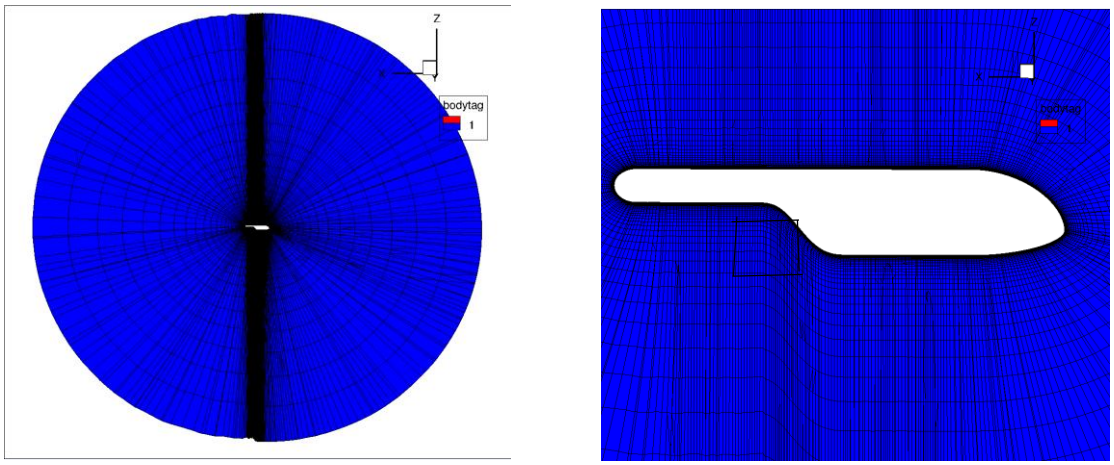


a. The subdivided mesh b. The HAMSTRAN mesh

Fig 69. The Hamiltonian loops extracted on the surface of the robin fuselage

2. The strand grids around the robin fuselage

Different from the previous geometry, this one has a large concave region, as can be seen in Figure 70b. In order to make sure the strand grids don't overlap each other, curved strands are utilized in this case, the details can be seen in Figure 70. In this way, the strands can still represent the wall-normal direction in the near-body region without crossing each other in the far field, the details of the curved strands are shown in Figure 71.



a. The overview of the entire mesh b. A blow-up around the robin fuselage

Fig 70. The strand grids around the robin fuselage

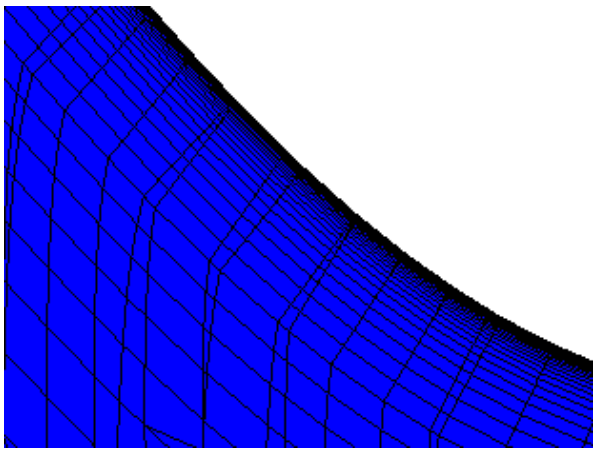


Fig 71. A blow-up to show the curved strands

3. The flow field around the robin fuselage

Figure 72 shows the variation of Mach number over the surface of the fuselage and in the longitudinal plane for inviscid flows. Stagnation points were observed at the nose of the fuselage just as expected. The flow was also noted to accelerate and decelerate over the concave section of the fuselage. In Fig 73, the flow solution captures the features of the turbulent flow qualitatively.

There is a substantial amount of turbulence in the flow past the concave section on the underside of the fuselage, which is carried over to the tail boom region as well. The flow separation and a recirculation zone were observed past the concave region of the fuselage for turbulent flow. Generally speaking, this mesh system is able to capture all the critical features in the flow.

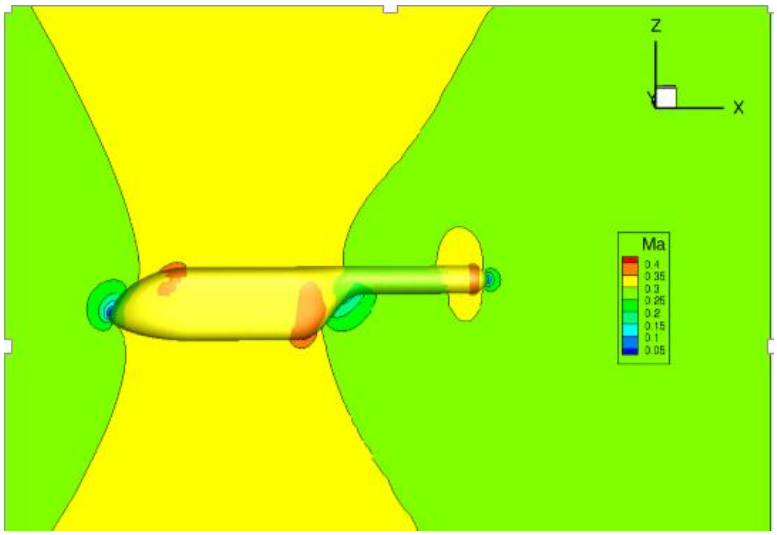


Fig 72. The Mach number contour of inviscid flow around the robin fuselage

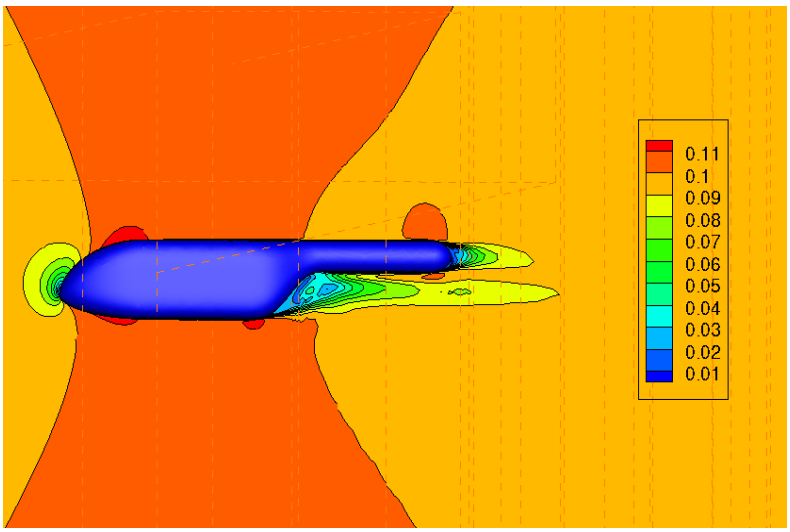
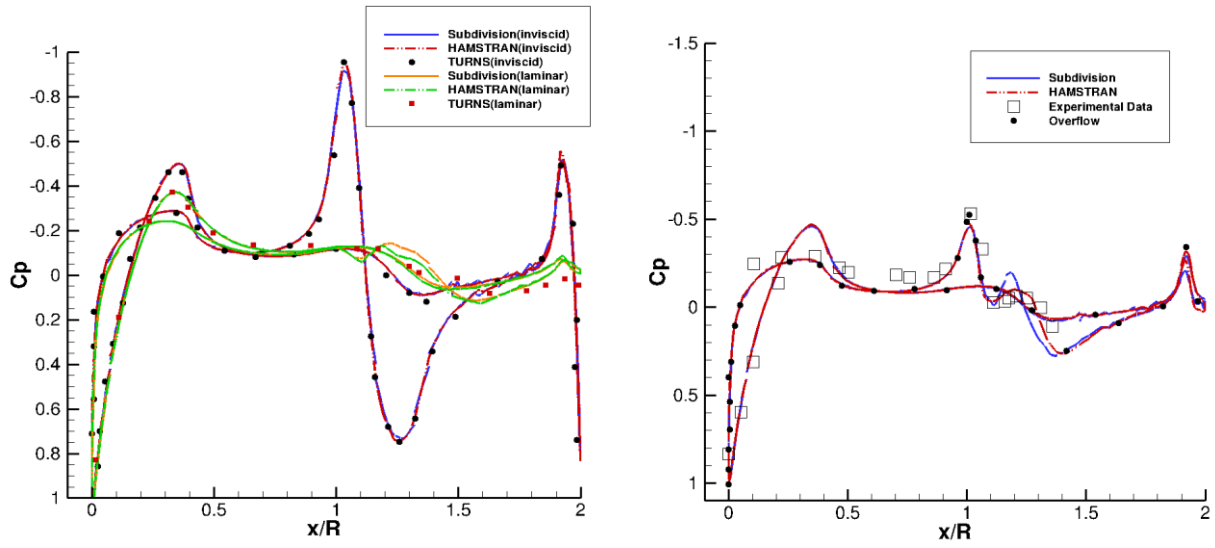


Fig 73. The Mach number contour of turbulent flow around the robin fuselage

4. The pressure distribution along the centerline of the robin fuselage

In this section, the turbulent pressure distribution is compared with the results from the $OVERFLOW$[12] solver, experimental results from NASA and the results from subdivision algorithm. Sadly, there is no experimental data for laminar and inviscid flow, so the results are only compared with the results from OVERTURNS solver. The subdivision algorithm creates 6288 grids on the surface of the geometry. For the laminar and inviscid case, 54 layers of strands extrude from the surface of the geometry, creating about 330,000 grids in the 3D volume. For the turbulent case, 75 layers of strands extrude from the surface of the geometry, creating about 470, 000 grid cells. Since the HAMSTRAN mesh is also subdivided once, both meshes have approximately same number of grid cells. As one can see in Figure 74, in the laminar and inviscid case, the results from both HAMSTRAN and subdivision showed good agreement with the results from OVERTURNS. However, in the turbulent case, results from HAMSTRAN showed much better agreement with experimental data.



a. Laminar and inviscid flow

b. Turbulent flow

Figure 74. Pressure coefficient along the centerline of the robin fuselage

As is known, inviscid and laminar flow doesn't exist in the real world. So HAMSTRAN has much better fidelity comparing to the subdivision algorithm. In order to further illustrate the superiority of HAMSTRAN, this work decided to subdivide the original mesh twice, creating a finer subdivided mesh. As can be seen from Figure 75, the results from the finer subdivided mesh shows much better agreement with the reference data, but still slightly worse than HAMSTRAN.

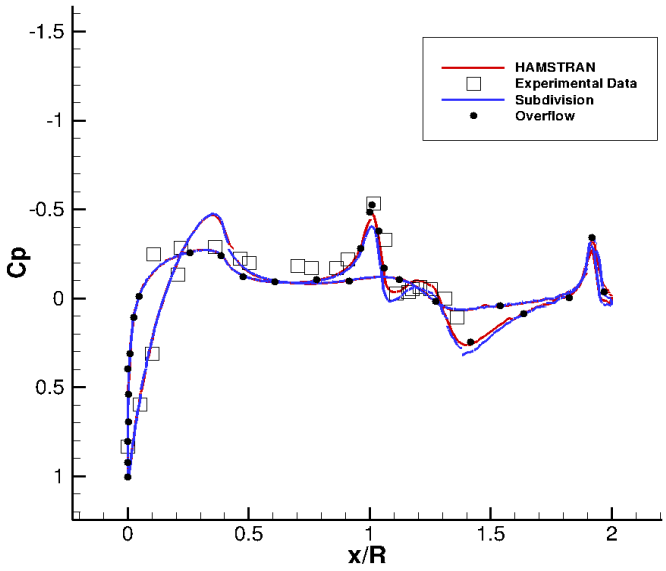


Fig 75. The turbulent pressure coefficient along the centerline of the robin fuselage

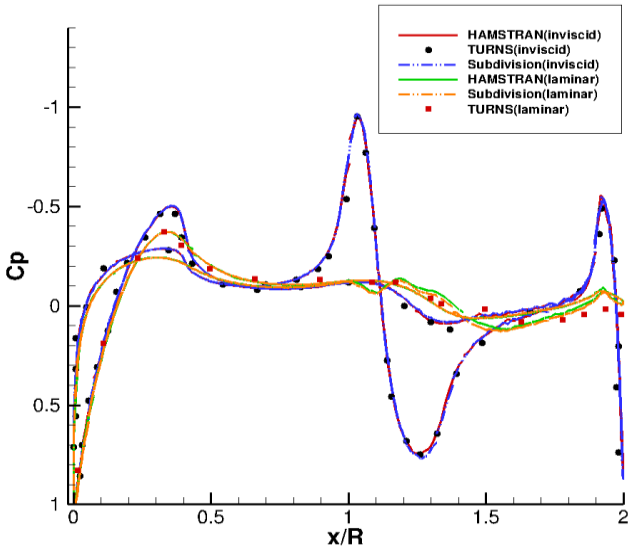


Fig 76. The laminar and inviscid pressure coefficient along the centerline of the robin fuselage

5. The simulation results

As can be seen from Fig 77, after implementing HAMSTRAN in the HAMSTR flow solver, the density residual converged faster, especially in laminar and inviscid flow. Figure 77 shows the centerline boundary layer profile obtained on the underside of the fuselage just before the concave region when x =0.9. The experiments were performed using a *particle image velocimetry* (PIV) [17]. As can be seen in the plot, the simulation results match well with the experimental data. The subdivision in this part refers to the finer subdivided mesh.

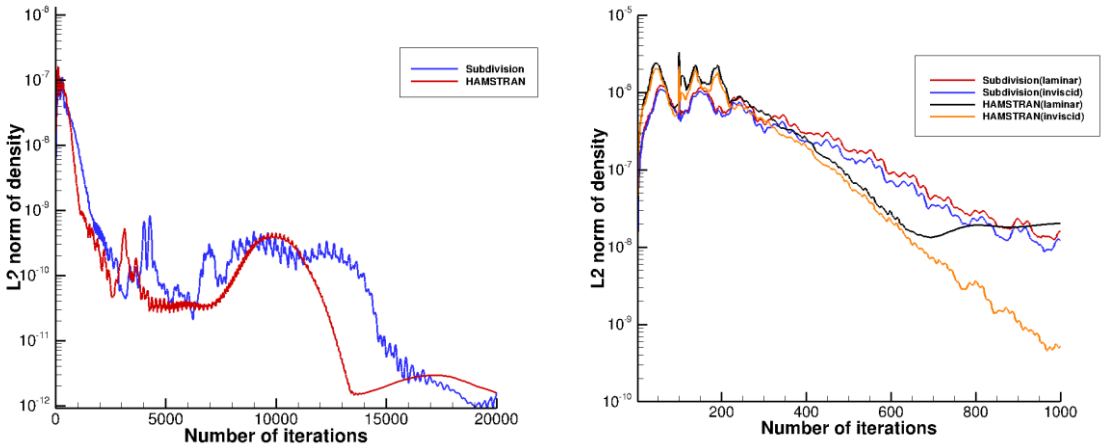


Fig 77. The convergence history of density (Left: Turbulent, Right: laminar and inviscid)

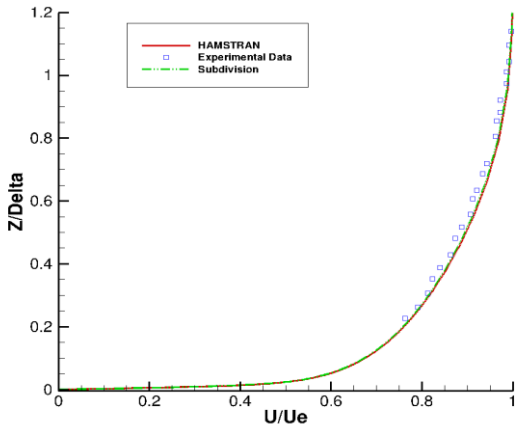


Fig 78. Velocity profile at the centerline boundary layer at x=0.9 (turbulent)

	Flow condition	Number of iterations	cputime
subdivision	Turbulent	20000	55056
HAMSTRAN	Turbulent	20000	16667
Subdivision	inviscid	1500	2085
HAMSTRAN	inviscid	1500	622
Subdivision	laminar	1500	2245
HAMSTRAN	laminar	1500	694

Table 15. Simulation results of the robin fuselage case

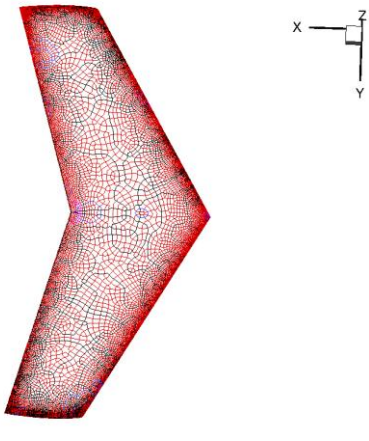
As can be seen from Table 15, HAMSTRAN is able to reach the same level of accuracy using only roughly 1/3rd of the time required for the subdivided mesh. This case tells us that HAMSTRAN works well not only with normal strand grids, but also with curved strand grids.

3.5. The OneraM6 wing overset mesh case

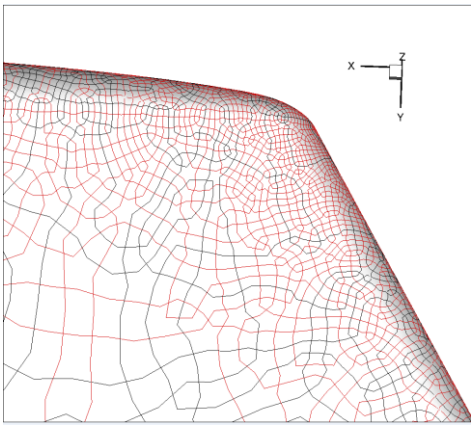
All the above cases only utilize the strand grids. In order to test the combination of HAMSTRAN and overset grid, this case reruns the OneraM6 wing case, but with overset mesh. This time the original input tessellation is an all-triangular mesh on the entire surface of the wing made of about 44,000 grids. After implementing HAMSTRAN, there are about 50,000 quad grids on the wing surface. First, 54 layers of strand extrude from the surface of the geometry, creating around 2.7 million grids in the near body strand mesh. The background Cartesian mesh consists of around 2.4 million grids. The details of the overset mesh can be seen in Fig 80 and 81.

1. Hamiltonian paths at the tip of the wing

As can be seen in Figures 78 and 79, long and smooth Hamiltonian loops are extracted from the HAMSTRAN mesh, unlike the subdivided mesh where the loops are small circles. This makes it much easier for line-implicit methods to be implemented on these paths, increasing the accuracy of the result.

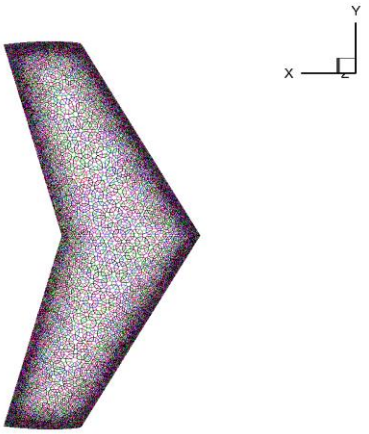


a. An overview of the entire wing

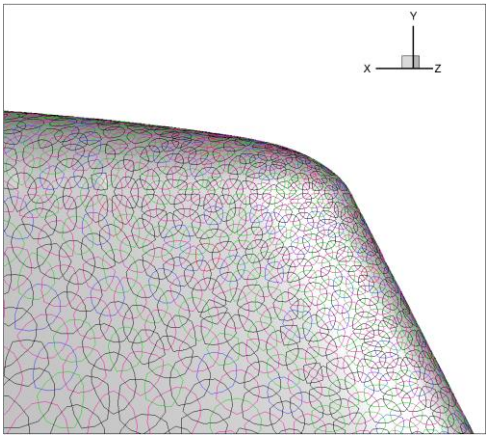


b. A blow up near the tip of the wing

Fig 79. The Hamiltonian loops extracted from the HAMSTRAN mesh (OneraM6 overset)



a. An overview of the entire wing

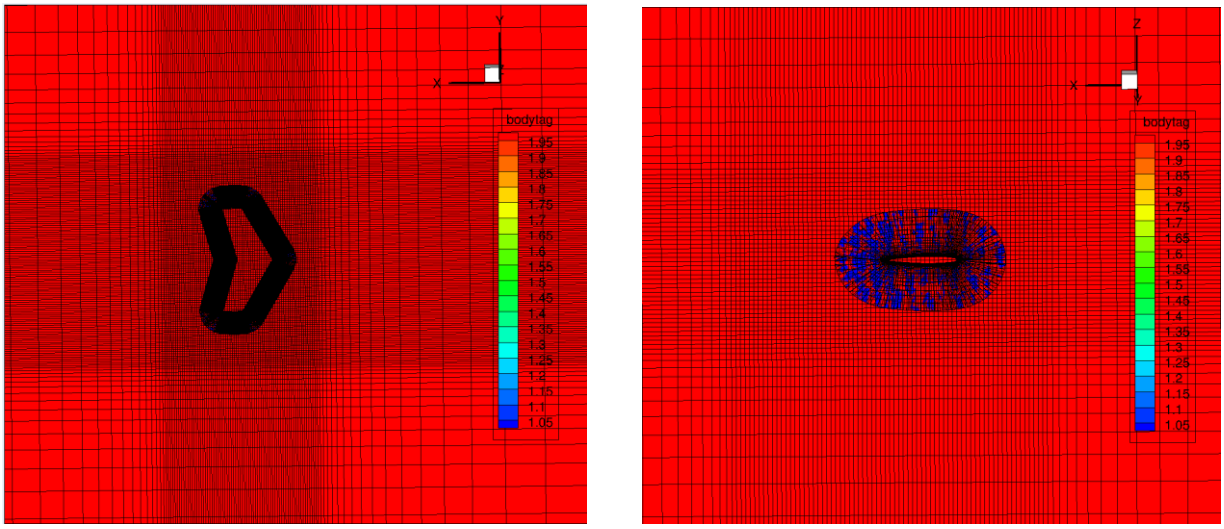


b. A blow up near the tip of the wing

Fig 80. The Hamiltonian loops extracted from the subdivided mesh (OneraM6 overset)

2. The overset grids around the wing

As can be seen in Figure 80, the background Cartesian mesh system is made of a small box and a big box. In the small box, all the grids are the same size, with a side length of 0.05 chord length. Then the mesh is stretched in all 3 directions with a stretching ratio of 1.2. The small box should be able to cover the entire strand grid system, as is shown in Fig 81.



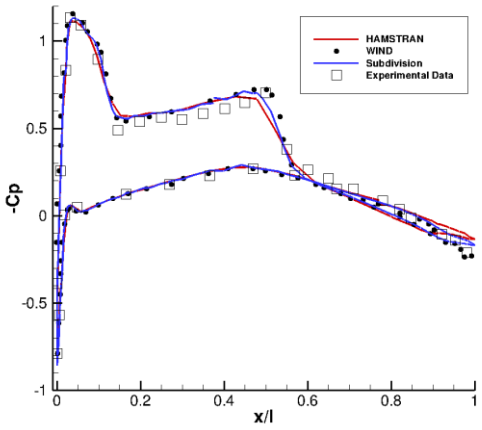
a. The mesh around the wing body

b. The mesh around the tip of the wing

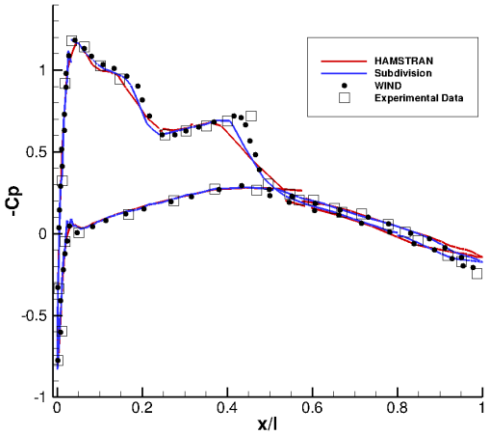
Fig 81. A cutaway view of the overset mesh around the OneraM6 wing

3. The pressure distribution at different areas of the airfoil

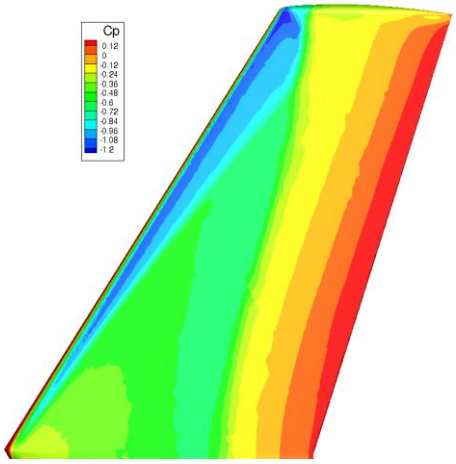
As can be seen in Figure 82, the pressure coefficient distribution calculated by both HAMSTRAN and the subdivision mesh match well with the experimental results, but not as good as the case using the hybrid mesh, this is probably due to the lack of structure in the main body, but overall speaking, HAMSTRAN and overset grids is a good combination.

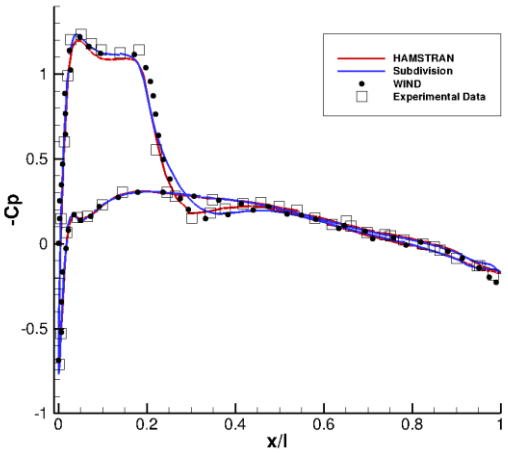


a. y/l=0.44

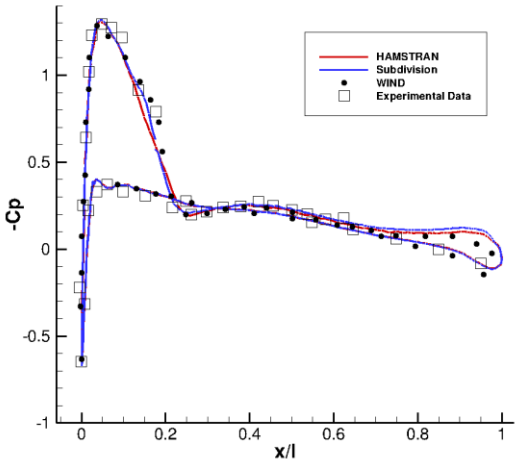


b. y/l=0.65





c. y/l=0.95



d. y/l=0.99

Fig 82. The pressure distribution on different areas of the wing (OneraM6 overset)

4. The simulation results

As can be seen from Figure 83, the results from the HAMSTRAN mesh converged a little faster comparing to the subdivided mesh. However, due to having fewer number of grids, it takes a much shorter time to reach the same level of convergence, as is shown in Table 16. Figures 84 and 85 show that the convergence of the lift and drag coefficient time histories are similar in terms of iterations.

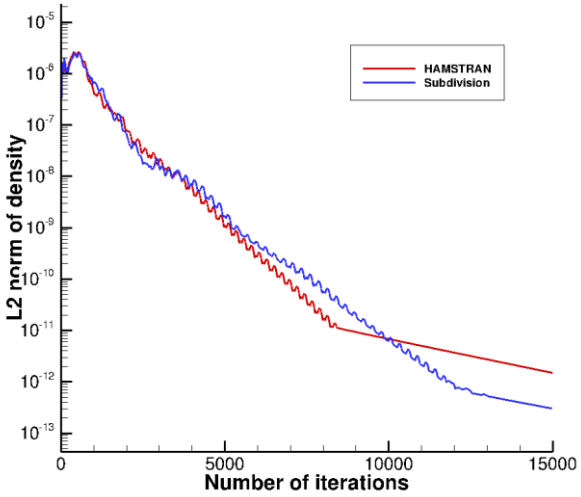


Fig 83. The convergence history of density residual (OneraM6 overset)

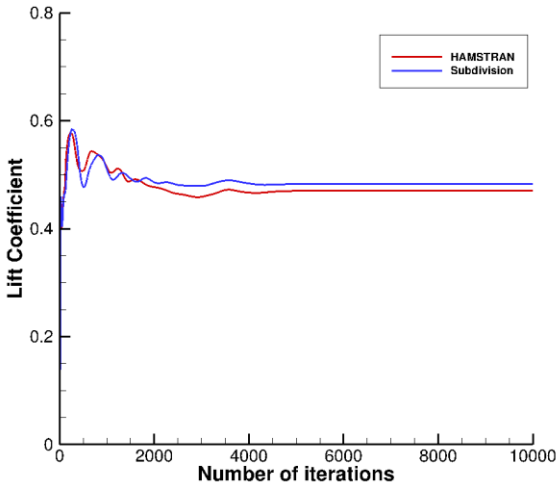


Fig 84. The convergence history of lift coefficient (OneraM6 overset)

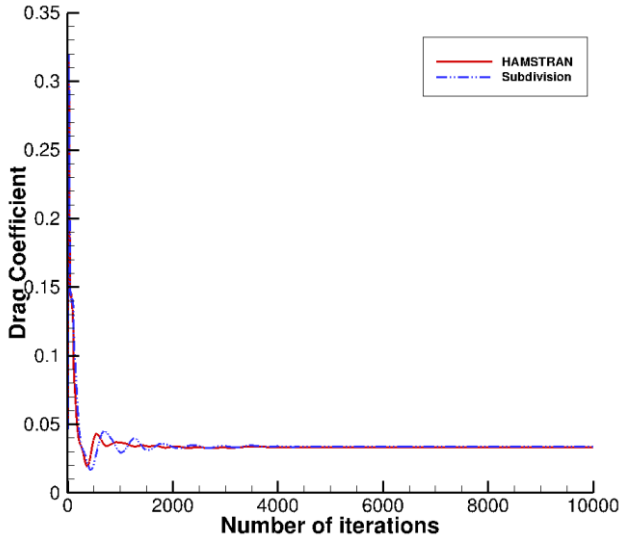


Fig 85. The convergence history of drag coefficient (OneraM6 overset)

	Number of iterations	**Lift coefficient**	**Drag coefficient**	**cputime**
subdivision method	15000	0.4830	0.0338	87035
HAMSTRAN	15000	0.4707	0.0335	36483

Table 16. Simulation results for OneraM6 wing overset case

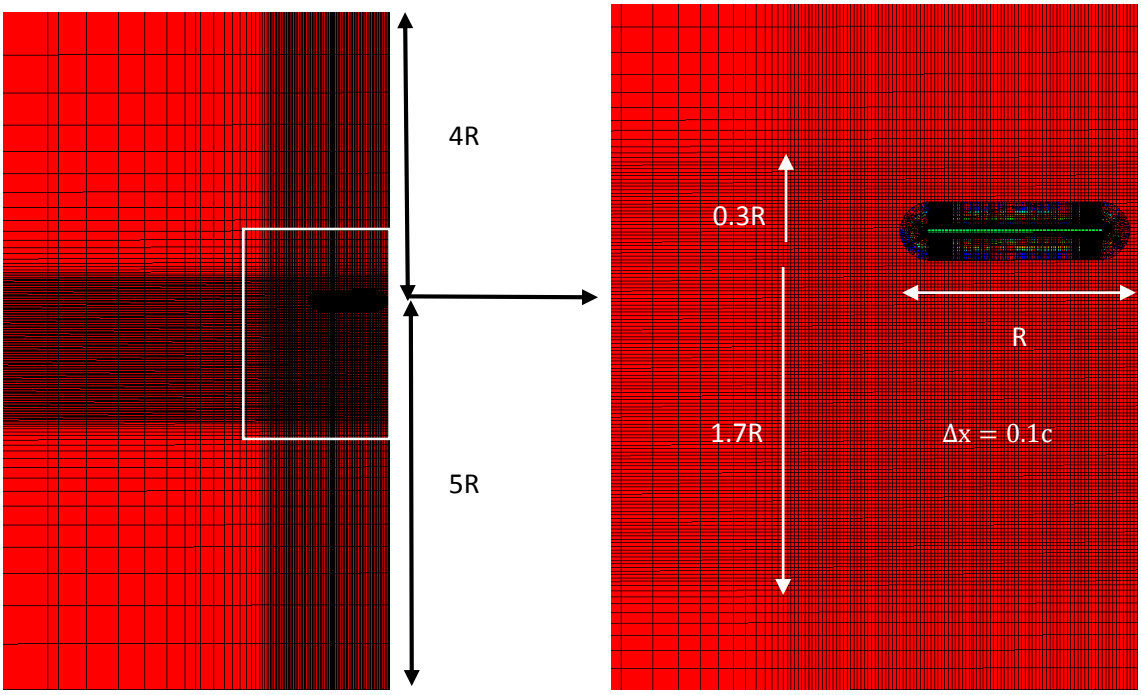
As can be seen from the above results, generally speaking, HAMSTRAN works well with overset grids. The mesh is able to successfully capture the pressure distribution at different areas of the wing in a much shorter time.

3.6. Overset lifting rotor

All the above cases tested are implemented on immobile mesh with no grid motion. So in order to test how HAMSTRAN works with moving grids, flow solutions around the *Caradonna-Tang* $rotor$[11] were computed using overset grids. This geometry is a lifting rotor consisting of 2 identical blades. Due to the periodic nature of flow around the lifting rotor, the flow around one blade is the same as the other one, so calculations were performed only for a single blade. In this case, the rotor rotates at a tip Mach number of 0.866 in a turbulent flow with a Reynolds number of 3.93 million. The blade also has a collective angle of 8 degrees. In this mesh, the near body strand grids move along with the rotor blade while the background Cartesian grids are static. The rotational speed of the rotor is 0.5° per iteration so it takes 720 iterations to complete a full revolution cycle. The simulation results are compared with the *experimental* $data$[18] and results from HELIOS[19].

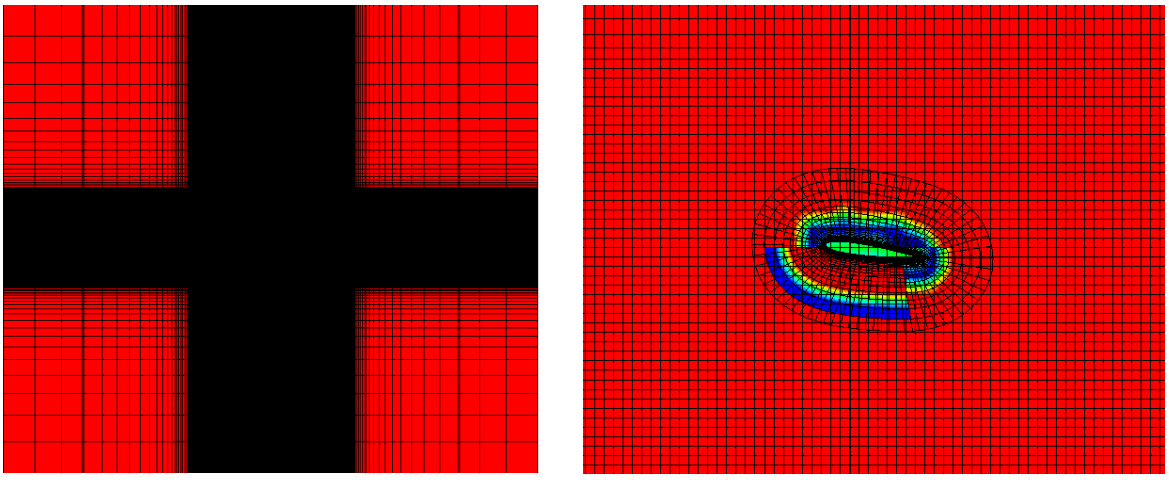
1. The overset mesh system

The input tessellation has 14,000 quad grids in the main body and 2100 triangular grids on the tip of the blade. The near body domain has 54 strand layers with an initial wall-spacing of 1×10^{-5} chord length and a stretching ratio of 1.2, resulting in 0.87 million grids in total. The background Cartesian mesh has about 4.6 million grid cells. The mesh can be seen in Figure 86. In the background Cartesian mesh, the length of a grid is 0.1 chord length in the small box and the mesh is stretched far into the outer boundary in an effort to capture the wake.



a. Overview of the overset mesh b. A blow up near the rotor blade

Fig 86. An overview of the overset mesh for the lifting rotor in the x plane

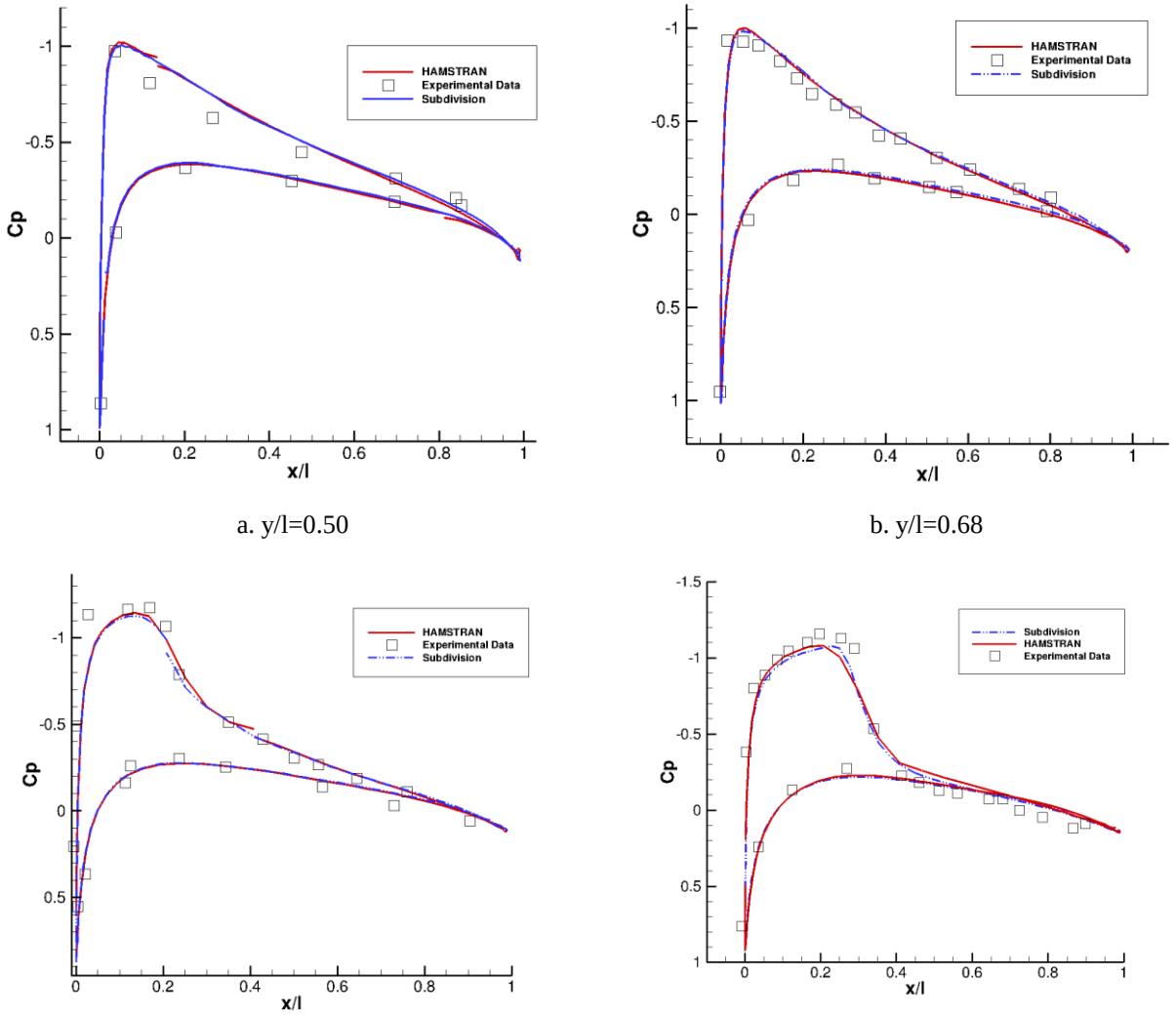


a. Overview of the entire mesh b. A blow up near the rotor blade

Fig 87. An overview of the overset mesh for the lifting rotor in the y plane

2. The pressure distribution

As can be seen in Figure 87, the results from HAMSTRAN are very similar to that of the subdivision algorithm, and both of them showed good agreement with experimental data. There is an obvious pressure jump in Figure 87d, indicating the mesh system successfully captured the shock at the tip of the rotor blade. The results show that HAMSTRAN works well when combined with moving grids.



a. y/l=0.50

b. y/l=0.68

c. y/l=0.80

d. y/l=0.89

Fig 88. Pressure distribution on different areas of the lifting rotor

3. The simulation results

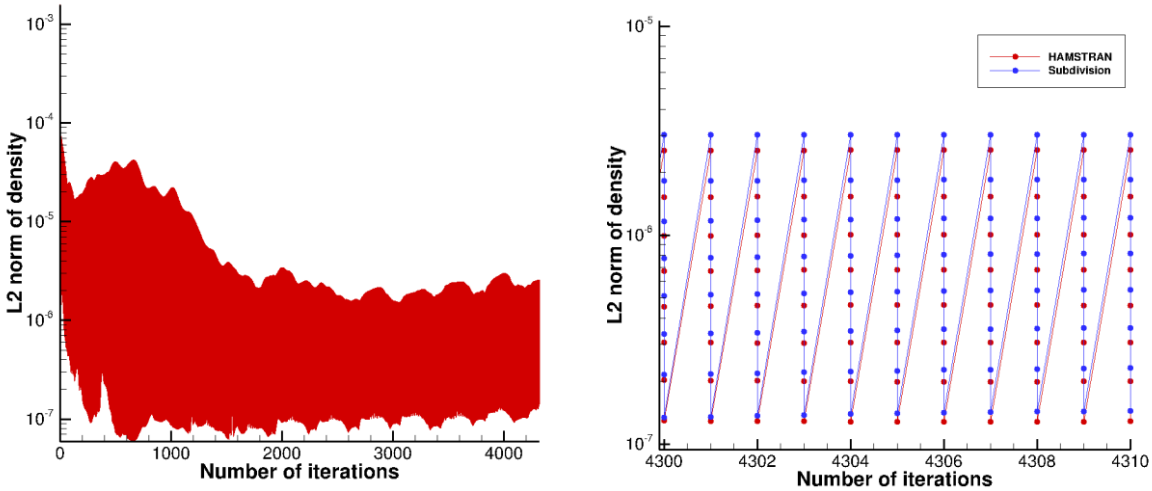


Fig 89. The convergence history of density residual (overset rotor)

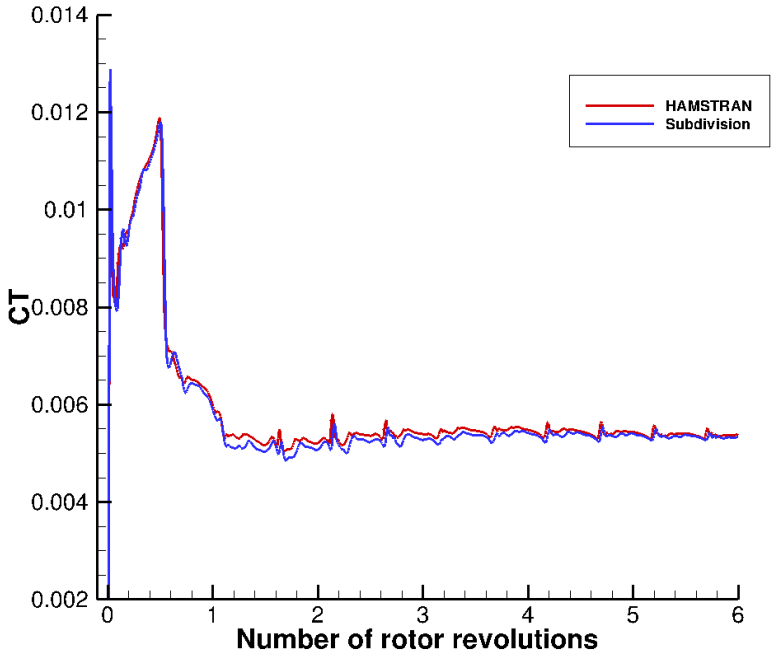


Fig 90. The convergence history of thrust coefficient (overset rotor)

	Number of iterations	**Thrust coefficient**	**Experiment C_T**	**cputime**
subdivision method	4320	0.00534	0.00473	198373
HAMSTRAN	4320	0.00538	0.00473	67637

Table 17 .Simulation results for the overset lifting rotor case

As can be seen from Figure 88, unsteady density residual dropped by 1-2 orders during the 8 sub-iterations. The rate of convergence is approximately the same for HAMSTRAN and subdivision algorithm. In this unsteady case, the density residual didn't give out much information, so this paper also plotted the convergence history of thrust coefficient, Figure 89. As can be seen in this plot, the thrust value appears converged after 3-4 rotor revolutions. As is shown in Table 17, after 6 revolutions, CT reached a value of 0.00538 for HAMSTRAN and 0.00534 using the subdivided mesh. Both are significantly different from the experimental value but are more in line with the $HELIOS$[18] results, which is 0.0052. So in this case, HAMSTRAN didn't improve the final results but at least it reduced the time it takes to reach convergence.

Chapter 4: Conclusion

The HAMSTRAN algorithm is an indirect algorithm to generate an all-quad surface mesh for both 2D and 3D geometries. In order to explain this new algorithm, this work is divided into 3 parts. The first part introduces the motivation of this entire research as well as the unique advantages of this method comparing to all the other methods. The second part explains the methodology in close details. Then this algorithm is implemented on several different geometries in both 2D and 3D to show the improvement in the mesh quality.

The third part is created to test the effectiveness of the newly generated mesh in the flow solving process. The results are compared with that of the subdivided mesh which was traditionally used. It was run on 6 cases: the triangular wedge in inviscid flow, the NACA0012 hybrid mesh in turbulent flow, the robin fuselage in low Mach-number flow, the lifting rotor case and 2 cases of OneraM6 wing transonic flow. The triangular wedge case is the simplest case of all, it is used to test the performance of the newly generated mesh in unsteady inviscid flow. Then the NACA0012 case is computed to test the application of HAMSTRAN in hybrid background meshes in turbulent flows. One of the OneraM6 wing case is built to test the combination of HAMSTRAN with strand grids and the other is built to test its combination with overset mesh. The robin fuselage case is built to test the combination of HAMSTRAN with curved strands. The last case, the lifting rotor case is built to test how HAMSTRAN corresponds with a moving mesh system.

The HAMSTRAN algorithm is mostly based on Q-Tran but also has a lot of advantages that is absent from Q-Tran, such as the ability to work on a wider range of background mesh, having more explicit steps to follow and adding a scheme to generate strand templates for each vertex. Comparing to the simple Catmull-Clark subdivision algorithm, HAMSTRAN is able to generate a mesh with far fewer irregular vertices and a much lower average skewness, not to mention it generates considerably fewer grids. One of the unique characteristics of this work is that the newly generated mesh is tested in the flow solver, unlike many other papers that only introduce the mesh generation scheme. After implementing HAMSTRAN, the computational cost reduced by almost $1/3^{rd}$ without reducing the accuracy of the simulation. Especially in the 2D NACA0012 case, even a coarser HAMSTRAN mesh produced more accurate results comparing to a finer Catmull-Clark subdivision mesh.

As can be seen in the previous chapter, all six cases can be considered as a success. The Hamiltonian loops became much longer and smoother after implementing HAMSTRAN into the flow solver. This makes it easier for line-implicit methods to be carried out along these strands, increasing the accuracy of the HAMSTR flow solver. The most successful case is the NACA0012 case. After implementing HAMSTRAN, not only was lift and drag coefficient showed much better agreement with the OVERTURNS results, the pressure plot also became much smoother, indicating there is less fluctuation in the pressure distribution along the NACA0012 airfoil. Even when the background mesh is subdivided twice, the results are still less accurate comparing to the HAMSTRAN algorithm.

The least successful cases are the overset cases. As can be seen in case 5 and 6, although the pressure distribution of the simulation results showed decent agreement with experimental data, it isn't as good as the cases implemented purely on strand grids. This can be studied more thoroughly in the future.

Although HAMSTRAN increased the quality of the mesh significantly, there is still room for improvement in the future. First, simulations on even more complicated geometries should be tested in the future, such as the hub or wing-body geometry. Second, one can work on building schemes to further reduce the number of irregular grids, increasing the quality of the mesh even further. Third, the way HAMSTRAN deals with quad-dominant mesh isn't as good as it could be, it still increased the number of grids by twice and the quality of the output tessellation isn't satisfying enough. Finally, in this research, all the newly generated vertices are generated linearly without using higher order schemes. On a very fine mesh, this isn't much of a problem. However, if the mesh is coarse, then schemes that are able to flush the vertices to the surface of the geometry are required.

References

[1]. John.D.Anderson, Computaional Fluid Dynamics, 1st edition, 1995.

[2]. T. J. Craft, The Navier Stokes Equations, the University of Manchester.

[3]. J. Blazek, Computational fluid dynamics: principles and applications, 2001.

[4]. Yong Su Jung, Bharath Govindarajan, James Baeder, A Hamiltonian-Strand Approach for Aerodynamic Flows Using Overset and Hybrid Meshes, 72nd Annual Forum of the American HelicopterSociety, 2016.

[5]. Javad Sadeghi, Faramarz F. Samavati, Smooth reverse Loop and Catmull-Clark subdivision, 2011.

[6]. Altair Hyperworks, Hypermesh overview.

[7]. J.-F. Remacle, J. Lambrechts, B. Seny, Blossom-Quad: a non-uniform quadrilateral mesh generator using a minimum cost perfect matching algorithm, International journal for numerical methods in engineering, 2010.

[8]. S. J. Owen, m. L. Staten, Q-Morph: an indirect approach to advancing front quad meshing, international journal for numerical methods in engineering, 1999.

[9]. Mohamed S. Ebeida, Kaan Karamete, Q-TRAN: A New Approach to Transform Triangular Meshes into Quadrilateral Meshes Locally, Applied Math and Applications, 2013.

[10]. Yong Su Jung, Bharath Govindarajan, James Baeder, Turbulent and Unsteady Flows on Unstructured Line-Based Hamiltonian Paths and Strands Grids, AIAA journals, 2016.

[11]. F. X. Caradonna, C. Tung, Experimental and Analytical Studies of a Model Helicopter Rotor in Hover, NASA Technical Memorandum, 1981.

[12]. Dr. Norman W. Schaeffler, Rotorcraft Fuselage Drag Reduction via Active Flow Control: A Combined CFD and Experimental Effort, Fundamental Aeronautics Program.

[13]. Gregory, N, O'Reilly, C. L, Low-Speed Aerodynamic Characteristics of NACA 0012 Aerofoil Sections, including the Effects of Upper-Surface Roughness Simulation Hoar Frost, NASA, Jan 1970.

[14]. Langley Research Center, Turbulence Modeling Resource, NASA, Stanford/Maryland, USA.

[15]. NPARC Alliance Validation Archive, OneraM6 wing, NASA.

[16]. Schmitt, V. and F. Charpin, Pressure Distributions on the ONERA-M6-Wing at Transonic Mach Numbers, *Experimental Data Base for Computer Program Assessment,* Report of the Fluid Dynamics Panel Working Group 04, May, 1979.

[17]. R. J. Adrian, J. Westerweel, Particle Image Velocimetry, 2011.

[18]. Gray R.B, Mchahon H.M, Surface pressure measurements at two tips of a model helicopter rotor in hover, NASA, May 1980.

[19]. Hariharan, N., Wissink,A., and Steen, M, Tip vortex field resolution using an adaptive dual-mesh computational paradigm," 49th Aerospace sciences meeting including the New Horizons forum and Aerospace Exposition, Jan, 2011.