Free-Form Deformation (FFD)

Author: Janeita Reid
Reviewed by: Ping He

February 18, 2021

Contents

1 General Theoretical and Mathematical Preliminaries

Before we explain how to setup the FFD points for the pyGeo[1] module from the MACH-Aero framework, it's essential to first understand underlying theoretical and mathematical formulations. In doing so, the user will gain a better understanding of pyGeo's strengths and weaknesses and how these influence constrained optimization in any of the disciplines supported by DAFoam (aerodynamics, solid mechanics and hydrodynamics). The guidelines covered in this section are for general applications using FFD. More specific guidelines will be given in the sections that follow to highlight the features of the FFD module in MACH-Aero.

pyGeo is closely tied to the concepts supporting free form deformation (FFD). FFD was first presented by Sederberg and Perry as a method for sculpting solid models by conducting global or local deformations [14]. The classic method involves a $R^3 \rightarrow R^3$ mapping using trivariate tensor product Bernstein polynomials[14], but this approach has evolved to also

[1] https://github.com/mdolab/pygeo

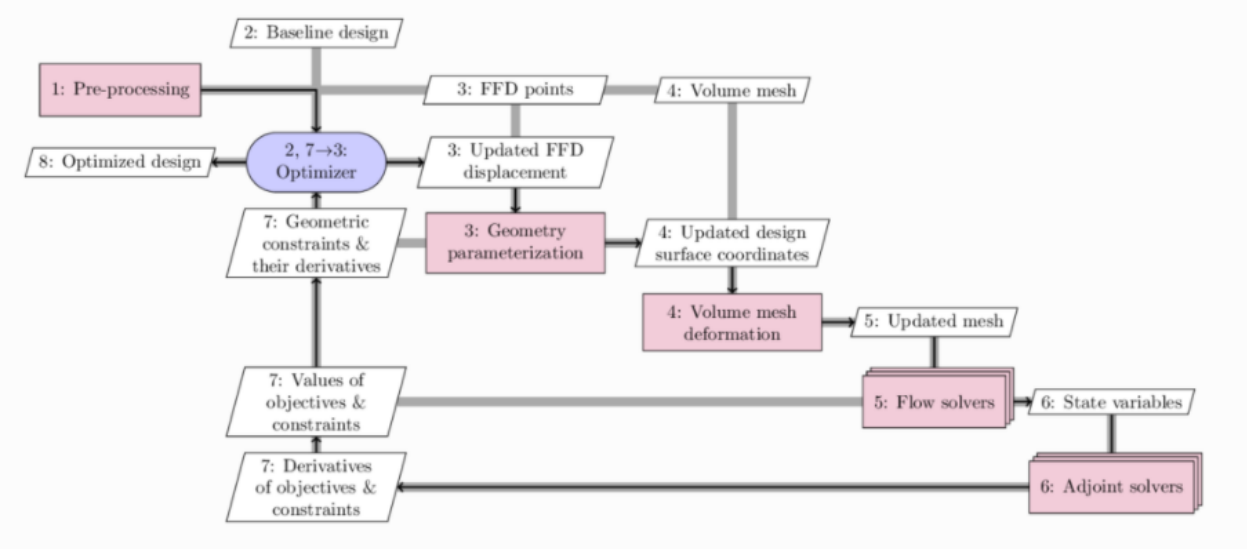

Figure 1: The pyGeo module for geometry parametrization (node 3) in the MACH-Aero framework [8]

.

use tri-variate Bézier, B-spline or NURBS volumes [5] for mapping and remapping object points. The FFD approach adopted in pyGeo was developed by Kenway and Martins [5].

To get a better understanding of how FFD works, one can imagine putting an object or any geometry into a clear, flexible plastic-like shape that is transparent. In other words, by using FFD the geometry of the object is embedded into a volume that can be manipulated by moving points at the surface known as FFD points [3].

In order to manipulate the object and arrive at an optimized geometry, there are three steps involved in FFD, namely :

1. Create an initial FFD box with volumetric control points (aka. FFD points).
2. Embed the design surface into the FFD box and find the mapping between the design surface's physical coordinates (x,y,z) and the FFD's parameter space (u,v,w).
3. Deform the design surface by moving the FFD control points.

To transform the steps above into mathematical formulations we take four things into consideration: the vertices of the initial embedded object given by $\overrightarrow{P}(u, v, w)$, the parametric space represented by (u_0, v_0, w_0), control points (FFD points) on the initial deformation lattice given by $\overrightarrow{Q}_{i,j,k}$, and the vertices of the deformed object represented by $P^{new}(u_0, v_0, w_0)$ with new FFD points $\overrightarrow{Q}^{new}_{i,j,k}$.

Additionally, basic knowledge of B-spline curves, surfaces or volumes presented by Kenway and Martins [5] is also necessary to understand the parametrization of different objects.

For curves we have:

$$C(u) = \sum_{i=0}^{N_u-1} N_{i,m_u}(u) Q_i \tag{1}$$

For surfaces we have:

$$S(u,v) = \sum_{i=0}^{N_u-1} \sum_{j=0}^{N_v-1} N_{i,m_u}(u) N_{j,m_v}(v) Q_{i,j} \tag{2}$$

And for volumes we have:

$$V(u,v,w) = \sum_{i=0}^{N_u-1} \sum_{j=0}^{N_v-1} \sum_{k=0}^{N_w-1} N_{i,m_u}(u) N_{j,m_v}(v) N_{k,m_w}(w) Q_{i,j,k} \tag{3}$$

A good explanation of B-spline can be found here [4]. With these in mind we develop the steps mentioned above considering the mathematical theory behind each of the three steps.

Step 1

The first step mentioned above represents the FFD module coupled with pyGeo. This tool will be elaborated further on in the tutorial.

Step 2

We need to consider that the embedded object which lies in a cartesian space is mapped into an initial trivariate B-spline volume using the equations below as given by Kenway and Martins [5] and Ronzheimer [12]. We have adopted the notations and overview presented by Ronzheimer [10, 12] with a small modification considering $Q^{new}_{i,j,k}$ instead of $R_{i,j,k}$.

For the mapping, a Newton method is used to calculate the parametric coordinates from given Cartesian coordinates[11, 3]. In other words, a mapping is determined between the FFD points (parameter space) and the surface geometry (physical space) [3]. This is done by applying the Newton search on Equation 4 for each vertex of $\overrightarrow{P}$ by solving for u_0, v_0, w_0 with the aim of mapping the points to the B-spline volume.

$$\overrightarrow{P}(u,v,w) = \sum_{i=0}^{n_u} N_{i,m_u}(u), N_{j,m_v}(v), N_{k,m_w}(w) Q_{i,j,k} \tag{4}$$

N_{i,m_u}, N_{j,m_v} and N_{k,m_w} are B-spline basis functions of degree m_u, m_v, and m_w. These are defined as knot vectors as follows:

1. $N_i : 0, .., 0, U_{m_{u+1}}..., U_{n_u}, 1, ..., 1$
2. $N_j : 0, .., 0, v_{m_{v+1}}..., v_{n_v}, 1, ..., 1$
3. $N_k : 0, .., 0, w_{m_{w+1}}..., w_{n_w}, 1, ..., 1$

The basis functions mentioned previously are calculated recursively using:

$$N_{i,1_u} = \begin{cases} 0, & u_i \leq u < u_{i+1}, \\ 1, & \text{otherwise.} \end{cases} \tag{5}$$

and

$$N_{i,m_u}(u) = \frac{u - u_i}{u_{i+m_{u-1}} - u_i} N_{i,m_{u-1}}(u) + \frac{u_{i+m_u} - u}{u_{i+m_u - u_{i+1}}} N_{i+1,m_{u-1}}(u)$$

Step 3
Finally, the B-spline volume that was calculated in Equation 4 is replaced by a new B-spline volume with control points $Q^{new}_{i,j,k}$. This is done by remapping the u_0, v_0, w_0 values previously calculated to obtain the deformed object point represented by $P^{new}(u_0, v_0, w_0)$.

$$\overrightarrow{P}^{new}(u_0, v_0, w_0) = \sum_{i=0}^{n_u} N_{i,m_u}(u_0), N_{j,m_v}(v_0), N_{k,m_w}(w_0) Q^{new}_{i,j,k} \tag{6}$$

The initial control point lattice $\overrightarrow{Q}_{i,j,k}$ in Equation 4 and the new one in Equation 6 $Q^{new}_{i,j,k}$ perform the same function: to control the deformation of the embedded geometry. The final deformation of the embedded object is determined by $Q^{new}_{i,j,k}$. From Figure 2 it can be seen that the control lattice been deformed. This means that the initial points for $Q_{i,j,k}$ are no longer at their original position and have shifted creating a new control lattice $Q^{new}_{i,j,k}$.

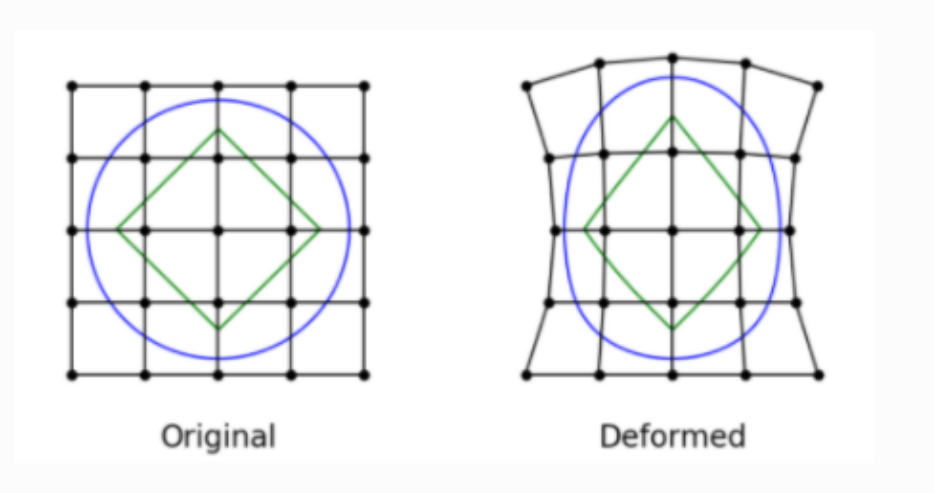

Figure 2: Before and after FFD points [6]

.

2 FFD considerations

FFD is best suited for small to medium deformations [13] and proper constraints should be in place to ensure feasible geometry changes [1]. As a volumetric deformation method,

FFD offers three advantages [1]: the embedded surface geometry has minimal effect on the computational cost, it is impartial to discrete geometry format, and the deformation quality is independent of the surface discretization quality. The main driver behind these advantages is that FFD volume parametrizes the *geometry change* rather than the geometry itself [5]; therefore, FFD eliminates the need for geometry abstraction and surface grid generation [13].

However, despite these advantages it's worth nothing that the grid topology is held fixed to ensure that as the design variables are modified, the function gradients are unaffected and remain continuous[5]. The implication of this is that lattice controls the deformation of the object and there is no continuous interactive deformation involved.

3 FFD in DAFoam: pyGeo

At the start of this tutorial, we noted that FFD allows the user to conduct both global and local deformation. Within pyGeo we can also control both the local and global shape of the geometry during optimization [3].The parametrization using pyGeo can be applied to structural domain, wetted surface of an aircraft [5], airfoil surface and wind turbine blade geometry.

In the example that follows, we will look at how to conduct a aerodynamic analysis of the NACA 0012 airfoil and the steps taken by pyGeo to parametrize the geometry.

3.0.1 Initial setup

Before you configure the settings in your FFD folder its important to perform the following steps first:

1. Load your mesh in the appropriate folder. You can either use a mesh you generated from another program or run the meshing option configured in DAFoam. If you plan to use another mesh generated from another program, it won't be necessary to run the **preProcessing.sh** command to generate your mesh in DAFoam. You'll need to type blockMeshDict to populate your Polymesh file in the case folder if you use a grid generated from another program.

2. Configure your runscript.py with the appropriate values for your particular case. For the case of optimizing for drag with lift constrained, you will need to first generate the correct angle of attack using the following command in your terminal: **mpirun -np 4 python runScript.py –task=solveCL**. But before you do that, continue to configure your runscript.py script until you reach the section **Design Variable Setup**. In the next sections we'll elaborate some more on what to do next.

3.0.2 Point selection and Troubleshooting

Within DAFoam you will be working with two files to modify your FFD points and control the geometric changes that are linked to the design variables in the adjoint equations: the

Design Variable Setup section in the **runscript.py** and the **genFFD.py**.

```
# ======================================================================
# Design variable setup
# ======================================================================
def alpha(val, geo):
    aoa = val[0] * np.pi / 180.0
    inletU = [float(U0 * np.cos(aoa)), float(U0 * np.sin(aoa)), 0]
    DASolver.setOption("primalBC", {"U0": {"variable": "U", "patches": ["inout"], "value": inletU}})
    DASolver.updateDAOption()


DVGeo = DVGeometry("./FFD/wingFFD.xyz")
DVGeo.addRefAxis("bodyAxis", xFraction=0.25, alignIndex="k")
# select points
iVol = 0
pts = DVGeo.getLocalIndex(iVol)
indexList = pts[:, :, :].flatten()
PS = geo_utils.PointSelect("list", indexList)
# shape
DVGeo.addGeoDVLocal("shapey", lower=-1.0, upper=1.0, axis="y", scale=1.0, pointSelect=PS)
daOptions["designVar"]["shapey"] = {"designVarType": "FFD"}
# angle of attack
DVGeo.addGeoDVGlobal("alpha", value=[alpha0], func=alpha, lower=0.0, upper=10.0, scale=1.0)
daOptions["designVar"]["alpha"] = {"designVarType": "AOA", "patches": ["inout"], "flowAxis": "x", "normalAxis": "y"}
```

Figure 3: Code snippet from runscript.py highlighting the genFFD.py section credit: DAFoam

Before changing the parameters for the FFD points, assess your geometry to determine which parts of should be optimized and which should remain fixed. Parts of the geometry that will be optimized will be enclosed with FFD control point lattices. These points will then be manipulated according to the mathematical formulations explained above.

The number of FFD control points will influence the design freedom of an optimization problem. Hence, the number of FFD control points used does affect the fidelity of the final design: too few control points and we risk ignoring certain possibilities in the design space; too many design points and we could have issues with mesh overlap, an over crowded lattice and slower optimization. This is why it is important to choose the correct number of design points through trial and error and assess the final design to determine if the results reflect feasible applications.

The effect of the number of design control points was clearly seen in a study conducted by Ronzheimer [11]. In the study it was shown that by doubling the number of FFD points from 4 to 8, the lift to drag ratio increased from 12% to 16% respectively. The optimization cycle also increased when the design points were doubled and clear differences in the final optimized shapes were noticed. Similar studies by Lyu, Kenway and Martins [9] demonstrated that the effect of reducing the number of airfoil control points from 48 to 24 resulted in negligible effect on optimal shape and pressure while further decrease to 12 and then to 6 showed noticeable differences in drag and pressure coefficients.

As a general guide we usually use 20 points for an airfoil section. However, users can use more or less depending on optimization goals.

3.0.3 Setting the Design Variables

We set the design variable using the pyGeo module in the MACH-Aero framework by following the steps below in the **Design Variable Setup** section of the runscript.py script:

1. Open the FFD folder in the case you are using for design optimization. It should look something like this below with the corresponding contents:

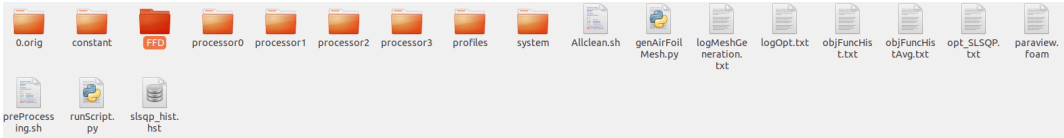

Figure 4: Typical case setup after running a simulation credit:DAFoam

Figure 5: FFD folder contents credit: DAFoam

2. Create a DVGeo object (DVGeo = DVGeometry("./FFD/wingFFD.xyz")) to manipulate the design surface geometry using the free-form deformation (FFD) approach. **NOTE: the FFD volume should completely contain the design surface**. The FFD file wingFFD.xyz is generated by running "python genFFD.py" in the FFD folder. We then add a reference axis ("addRefAxis") for twist. Note that in this case we run 2D airfoil optimization, so no twist is needed, and "bodyAxis" is not used [2].

   There are two options to visualize the FFD points:

   - First copy the constant folder, the system folder, and paraview.foam to the FFD folder, then go to the FFD folder and run this command to convert the FFD mesh (plot3D format) to the OpenFoam mesh (plot3DToFoam -noBlank wingFFD.xyz) and use Paraview to load the FFD/paraview.foam file to view the FFD points.
   - In version 2.2.1, we added a new utility to convert the plot3D file (*.xyz) format to the Tecplot format, such that you can load the FFD point in Paraview more easily (Paraview sometimes crashes when loading Plot3D files). To use it, just copy over the dafoam/dafoam/scripts/dafoam_plot3d2tecplot.py file and run `python`

`dafoam_plot3d2tecplot.py wingFFD.xyz wingFFDTecplot.dat`. **NOTE: Be sure to copy dafoam_plot3d2tecplot.py script to where you store the wingFFD.xyz file or else you will encounter problems**.

Now that you have done the necessary configurations in the genFFD.py file, the next step is to modify the runscript.py script at the Design variable setup mentioned earlier.

3. Once the FFD file is loaded and the reference axis is created, we select FFD points to move. The FFD file supports multi block meshes, but in this case we have only one block in the FFD, so we select "iVol = 0". We allow all the points to move so we set "pts[:, :, :]" for "indexList". Alternatively, we can select a subset of indices to move by setting a range for pts to move, e.g., indexList = pts[1:2, 3, 5:6].flatten() [2].

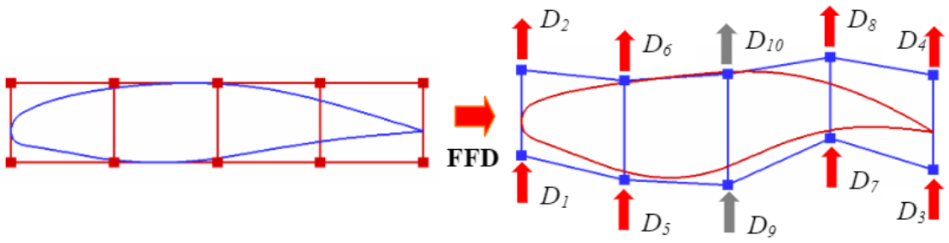

Figure 6: Example with 8 FFD points where D9 and D10 are fixed [12]

If you intend to use one block to run your simulation then your block setup will look like this below.

```
################ FFD ##############
nBlocks = 1

nx = [5]
ny = [2]
nz = [2]

corners = np.zeros([nBlocks,8,3])

corners[0,0,:] = [-0.010,-.0700,0.0]
corners[0,1,:] = [-0.010,-0.0700,0.1]
corners[0,2,:] = [-0.010,0.0700,0.0]
corners[0,3,:] = [-0.010,0.0700,0.1]
corners[0,4,:] = [ 1.01,-.07,0.0]
corners[0,5,:] = [ 1.01,-.07,0.10]
corners[0,6,:] = [ 1.01,0.07,0.0]
corners[0,7,:] = [ 1.01,0.07,0.1]
```

Figure 7: FFD setup in genFFD.py

We'll notice that there is a nx, ny, nz. These mean the number of points in the x, y, z coordinates.

For corners = np.zeros([nBlocks,8,3]):

The first entry is the FFD block number. In this case, there is only one block, that is why it is always zero. For examples using more than one blocks you can check the DPW4 case or the 30N30P Multi-element airfoil case where multiple FFD blocks are used, e.g., one for the wing and one for the tail.

The 2nd entry is the 8 corner indices, and the 3rd entry is the x,y,z coordinates of the corners.

If you intend to use two blocks then you'll need to modify the code to consider this type of configuration.

3.0.4 Simulation Example: NACA 0012

In this section we'll do a few examples to practice how to setup the getFFD.py and configure runscript.py.

Step 1: Understand the Geometry You have
Before we do any changes in the FFD files, we need to understand the type of geometry we are working with to know the limits of the FFD control points to consider. For a 2D airfoil such as NACA 0012 this means that were interested in the variations along the x and y axis. In this case we are using an airfoil with a blunt trailing edge where we removed a section. The coordinates the airfoil have been provided below for clarity.

0.0000000 0.0000000	0.2570083 -.0595755
0.0005839 -.0042603	0.2784042 -.0599102
0.0023342 -.0084289	0.3003177 -.0600172
0.0052468 -.0125011	0.3226976 -.0599028
0.0093149 -.0164706	0.3454915 -.0595747
0.0145291 -.0203300	0.3686463 -.0590419
0.0208771 -.0240706	0.3921079 -.0583145
0.0283441 -.0276827	0.4158215 -.0574033
0.0369127 -.0311559	0.4397317 -.0563200
0.0465628 -.0344792	0.4637826 -.0550769
0.0572720 -.0376414	0.4879181 -.0536866
0.0690152 -.0406310	0.5120819 -.0521620
0.0817649 -.0434371	0.5362174 -.0505161
0.0954915 -.0460489	0.5602683 -.0487619
0.1101628 -.0484567	0.5841786 -.0469124
0.1257446 -.0506513	0.6078921 -.0449802
0.1422005 -.0526251	0.6313537 -.0429778
0.1594921 -.0543715	0.6545085 -.0409174
0.1775789 -.0558856	0.6773025 -.0388109
0.1964187 -.0571640	0.6996823 -.0366700
0.2159676 -.0582048	0.7215958 -.0345058
0.2361799 -.0590081	0.7429917 -.0323294

0.7638202 -.0301515
0.7840324 -.0279828
0.8035813 -.0258337
0.8224211 -.0237142
0.8405079 -.0216347
0.8577995 -.0196051
0.8742554 -.0176353
0.8898372 -.0157351
0.9045085 -.0139143
0.9182351 -.0121823
0.9309849 -.0105485
0.9427280 -.0090217
0.9534372 -.0076108
0.9630873 -.0063238
0.9716559 -.0051685
0.9791229 -.0041519
0.9854709 -.0032804
0.9906850 -.0025595
0.9947532 -.0019938
0.9976658 -.0015870
0.9994161 -.0013419

From the airfoil coordinates above, we see that your largest x value is 0.9994161, while our largest y value is 0.0600172. This means that the lattice/ block we create needs to be a bit bigger than 0.9994161 x 0.0600172. Let us choose one that is 1.01 x 0.07. It is important that the control lattice is body-fitted. If the lattice is too big for example 100 x 100 for the example above, then the gradient of dCD/dFFD will become very small (because the FFD point is far away from the design surface). The problem with this is that dCD/dFFD will have very different magnitudes compared with other design variables, say dCD/dAlpha with Alpha being the angle of attack. This will cause problems for the optimizer because it always wants to receive gradients with similar orders of magnitude. Hence, body-fitted FFD points should be used at all times.

Step 2: Setup the FFD control points
We know that we intend to use one block and that this one will have a dimension of 1.01 x 0.07. We now need to map out the points in this block. These points will represent the vertices of our control lattice.

To create our FFD points we need to remember that NACA 0012 is a symmetric airfoil. We put our axis in the symmetric plane xy = 00 and start to map out our FFD control points.The order we choose to map out the control points is as follow: start with leading edge and then move to trailing edge. We have chosen to use 8 vertices for our block and will modify our genFFD.py to represent this.

The first point we enter is our coordinate with vertice (x,y) = (0,0). Although we have 2D airfoil, we are actually mapping out points for a 3D lattice as shown in the corners = np.zeros([nBlocks,8,3]). Taking this into consideration the point (0,0) will have coordinates $(-0.010, -.0700, 0.0)$ on our lattice. This represents the leading edge control point in the negative y direction. The second point we consider is $(-0.010, -0.0700, 0.1)$ which takes into consideration the thickness of our airfoil. Still at the leading edge we consider two more points to reflect the symmetry of our airfoil:[-0.010,0.0700,0.0] and [-0.010,0.0700,0.1]. The first point represents control point on the positive y direction, while the second point takes the thickness into consideration.

You will notice that so far only the leading edge and trailing edge points have been considered. This is because we need to prescribe the corners points for the FFD box, then the genFFD.py

script will generate the coordinates for all other FFD points. To get a better understanding of this, check the **writeFFDFile** and **returnBlockPoints** functions in genFFD.py.

Step 3: Visualize the control points
It's important that the control points cover the geometry, otherwise there will be errors. To visualize the control points, you can choose from two options as mentioned above.

Step 4: Apply constraints
Next we need to setup constraints to ensure that the optimized geometry is within feasible limits. To do this we use our runscript.py to modify the constraints. Here we can apply thickness constraints and volume constraints. FFD allows one to maintain constant volume even after geometry changes.

Lower and upper projection points are used by the FFD during optimization to calculate the new thickness and volume change. An example of this is shown in the image below. More details can be obtained at Mach Aero Tutorials.

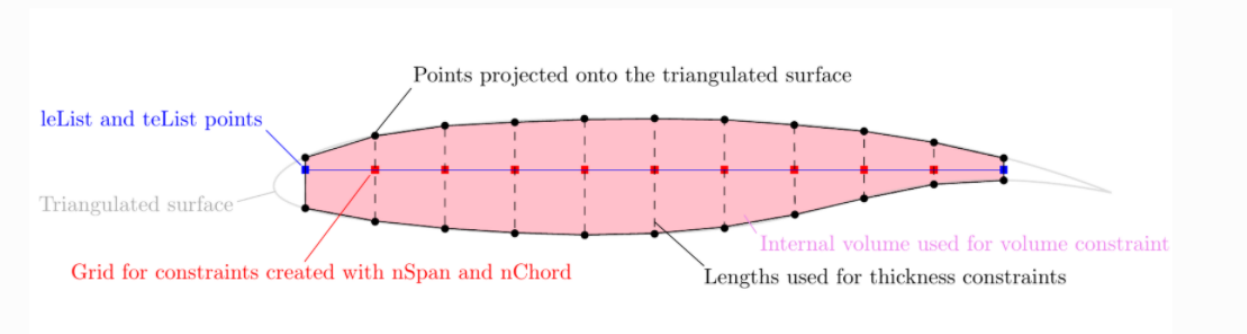

Figure 8: Example showing how volume and thickness constraints are calculated [7]

.

Step 5: Run simulation
When all the necessary parameters are entered the next step is to run **mpirun -np 4 python runScript.py**. Check the result and move the control points if needed once the optimization is complete.

4 Advanced Tips and Tricks

Ensure that your FFD points fully cover the design surface of your airfoil. You can convert *.xyz file (FFD file) to the OpenFOAM format using this command: plot3dToFoam -noBlank wingFFD.xyz Then you can load the FFD points and your OpenFOAM mesh into Paraview to check if FFD points fully contain the design surface.

5 Forum

There is no better way to learn than to ask questions. You can post questions on the DAFoam forum to get answers on bugs, general questions, tutorials, and installation. Before you post a question, first check out the best practices below:

1. Check first to see if your question has already been answered.
2. If your question has not been answered create a new discussion ensuring that:
   - The title of the question matches the details you'll provide.
   - The most appropriate category has been selected for your question.
   - Your question covers all the necessary details: what you did, what did not work, screenshots clearly showing details of your question, the results you were expecting and so on.
3. If you find that you'll need to ask multiple questions on different topics, consider posting each topic separately as a new discussion. This way people are more likely to find the question instead of scrolling through a long thread. Of course, if keeping the discussion in one thread reduces the likelihood of confusion and makes it easier to follow how the problem gets solved then consider doing so.

References

[1] George Anderson, Michael Aftosmis, and Marian Nemec. "Parametric Deformation of Discrete Geometry for Aerodynamic Shape Design". In: *AIAA Paper 2012–0965* 965 (Jan. 2012). DOI: 10.2514/6.2012-965.

[2] DAFoam. *Details of Run script*. URL: https://dafoam.github.io/mydoc_get_started_runscript.html. (accessed: 17.02.2021).

[3] Ping He et al. "An Aerodynamic Design Optimization Framework Using a Discrete Adjoint Approach with OpenFOAM". In: *Computers & Fluids* 168 (Apr. 2018). DOI: 10.1016/j.compfluid.2018.04.012.

[4] Kenneth I. Joy. *DEFINITION OF A B-SPLINE CURVE*. URL: https://www.cs.unc.edu/~dm/UNC/COMP258/LECTURES/B-spline.pdf.

[5] Gaetan Kenway, Graeme Kennedy, and Joaquim Martins. "A CAD-Free Approach to High-Fidelity Aerostructural Optimization". In: Sept. 2010. DOI: 10.2514/6.2010-9231.

[6] MDO Lab. *MACH-Aero Tutorials: FFD*. URL: https://mdolab-mach-aero.readthedocs-hosted.com/en/latest/machAeroTutorials/opt_ffd.html#opt-ffd. (accessed: 17.02.2021).

[7] MDO Lab. *MACH-Aero Tutorials: Geometric Constraints*. URL: https://mdolab-mach-aero.readthedocs-hosted.com/en/latest/machAeroTutorials/opt_aero.html#geometric-constraints. (accessed: 14.02.2021).

[8] MDO Lab. *Overview of MACH-Aero*. URL: https://mdolab-mach-aero.readthedocs-hosted.com/en/latest/machFramework/MACH-Aero.html. (accessed: 13.02.2021).

[9] Zhoujie Lyu, Gaetan K.W. Kenway, and Joaquim R.R.A. Martins. "RANS-based aerodynamic shape optimization investigations of the common research modelwing". In: *52nd Aerospace Sciences Meeting* 2014 (2014). DOI: 10.2514/6.2014-0567.

[10] Arno Ronzheimer. "Post-Parametrization of CAD-Geometries Using Freeform Deformation and Grid Generation Techniques". In: Jan. 2002, pp. 382–389. ISBN: 3-540-20258-7. DOI: 10.1007/978-3-540-39604-8_48.

[11] Arno Ronzheimer. "Prospects of Geometry Parameterization based on Freeform Deformation in MDO". In: Nov. 2006, pp. 1–10. ISBN: 84-85650-12-3.

[12] Arno Ronzheimer. "Shape Parameterisation Based on Freeform Deformation in Aerodynamic Design Optimization". In: Jan. 2004.

[13] Jamshid A. Samareh. "Aerodynamic shape optimization based on free-form deformation". In: *Collection of Technical Papers - 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference* 6 (2004), pp. 3672–3683. DOI: 10.2514/6.2004-4630.

[14] Thomas Sederberg and Scott Parry. "Free-form deformation of solid geometric models". In: vol. 20. Aug. 1986, pp. 151–160. ISBN: 0897911962. DOI: 10.1145/15886.15903.