AIAA AVIATION Forum
12-16 June 2023, San Diego, CA and Online
AIAA AVIATION 2023 Forum

10.2514/6.2023-3601

Free-Form Deformation of Parametric CAD Geometry via B-Spline Transformations

Marlena C. Gomez*, Marshall C. Galbraith†, Robert Haimes ‡
Massachusetts Institute of Technology, Cambridge, MA, 02139

This paper presents a method for local shape modification using free-form deformation to morph parametric geometry. The goal is analysis driven shape design which combines CAD-like parametric solid model geometry construction with free-form-like local deformation. A free-form deformation box is created and the geometry inside the box is deformed, where any analytic geometry inside the box which is not by default defined by a B-spline is converted to a B-spline. The free-form deformation is then used to move the surface B-spline control point net. This allows for the generation of smooth geometry, while keeping the number of degrees of freedom manageable for the optimizer. This method is demonstrated using optimization with the objective function minimizing the L^2-norm difference between a morphed and target shape. The set-up for a generalized geometric input is also presented, along with the computed sensitivity calculations which are necessary for shape design driven by analysis.

I. Nomenclature

A_i	=	area of a triangle element in FFD grid
m	=	minimum area of triangle elements divided by sum of neighboring triangle elements normalized to one
ϵ	=	integrated L^2-norm difference
n	=	length of B-spline knot sequence
p	=	degree of B-spline
s_i	=	L^2-norm error between two B-spline knots
ε_j	=	square error between two points on two B-splines at the same knot value
t_j	=	fractional distance between two B-spline knot values
w_j	=	weight of a coordinate c_j using Gaussian quadrature
$\Delta\mathbf{P}$	=	FFD control point perturbations

II. Introduction

Aerodynamic shape design is a continuously explored area of research. Solving a shape optimization problem using Computational Fluid Dynamics (CFD) analysis involves many steps; including geometry parameterization, surface and volume meshing, flow analysis, and the optimization itself. Often these steps may involve a number of different tools and disparate codes, and successful execution relies on specific setup prerequisites and careful orchestration of all of these. Consequently, extensive knowledge of the design problem and many tools involved is necessary, and often results in a design process that is not practical in general.

Part of the reason for the large number of codes involved in the shape optimization process comes from the geometric modelers used. The aerospace design community tends to develop custom in-house discrete, mesh-based tools, and commercially available shape control systems (CAD-based and CAD-free) have drawbacks. One drawback of CAD-based tools is, for proprietary reasons, they may hide geometric components necessary to compute values like parametric sensitivities. Furthermore, customization may be limited with these tools, so re-parameterization of the design before or during optimization may be necessary, which is time-consuming [1]. Alternatively, CAD-free tools lack meaningful design parameters, such as sweep or aspect ratio. Instead, the design parameters are typically defined

*Graduate Student, Computer Science and Engineering.
†Research Engineer, Department of Aeronautics & Astronautics, AIAA Senior member.
‡Principal Research Engineer, Department of Aeronautics & Astronautics, AIAA member.

via Free-form Deformation (FFD) boxes, which allow for generic shape changes, but do not provide any further insight into the parameter space [2, 3].

Engineering Sketch Pad (ESP) is a feature-based and parametric solid modeler [4]. The build process for a model involves the use of standard primitives solids, applied features, Boolean operators, transformations, and user-defined primitives and functions. The goal of the proposed work is to create a user-defined function within ESP for local deformation and optimization of aerospace geometries.

In previous research, geometry defined by B-splines was locally deformed and optimized with analysis in ESP by directly manipulating B-spline control points defining the geometry [5]. Although this work yielded promising results, there were also issues, including the large number of degrees of freedom. Generally, even small shapes defined by B-splines surfaces, like the rounded wingtip described in the paper, may be defined using significant number of control points. With each control point serving as a design parameter, this results in more degrees of freedom than is ideal. Additionally, the design parameters need to be cleverly constrained in order to maintain valid shapes. Specifically, the control point locations were limited to only allow movement in the positive normal direction relative to the original geometry surface. This restriction prevents the generation of self-intersecting geometry, but also limits the possible shapes which can be generated by the design optimization process. Without this constraint, the optimization process regularly generates self-intersecting geometry. An example of this is shown in Fig. 1, which is a self-intersecting geometry generated for an optimization where each control point could move without restriction in the x, y, and z-directions.

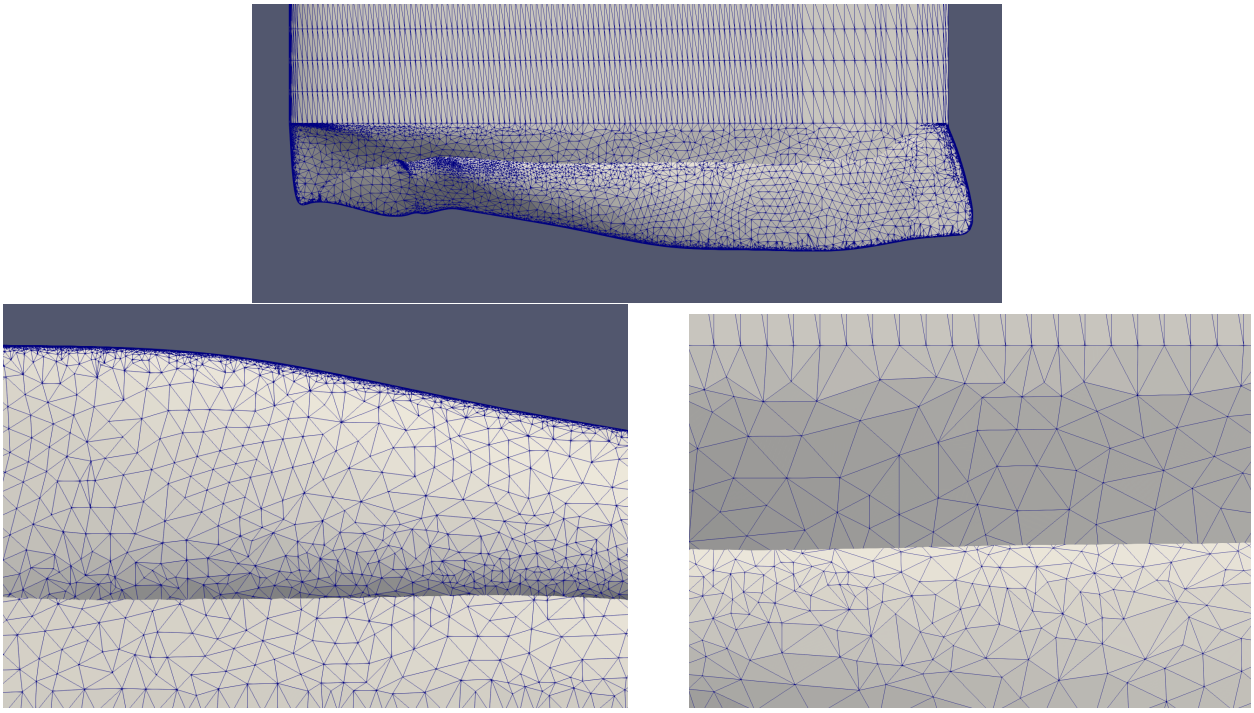


Fig. 1 Self-intersecting geometry resulting from an optimization of a wingtip surface where the B-spline x, y, and z control point locations served as independent design parameters.

Another issue observed previously was geometry generated during the optimization with undesirable "bumps" or "oscillations". For example, Fig. 2 shows two intermediate shapes (generated via an optimization where CFD analysis is driving the B-spline surface deformation of a wingtip) with significant bumps. While the final wingtip surface from this optimization was smooth, many intermediate geometries exhibited these type of "bumps". This is an issue because it is difficult for the flow server to converge on geometries like this, if it does, and the solver will spend a significant amount of time analyzing this complicated shape. It is preferable to exclude geometries like this from the design space, so the solver is not wasting time trying to find a solution for a shape which is "obviously" not optimal.

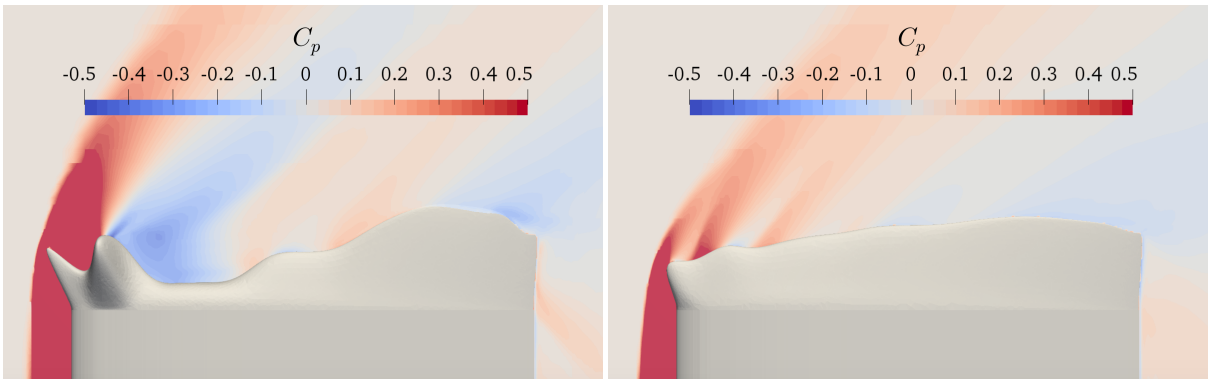


Fig. 2 Intermediate shapes in an optimization of a wingtip surface using the `Cart3D` design framework, where B-spline control points served as the design parameters.

Free-form deformation provides a way to address these concerns above [6, 7]. FFD is a technique for deforming a solid geometric model by deforming the space of the model. This is accomplished by first embedding the model in a FFD box defined by a tensor product trivariate B-spline (or Bernstein) polynomial [8] with a grid of control points, and then moving the trivariate B-spline control points to warp the space.The B-spline FFD box must be of a topologically higher dimension than the embedded geometry. For example, in order to morph a 1-dimensional curve in 2-dimensional Cartesian space, the FFD grid must be topologically 2-dimensional. FFD is extremely useful in that it preserves the connectivity and continuity of the underlying geometry when deforming it. Historically, this has been used frequently in a CAD-free manor to morphing mesh points in aerodynamic shape design [9]. However, deforming a mesh in this way may cause significant distortion to the mesh elements, resulting in a need to reparameterize and remesh the geometry, which is a time-consuming process. Furthermore, the final shape is a discrete mesh, rather than a Boundary Representation (Brep) geometry which can be subsequently used for other analyses or manufacturing. In order to preserve the Brep geometry, this effort applies to FFD to deform the B-spline control point net of B-spline surfaces instead of the points from a discrete mesh of the Brep surface. Since the geometry remains a Brep throughout the process with this approach, there is no need for the steps of reparameterizing or remeshing the geometry.

Using FFD to morph the geometry has a number of advantages over using the geometric B-spline control points directly as the design parameters. Firstly, this gives the user explicit control over the number of design parameters, so they can either decrease or increase the number of degrees of freedom as desired. Secondly, the movement of the control points of the FFD box can be restricted in such a way as to prevent the underlying geometry from self-intersecting while maintaining a large number of possible expressed shapes. So long as the Jacobian of the FFD mapping does not change signs, the embedded solid body will not self-intersect. In other words, if the minimum Jacobian in the FFD box is positive (and was positive initially), the embedded solid body will not be self-intersecting [10]. The remainder of the paper includes a description of the method, the penalty function formulated to prevent the Jacobian of the FFD mapping from becoming negative, and the design of the `ESP` the user-defined primitive are outlined in Section III. Sections IV and V are demonstrations in 2 and 3-dimensions respectively of gradient based optimization where the objective function minimizes the L^2-norm difference between a target and morphing shape.

III. Morphing Underlying B-Spline Shape with FFD

To deform a B-spline curve or surface using FFD, first the FFD parametric coordinates, (u, v), are computed for each B-spline Cartesian, (x, y), control point through a linear transformation. This forms a set of FFD parametric coordinate and Cartesian B-spline control point pairs. When the control points of the FFD box are moved in Cartesian space, the Cartesian coordinates of the B-spline control points are updated by computing new Cartesian coordinates via a forward evaluation of the deformed FFD box using the corresponding FFD parametric coordinate. Figure 3 shows an example of a B-spline curve encompassed by an FFD surface along with the control points for each shape. The cubic B-spline curve is approximating a NACA 0012 airfoil using 103 control pints, and the FFD surface is a cubic B-spline defined by a 4×4 net of control points.

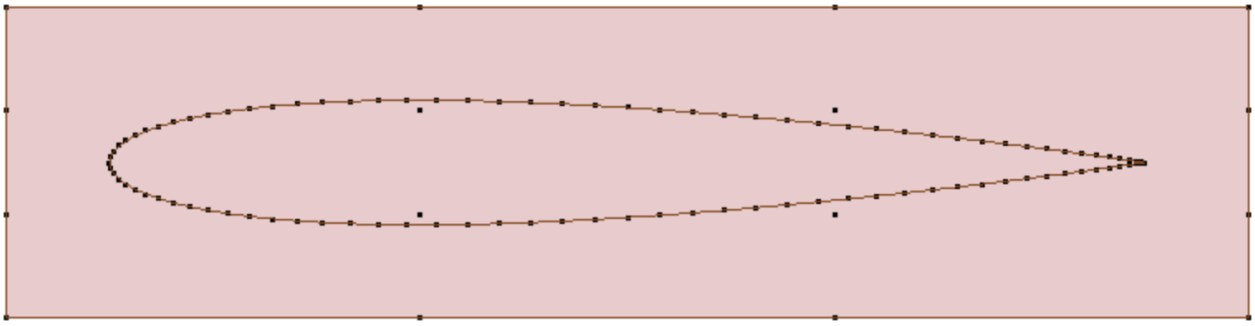


Fig. 3 NACA 0012 airfoil B-spline with 103 control points, and a grid of 16 FFD control points.

Figure 4 shows the effect on the airfoil geometry of moving one control point of the FFD grid. Because the FFD grid is a cubic B-spline, the movement of the single control point has a non-local impact, and actually deforms the entire airfoil in this example. As a result, this new geometry does not have a sharp bump, as happens when moving one of the B-spline control points directly, as show in Fig. 5. The extent of the non-locality is controllable by increasing or decreasing the polynomial degree of the FFD B-spline.

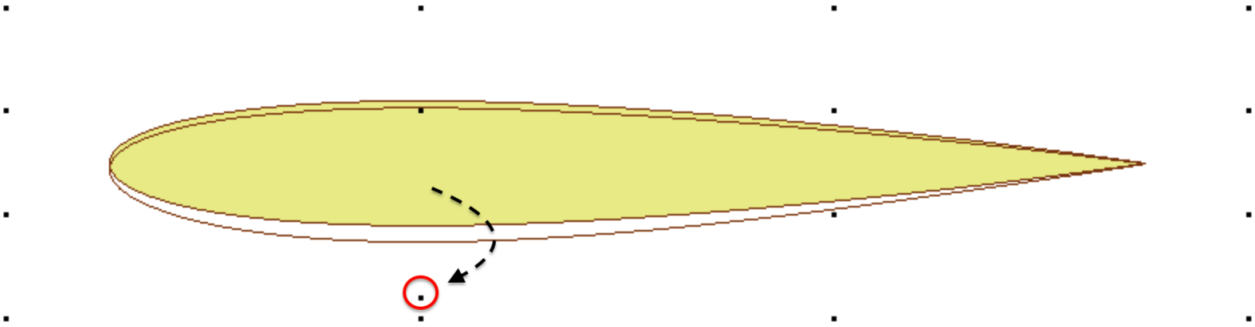


Fig. 4 Movement of one control point of an FFD grid, and the resulting new geometry. The yellow surface is the original NACA 0012 airfoil, and the new airfoil after perturbation via FFD is shown as a red outline.

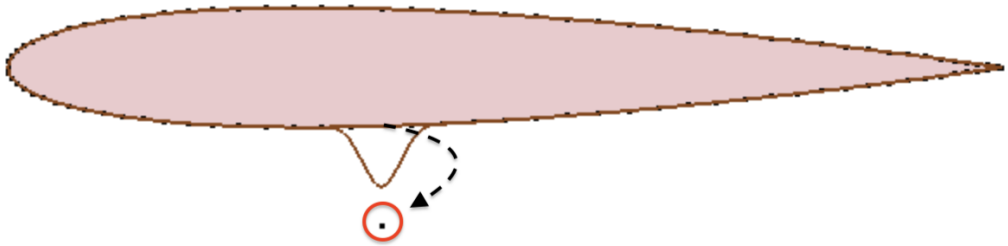


Fig. 5 Movement of one control point of the B-spline of the airfoil. The pink surface is the original NACA 0012 airfoil, and the new airfoil after moving one control point is shown as a red outline.

However, using FFD to morph the B-spline may still produce invalid self-intersecting geometry. As shown in Fig. 6, where the degree of the FFD B-spline polynomial is reduced to one, moving one of the control points of the FFD grid such that the FFD grid has a negative Jacobian results in an invalid curve with self-intersections. In order to prevent this, the control point locations of the FFD grid must be restricted such that there are no negative Jacobians, and therefore, no invalid geometry shapes generated. So long as the FFD surface does not have folds, then the embedded geometry will also not have folds.

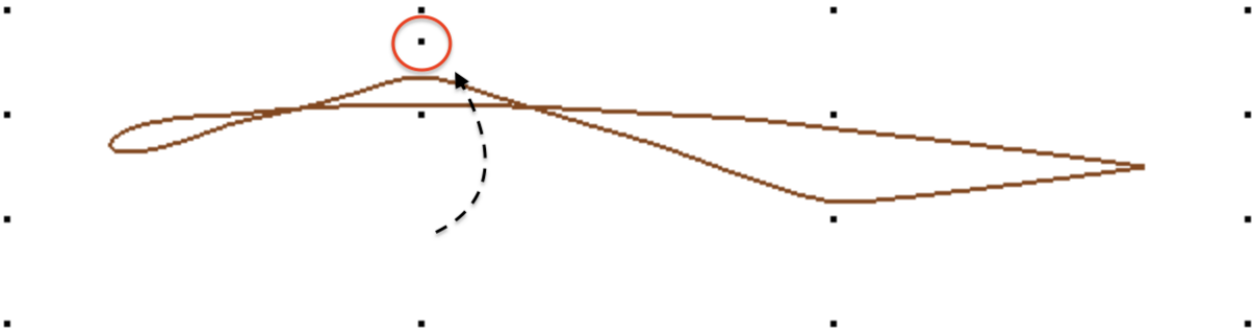
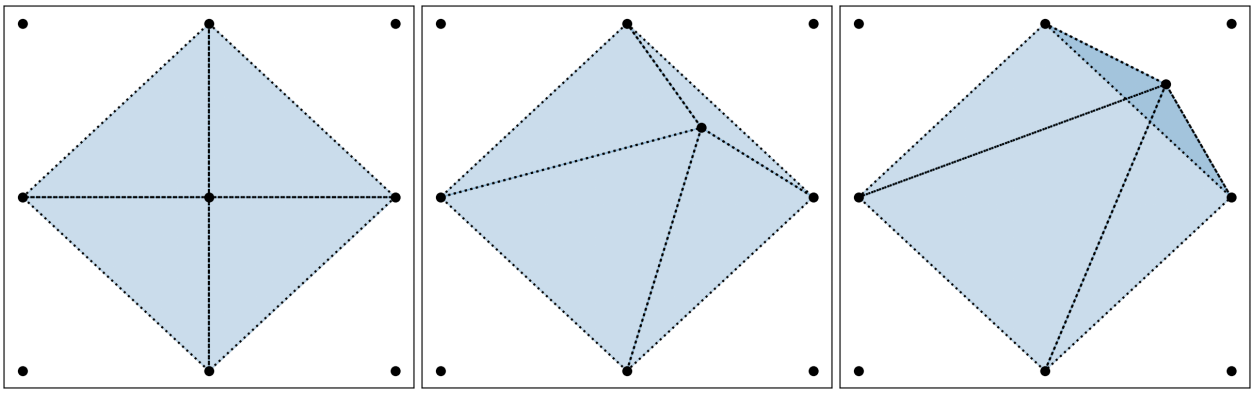


Fig. 6 A control point in the FFD grid moved such that there is a negative Jacobian in the FFD grid, and the resulting geometry is invalid.

A. Penalty Function

In order to prevent negative Jacobians in the FFD control point grid, any FFD control point must not move outside of the bounds of its neighbors. One way to do this is to ensure the areas formed by an FFD control point and its neighboring control points are always positive. For an interior control point in the FFD grid, there are four triangular areas formed by neighboring control points. The triangle elements shown for an individual point, which implies element are overlapping throughout the mesh. Initially, these areas are all equal and positive. To prevent these areas from becoming negative during the optimization, a penalty term is added to the objective function. For the initial FFD grid, there is no penalty value. During the optimization, a control point may move, and if it does so while maintaining a positive area for all four of the triangles, then there still should not be any penalty value. However, when any one of the triangle areas becomes negative, as seen for an interior control point in Fig. 7c, there should be a penalty. The penalty term should scale such that, when added to the objective function, the optimizer will avoid negative triangular areas. Additionally, the penalty term should scale such that larger negative areas are penalized more than smaller negative areas. This will drive the optimizer towards the situation where all the triangular elements of the FFD grid have positive areas, and all the FFD control points are therefore within the bounds of their neighbors.



(a) Penalty = 0 **(b) Penalty = 0** **(c) Penalty > 0**

Fig. 7 Interior FFD control point.

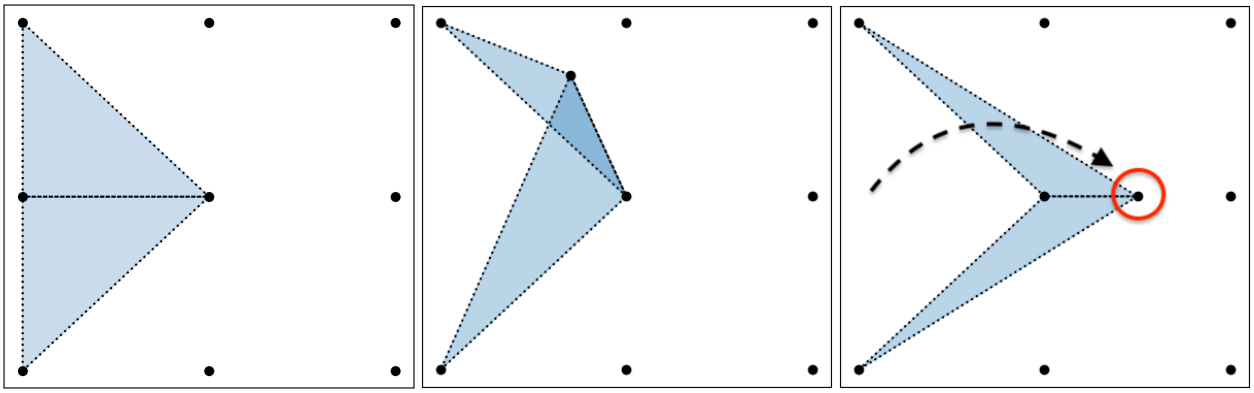
The minimum area of the neighboring triangle elements divided by the summation of all neighboring triangle element areas serves as a an indicator, m, for the penalty function. This indicator is positive when the center control point lies within the neighbors, grows linearly, and is normalized to one for the initial FFD grid where all triangle

elements have equivalent areas:

$$m = \begin{cases} 4 \cdot \frac{\min(\mathrm{A_i})}{\sum_{i=1}^{4} A_i} & \text{if interior control point} \\ \\ 2 \cdot \frac{\min(\mathrm{A_i})}{\sum_{i=1}^{2} |A_i| + \varepsilon \sum_{i=1}^{2} A_i^0} & \text{else (boundary control point)} \end{cases} \tag{1}$$

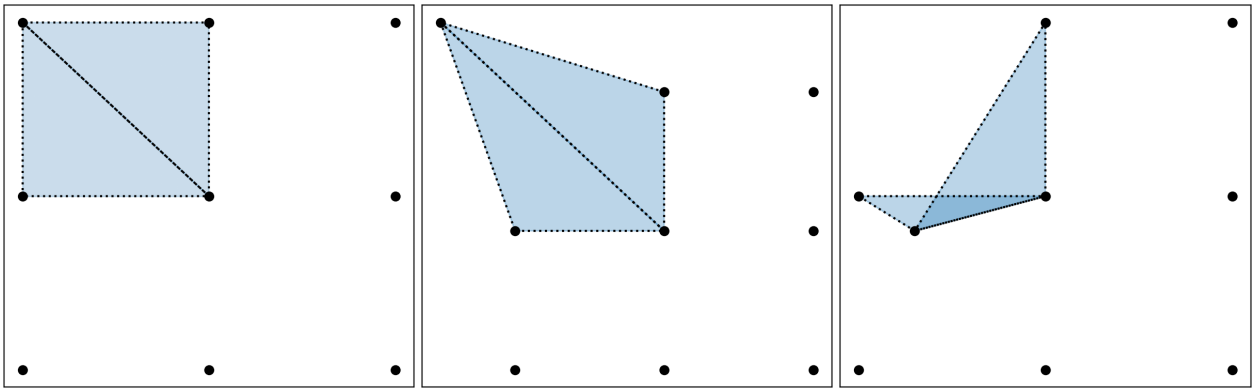
where A_i is the area of a triangle neighboring the a given control point, A_i^0 are the triangle areas of the un-deformed FFD, and ε is a small number (i.e. 10^{-11}). The ε term in the denominator serves to prevent division by zero.

For control points on the boundary of the FFD grid, only two triangle areas are influenced by the movement of an edge point, as shown in Fig. 8 and Fig. 9. Hence, the indicator m only includes two triangles, and becomes negative if one triangle area changes sign, as shown in Fig. 8b. However, as shown in Fig. 8c, it is possible for both of the influenced triangle to have negative areas. In order for the indicator to change sign in this situation, the denominator for boundary points is the sum of absolute values.



(a) Penalty = 0 **(b) Penalty > 0** **(c) Penalty > 0**

Fig. 8 Boundary FFD control point.



(a) Penalty = 0 **(b) Penalty = 0** **(c) Penalty > 0**

Fig. 9 Corner FFD control point.

For the gradient-based optimization, the penalty function needs to be differentiable, smooth, and monotonic. The penalty function also needs to be zero when the indicator is positive, and grow in magnitude when the indicator is "close to" or below zero. This behavior is readily accomplished with a switch-like function. Since gradient-based optimization is used, the switch function, as well as the penalty function, must be continuous and differentiable. The complementary

error function, shown in Fig. 10 is used for this purpose:

$$\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt \tag{2}$$

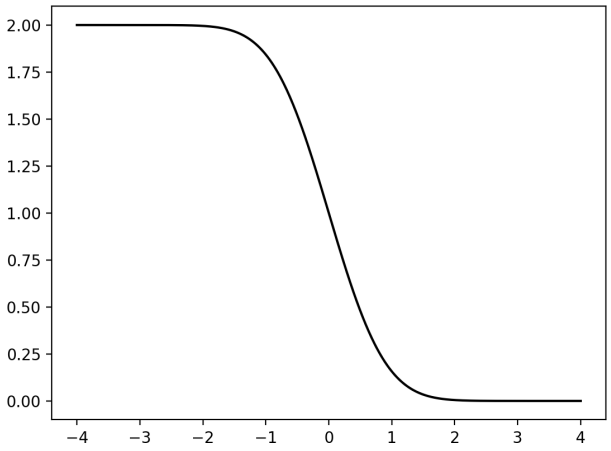


Fig. 10 Complementary error function (erfc)

Combing the function for the variable m and the complementary error function provides a framework for testing out a number of constants to appropriately scale the penalty term, so enough of a penalty is introduced in comparison to the objective function. A number of possible penalty functions formulated around the indicator m were considered and tested using the examples in Sections IV and V in order to get the desired behavior. Equation 3 shows the penalty function formulated with the constants determined through testing, and Fig. 11 shows the penalty value versus the value of m. The penalty function approximates zero until the argument m starts to approach zero. The value of m is zero when any of the triangle elements have an area of zero, so in order to prevent negative triangle element areas, the penalty value is introduced as the minimum element approaches zero. As the value of m decreases past zero, indicating a larger negative triangle element area, the penalty value increases. This penalty function, where the growth is linear past a value of $m = -1$, was found to work the best with the optimizer of the penalty functions tested.

$$P(m) = \operatorname{erfc}(10 \cdot \mathrm{m}) \cdot \left(\frac{1}{4} - \frac{1}{2} \cdot \mathrm{m}\right) \tag{3}$$

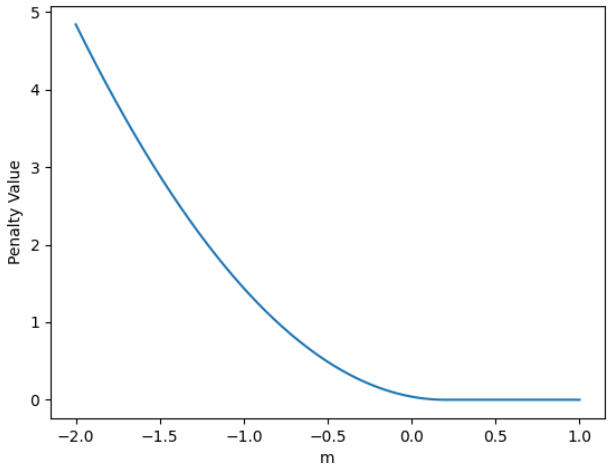


Fig. 11 Penalty Function

Figure 12 shows the penalty functions contour values when an interior control point is moved while neighboring point are fixed. The penalty function is zero as long as the middle control point remains within the neighboring triangles, and grows if any of the triangle areas are negated. However, the penalty function is not differentiable due to the minimum function used to compute m, which is only $C0$-continuous. As shown in Fig. 12, the sharp corners in the contours indicate where the penalty function is $C0$-continuous.

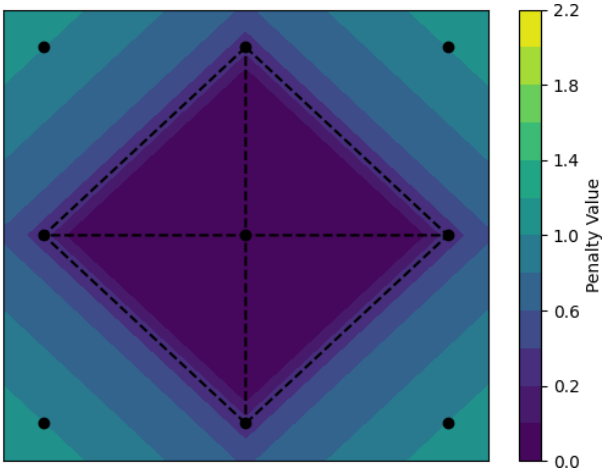
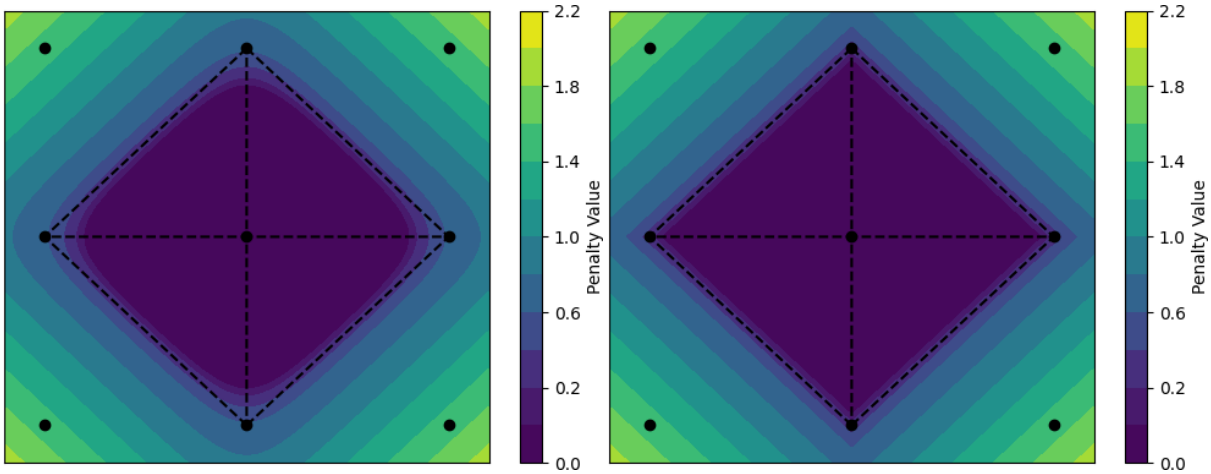


Fig. 12 Colormapping of penalty function where the interior FFD control point is moved to that location, and the outer frame of control points are fixed.



(a) $\alpha = 10$ **(b)** $\alpha = 1000$

Fig. 13 Colormapping of the penalty value when moving an interior FFD control point, using the smooth minimum function.

To resolve this issue, a smooth minimum function, which is a differentiable approximation of the minimum function, is used, as defined in Eq. 4 [11]. As illustrated in the plot in Fig. 13a, there are no sharp corners in the plot, which shows the function is continuous and differentiable. The variable α in the smooth minimum function controls the function approximation, a higher value of α results in sharper corners, as shown with Fig. 13b. It is more difficult to tell that the function is differentiable at the corners visually, but the smooth minimum function is still differentiable for any value of α. For the optimization results shown, the penalty function uses $\alpha = 1000$.

$$\text{smoothmin}(A_0, A_1) = \begin{cases} A_0 - \frac{\log(1+\exp(-\alpha*(A_1-A_0)))}{\alpha} & \text{if } A_0 < A_1 \\ A_1 - \frac{\log(1+\exp(-\alpha*(A_0-A_1)))}{\alpha} & \text{else} \end{cases} \tag{4}$$

B. ESP User-Defined Functions

The goal of this work is to incorporate the method of shape control defined above into a user-defined function (UDF) in ESP. This allows a user to define an FFD box, with a desired number of control points and spline degree, around any model they wish to morph. Additionally, the UDF provides the sensitivities required for optimization, so it can be used in a design optimization loop. The UDF requires an input model for the body to be morphed, for example, the geometry shown in Fig. 14 where a hemisphere joined with a cylinder.

Notably, none of the underlying geometry used to construct the Hemisphere-Cylinder are B-splines. Hence, before this geometry can be morphed, the appropriate geometric components of the geometry must be converted to B-splines. This results in a two-step process implemented with two UDFs: a conversion UDF and a morphing UDF. The process is broken into two UDF's because, during an optimization, the UDF which morphs the geometry is called each design iteration, but the conversion to B-splines of the model only needs to occur once. Hence, the conversion UDF accomplishes the B-spline conversion, and the output model from the conversion UDF is the input model for the morphing UDF.

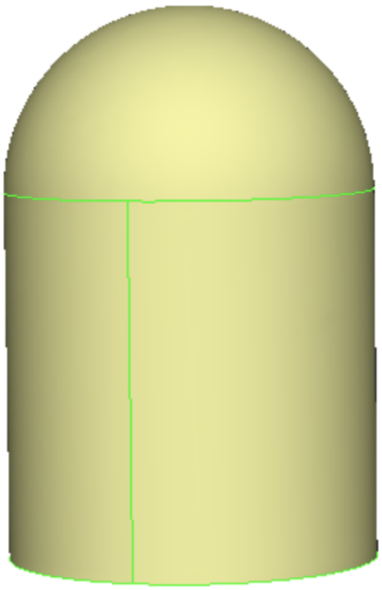


Fig. 14 Hemisphere-Cylinder Body.

Both UDFs take as input the body to be morphed and an FFD box body, which is created with the desired dimensions using the BOX primitive in ESP. The FFD box may either encompass some or all of the geometry, as shown in Fig. 15. The conversion UDF converts any surfaces and edges of the morphing body inside the FFD box to B-splines. This UDF uses the intersection between the two bodies to determine which parts of the model are inside the FFD box, and then converts only the relevant geometric entities to B-splines using the EGADS framework in ESP. This converted body is the input to the second morphing UDF.

The FFD box for the morphing UDF must be attributed in the ESP script to have the desired number of control points and degree for the FFD splines in the x, y, and z-directions. Figure 16 shows the FFD box and a user specified number of control points for the box. The morphing UDF takes in a list of design parameter values which correspond to the (x, y, z) perturbation of the FFD control points. The user can manipulate these values to control and shape the geometry the FFD box encompasses. Figure 17 shows two possible deformations of the Hemisphere-Cylinder body by moving select control points in the FFD box. Using perturbations to deform the FFD box simplifies setting constraints on various control points in an optimization statement.

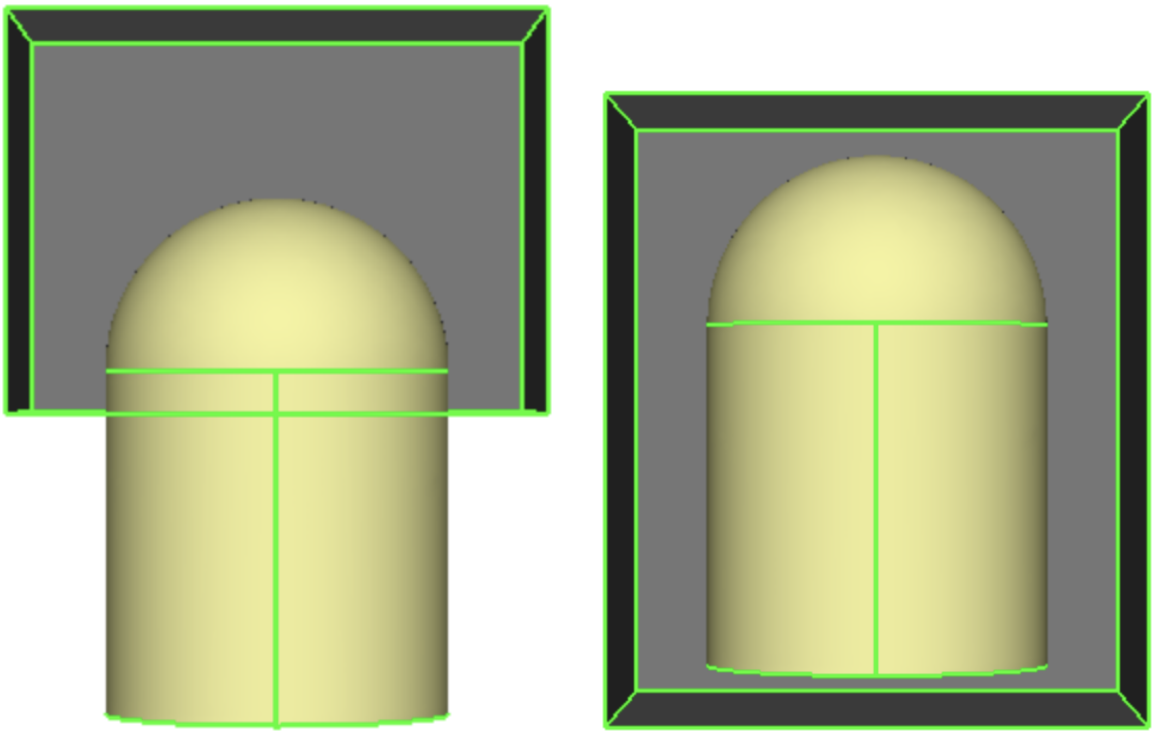


Fig. 15 Body to be morphed and FFD box.

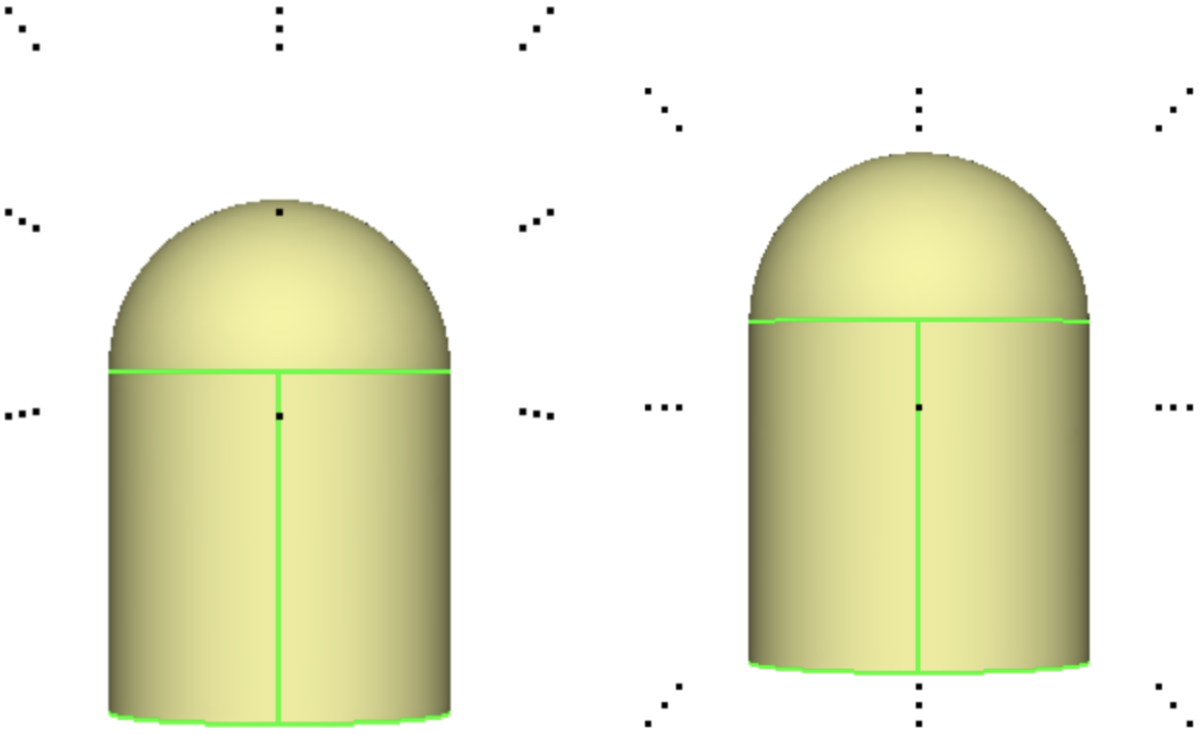


Fig. 16 Body to be morphed and FFD box points.

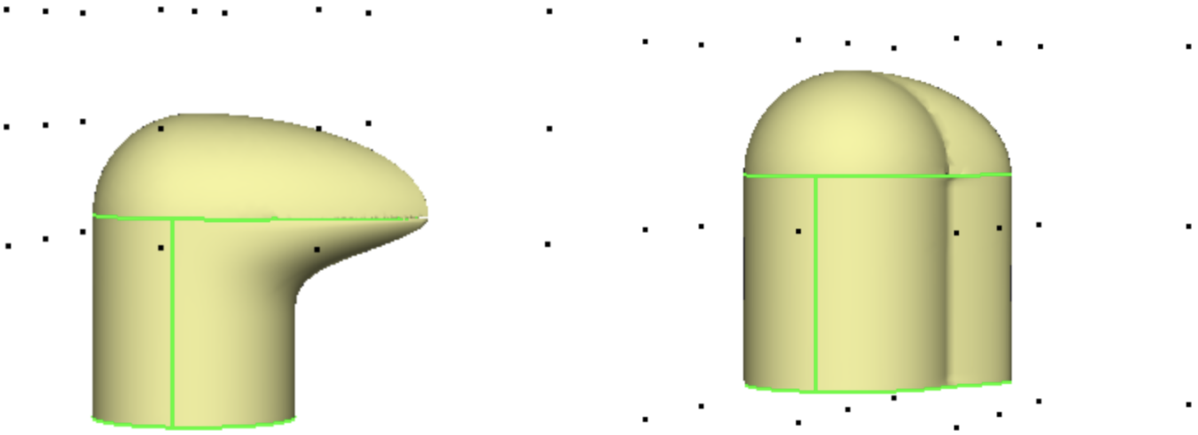
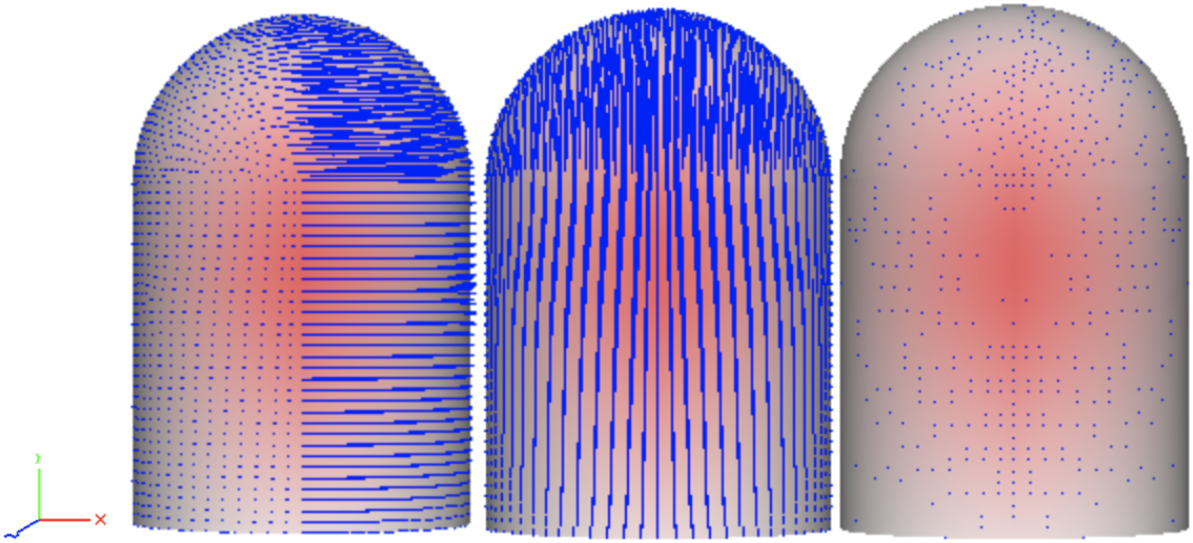


Fig. 17 Example of morphed bodies.

For gradient-based optimization, the sensitivities of the tessellation must be computed with respect to the design parameters, which for the FFD framework is the perturbation values of the FFD control points. These are computed for the FFD box control points using operator overloaded automatic differentiation [12]. Figure 18 depicts the sensitivities for the three components of a single FFD control point.



(a) X **(b) Y** **(c) Z**

Fig. 18 Sensitivities of an FFD box control point in the x, y, and z-directions.

The implementation of the analytic sensitivities computed using operator overloaded automatic differentiation are verified with a comparison to finite differences. The error between the analytic and finite difference sensitivities for two FFD control points are shown in Fig. 19 for a range of finite difference step sizes. The error is decaying at the anticipated 1st-order rate; consistent with the order of accuracy of the finite difference scheme. This is reasonable verification that the analytic sensitivities are correctly implemented.

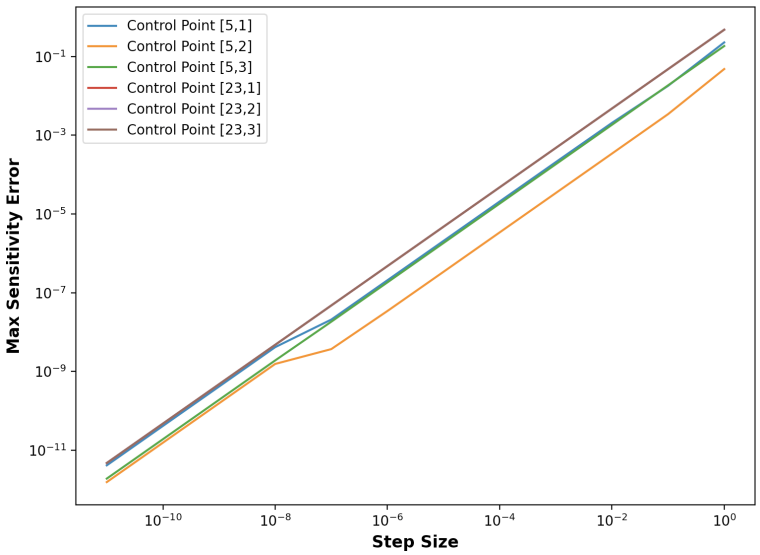


Fig. 19 Error of sensitivities using operator overloaded automatic differentiation for a number of control points.

.

IV. Optimization in 2D

A. 2D L^2-difference Minimization Examples

The results from the first case described in Section II of Ref. [5] are reproduced here using FFD to transform the underlying B-spline geometry. This case morphs a B-spline from an initial NACA 0012 airfoil shape to a NACA 8416 shape using the NLopt optimizer with the LBFGS algorithm [13, 14].

In order to do this, the B-splines for the airfoils are created using the `EGADS` framework in `ESP` [15] and the NACA 4-series equations. The starting airfoil B-spline is fitted to the shape of a NACA 0012 airfoil and the target airfoil is fitted to the shape of a NACA 8416 airfoil. The last coefficient of the NACA 4-series equations is changed from -0.1015 to -0.1036 to produce airfoils with a sharp trailing edge [16]. The B-spline curve defining each of the two airfoils is made up of 103 control points. Figure 20 shows the two airfoil B-splines.

(a) NACA 0012 **(b) NACA 8416**

Fig. 20 The starting and target airfoil B-splines.

A cubic B-spline square FFD surface that encompasses the space of the airfoil, which is defined by 16 control points, is also created as shown in Fig. 3. Perturbations to the FFD control points serve as the design parameters in the optimization. Since each control point can move in x and y, it follows there are $16 \cdot 2 = 32$ degrees of freedom. This is a significant reduction compared to the 103 B-spline control points which serve as design parameters in Ref. [5], which is 206 degrees of freedom.

As in the case from Ref. [5], the objective function is the L^2-norm difference between a morphing B-spline and a target B-spline. To compute the total L^2-norm error, ϵ, over a knot sequence of length n of a B-spline with degree p is:

$$\epsilon = \sqrt{\sum_{i=p+1}^{n-p-1} s_i}, \tag{5}$$

The L^2-norm error between two knot values, s_i is defined as:

$$s_i = \sum_{j=0}^{3} \varepsilon_j. \tag{6}$$

The square error of the two points (x_0, y_0) of the morphing B-Spline and (x_1, y_1) of the target B-spline at the knot value t_j are multiplied by the size of the interval to calculate ε_j:

$$\varepsilon_j = 2w_j(t_{i+1} - t_i)[(x_0 - x_1)^2 + (y_0 - y_1)^2], \quad j \in [0, 3] \tag{7}$$

The knot value t_j represents the fractional distance between two knots t_i and t_{i+1}:

$$t_j = c_j(t_{i+1} - t_i) + t_i, \quad j \in [0, 3] \tag{8}$$

The w_j and c_j values are the weights and coordinates for a Gaussian quadrature rule, which approximates an integral as:

$$\int_0^1 f(x)dx \approx \sum_{i=1}^{n} w_i f(x(c_i)). \tag{9}$$

Both the morphing and target airfoils shapes are cubic B-splines, so the polynomial which results from the square difference calculation has a degree of six. A 4-point Gaussian quadrature, with values shown in Table 1, integrates the L^2-norm exactly, since the p-point Gaussian quadrature rule integrates an exact result for polynomials of degrees less than $2p - 1$.

Table 1 4-point Gaussian Quadrature.

Index (j)	Coordinate (c_j)	Weight (w_j)
0	$\frac{1}{2}\sqrt{\frac{3}{7} - \frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{1}{2}\frac{18+\sqrt{30}}{36}$
1	$-\frac{1}{2}\sqrt{\frac{3}{7} - \frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{1}{2}\frac{18+\sqrt{30}}{36}$
2	$\frac{1}{2}\sqrt{\frac{3}{7} + \frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{1}{2}\frac{18-\sqrt{30}}{36}$
3	$-\frac{1}{2}\sqrt{\frac{3}{7} + \frac{2}{7}\sqrt{\frac{6}{5}}}$	$\frac{1}{2}\frac{18-\sqrt{30}}{36}$

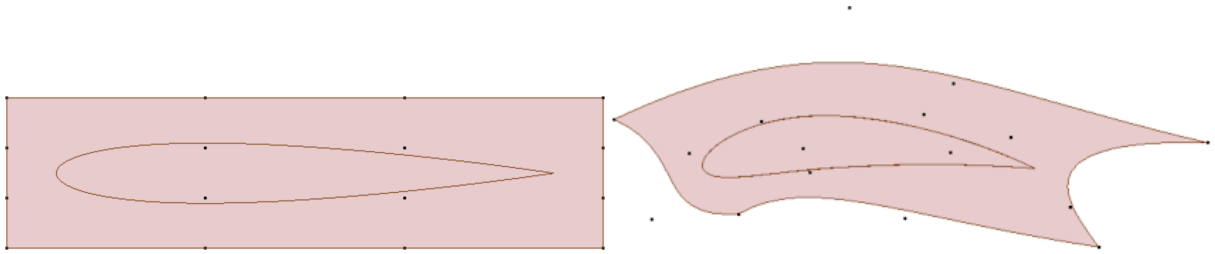
Adding the penalty function in Eq. 3 to Eq. 5 give the unconstrained optimization statement:

$$\underset{\Delta\mathbf{P}}{\text{minimize}} \quad \epsilon(\Delta\mathbf{P}) + P(m(\Delta\mathbf{P})) \tag{10}$$

where $\Delta\mathbf{P}$ is the set of control point perturbations that deform the FFD B-spline. In order to use gradient-based optimization, the derivative of the objective function is computed with respect to the $\Delta\mathbf{P}_i^0$ values grid using operator overloaded automatic differentiation [12] for both the L^2-norm error and penalty functions, and the derivatives are verified using finite difference.

Two cases of this optimization are shown with different numbers of FFD grid control points for comparison. The FFD box is made up of cubic B-Splines, and the control points of the FFD grid are evenly distributed in the x and y directions. The initial NACA 0012 airfoil B-spline is defined in space from $x = 0$ to $x = 1$.

In the first case shown in Fig. 21, the FFD box has a 4 by 4 grid of control points, which is the minimum number possible using cubic B-splines, and, hence, the optimization has $4 \times 4 \times 2 = 32$ degrees of freedom. The FFD surface is defined in the Cartesian range $[-0.1, 1.1] \times [-0.15, 0.15]$. This optimization completed after 244 objective function iterations, with a final objective function value of 4.0×10^{-7}. The optimizer terminated when the difference between the value of subsequent objective function evaluations was less than or equal to 1×10^{-17}. Despite a small number of degrees of freedom in the optimization, the L^2-norm difference between the morphing and target airfoil shapes is remarkably small.



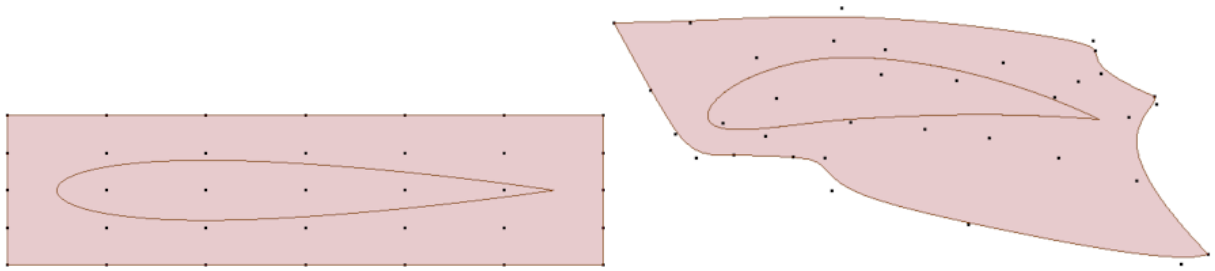
(a) Initial **(b) Final**

X-range	[-0.1, 1.1]
Y-range	[-0.15, 0.15]
Degrees of Freedom	32
Number of Iterations	244
Best Objective Function Value	4.0×10^{-7}

(c) Summary of the optimization

Fig. 21 A 4 by 4 cubic FFD grid optimization. The FFD grid surface geometry is in pink and the control point locations are the small black squares.

In the second case, shown in Fig. 22, the number of control points defining the FFD grid is increased such that there are 7 control points in the x-direction and 5 control points in the y-direction. There are therefore, $7 \times 5 \times 2 = 70$ degrees of freedom in the optimization. The FFD surface is defined by the same Cartesian range as in the first case, $[-0.1, 1.1] \times [-0.15, 0.15]$. This optimization completed after 682 objective function iterations. Since the number of degrees of freedom increased in comparison to the previous case, a larger number of objective function iterations is expected. However, the increased number of degrees of freedom also allows for about an order of magnitude further reduction in the objective function relative to the first case.



(a) Initial **(b) Final**

X-range	[-0.1, 1.1]
Y-range	[-0.15, 0.15]
Degrees of Freedom	70
Number of Iterations	682
Best Objective Function Value	1.2×10^{-8}

(c) Summary of the optimization

Fig. 22 A 7 by 5 cubic FFD grid optimization. The FFD grid surface geometry is in pink and the control point locations are the small black squares.

Figure 23 shows the convergence history for the optimization case above with the 7 by 5 FFD control point grid. This shows that most of the optimizations iterations are actually used trying to lower the objective function from $\sim 10^{-7}$ to $\sim 10^{-8}$. Assuming an objective of $\sim 10^{-7}$ is acceptable, the optimization would terminate in ~20 iterations. This shows the power of gradient based optimizations to rapidly converge.

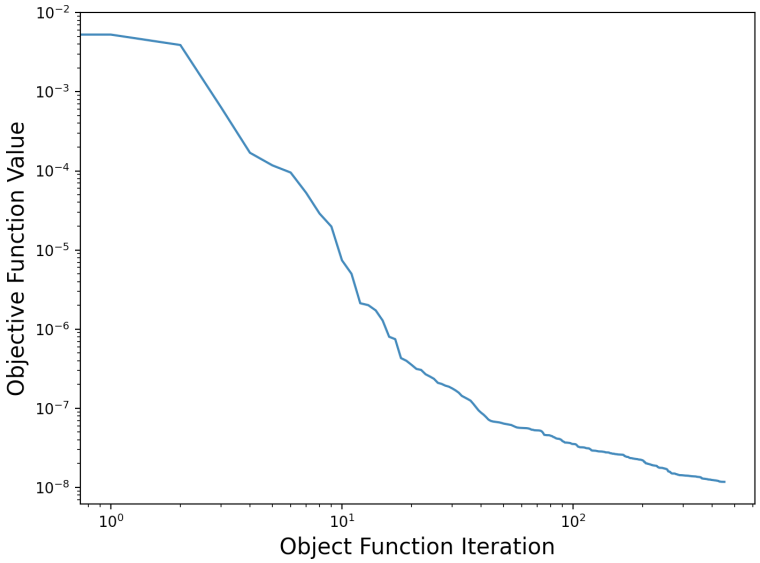


Fig. 23 Objective Function Convergence History

The results indicate this method of using FFD to perturb the control points of an underlying B-spline is accurate, while reducing the number of degrees of freedom when compared to using the B-spline control points themselves as design parameters. In fact, even using the minimum number of control points, as used in the cubic FFD box case shown in Fig. 21 where there were 16 total control points, this method is highly flexible in morphing a B-spline to accurately match a target shape.

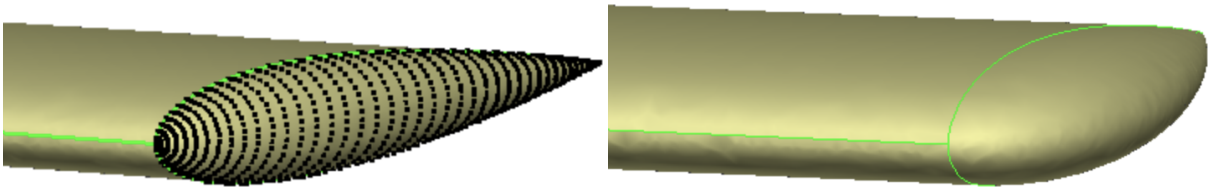
V. Optimization in 3D

A. Penalty Function

In 3-dimensions, the same penalty function and constants can be used, but instead of computing triangle areas formed by neighboring control points of the FFD grid, the FFD elements are tetrahedron volumes.

B. 3D L^2-difference Minimization Examples

To verify the FFD method of shape design in 3-dimensions, the same optimization is performed using NLopt, where the objective function is a 3-dimensional L^2-norm difference with the addition of the penalty function. However, instead of an airfoil, the geometry optimized is a wingtip. For this case, a B-spline surface defining the wingtip of a NACA 0012 airfoil rectangular wing is made using the `ESP blend` function [17]. Figure 24 depicts the initial and target wingtips. The wingtip is defined by 25 control points in the vertical direction, and 47 control points in the chord length direction, which is 1175 control points in total (i.e. 3525 degrees of freedom).



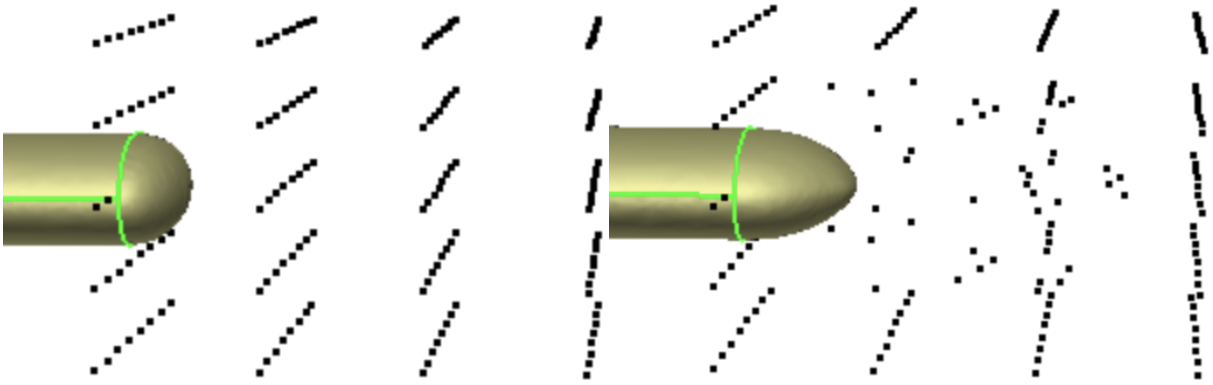
(a) Initial geometry for the optimization, which is a wingtip of radius one, shown with the control points defining it.

(b) Target geometry of the optimization, which is a wingtip of radius two.

Fig. 24 Wingtips used in optimization.

The FFD box encompasses only the wingtip surface. The outer frame of control points of the FFD box are fixed, so

only the wingtip surface is moved during the optimization, and this surface does not separate from the rest of the wing. The FFD grid has 8 control points in the x-direction, 5 control points in the y-direction, and 4 control points in the z-direction (which is the spanwise direction), as shown in Fig. 25a. The FFD box is made of linear B-splines, instead of the cubic B-splines used previously, and the final shape along with a summary of this optimizations is shown in Fig. 25. The optimization completes in 112 iterations, with a best objective function value of 9.6×10^{-6}.



(a) Initial **(b) Final**

X-range	[-0.25, 1.25]
Y-range	[-0.17, 0.17]
Z-range	[-0.01, 0.5]
Degrees of Freedom	480
Number of Iterations	112
Best Objective Function Value	9.6×10^{-6}

(c) Summary of the optimization

Fig. 25 NLopt wingtip optimization.

VI. Conclusion

A method of using free-form deformation to morph underlying B-spline of Brep geometry is presented. The method decreases the number of degrees of freedom in optimization when compared to directly manipulating the B-spline control point net, while allowing for a broader range of possible shapes. Self-intersections of geometry are prevented by formulating a penalty function which serves to prevent folds in the FFD control point net, and thus the underlying B-spline control point net. A two-step process of user-defined functions in the ESP framework generalizes the technique to any input body. The first UDF converts the necessary surfaces and edges to B-splines, and the second UDF step morphs the geometry and provides the sensitivities needed for optimization. The method is demonstrated on optimization cases in 2D and 3D which minimized the L^2-norm difference between a morphed and target shape.

Future work will use the generalized UDF functions to solve shape optimization problems driven with CFD analysis. The UDF can be connected with CFD analysis in either the Cart3D design framework, or using CAPS in ESP combined with an optimizer like OpenMDAO. Each design iteration the optimizer updates the design parameters, which are inputs to the UDF, and the UDF outputs the morphed geometry with tessellation sensitivities. This morphed geometry can then go through the analysis framework and optimization algorithm to complete the design cycle.

Acknowledgements

This work was funded by the EnCAPS project, AFRL Contract FA8650–20–2–2002: “EnCAPS: Enhanced Computational Aircraft Prototype Syntheses”, with Dr. Ryan Durscher as the Technical Monitor.

References

[1] Martin, M., Andres, E., Widhalm, M., Bitrian, P., and Lozano, C., "Non-uniform rational B-splines based aerodynamic shape design optimization with the DLR TAU code," *Proceedings of the Institution of Mechanical Engineers Part G Journal of Aerospace Engineering*, Vol. 226, 2012, pp. 1225–1242. https://doi.org/10.1177/0954410011421704.

[2] Kenway, G., Kennedy, G., and Martins, J., "A CAD-Free Approach to High-Fidelity Aerostructural Optimization," 2010. https://doi.org/10.2514/6.2010-9231.

[3] Hicken, J. E., and Zingg, D. W., "Aerodynamic Optimization Algorithm with Integrated Geometry Parameterization and Mesh Movement," *AIAA Journal*, Vol. 48, No. 2, 2010, pp. 400–413. https://doi.org/10.2514/1.44033.

[4] Haimes, R., and Dannenhoffer, J, "The Engineering Sketch Pad: A solid-modeling, feature-based, web-enabled system for building parametric geometry," 2013.

[5] Gomez, M., Galbraith, M. C., and Haimes, R., "On Analysis Driven Shape Design Using B-Splines," *AIAA 2022-1736*, 2022.

[6] Lee, C., Koo, D., and Zingg, D. W., "Comparison of B-Spline Surface and Free-Form Deformation Geometry Control for Aerodynamic Optimization," *AIAA Journal*, Vol. 55, No. 1, 2017, pp. 228–240. https://doi.org/10.2514/1.J055102.

[7] Gagnon, H., and Zingg, D. W., "Two-Level Free-Form and Axial Deformation for Exploratory Aerodynamic Shape Optimization," *AIAA Journal*, Vol. 53, No. 7, 2015, pp. 2015–2026. https://doi.org/10.2514/1.J053575.

[8] Sederberg, T. W., and Parry, S. R., "Free-Form Deformation of Solid Geometric Models," *SIGGRAPH Comput. Graph.*, Vol. 20, No. 4, 1986, p. 151–160. https://doi.org/10.1145/15886.15903, URL https://doi.org/10.1145/15886.15903.

[9] Anderson, G., Aftosmis, M., and Nemec, M., *Parametric Deformation of Discrete Geometry for Aerodynamic Shape Design*, ???? https://doi.org/10.2514/6.2012-965, URL https://arc.aiaa.org/doi/abs/10.2514/6.2012-965.

[10] Wang, X., and Qian, X., "An optimization approach for constructing trivariate B-spline solids," *Computer-Aided Design*, Vol. 46, 2014, pp. 179–191. https://doi.org/https://doi.org/10.1016/j.cad.2013.08.030, URL https://www.sciencedirect.com/science/article/pii/S001044851300170X, 2013 SIAM Conference on Geometric and Physical Modeling.

[11] Blanchard, P., Higham, D. J., and Higham, N. J., "Accurately computing the log-sum-exp and softmax functions," *IMA Journal of Numerical Analysis*, Vol. 41, No. 4, 2020, pp. 2311–2330. https://doi.org/10.1093/imanum/draa038, URL https://doi.org/10.1093/imanum/draa038.

[12] Marshall C. Galbraith and Steven R. Allmaras and David L. Darmofal, "A verification driven process for rapid development of CFD software," AIAA 2015-0818, January 2015.

[13] Johnson, S. G., "The NLopt nonlinear-optimization package," 2007.

[14] J. Nocedal, "Updating quasi-Newton matrices with limited storage," *Math. Comput.*, Vol. 35, 1980, pp. 773–782.

[15] Haimes, R., and Drela, M., "On The Construction of Aircraft Conceptual Geometry for High-Fidelity Analysis and Design," *AIAA Paper 2012–0683*, 2012.

[16] Charles L. Ladson, and Cuyler W. Brooks, Jr., and Acquilla S. Hill, and Darrell W. Sproles, "Computer Program to Obtain Ordinates for NACA Airfoils," *National Aeronautics and Space Administation*, 1996.

[17] Galbraith, M. C., and Haimes, R., "A Parametric G1-continuous Rounded Wing Tip Treatment for Preliminary Aircraft Design," *AIAA 2022-1734*, 2022.