Design Sensitivity Calculations Directly on CAD-based Geometry

John F. Dannenhoffer, III*
Aerospace Computational Methods Laboratory
Syracuse University, Syracuse, New York, 13244

Robert Haimes†
Aerospace Computational Design Laboratory
Massachusetts Institute of Technology, Cambridge, Massachusetts, 02139

Multi-disciplinary analysis and optimization (MDAO) has been a long-standing goal in the aerospace community. In order to employ MDAO effectively, one needs to be able to compute the sensitivity of the objective function with respect to the driving parameters in a robust and efficient manner. Over the past decade there have been considerable efforts towards the generation of "adjoint" versions of flow solvers in order to help in this process. Unfortunately, the corresponding efforts have not been expended in the geometry and grid generation processes, especially when the geometries are generated parametrically with a modern computer-aided design (CAD) or CAD-like system.

Contained herein is a pair of complementary techniques for computing configuration sensitivities directly on parametric, CAD-based geometries. One technique computes the configuration sensitivity analytically by differentiating the geometry-generating process; the other employs a new finite-difference technique that overcomes the difficulties previously encountered. Modifications to the Engineering Sketch Pad (`ESP`) (which is built on top of `OpenCSM`, `EGADS`, and `OpenCASCADE`) are described. Then the use of these configuration sensitivities in the computation of the sensitivity of grid-points is discussed. The results of these new techniques are shown on several configurations.

I. Introduction

Geometry management is an essential component for achieving multi-disciplinary analysis and optimization (MDAO), particularly if high-fidelity analysis tools are employed. When using a gradient-based optimization scheme, a key challenge is obtaining the necessary sensitivity information, that is, the derivative at any point on the body with respect one or more of the design parameters. There are three major methods for obtaining such derivatives:

- Analytic derivatives. This technique involves differentiating all operations within the CAD system analytically. Such derivatives are not generally susceptible to truncation errors, but can be problematic for complex construction operations, such as fillets and chamfers. Additionally, a description of the algorithm (or source code) is sometimes not available for the CAD system, so direct analytic differentiation may not be possible.

*Associate Professor, Mechanical and Aerospace Engineering, AIAA Associate Fellow.
†Principal Research Engineer, Department of Aeronautics & Astronautics, AIAA Member.

- Code differentiation. Here the source code for the construction operations is differentiated by a compiler-like function whose output is software (usually in the same language) that can then be compiled. Examples of these code processors include ADIFOR[1] and TAPENADE.[2] Again, this is only possible when the source code is available.

- Finite differences. In this approach, a base and a perturbed configuration are generated, and the differences between them are computed. While conceptually simple, special care must be taken to ensure the selection of an appropriate step-size (or perturbation), which is a balance between truncation and round-off errors. Another difficulty with this technique is that it is generally difficult to find points in the two configurations that are images of each other.

This paper describes recent efforts for obtaining design sensitivities in a robust and efficient manner. For analytic derivatives, enhancements of the `OpenCSM`[3] CAD system are described for calculating the sensitivity of surface points with respect to one or more driving parameters; this is implemented for most primitives and operations. Currently, where there is not an analytic solution, finite-differences are used. For finite differences, a new procedure is described that eliminates the issues associated with traditional finite differences.

Since frequently the goal is to compute the sensitivity of points in a grid that is generated for the configuration, a process for converting configuration sensitivities into grid sensitivities is also described.

Finally, all of these techniques are combined and demonstrated on a few sample configurations.

II. Analytic Sensitivities

In many modern computer-aided design (CAD) systems, users create complex solid models through the Boolean combination of a variety of primitives; `OpenCSM` is one such system that uses this constructive solid modeling technique.

An accurate and computationally-efficient method for finding the sensitivity of a configuration's Nodes, Edges, and Faces is to analytically differentiate the process by which a configuration is built. The accuracy comes from the fact that the derivatives in each step are computed exactly. The computational efficiency comes from the fact that this process does not require the configuration to be perturbed, thereby avoiding the excessive regeneration CPU time that is often consumed by the Boolean operations associated with complex configurations.

The only disadvantage with finding the sensitivities analytically is that one needs to have access to the processes used to generate the various operations; this includes the generation of the primitives as well as construction operations such as filleting. Analytical derivatives can also be used if the construction process for the primitives can be reverse engineered, as was done here in `OpenCSM`.

A. Constructive Solid Modeling process

The basic idea in the constructive solid modeling technique is to generate one or more primitives and then combine them, with Boolean operations, into the final configuration. This is best described by an example.

Consider the `OpenCSM` description file:

```
      # bolt example

      # design parameters
 1:   despmtr   Thead     1.00   # thickness of head
 2:   despmtr   Whead     3.00   # width     of head
 3:   despmtr   Fhead     0.50   # fraction  of head that is flat

 4:   despmtr   Dslot     0.75   # depth of slot
 5:   despmtr   Wslot     0.25   # width of slot
```

```
 6:   despmtr   Lshaft    4.00   # length   of shaft
 7:   despmtr   Dshaft    1.00   # diameter of shaft

 8:   despmtr   sfact     0.50   # overall scale factor

      # head
 9:   box       0      -Whead/2  -Whead/2  Thead   Whead     Whead
10:   rotatex   90  0  0
11:   box       0      -Whead/2  -Whead/2  Thead   Whead     Whead
12:   rotatex   45  0  0
13:   intersect

14:   set       Rhead  (Whead^2/4+(1-Fhead)^2*Thead^2)/(2*Thead*(1-Fhead))

15:   sphere    0             0   0    Rhead
16:   translate Thead-Rhead   0   0
17:   intersect

      # slot
18:   box       Thead-Dslot  -Wslot/2   -Whead   2*Thead   Wslot    2*Whead
19:   subtract

      # shaft
20:   cylinder  -Lshaft   0   0   0   0   0   Dshaft/2
21:   union

22:   scale     sfact

23:   end
```

This yields the bolt shown in Fig. 1. The file begins with the definition of eight design parameters (in lines 1 through 8). The head of the bolt starts with the generation of two identical boxes (lines 9 and 11) and then rotating them about the x-axis (in lines 10 and 12). The Boolean intersection (line 13) of these two boxes yields an octagonally-shaped outline of the head. This is then intersected with a sphere (whose radius is computed in line 14 and which is generated in line 15), translated (in line 16) and then intersected with the octagon (in line 17). Then the box that represents the slot (which is generated in line 18) is subtracted (in line 19) from the head. Finally a cylinder that represents the shaft is generated in line 20 and then unioned with the head in line 21. The whole bolt is then scaled in line 22.

The feature tree form of the description file is given in Fig. 2. Note that the primitives (`box`, `sphere`, and `cylinder`) do not have parents, but have one child; transformations (such as `translate` and `rotatex`) each have one parent and one child; the Boolean operations (`union`, `intersect`, and `subtract`) each have two parents and one child. The last element in the tree, `scale`, does not have any children and is called the tree's root.

B. Overview of process for computing analytic sensitivities

The basic process for finding sensitivities analytically involves the following steps:

- differentiate the expressions that are used to compute the arguments to the various operations
- for each Face in the configuration
  - determine the primitive that created the Face

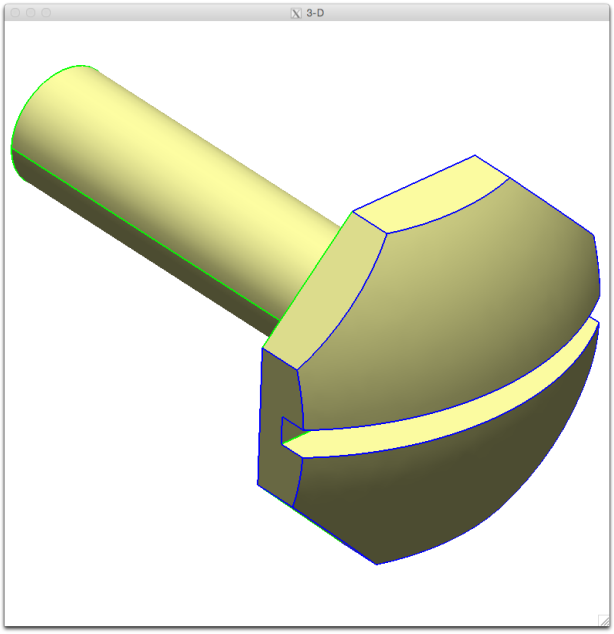


Figure 1. Sample bolt configuration

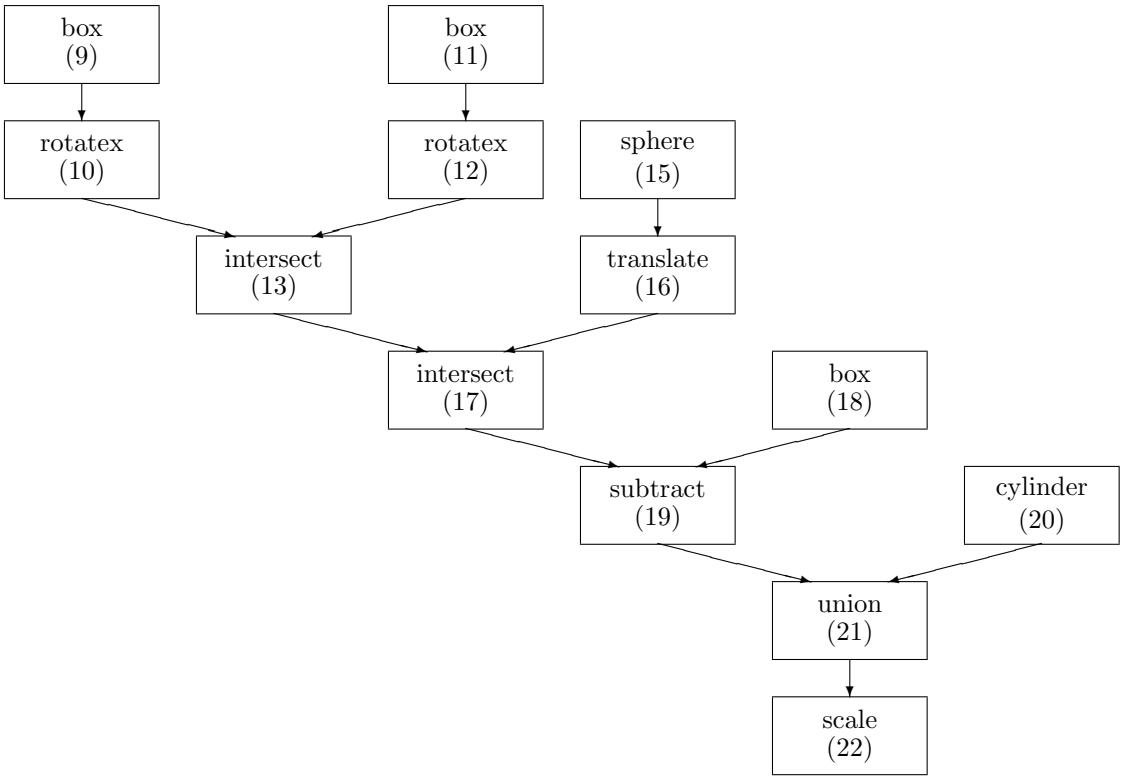


Figure 2. Feature tree associated with bolt configuration

  - differentiate the functions used to generate the Faces in the original primitive
  - apply the appropriate transformations to the sensitivities
  - compute the component of the sensitivity that is normal to the Face
- for each Edge in the configuration
  - compute the sensitivities for the two Faces that are incident to the Edge
  - find an Edge sensitivity that is consistent with the two Face sensitivities and which has a zero component in a direction that is tangent to the Edge
- for each Node in the configuration
  - compute the sensitivities for all the Edges that are incident to the Node
  - find a Node sensitivity that is consistent with the Edge sensitivities

C. Argument sensitivities

The first step in the process involves the differentiation of the expressions that are used as arguments to the various commands in the `OpenCSM` description file.

As an example, assume that one wants to compute the sensitivity of this configuration with respect to the thickness of the head, or $P \equiv \text{Thead}$. The first argument of the `translate` command (line 16) is $A_1 = \text{Thead} - \text{Rhead}$. The derivative of this is simply

$$\frac{dA_1}{dP} = 1 - \frac{d(\text{Rhead})}{dP} = \frac{1 + \text{Fhead}}{2} + \frac{\text{Whead}^2}{8\text{Thead}^2(1 - \text{Fhead})}$$

when the derivative of the `set` statement in line 14 is used. The other arguments are treated in a similar way.

D. Face sensitivities

As mentioned above, when computing analytic sensitivities on Faces, the first step is to determine the primitive that originally created that Face. For example, the Face on the side of the bolt head that is cut by the slot was originally generated by the `box` command in line 9. Fortunately determining this is trivial in `OpenCSM/EGADS`[4] since the Faces are attributed when they are first created, and `OpenCSM/EGADS` ensures that these attributes are properly tracked throughout the generation process.

Once the correspondence between the root (final) and primitive (original) Faces is known, the point at which the sensitivity is desired, say $\vec{x}_{\text{root}}$ is transformed into $\vec{x}_{\text{prim}}$ by walking up the feature tree from the root to the element that created the Face, and applying the inverse of the transformations that were traversed. For example, for points that where originally created by the `sphere` command (in line 15), one would traverse the `scale` command (line 22), the `union` (line 21), the `subtract` (line 19), the `intersect` (line 17), and the `translate` (line 16); hence the $\vec{x}_{\text{prim}}$ would be modified by the inverse of the `scale` and the inverse of the `translate`.

The next step is to find the derivatives of the functions used to originally generate the primitive. Since `OpenCSM/EGADS` is built upon `OpenCASCADE`, and since the `OpenCASCADE` code is quite large and complex, differentiating it directly was not feasible. However, it was a rather simple process to reverse-engineer the algorithm used and then differentiate the reverse-engineered algorithm.

A box is defined by six parameters: the base point $\vec{x_0}$ and a size $\vec{S}$. The six Faces are planes at $\vec{x} = \vec{x_0}$ and $\vec{x} = \vec{x_0} + \vec{S}$.

Once the box is created, the sensitivity of any point on the surface of the box with respect to a design parameter P is given by

$$\left(\frac{\partial \vec{x}}{\partial P}\right)_{\text{prim}} = \frac{\partial \vec{x_0}}{\partial P} + \frac{\partial \vec{S}}{\partial P}\left(\frac{\vec{x}_{\text{prim}} - \vec{x_0}}{\vec{S}}\right)$$

In a similar way, a sphere is defined by four parameters: the center $\overrightarrow{x_0}$ and the radius r. Once a sphere is created, the sensitivity of any point on the surface of the sphere is simply

$$\left(\frac{\partial \vec{x}}{\partial P}\right)_{\text{prim}} = \frac{\partial \overrightarrow{x_0}}{\partial P} + \frac{\partial r}{\partial P}\left(\frac{\vec{x}_{\text{prim}} - \overrightarrow{x_0}}{r}\right)$$

`OpenCSM` has similar sensitivity calculations for a `cylinder`, `cone`, `torus`, `extrude`, `revolve`, and `rule`; the latter three use sensitivities computed on the sketches that are employed by the `extrude`, `revolve`, or `rule`.

Once the sensitivity is computed on the primitive, it is transformed by traversing the feature tree from the primitive down to the root. For example, the sensitivity computed on the sphere is modified by the `translate` (in line 16) by

$$\left(\frac{\partial \vec{x}}{\partial P}\right)_{16} = \left(\frac{\partial \vec{x}}{\partial P}\right)_{\text{prim}} + \frac{d\overrightarrow{x_0}}{dP}$$

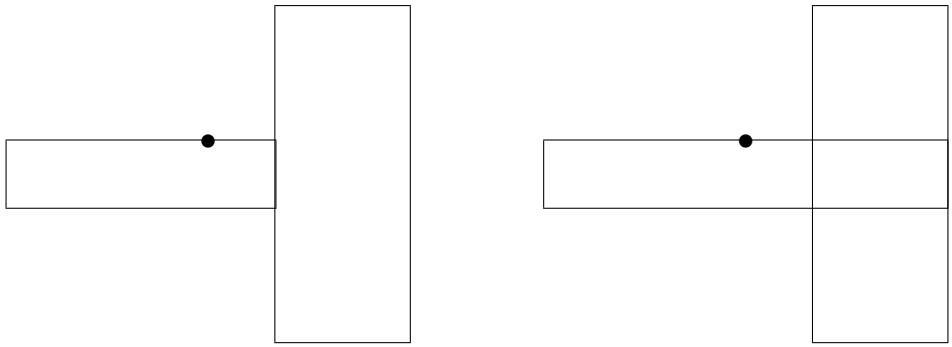
where $\overrightarrow{x_0}$ is the translation distance and then by the `scale` (in line 22) by

$$\left(\frac{\partial \vec{x}}{\partial P}\right)_{root} = S\left(\frac{\partial \vec{x}}{\partial P}\right)_{16} + \frac{\partial S}{\partial P}\vec{x}_{16}$$

where S is the scalar scale factor.

Finally, there is the question as to how the sensitivity of the Face should be expressed. Recall that we are asking at the sensitivity at some $\vec{x}$; does this mean a specific location in space, or does it mean a fractional distance along the body?

On the one hand, if one interprets $\vec{x}$ to be a fractional distance along the body, then the body's original size is used in the sensitivity calculation, even if the body is trimmed by some Boolean operation. For example, consider the bodies in Fig. 3, where the horizontal rectangle is "trimmed" by the union with the vertical rectangle. In part (a), the horizontal rectangle was constructed to abut the vertical rectangle, and so the indicated points is at about the 75% position. In part (b), the horizontal rectangle was constructed to include the whole width of the vertical rectangle, and so the indicated point is at about the 50% position. Given the equations above, one would compute very different sensitivities — even though the final bodies (in (a) and (b)) are identical; this is clearly not a desired result.



(a) small horizontal rectangle **(b) large horizontal rectangle**

Figure 3. Two different constructions of the same shape.

On the other hand, if one interprets $\vec{x}$ to mean a specific position in space, then the sensitivity at the surface is only a function of the motion of the surface normal to the surface. When interpreted this way, the results for (a) and (b) are identical. In fact, only vertical movements of the horizontal rectangle (or changes in the horizontal rectangle's height) would yield a non-zero sensitivity at the indicated point.

In everything that follows, the sensitivity of a point on a Face (with respect to some driving parameter, P) is taken as the normal component of the sensitivity computed above. In other words:

$$\frac{\partial w}{\partial P} \equiv \frac{\partial \vec{x}}{\partial P} \bullet \vec{n}$$

where $\vec{n}$ is the local outward normal of the Face. Positive sensitivities imply that the body grows locally whereas negative sensitivities indicate that the body shrinks locally.

E. Edge sensitivities

In any given configuration, some of the Edges come from part of the creation of a primitive (for example, there are twelve Edges generated as part of `box` construction), whereas others were created from the intersection curve that results from a Boolean operation.

For those Edges that were created during the construction of a primitive, similar operations as were described above for Face sensitivities are used. Once the 3D sensitivity is computed, only the components normal to the Edge are returned. (The reason for this is the same as described above.)

For those Edges that arise from a Boolean operation, the sensitivities of points along Edges can be obtained from the sensitivities on the two Faces that are incident to the Edge. This is similar to the approach taken by Lazzara.[5]

Consider an Edge that bounds two Faces, called "left" and "right". Immediately adjacent to a specified point on the Edge, evaluation of the "left" Face gives $(\partial w/\partial P)_{\text{left}}$, which is pointed in the $\vec{n}_{\text{left}}$ direction; similar evaluation on the "right" Face gives $(\partial w/\partial P)_{\text{right}}$ and $\vec{n}_{\text{right}}$.

The key now is to find a Edge sensitivity vector, $(\partial \vec{x}/\partial P)_{\text{edge}}$ that is consistent with the two Face sensitivities and which is locally normal to the Edge. This is obtained by solving the matrix equation:

$$\begin{bmatrix} n_{x,\text{left}} & n_{y,\text{left}} & n_{z,\text{left}} \\ n_{x,\text{right}} & n_{y,\text{right}} & n_{z,\text{right}} \\ t_{x,\text{edge}} & t_{y,\text{edge}} & t_{z,\text{edge}} \end{bmatrix} \begin{bmatrix} (\partial x/\partial P)_{\text{edge}} \\ (\partial y/\partial P)_{\text{edge}} \\ (\partial z/\partial P)_{\text{edge}} \end{bmatrix} = \begin{bmatrix} (\partial w/\partial P)_{\text{left}} \\ (\partial w/\partial P)_{\text{right}} \\ 0 \end{bmatrix}$$

for $(\partial \vec{x}/\partial P)_{\text{edge}}$. Here, $n_{x,\text{left}}$ means the x component of $\vec{n}_{\text{left}}$ and $t_{x,\text{edge}}$ means the x component of the tangent vector along the Edge (that is, dx/dt along the Edge, where t is the Edge parametric coordinate).

As with the Face sensitivity, the Edge sensitivity only gives changes normal to the Edge; the component of the sensitivity along the Edge is automatically set to zero.

F. Node sensitivities

As with Edges, some Nodes in a configuration come directly from the construction of a primitive (for example the eight Nodes at the corners of a `box`); these are treated in a way analogous to the Face sensitivities computed above, with the exception that the returned sensitivity is truly a 3D vector, $\partial \vec{x}/\partial P$ (since there no ambiguity associated with the interpretation of $\vec{x}$ for Nodes).

The sensitivities associated with Nodes that are a result of a Boolean operation are treated in a similar manner to Edge sensitivities described above. Specifically, Nodes have three or more incident Edges. The Node sensitivity is thus the sensitivity that is consistent with the sensitivities of the incident Edges. This is done by solving the equation

$$\begin{bmatrix} \vec{t}_1 \bullet \vec{t}_1 & -\vec{t}_1 \bullet \vec{t}_2 \\ -\vec{t}_1 \bullet \vec{t}_2 & \vec{t}_2 \bullet \vec{t}_2 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} ((\partial \vec{x}/\partial P)_2 - (\partial \vec{x}/\partial P)_1) \bullet \vec{t}_1 \\ ((\partial \vec{x}/\partial P)_1 - (\partial \vec{x}/\partial P)_2) \bullet \vec{t}_2 \end{bmatrix}$$

for A and B. Here, $\vec{t}_1$ is the tangent vector of the first Edge and $\vec{t}_2$ is the tangent of the second Edge. Even though Nodes have three or more incident Edges, the two Edges used here are the two that are the least parallel to each other at the Node.

The final Node sensitivity is then given by

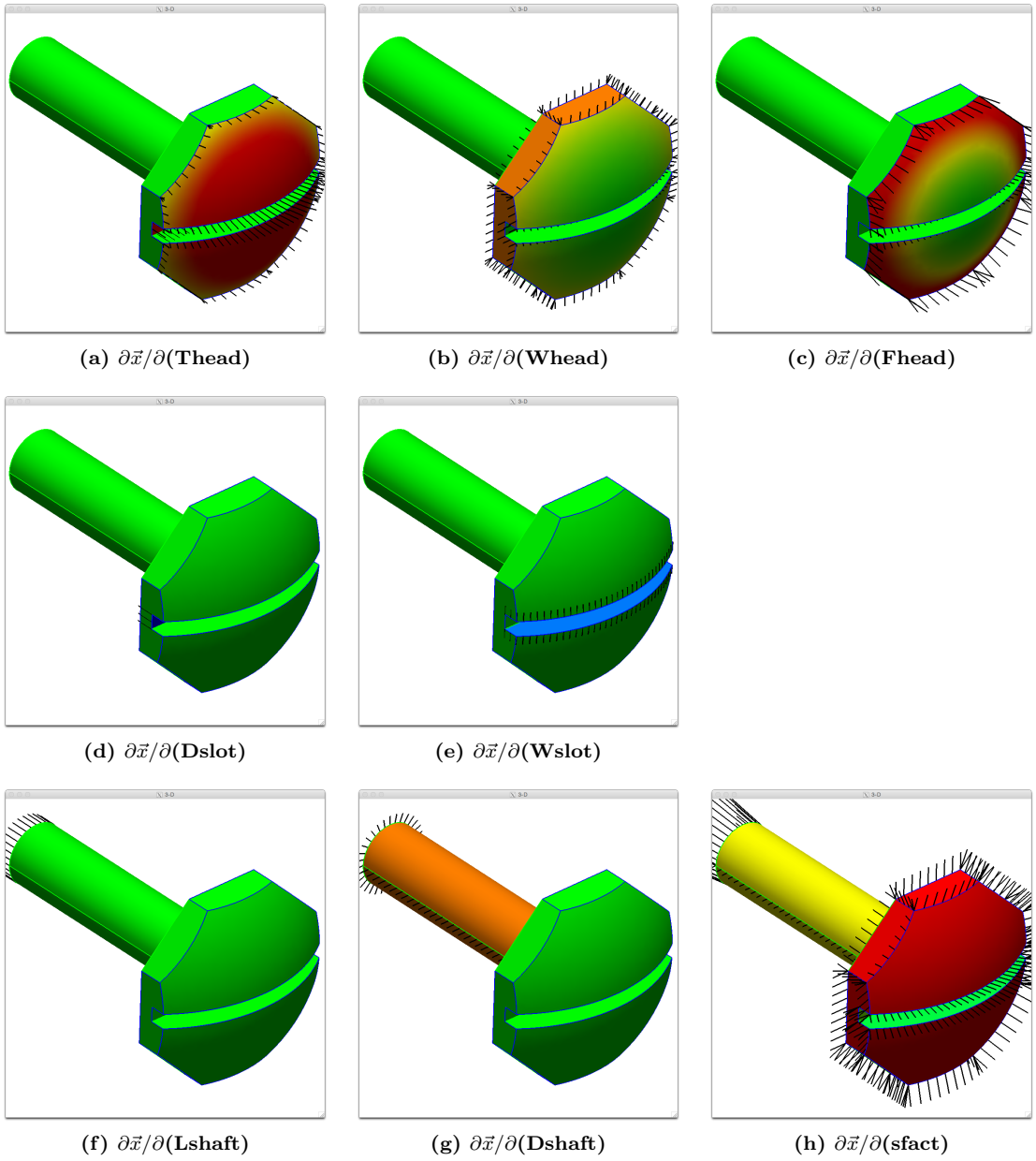
$$\left(\frac{\partial \vec{x}}{\partial P}\right)_{\text{node}} = \left(\frac{\partial \vec{x}}{\partial P}\right)_{\text{edge1}} + A\left(\frac{\partial \vec{x}}{\partial t}\right)_{\text{edge1}}$$

Note that since a Node location is uniquely specified (that is, it does not have an Edge parametric coordinate t or a Face parametric coordinate $\vec{u}$), it is a true vector in three dimensions.

G. Example

The results of the sensitivity computations for the bolt configuration, as described above, are shown in Fig. 4. Each of the panels in the figure depict the sensitivity of the configuration with respect to one of the eight driving (design) parameters. The color on the Faces show the normal Face sensitivities (with green being zero). At several locations along each Edge, small tufts show the "normal" sensitivity associated with the Edge. Also shown in the figure are tufts at the Nodes.

Note that there are actually four (or more) tufts emanating from each Node. This is because one of the tufts is associated with the Node, and the others are the normal components associated with each incident Edge. When employing the sensitivity information at these locations, it is appropriate to use the Node information.



(a) $\partial\vec{x}/\partial$(**Thead**) (b) $\partial\vec{x}/\partial$(**Whead**) (c) $\partial\vec{x}/\partial$(**Fhead**)

(d) $\partial\vec{x}/\partial$(**Dslot**) (e) $\partial\vec{x}/\partial$(**Wslot**)

(f) $\partial\vec{x}/\partial$(**Lshaft**) (g) $\partial\vec{x}/\partial$(**Dshaft**) (h) $\partial\vec{x}/\partial$(**sfact**)

Figure 4. Face, Edge, and Node sensitivities with respect to each driving parameter. Face sensitivities shown as green= 0, red> 0, and blue< 0. Edge and Node sensitivities shown as tufts.

III. Finite-differenced Sensitivities

Any approach to using finite-differencing in the calculation of parametric sensitivities first perturbs the parameter (or parameters) of interest and then regenerates the geometry. This requires being able to perform point associations; that is, given a point on the source, where is the corresponding position on the perturbed body? To be able to answer this questions one needs to be able to map the boundary representation (BRep) entities and mapping requires that the BRep topologies match. If a check indicates that the source and newly perturbed model have the same topology, then the process can continue. Otherwise, it is common to reduce the perturbation size and rebuild the model until either the size gets as small as the model tolerance or the perturbed model finally displays the same BRep topology.

A. The traditional finite-difference scheme for geometry

When using the traditional finite-difference approach, a point perturbation can be computed by knowing what BRep Face (and $\vec{u} = [u, v]^T$ coordinates) the point resides on in the source body. Using the $\vec{u}$ bounding box of the source Face gives a relative position in that box. The corresponding Face is selected in the perturbed body and the relative position is used with the $\vec{u}$ bounding box for the perturbed Face to evaluate the perturbed $\vec{x} = [x, y, z]^T$. The difference between the initial point and the evaluated location (divided by the perturbation size) provides the parametric derivative at that location. Examples of the use of this finite difference method can be found in use with Cart3D[6–8] and other CAD-based adjoint-enabled design optimization frameworks.[9] Note that this scheme has the following assumptions:

Assumption 1: Any dilation, contraction or other change to the Face is related to the geometry kernel's reported $\vec{u}$ bounding box for both the source and the corresponding perturbed Face. And any change to the $\vec{u}_{pert}$ bounding box implies point movement over the entire Face.

Assumption 2: Linear adjustments in the Face's $\vec{u}$ is related to arc-length in $\vec{x}$. Simply stated, the geometry's parametrization can be used to map point movement.

Though this finite-difference scheme has worked for many researchers, it does have its problems and therefore limitations. The selection of design parameters when the Cart3D framework is employed is done carefully and judicially so that the problems posed by the assumptions listed above are avoided. Also, Brock[9] initially selected wing twist as an adjustable parameter until it was noticed that the surface parametrization at the tip of the wing was breaking Assumption 2 (the Face parametrization was not rotating with the wing – see below). The result was that this parameter was removed from the design space for the design study. The traditional scheme is therefore not a general solution and needs improvement so that finite differences can be used without regard to the parametrization and construction of the geometry.

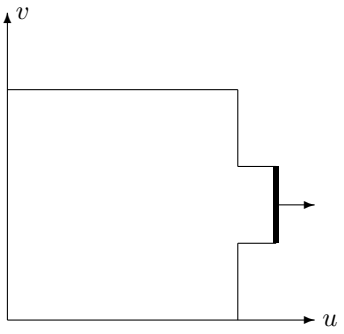


Figure 5. Local adjustment of a Face in parameter space.

Assumption 1 issues:

- The $\vec{u}$ parametrization adjusts with perturbations of the model. Consider the situation where a design parameter controls an angle (for example the twist of a wing) and the object is cut (or capped) via a planar surface. For this finite-difference scheme to work it requires the parametrization of planar surface to respond to the angle change. If the plane is defined in a static manner, then the plane's parametrization will be invariant to the angle parameter. After a perturbation and rebuild, the $\vec{u}$ box will differ and see changes do to the rotation (which would not be seen if the parametrization also rotated). Also because the parametrization did not change, any finite-differences will display large-scale movement.
- Changes to the $\vec{u}$ box effect the entire Face. Consider the simple example seen in Fig. 5 where only a small protuberance has been changed during the perturbation process. In this case most of the Face should be invariant to that small Edge's perturbation, but the $\vec{u}_{pert}$ box has changed and will therefore result in point movement throughout the entire Face.

Assumption 2 notes:

- The notion that geometric parameter space adjustments approach/approximate changes in arc-length works out well for all of the analytic geometric descriptions supported by `EGADS`. These include, planar, spherical, cylindrical, conical and toroidal shapes (where the *circular* parameter(s) are based on radians). Bezier shapes also provide a parametric model that does not seem to break this assumption.
- The parametrization of B-Spline and NURBS geometry is based on the knot sequence alone. In many cases this is defined when fitting data to produce the geometry and tends to relate to arc-length (computed in a piecewise manner), but only loosely. For the parameter spaces to match up, the knot sequences must be the same for both entities. If there are any differences then there will be a problem! A minor difference in the sequence can produce rather non-linear errors in the mapping. Therefore special attention is required for this type of geometry in order that the finite-difference calculation provides meaningful results. An obvious requirement may be that the knots match between the source and perturbed geometry. This is difficult in a construction system that does not provide methods to do so.

B. A new finite-difference scheme for geometry

The problems discussed above have resulted in a rethinking of the traditional approach to finite-differencing geometry where better and more robust control over the mappings is achievable. The first fundamental change is to remove the use of the $\vec{u}$ bounding box from the scheme. Clearly another method to perform the perturbed mappings is needed. The obvious candidate for more accurate representation of the bounds of the Face is the BRep Loops (collections of Edges). In fact this is the definition of how the Faces are trimmed and is the appropriate place to track any changing shape of the bounds of the Face.

It is possible to use the Loops because there is topological matching between the source and perturbed bodies, so that we can then compare the Edges contained in the Loop(s) directly. But the question arises: how can the mapping be performed with the actual trimming of the Face? The answer is quite simple: use a triangulation. This has many advantages, including being able to localize perturbations, Barycentric coordinates can be used (which are perfectly valid in linear mappings) and provides a scheme that can be rotationally invariant. The clear disadvantage is requiring a tessellation of the object (the additional memory and time to generate the triangulation on the source) to be finite-differenced.

The requirement of a triangulation of the body is, in fact, not onerous. Seeing that finite-differencing is performed at specific locations on the source geometry and these locations either come from a triangulation of the body or a Face-based triangulation can be derived from these positions, the triangulation is NOT an additional requirement but part of the scheme anyway.

The *Face to Face* mapping is derived from the following algorithm:

1. Use the `EGADS` triangulator to provide the source tessellation. In this scheme:
   - The Edges, found in the bounding Loop(s), are discretized first.
   - A coarse $\vec{u}$ tessellation which only uses the bounding vertices (from the Edges) is the next stage in the `EGADS` Face triangulator which uses a similar iterative scheme as described in Haimes & Aftosmis.[10]
   - This coarse triangulation is then saved away with the `EGADS` tessellation Object.
2. From perturbed body – set the Nodes.
3. Discretize the perturbed Edges based on the relative positions in the source (using the Edge limits t_{min} and t_{max} from each). The Edge in the perturbed body is evaluated at t_{pert} to set $\vec{x}_{pert}$ for the vertex at the bounds.

   If the underlying curves for any Edges in the Loop(s) are B-Splines/NURBS and the knot sequence is not the same between source and perturbed objects, an additional correction is required. In this case `OpenCASCADE` functions are used to determine the arc-length relative position of t_{src} for the Edge. The inverse function is used to get t_{pert}, that is: return the t that gives the specified relative arc-length along the perturbed Edge.
4. Use the `EGADS` *PCurve* function to map t_{pert} to $\vec{u}_{pert}$. This is done for each Face in the perturbed model setting up the *frame* of the bounds for that Face.
5. Any location in the source can be found in the perturbed body by first finding the enclosing coarse triangle and its Barycentric coordinates for $\vec{u}_{src}$. $\vec{u}_{pert}$ at the selected position is computed by using $\vec{u}_{pert}$ of the vertices of the enclosing triangle and the Barycentric coordinates. $\vec{u}_{pert}$ is evaluated to get the $\vec{x}_{pert}$.

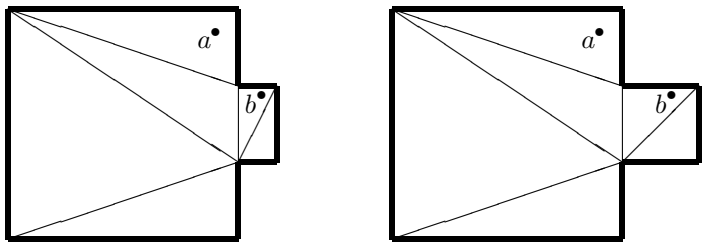


Figure 6. Coarse triangle-based mapping for finite-differencing.

Fig. 6 depicts the use of the tessellation to perform the mapping required for point tracking as seen in the situation highlighted by Fig. 5. The point labeled a does not move and the b point moves with the changing protuberance.

There is still a problem when the geometry of the corresponding Faces is either a B-Spline or NURBS surface and the u and/or v knot sequences do not match. Again, under these circumstances, the geometric parameterizations are not the same and cannot be used directly in the mapping. There is no simple fix (as used for the Edge correction). This arc-length correction could be used by selecting the appropriate isocline curve but is would be very expensive (a different curve needs to be constructed for each point). Also this correction can only work if either the u or the v knots differ, but not both (which is because it becomes undetermined which isocline to select).

It has been found that knot insertion in the perturbed Face works out well. Any source knot, not matching a knot in the perturbed Face, is inserted into the B-Spline surface in both u and v. Knot insertion is designed to leave the shape of the surface unchanged. The end result is not an identical knot sequences, but the surface parameterizations display similar arc-length representations. Note that knot removal does

change the surface; therefore its use generates large finite-difference errors, and it cannot be used to attempt to match the knot sequences exactly.

C. Examples

Two separate cases are used here to show the results of computing sensitivities via finite differences. The first case required finite differences due to the presence of the fillets and chamfers. The second case required finite differences because of the `blend` used to create the fuselage; work is underway to differentiate the `blend` function code via an automatic code differentiator, such as ADIFOR or TAPENADE.

Fig. 7 shows the surface sensitivities associated with changing the box length. Notice that the fillets associated with the box's left surface have a "graded" normal sensitivity. Also note the non-zero sensitivities on the surfaces of the two holes in the box; these sensitivities are non-zero because the original configuration description has the holes equally-spaced in the box (and changing the box's length changes the positions of the holes and hence their surface sensitivities).

Fig. 8 shows the surface sensitivities associated with changing the holes' radii. In this case, all the holes' surface sensitivities are negative (since the surface moves outward when the radius is increased but the velocity is measured based upon the outward normal — which is pointed toward the center of the hole). The points on the associated chamfers have sensitivities too because of the radius change, but their sensitivities are smaller than the hole's sensitivities because the chamfer moves away from the holes' centerlines, but the chamfer's outward normal is oblique to the direction of the motion.

The second case corresponds to a wing/body case. Here the wing is defined by a `ruled` surface (for which analytical derivatives are available) but the fuselage is comprised of a `blended` body, which is not differentiated at this time. Hence, the whole configuration was computed via finite differences.

Fig. 9 displays screen shots from `ESP`[11] after the construction of multi-fidelity geometry. In this case there are actually 3 `EGADS` bodies: a *SHEETBODY* of the Mid-Surface Aero (MSA), a Built-up Element Model (BEM) (also a *SHEETBODY*) and the Outer Mold-Line (OML) represented as an `EGADS` *SOLIDBODY*. In the left hand picture both the BEM and OML are not visible. In the right-hand side only the OML is depicted. The design parameter **camber** is selected and the design derivatives are displayed in color.

Fig. 10 displays the same geometry as seen if Fig. 9 except that the design parameter highlighted is **thickness**. Because the MSA is a simple surface (without any thickness), the design sensitivity is obviously *zero* and therefore displayed in green.

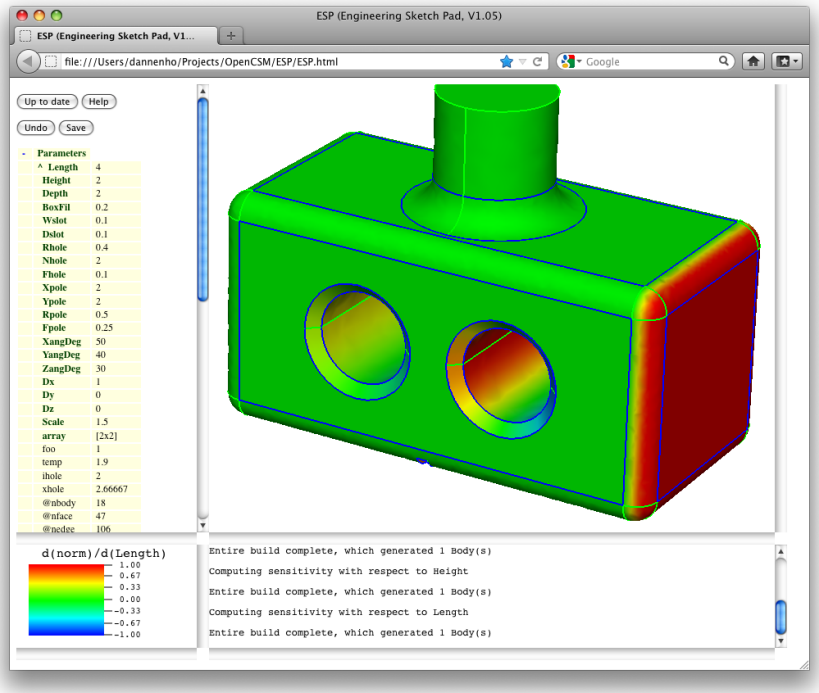


Figure 7. Velocity of surface points for a change in the box's length.

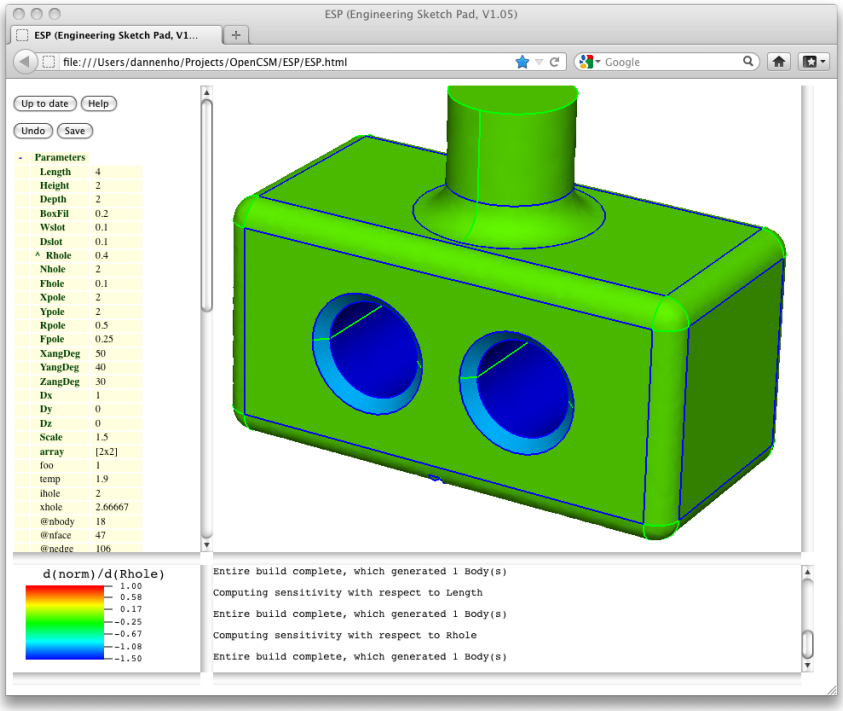
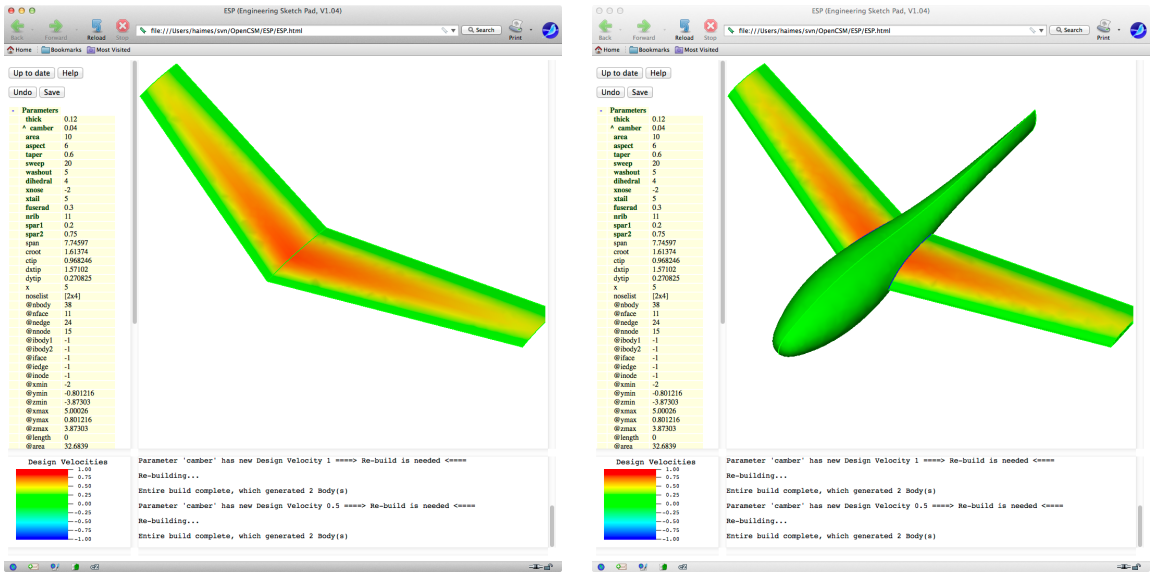
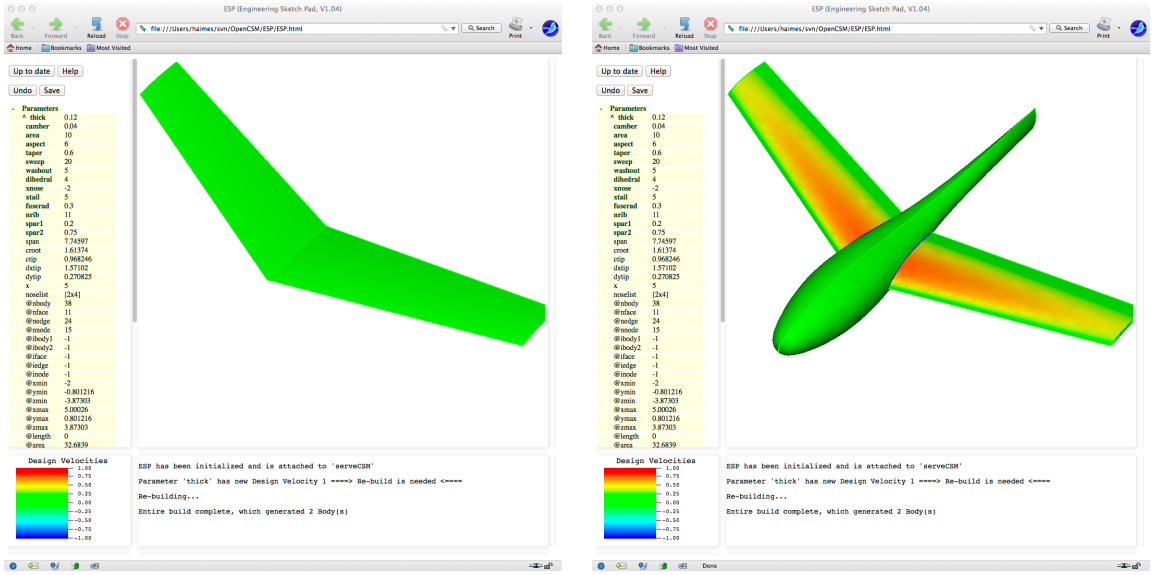


Figure 8. Velocity of surface points for a change in the holes' radii.



(a) Mid-surface aero (MSA) (b) Outer mold line (OML)

Figure 9. Screen shots showing design sensitivities from within ESP. Design parameter: camber.



(a) Mid-surface aero (MSA) (b) Outer mold line (OML)

Figure 10. Design parameter: thickness. Note that since the MSA is a simple surface, it has no thickness.

IV. Application to Grid Sensitivities

Once the configuration's sensitivities are known on the Node, Edges, and Faces, it is common to propagate them to the sensitivities of grid points that are created on the configuration. This latter step is needed if one ultimately wants to get the sensitivity of some objective function (for example, lift-to-drag ratio) as a function of the design (driving) parameters of the CAD-based model. This section addresses the computation of grid sensitivities from configuration sensitivities that are either computed analytically or via finite differences.

In this section, a reasonably simple algebraic grid generator will be used for illustrative purposes; this was done to highlight the essential operations needed without getting distracted by the details of the grid generator. The same process for finding grid sensitivities from configuration sensitivities can be applied to any other structured or unstructured grid generator if the governing equations are properly differentiated.

A. Grid Generation

As mentioned above, the grid generation process used here is a simple algebraic transfinite interpolation technique, which is applied independently to each four-sided Face of a configuration.

Each Face is described by Face parametric coordinates $\vec{u} = [u, v]^T$ and associated physical coordinates $\vec{x} = [x, y, z]^T$. There exists functions (in `EGADS`) that return the physical coordinates given the Face parametric coordinates, as in

$$\vec{x} = \mathbf{face}(\vec{u})$$

as well as the Face slopes

$$\frac{\partial \vec{x}}{\partial \vec{u}} = \mathbf{faceDeriv}(\vec{u})$$

On the Edges that surround the Face, they are each described in terms of its own Edge parametric coordinate t, which is bounded by t_{beg} and t_{end}. Again, there are functions in `EGADS` to return the Face's parametric coordinate:

$$\vec{u} = \mathbf{edge}(t)$$

as well as the Edge slopes:

$$\frac{d\vec{u}}{dt} = \mathbf{edgeDeriv}(t)$$

1. *Nodes*

For the grid generation, no explicit processes are needed at the Nodes (corners of the grid); instead the Edge generation processes are employed that their extrema.

2. *Edges*

Along each Edge, the points are obtained by using equally spaced points in t, finding the associated $\vec{u}$, and finally the corresponding physical coordinates $\vec{x}$.

For example, consider the "south" Edge of a Face, which corresponds to the $j = 0$ grid line. Define the fractional distance along the Edge as

$$f_i = \frac{i}{I-1} \qquad i = 0, \ldots, I-1 \tag{1}$$

The equally-spaced t's are then defined by

$$t_i = (1 - f_i)t_{\text{beg}} + (f_i)t_{\text{end}} \tag{2}$$

Edge and Face evaluation functions are the used to find the Edge parametric and physical coordinates, as in

$$\vec{u}_{i,0} = \mathbf{edge}(t_i)$$

and

$$\vec{x}_{i,0} = \mathbf{face}(\vec{u}_{i,0})$$

3. Face

On the Face, the interior grid points are generated via transfinite interpolation[12] in terms of the Face parametric coordinates $\vec{u}$. First we define the fractional distances in each coordinate direction:

$$\begin{aligned} f_i &= \frac{i}{I-1} \qquad & i = 1, \ldots, I-2 \\ g_j &= \frac{j}{J-1} \qquad & j = 1, \ldots, J-2 \end{aligned} \tag{3}$$

and then obtain the interior points by

$$\begin{aligned} \vec{u}_{i,j} &= (1-f_i)\vec{u}_{0,j} + (f_i)\vec{u}_{I-1,j} + (1-g_j)\vec{u}_{i,0} + (g_j)\vec{u}_{i,J-1} \\ &- (1-f_i)(1-g_j)\vec{u}_{0,0} - (f_i)(1-g_j)\vec{u}_{I-1,0} - (1-f_i)(g_j)\vec{u}_{0,J-1} - (f_i)(g_j)\vec{u}_{I-1,J-1} \end{aligned} \tag{4}$$

Finally, the physical coordinates of the interior points are obtained by using the surface evaluation routine, such as

$$\vec{x}_{i,j} = \textbf{face}(\vec{u}_{i,j})$$

B. Grid Sensitivity

In this section, the sensitivity of each grid point $\vec{x}_{i,j}$, with respect to some design parameter, P, are found. This is done using the Edge and Face evaluation functions described above, as well as three new functions. The first configuration sensitivity function that is provided by `OpenCSM`:

$$\frac{d\vec{x}}{dP} = \textbf{nodeSensit}()$$

returns the vector displacement of a Node with respect to the design parameter. It is expressed as a total derivative since the Node location is determined by the intersection of three or more Faces, and not by specifying a parametric coordinate (t or $\vec{u}$). To simplify the discussion, only one parameter is allowed to change at a time; if more than one changes simultaneously, the sensitivities associated with each parameter change can be summed.

The second configuration sensitivity function provided by `OpenCSM`:

$$\frac{\partial\vec{x}}{\partial P} = \textbf{edgeSensit}(t)$$

returns the vector displacement that is normal to the Edge at a point along the Edge (defined by t). Here this is expressed as a partial derivative since the change in the coordinate along the Edge (which occurs when the parameter t is changed) is ignored. In other words, the term $\partial\vec{x}/\partial P$ only represents the change in $\vec{x}$ that occurs because of a movement of the Edge location due to a change in the design parameter P (and not because of a change in t).

The third configuration sensitivity function provided by `OpenCSM`:

$$\frac{\partial w}{\partial P} = \textbf{faceSensit}(\vec{u})$$

returns the displacement "normal" to the Face due to a change in the design parameter P. Once again, this is a partial derivative since only changes in the Face location are considered, the Face parametric coordinates $\vec{u}$ are held constant). Also, since the displacement "normal" to the surface is simply a scalar, the dependent variable is written as w.

1. Nodes

Since the geometric sensitivity at Nodes is defined as a true vector in 3D space, it is used to find the changes in the Face parametric coordinates.

Consider the southwest Node on a given Face, corresponding to $i = 0$ and $j = 0$. The sensitivity of the grid point is simply the same as the sensitivity of the Node, or:

$$\left(\frac{d\vec{x}}{dP}\right)_{0,0} = \textbf{nodeSensit}(\text{southwest})$$

This is converted to changes in the Face's parametric coordinates by solving the 3×3 system for $d\vec{u}/dP$:

$$M\left(\frac{d\vec{u}}{dP}\right)_{i,j} = \left(\frac{d\vec{x}}{dP}\right)_{i,j}$$

where

$$M = \begin{bmatrix} x_u & x_v & (y_u z_v - z_u y_v) \\ y_u & y_v & (z_u x_v - x_u z_v) \\ z_u & z_v & (x_u y_v - y_u x_v) \end{bmatrix} \tag{5}$$

and, for example, $x_u \equiv \partial x/\partial u$.

2. Edges

There are several steps needed to get the proper grid sensitivities along the Edges. To begin, one needs to find the sensitivities of the bounding Edge parameters, t_{beg} and t_{end}. This is easily done since we know the sensitivity of the physical coordinates at the Nodes. Specifically,

$$\frac{dt_{\text{beg}}}{dP} = \left[\left(\frac{d\vec{u}}{dP}\right)_{\text{beg}}^{T} \bullet \left(\frac{d\vec{u}}{dt}\right)_{\text{beg}}\right] \Bigg/ \left[\left(\frac{d\vec{u}}{dt}\right)_{\text{beg}}^{T} \bullet \left(\frac{d\vec{u}}{dt}\right)_{\text{beg}}\right]$$

where all the derivatives on the right hand side are evaluated at t_{beg}. The derivative $(d\vec{u}/dP)_{\text{beg}}$ comes from the Node derivative computed above, and

$$\left(\frac{d\vec{u}}{dt}\right)_{\text{beg}} = \textbf{edgeDeriv}(t_{\text{beg}})$$

The sensitivity is computed in a similar way at t_{end}.

The sensitivity along the Edge at all intermediate points can be computed by differentiating Eq. (2), yielding

$$\left(\frac{dt}{dP}\right)_i = (1 - f_i)\left(\frac{dt}{dP}\right)_{\text{beg}} + (f_i)\left(\frac{dt}{dP}\right)_{\text{end}}$$

The derivative of the Face parametric coordinates with respect to the change in the Edge parametric coordinate (t) is computed by

$$\left(\frac{d\vec{u}}{dP}\right)_{i,0} = \left(\frac{d\vec{u}}{dt}\right)_i \left(\frac{dt}{dP}\right)_i$$

where

$$\left(\frac{d\vec{u}}{dt}\right)_i = \textbf{edgeDeriv}(t_i)$$

To compute the grid sensitivity along the Edge, one needs to combine the sensitivity due to the motion of the Edge and well as the change in position along the Edge. This is written (for a point along the south Edge) as

$$\left(\frac{d\vec{x}}{dP}\right)_{i,0} = \left(\frac{\partial\vec{x}}{\partial P}\right)_{i,0} + \left(\frac{\partial\vec{x}}{\partial\vec{u}}\right)_{i,0} \left(\frac{d\vec{u}}{dP}\right)_{i,0}$$

where

$$\left(\frac{\partial \vec{x}}{\partial P}\right)_i = \textbf{edgeSensit}(t_i)$$

and

$$\left(\frac{\partial \vec{x}}{\partial \vec{u}}\right)_{i,0} = \textbf{face}(\vec{u}_{i,0})$$

Finally, the sensitivity of the Face parametric coordinates must be updated at the Edges to account for the changes brought about by the Edge sensitivities. This extra step is needed in preparation for the Face procedure, described below. The process is done in a similar manned for transferring the Node sensitivities to the Edges; here, one solves

$$M\left(\frac{d\vec{u}}{dP}\right)_{i,0} = \left(\frac{d\vec{x}}{dP}\right)_{i,0}$$

for $(d\vec{u}/dP)_{i,0}$, where M is given in Eq. (5).

3. Face

To find the grid sensitivity for each point on a Face, start by differentiating the transfinite interpolation equation, Eq. (5):

$$\begin{aligned}\left(\frac{d\vec{u}}{dP}\right)_{i,j} &= (1-f_i)\left(\frac{d\vec{u}}{dP}\right)_{0,j} + (f_i)\left(\frac{d\vec{u}}{dP}\right)_{I-1,j} + (1-g_j)\left(\frac{d\vec{u}}{dP}\right)_{i,0} + (g_j)\left(\frac{d\vec{u}}{dP}\right)_{i,J-1} \\ &- (1-f_i)(1-g_j)\left(\frac{d\vec{u}}{dP}\right)_{0,0} - (f_i)(1-g_j)\left(\frac{d\vec{u}}{dP}\right)_{I-1,0} \\ &- (1-f_i)(g_j)\left(\frac{d\vec{u}}{dP}\right)_{0,J-1} - (f_i)(g_j)\left(\frac{d\vec{u}}{dP}\right)_{I-1,J-1}\end{aligned} \tag{6}$$

where f_i and g_j are defined by Eq. (4).

For each point on a Face, evaluate the Face slopes at each point, giving

$$\left(\frac{\partial \vec{x}}{\partial \vec{u}}\right)_{i,j} = \textbf{faceDeriv}(\vec{u}_{i,j})$$

from which one can compute the surface normal, $\vec{n}_{i,j}$.

Then, the surface sensitivity is computed:

$$\left(\frac{\partial w}{\partial P}\right)_{i,j} = \textbf{faceSensit}(\vec{u}_{i,j})$$

Recall that the surface sensitivity procedure only returns the normal component of the surface sensitivity, in this case the scalar w.

The grid sensitivity is finally obtained by combining the changes due to the surface moving (as given by the surface sensitivity) with the changes due to the movement of the grid point along the surface. This results in

$$\left(\frac{d\vec{x}}{dP}\right)_{i,j} = \left(\frac{\partial w}{\partial P}\right)_{i,j}\vec{n}_{i,j} + \left(\frac{\partial \vec{x}}{\partial \vec{u}}\right)_{i,j}\left(\frac{d\vec{u}}{dP}\right)_{i,j}$$

C. Examples

The above scheme is demonstrated on a configuration which is composed of a rectangular box from which a cylinder has been subtracted, resulting in a configuration that has six four-sided Faces. There are seven design parameters for this case, namely

$$
\begin{aligned}
P_1 &= x_{\text{base}} \text{ of the box} \\
P_2 &= y_{\text{base}} \text{ of the box} \\
P_3 &= z_{\text{base}} \text{ of the box} \\
P_4 &= \text{length of the box (in } x \text{ direction)} \\
P_5 &= \text{height of the box (in } y \text{ direction)} \\
P_6 &= \text{depth of the box (in } z \text{ direction)} \\
P_7 &= x \text{ distance by which right-hand Face is depressed}
\end{aligned}
$$

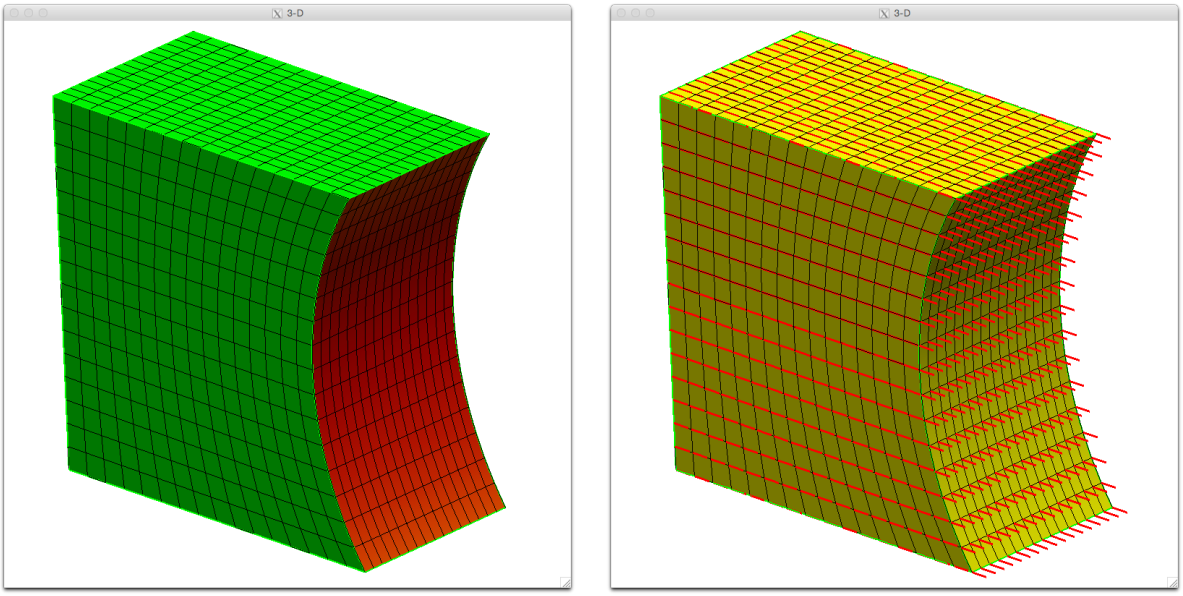
Figs. 11 to 17 show the surface and grid sensitivity to each of the design parameters. In part (a) of the figures, the surfaces are colored with the magnitude of the surface sensitivity (in which green corresponds to zero, red is positive, and blue is negative). Recall that the surface sensitivity for Faces consists just of the normal component (and is hence a scalar); for Edges, it consists just of the components that are normal to the Edge (locally); and for Nodes, it consists of the full three-dimensional sensitivity. In part (b) of the figures, tufts are placed at all the grid points showing the sensitivity of the grid points.

In Fig. 11, which corresponds to the sensitivity of $P_1 = x_{\text{base}}$, the only Faces that have a non-zero sensitivity are the left- and right-hand Faces. The green Faces (in part (a) of the figure) actually slide in a direction that is perpendicular to their normals, and hence are shown with a zero Face sensitivity. However, when one considers the sensitivity of the grid points, the effect of the translation of the boundaries of the Face (in the x-direction) is added to the (zero) Face sensitivity, resulting in a non-zero grid sensitivity (as shown by non-zero tufts in part (b) of the figure). In other words, as expected, every grid point has a sensitivity that is given by the vector $(1, 0, 0)$. Similar results are shown in Figs. 12 and 13 for the sensitivities of the box's base point in the other coordinate directions.

In Fig. 14, the sensitivity with respect to P_4 = length of the box, Face sensitivities are very similar those shown in Fig. 11(a). (In other words, the top, bottom, front, and back Faces all have a zero Face sensitivity.) However, the grid sensitivities are no longer constant on those Faces; those at the left (at $x = x_{\text{base}}$) are zero and those at the right-hand side (x near $(x_{\text{base}} + L)$) are approximately $(1, 0, 0)$. Fig. 14(b) shows the grid sensitivities to be smoothly graded in between.

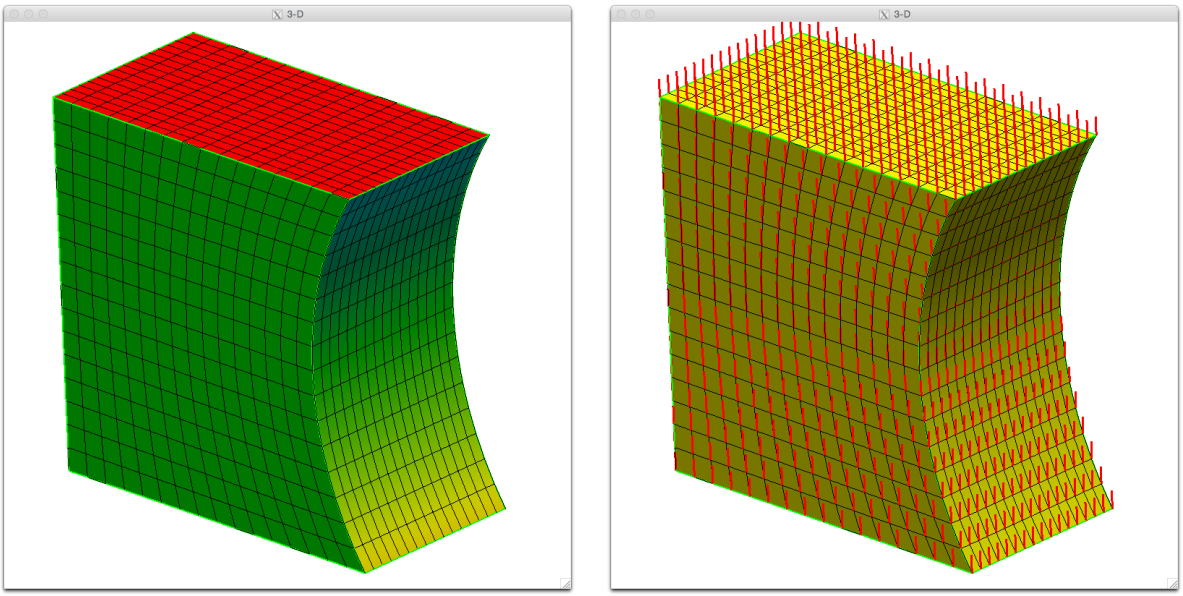
Fig. 17 shows the sensitivity with respect to P_7 = the amount by which the right-hand Face is depressed (which is related to the radius of the cylinder that is subtracted from the box). In this case, the tufts on the front Face point to the left, since a positive change in P_7 results in a leftward motion of the right-hand Face. The grid sensitivities on the top and bottom Faces are all zero since the location of the right-hand Face has not changed — just its curvature has changed. Again, the proper grid sensitivities are reconstructed from the Face, Edge, and Node sensitivities.

Again, in this demonstration, a simple algebraic grid generator was used for illustrative purposes (in order to keep the description reasonably compact); the same process for finding grid sensitivities from configuration (Face, Edge, and Node) sensitivities can be applied to any structured or unstructured grid generator if the governing equations are properly differentiated.



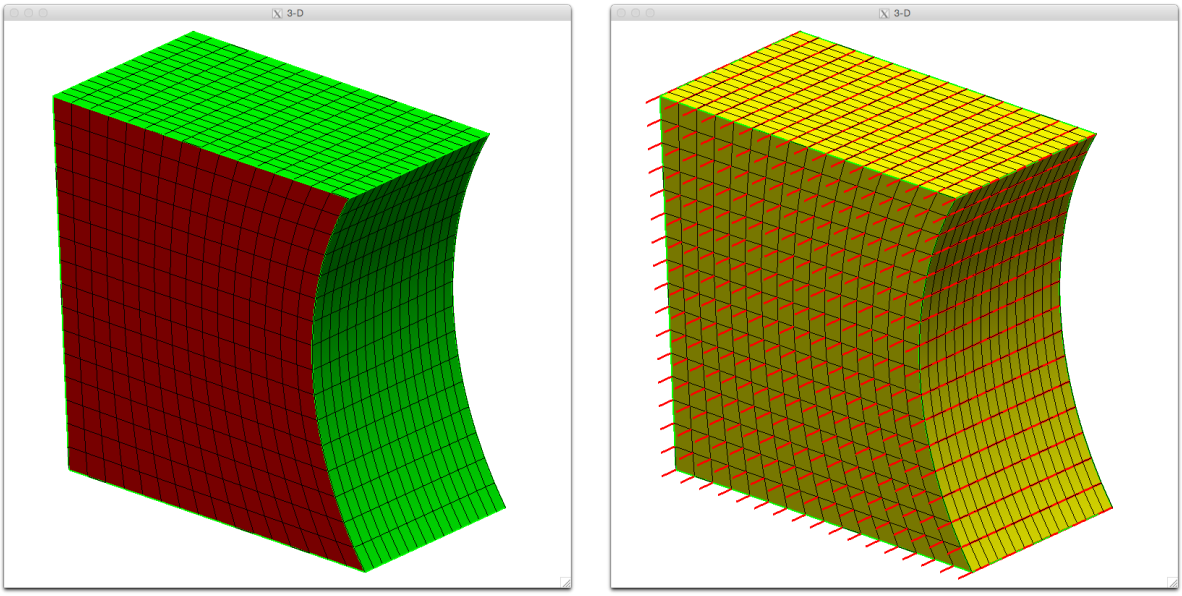
(a) configuration sensitivity **(b) grid sensitivity**

Figure 11. Computed sensitivity with respect to P_1 ($x_{\mathbf{base}}$ of the box)



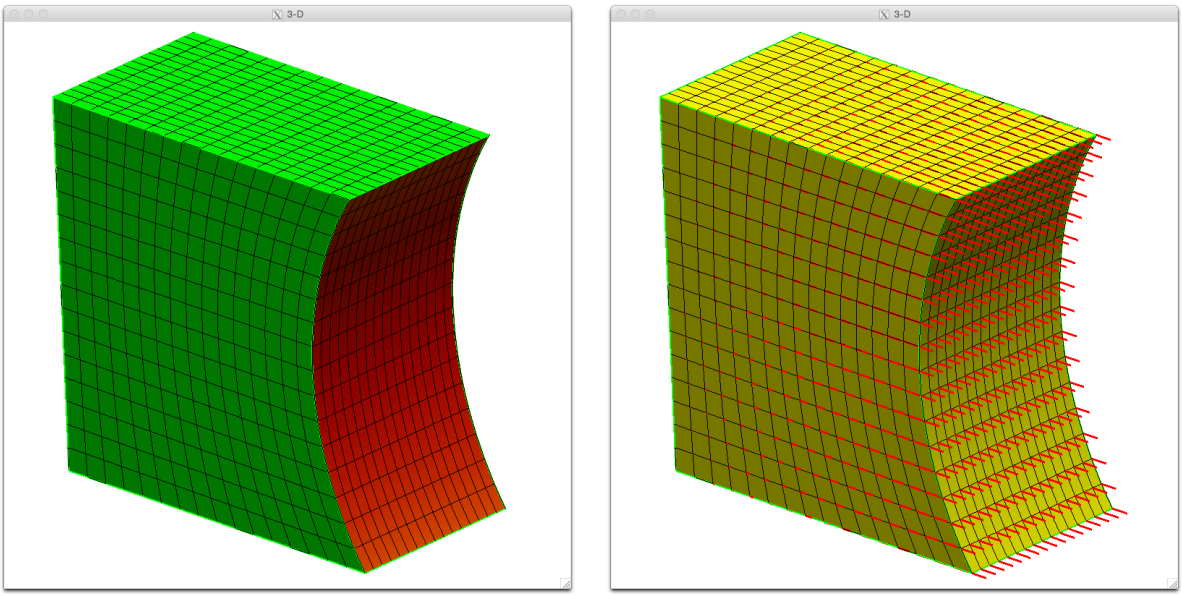
(a) configuration sensitivity **(b) grid sensitivity**

Figure 12. Computed sensitivity with respect to P_2 ($y_{\mathbf{base}}$ of the box)



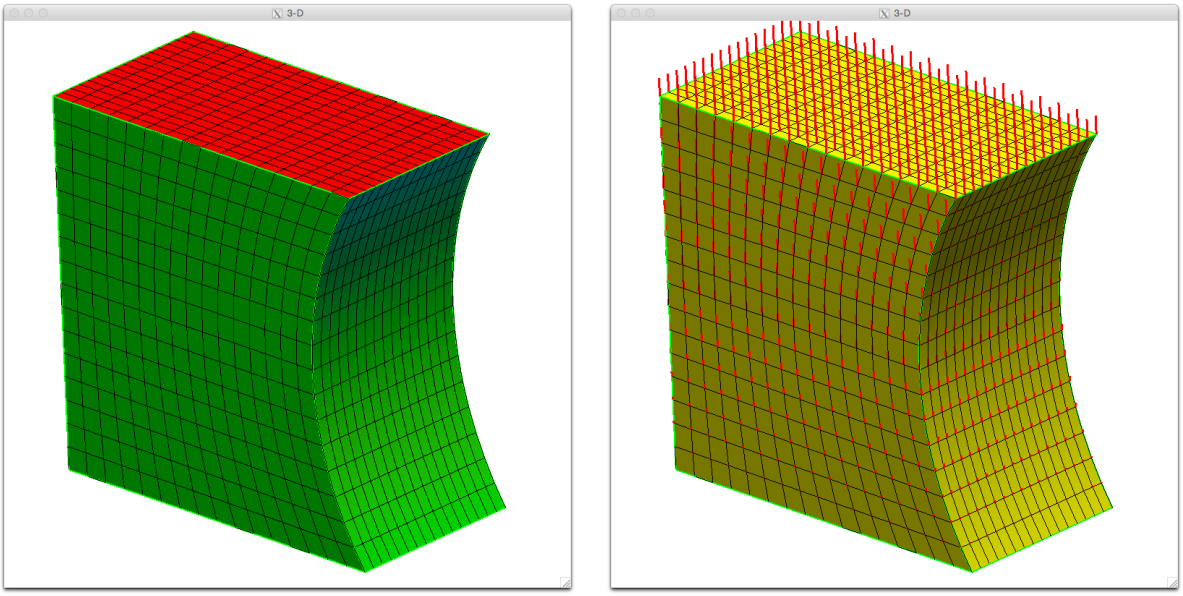
(a) configuration sensitivity (b) grid sensitivity

Figure 13. Computed sensitivity with respect to P_3 (z_{base} of the box)



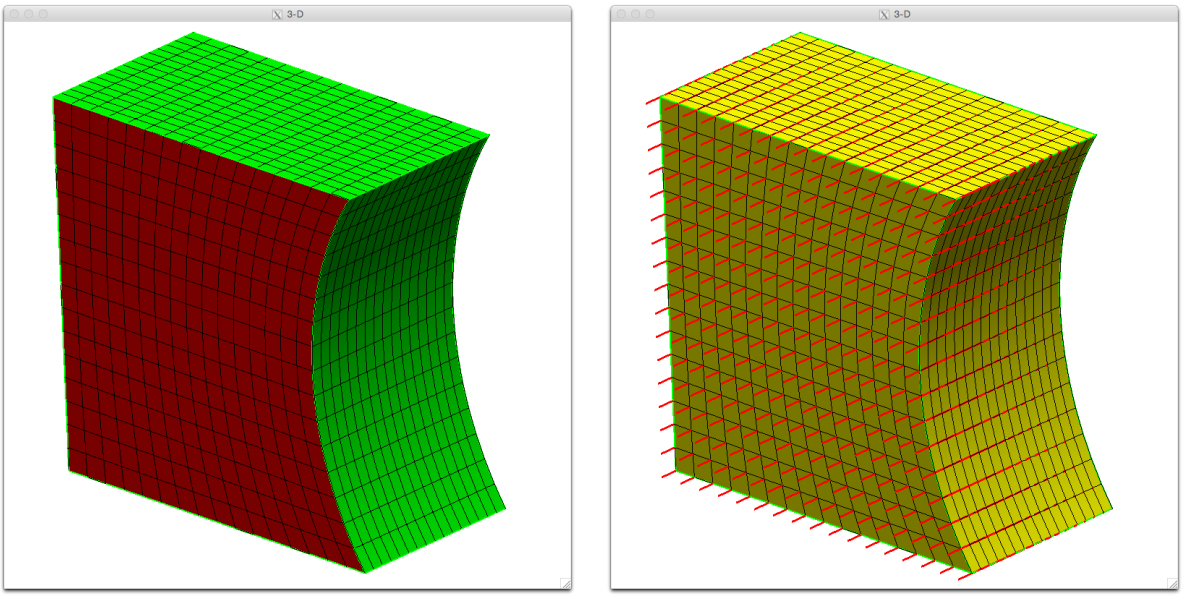
(a) configuration sensitivity (b) grid sensitivity

Figure 14. Computed sensitivity with respect to P_4 (length of the box (in x-direction))



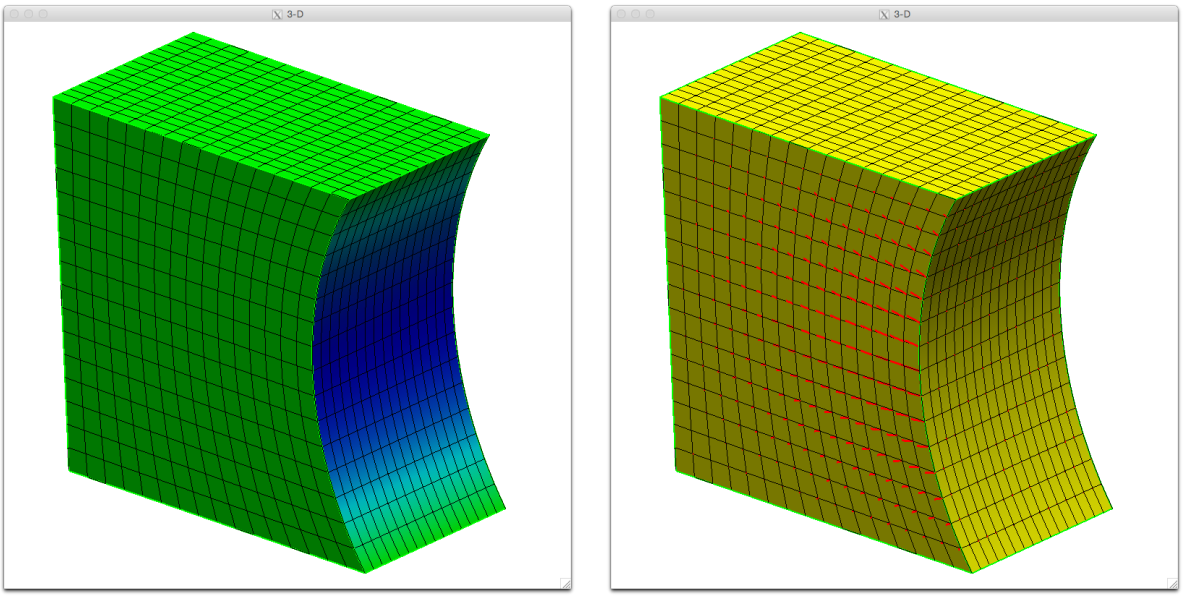
(a) configuration sensitivity (b) grid sensitivity

Figure 15. Computed sensitivity with respect to P_5 (height of the box (in y-direction))



(a) configuration sensitivity (b) grid sensitivity

Figure 16. Computed sensitivity with respect to P_6 (depth of the box (in z-direction))



(a) configuration sensitivity **(b) grid sensitivity**

Figure 17. Computed sensitivity with respect to P_7 (x distance by which right-hand Face is depressed)

V. Summary

New techniques for computing the sensitivity of a parametric, CAD-generated configuration with respect to its driving parameters are described. For most configurations, analytically-computed derivatives that are robust and efficient are employed. For cases where this is not possible (such as fillets), a new finite-difference approach is used. Both of these techniques have been implemented in the Engineering Sketch Pad (`ESP`) (which is built on top of `OpenCSM`, `EGADS`, and `OpenCASCADE`). Results of these new techniques are shown on a few configurations.

Also, a technique for computing the sensitivity of points in a computational grid given the configuration sensitivities is described. While the details of this process are only described for a simple algebraically-generated structured grid, the concepts described are equally applicable to other structured and unstructured grid generation techniques.

Given these processes, it is now possible to compute the sensitivity of a MDAO objective function with respect to the design parameters that describe a complex three-dimensional configuration. This overcomes one of the last major impediments to the widespread adoption of multi-disciplinary optimization processes for design.

Acknowledgement

This work was funded by NASA Cooperative Agreement NNX11AI66A: Topic 2.1 - MDAO Open-source Engineering Framework; Chris Heath is the Technical Monitor.

References

[1]Bischof, C., Carle, A., Corliss, G., Griewank, A., and Hovland, P., "ADIFOR — Generating Derivative Code from Fortran Programs", *Sci, Program.*, Vol. 1, No. 1, pp 11–29, January 1992.

[2]Hascoet, L, and Valérie, P., "The Tapenade Automatic Differentiation Tool: Principles, Model, and Specification", *ACM Trans. Math. Softw.*, Vol 39, No. 3, pp 20:1–20:43, May 2013.

[3]Dannenhoffer, J.F., "OpenCSM: An Open-Source Constructive Solid Modeler for MDAO", AIAA-2013-0701, 51^{st} AIAA Aerospace Sciences Meeting, Grapevine, TX, January 2013.

[4]Haimes, R., and Drela, M., "On the Construction of Aircraft Conceptual Geometry for High Fidelity Analysis and Design", AIAA-2012-0683, January 2012.

[5]Lazzara, D.S., "Modeling and Sensitivity Analysis of Aircraft Geometry for Multidisciplinary Optimization Problems", PhD. Thesis, Massachusetts Institute of Technology, June 2012.

[6]Nemec, M., Aftosmis, M.J., and Pulliam, T.H., "On the use of CAD and Cartesian methods for aerodynamic optimization", Proceedings of the 3^{rd} International Conference on Computational Fluid Dynamics , Toronto, Canada, Springer-Verlag, July 2004.

[7]Nemec, M. and Aftosmis, M.J., "Aerodynamic shape optimization using a Cartesian adjoint method for CAD geometry", AIAA-2006-3456, 24^{th} AIAA Applied Aerodynamics Conference, June 2006.

[8]Nemec, M., and Aftosmis, M.J., "Parallel Adjoint Framework for Aerodynamic Shape Optimization of Component-Based Geometry", AIAA-2011-1249, Jan. 2011.

[9]Brock, W., Burdyshaw, C., Karman, S., Betro, V., Hilbert, B., Anderson, K., and Haimes, R., "Adjoint-Based Design Optimization Using CAD Parameterization Through CAPRI", AIAA-2012-968, 50^{th} AIAA Aerospace Sciences Meeting, Nashville, TN, January 2012.

[10]Haimes, R. and Aftosmis, M., "Watertight Anisotropic Surface Meshing Using Quadrilateral Patches", Proceedings of the 13^{th} International Meshing Roundtable, Williamsburg, VA, September 2004.

[11]Haimes, R. and Dannenhoffer, J.F., "The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry", AIAA-2013-3073, AIAA Computational Fluid Dynamics Conference, San Diego, CA, June 2013.

[12]Eriksson, L.-E., "Generation of boundary conforming grids around wing-body configurations using transfinite interpolation", *AIAA J.,* Vol. 20, pp. 1313–1320, 1982.