AIAA SciTech Forum
8-12 January 2024, Orlando, FL
AIAA SCITECH 2024 Forum

10.2514/6.2024-1316



# Using Faceted Geometries for Analysis and Design

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

**Many analysis and design tasks start from a previous configuration that is (unfortunately) defined by tessellated surfaces, such as in a stereo-lithography file. Sometimes these definitions come from legacy CAD tools that are no longer supported, and sometimes they come from laser scans of a real configuration (such as a wing with ice growing on the leading edge). While these files define a watertight configuration that look good in the eye-ball norm, the fact that the surfaces are faceted yields less-than-desirable results when used in CFD and FEM tools. This paper describes two techniques for transforming faceted representations into smooth boundary representations.**

**The first, called Slugs, generates a static representation by breaking the facets into contiguous groups and then fitting each group with a B-spline; the configuration is called "static" since it cannot easily be modified, except possibly via free-form deformation tools that operate on the associated B-splines.**

**The second, called Plugs, automatically modifies the design parameters of a user-defined parametric model to best-fit the points in the tessellated surfaces.**

**This paper describes these two techniques and the various enhancements in them since their introduction is previous publications.**

## Generating a Static Representation via Slugs

The generation of static representations via `Slugs` was first described by Dannenhoffer.[1] The overall process consists of the following steps:

1. Import the configuration. This is accomplished using one of the following input schemes:
   - input a completely disconnected suite of points;
   - read an `STL` file that contains a set of points that are at the corners of a collection of triangles, which may or may not be fully connected;
   - read an `IGES` file and tessellate the trimmed surfaces. This provides a suite of partially connected points through each tessellation; or
   - read a `STEP` file and perform the same function as the `IGES` file for non-manifold bodies.

   `EGADS`[2] can be used to generate the tessellations from these **IGES** and **STEP** inputs.
2. Separate the points into "colors", where each color represents a part of a component. For example, there may be one color that represents the leading edge region of the left wing. The amount of user information needed in the separation process is greatly reduced if the points are connected via triangles.
3. Break each colored region into 2-, 3-, or 4-sided regions.
4. Create Boundary Representation (Brep) Nodes at any location where the adjacent triangles have more than 2 colors. Create Brep Edges at any location where the adjacent triangles have only two colors. Finally create Brep Faces in the regions of a single color.

*Associate Professor, Mechanical and Aerospace Engineering, AIAA Associate Fellow.

### Segregating the Tessellation into Groups

The most time-consuming process in the above is involved with breaking the configuration into colors. For example, consider the simple wing-body configuration shown in Fig. 1. This configuration was imported into `SLUGS` via an `STL` file, which contains a set of triangular facets.

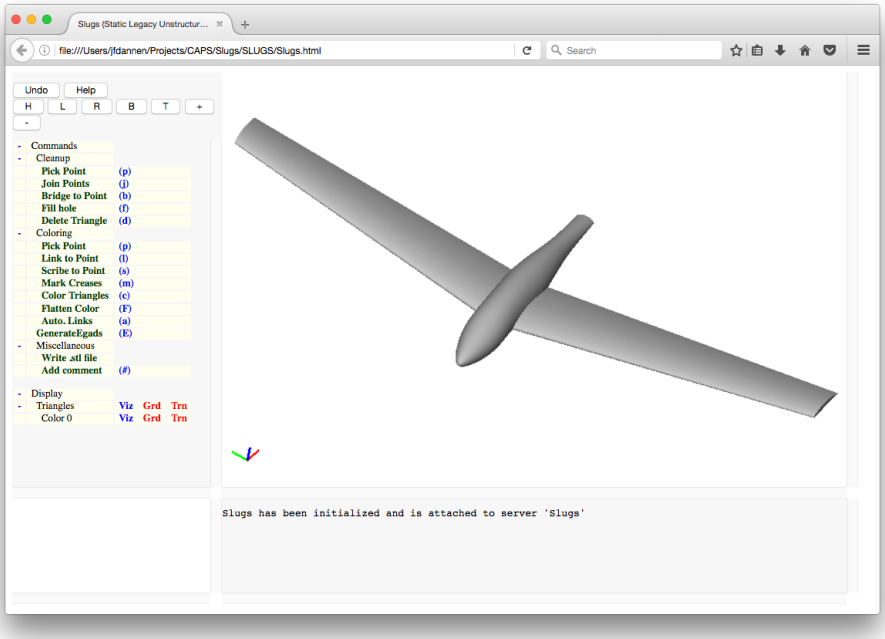



**Figure 1. Initial wing-body configuration.**

The first step is to break up the configuration by marking triangle sides that are associated with "creases", where a crease is defined as the sides between triangles that have surface normal directions that differ by more that a user-specified amount. If the crease detector is used, with a tolerance of 45°, the sides that are shown in white in Fig. 2 are automatically marked. The colors are assigned by selecting one triangle (in each region) and then informing the system to color it and every other triangle that is adjacent to it; the propagation process stops whenever all the triangles are either adjacent to a triangle of the same color or adjacent to a "marked" side. This whole marking and filling process requires less than one minute for a configuration with a few dozen colors.

Note that the facets (from the `STL` file) are only used to determine the color associated with each point in the cloud. As such, there is no requirement here that the facets be connected, or that the points are even associated with facets, although having them defined in terms of connected facets makes make their separation into colors a simpler process.

In addition to the above, it is often desirable to split some components further before coloring. An example would be to split the upper and lower surfaces of a wing or the two sides of the fuselage. The brute-force way of doing this is to (graphically) mark each of the associated triangle sides, and then re-color the facets on one side of the newly-marked sides. This process is VERY time-consuming and error-prone. To circumvent this, an automatic marking process has been employed that uses a variant of the Dijkstra algorithm[3] for finding the shortest path through a graph (in this case, connected triangles). Hence, this process can be made very efficient by the user graphically selecting a starting point and a target point; then the Dijkstra algorithm marks all the triangle sides on the shortest path. At the end of this, the previous "target" point becomes the starting point for the next operation. Using this operation (and subsequent

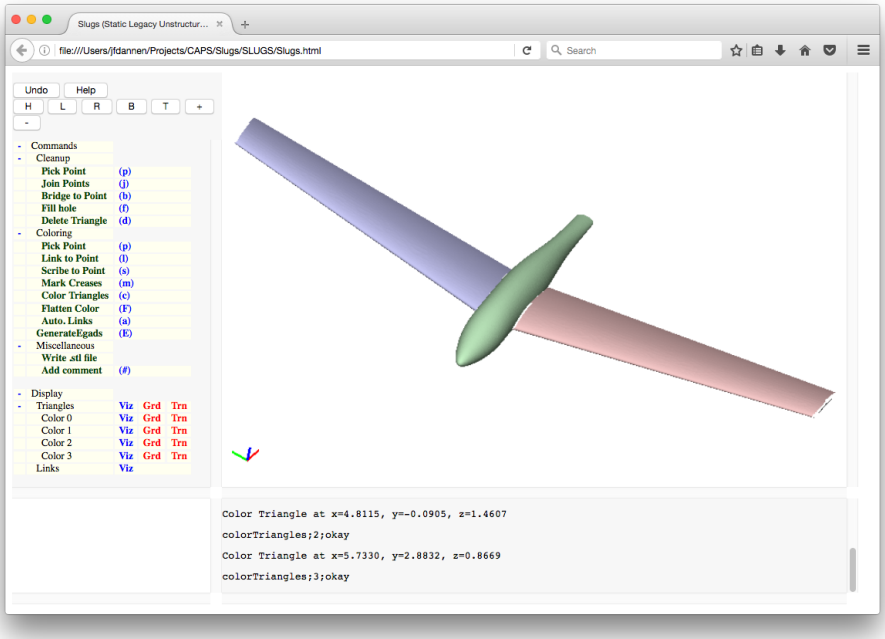



**Figure 2. Colored wing-body configuration after automatic crease detector is applied.**

coloring), one can obtain the results shown in Fig. 3 interactively in about 4 minutes.

The above algorithm works well when there is a set of triangle sides that form a relatively straight path between the starting and target points; this occurs when the original set of triangles come from a tessellation of a series of surface patches, such as from an `IGES` or `STEP` file. For the wing-body configuration, one can see this along the symmetry plane and at the sharp junction between the fuselage and the tail, as shown in Fig 3.

Unfortunately, this is not always the case. For example, if one wanted to split the fuselage into two colors (one above and one in front of the wing), the set of triangle sides forms a rather jagged path (also as shown in Fig. 3). Using a path such as this would result in large fitting errors (because the fitter assumes a relatively smooth curve through the points between two colors).

To remedy this, a "scribe" operation is available in `SLUGS` that breaks the triangles along a "straight" path between the source and target; the Dijkstra algorithm is then used to mark the new triangle sides in preparation for coloring. The algorithm used to scribe consists of the following steps:

- Use the Dijkstra algorithm to find the shortest (jagged) path through the triangulation between the source and target points, such as was shown in Fig. 3;

- Iteratively smooth the path by placing each point on the path on the straight line between its neighboring path points. Since this tends to move the point off the surface, the new locations are projected back onto the tessellation. The smoothing process stops when all the angles between the adjacent segments in the path are below some tolerance (for example, 1°) or until the angles do not change between smoothing steps. (The latter often occurs because of the curvature of the surface that is being scribed.) This smoothing and projection process usually requires about 200 iterations and executes in less than a minute on a laptop computer.

- Break the triangles along the new path. This is done by traversing the smoothed path and bisecting

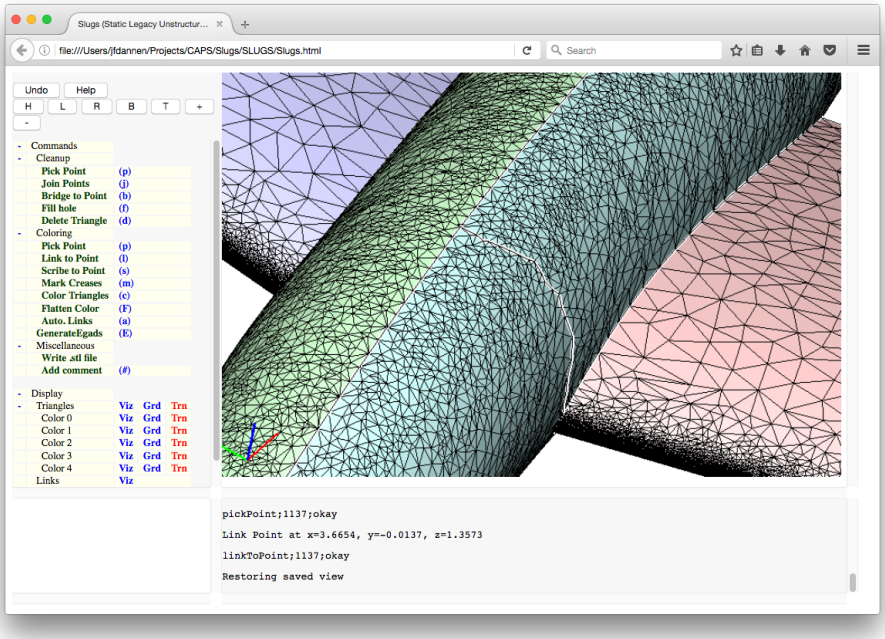



**Figure 3. Triangles on fuselage of wing-body configuration, showing nicely aligned triangle sides along the symmetry plane but a jagged set of triangle sides along the fuselage surface.**

the triangles that are intersected.

- Use the Dijkstra algorithm to mark the "new" shortest path between the source and the target.

Fig. 4 shows the result of applying the scribe operation to the fuselage. The most time-consuming part of this process is associated with the projections to the tessellation. This can be done by finding the smallest distance between the point and each triangle; an iterative solver whose independent variables are the barycentric triangle coordinates is used for each triangle.

Since in general there are a large number of triangles, to be efficient one needs to carefully select which triangles to check. In `SLUGS`, this is done by traversing an octree of the triangles, in which each octant contains at most $n_{\text{tri}}$ triangles, where $n_{\text{tri}}$ is selected as a balance between the time needed to create and subsequently traverse the octree and the time needed to perform the "nearest to" calculation to the (up to) $n_{\text{tri}}$ triangles in the octant. It was found experimentally that $n_{\text{tri}} = 250$ yields the fastest scribes.

### Fitting B-spline Curves Between the Groups

The next step is to create the Edges between the colors. This is done by looping through each pair of colors. For each pair, all the associated triangle sides are put into a list and sorted, so as to create a list of consecutive points. These points are then fit into a B-spline Curve using the Levenberg-Marquardt algorithm,[4,5] which is a combination of the gradient descent and Newton methods. Levenberg and Marquardt combined the two methods with a damping parameter, $\lambda$, which is automatically adjusted during the optimization process.

The design variables in the optimization consist of the interior points in the B-spline control net as well as the parametric location of each of the points in the cloud. The latter are initialized based upon the arc-length distances between the points to be fit. Details are given in.[1] The Levenberg-Marquardt process is very fast due to the availability of gradients from the `ESP` system.[6] Details are given in.[1]

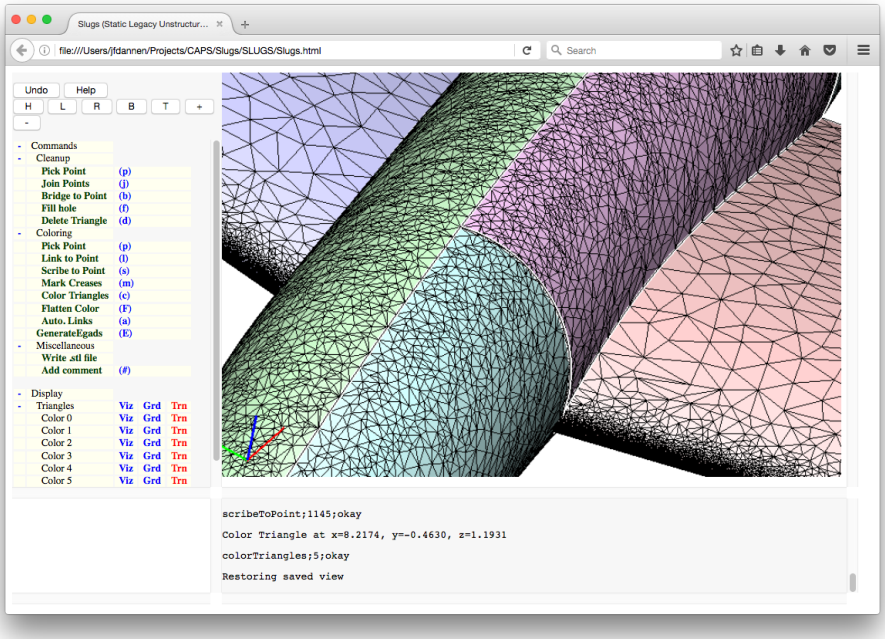



**Figure 4. Triangles on fuselage of wing-body after scribing.**

### Fitting B-spline Surfaces to each of the Group

To create the Brep Faces, a similar process is used as above. Although the matrix structure and computations are a bit more complicated than for the case of the Edges, the math is the same and the process converges in well under 1000 Levenberg-Marquardt steps. The design variables consist of the locations of the (two-dimensional array of) interior points in the control net and the parametric coordinate locations associated with each point in the cloud. Cases with up to 5000 points in the cloud and a control net of $11 \times 11$ control points are routinely solved in under a minute on a laptop computer. Because of the sparsity of the Jacobian matrix, it has been found that the time required scales linearly with the number of points in the cloud.

### Combining the B-splines into a Boundary Representation

Once the Edges and Faces are known, it is a straightforward process to assemble them into a closed boundary representation (Brep). This is because the fitting processes for Edges and Faces only adjust the interior points; the endpoints for Edges are fixed at the Nodes (junctions of more than two colors) and the outer boundaries of Faces are fixed by the Edges.

### Advantages and Disadvantages of `Slugs`

The biggest advantage of `Slugs` is that one does not need to start with a (parametric) model of the Body to be fit. Also, the fitting process is very fast due to the availability of analytic sensitivities.

The chief disadvantage of `Slugs` is that `Slugs` produces a non-parameterized static Brep. Also, the separation of the points in the cloud into colors can be very tedious and prone to errors. Lastly, all the Faces in the Brep are limited to be 2-, 3-, or 4-sided; Faces that contain holes must be broken up (via scribing and multiple colors) into smaller Faces (as can be seen with the fuselage in the example).

## Generating a Parametric Representation via Plugs

The generation of parametric representations via `Plugs` was first described by Jia and Dannenhoffer.[7] The overall process consists of the following steps:

1. Generate a parametric model, with several design parameters (DESPMTRs);
2. Change the DESPMTRs so that the bounding boxes of the cloud of points and the model match;
3. Mark all the points in the cloud as unclassified;
4. Perform several passes:
   - For each point in the cloud, determine to which Face (if any) it should be classified. If no cloud points are classified differently than in the previous pass, stop.
   - Modify the DESPMTRs to least-squares minimize the distances between the classified cloud points and their respective Faces.

### Generating the Baseline Parametric Models

This step is performed using the normal `ESP` build process. The only real restriction is that the model must be parameterized. The user must prescribe initial values, and lower and upper bounds for each DESPMTR. Fig. 5 shows such a model for a duct that has 15 DESPMTRs (5 for each internal cross-section).

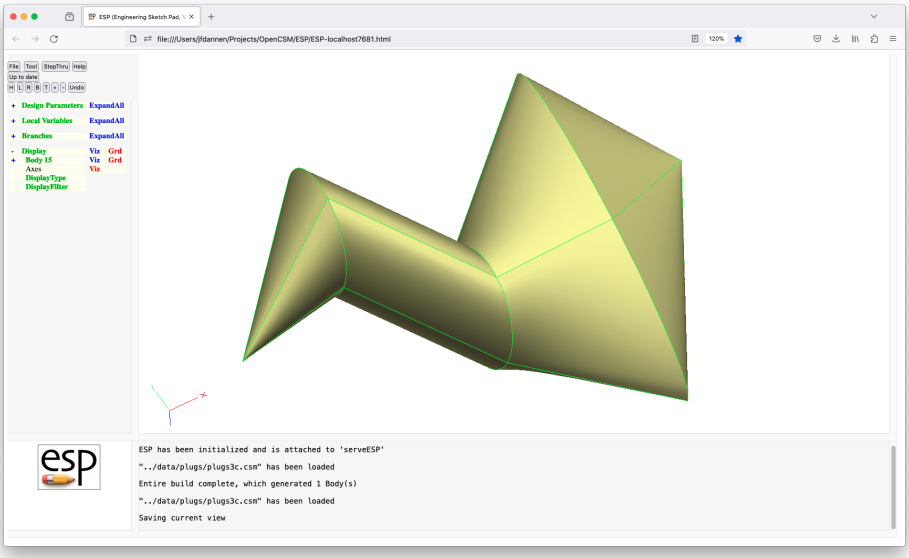



**Figure 5. Initial configuration with 15 design parameters (DESPMTRs)**

### Associating Points in the Cloud with Model Faces

The first step is to read in the cloud of points, which is simply a file with $(x, y, z)$ triplets. Fig. 6 shows the initial configuration with the points superimposed; the points are all colored red to indicate that they are unclassified.

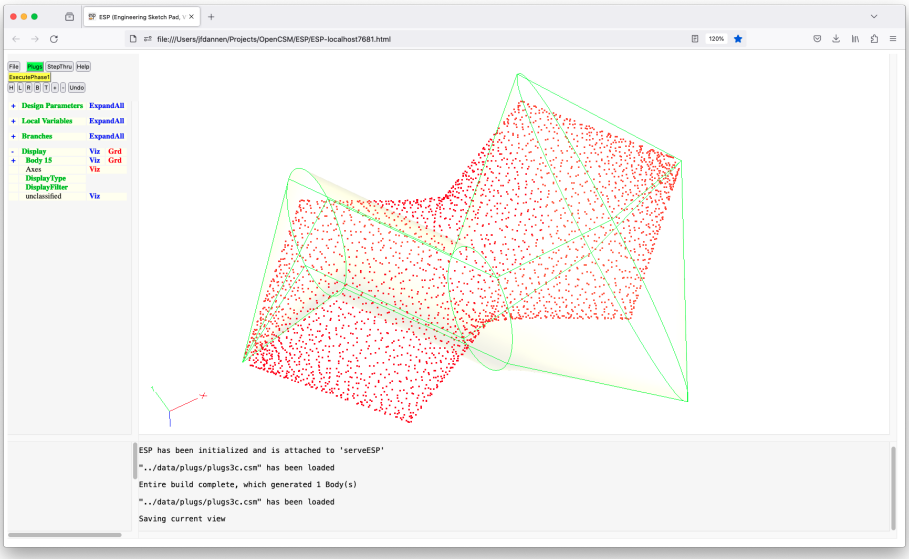



**Figure 6. Initial configuration with cloud points superimposed.**

### Adjusting the Bounding Box

The Levenberg-Marquardt[4,5] optimizer to then used to modify the DESPMTRs so that the bounding boxes of the point cloud and the (modified) configuration match. The results of this is shown in Fig. 7.

Note that for many configurations (where the initial model is relatively close to the cloud of points), this step can be skipped.

### Optimizing the Fit Between the Model and the Points

The main computational effort then is in adjusting the DESPMTRs to minimize, in a least-squares sense, the distances from the points in the cloud to the configuration. This is done in a series of passes.

At the beginning of a pass, each point in the cloud is visited and the distance of that point to every Face in the configuration is determined. If the distance to any one Face is significantly smaller than all the other Faces, the point is classified to belong to that Face. Points that are near the junctions between two Faces generally remain unclassified in early passes; in later passes, when the configuration gets close to the points in the cloud, these points will get classified.

In the event that every cloud point is classified the same as it was in the prior pass, the optimization process stops.

After the classification step, the Levenberg-Marquardt[4,5] is executed, where the objective is to reduce the root-mean-square (RMS) error with the design variables being the DESPMTRs and the parametric coordinates of each cloud point in the Face to which it is classified. Unclassified points are skipped in this step.

Fig 8 shows the evolution of the configuration after each pass of the Levenberg-Marquardt optimizer. In each case, the classified points are shown in black and the unclassified points are shown in red.

At the beginning of the fist pass, all 3704 points are classified and the optimizer modified the DESPMTRs to produce an RMS error of $1.6 \times 10^{-2}$ (see Fig. 8(a)). At the beginning of pass 2, 703 of the cloud points are reclassified and the optimizer reduces the RMS error to $2.9 \times 10^{-6}$ as shown in Fig. 8(b). The third pass reclassifies 239 points and the RMS error is further reduced to $1.1^{-7}$, shown in Fig. 8(c). At the beginning of the fourth pass, none of the points are reclassified and this the algorithm exits.

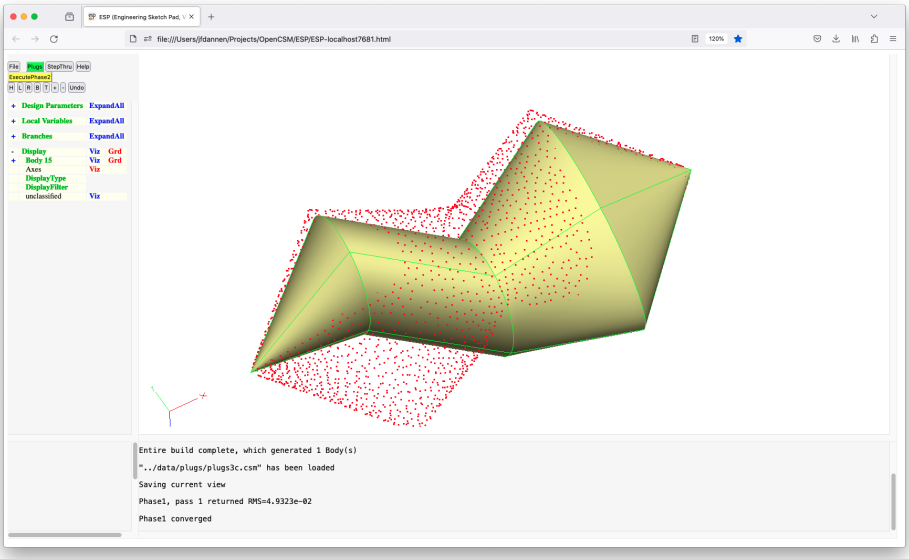



**Figure 7. Configuration after matching bounding boxes.**

### Advantages and Disadvantages of `Plugs`

The biggest advantage of `Plugs` is that it results in a parametric model (which could serve as the basis for design analysis or optimization). Another big advantage is that the points in the cloud do not need to be classified by the user. Lastly, `Plugs` is very fast, especially when `ESP` provides the required sensitivities analytically.

The main disadvantage of `Plugs` is that the user must create the parametric model to be fit; this model must be such that there is a set of its DESPMTRs that matches the points in the cloud.

## Acknowledgment

This work was supported by the EnCAPS Project (AFRL Contract FA8650-20-2-2002): "EnCAPS: Enhanced Computational Prototype Syntheses", with Ryan Durscher as the Technical Monitor.

## References

[1]Dannenhoffer, J.F., "The Creation of a Static BRep Model Given a Cloud of Points", AIAA-2017-0138, presented at the AIAA SciTech 2017, January 2017.

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

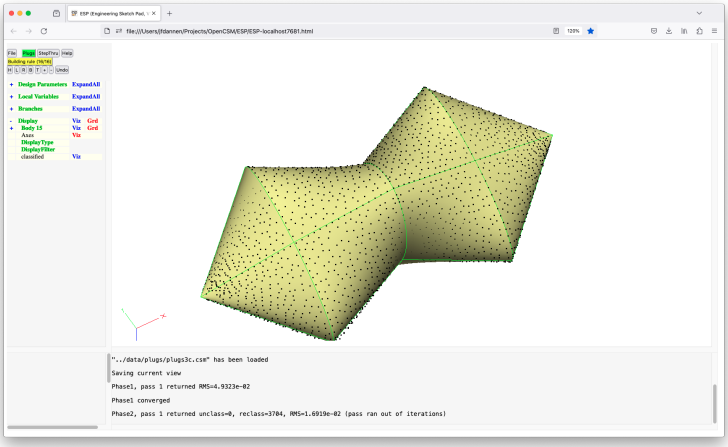
[3]Dijkstra, E.W., "A note on two problems in connexion with graphs", *Numerische Mathematik*, Vol 1, pp 269-271, 1959.

[4]Levenberg, K., "A Method for the Solution of Certain Non-Linear Problems in Least Squares," *Quarterly of Applied Mathematics*, Vol 2, pp 164–168, 1944.

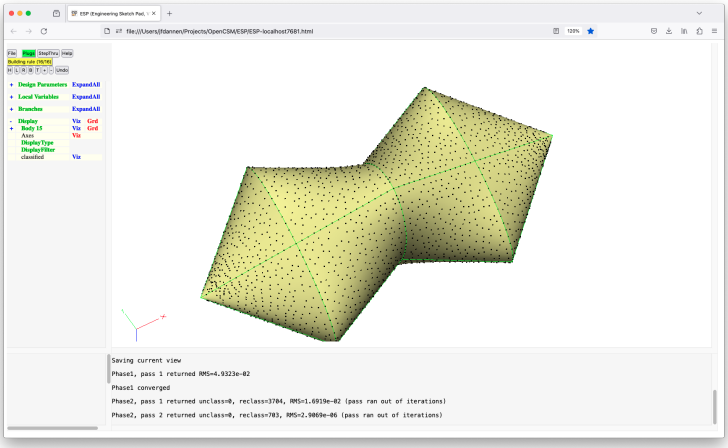
[5]Marquardt, D., "An Algorithm for Least-Squares Estimation of Non-linear Parameters", *SIAM Journal on Applied Mathematics*, Vol 11, No 2, pp 431–441, 1963.

[6]Dannenhoffer, J.F., and Haimes, R., "Design Sensitivity Calculations Directly on CAD-based Geometry", AIAA-2015-1370, January 2015.

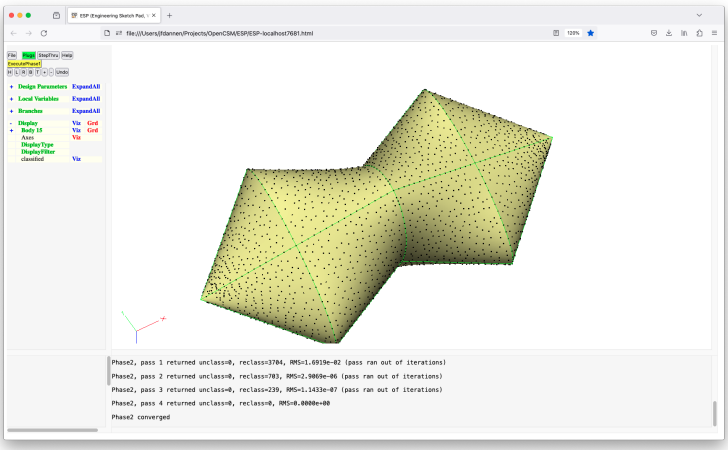
[7]Jia, P, and Dannenhoffer, J.F., "Generation of Parametric Aircraft Models from a Cloud of Points", AIAA-2016-1926, presented at AIAA SciTech 2016, January 2016.




**(a) after first iteration**




**(b) after second iteration**




**(c) after third iteration**

**Figure 8. Configuration history when matching configuration to cloud of points.**