The Creation of a Static BRep Model Given a Cloud of Points

John F. Dannenhoffer, III*

Aerospace Computational Methods Laboratory

Syracuse University, Syracuse, New York, 13244

Modern high-fidelity aerospace analysis and design tools require watertight, analytically-smooth representations of the configuration under consideration. Most recently created geometric models have appropriate representations; unfortunately, *legacy* configurations often do not have a geometric representation commensurate with the analysis task at-hand. Either the analytic data has been lost and the remaining configuration is in a discrete setting, or the legacy analytic information is not smooth or is not closed at the required precision.

In order to utilize high-fidelity analyses on these legacy configurations, it is necessary to have a tool through which a user can easily create the required smooth watertight geometric model. Described here is a new interactive system, named `SLUGS` (`S`tatic `L`egacy `U`nstructured `G`eometry `S`ystem), which starts with a cloud of points (which may be fully- or partially-connected through a surface tessellation) and which "fits" the cloud to a watertight set of B-spline surface patches driven by the user. The process for this conversion, as well as the key technologies that are used, are described in this paper.

I. Background

Over the past decade, several design and analysis tools have become available that operate well on configurations that are described in terms Boundary representations (Breps) that consist of Faces whose geometry is defined in terms of B-splines (or Nurbs). Unfortunately, there are at least two situations in which watertight B-spline-based Breps are not readily available.

The first situation involves configurations that are described as `IGES` files. Here the geometry is frequently based upon B-splines (or Nurbs), but the various surface patches do not form a watertight representation (since the `IGES` files do not contain topological information). Several tools for fixing this include CadFix,[1] which is an "advanced healing and repair engine to correct and repair geometric and topological flaws within the CAD model". This works well in some cases, but in others it creates sliver Faces that make downstream analyses problematic.

The second situation occurs when a configuration is defined by point data that comes from 3D laser scans or other probes. In this case, not only are B-spline-based surface not available, but also the connectivity amongst points is unknown.

There are currently at least two commercially-available software tools that allow one to create an analyzable model from a cloud of points.

Geomagic Wrap[2] "enables users to transform point cloud data, probe data and imported 3D formats into 3D polygon meshes for use in manufacturing, analysis, design, entertainment, archeology and analysis". A tool such as this may be appropriate for design and analysis processes that can use geometry defined in terms of polygonal meshes, but not be appropriate if smooth surfaces are needed; the latter is the case when surface slope and curvature have a strong influence on the predicted performance.

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

ScanIP+Nurbs[3] claims to "create Non-Uniform Rational B-Splines (NURBS) which provides a route from image data to computer aided design (CAD) packages". From the pictures on the website, it appears that this tools creates many small Nurbs patches. Such patches may not be appropriate for structured mesh generators that prefer fewer large surface patches.

Described here is a new tool, named `SLUGS`, which starts with a cloud of points and which "fits" the cloud to a watertight set of B-spline surface patches driven by the user. First the overall process is described, and then each of the key technologies are described in detail: these include the process for breaking the points into "colors", each of which will become a B-spline-based Face in the final configuration; and the process for fitting B-splines to the points along the Edges and within the Faces with B-spline curves and surfaces.

II. Overall Process

While it would be desirable to have a completely automatic system for generating a watertight representation from any legacy configuration, such a capability may always be out of reach. Hence, the system developed and described here requires a user-assisted approach. The driving guidelines for such an approach are:

- it should require the minimum amount of user interaction as possible, and
- it should allow the maximum flexibility to the user.

To accomplish this, a new process has been developed that uses the following strategy:

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 connected
   - read an `IGES` file and tessellate the trimmed surfaces. This provides a suite of partially connected points through each tessellation.
   - read a `STEP` file and perform the same function as the `IGES` file for non-manifold bodies

   `EGADS`[4] 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. (The 2- and 3-sided regions are only allowed for co-planar sets of points).
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.

The technical details of each of these steps is described in the sections that follow.

III. Breaking into colors

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.

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

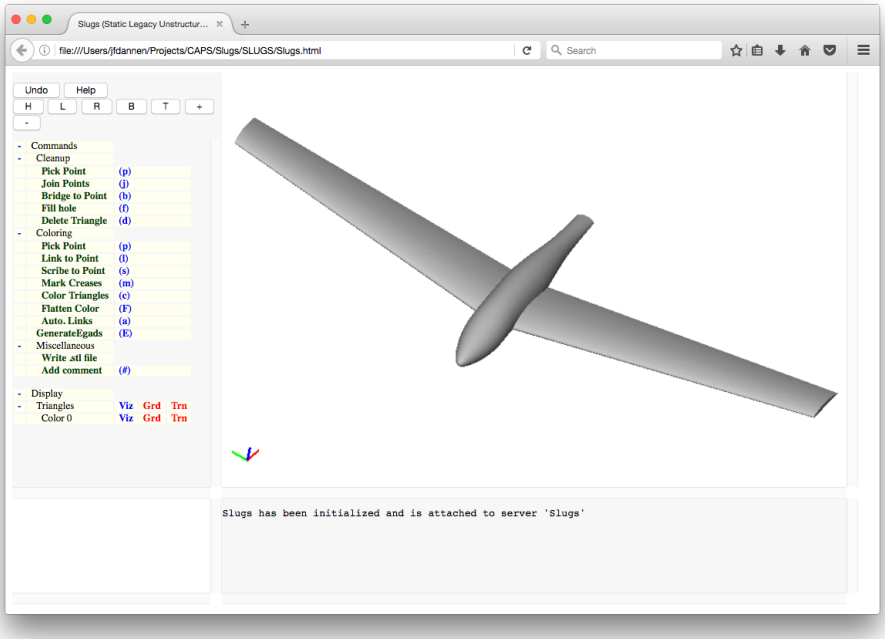


Figure 1. Initial wing-body configuration.

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[5] 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 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.

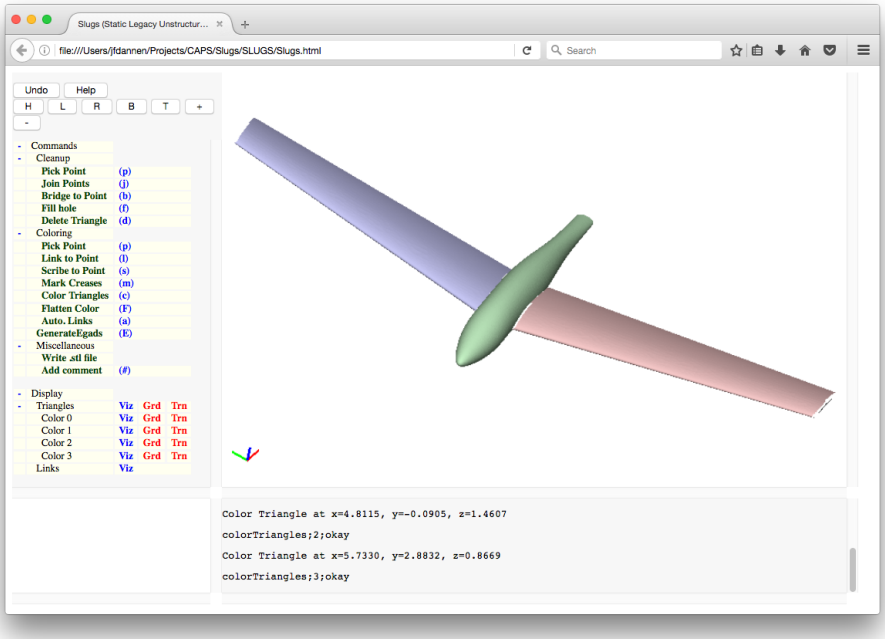


Figure 2. Colored wing-body configuration after automatic crease detector is applied.

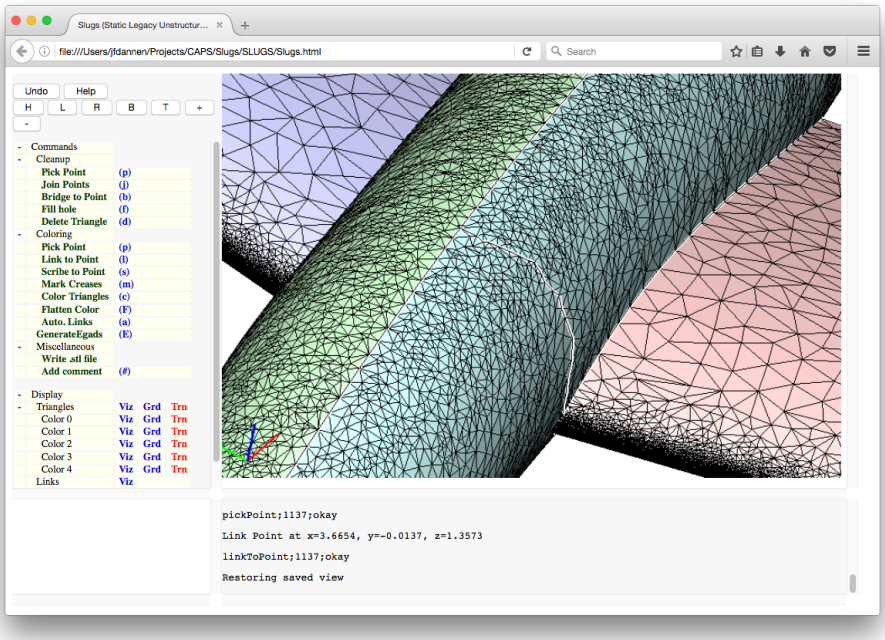


Figure 3. Triangles on fuselage of wing-body configuration, showing nicely aligned triangle sides along the symmetry plane but a jagged set of triangle sides along the fuselage surface.

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

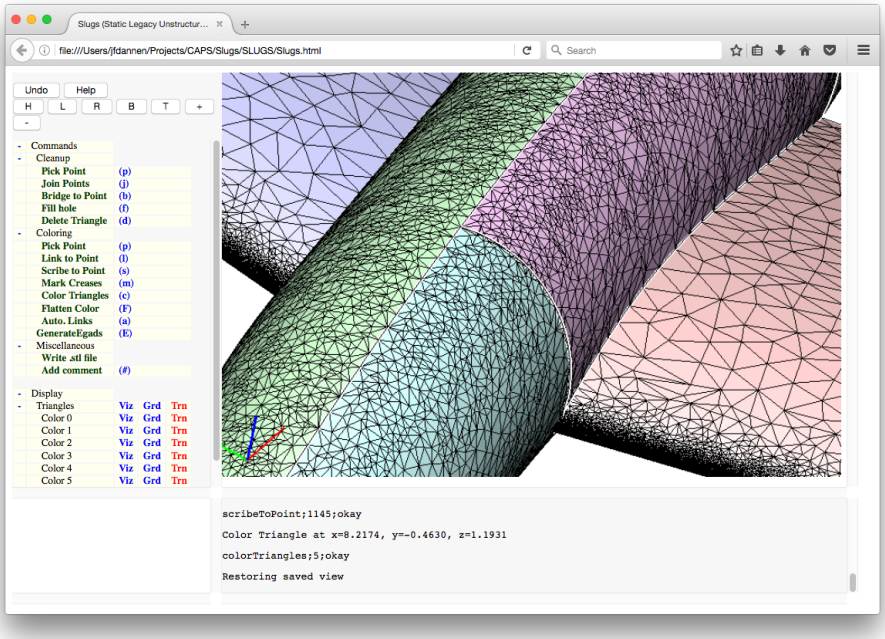


Figure 4. Triangles on fuselage of wing-body after scribing.

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_{tri} triangles, where n_{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_{tri} triangles in the octant. It was found experimentally that $n_{\text{tri}} = 250$ yields the fastest scribes.

IV. Creating B-spline fits for Edges

To create BRep Edges, one needs a collection of points to which a B-spline must be fit. (In anticipation of the Face fitting below, we are going to assume that the points in the cloud are not ordered, except that the two boundary (Node) points are known.)

The fitting process is an optimization problem that can be efficiently solved using the Levenberg-Marquardt algorithm,[6,7] which is a combination of the gradient descent and Newton methods. Levenberg and Marquardt combined the two methods with a damping parameter, λ, which is automatically adjusted during the optimization process, as described below.

Denote the n points in the cloud as (x_i, y_i, z_i) for $i = 1, \ldots, n$; these are known ahead of time. Also denote the m interior control points in the (resulting) B-spline to be (X_j, Y_j, Z_j) for $j = 1, \ldots, m$; these will be determined during the optimization process. Note that the B-spline will actually have $m + 2$ control points, but that the locations of the first and last control points coincide with the known boundary points in the cloud, or $X_0 = x_1$, $Y_0 = y_1$, $Z_0 = z_1$, $X_{m+1} = x_n$, $Y_{m+1} = y_n$, and $Z_{m+1} = z_n$. These boundary conditions are required to ensure that the ends of the various Edges exactly match at the Nodes of the resulting BRep.

The design variable vector for the optimization problem, $\vec{\beta}$, is given by

$$\vec{\beta} \equiv [X_1, Y_1, Z_1, \ldots, X_m, Y_m, Z_m, t_1, \ldots, t_n]$$

The first part consists of the locations of the interior B-spline control points (X_j, Y_j, Z_j) for $j = 1, \ldots, m$; the second part consists of are the parametric coordinates, t_i, associated with each of the points in the cloud.

The objective function to be minimized, which corresponds to the sum of the squares of the distances of the points in the cloud to the B-spline, is defined by

$$S(\vec{\beta}) \equiv \sum_{i=1}^{n} \left(q_i^2 + q_{i+n}^2 + q_{i+2n}^2 \right)$$

where

$$\begin{aligned} q_i &= B_x(\vec{X}, t_i) - x_i \\ q_{i+n} &= B_y(\vec{Y}, t_i) - y_i \\ q_{i+2n} &= B_z(\vec{Z}, t_i) - z_i \end{aligned}$$

for $i = 1, \ldots, n$. Here, for example, $\vec{X} \equiv [X_1, \ldots, X_m]$ and $B_x(\vec{X}, t_i)$ is the evaluation at t_i of the B-spline whose control points are $[X_0, \vec{X}, X_{m+1}]$.

In order to use the Levenberg-Marquardt algorithm, one will need to define the Jacobian matrix, J, where $J_{(i,j)} \equiv \partial q_i / \partial \beta_j$. Note that the first $3m$ columns of J are full (because the control point locations affect most of the B-spline evaluations) but that the last n columns are block-diagonal (because each parametric coordinate t_i only effects the B-spline evaluations for point i). This causes the whole J matrix to be quite sparse, which will make computations with it quite efficient, since in general $n >> m$.

The Levenberg-Marquardt (LM) algorithm efficiently optimizes this kind of least-squares problem with the following process:

- define an initial set of design parameters $\vec{\beta}^{(0)}$ (where the superscript denotes the optimization step)

- compute the objective function, $S^{(0)}$, which is based upon $\vec{\beta}^{(0)}$
- initialize the LM damping parameter $\lambda = 1$
- for each step, $k = 1, \ldots, k_{\max}$, of the optimization
  - compute a provisional step, $\vec{\delta}$ by solving the matrix equation

$$\left[J^T \cdot J - \lambda \cdot \mathrm{diag}(J^T \cdot J)\right] \vec{\delta} = -(J^T \cdot \vec{q})$$

    where the matrix equation is solved efficiently using a conjugate gradient technique[8] because of the sparsity of J and J^T.
  - compute the objective function if the new step were to be taken

$$S^{\mathrm{temp}} = S(\vec{\beta}^{(k-1)} + \vec{\delta})$$

  - if $S^{\mathrm{temp}} < S^{(k-1)}$
    - accept the step, making $\vec{\beta}^{(k)} = \vec{\beta}^{(k-1)} + \vec{\delta}$
    - halve λ (which makes the next step more Newton-like)
  - otherwise
    - reject the step, making $\vec{\beta}^{(k)} = \vec{\beta}^{(k-1)}$
    - double λ (which makes the next step more gradient-descent-like)
  - stop if $||\vec{\delta}||$ is smaller than some user-specified tolerance

In order make sure that the control polygon does not get tangled, the process is executed twice, where in the first pass the control polygon is initially assumed to be a straight line (between the known endpoints) and in the second pass it is assumed to be evenly-spaced along the B-spline generated in the first pass. In practice, the Levenberg-Marquardt optimizer completes in a few hundred steps, requiring less than a second of CPU time on a modern laptop computer.

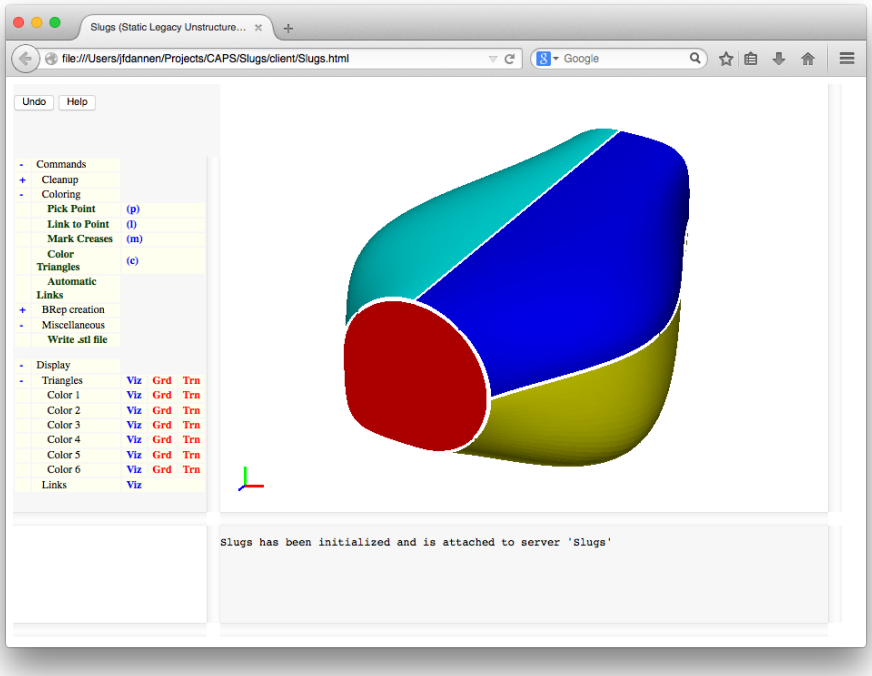
V. Creating B-spline fits for Faces

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, $\vec{\beta}$, consist of the locations of the (two-dimensional array of) interior points in the control net and the parametric coordinate locations (u_i, v_i) associated with each point in the cloud. Cases with up to 5000 points in the cloud and a control net of 11×11 control points are routinely solved in under a minute on a laptop computer. Because of the sparsity of J, it has been found that the time required scales linearly with the number of points in the cloud.

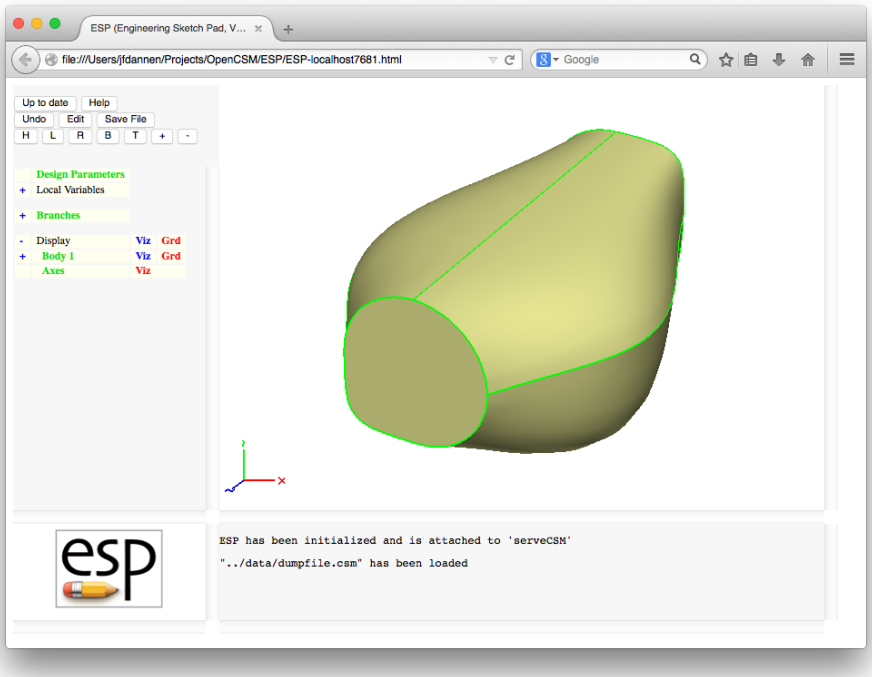
VI. Sample configurations

Fig. 5a shows the colored points and Fig. 5b shows the resulting B-spline surfaces for a simple duct-like configuration. In this case, cubic B-splines with 7 control points along each Edge and 7×7 control points in each Face are used. The RMS error between the points in the cloud and the resulting surface is less than 3×10^{-4} times the body length.

The second demonstration case is a simpler geometry, but it contains a set of triangles that are not fully connected. This case, which is shown in Fig. 6, was set up and executed by the user in under 2 minutes. Again 7×7 control points were used for each Face and the RMS error between the points in the cloud and the resulting surface less than 7.7×10^{-5} times the body length.

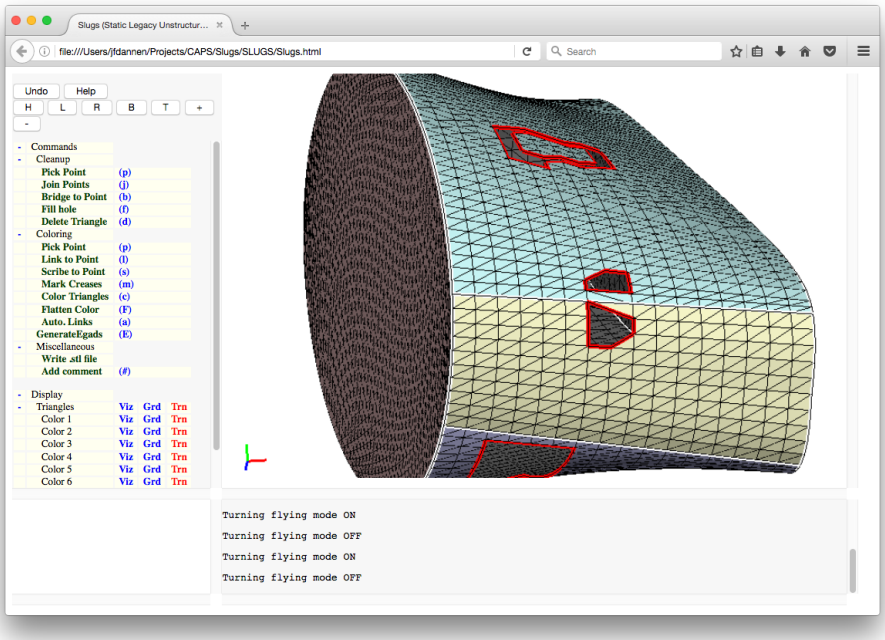


(a) configuration broken into colors

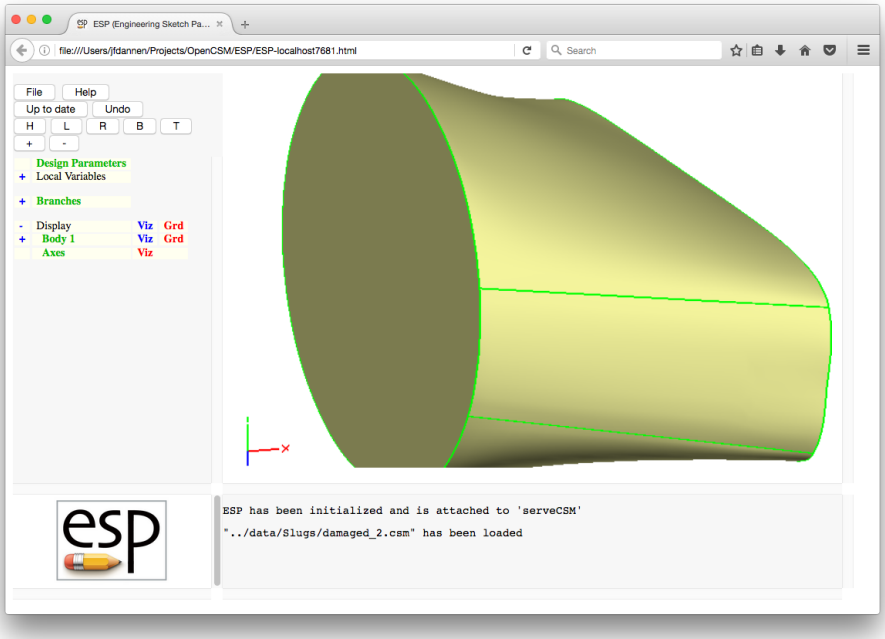


(b) configuration fit with B-spline Edges and Faces

Figure 5. Duct-like configuration



(a) configuration broken into colors



(b) configuration fit with B-spline Edges and Faces

Figure 6. Simpler configuration that shows that the process works even when the triangles are not connected.

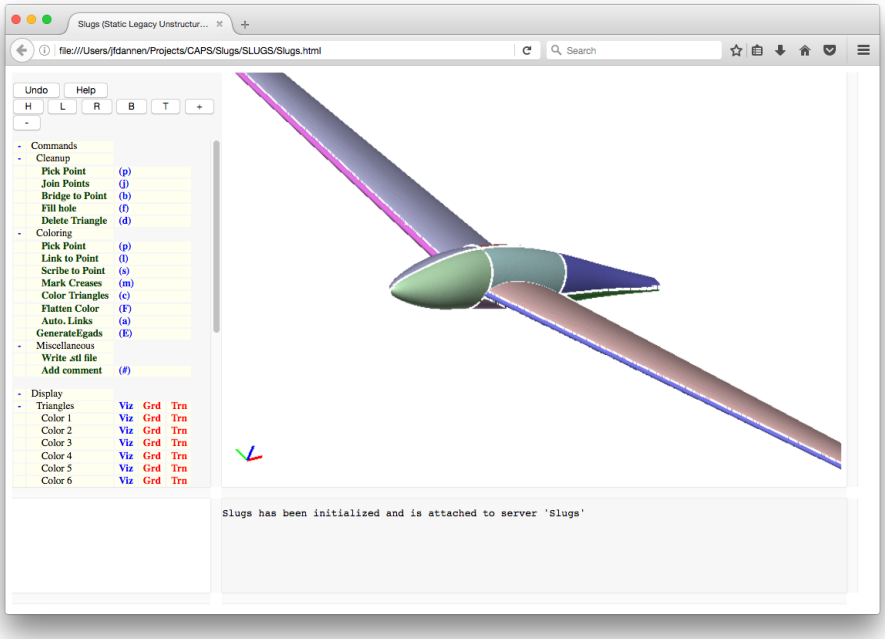
The last demonstration case consists of a wing-body; it is shown in Fig. 7. This case was quite a bit more complicated than the others, and required about 8 minutes to color the points interactively and then generate the BRep. With only 7×7 control points in each Face, the RMS error for this case was 1.5×10^{-5} times the body length.

More complex configurations have been executed to date (but are not shown here). The most complicated case contains about 70 colors and required about 2 hours of the user's time. The most time-consuming part of the process was in designing the BRep, that is, in determining how to break up the configuration into regions that were bounded by only four other colors.

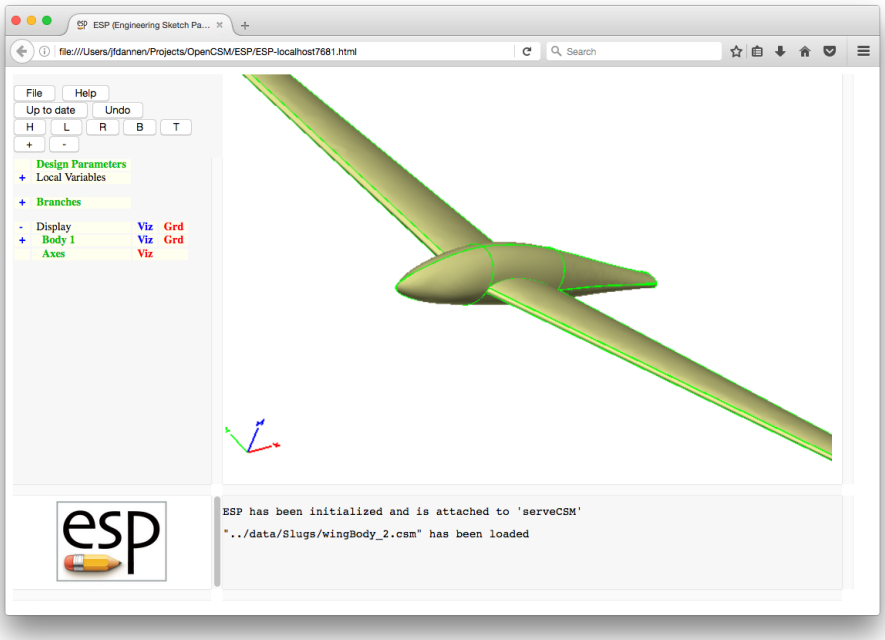
VII. Conclusions

The `SLUGS` system has been applied successfully to several configurations that were described in terms of a cloud of points. In order to limit the extent of the required user interaction, and to make the interaction as rapid as possible, several unique capabilities have been used:

- the separation of the cloud of points into colors is aided by use of the Dijkstra algorithm to quickly mark triangle sides;
- creation of a set of triangle sides that are smooth is aided by a smoothing algorithm, together with a projection technique that is made more efficient through the use of octree searches;
- Levenberg-Marquardt optimization is used to find the B-spline control points (and simultaneously the parametric coordinates associated with each point in the cloud) that minimizes the distances between the cloud and the resulting B-spline;
- sparse matrix techniques are used to improved the efficiency of the Levenberg-Marquardt optimization, making its compute time scale linearly with the number of points in the cloud; and
- the Nodes, Edges, and Faces associated with the BRep are automatically found, based upon the colors associated with the points in the cloud.



(a) configuration broken into colors



(b) configuration fit with B-spline Edges and Faces

Figure 7. Wing-body configuration.

Acknowledgement

This work was funded by the CAPS project, which was funded under AFRL Contract FA8050-14-C-2472: "CAPS: Computational Aircraft Prototype Syntheses"; Ed Alyanak is the Technical Monitor. The author also acknowledges the discussions with Bob Haimes and Pengcheng Jia during the formulation of various parts of this algorithm and during the preparation of this manuscript.

References

[1] "CADfix Fact Sheet", http://www.iti-global.com/cadfix

[2] "Geomagic Fact Sheet", http://www.geomagic.com/en/products-landing-pages/re-designx-wrap

[3] "ScanIP+Nurbs Fact Sheet", https://www.simpleware.com/software/nurbs-module/

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

[6] 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.

[7] 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.

[8] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical recipes in C*, Cambridge University Press, pp 78–81 , 1988.