



**AFRL-RQ-WP-TR-2012-0291**

**ROTATESTL: A MATLAB ROTATION ALGORITHM FOR  
THE ANALYSIS OF COMPUTATIONAL MESHES IN  
STEREOLITHOGRAPHY FILE FORMAT**

**James A. Tancred**

**Aerodynamic Configuration Branch  
Vehicle Aerodynamics Division**

**SEPTEMBER 2012  
Interim Report**

**Approved for public release; distribution unlimited.**

*See additional restrictions described on inside pages*

**STINFO COPY**

**AIR FORCE RESEARCH LABORATORY  
AEROSPACE SYSTEMS DIRECTORATE  
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542  
AIR FORCE MATERIEL COMMAND  
UNITED STATES AIR FORCE**

<b>REPORT DOCUMENTATION PAGE</b>				<i>Form Approved</i> OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. <b>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</b></p>					
<b>1. REPORT DATE (DD-MM-YY)</b> September 2012		<b>2. REPORT TYPE</b> Interim		<b>3. DATES COVERED (From - To)</b> 20 August 2010 – 31 August 2012	
<b>4. TITLE AND SUBTITLE</b> ROTATESTL: A MATLAB ROTATION ALGORITHM FOR THE ANALYSIS OF COMPUTATIONAL MESHES IN STEREO LITHOGRAPHY FILE FORMAT				<b>5a. CONTRACT NUMBER</b> In-house	
				<b>5b. GRANT NUMBER</b>	
				<b>5c. PROGRAM ELEMENT NUMBER</b> 62201F	
<b>6. AUTHOR(S)</b> James A. Tancred				<b>5d. PROJECT NUMBER</b> 2404	
				<b>5e. TASK NUMBER</b> N/A	
				<b>5f. WORK UNIT NUMBER</b> Q05R	
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Aerodynamic Configuration Branch (AFRL/RQAA) Vehicle Aerodynamics Division Air Force Research Laboratory, Aerospace Systems Directorate Wright-Patterson Air Force Base, OH 45433-7542 Air Force Materiel Command, United States Air Force				<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b> AFRL-RQ-WP-TR-2012-0291	
<b>9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b> Air Force Research Laboratory Aerospace Systems Directorate Wright-Patterson Air Force Base, OH 45433-7542 Air Force Materiel Command United States Air Force				<b>10. SPONSORING/MONITORING AGENCY ACRONYM(S)</b> AFRL/RQAA	
				<b>11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)</b> AFRL-RQ-WP-TR-2012-0291	
<b>12. DISTRIBUTION/AVAILABILITY STATEMENT</b> Approved for public release; distribution unlimited.					
<b>13. SUPPLEMENTARY NOTES</b> PA Case Number: 88ABW-2012-5233; Clearance Date: 28 Sep 2012. This report contains color.					
<b>14. ABSTRACT</b> With the continued increase in power of the modern computer, more computational problems are tractable via computer simulation, leading to fast production of data, analysis, and quantitative solutions. Discrete three dimensional analyses, including those involving computational fluid dynamics, now show even more promise as the time required to meet solutions has been shown, in some instances, to be on the order of hours and days rather than weeks or months for the same problem. As a means to exploit such computational power, a MATLAB algorithm has been developed for use with computational meshes in applications requiring the rotation of a mesh about an axis. The algorithm, called ROTATESTL, has been developed such that any number of original mesh files in stereolithography file format can be rotated by any number of angular deflections about specified rotation axes. The algorithm is platform portable and runs on both Windows and Linux operating systems supporting MATLAB 7.					
<b>15. SUBJECT TERMS</b> vehicle configuration design, aero database, rapid prototyping, CART3D, CFD, stereolithography, STL, standard tessellation language					
<b>16. SECURITY CLASSIFICATION OF:</b>			<b>17. LIMITATION OF ABSTRACT:</b> SAR	<b>18. NUMBER OF PAGES</b> 74	<b>19a. NAME OF RESPONSIBLE PERSON (Monitor)</b> James H. Miller
<b>a. REPORT</b> Unclassified	<b>b. ABSTRACT</b> Unclassified	<b>c. THIS PAGE</b> Unclassified			

## Table of Contents

Table of Figures .....	ii
List of Tables .....	iv
1.0. Introduction.....	1
1.1. Program Description and Portability.....	1
1.2. The ASCII Stereolithography File Format.....	1
2.0. Program Functionality .....	5
2.1. The <i>runX</i> Folder System .....	6
2.2. Setting the Path for <i>rotatestl</i> .....	6
2.3. The Help Command .....	8
2.4. Algorithm Coordinate System.....	9
3.0. Input Descriptions.....	10
3.1. Input Argument Syntax .....	10
3.2. Input Designator: <i>world</i> .....	11
3.3. Input Designator: <i>parentFN</i> .....	12
3.4. Input Designator: <i>childrenFN</i> .....	15
3.5. Input Designator: <i>hp1FN</i> or <i>hp2FN</i> .....	16
3.6. Input Designator: <i>deflectionsFN</i> .....	19
3.7. Input Designator: <i>visualflagFN</i> .....	27
3.8. Input Designator: <i>reverseflagFN</i> .....	35
3.9. Input Designator: <i>delimiter</i> .....	37
4.0. Sample Program Execution and Results.....	39
5.0. Conclusion .....	45
6.0. References.....	46
Bibliography .....	47
Appendix – Source Code for <i>rotatestl</i> .....	48
LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS .....	64

## List of Figures

1. Beginning of ASCII STL Text File with Example Geometry .....	2
2. Input and Output Work Flow for <code>rotatestl</code> .....	7
3. Excerpt from Help Menu Invoked at MATLAB Command Prompt.....	8
4. Coordinate System Convention Used by <code>rotatestl</code> .....	9
5. Generic Hypersonic Vehicle Geometry for the Illustration of STL Mesh Rotation.....	10
6. Example Input Argument of World Origin to <code>rotatestl</code> .....	11
7. Example Input Argument for Input Designator, <code>parentFN</code> .....	12
8. The Five Example Control Surfaces of a Generic Hypersonic Vehicle .....	14
9. Actual Body Flap Geometry from <code>Bflap.stl</code> for Input to <code>rotatestl</code> .....	15
10. Example Input Argument for Input Designator, <code>childrenFN</code> .....	15
11. Example Input Argument for Input Designators <code>hp1FN</code> and <code>hp2FN</code> .....	17
12. Display of Hinge Line and Point <code>hp1</code> for Example Body Flap.....	19
13. Example Input Argument for Input Designator, <code>deflectionsFN</code> .....	19
14. General Input Format for <code>deflectionsFN</code> File.....	21
15. Unordered Input Including Zeros along Columns of <code>deflectionsFN</code> File.....	23
16. Unordered Input with Repeated Values along Columns of <code>deflectionsFN</code> File .....	24
17. Ordered Input with Comma Delimiter for <code>deflectionsFN</code> File.....	25
18. Ordered Input with Zero Values and Comma Delimiter for <code>deflectionsFN</code> File.....	26
19. Example Input Argument for Input Designator <code>visualflagFN</code> .....	27
20. Initial Display of Rotated Body Flap for a Generic Hypersonic Vehicle.....	28
21. Initial Display of Rotated Left Elevon for a Generic Hypersonic Vehicle.....	29
22. Description of Figure Legend for Child STL Body Flap.....	30
23. Zoom, Pan, and Rotate Commands of MATLAB Figures .....	30
24. Zoom In Command Enabled in MATLAB Figure Display .....	31
25. Creating a Box around the Child STL Body Flap with the Zoom Command .....	32
26. Initial Zoomed Image of Child STL Body Flap.....	32
27. Rotate 3D Command Enabled in MATLAB Figure Display.....	33
28. Example Rotation of Body Flap .....	33
29. Pan Command Enabled in MATLAB Figure Display .....	34
30. Example Pan of Body Flap .....	34
31. Example Input Argument for Input Designator, <code>reverseflagFN</code> .....	35
32. Example Set-up of <code>reverseflagFN</code> File .....	36
33. Example of Rotation Reversal for +20° Rotation of <code>Bflap.stl</code> File.....	37
34. Example Input Argument for Input Designator, <code>delimiter</code> .....	38
35. Contents of Input Files for Program Execution Example.....	40
36. Command Window Output upon Successful Execution of <code>rotatestl</code> .....	41
37. Folder System Created after Execution of <code>rotatestl</code> .....	41

## List of Figures (Concluded)

38. Child STL Files Created after Successful Execution of <code>rotatestl</code> .....	42
39. Generic Hypersonic Vehicle STL Geometry with Rotated Control Surfaces .....	43

## List of Tables

1. Input Argument Descriptions for <code>rotatest1</code> .....	5
2. Format for <code>hp1FN</code> and <code>hp2FN</code> Input Files .....	18
3. Example Hinge Point Files for Generic Hypersonic Vehicle .....	18
4. Variable Descriptions for <code>deflectionsFN</code> File Format .....	20
5. Summary of Visualization Choice Input to <code>visualflagFN</code> File .....	27
6. Example Set-up of <code>visualflagFN</code> File .....	28
7. Summary of Reverse Flag Input to <code>reverseflagFN</code> File .....	36

## 1.0. Introduction

The use of grids and meshes to transfer geometric information is a basic requirement for many three dimensional analyses, including flow analysis via computational fluid dynamics (CFD) and finite element analysis (FEA) of structural systems and components. In some cases, a part (or all) of the mesh at hand requires rotation such that a new geometry may be subject to analysis. A simple example may include the analysis of flow about a flight vehicle mesh with control effectors deflected in some configuration. If the originally constructed mesh of the vehicle geometry has only control surfaces with no deflection, then one of two feasible options would be available to analyze deflected control surface configurations: either manual reconstruction of the mesh would be required or the surfaces could be rotated separately in an automatic process. The latter option is possible with the use of a recently developed rotation algorithm called `rotatestl`.

### 1.1. Program Description and Portability

The rotation algorithm, `rotatestl`, is a MATLAB<sup>®</sup>-implemented, platform-portable m-file (MATLAB file format) script that takes any number of original mesh files in Stereolithography (STL) format and rotates them by any number of angular deflections about specified rotation axes. The algorithm (named explicitly as `rotatestl.m`) is designed to run on any operating system supporting MATLAB 7. It was written in MATLAB 7.11.0 (R2010b) and has been successfully tested on 32-bit Windows XP (Service Pack 3) and Red Hat Enterprise Linux 5.2 operating systems. Input is provided through ASCII formatted files to further enable platform portability. The software description that follows assumes that the user is familiar with the general use of MATLAB commands, syntax, and operation.

### 1.2. The ASCII Stereolithography File Format

The rotation algorithm requires the ASCII Stereolithography (STL) file format for geometric input of a given computational mesh. This particular file format was chosen as the main geometric input to the program due to its wide use in manufacturing and rapid prototyping (RP) systems. In fact, it is considered the standard file format for the transfer of geometric information from computer-aided design (CAD) packages to many RP systems.<sup>(1)</sup> Despite some of the formatting disadvantages of STL files—such as duplication of mesh nodes, unordered tessellated facets, and the absence of topological information<sup>(1)</sup>—its format is rather simple and is widely accepted by many RP systems. Therefore, its use in computational applications permits the applications themselves to be subject to the multitude of tools and documentation readily available for STL geometric data transfer.

The STL file is a triangular approximation to the topological geometry produced by a CAD system or other topological data-generating software.<sup>(1)</sup> The surface of the geometry is approximated by triangles. The individual surface of each triangle is denoted as a *facet* or *face*, and the triangle vertices are denoted as *nodes*. Within the STL file, the geometric data may be written in binary or in ASCII format. The binary format tends to be more compact and may load faster onto RP machines or into various software packages that can read an STL file. However, unlike the binary format, the ASCII STL format is human-readable, which enables users to create simple parsing scripts in various programming languages for reading and writing the ASCII STL file directly without the need to interpret binary data. This approach to reading and writing the ASCII STL file was implemented in `rotatestl`. A general convention of the ASCII STL file is standardized. For the purposes of this manual, a simple description of the file format is discussed.

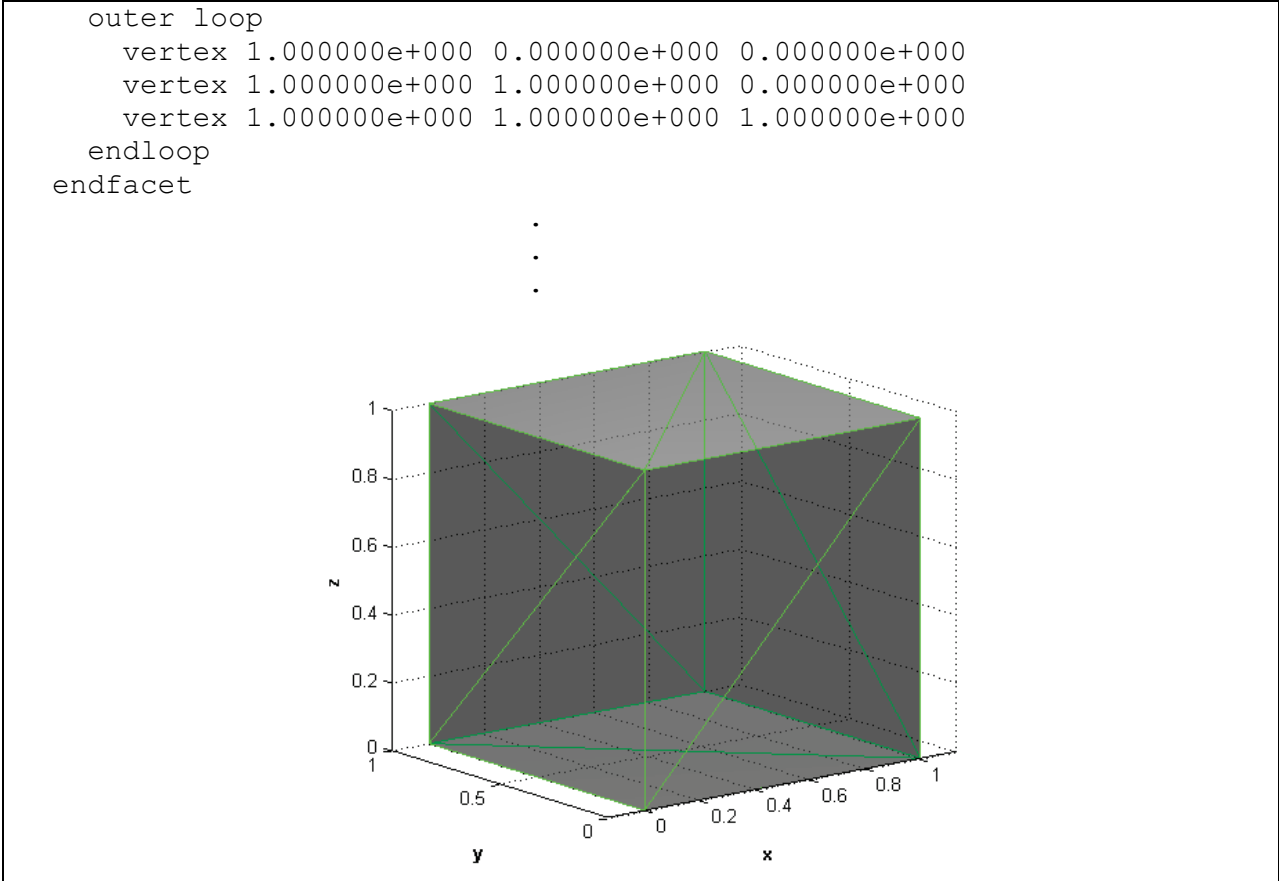
The extension of the STL file is `<.stl>`. The extensions in which `rotatestl` accepts are `<.stl>` and `<.STL>`. If a different file extension is included with a file name input to `rotatestl`, an error prompts the user to check the extension (more will be discussed on user input later).

There are standard text indicators within the ASCII STL file that direct any parsing algorithm about what is to follow in the file as it is read. As an example, the beginning of an ASCII STL file for a simple cube of edge length unity, approximated with twelve triangles, is shown below in Figure 1. The first line of the STL file always starts with the word, `solid`.

```
solid OBJECT
  facet normal 0.000000e+000 -1.000000e+000 0.000000e+000
    outer loop
      vertex 0.000000e+000 0.000000e+000 0.000000e+000
      vertex 1.000000e+000 0.000000e+000 0.000000e+000
      vertex 1.000000e+000 0.000000e+000 1.000000e+000
    endloop
  endfacet
  facet normal 0.000000e+000 -1.000000e+000 0.000000e+000
    outer loop
      vertex 1.000000e+000 0.000000e+000 1.000000e+000
      vertex 0.000000e+000 0.000000e+000 1.000000e+000
      vertex 0.000000e+000 0.000000e+000 0.000000e+000
    endloop
  endfacet
  facet normal 1.000000e+000 0.000000e+000 0.000000e+000
```

**Figure 1. Beginning of ASCII STL Text File with Example Geometry**





**Figure 1. Beginning of ASCII STL Text File with Example Geometry (Concluded)**

It is followed by any general, descriptive text written by the software or machine that creates the STL file. In the case of the cube above, the descriptive text is given as `OBJECT`. For the use of `rotatestl`, there should be no space or whitespace within this descriptive text. Otherwise, the parsing algorithm within `rotatestl` improperly reads the STL at hand. For instance, if the first line of the STL file reads

```
solid OBJECT Cube
```

the parsing algorithm of `rotatestl` would interpret `Cube` as the next set of information to read instead of incorporating the entire set of text `<OBJECT Cube>` as one entity. However, if the first line reads

```
solid OBJECT_Cube
```

then the parsing algorithm would read the STL file properly, assuming the rest of the file is properly formatted.

Formatting of the STL file continues with a listing of each facet or face (i.e. each triangle) describing the geometry. A single facet of the geometry has the following format:

```

facet normal nx ny nz
  outer loop
    vertex xv1 yv1 zv1
    vertex xv2 yv2 zv2
    vertex xv3 yv3 zv3
  endloop
endfacet

```

where  $nx$ ,  $ny$ , and  $nz$  are the  $x$ ,  $y$ , and  $z$  components, respectively, of the surface normal vector of the facet and  $xvi$ ,  $yvi$ , and  $zvi$  are the  $x$ ,  $y$ , and  $z$  components, respectively, of each vertex of the triangle for  $i = 1, 2, 3$ . Each triangle of the mesh is listed until the last triangle is accounted within the STL file. For each facet subsection of the file, only text changes occur for  $nx$ ,  $ny$ ,  $nz$ ,  $xvi$ ,  $yvi$ , and  $zvi$ . The indicators `<facet normal>`, `<outer loop>`, `<vertex>`, `<endloop>`, and `<endfacet>` remain the same, as above, throughout the file for each facet of the geometry. The end of the STL file is reached when the term `<endsolid>` is read, as shown below:

```

.
.
.
facet normal 0.000000e+000 0.000000e+000 -1.000000e+000
  outer loop
    vertex 1.000000e+000 1.000000e+000 0.000000e+000
    vertex 1.000000e+000 0.000000e+000 0.000000e+000
    vertex 0.000000e+000 1.000000e+000 0.000000e+000
  endloop
endfacet
facet normal 0.000000e+000 0.000000e+000 1.000000e+000
  outer loop
    vertex 0.000000e+000 0.000000e+000 1.000000e+000
    vertex 1.000000e+000 0.000000e+000 1.000000e+000
    vertex 1.000000e+000 1.000000e+000 1.000000e+000
  endloop
endfacet
facet normal 0.000000e+000 0.000000e+000 1.000000e+000
  outer loop
    vertex 1.000000e+000 1.000000e+000 1.000000e+000
    vertex 0.000000e+000 1.000000e+000 1.000000e+000
    vertex 0.000000e+000 0.000000e+000 1.000000e+000
  endloop
endfacet
endsolid OBJECT          <----- End of STL reached

```

The descriptive text `OBJECT` is once again listed after `<endsolid>`, and the STL file ends.

The algorithm for `rotatestl` does not check to ensure that the continuity of the geometry from the STL file is valid without gaps, overlapping or degenerate facets, etc. Verification of properly posed and formatted geometry within the ASCII STL file is left to the user. Any STL in which `rotatestl` reads successfully will proceed with the rotation algorithm, regardless of the state of the geometry within the STL.

## 2.0. Program Functionality

The functionality of `rotatestl` is rather straightforward: input geometric and user-defined information, perform user-specified rotations, and output the resulting geometry in Stereolithography file format. All input files are written in ASCII formatted text to enable platform portability from one operating system to another. Binary input files have not been implemented for `rotatestl`. The general input and output work flow is shown in Figure 2 on page 7. Note that there are nine major input components required for successful program execution, seven of which are the ASCII formatted input files. The output is a file structure containing the newly rotated STL files. Table 1 gives a brief description of each input.

**Table 1. Input Argument Descriptions for `rotatestl`**

Expected Input Argument Order	Input Designator	Input Argument Type	Description
1	<code>world</code>	Floating Point	1 X 3 Vector; World origin of geometry in the form [x, y, z]
2	<code>parentFN</code>	String	File name with extension; refers to file containing original STL file names
3	<code>childrenFN</code>	String	File name with extension; refers to file containing child names used to label rotated STL files
4	<code>hp1FN</code>	String	File name with extension; refers to file containing hinge line coordinates for "first" points on hinge line
5	<code>hp2FN</code>	String	File name with extension; refers to file containing hinge line coordinates for "second" points on hinge line
6	<code>deflectionsFN</code>	String	File name with extension; refers to file containing deflection angles in degrees
7	<code>visualflagFN</code>	String	File name with extension; refers to file containing decision flags to show <u>visualization of rotation</u>
8	<code>reverseflagFN</code>	String	File name with extension; refers to file containing decision flags to reverse direction of rotation
9	<code>delimiter</code>	String	Printable ASCII character; delimiter character used within input files

## 2.1. The *runX* Folder System

Taking note of the output, a file system is generated for each run of `rotatest1`. The top directory of the file system is designated *runX*, where *X* refers to the number of times `rotatest1` has been executed consecutively within the present working directory. The algorithm itself searches in the present working directory for the file *runX* until it cannot be found. For instance, if `rotatest1` is executed for the fifth time, the algorithm searches the present working directory for *run1* through *run4*. Once it determines that *run5* is not in the present working directory, it creates a new system of directories, in the present working directory, with the top directory labeled *run5*. In this way, `rotatest1` does not overwrite files that it generates from previous executions. Further note, however, that if a former *runX* system is deleted manually by the user, a subsequent execution of the program then creates a file system with a label having the same name as the deleted folder system. For example, in the previous case of executing `rotatest1` for the fifth time, if the file system *run3* were deleted and if the program is executed for the sixth time, a *new* folder system called *run3* would be created. Moreover, a seventh execution in this instance, however, would produce the directory system *run6*, not *run7*, because *run6* does not yet exist. Thus, even though `rotatest1` does not overwrite files, the user must still take care to book-keep executions if *runX* folder systems are deleted or labeled differently during consecutive runs to prevent confusion among run folders.

The *runX* folder system itself is composed of subdirectories containing the rotated STL files produced by an execution of `rotatest1`. There is precisely the same number of *subdirectories* as there are original STL files. That number is noted as *N* in the output section of Figure 2. Within each subdirectory is the storage of each rotated STL file derived from an original “parent” STL file. The number of “children” STL files in each subdirectory depends upon how many rotations the user has specified for a given “parent” STL file.

## 2.2. Setting the Path for `rotatest1`

The execution of `rotatest1` may take place locally in the present working directory or may be invoked from a designated path set by the user. In either case, the *runX* folder is written to the present working directory. A path for the algorithm may be set manually through MATLAB by using either the `addpath` or `pathtool` commands.

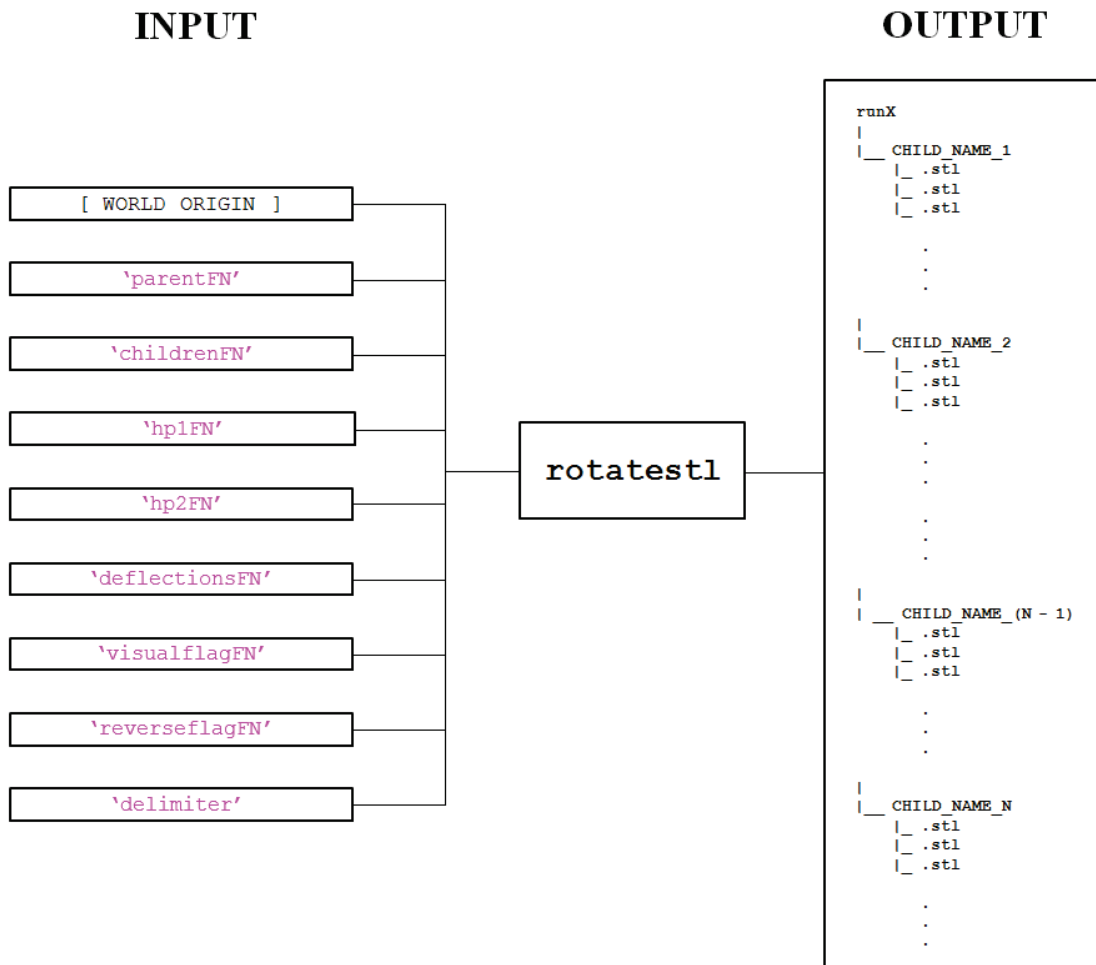
To create a temporary path that expires after the current session of MATLAB is closed, the `addpath` command may be used. First, place `rotatest1.m` in the desired directory from which it is to be invoked. Then execute the following command at the MATLAB command prompt:

```
>> addpath('YOUR_FAVORITE_PATH')
```

Place any path within the quotes, including directories with white space. For example, if `rotatestl.m` were placed in the folder `<C:\Tools\Rotatestl>` on a Windows operating system, then the following would be entered using the `addpath` command in MATLAB:

```
>> addpath('C:\Tools\Rotatestl\')
```

For a path that is remembered automatically by MATLAB at every startup, use the `pathtool` command. Please see the MATLAB documentation for more information on setting the path for a given operating system.



**Figure 2. Input and Output Work Flow for rotatestl**

## 2.3. The Help Command

In general, for a quick reference to questions about work flow or input and output while running `rotatestl`, a general help menu may be invoked at the command prompt of MATLAB. At the command prompt simply type

```
>> help rotatestl
```

A path must be established for `rotatestl` (or the program must be in the present working directory) for the help menu to appear. A general help menu prints to the screen. The first section of the help menu is shown below in Figure 3:

```
ROTATESTL      STL file rotation algorithm
ROTATESTL(world,parentFN,childrenFN,hp1FN,hp2FN,deflectionsFN, ...
            visualflagFN,reverseflagFN,delimiter)

transforms a stereolithography (STL) file via rotation angles
specified by the user. The user provides input through files in
ASCII format.

For each execution, ROTATESTL will generate a run folder called
"runX," where X is an integer counting each run. For example, if
ROTATESTL is executed for the 5th time, the folder "run5" will
be generated. Within each runX folder, a sub-folder, containing
rotated STL files, will be created for each parent STL file.
Therefore, the file structure of the runX folder becomes

        runX
        |
        |__ CHILD_NAME_1
        |
        |__ CHILD_NAME_2
        .
        .
        .
        |
        |__ CHILD_NAME_N

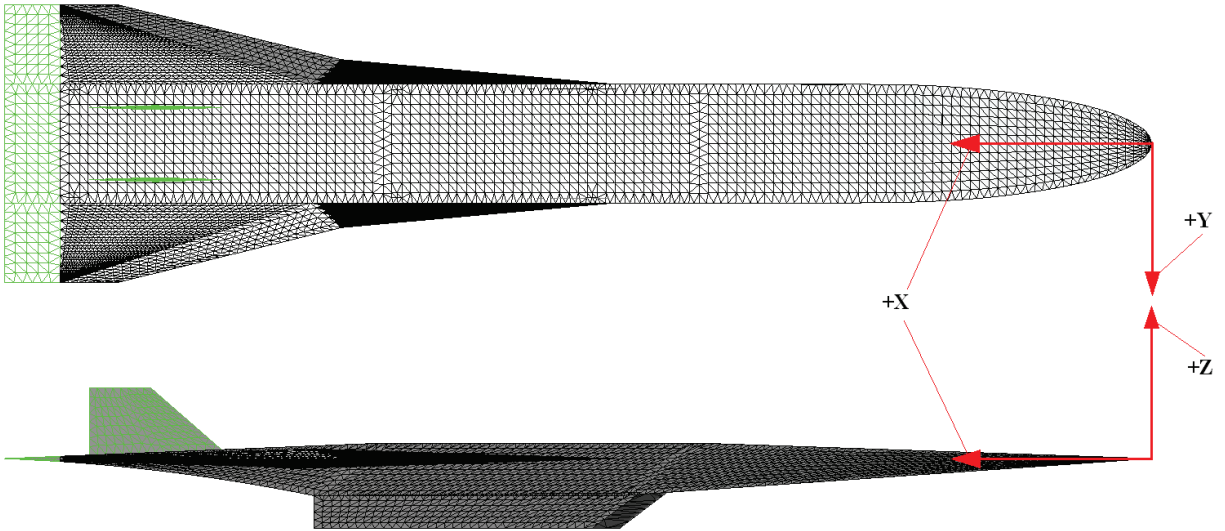
ROTATESTL will not overwrite a runX folder that all ready
exists.

When rotations are performed, a rotation of zero is not
executed (even if a zero is specified as a desired rotation),
as this would result in a file with points identical to those of the
parent STL file. Furthermore, if identical rotation angles are
specified for a given parent STL, the rotation angle will only be
recorded once, and only one child STL file will be rotated for that
angle. This enables the user to specify any set of deflections,
whether repeated, non-repeated, or zero, for a parent STL. Only
unique, nonzero rotation angles will be rotated. The user does not
have to sort through a set of rotations. ROTATESTL will sort through
the rotations automatically.
```

**Figure 3. Excerpt from Help Menu Invoked at MATLAB Command Prompt**

## 2.4. Algorithm Coordinate System

The coordinate system used by `rotatestl` is that of body coordinates used in various aerospace engineering applications. In this case, the x-axis is positive aft (from the front of body to the back of the body). The y-axis is positive to the right when looking at the rear of the body. The z-coordinate is the direction of the cross-product of x with y (in most cases, usually vertical). This convention must be used in the geometry exported to STL format for rotations to take place properly and as desired. A diagram of the coordinate system orientation is given in Figure 4.



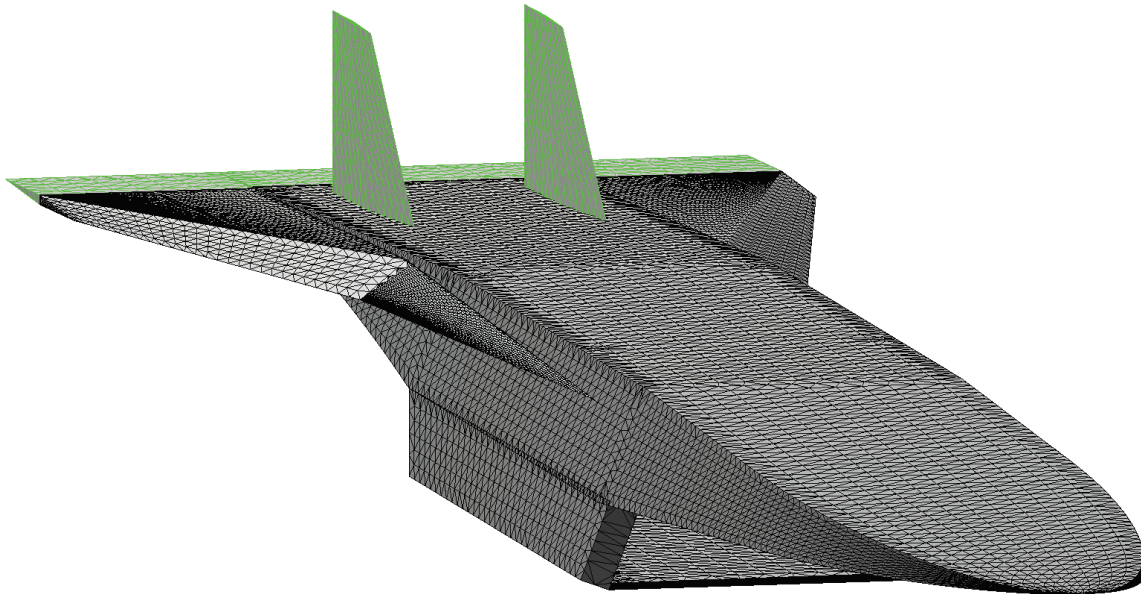
**Figure 4. Coordinate System Convention Used by `rotatestl`**

The units of the STL files are implied by the magnitude of the node components within each file. It is the user's responsibility to ensure that appropriate units are associated with each original STL file. Conversion from one unit type to another is not implemented in `rotatestl`.

### 3.0. Input Descriptions

There are nine major input arguments to `rotatestl`. Each argument must be passed to `rotatestl` for proper execution of the program. The set-up of ASCII file input must also be prepared correctly for successful execution. Input files are placed in the present working directory. Descriptions of each input argument and ASCII file set-up are discussed next.

As a running example to illustrate input and output to `rotatestl`, the mesh of a generic hypersonic flight vehicle with various control surfaces is visualized throughout the program description. A picture of the vehicle mesh is given in Figure 5. Control surfaces are shown in green. The intent of the example is to show rotations of the vehicle control surfaces, which emphasize the main capability of `rotatestl`: to rotate tessellated geometric data in STL format about a desired rotation axis.



**Figure 5. Generic Hypersonic Vehicle Geometry for the Illustration of STL Mesh Rotation**

### 3.1. Input Argument Syntax

Input arguments to `rotatestl` are passed in a function format. The nine input argument types listed in Table 1 are passed to `rotatestl` for program execution. The general syntax for input arguments takes the following form:

```
rotatestl(world,parentFN,childrenFN,hp1FN,hp2FN,deflectionsFN,visualflagFN,reverseflagFN,delimiter)
```

where each input designator above must be of input argument type as listed in Table 1. Note that



strings in MATLAB are entered with single quotes. For example, a set of rotations are performed by entering the following at the MATLAB command prompt or in an m-file:

```
rotatestl([0,0,0], 'parents.txt', 'children.txt', 'hp1.txt', 'hp2.txt', ...  
'deflections.txt', 'visualflag.txt', 'reverseflag.txt', ',');
```

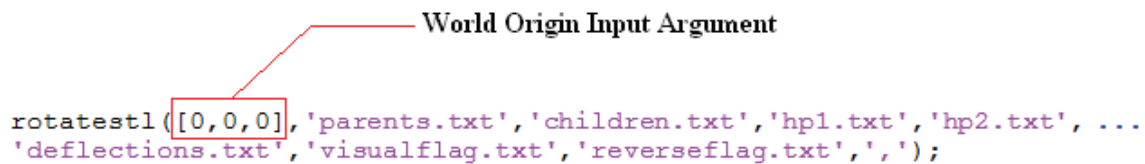
Further note that all string input arguments, except the delimiter, are file names. The full file name with extension should be included as input. Once again, the format of the files should be in ASCII format (complete format to be discussed later). The extension to each input argument file name does not necessarily matter, as long as the files to which the input strings refer are formatted with printable ASCII (plain text) characters. The ellipsis character above `<...>` is simply MATLAB syntax that continues a command to the next line.

### 3.2. Input Designator: `world`

Input designator `world` refers to the world origin. The world origin of the geometry passed to `rotatestl` must be specified. Use the coordinate system convention discussed previously for proper origin input. The algorithm assumes that the entered origin pertains to all files that it reads and writes for a given execution. Input itself is a 1 X 3 row vector, where the first, second, and third elements are the x, y, and z components, respectively, of the world origin entered as floating point data types. As in the above example regarding syntax, the world origin is given by

```
[0,0,0]
```

within the function call to `rotatestl`:



The diagram shows a MATLAB function call to `rotatestl`. The first argument, `[0,0,0]`, is enclosed in a red box. A red line extends from the top-right corner of this box to the text "World Origin Input Argument" located above and to the right of the box. The rest of the function call is: `, 'parents.txt', 'children.txt', 'hp1.txt', 'hp2.txt', ... 'deflections.txt', 'visualflag.txt', 'reverseflag.txt', ',');`

**Figure 6. Example Input Argument of World Origin to `rotatestl`**

The origin does not have to be `[0,0,0]`. It can be any origin, as long as the same origin is used for each original STL file read by the program.

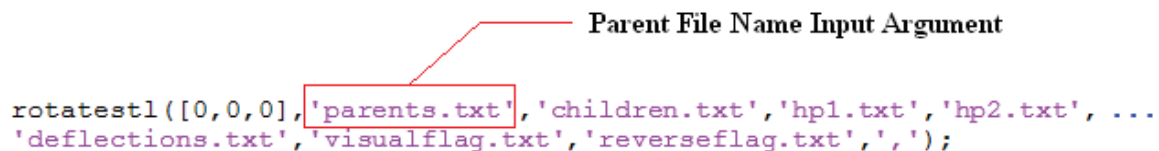
### 3.3. Input Designator: `parentFN`

The input designator `parentFN` implies “parent file name.” The corresponding input argument for `parentFN` is a string of a file name. The file to which it refers is an ASCII file in the present working directory containing all file names of the original STL files to be rotated.

Using the syntax example and the generic hypersonic vehicle as a guide, a notional input argument to `rotatestl` is

```
'parents.txt'
```

within the `rotatestl` function call:



```
rotatestl([0,0,0], 'parents.txt', 'children.txt', 'hp1.txt', 'hp2.txt', ...  
'deflections.txt', 'visualflag.txt', 'reverseflag.txt', ', ');
```

Parent File Name Input Argument

**Figure 7. Example Input Argument for Input Designator, `parentFN`**

The file itself, `parents.txt`, would have the following syntax for the rotation of the various control surfaces of the hypersonic vehicle in Figure 5:

```
Bflap.stl  
Lelevon.stl  
Relevon.stl  
Ltail.stl  
Rtail.stl
```

where each line of the file contains the file name of each original, or “parent,” STL file prepared for rotation. In this case, a body flap (`Bflap.stl`), left and right elevons (`Lelevon.stl` and `Relevon.stl`), and left and right vertical tails (`Ltail.stl` and `Rtail.stl`) are considered for rotation about their hinge lines. Each of the files `parents.txt`, `Bflap.stl`, `Lelevon.stl`, `Relevon.stl`, `Ltail.stl`, and `Rtail.stl` would be placed in the present working directory where the execution of `rotatestl` is to take place.

The rotation algorithm requires a standard syntax for the `parents.txt` file. The extensions listed for each parent STL file should be either `<.stl>` or `<.STL>`. Secondly, there should be only one period or decimal `<.>` within the filename. That is, the period character is reserved exclusively for the extension of each STL file. The program searches for this period character and determines if the remaining extension is properly entered into `parents.txt`. If there is an error in the extensions, such as disorder of characters or improper syntax, `rotatestl` will return an error to the MATLAB command window. For example, if the input to `parents.txt` were given to be

```
Bflap.stl
Lelevon.st
Relevon.Stl
Ltail.tlS
Rtail.STL
```

then `rotatestl` would return the following error message (assuming there are no other errors):

```
++++ Error in file <parents.txt> +++++
File with name [Lelevon.st] does not have an extension of <.stl> or <.STL>.
Please include <.stl> or <.STL> extension to the end of the parent filename.

++++ Error in file <parents.txt> +++++
File with name [Relevon.Stl] does not have an extension of <.stl> or <.STL>.
Please include <.stl> or <.STL> extension to the end of the parent filename.

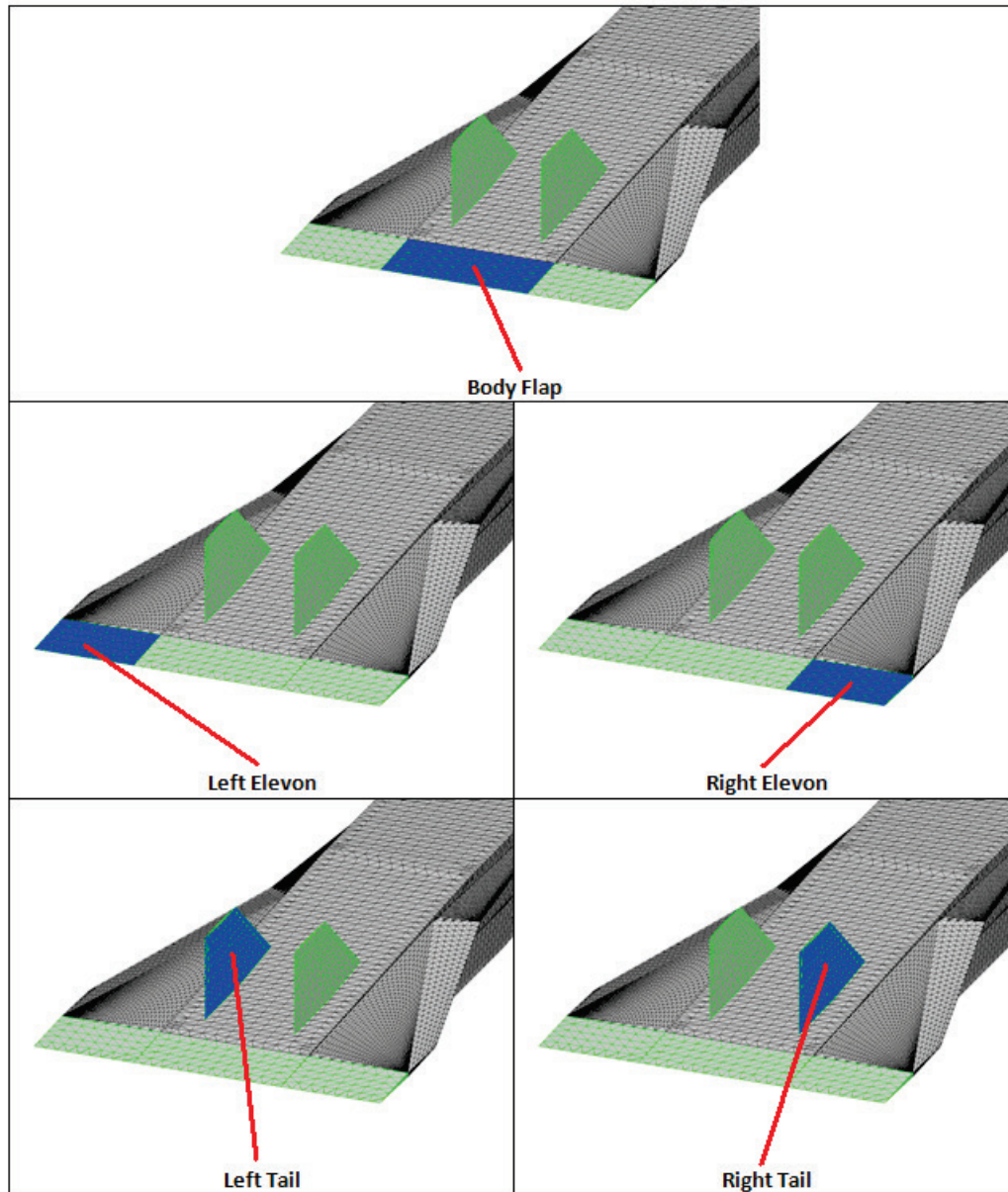
++++ Error in file <parents.txt> +++++
File with name [Ltail.tlS] does not have an extension of <.stl> or <.STL>.
Please include <.stl> or <.STL> extension to the end of the parent filename.

Not all parent STL file names have been input correctly.
Please check file extensions of parent STL file names.
```

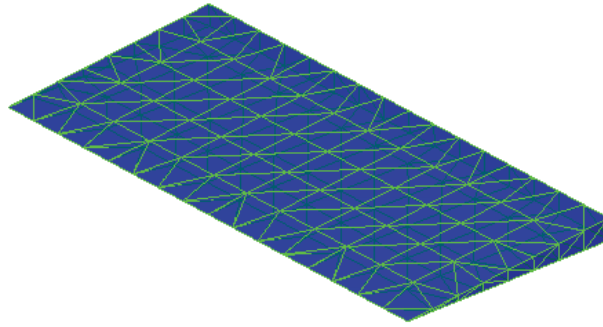
Furthermore, there should be no spaces within file names provided in the `parentFN` file. If there are spaces within the file names, the text on a given line will be merged and interpreted as one file name. The five control surfaces listed in `parents.txt` are highlighted in blue in Figure 8.

It is recommended that each of the STL files in `parents.txt` only contain the geometric information of the specific geometry desired for rotation. Usually, the geometry is a closed surface with no gaps. For example, the STL `Bflap.stl` should only contain the nodes and facets for the body flap. The other control surfaces and body components should not be exported to the file. In this way the user knows exactly what particular geometry is passed to `rotatestl`. The actual geometry to be rotated from `Bflap.stl` is shown in Figure 9 to emphasize this point. The algorithm will rotate any STL that is input to it, even if the geometry is a combination of nodes and facets unwanted by the user. The quality and nature (presence of degenerate facets, gaps, smoothness of the mesh, etc.) of the *input* parent STL files are left to the user to troubleshoot.

The file designated by the `parentFN` input argument sets the stage for the rest of the execution of `rotatestl`. The number of parent STL files listed in `parents.txt`, for instance, governs the amount of information input to `rotatestl`. The necessary constraints on input are discussed with each of the different input arguments that follow.



**Figure 8. The Five Example Control Surfaces of a Generic Hypersonic Vehicle**



**Figure 9. Actual Body Flap Geometry from Bflap.stl for Input to rotatestl**

### 3.4. Input Designator: childrenFN

The input file designator `childrenFN` implies “children file name.” A string argument is passed to `rotatestl` denoting a file name. The file itself is an ASCII text file in the present working directory containing names that label rotated STL files. Each line in `childrenFN` corresponds to a label for the respective line in `parents.txt`. For output STL naming consistency, this line-by-line correspondence between the `childrenFN` file and the `parentFN` file should be enforced.

Any text may be entered as a label for each line of `childrenFN`, provided that the text does not violate file name restrictions of the operating system. An example input argument is

```
'children.txt'
```

and is boxed below in the `rotatestl` function call:

**Children File Name Input Argument**

```
rotatestl([0,0,0], 'parents.txt', 'children.txt', 'hp1.txt', 'hp2.txt', ...
'deflections.txt', 'visualflag.txt', 'reverseflag.txt', ',');
```

**Figure 10. Example Input Argument for Input Designator, childrenFN**

The file `children.txt` to which the above input argument denotes could contain the following format for the example of the generic hypersonic vehicle:

```
bflap
Lelevon
Relevon
Ltail
Rtail
```

After a rotation is executed, `rotatestl` creates a new file name each rotated STL file of the form

```
<CHILD_NAME><DEFLECTION ANGLE><deg.stl>
```

For example, if the file `Bflap.stl` is rotated by -10.5 degrees, the resulting rotated file name becomes

```
bflap-10.5deg.stl
```

The newly created STL file is then placed in a respective folder called `bflap` in the *runX* file system. In a similar manner, the other input STL files are passed to `rotatestl`, and the resulting output files are placed in the respective subfolders of `bflap`, `Lelevon`, `Relevon`, `Ltail`, and `Rtail` within the *runX* file structure.

Note that there are five label names in `children.txt` and that there are five input STL files listed in `parents.txt`. There should be the same number of label names in the `childrenFN` file as the number of parent STL files input for rotation. If this is not the case, `rotatestl` will prompt the user with an error similar to the following:

```
The number of parent .stl files provided determines the
number of specified child names, the sizes of the hinge
point matrices, and the size of the deflection structure.
The number of rows in the hinge point matrices, the
number of columns in the deflection structure, and the
number of child names specified must be equal to the number
of parent .stl files provided.
```

```
Number of specified names for child .stl files
not equal to the number of parent .stl files provided.
```

In this case, the prompt simply tells the user that the number of labels in the `childrenFN` file is not equal to the total STL files provided. Like the `parentFN` file, the `childrenFN` file must contain names with no spaces within the name along a given line of input.

### 3.5. Input Designator: `hp1FN` or `hp2FN`

The input designator `hp1FN` or `hp2FN` implies “hinge point [1 or 2] file name.” The input argument to `rotatestl` is a string denoting one of two file names. The file denoted by the input argument is one of two ASCII files in the present working directory that contains the Cartesian coordinates of points along a specified rotation axis.

For each rotation that is performed, a rotation axis must be specified. One rotation axis must correspond to one input (parent) STL file. Because a line may be defined by two points, two input files are required. One input file contains the Cartesian coordinates of one point on a given hinge line while the other input file contains the coordinates of the second point the hinge line. Example input arguments to `rotatestl` regarding the vehicle in Figure 5 include

`'hp1.txt'`

and

`'hp2.txt'`

within the function call to `rotatestl`:

#### Hinge Point File Name Input Arguments

```
rotatestl([0,0,0], 'parents.txt', 'children.txt', 'hp1.txt', 'hp2.txt', ...  
'deflections.txt', 'visualflag.txt', 'reverseflag.txt', ',,');
```

**Figure 11. Example Input Argument for Input Designators `hp1FN` and `hp2FN`**

The coordinates of the first input file, `hp1.txt`, are denoted collectively as the hinge point 1 matrix, or *hp1*. Similarly, the second set of coordinates of `hp2.txt` is denoted as the hinge point 2 matrix, or *hp2*. The algorithm in `rotatestl` then defines a vector along each given hinge line such that the right hand rule points in the direction of *hp1*. That is, when one's thumb is placed in the direction of *hp1*, a positive rotation of a parent STL follows the motion of one's hand using the right hand rule convention. Furthermore, the order of input arguments four and five determines *hp1* and *hp2*. Input argument four (in this case, `'hp1.txt'`) is always denoted as *hp1* in `rotatestl` while input argument five (in this case, `'hp2.txt'`) is always denoted as *hp2*.

Each row in the hinge point files `hp1.txt` or `hp2.txt` contains, strictly speaking, the hinge point coordinates along the hinge line corresponding to the input STL file of the respective row in the `parentFN` input file. Let each coordinate on a hinge line be defined with the following convention:

$$C_{p,j} \quad (1)$$

where  $C$  denotes a Cartesian coordinate X, Y, or Z; the subscript  $p$  is the denoted point on the hinge line (point 1 or point 2); and the subscript  $j$  is a specific row of the `parentFN` file to which the hinge line corresponds. Therefore, `hp1FN` and `hp2FN` have the form shown in Table 2. Note that  $N$  is the total number of input STL files listed in `parentFN`. By convention of the algorithm, only one hinge line corresponds to one input STL file. Once again, positive rotations follow the right-hand-rule and point in the direction of *hp1*.

Continuing the example with the generic hypersonic vehicle, the input files `hp1.txt` and `hp2.txt` are shown in Table 3 with white space separating each coordinate. When observing

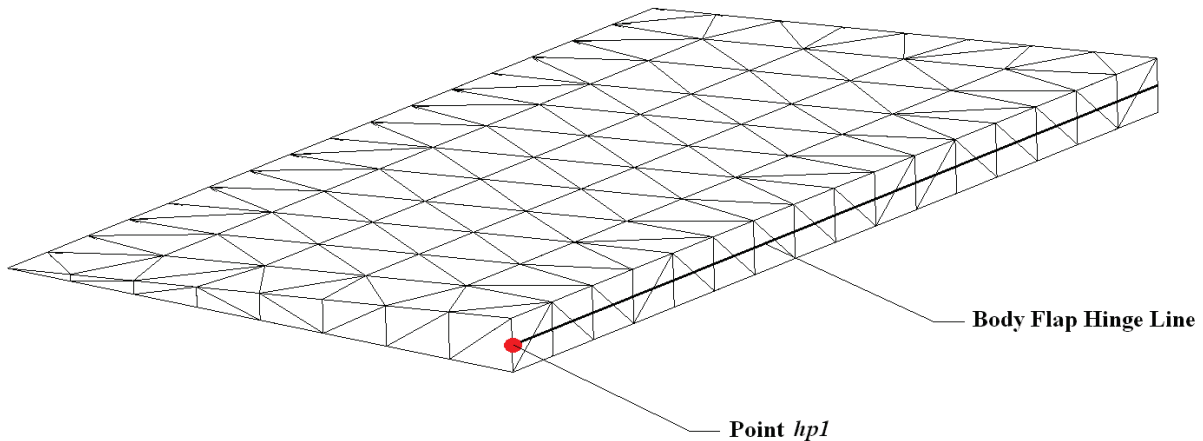
Table 2. Format for hp1FN and hp2FN Input Files					
hp1FN			hp2FN		
LINE 1:	$X_{11}$	$Y_{11}$	$Z_{11}$	LINE 1:	$X_{21}$ $Y_{21}$ $Z_{21}$
LINE 2:	$X_{12}$	$Y_{12}$	$Z_{12}$	LINE 2:	$X_{22}$ $Y_{22}$ $Z_{22}$
		$\vdots$			$\vdots$
LINE N:	$X_{1N}$	$Y_{1N}$	$Z_{1N}$	LINE N:	$X_{2N}$ $Y_{2N}$ $Z_{2N}$

Table 3. Example Hinge Point Files for Generic Hypersonic Vehicle					
hp1.txt:			hp2.txt:		
11.766	0.638	-0.004	11.766	-0.638	-0.004
11.766	-0.638	-9.51685e-6	11.766	-1.500	-9.51685e-6
11.766	1.500	-9.51685e-6	11.766	0.638	-9.51685e-6
10.778	-0.388	1.066	10.737	-0.388	0.067
10.778	0.388	1.066	10.737	0.388	0.067

Table 3, combining Line 1 of `hp1.txt` and Line 1 of `hp2.txt` creates the hinge line for the body flap STL file, `Bflap.stl`, listed on Line 1 of `parents.txt`. Line 2 of `hp1.txt` and Line 2 of `hp2.txt` create the hinge line for the STL file listed on Line 2 of `parents.txt`—in this case `Lelevon.stl`. Other specified hinge lines are created in a similar manner. To visualize a hinge line, the body flap hinge line is shown in Figure 12. The point *hp1* of the body flap hinge line is represented by the red dot on the body flap right side. With this input, a positive rotation (by the right hand rule as previously described) of the body flap creates a *downward* deflection (the aft end of the flap displaces in a negative z-direction relative to the reference frame convention of `rotatestl`).

It is emphasized that only coordinates separated by white space or a delimiter may be placed in the `hp1FN` and `hp2FN` files. Inserting other information or characters will prevent successful execution of `rotatestl`.





**Figure 12. Display of Hinge Line and Point *hp1* for Example Body Flap**

### 3.6. Input Designator: `deflectionsFN`

The input designator `deflectionsFN` implies “deflections file name.” A string denoting a file name is the input argument to `rotatestl`. The file to which the name refers is an ASCII file in the present working directory that tabulates user-specified angular deflections, in degrees. The algorithm of `rotatestl` rotates, about the specified hinge lines, the parent STL file geometries by the angular deflections.

The generic hypersonic vehicle is again used as an example for input. An ASCII file named `deflections.txt` is used to store the deflection information (any name is suitable as long as the file name does not violate naming restrictions of the operating system). The string input could be

```
'deflections.txt'
```

and is shown as an assignment below to `rotatestl`:

```
rotatestl([0,0,0], 'parents.txt', 'children.txt', 'hp1.txt', 'hp2.txt', ...
'deflections.txt', 'visualflag.txt', 'reverseflag.txt', ', ', ', ');
```

Deflection Angle File Name Input Argument

**Figure 13. Example Input Argument for Input Designator, `deflectionsFN`**

Each column of the deflections input file implicitly represents the schedule of deflections for each parent STL file. If `parents.txt` as previously described were used for STL input, then `deflections.txt` would have five columns, one for each input STL file. Any number of

deflections in degrees, in any order, may be listed under each column of the `deflections.txt` file. Within a column, the deflection angles may be positive, negative, repeated, and may even be zero, but only the unique, non-zero rotation angles within a column are rotated for a given STL file. This enables the user to give any (real) number, in degrees, as input to the deflection input file, and `rotatestl` sorts through the angles automatically. If an angle is repeated, `rotatestl` rotates via the angle only once (per specified parent STL file). Again, only non-zero rotations are considered by the algorithm, as a zero deflection will result in no change to the geometry orientation.

The general format of the `deflectionsFN` file is further described as follows. Using the following definitions in Table 4, a general formulation of deflection input to the `deflectionsFN` file is shown in Figure 14.

**Table 4. Variable Descriptions for `deflectionsFN` File Format**

Variable	Description
<b>N</b>	Total number of input parent STL files
<b>j</b>	$j^{\text{th}}$ column in the <code>deflectionsFN</code> file or $j^{\text{th}}$ row in <code>parentFN</code> file $j = 1, 2, 3, \dots, N$
<b>ND(j)</b>	Total number of deflections scheduled for the $j^{\text{th}}$ parent STL file
<b>k</b>	$k^{\text{th}}$ row of the $j^{\text{th}}$ column of the <code>deflectionsFN</code> file $k = 1, 2, 3, \dots, \text{ND}(j)$
<b><math>d_{j,k}</math></b>	Specific deflection value, in degrees, for the $k^{\text{th}}$ row of the $j^{\text{th}}$ column of the <code>deflectionsFN</code> file

Each column represents the deflection schedule for a single parent STL file and is independent of all other columns. Moreover, the total number of deflections specified for a given parent STL file—that is, the value of  $\text{ND}(j)$ —may be different for any or all parent STL files. For instance, five rotations might be scheduled for the body flap of the generic hypersonic vehicle (in this case  $\text{ND}(1) = 5$  and  $j = 1$ , representing total deflections `Bflap.stl` in the first row of the `parentsFN` file and signifying the first column of the `deflectionsFN` file) while the right elevon might be scheduled for ten deflections (in this case  $\text{ND}(3) = 10$  and  $j = 3$ , representing 10 total deflections for `Relevon.stl` in the third row of the `parentsFN` file and signifying the third column of the `deflectionsFN` file). This explains why the deflection angle inputs in Figure 14 occupy differing lengths along the rows of each column. The maximum row dimension is equal to the maximum number of scheduled rotations across all STL files or, explained numerically,  $\max[\text{ND}(j)]$ . Thus, the total dimension of the deflection input to `deflectionsFN` should be

$$\max[\text{ND}(j)] \times N, \tag{2}$$

If the total number of deflections is not the same for all STL files (i.e.  $\max[\text{ND}(j)] \neq \text{ND}(j)$  for all  $j$ ), then the remaining rows of the columns with a total number of input deflections less than  $\max[\text{ND}(j)]$  should be filled with zeros [ 0 ]. This is shown by the zeros in Figure 14. Any column can have the maximum total number of deflections, or all columns may be completely filled with the same total number of deflections. The display shown in Figure 14 is generalized and illustrates the case when not all parent STL files have the same total number of deflections specified. By convention, however, all columns of every row must be filled. If no deflection is necessary for a given column along a row, fill the column with a zero [ 0 ]. If not all columns of all rows are specified with a numeric value, `rotatestl` prints an error to the MATLAB command prompt and terminates the program.

To reiterate, any numeric input (excluding complex numbers, of course) in degrees may be entered in the `deflectionsFN` file. It is up to the user to determine the upper and lower bounds of rotations to perform based upon the geometry at hand and upon the necessary analysis that requires the use of `rotatestl`. Once again, the numeric input may be repeated or unordered, but only the unique, non-zero deflection angles are rotated by the algorithm. For instance if, in a

	Col. 1	Col. 2	...	Col. ( j )	...	Col. ( N - 2 )	Col. ( N - 1 )	Col. N
LINE 1	$d_{11}$	$d_{21}$	...	$d_{j1}$	...	$d_{(N-2),1}$	$d_{(N-1),1}$	$d_{N,1}$
LINE 2	$d_{12}$	$d_{22}$		$d_{j2}$		$d_{(N-2),2}$	$d_{(N-1),2}$	$d_{N,2}$
LINE 3	$d_{13}$	$d_{23}$		$d_{j3}$		$d_{(N-2),3}$	$d_{(N-1),3}$	$d_{N,3}$
	$\vdots$	$\vdots$		$\vdots$			$\vdots$	
	$d_{1,\text{ND}(1)}$			$d_{j,k}$			$d_{(N-1),\text{ND}(N-1)}$	
	0	$d_{2,\text{ND}(2)}$					0	$\vdots$
$\vdots$		0		$d_{j,\text{ND}(j)}$				
				0			$\vdots$	$d_{N,\text{ND}(N)}$
		$\vdots$		$\vdots$				0
LINE { $\max[\text{ND}(j)] - 1$ }				$\vdots$				$\vdots$
LINE { $\max[\text{ND}(j)]$ }	0	0	...	0	...	$d_{(N-2),\text{ND}(N-2)}$	0	0


Actual input to `deflectionsFN` file is bound by the box above.

Figure 14. General Input Format for `deflectionsFN` File


single column of `deflectionsFN`, two values of 5 are listed, `rotatestl` takes only the first occurrence of the value 5 in the column and generates a child STL file rotated by 5°. The extraneous value of 5 is ignored. Furthermore, if values are not listed in numerical order, this has no effect on the ability of the algorithm to rotate STL files by the specified angles. Finally, if zeros are encountered in the columns of `deflectionsFN`, they are ignored.

The general format shown in Figure 14 may be described more plainly. For every parent STL, specify a set of rotation angles for `rotatestl` to perform. Determine the maximum number of rotations specified for any given parent STL. This maximum number is the total number of rows of input in the `deflectionsFN` file. Then, place each individual set of rotations (one per parent STL) into a column. Stack the columns of rotations side by side (left-to-right) in the `deflectionsFN` in the same order as the parent STL files are listed in the `parentFN` file (row per row). Fill any unfilled columns along a row in the `deflectionsFN` file with zeros until every column has the same number of rows (equal to the maximum number determined previously). The `deflectionsFN` file should then be ready for input to `rotatestl`.

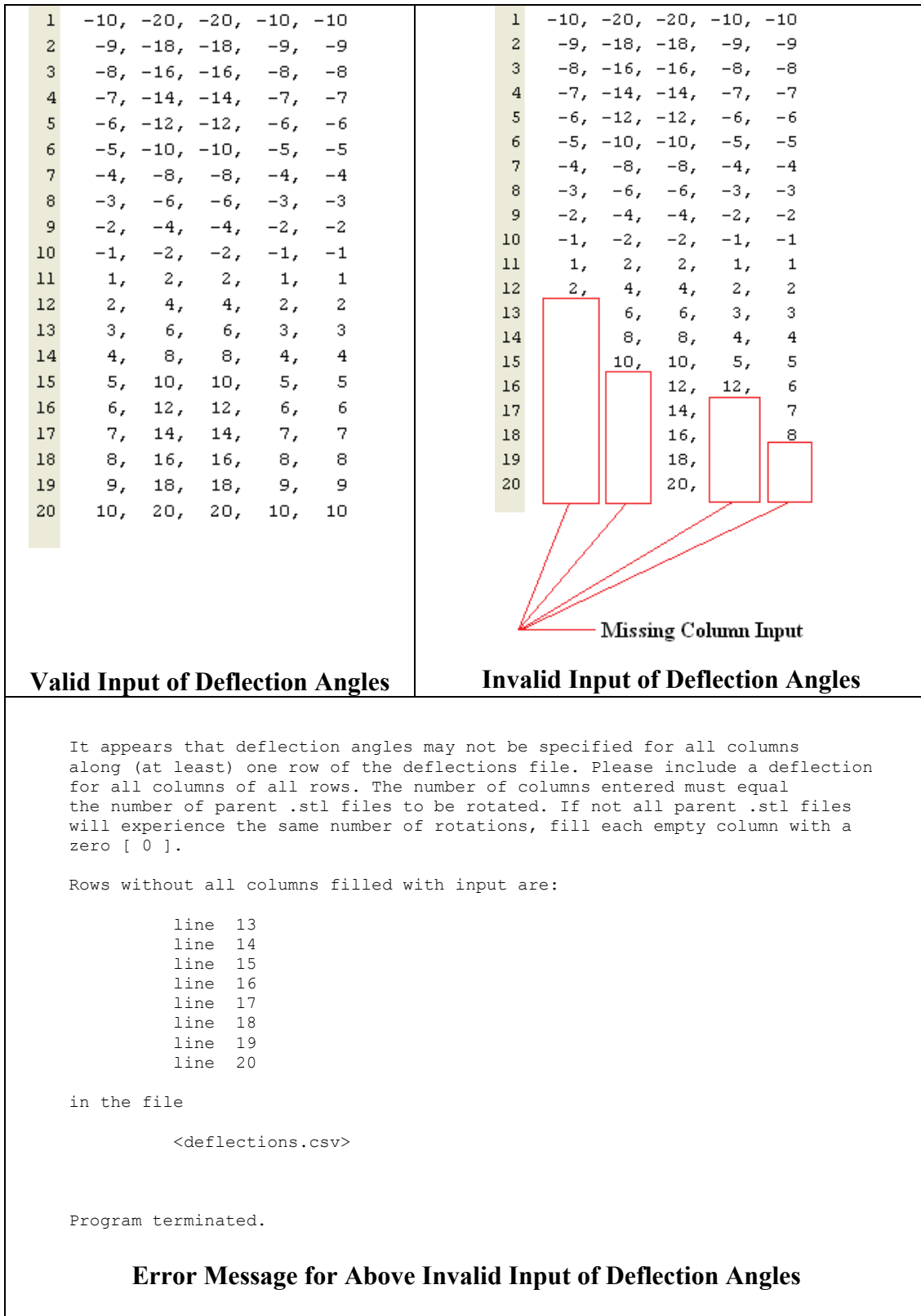
Examples of input to the `deflectionsFN` file are given below. Each example shows valid and invalid files with the same intended input deflection angles in degrees. For the invalid cases, a pertinent error message is included in each example that `rotatestl` would print to the MATLAB command were the invalid file used as input. Figure 15 shows angular deflections that are unordered along each column (the tan shaded column at the left is a labeling of the line number of the file and is not strictly text within the file). Recall that unordered columns have no effect on successful execution of `rotatestl`. Zero values are also included in columns where deflections are unspecified. Note that the invalid case is rejected by `rotatestl` because columns on lines 4 and 5 of the `deflectionsFN` file have missing input. Figure 16 shows input that is unordered and that has repeated deflection values along columns. Further recall that repeated values along a column (specified for a specific parent STL file) is only implemented as a rotation once. The algorithm again notes missing input along columns if the invalid file is used. Figure 17 shows ordered data, from least to greatest, along columns. The input is also comma-delimited, as is implemented by common `<.csv>` files. If multiple columns have missing input in the `deflectionsFN` file, the rows in which the missing data occurs are identified in the error message. Finally, Figure 18 shows a similar comma-delimited input file. In this case, zeros are included along columns where input is not specified. Missing input is identified if the deflections input file is invalid. In all cases, the columns do not have to line up directly. As long as white space, a specified delimiter, or both are used to separate data from column to column, the input should load successfully into `rotatestl`.

<pre> 1  5      2      2     -5  -5 2  -5     5.5    5.5    5   5 3  10    -5.5   -5.5    0   0 4 -10.1  -2     -2     0   0 5  12.5   0      0     0   0 </pre> <p><b>Valid Input of Deflection Angles</b></p>	<pre> 1  5      2      2     -5  -5 2  -5     5.5    5.5    5   5 3  10    -5.5   -5.5    0   0 4 -10.1  <span style="border: 1px solid red; display: inline-block; width: 40px; height: 20px; vertical-align: middle;"></span> 0   0 5  12.5  <span style="border: 1px solid red; display: inline-block; width: 40px; height: 20px; vertical-align: middle;"></span> 0   0 </pre> <p style="text-align: center;">  <b>Missing Column Input</b> </p> <p><b>Invalid Input of Deflection Angles</b></p>
<p>It appears that deflection angles may not be specified for all columns along (at least) one row of the deflections file. Please include a deflection for all columns of all rows. The number of columns entered must equal the number of parent .stl files to be rotated. If not all parent .stl files will experience the same number of rotations, fill each empty column with a zero [ 0 ].</p> <p>Rows without all columns filled with input are:</p> <pre>       line 4       line 5 </pre> <p>in the file</p> <pre>       &lt;deflections.txt&gt; </pre> <p>Program terminated.</p> <p style="text-align: center;"><b>Error Message for Above Invalid Input of Deflection Angles</b></p>	

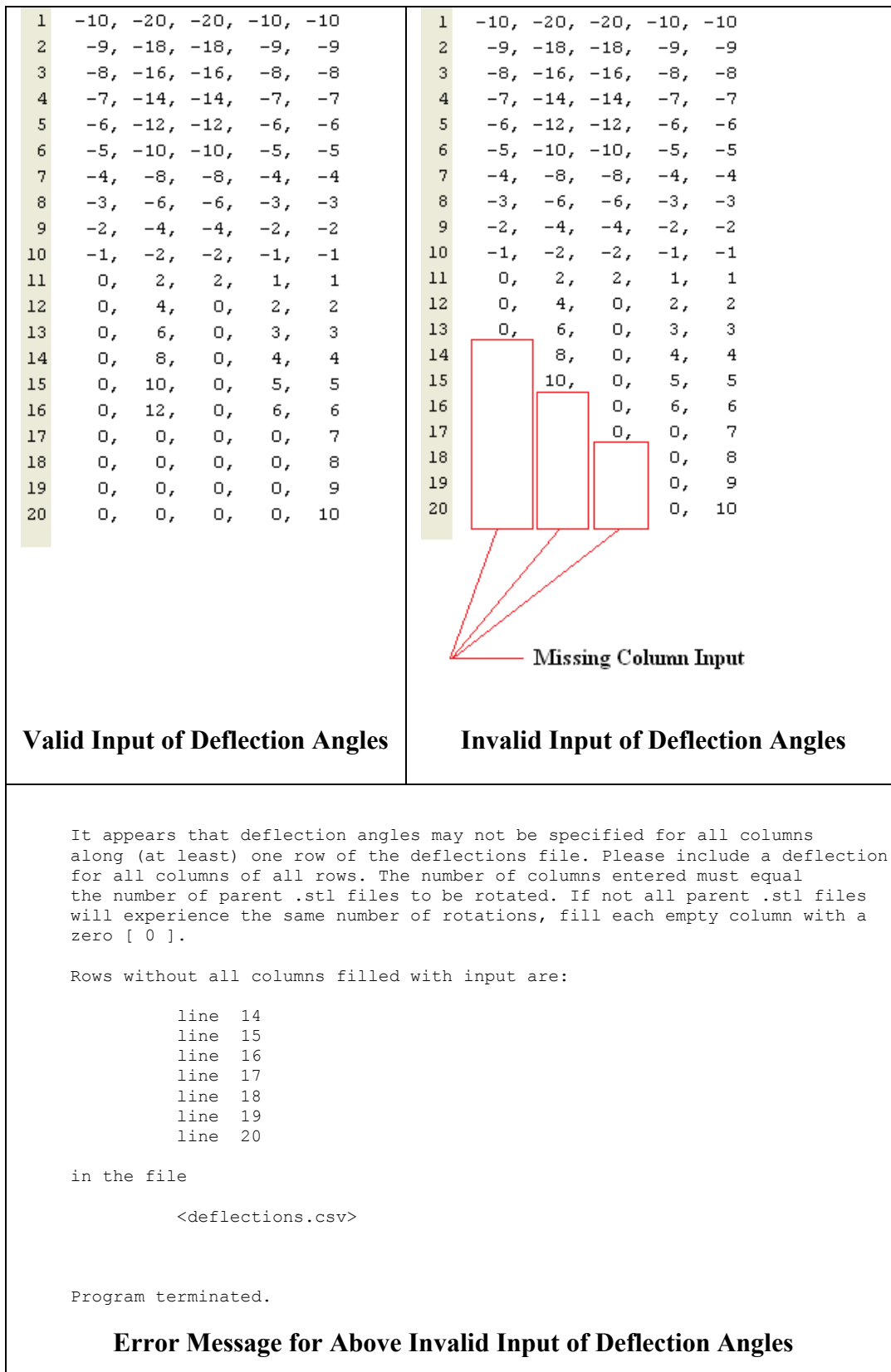
**Figure 15. Unordered Input Including Zeros along Columns of deflectionsFN File**

<pre> 1  5  2  2  5 -5 2  5  5  5.5 5  5 3 10  5 -5.5 2  2 4 10 -2  2  -2  2 5 12.5 6  6  10 10 </pre> <p style="text-align: center;"><b>Valid Input of Deflection Angles</b></p>	<pre> 1  5  2  2  5 -5 2  5  5  5.5 5  5 3 10  5 -5.5 4 10 -2  2 5 12.5 6  6 </pre> <div style="text-align: center;">  <p><b>Missing Column Input</b></p> </div> <p style="text-align: center;"><b>Invalid Input of Deflection Angles</b></p>
<p>It appears that deflection angles may not be specified for all columns along (at least) one row of the deflections file. Please include a deflection for all columns of all rows. The number of columns entered must equal the number of parent .stl files to be rotated. If not all parent .stl files will experience the same number of rotations, fill each empty column with a zero [ 0 ].</p> <p>Rows without all columns filled with input are:</p> <pre> line 3 line 4 line 5 </pre> <p>in the file</p> <pre> &lt;deflections.txt&gt; </pre> <p>Program terminated.</p> <p style="text-align: center;"><b>Error Message for Above Invalid Input of Deflection Angles</b></p>	

**Figure 16. Unordered Input with Repeated Values along Columns of deflectionsFN File**



**Figure 17. Ordered Input with Comma Delimiter for deflections<sub>FN</sub> File**



**Figure 18. Ordered Input with Zero Values and Comma Delimiter for deflectionsFN File**



### 3.7. Input Designator: `visualflagFN`

The input designator `visualflagFN` implies “visualization flag file name” and requires a string input argument of a file name. The file to which it refers is an ASCII file containing a listing of choices to show, or not to show, the maximum rotation specified for a given parent STL file. If visualization is desired, a figure of the desired parent STL and its child STL of maximum rotation is displayed. The `visualflagFN` file is placed in the present working directory.

Provide `rotatestl` with a string for the `visualflagFN` designator. An example input argument may be

```
'visualflag.txt'
```

and is shown below in the call to `rotatestl`:

```
rotatestl([0,0,0], 'parents.txt', 'children.txt', 'hp1.txt', 'hp2.txt', ...  
'deflections.txt', 'visualflag.txt', 'reverseflag.txt', ',');
```

Visual Flag File Name Input Argument

**Figure 19. Example Input Argument for Input Designator, `visualflagFN`**

Input to the file `visualflag.txt` is a single column of ones [ 1 ] or zeros [ 0 ]. Each row of the column corresponds to a visualization choice regarding the output derived from the parent STL of the respective row in the `parentFN` file. A value of one indicates that the user requires that a child STL be displayed. The STL file displayed is that file of maximum rotation for a given parent STL file. A zero indicates that no visualization is required, and no display is shown for the respective parent STL. These visualization choices are summarized in Table 5. All choices for every parent STL file are required to run `rotatestl`. The default value for visualization is zero, which gives no display. If a large number of rotations is being performed, it is recommended that the user choose zero for all visualization flags to prevent the computer from running out of memory.

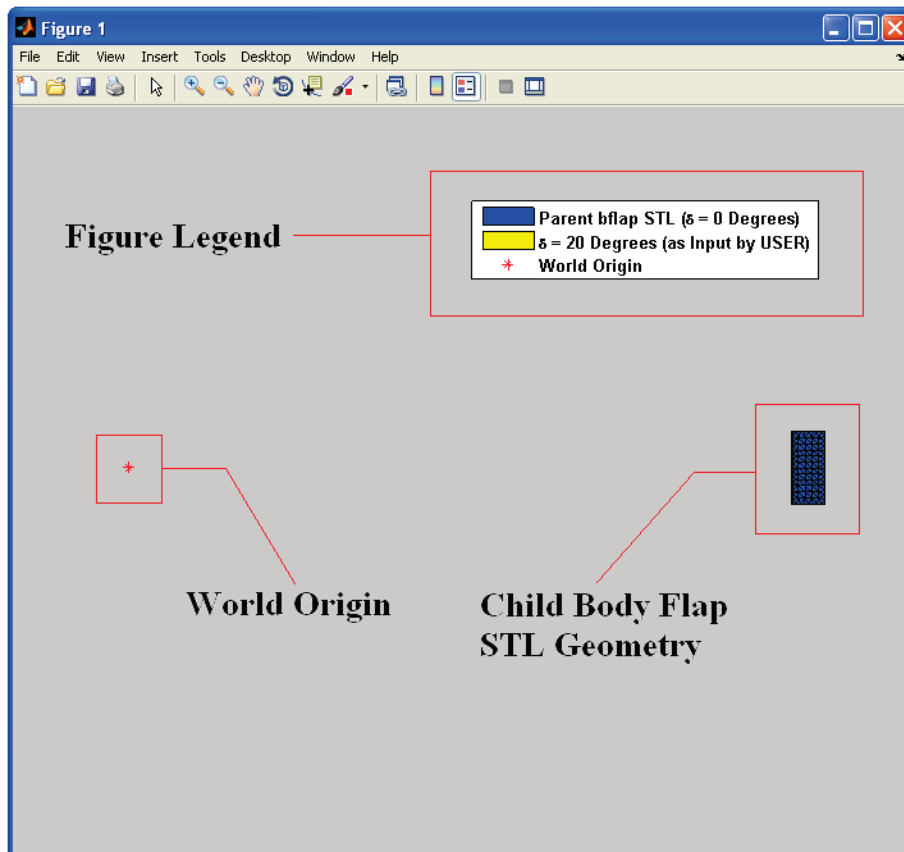
**Table 5. Summary of Visualization Choice Input to `visualflagFN` File**

Flag Value	Result
0	No visualization displayed (default input)
1	Display child STL file of maximum specified rotation

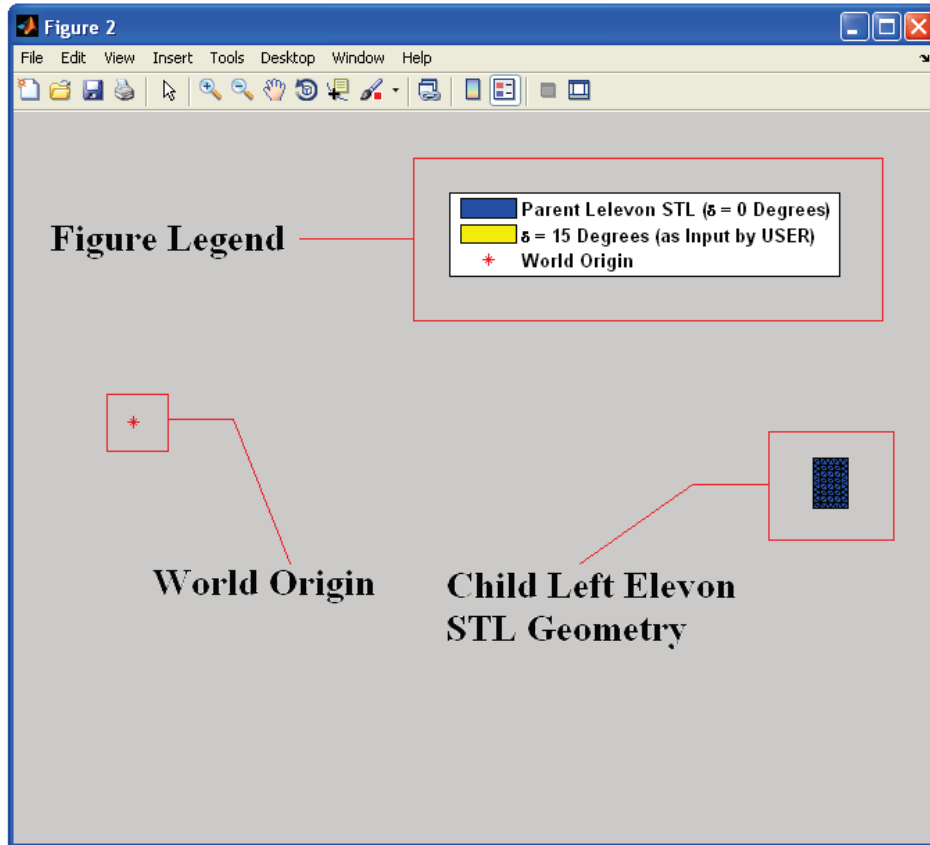
An example visualization set-up for the generic hypersonic vehicle is provided in Table 6. On the left is the `visualflag.txt` file. On the right is the `deflections.txt` file with a simple schedule of rotations for the `parents.txt` file discussed previously. The hinge point files of Table 3 also

apply. Because the `parents.txt` file lists five parent STL files, the visualization flag file should contain five input choices along a single column. The body flap and left elevon flags—the first and second rows, respectively, of the `visualflag.txt` file—have been set to one as a means to emphasize the visualization feature of `rotatestl`. By convention, the child STL that has been rotated by the largest angular deflection is displayed. In this case, the child body flap that is to rotate by  $20^\circ$  is displayed upon execution of `rotatestl`. Similarly, the left elevon that is to rotate by  $15^\circ$  is displayed upon execution of the program. If the set-up of Table 6 is executed via the command of Figure 19 (assuming all proper input), the resulting displays are shown in Figure 20 and Figure 21.

visualflag.txt		deflections.txt					
1	1	1	-20	-15	-15	-10	-10
2	1	2	20	15	15	10	10
3	0						
4	0						
5	0						



**Figure 20. Initial Display of Rotated Body Flap for a Generic Hypersonic Vehicle**



**Figure 21. Initial Display of Rotated Left Elevon for a Generic Hypersonic Vehicle**

Each display initially shows the world origin with a red asterisk, the parent STL file in blue, and child STL file in yellow. Also note that a legend appears in the upper right-hand corner of the figure. Within the legend, the parent and child STL file colors are labeled, and the deflection angle of the child STL file is displayed. The world origin label is also included in the legend. Furthermore, the angle of deflection shown for each STL geometry is the maximum deflection input for a given parent STL file along a column of the `deflectionsFN` file. An enhanced view and description of the legend is given in Figure 22 with a side comparison of the `deflectionsFN` file.

At the top of the figure display is a tool bar with numerous display options and functions. Of particular use in immediate analysis with `rotatestl` are the zoom, pan, and rotate commands. These commands are noted in Figure 23.

To use either of the Zoom In or Zoom Out commands, left click on either icon to enable the command. The Zoom In command is shown in the enabled position in Figure 24. Once the zoom command is enabled, create a box around the area of interest in the display by left clicking, holding the left click button, and dragging the cursor over the area of interest. To show this procedure, Figure 25 shows a box around the body flap as the zoom command is enabled. An alternative approach to zooming is to simply left click and release over a certain area of the

figure display. After creating the box around the body flap, the left click button may be released. The display zooms, showing an enlarged view of the body flap, as shown in Figure 26. Zooming in or out may continue by repeating the above procedure with either Zoom In or Zoom Out enabled. Double click the left mouse button (with zoom, pan, or rotate enabled) in the figure display to reset the view.

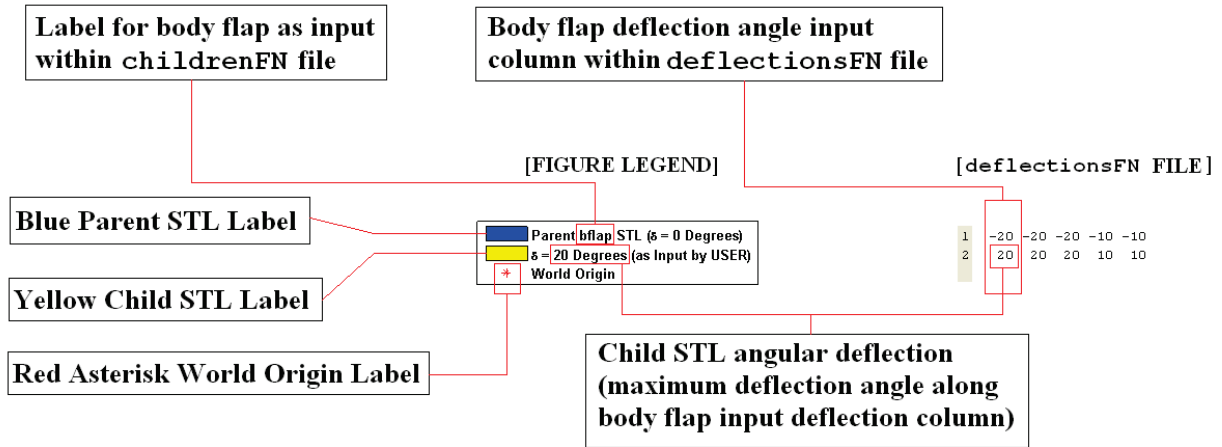


Figure 22. Description of Figure Legend for Child STL Body Flap

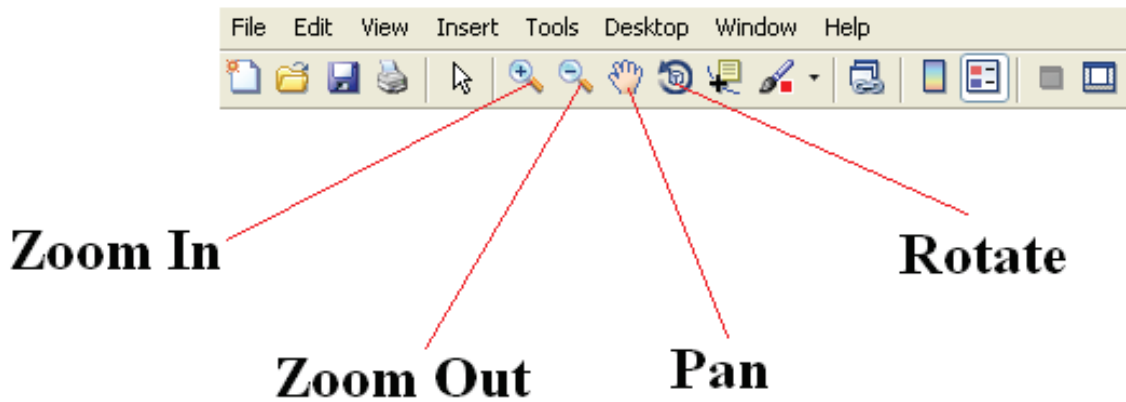
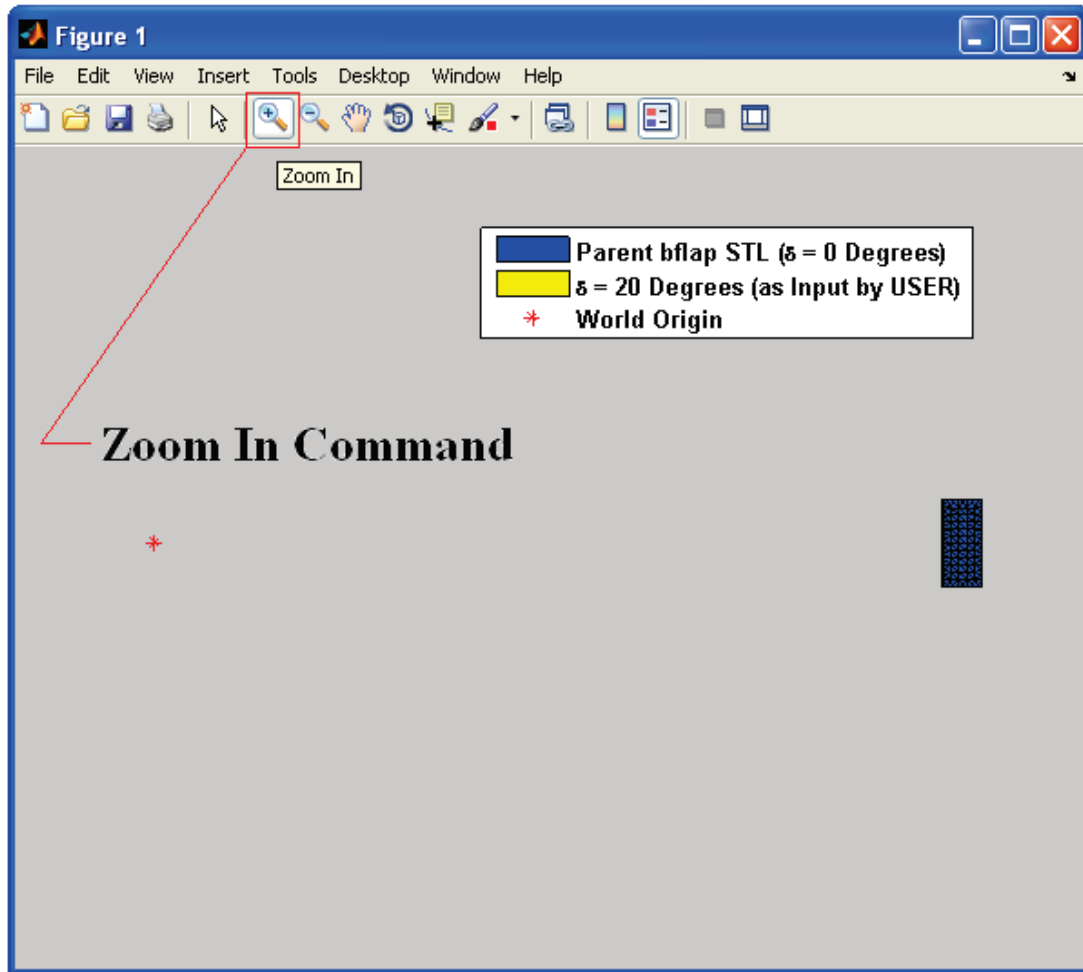
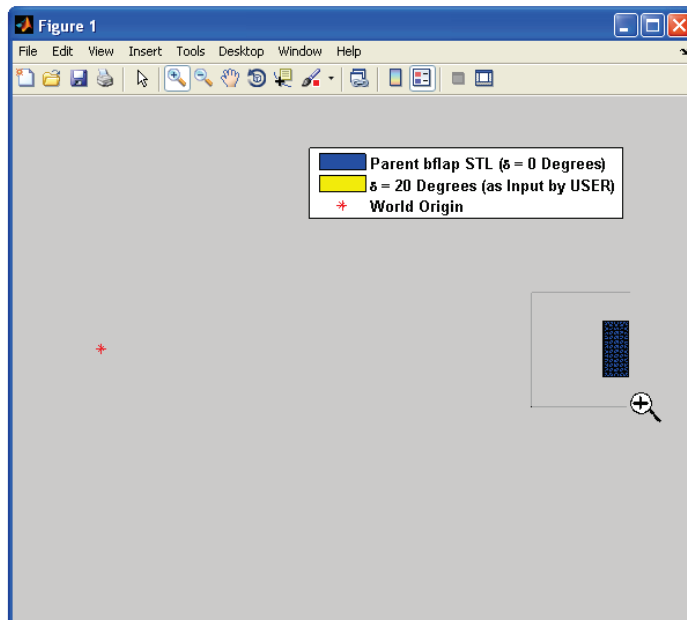


Figure 23. Zoom, Pan, and Rotate Commands of MATLAB Figures

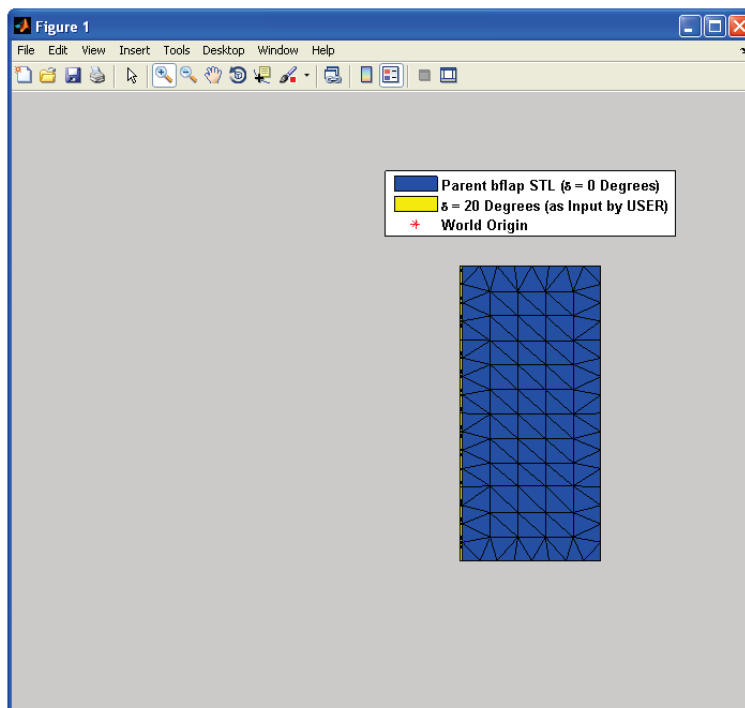


**Figure 24. Zoom In Command Enabled in MATLAB Figure Display**

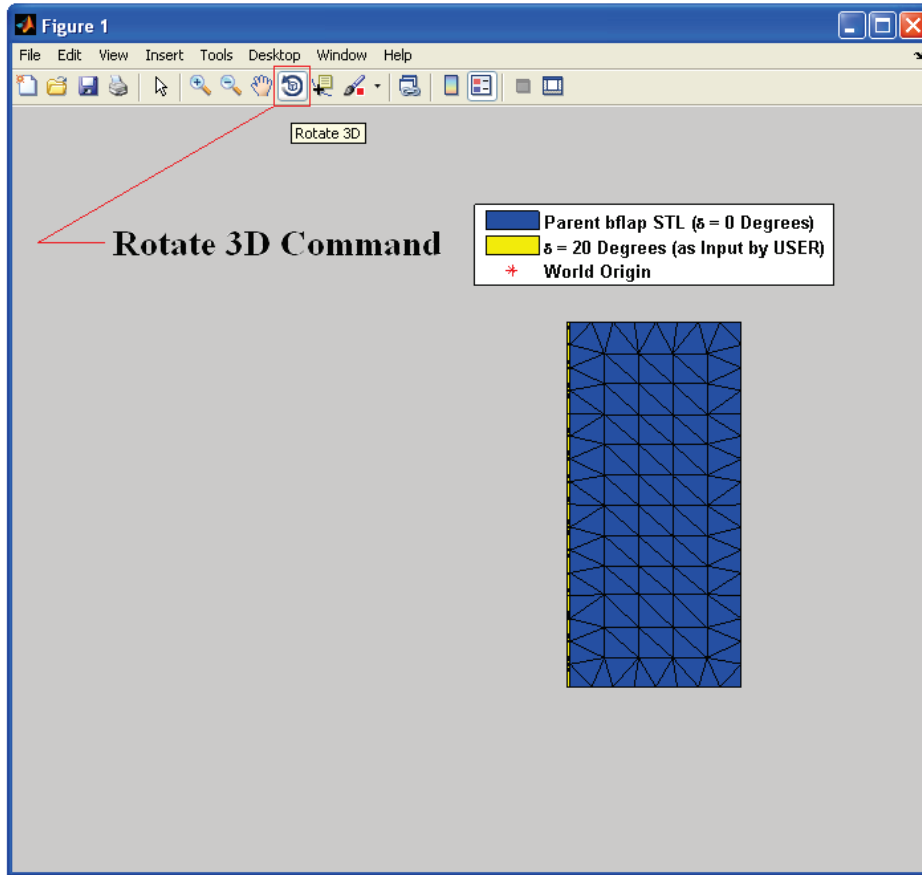
The rotate command may be utilized to view other perspectives of the rotated STL files. When in the zoom position as shown in Figure 26, enable the rotate command at the toolbar by left clicking the Rotate 3D button that shows a circular arrow surrounding a box, shown in Figure 27. Similar to the zoom commands, the Rotate 3D command may be executed by dragging the mouse icon while clicking and holding the left mouse button. With the rotate command now enabled, left click, hold, and drag a section of the figure. The body flap rotates via the mouse commands. An example rotation of the body flap is shown in Figure 28. The rotated child STL file is shown in yellow. Note that the image in the figure is simply a tool to observe how a rotation has taken place based upon user input and upon the convention of rotation discussed in Section 3.5 relative to hinge point input. The blue image of the body flap is the original parent STL file, shown to visualize the child STL rotation. The actual child STL files, however, contain solely the rotated geometric parent STL information. To be clear, only the geometry for a single component is contained in each child STL file. As a hint for using the rotate command with `rotatestl`, zoom around the desired image with a box first, and then rotate the image. This allows for more intuitive rotation.



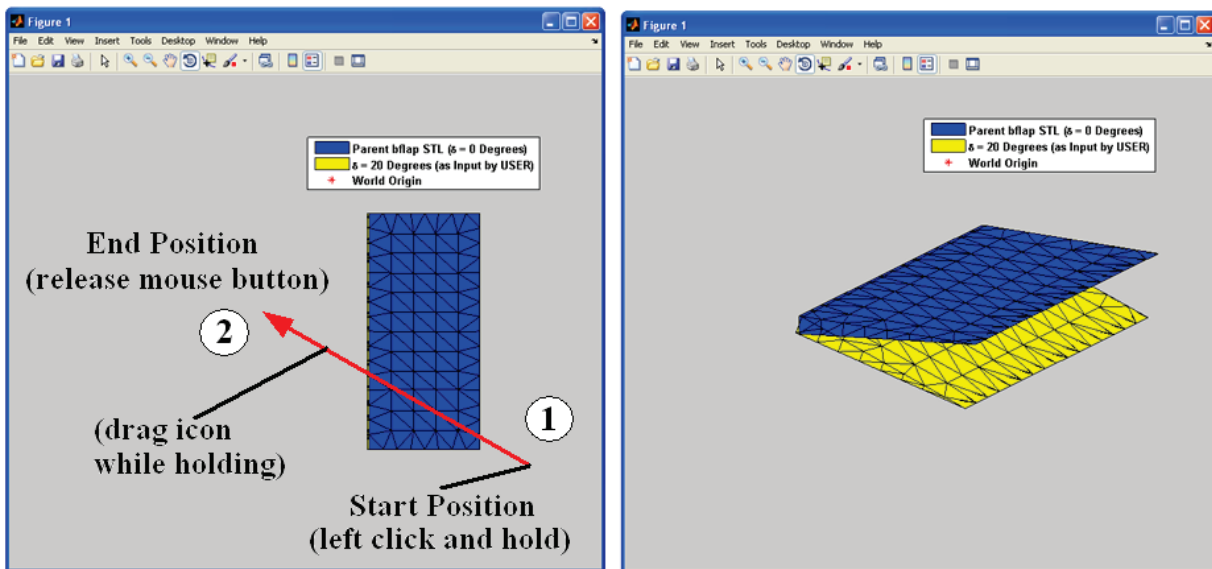
**Figure 25. Creating a Box around the Child STL Body Flap with the Zoom Command**



**Figure 26. Initial Zoomed Image of Child STL Body Flap**



**Figure 27. Rotate 3D Command Enabled in MATLAB Figure Display**

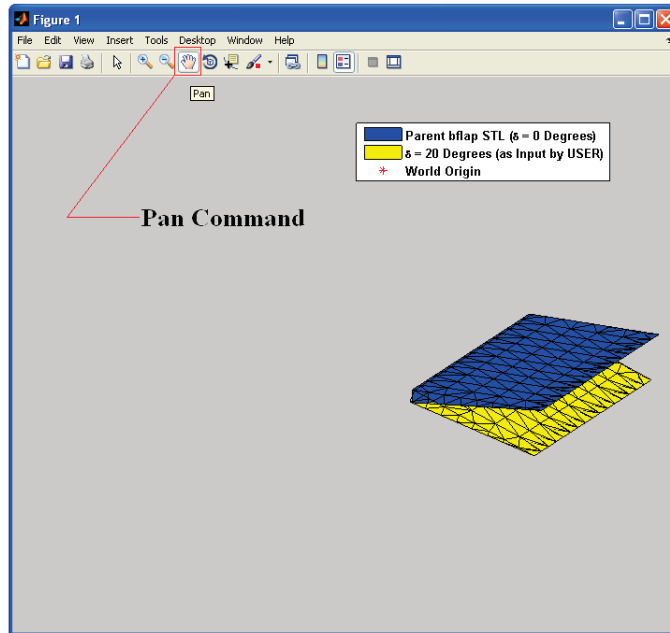


**Mouse Movements to Rotate Figure**

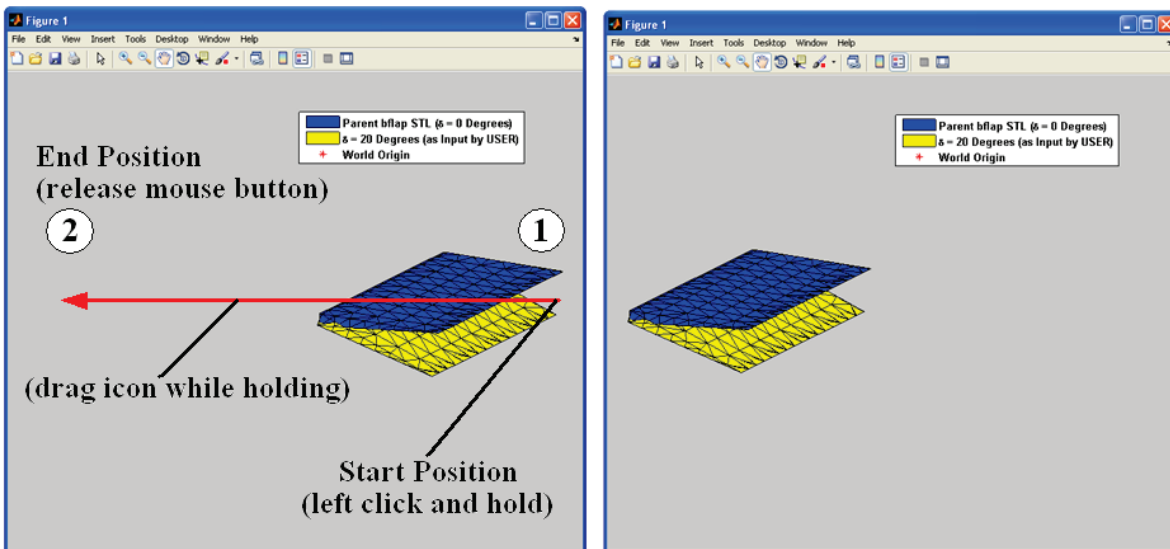
**Resulting Rotated Figure**

**Figure 28. Example Rotation of Body Flap**

If translation of the figure is necessary, use the pan command. The pan command will translate the present image on the figure to a new location specified by user mouse movements. To enable the pan command, left click on the Pan icon, denoted by a hand shown in Figure 29. Using the pan command is similar to that of using the rotate command. Clicking and dragging the mouse icon translates an image of the figure from one point to another. The procedure is shown in Figure 30. Repeat the procedure to pan an image to other locations in the figure.



**Figure 29. Pan Command Enabled in MATLAB Figure Display**



**Mouse Movements to Pan Figure**

**Resulting Figure**

**Figure 30. Example Pan of Body Flap**



Many other features that are inherent capabilities within MATLAB are embedded within each figure. These features include, among others, printing to various output formats, figure labeling, and viewing of data points directly on the figure. To find more information about the use of figures within MATLAB, please see the MATLAB documentation.

### 3.8 Input Designator: `reverseflagFN`

Input through the designator `reverseflagFN` implies “reverse flag file name.” The input argument is a string denoting a file name. The file name to which the input refers is an ASCII formatted file containing decision flags, either one or zero, that direct `rotatestl` to change (or not to change) the direction of rotation of a set of child STL file.

The general input argument of the `reverseflagFN` designator is a string. Example input to `rotatestl` could be

```
'reverseflag.txt'
```

which would denote the file `reverseflag.txt`. The representation of such input is shown below in Figure 31:

```
rotatestl([0,0,0], 'parents.txt', 'children.txt', 'hp1.txt', 'hp2.txt', ...  
'deflections.txt', 'visualflag.txt', 'reverseflag.txt', ',');
```

Reverse Flag File Name Input Argument

**Figure 31. Example Input Argument for Input Designator, `reverseflagFN`**

The input file itself, `reverseflag.txt`, contains a column of ones and zeros denoting the user decision flags for STL rotation reversal. There is only one flag per row of the column, each row corresponding to the STL file listed in the respective row of the `parentFN` file. The decision to reverse the direction of rotation is denoted by the value one [ 1 ]. The decision to leave the direction of rotation unchanged is denoted by the value zero [ 0 ]. The default input to `rotatestl` is zero. These decision inputs are reiterated in Table 7. A reverse flag value must be supplied for every parent STL file provided in the `parentFN` file. Every child STL corresponding to a given parent STL is rotated in the direction according to the decision implied by the reverse flag value.

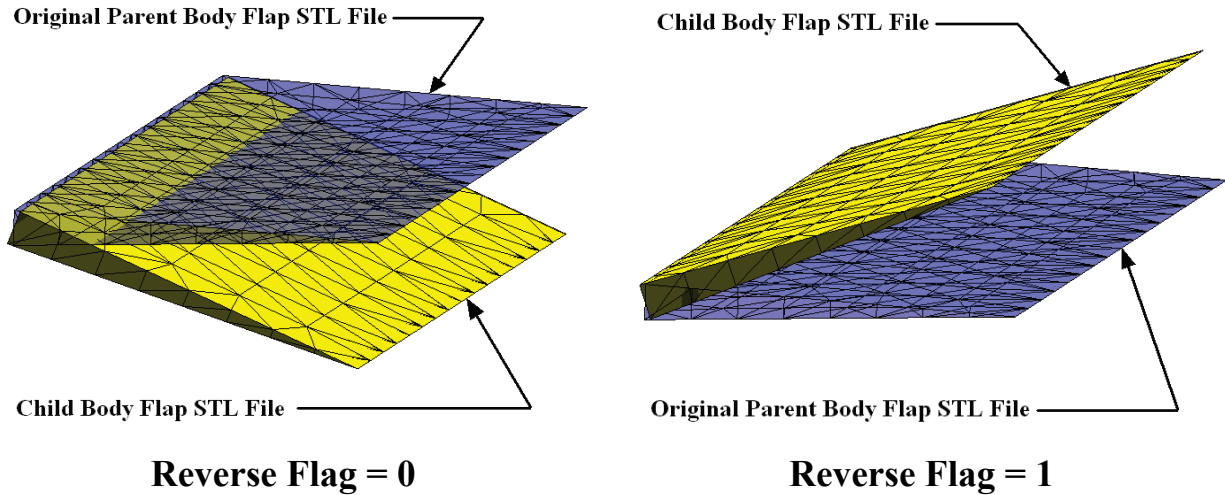
**Table 7. Summary of Reverse Flag Input to `reverseflagFN` File**

<b>Flag Value</b>	<b>Result</b>
0	Direction of rotation unchanged (default input)
1	Reverse direction of rotation of child STL file

An example column of input to the `reverseflag.txt` file is shown in Figure 32. Note that the input to `reverseflag.txt` in this case would cause `rotatestl` to reverse the direction of rotation of the child STL files corresponding to `Bflap.stl` of the `parentFN` file discussed previously in Section 3.3. A comparison of  $+20^\circ$  rotation with and without reversal for the `Bflap.stl` file is shown in Figure 33. The idea here is that, regardless of the direction in which `rotatestl` rotates an STL file about a hinge line, the reverse flag option may be used to *denote* the rotation direction. For the case of the body flap above, both rotations shown in Figure 33 denote positive rotation. However, the convention of positive rotation for the body flap in which `rotatestl` implements depends upon the value of the reverse flag option that the user specifies. Once again, all child STL files are rotated relative to the convention implied by the reverse flag option of a given parent STL file. The child body flap STL files, linked to the example set-up of Figure 32, would take on positive rotations for a reverse flag option equal to one, shown on the right side of Figure 33. The user must determine which convention of positive rotation is appropriate for the application at hand.

<b>reverseflag.txt</b>	
1	1
2	0
3	0
4	0
5	0

**Figure 32. Example Set-up of `reverseflagFN` File**



**Figure 33. Example of Rotation Reversal for +20° Rotation of Bflap.stl File**

As a note on a potential issue with the reverse flag option, if the hinge line is nearly parallel with the z-axis (as defined by the coordinate system convention of Figure 4), the reverse flag option does not appear to reverse the direction of rotation properly. The user may need to be mindful of the labeling of child STL files if the opposite rotation direction implemented by `rotatestl` is desired in this case. Positive and negative input deflection angles to the `deflectionsFN` file still produce opposite rotations as expected. Therefore, if the opposite rotation direction is desired for STL files in this case, the user may need to swap the sign of the respective deflection angle in the labeling of the child STL files.

### 3.9. Input Designator: `delimiter`


The input designator `delimiter` is a string character that denotes a user-specified delimiter, or separating character, found within the input files for `rotatestl`. The default delimiter is white space or repeated white space and is readily implied for user input. If the user desires to supply a delimiter, any printable ASCII character may be entered as a string input argument except for numbers, the dash symbol `<->`, and the backslash symbol `<\>`. In fact any combination of characters (excluding those just mentioned) may also be used as a user-defined delimiter. However, this is not recommended, as `rotatestl` may perform unexpectedly. A single ASCII character specified as a user-defined delimiter is sufficient. The user-defined delimiter applies only to the `hp1FN`, `hp2FN`, and `deflectionFN` input files. The `parentFN`, `childrenFN`, `visualflagFN`, and `reverseflagFN` files all have input syntax, as defined in the previous

sections, that do not require delimiters. The delimiter input argument itself, however, is required for successful program execution.

The input argument to `rotatest1` should be a printable ASCII character placed within single quotes. If the user chooses not to supply a delimiter, place a space within single quotes as the input argument `< \ ' >`. An example input argument for the use of a comma as the user-specified delimiter would be input as `< \ , >`. This is shown in Figure 34.

```
rotatest1([0,0,0], 'parents.txt', 'children.txt', 'hp1.txt', 'hp2.txt', ...  
'deflections.txt', 'visualflag.txt', 'reverseflag.txt', ',');
```

Delimiter Input Argument



**Figure 34. Example Input Argument for Input Designator, `delimiter`**

It is recommended that the user does not provide the decimal character `< . >`, the plus character `< + >`, or the asterisk character `< * >` as a user-specified delimiter. These characters may interfere with valid input such as a file extension or a decimal point for a deflection angle, for example. If `.csv` files are used for input, provide the comma as the specified delimiter, as shown in Figure 34. Consecutively repeated delimiter characters, such as `< , , , >`, cause `rotatest1` to read input improperly and should not be placed within input files. The program can accommodate, however, repeated white space inside the input files. Furthermore, if a user-specified delimiter is provided, both white space and the delimiter are ignored. For instance, if the comma delimiter is specified as in Figure 34, `rotatest1` would still successfully read the valid `deflectionsFN` files of Figures 15 through 18.

In general, the input delimiter must follow the rules of the `textscan` function in MATLAB. Please see the MATLAB help documentation, if necessary, for more information on the `textscan` function.

#### 4.0. Sample Program Execution and Results

Throughout the former discussion of the use of `rotatest1`, a set of examples were introduced in order to describe the program with the geometry of a generic hypersonic vehicle, shown in Figure 5. The discussion now continues, showing the execution of the running example previously noted and highlighting the implications of the use of `rotatest1` for applications to computational analyses.

Using the example input information provided in Section 3.0, the component control surfaces of the generic hypersonic vehicle of Figure 5 are rotated as follows. The various input files discussed in Section 3.0 are shown together below in Figure 35 on page 40. The command input discussed in Section 3.1 is used to execute `rotatest1` and is repeated below:

```
rotatest1([0,0,0], 'parents.txt', 'children.txt', 'hp1.txt', 'hp2.txt', ...  
'deflections.txt', 'visualflag.txt', 'reverseflag.txt', ',,');
```

Upon full execution of the program, `rotatest1` prints a message to the MATLAB command prompt as shown in Figure 36 on page 41. The presented message indicates that the *runX* folder system has been created for the first execution of `rotatest1` within the present working directory and displays the elapsed time necessary to perform all rotations requested by the user. Execution times vary depending upon the operating system used, the number of rotations requested, the size of each parent STL file, and whether or not visualization is needed. The *runX* folder system is labeled *run1* and is shown on page 41 in Figure 37. Note that the sub-directories are labeled according to the input found in the `children.txt` file. Moreover, within each sub-directory, each child STL is labeled according to the convention specified in Section 3.4 in which the respective deflection angle is combined with the child name indicated by the user. The actual rotated child STL geometries are shown in Figure 28. The geometries are shown at a perspective to emphasize the rotation that has taken place and are not shown to scale. Following the input to the `visualflag.txt` file, `rotatest1` generates figures that look exactly like those shown in Figure 20 and Figure 21. Finally, note that, in Figure 38, the positive rotation convention of the `bflap20deg.stl` file is the same as that displayed to the right of Figure 33 due to the reverse flag option value of one, set for the body flap in the `reverseflag.txt` file.

The net effect of rotating the control surfaces may be realized if the surfaces are rejoined with the rest of the hypersonic vehicle. Some perspectives of the vehicle are shown in Figure 39 on page 43 with rotated control surfaces superposed to wing and body geometry. In general, if the rotated STL surfaces are combined with other geometry, such as a vehicle mesh in the case above, the door to various computational analyses is opened. In the case of the vehicle described, forces and moments about the vehicle in a flow field may be determined through CFD for any configuration change to the control surfaces. This allows the engineer or the designer to characterize the aerodynamic response of the vehicle at different flight conditions. Other

applications may be possible, such as rotation of meshes for graphics or rapid prototyping. Though, the latter two applications have not yet been implemented with `rotatestl`. Regardless of the application, however, the intent of the program is to change the orientation of a mesh directly, without the need to re-create or to re-mesh new surface topology. Such utility enables tools and methods involving grid generation to take place at much faster time scales than would be possible otherwise.

<b>parents.txt</b>		<b>hp2.txt</b>	
1	Bflap.stl	1	11.766 -0.638 -0.004
2	Lelevon.stl	2	11.766 -1.500 -9.51685e-6
3	Relevon.stl	3	11.766 0.638 -9.51685e-6
4	Ltail.stl	4	10.737 -0.388 0.067
5	Rtail.stl	5	10.737 0.388 0.067
<b>children.txt</b>		<b>deflections.txt</b>	
1	bflap	1	-20 -15 -15 -10 -10
2	Lelevon	2	20 15 15 10 10
3	Relevon		
4	Ltail		
5	Rtail		
<b>hp1.txt</b>		<b>visualflag.txt</b>	
1	11.766 0.638 -0.004	1	1
2	11.766 -0.638 -9.51685e-6	2	1
3	11.766 1.500 -9.51685e-6	3	0
4	10.778 -0.388 1.066	4	0
5	10.778 0.388 1.066	5	0
<b>reverseflag.txt</b>			
	1	1	
	2	0	
	3	0	
	4	0	
	5	0	

**Figure 35. Contents of Input Files for Program Execution Example**

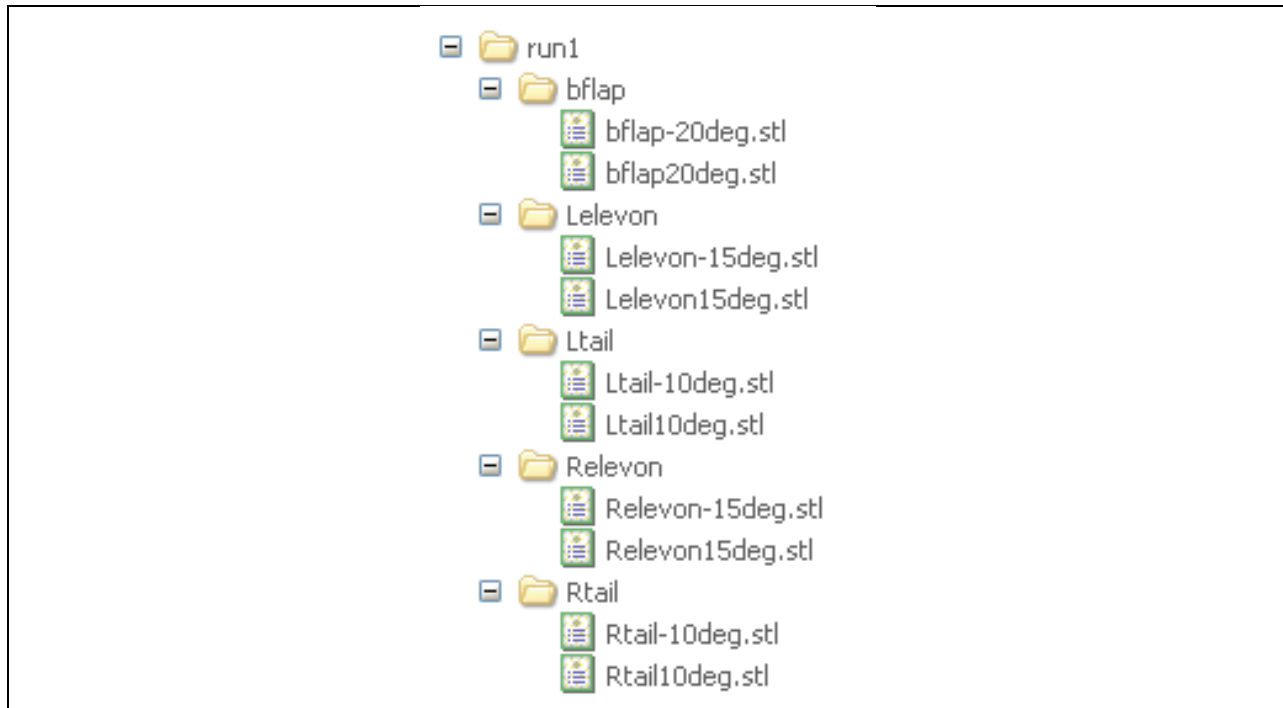
```
Command Window

Storage folder for new .stl files created: run1

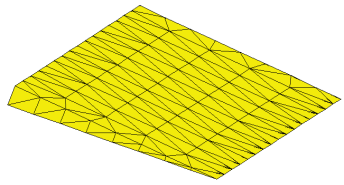
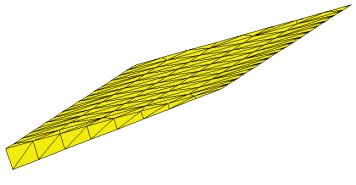
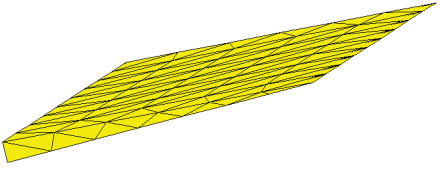
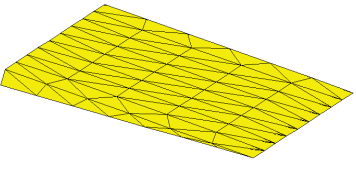
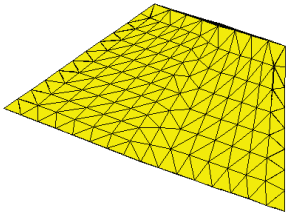
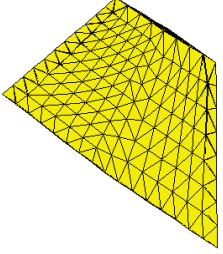
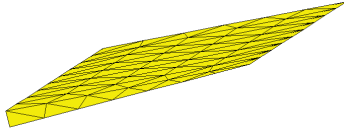
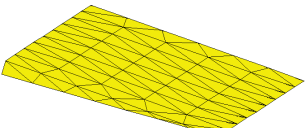
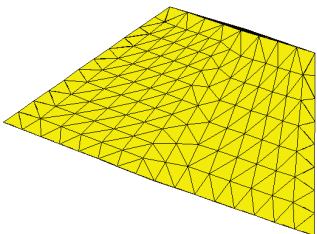
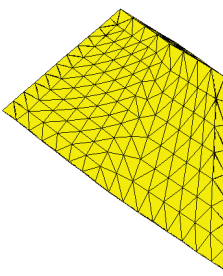
Rotation Algorithm Performance: Elapsed time is 4.336697 seconds.

fx >>
```

**Figure 36. Command Window Output upon Successful Execution of rotatestl**

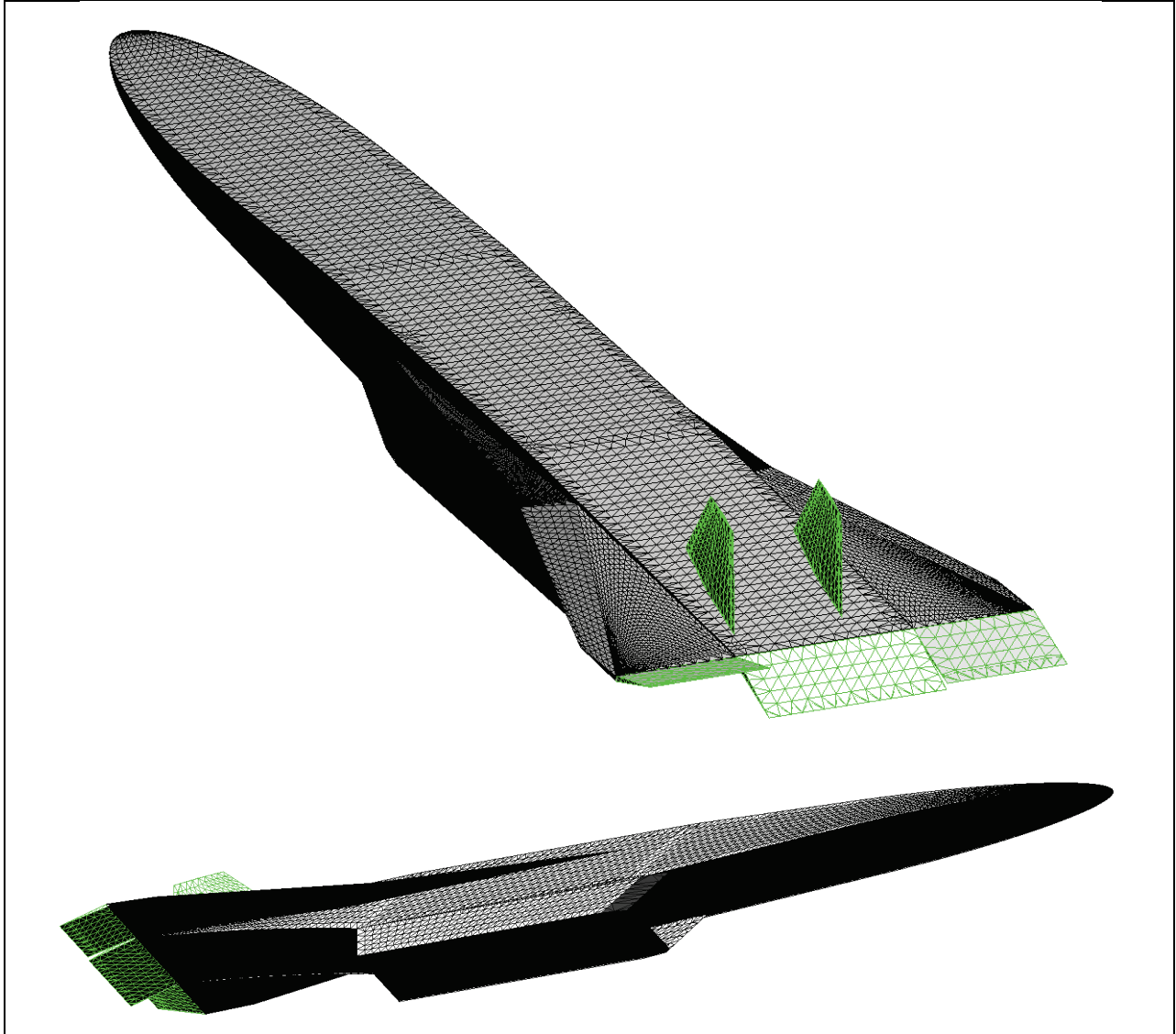


**Figure 37. Folder System Created after Execution of rotatestl**

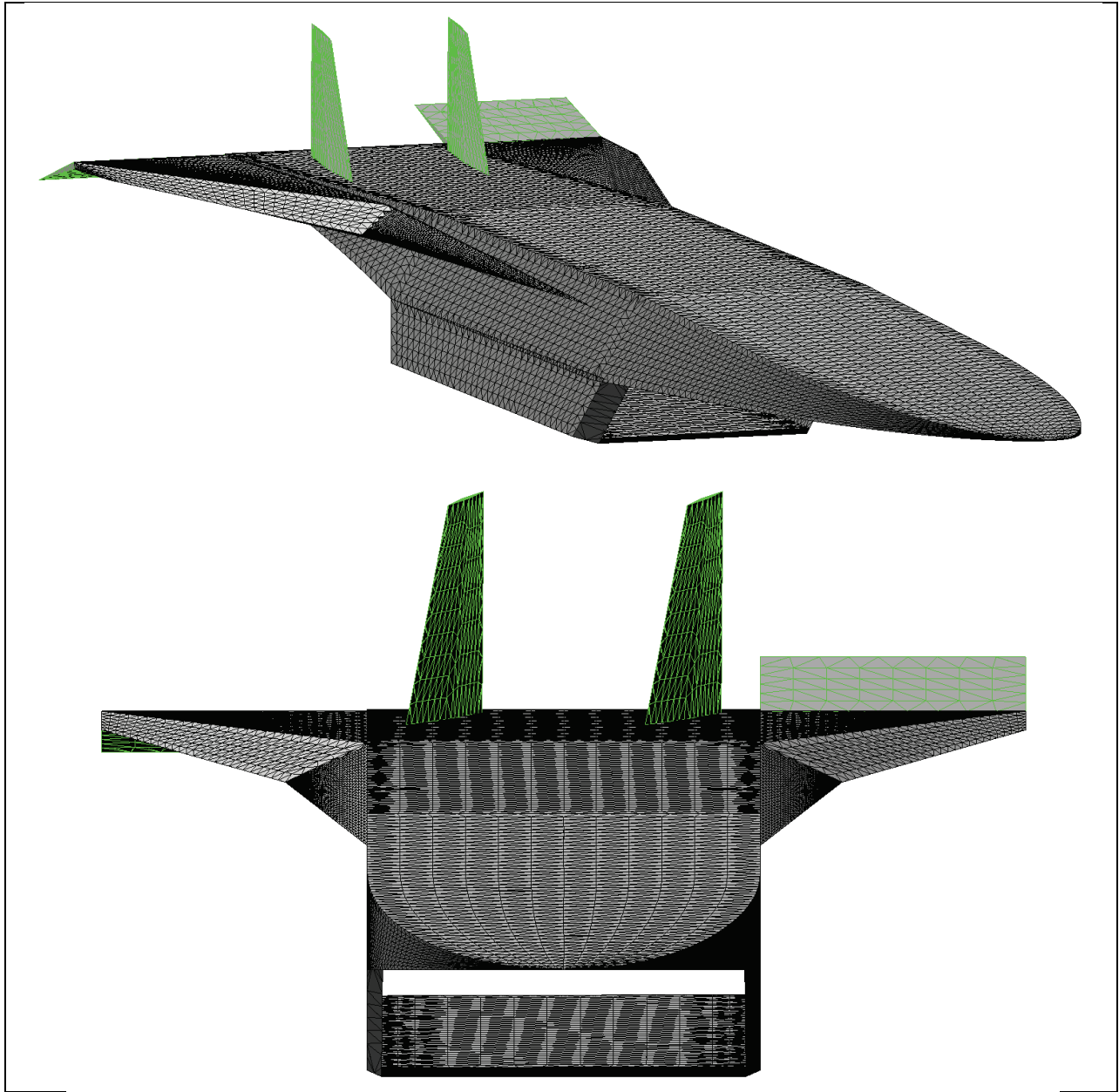
<b>bflap-20deg.stl</b>	<b>bflap20deg.stl</b>
	
<b>Lelevon-15deg.stl</b>	<b>Lelevon15deg.stl</b>
	
<b>Ltail-10deg.stl</b>	<b>Ltail10deg.stl</b>
	
<b>Relevon-15deg.stl</b>	<b>Relevon15deg.stl</b>
	
<b>Rtail-10deg.stl</b>	<b>Rtail10deg.stl</b>
	

**Figure 38. Child STL Files Created after Successful Execution of rotatestl**





**Figure 39. Generic Hypersonic Vehicle STL Geometry with Rotated Control Surfaces**



**Figure 39. Generic Hypersonic Vehicle STL Geometry with Rotated Control Surfaces  
(Concluded)**

## 5.0 Conclusion

In some computational applications, such as computational fluid dynamics or finite element analysis, mesh rotation can be a necessary requirement to change the geometric object at hand in order for new analysis to take place. Performing grid generation manually through a re-triangulation of rotated geometry can be time consuming, especially if many different rotations are being considered. The MATLAB rotation algorithm, `rotatestl`, is a tool that provides a means to rotate grid or mesh geometry in the form of the Stereolithography file format. Any number of rotations may be specified for any number of original STL files. The program has nine major inputs required for execution and has helpful features such as rotation visualization, rotation direction reversal, and the accommodation of user-specified delimiters. Moreover, the algorithm is platform portable to both Window and Linux operating systems. With its use, `rotatestl` may reduce the time necessary to change components of tessellated meshes requiring rotation about an axis or a hinge line. Thus, computational applications requiring many rotational changes may be enhanced through the use of the algorithm.

## 6.0 References

- [1] Kai, Chua Chee, and Leong Kah Fai. *Rapid Prototyping: Principles and Applications in Manufacturing*. New York: John Wiley & Sons, Inc., 1997.

## Bibliography

Greenwood, Donald T. *Principles of Dynamics*. 2<sup>nd</sup> ed. Englewood Cliffs: Prentice-Hall, Inc., 1988.

*MATLAB*. Vers. 7.11.0.584 (R2010b). The MathWorks, 16 Aug. 2010.  
<<http://www.mathworks.com/products/matlab/>>.

*Rhinoceros 3D*. Vers. 3.0 (SR5b). Robert McNeel & Associates, 2006. <[www.rhino3d.com](http://www.rhino3d.com)>.

# Appendix

## Source Code for `rotatest1`

```
function [] = rotatestl(world,parentFN,childrenFN,hp1FN,hp2FN,deflectionsFN,visualflagFN,reverseflagFN,delimiter)
%ROTATESTL STL file rotation algorithm
% ROTATESTL(world,parentFN,childrenFN,hp1FN,hp2FN,deflectionsFN, ...
% visualflagFN,reverseflagFN,delimiter)
```

```
%
% transforms a stereolithography (STL) file via rotation angles
% specified by the user. The user provides input through files in
% ASCII format.
```

```
%
% For each execution, ROTATESTL will generate a run folder called
% "runX," where X is an integer counting each run. For example, if
% ROTATESTL is executed for the 5th time, the folder "run5" will
% be generated. Within each runX folder, a sub-folder, containing
% rotated STL files, will be created for each parent STL file.
% Therefore, the file structure of the runX folder becomes
```

```
runX
|
|__ CHILD_NAME_1
|
|__ CHILD_NAME_2
.
.
.
|__ CHILD_NAME_N
```

```
%
% ROTATESTL will not overwrite a runX folder that all ready
% exists.
```

```
%
% When rotations are performed, a rotation of zero is not
% executed (even if a zero is specified as a desired rotation),
% as this would result in a file with points identical to those of the
% parent STL file. Furthermore, if identical rotation angles are
% specified for a given parent STL, the rotation angle will only be
% recorded once, and only one child STL file will be rotated for that
% angle. This enables the user to specify any set of deflections,
% whether repeated, non-repeated, or zero, for a parent STL. Only
% unique, nonzero rotation angles will be rotated. The user does not
% have to sort through a set of rotations. ROTATESTL will sort through
% the rotations automatically.
```

```
%
% Input to ROTATESTL:
```

```
% NOTE: All input files are saved in ASCII format. The
% extension of the input files does not matter, as long as the
% files are saved in ASCII format. CSV files (comma delimited,
% DOS format, and MAC format) are also supported. All input
% arguments to ROTATESTL are entered as strings except the world
% origin.
```

```
% world -world origin
% -argument assignment is a row vector of the form [x,y,z]
```

```
% Specify the world origin designated for the geometry at
% hand. Note that body coordinates are assumed for each STL
% such that:
```

```
% x is positive aft
% y is positive spanwise to the right (when viewing the
% geometry from the aft end forward)
% z is positive by the right-hand-rule crossing x and y (up)
```

```
% The origin does not necessarily have to be [0,0,0]. As
% long as the above convention is used, any origin may be
% specified.
```

```
% parentFN -file name with extension; refers to file containing
% original STL file names
% -argument assignment is a string
```

```
% parentFN is the file name, entered as a string, of a file
% containing the file names of each original STL file to be rotated.
% Place a single file name per line of parentFN in ASCII text
% with an extension of either <.stl> or <.STL>. For
% example, a file called 'parents.txt' with 5 original STL
% file names could have the following text:
```

```
% LINE 1: Bflap.stl
% LINE 2: Lelevon.stl
```

```

LINE 3:      Relevon.STL
LINE 4:      Ltail.stl
LINE 5:      Rtail.STL

```

Note the included use of the two different STL extensions. ROTATESTL will report an error if more than one period or decimal <.> is included in any of the file names listed in parentFN. An error will also be reported if any of the extensions of .stl or .STL are incomplete or disordered (for example .st or .TSL).

```

childrenFN  -file name with extension; refers to file
             containing child names used to label rotated STL files
             -argument assignment is a string

```

childrenFN is the file name, entered as a string, of a file containing a list of generic names that will be assigned to each rotated STL file. Each line in childrenFN corresponds to a label for the respective line in parentFN. For instance, using the above parentFN example, a file called 'children.txt' could have the following input:

```

LINE 1:      bflap
LINE 2:      Lelevon
LINE 3:      Relevon
LINE 4:      Ltail
LINE 5:      Rtail

```

Any text may be entered as a label for each line of childrenFN, provided it does not violate file name restrictions of the operating system. A generic name will be assigned to each rotated STL file via the following format:

```
<CHILD_NAME><DEFLECTION ANGLE><deg.stl>
```

If the file Bflap.stl is rotated by -10.5 degrees, the resulting rotated file name will become

```
<bflap-10.5deg.stl>
```

```

hp1FN, hp2FN -file name with extension; refers to file
              containing hinge line coordinates
              -argument assignment is a string

```

Provide point coordinates of 2 points on the hinge line designated for each surface such that

```

hp1      is a matrix of all "first" points on the hinge line
hp2      is a matrix of all "second" points on the hinge line

```

Each matrix contains the Cartesian coordinates of each point of each respective hinge line. Enter each matrix via the following format in the files hp1FN and hp2FN:

```

hp1FN:          hp2FN:
  XP1_1,YP1_1,ZP1_1      XP2_1,YP2_1,ZP2_1
  XP1_2,YP1_2,ZP1_2      XP2_2,YP2_2,ZP2_2
  .                      .
  .                      .
  .                      .
  XP1_n,YP1_n,ZP1_n      XP2_n,YP2_n,ZP2_n

```

where each row of hp1 or hp2 corresponds to the coordinates of points on the hinge line of the nth parent STL file. The coordinates may be separated along a line by a space or a specified delimiter.

Note that a positive rotation by the right-hand-rule will point in the positive direction of the hinge line vector. Placing one's thumb in the direction of hp1, the rotation of the STL will follow the rotation of one's hand while following the right hand rule. Therefore, positive input deflection angles point in the direction of hp1 by the right hand rule (by the convention of the algorithm):

```
hp1 <----- hp2
```

```

deflectionsFN -file name with extension; refers to file
               containing deflection angles in DEGREES

```



-argument assignment is a string

Specify any number of deflection angles, in DEGREES, for each parent STL file, implicitly represented by each column of deflectionsFN. For example, column 1 of deflectionsFN represents all deflections for parent STL of LINE 1 of parentFN. The deflection angles for a given parent STL are listed along the rows of each respective column. The input to deflectionsFN must be entered such that a rectangular matrix may be read. If some parent STL files have a different number of scheduled rotations, fill any remaining columns with zero [0].

If the above parent STL files were to be used, a possible format to a file deflectionsFN named 'deflections.txt' could be

	bflap	Lelevon	Relevon	Ltail	Rtail
LINE 1	DEGREES	DEGREES	DEGREES	DEGREES	DEGREES
LINE 2	DEGREES	DEGREES	DEGREES	DEGREES	DEGREES
LINE 3	DEGREES	0	DEGREES	DEGREES	DEGREES
LINE 4	DEGREES	0	DEGREES	DEGREES	DEGREES
LINE 5	DEGREES	0	0	DEGREES	DEGREES
LINE 6	DEGREES	0	0	DEGREES	0
LINE 7	DEGREES	0	0	0	0

Actual input begins on LINE 1. White space and specified delimiters are ignored.

visualflagFN -file name with extension; refers to file containing decision flags to show visualization  
-argument assignment is a string

Choose a visualization option by specifying a value for visualFLAG:

visualFLAG = 0 for no visualization  
visualFLAG = 1 to show a rotation  
Default is visualFLAG = 0 (no visualization)

The rotation shown will be the maximum value deflection for the chosen parent STL file and is relative to the assigned direction of the hinge line vector based upon the user-defined sign of rotation and the vector convention of the algorithm (as noted above in the hinge line input file description).

Visualization flag input is expected by the algorithm. White space and specified delimiters are ignored. A simple way to input the flags to visualflagFN is simply to place a single flag (0 or 1) per row for each respective parent STL file. There should be exactly the same number of flags in visualflagFN as there are parent STL files.

For ease of viewing, zoom with a box first, then rotate. Double click while zoom or rotate is enabled to reset the original view.

reverseflagFN -file name with extension; refers to file containing decision flags to reverse direction of rotation  
-argument assignment is a string

If the rotation made by ROTATESTL is opposite that of the desired rotation direction, reverseFLAG can be set to reverse the direction of rotation for the nth parent STL file.

Choose a reverse option by specifying a value for reverseFLAG:

reverseFLAG = 0 for no change in rotation direction  
reverseFLAG = 1 to reverse direction of rotation  
Default is reverseFLAG = 0 (no rotation reversal)

Reverse flag input is expected by the algorithm. White space and specified delimiters are ignored. A simple way to input the flags to reverseflagFN is simply to place a single flag (0 or 1) per row for each respective parent

```

%
% STL file. There should be exactly the same number of flags
% in reverseflagFN as there are parent STL files.
%
% delimiter -delimiter character used within input files
% -argument assignment is a string
%
% By default, whitespace is ignored within each of the ASCII
% input files. If a different delimiter is used, specify it
% as a string input. If a delimiter is specified, it applies
% to all ASCII files such that both white space and the
% desired delimiter will be ignored.
%
% The delimiter string must be specified as input. The
% following are common delimiters:
%
% ' ' -space or whitespace
% ',' -comma; use for .csv files
% '#' -pound symbol
%
% Because white space is ignored automatically, entering the
% comma delimiter string will be valid input for any ASCII
% file, including .csv files, that includes white space
% delimiters only, comma delimiters only, or both white
% space and comma delimiters combined.
%
% Written by James Tancred, Air Force Research Laboratory
% Last Modified: 27 AUGUST 2012
%
%===== READ INPUT FILES =====
%===== Read STL parent name storage file =====
stlparent = readnames(parentFN,'stlparent','file');
%Check for ".stl" extension for the filename
[mp np] = size(stlparent);
parentGO = 0; %initialize proceed flag for parent files; 1 = proceed
parenttally = 0; %Initialize counter for proper input file names
for plcv = 1:np
    %Get length of current recorded filename
    parentlength = length(stlparent(plcv).file);
%
%Sample end of filename to look for .stl file extension
%Find period <.> of file extension
decimalcount = 1;
walk1 = 1;
decimalfound = 0;
while(walk1 <= parentlength && ~decimalfound)
    getchar = stlparent(plcv).file(walk1);
%
    if(~strcmp(getchar, '.') && walk1 < parentlength)
        decimalcount = decimalcount + 1;
        walk1 = walk1 + 1;
%
    elseif(~strcmp(getchar, '.') && walk1 == parentlength)
        walk1 = walk1 + 1;
%
    elseif(strcmp(getchar, '.') && walk1 <= parentlength)
        decimalfound = 1;
%
    else
        end
end
%Assess filename
if(decimalfound)
    sremain = parentlength - decimalcount;
%3 characters exist past period of the extension
if(sremain == 3)
    decimalchar = stlparent(plcv).file(decimalcount);
    extchar1 = stlparent(plcv).file(decimalcount + 1);
    extchar2 = stlparent(plcv).file(decimalcount + 2);
    extchar3 = stlparent(plcv).file(decimalcount + 3);
%
    %If <.stl> or <.STL> extension exists, take note that the
    %proper extension was included
    if(strcmp('.stl',[decimalchar extchar1 extchar2 extchar3]) ...
        || strcmp('.STL',[decimalchar extchar1 extchar2 extchar3]))

```

```

    %Count number of properly input filenames
    parenttally = parenttally + 1;

    %If <.stl> or <.STL> extensions do not exist, tell the user.
    elseif(~strcmp('.stl',[decimalchar extchar1 extchar2 extchar3]) ...
        || ~strcmp('.STL',[decimalchar extchar1 extchar2 extchar3]))

        fprintf('\n\n    +++ Error in file <%s> ++++',parentFN);
        fprintf('\n    File with name [%s] does not have an extension of <.stl> or <.STL>.\n',stlparent(plcv).file);
        fprintf('    Please include <.stl> or <.STL> extension to the end of the parent filename.\n');

    else
    end

    %Fewer than 3 characters exist past period of the extension.
    %Proper extension does not exist. Report to user.
    elseif(sremain < 3)
        fprintf('\n\n    +++ Error in file <%s> ++++',parentFN);
        fprintf('\n    File with name [%s] does not have an extension of <.stl> or <.STL>.\n',stlparent(plcv).file);
        fprintf('    Please include <.stl> or <.STL> extension to the end of the parent filename.\n');

    %Greater than 3 characters exist past period of the extension.
    %Proper extension does not exist. Report to user.
    elseif(sremain > 3)
        fprintf('\n\n    +++ Error in file <%s> ++++',parentFN);
        fprintf('\n    File with name [%s] does not have an extension of <.stl> or <.STL>.\n',stlparent(plcv).file);
        fprintf('    Please include <.stl> or <.STL> extension to the end of the parent filename.\n');

    end

    %Period of extension not found. Report to user.
    elseif(~decimalfound)
        fprintf('\n\n    +++ Error in file <%s> ++++',parentFN);
        fprintf('\n    File with name [%s] does not have an extension of <.stl> or <.STL>.\n',stlparent(plcv).file);
        fprintf('    Please include <.stl> or <.STL> extension to the end of the parent filename.\n');

    else
    end

end

%If all recorded file names include a file extension, set parent file name
%proceed flag to 1. (If parentGO == 1, then file name input is entered
%as required).
if(parenttally == np)
    parentGO = 1;
else
end

%===== Read STL child name storage file =====
stlchildname = readnames(childrenFN,'stlchildname','name');

%===== Read first set of hinge points =====
fid = fopen(hp1FN,'rt');
hp1cell = textscan(fid,'%f %f %f','Delimiter',delimiter);
fclose(fid);

hp1 = cell2mat(hp1cell);    %convert to matrix format

%===== Read second set of hinge points =====
fid = fopen(hp2FN,'rt');
hp2cell = textscan(fid,'%f %f %f','Delimiter',delimiter);
fclose(fid);

hp2 = cell2mat(hp2cell);    %convert to matrix format

%=====
%Check deflection file to determine if all columns are filled with input
%=====
[badrowstore] = checkcolumns(deflectionsFN,np,delimiter);

%Do not proceed if deflections are not input properly
if(~isempty(badrowstore))
    fprintf('\n\n    It appears that deflection angles may not be specified for all columns\n');
    fprintf('    along (at least) one row of the deflections file. Please include a deflection\n');
    fprintf('    for all columns of all rows. The number of columns entered must equal\n');
    fprintf('    the number of parent .stl files to be rotated. If not all parent .stl files\n');
    fprintf('    will experience the same number of rotations, fill each empty column with a\n');
    fprintf('    zero [ 0 ].\n');
    fprintf('    Rows without all columns filled with input are:\n\n');
end

```

```

for badlcv = 1:length(badrowstore)
    fprintf('          line %d\n',badrowstore(badlcv));
end

fprintf('\n      in the file\n\n');
fprintf('          <%s>\n\n',deflectionsFN);
fprintf('\n\n      Program terminated.\n\n');

%Proceed if deflections are input properly
else

    %===== Read deflections =====
    %Create the format for scanning the deflections based upon the number of
    %parent STL files. The number of parent STL files is equal to the number of
    %surfaces requiring N deflections.
    inputscan = [];
    scantype = '%f';
    space = ' ';
    for lcv3 = 1:np
        if(lcv3 ~= np)
            inputscan = [inputscan scantype space];

        elseif(lcv3 == np)
            inputscan = [inputscan scantype];

        else
            end
    end

    fid = fopen(deflectionsFN,'rt');
    deflectionsCell = textscan(fid,inputscan,'Delimiter',delimiter);
    fclose(fid);

    deflectionsMAT = cell2mat(deflectionsCell);    %convert to matrix format

    %===== Read visual flags =====
    fid = fopen(visualflagFN,'rt');
    visualCell = textscan(fid,'%f','Delimiter',delimiter);
    fclose(fid);

    visualMAT = cell2mat(visualCell);    %convert to matrix format

    %===== Read reverse flags =====
    fid = fopen(reverseflagFN,'rt');
    reverseCell = textscan(fid,'%f','Delimiter',delimiter);
    fclose(fid);

    reverseMAT = cell2mat(reverseCell);    %convert to matrix format

    %Place deflections, visual flags, and reverse flags into structure arrays
    for lcv4 = 1:np
        %Deflections structure
        %Keep only only nonzero deflections. Record only unique deflections.
        deflections(lcv4).hinge = nonzeros(unique(deflectionsMAT(:,lcv4)))';

        %Visual flag structure
        visual(lcv4).choice = visualMAT(lcv4);

        %Reverse flag structure
        reverseFLAG(lcv4).reverse = reverseMAT(lcv4);

    end

%===== END READING INPUT =====

%==== Get size of inputs ====
[mparent nparent] = size(stlparent);    %stlparent filetype is a structure
[mchild nchild] = size(stlchildname);    %stlchildname filetype is a structure
[mp1 np1] = size(hp1);    %hp1 is a matrix
[mp2 np2] = size(hp2);    %hp2 is a matrix
[mdef ndef] = size(deflections);    %deflections filetype is a structure

%Check user input
if((nparent == nchild) && (nparent == mp1) && (nparent == mp2) && (nparent == ndef))

    if(parentGO == 1)

        %Verify that parent .stl files are in the present working
        %directory
        absentstl = 0;
        for index = 1:nparent

```

```

ncheck = stlparent(index).file(1,:);

if(exist(ncheck,'file') == 2)
    %Allow program to proceed
else
    fprintf('\n\n    File <%s> not detected in present working directory.\n',ncheck);
    absentstl = absentstl + 1;
end
end

if(absentstl)
    %Proceed no further - absent .stl files detected.

    %Continue - all .stl files detected
elseif(~absentstl)
    tic

    %==== Initialize Visualization Parameters
    viscount = 0;
    for k1 = 1:nparent
        tally(k1).figureup = 0;    %counter to denote that a figure has been created for a rotation
        viscount = viscount + visual(k1).choice;    %number of figures to create
    end
    visfignum = [1:viscount];    %vector numbering each figure
    viswalk = 1;    %counter to walk through visfignum

    %==== Generate folder system to store new .stl files====
    %Create new run folder for each run of this function:
    matchcount = 1;

    %Check to see if directory runX exists, search for runX directory
    eval(['stat = exist('' '' 'run' num2str(matchcount) '' '' ',' '' '' 'dir' '' '' ');']);

    %If runX directory exists, continue to search for runX until a runX folder
    %is no longer found
    while(stat == 7)

        %If runX does not exist, do not update counter
        if(stat ~= 7)

            %If runX does exist, update counter, continue searching
            else

                matchcount = matchcount + 1;
                eval(['stat = exist('' '' 'run' num2str(matchcount) '' '' ',' '' '' 'dir' '' '' ');']);
            end
        end
    end
    %all runX directories identified

    %Create a new runX directory that does not have the same name as the other
    %run directories (tag counter number to "run" name)
    eval(['mkdir('' '' 'run' num2str(matchcount) '' '' ');']);

    %Tell user what storage folder is created
    folder1 = 'run';
    folder2 = num2str(matchcount);
    folder = [folder1 folder2];
    fprintf('\n    Storage folder for new .stl files created: %s\n\n',folder);

    %Make subdirectories for each component
    for sublcV = 1:nchild
        eval(['mkdir('' '' 'run' num2str(matchcount) '' '' ',' '' '' stlchildname(sublcV).name(1,:) '' '' ');']);
    end

    % %==== Define the hinge line vector ====
    %Define hinge position vectors relative to world body origin
    for wlcV = 1:nparent
        for rlcV = 1:3;
            worldmat(wlcV,rlcV) = world(rlcV);
        end
    end

    r = hp2 - worldmat;
    r0 = hp1 - worldmat;

    %*****
    h = r - r0;    %hinge line vector r - r0 in world body c-syst

    hreverse = r0 - r;    %reversed-direction hinge line vector for reverse rotation option
    %*****

```

```

%=====
%       Perform deflections and generate new .stl with rotated geometry
%=====

%==== Parent STL loop
for plcv = 1:nparent

    %=== Extract data for current parent .stl deflection schedule ===

    %Get hinge line
    if(reverseFLAG(plcv).reverse == 0)
        %Reverse option OFF
        hi = h(plcv,:);           %extract hinge line

    elseif(reverseFLAG(plcv).reverse == 1)
        %Reverse option ON
        hi = hreverse(plcv,:);   %extract hinge line

    else
        %Tell user to enter proper value for reverse
        %option
        fprintf('\n\n    The value for the reverse flag must be either 0 or 1.');
```

```

        fprintf('\n    Ensure that the entries in the reverse flag file are entered correctly.\n\n');
    end

    p2 = hp2(plcv,:);           %extract p2 (used for translation vector)
    filename = stlparent(plcv).file(1,:);   %get parent .stl filename
    child = stlchildname(plcv).name(1,:);   %get child name

    %===== Read the parent .stl file =====
    %Open the .stl file
    fidold = fopen(filename,'rt');

    count = 1;                 %initialize a counter to aid in the storage of vertices
    normalcount = 1;          %initialize a counter for surface normal storage

    while(~feof(fidold))
        %Read the next occurring delimited string
        readline = textscan(fidold,'%s',1);

        %Get size currently read string
        [mread nread] = size(readline{1});

        %ID leading whitespace
        if(mread == 0)
            %Do nothing if leading whitespace is read

            %ID "solid name" label
        elseif(strcmp(readline{1},'solid'))
            %If first line is read, read the rest of the first line
            readline = textscan(fidold,'%s',1);
            stllabel = readline{1};   %record the label in .stl file

        elseif(strcmp(readline{1},'facet'))
            %Read the rest of "facet normal" line
            readline2 = textscan(fidold,'%s %f %f %f');

            %Record surface normal values
            normalvals = cell2mat(readline2);
            normalStore(normalcount,:) = normalvals;

            %Read "outer loop" label
            readline3 = fgetl(fidold);

            %Read vertex points
            c = textscan(fidold,'%s %f %f %f',3);
            vert = cell2mat(c);           %temporarily store vertex points of current face
            vl(normalcount,:) = vert(1,:); %store first vertex point for normal calculations

            %Store Vertices
            vertRow = 1;   %row counter for variable "vert"
            for row = count:(count + 2)
                for col = 1:3
                    %Store current verticies
                    vertStore(row,col) = vert(vertRow,col);
                end
                vertRow = vertRow + 1;
            end
            count = count + 3;   %"count" jumps by 3 to allow printing, in
            %vertStore, of vertices in sequence as they

```

```

%are read

%Read entire "endloop" label line
readline3 = fgetl(fidold);
%Read entire "endfacet" label line
readline4 = fgetl(fidold);

%Update surface normal counter
normalcount = normalcount + 1;

%ID "endsolid" label
elseif(strcmp(readline{1},'endsolid'))
    %If this line is read, do nothing.

else
end
end

%Close original .STL file
status = fclose(fidold);

%===== End read parent .stl =====

%Deflection loop
for deflcv = 1:length(deflections(plcv).hinge)

    magh = sqrt(hi(1).^2 + hi(2).^2 + hi(3).^2);    %magnitude of h

    %Direction cosines to world body primary directions
    alph = (180/pi)*acos(hi(1)./magh);    %angle between h and x-world vectors, degrees
    bet = (180/pi)*acos(hi(2)./magh);    %angle between h and y-world vectors, degrees
    gam = (180/pi)*acos(hi(3)./magh);    %angle between h and z-world vectors, degrees

    %===== Catch angles that cause singularities in calculations ====
    %ALPHA = 90 DEG, GAMMA = 0, 180 DEG

    %Check ALPHA
    if(alph == 90)
        alph = 90 - 0.000001;
    else
    end

    %Check GAMMA
    if((~gam) || (gam == 180))
        %GAMMA = 0 DEG
        if(~gam)
            gam = gam + 0.000001;
            %GAMMA = 180 DEG
        elseif(gam == 180)
            gam = 180 - 0.000001;
        else
        end
    else
    end

    %Define translation vector from world origin to a point on hinge line
    delta = [p2 - world]';    %column translation vector

    %First rotation angle
    thetal = (180/pi)*acos(((cosd(alph)^2) + (sind(gam)^2) - (cosd(bet)^2))/(2*cosd(alph)*sind(gam)));    %degrees

    %Second rotation angle
    theta2 = 90 - gam;    %degrees

    %===== Determine octant to which the hinge line vector points ====
    %OCTANT 1
    if((hi(1) >= 0) && (hi(2) >= 0) && (hi(3) >= 0))
        %Do nothing

        %OCTANT 2
    elseif((hi(1) < 0) && (hi(2) >= 0) && (hi(3) >= 0))
        %Do nothing

        %OCTANT 3
    elseif((hi(1) < 0) && (hi(2) < 0) && (hi(3) >= 0))
        thetal = -thetal;

        %OCTANT 4
    elseif((hi(1) >= 0) && (hi(2) < 0) && (hi(3) >= 0))
        thetal = -thetal;

```

```

    %OCTANT 5
elseif((hi(1) >= 0) && (hi(2) >= 0) && (hi(3) < 0))
    theta2 = -theta2;

    %OCTANT 6
elseif((hi(1) < 0) && (hi(2) >= 0) && (hi(3) < 0))
    theta2 = -theta2;

    %OCTANT 7
elseif((hi(1) < 0) && (hi(2) < 0) && (hi(3) < 0))
    theta1 = -theta1;
    theta2 = -theta2;

    %OCTANT 8
elseif((hi(1) >= 0) && (hi(2) < 0) && (hi(3) < 0))
    theta1 = -theta1;
    theta2 = -theta2;

else
end

%===== Calculate a point at the end of surface normals =====
[mv1 nv1] = size(v1);

for normlcv = 1:mv1
    %Position vector from the origin to vertex 1 of a given
    %face
    rv1i = v1(normlcv,:) - world;

    %Equation of line in direction of normal vector, staring at
    %vertex 1; Equivalently the position vector from the origin
    %to point P on a line passing through the normal vector and
    %vertex 1
    rpi = rv1i + normalStore(normlcv,:);

    %Point P along line described by above assignment
    Px(normlcv) = world(1) + rpi(1);
    Py(normlcv) = world(2) + rpi(2);
    Pz(normlcv) = world(3) + rpi(3);
end

%===== Perform a rotation =====

thetaH = deflections(plcv).hinge(deflcv);    %get deflection angle

%Rotation Matrix 1
L1 = [cosd(theta1) sind(theta1) 0;...
      -sind(theta1) cosd(theta1) 0;...
      0 0 1];

%Rotation Matrix 2
L2 = [cosd(theta2) 0 sind(theta2);...
      0 1 0;...
      -sind(theta2) 0 cosd(theta2)];

%Rotation Matrix 3
L3 = [1 0 0;...
      0 cosd(thetaH) sind(thetaH);...
      0 -sind(thetaH) cosd(thetaH)];

%Execute rotation of vertex points
[rowVert colVert] = size(vertStore);
for rowlcv = 1:rowVert
    for collcv = 1:colVert
        if(collcv == 1)
            x = vertStore(rowlcv,collcv);    %get x
        elseif(collcv == 2)
            y = vertStore(rowlcv,collcv);    %get y
        elseif(collcv == 3)
            z = vertStore(rowlcv,collcv);    %get z
        else
            end
        end
    end

    %Get a vertex relative to world body coordinates
    Xb = [x y z]';

    %ROTATION TRANSFORMATION
    newVertices(rowlcv,:) = inv(L1)*inv(L2)*L3*L2*L1*(Xb - delta) + delta;
end

```



```

%Execute rotation of surface normals
for rotnlcv = 1:mvl

    XbP = [Px(rotnlcv) Py(rotnlcv) Pz(rotnlcv)];
    Xbv1 = [v1(rotnlcv,1) v1(rotnlcv,2) v1(rotnlcv,3)];

    %ROTATION TRANSFORMATION
    newP = inv(L1)*inv(L2)*L3*L2*L1*(XbP - delta) + delta;
    newv1 = inv(L1)*inv(L2)*L3*L2*L1*(Xbv1 - delta) + delta;

    %Calculate new normal vector
    newnorm = newP - newv1;

    %Store the new unit normal vector
    normal(rotnlcv,:) = newnorm./norm(newnorm);
end

%===== Print new STL =====
[mnorm nnorm] = size(normal);
vertcount = 1;
s1 = child;

s2 = num2str(thetaH);
s3 = 'deg.stl';
stringIN = [s1 s2 s3];
eval(['fidnew = fopen('' eval(['stringIN]) '' ',' '' 'wt' '' ');']);

%==== Print to new .stl file ====
fprintf(fidnew,'solid ');
fprintf(fidnew,'%s\n',stllabel{1}); %include original label of .stl file
for nlcV = 1:mnorm
    fprintf(fidnew,' facet normal %4.6e %4.6e %4.6e\n',normal(nlcV,1),normal(nlcV,2),normal(nlcV,3));
    fprintf(fidnew,' outer loop\n');
    for clcv = 1:3
        fprintf(fidnew,' vertex %4.6e %4.6e
%4.6e\n',newVertices(vertcount,1),newVertices(vertcount,2),newVertices(vertcount,3));

        if(clcv == 3 & nlcV == mnorm)
            else
                vertcount = vertcount + 1;
            end
        end
        fprintf(fidnew,' endloop\n');
        fprintf(fidnew,' endfacet\n');
    end
    fprintf(fidnew,'endsolid ');
    fprintf(fidnew,'%s\n',stllabel{1}); %include original label of .stl file

status = fclose(fidnew);

%===== Place new .stl in its respective component directory =====
eval(['movefile('' stringIN '' ',' '' 'run' num2str(matchcount) '/' child '' ');']);

%===== VISUALIZATION =====
%Generate plot of rotation (if specified by user)
if(visual(plcv).choice == 0)
    %Visualization not chosen

elseif((visual(plcv).choice == 1) && (max(deflections(plcv).hinge)) == thetaH && ~tally(plcv).figureup)

    %Visualization CHOSEN
    facevert = [1:rowVert]; %create counter vector of
                        %length equal to row dim of
                        %newVertices (row dim =
                        %rowVert)

    %Create coordinate plot sequence matrix (facemat)
    fcount = 1;
    for k2 = 1:(length(facevert)/3)
        for col = 1:3
            facemat(k2,col) = facevert(fcount);
            fcount = fcount + 1;
        end
    end

    %Plot a chosen rotation
    figure(visfignum(viswalk))
    patch('Vertices',vertStore,'Faces',facemat,'Edgecolor','k','Facecolor','b')
    axis equal

```

```

axis off
hold on
patch('Vertices',newVertices,'Faces',facemat,'Edgecolor','k','Facecolor','y')

%Plot origin
plot3(world(1),world(2),world(3),'r*')

%Label surfaces and origin
eval(['legend(' '' '\bf{Parent ' eval(['child']) ' STL (\delta = 0 Degrees)'} '' ', ' '' '\bf{\delta = '
' num2str(thetaH) ' Degrees (as Input by USER)'} '' ', ' '' '\bf{World Origin}' '' ');']);

%Update or Reinitialize counters
viswalk = viswalk + 1;
tally(plcv).figureup = 1;
clear facevert
clear facemat

else
end
%=====

end

%Re-initialize storage matrices
clear vertStore
clear newVertices
clear normal
clear normalStore
clear v1
clear Px
clear Py
clear Pz

end
fprintf('      Rotation Algorithm Performance: ');
toc
fprintf('\n\n');

end

elseif(~parentGO)
    fprintf('\n\n      Not all parent STL file names have been input correctly. ');
    fprintf('\n      Please check file extensions of parent STL file names.\n\n');
else
end

else
    fprintf('The number of parent .stl files provided determines the\n');
    fprintf('number of specified child names, the sizes of the hinge\n');
    fprintf('point matrices, and the size of the deflection structure.\n');
    fprintf('The number of rows in the hinge point matrices, the\n');
    fprintf('number of columns in the deflection structure, and the\n');
    fprintf('number of child names specified must be equal to the number\n');
    fprintf('of parent .stl files provided.\n\n');

    %ID child names not equal to number of parent .stl's
    if(nparent ~= nchild)
        fprintf('Number of specified names for child .stl files\n');
        fprintf('not equal to the number of parent .stl files provided.\n\n');
    else
    end

    %ID number of rows in hinge point 1 matrix not equal to number of parent .stl's
    if(nparent ~= mpl)
        fprintf('Number of rows for hinge point 1 matrix\n');
        fprintf('not equal to the number of parent .stl files provided.\n\n');
    else
    end

    %ID number of rows in hinge point 2 matrix not equal to number of parent .stl's
    if(nparent ~= mp2)
        fprintf('Number of rows for hinge point 2 matrix\n');
        fprintf('not equal to the number of parent .stl files provided.\n\n');
    else
    end

    %ID number of columns of deflection structure not equal to number of parent .stl's
    if(nparent ~= ndef)
        fprintf('Number of columns of deflection structure\n');
        fprintf('not equal to the number of parent .stl files provided.\n\n');
    end
end

```

```

        else
        end

    end

end

end

%=====
%                               SUBFUNCTIONS
%=====

function [badrowstore] = checkcolumns (FN,np,delimiter)

fidcheck = fopen(FN,'rt');
rowcount = 0;
badrow = 0;
badrowstore = [];
while(~feof(fidcheck))
    rowcount = rowcount + 1;
    readline = fgetl(fidcheck);

    walk2 = 1;
    foundinput = 0;

    while(walk2 <= length(readline))
        look = readline(walk2);

        %If the current character is a space, the specified delimiter,
        %or the newline character, continue to look for user input
        if(strcmp(look, ' ') | strcmp(look,delimiter) | strcmp(look,'\n'))
            walk2 = walk2 + 1;
        else

            lookahead = walk2 + 1;

            %If something other than the above condition is found and the
            %next character is greater than the length of the line, then
            %input is found at the last character of the current line
            if(lookahead > length(readline))
                foundinput = foundinput + 1;
                walk2 = walk2 + 1;

            %If something is detected and the next character is a space or
            %delimiter, then input is found
            elseif(strcmp(readline(lookahead),' ') | strcmp(readline(lookahead),delimiter))
                foundinput = foundinput + 1;
                walk2 = walk2 + 1;

            %If something is detected, and more is detected, continue to
            %walk until nothing more is detected
            elseif(~strcmp(readline(lookahead),' ') & ~strcmp(readline(lookahead),delimiter))
                walk2 = walk2 + 1;

            else
            end
        end
    end

    %If input is found (i.e. foundinput is nonzero) and the total input in
    %the row is not equal to the number of parent .stl files, store the row
    %to report to user for more input
    if(foundinput ~= np & foundinput ~= 0)
        badrow = badrow + 1;
        badrowstore(badrow) = rowcount;
    else
    end
end
fclose(fidcheck);

%==== END checkcolumns ====

function [str] = readnames(fname,strname,strfield)
%===== Read name storage file =====
fid = fopen(fname,'rt');
pnamecount = 1;
while(~feof(fid))
    readline = fgetl(fid);

    %Inspect current line
    recordcounter = 1;    %counter for the recording of filename characters
    for lcv1 = 1:length(readline)

```

```

%Extraction of current character of current line
examine = readline(lcv1);

%Check for white space. If it exists, do not record. Otherwise,
%record string.
if(strcmp(examine, ' '))
    %White space exists. Do not record.

elseif(lcv1 ~= length(readline) && ~strcmp(examine, ' '))
    %White space does not exist and not at end of readline. Record
    %string. Update recordcounter.
    namerecord(recordcounter) = examine;
    recordcounter = recordcounter + 1;

elseif(lcv1 == length(readline) && ~strcmp(examine, ' '))
    %White space does not exist and end of readline has been
    %reached. Record string. Do not update recordcounter.
    namerecord(recordcounter) = examine;

else
end

end

%==== Place current filename in a structure array ====
%If at the end of the file record nothing, do not update.
if(feof(fid) & ~exist('namerecord','var'))
    %Do not record, do not update

%If at the end of the file, but the name recorded is the end-of-file
%indicator from FGETL (i.e. the end of file is reached, FGETL
%returns a -1, and -1 is assigned to namerecord), then do not record;
%do not update.
elseif(feof(fid) & (exist('namerecord','var') == 1) & (readline == -1))
    %Do not record, do not update

%If end of the file is reached and if a filename has been correctly
%detected, record the filename in a structure array and update the
%structure counter. This will occur if a new line is not entered after
%the last name in the file.
elseif(feof(fid) & (exist('namerecord','var') == 1) & (readline ~= -1))
    %Place inspected filename string into structure array
    eval([strname '(' num2str(pnamecount) ')'. ' strfield ' = namerecord;]);

    pnamecount = pnamecount + 1;
    clear namerecord
    clear examine
    clear readline

%If not at the end of the file and a filename has been detected, record
%the filename in a structure array and update structure counter.
elseif(~feof(fid) & (exist('namerecord','var') == 1))
    %Place inspected filename string into structure array
    eval([strname '(' num2str(pnamecount) ')'. ' strfield ' = namerecord;]);

    pnamecount = pnamecount + 1;
    clear namerecord
    clear examine
    clear readline

else
    %Otherwise, do not record; do not update
end

clear readline
clear examine
end
fclose(fid);

```

```
%Storage structure array  
str = eval([strname]);  
  
%==== END readnames ====
```

## LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

### Acronyms

ASCII	: American Standard Code for Information Interchange
CAD	: Computer-Aided Design
CFD	: Computational Fluid Dynamics
FEA	: Finite Element Analysis
MATLAB	: MATrix LABoratory, numerical software language used to implement <code>rotatestl</code>
RP	: Rapid Prototyping
STL	: Stereolithography or Standard Tessellation Language

### Abbreviations

.csv	: Comma-Separated Values file format extension
.m	: MATLAB script file format extension
m-file	: MATLAB script file
.stl or .STL	: Stereolithography file format extension
.txt	: ASCII text document file format

### Symbols

$C_{p,i}$	= Cartesian coordinate x, y, or z of point p on the $j^{\text{th}}$ row of hinge point input files
$d_{j,k}$	= Specific angular deflection value, in degrees, for the $k^{\text{th}}$ row of the $j^{\text{th}}$ column of the <code>deflectionsFN</code> file
$\delta$	= Deflection angle, in degrees, noted in visualization figure legend
$i$	= Vertex of a specific triangle in an STL file; $i = 1, 2, 3$
$N$	= Total number parent STL files
$ND(j)$	= Total number of angular deflections scheduled for the $j^{\text{th}}$ parent STL file
$n_x$	= x-component of the surface normal vector of a specific triangle in an STL file
$n_y$	= y-component of the surface normal vector of a specific triangle in an STL file
$n_z$	= z-component of the surface normal vector of a specific triangle in an STL file
$X$	= Number of consecutive executions of <code>rotatestl</code> within the present working directory
$x$	= Cartesian x-coordinate in 3D space
$x_{vi}$	= x-component of the $i^{\text{th}}$ vertex for a triangle in an STL file
$y$	= Cartesian y-coordinate in 3D space
$y_{vi}$	= y-component of the $i^{\text{th}}$ vertex for a triangle in an STL file
$z$	= Cartesian z-coordinate in 3D space
$z_{vi}$	= z-component of the $i^{\text{th}}$ vertex for a triangle in an STL file

### Subscripts

$j$	= specific row of the <code>parentFN</code> file or specific column of the <code>deflectionsFN</code> file; $j = 1, 2, 3, \dots, N$
$k$	= specific row of the $j^{\text{th}}$ column of the <code>deflectionsFN</code> file; $k = 1, 2, 3, \dots, ND(j)$
$p$	= hinge point; $p = 1, 2$