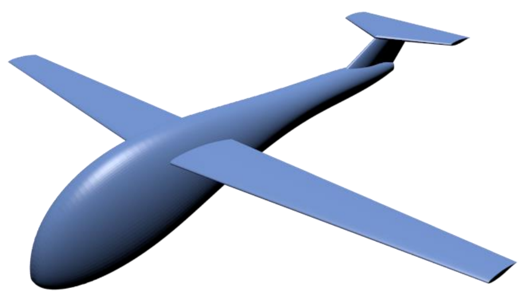
Northrop Grumman Student Design Project



CECS 491 - Team 4

Ben Kray
Bryce Burnett
Keith Farnham
Lisa Tran
Marvin Trajano
Victor Tran

Abstract

Currently, the available tools to create various aircraft components in a 3D modeling program is lacking in the degree of customizability and capability. Our project, presented by Phil Barnes of Northrop Grumman, was to create a program that would take advantage of the highly capable Blender 3D program. The final goal of this project was to be able to create aircraft components utilizing a parametric cubic spline and allow for 3D printable models. The parametric cubic spline, a piecewise curve that passes through a set of points while maintaining continuity, is taken advantage of in order to create curves that appear smooth to flowing air. This method of curve generation also reduces the number of required input points to create a component from hundreds on some components to just a few points. This simplification coupled with other features including the importing and exporting of CSV data, the ability to change the colors of individual components, and the amount of customization available for each component allows for a highly powerful aircraft modeling tool. With these features in mind, we utilized Python to create our Blender addon that allows for the generation of customizable aircraft components for 3D printing purposes.

Table of Contents

1. Introduction and Background

1.1 Statement of Problem Area

There exists software available to the public that can create aircraft objects such as NASA's VSP. However, current products such as VSP are limited in regards to integration of specific aircraft components as well as quality of graphics and customizability.

1.2 Previous and Current Work

In traditional aircraft modelling, with airfoils for example, hundreds of points would have to be implemented by hand around the component. Individuals would have to solve for the points and manually plot them and sketch out the curvature themselves. In this project, the parametric cubic splines allow for a much more efficient procedure, where the computer calculates a spline to fit around a small set of points.

1.3 Background

Blender is a 3D modeling software with very high capabilities (e.g. geometric features, higher quality, powerful rendering engine). Because of its open-source nature, Blender was found to be very applicable to scientific and engineering applications. In order to take advantage of Blender's superior graphics engine, Phil Barnes of Northrop Grumman sought for a team to create an application that will allow a user to modify multiple aspects of aircraft geometry for a 3D model in Blender. This 3D model could then be used for 3D printing.

1.4 Brief Project Description

This project utilizes Blender as an application platform that will be easily distributed to users. Users can create different 3D aircraft components with a few inputs, as a result of implementing the parametric cubic spline for the generation of geometries.

1.5 Purpose/Objectives/Justification of project

The main objectives are to learn Blender, Python, and theories of parametric cubic splines. With this knowledge, an application can be developed to allow the user to easily edit a file representing an aircraft that will be used mathematically model the geometry and visualize its shape. The user will then be able to 3D print the object(s).

1.6 Team Work Assignment and Accomplished

For this project we were able to generate 3D aircraft objects within Blender with editable geometric properties, create an easy to navigate UI, implement functionalities to import and export files representing data for aircraft geometries, focus the 3D view to specific components when selecting from a list, add color to the components, and generate a complete parametric cubic spline. Each team member was in charge of their own smaller, specific tasks and worked in small groups for larger tasks. The small groups consisted of large-scale tasks such as creating the UI, implementing CSV to import and export points, and the implementation of the parametric cubic spline for aircraft geometry.

2. System Functional Specification

2.1 Functions Performed

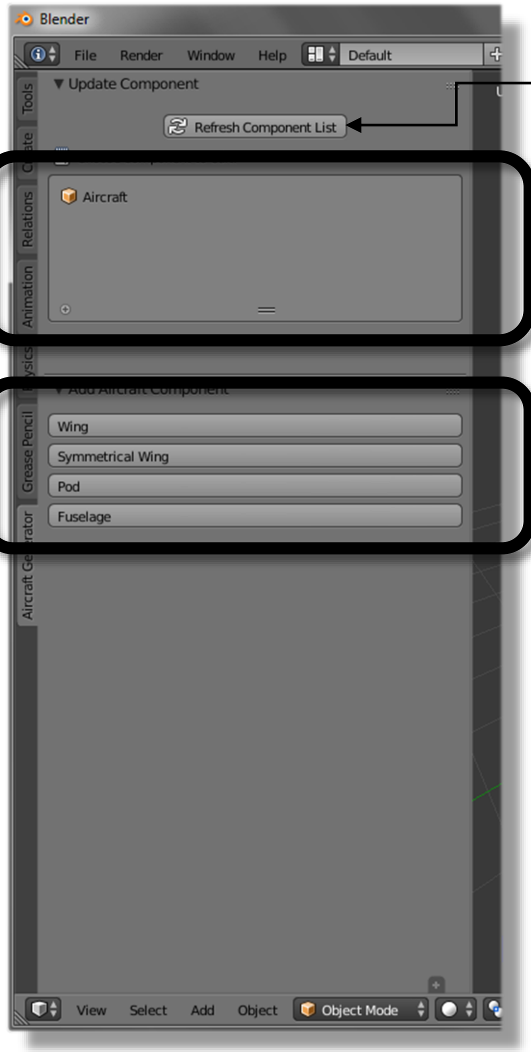
#	Name	Description
2.1.1	Add Wing	Generates and displays wing in the 3D view
2.1.2	Add Symmetrical Wings	Generates and displays symmetrical wings in the 3D view
2.1.3	Add Pod	Generates and displays pod in the 3D view
2.1.4	Add Fuselage	Generates and displays fuselage in the 3D view
2.1.5	Refresh Component List	Refreshes the list of all components within the 3D view
2.1.6	Import CSV	Imports points from a CSV file to generate an aircraft component
2.1.7	Add Color	Changes color and texture of the object (wing, symmetrical wings, pod, fuselage)
2.1.8	Update Wing	Updates and changes selecting wing based on geometric adjustments
2.1.9	Update Symmetrical Wings	Updates and changes selected symmetrical wing based on geometric adjustments
2.1.10	Update Pod	Updates and changes selected pod based on geometric adjustments
2.1.11	Update Fuselage	Updates and changes selected fuselage based on geometric adjustments
2.1.12	Delete Wing	Deletes wing from the 3D view
2.1.13	Delete Symmetrical Wings	Deletes symmetrical wings from the 3D view
2.1.15	Delete Pod	Deletes pod from the 3D view
2.1.16	Delete Fuselage	Deletes fuselage from the 3D view
2.1.17	Export to CSV	Exports points describing an aircraft component to a CSV file

2.2 User Interface Design

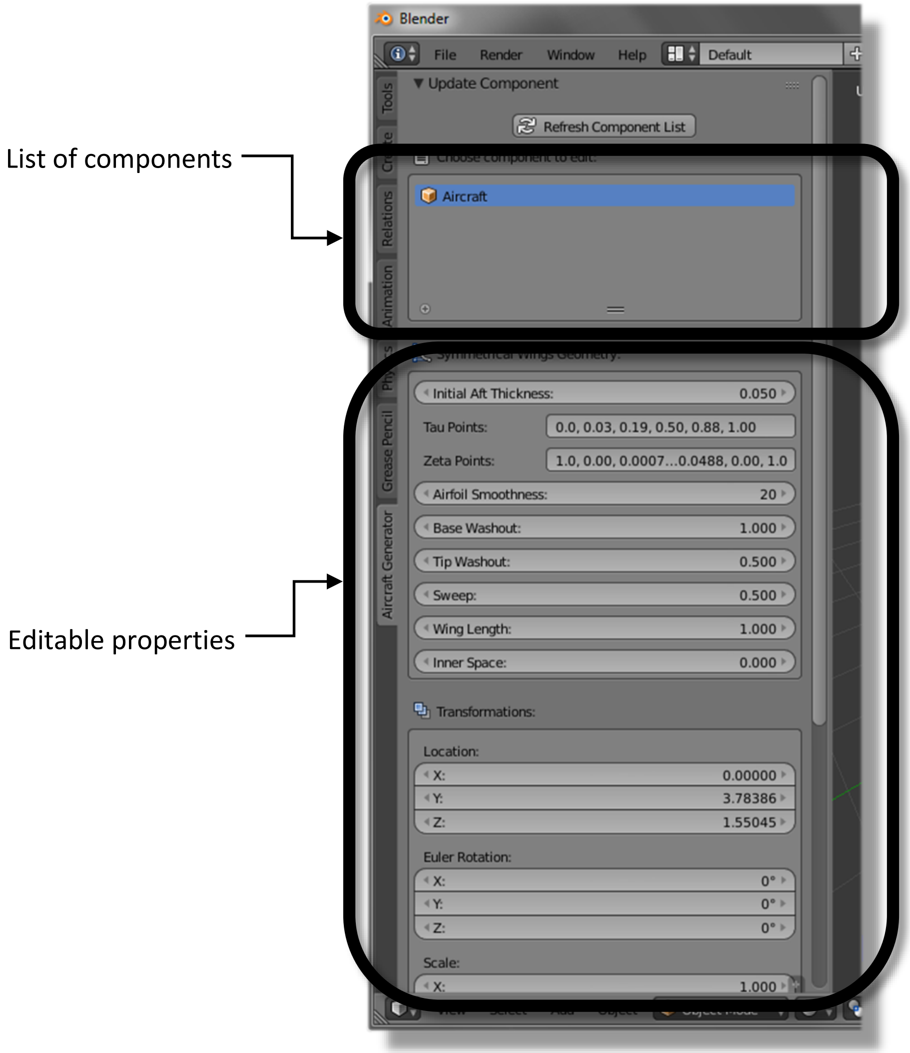
List of components

Refreshes the component list

Add component buttons

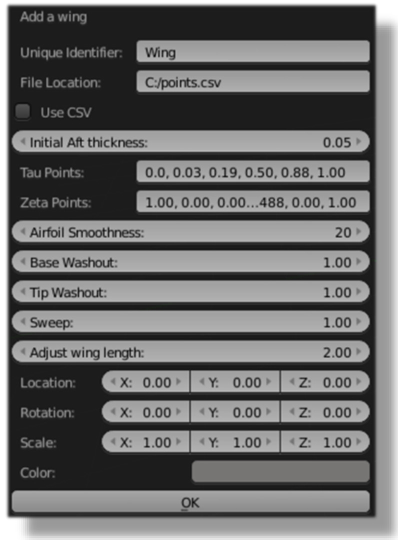


Before selecting a component



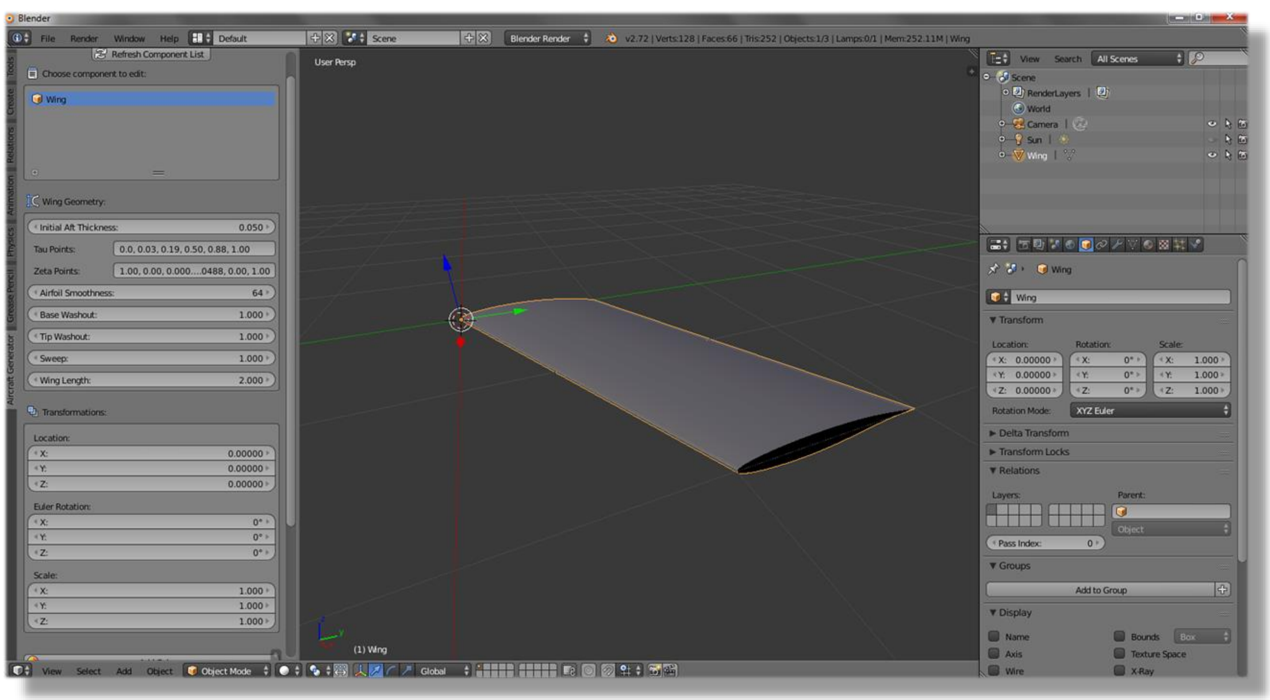
After selecting a component

2.2 User Input Preview



Preview of input dialog box for a new wing

2.3 User Output Preview



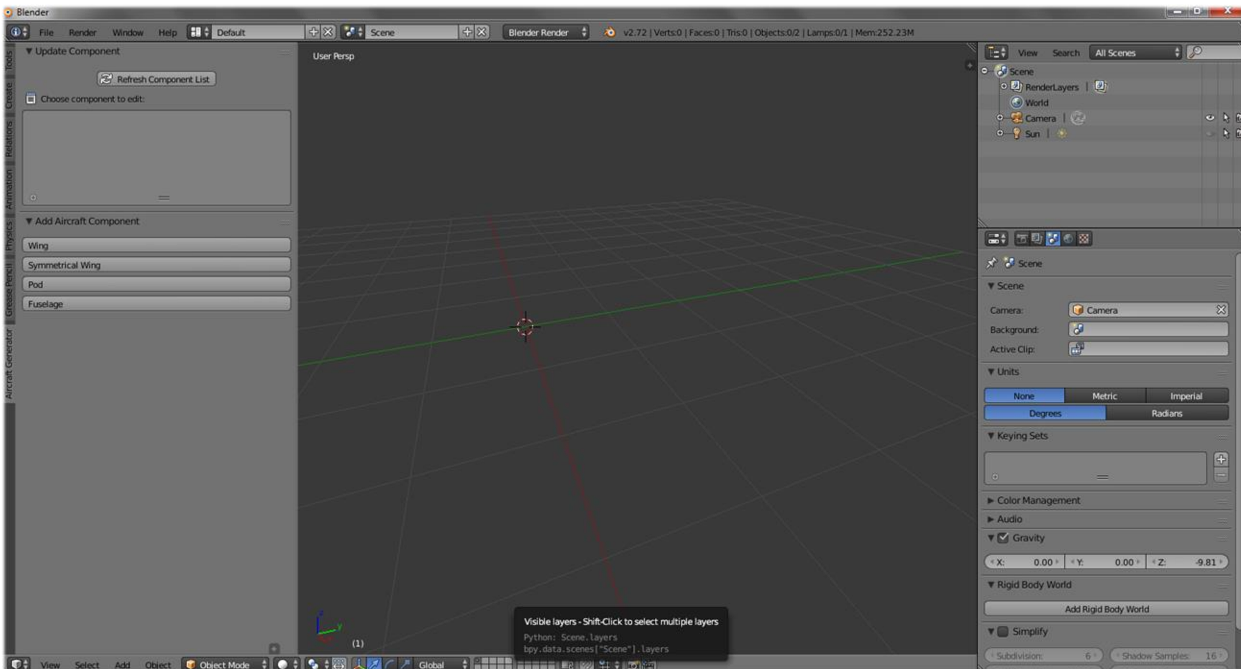
Preview of output in 3D View after adding a wing

2.4 User Interface Specification

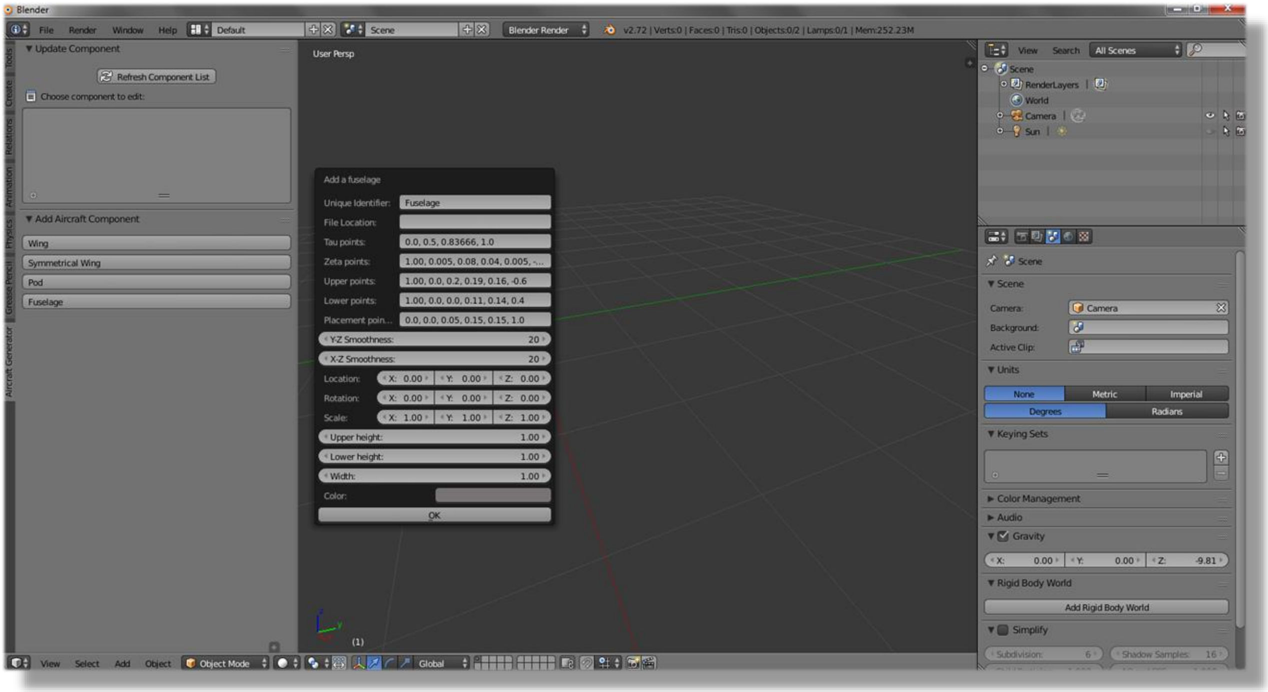
2.4.1 Interface Metaphor Model

Our interface can be compared to a remote control airplane. We have one module on the left-hand side that serves as a menu bar for the components. This can be seen as the remote control. We can compare our UI to having a remote control for each aircraft component: wing, symmetrical wings, pod, and fuselage. Each remote control can change the location, rotation, length, color, etc. for each component.

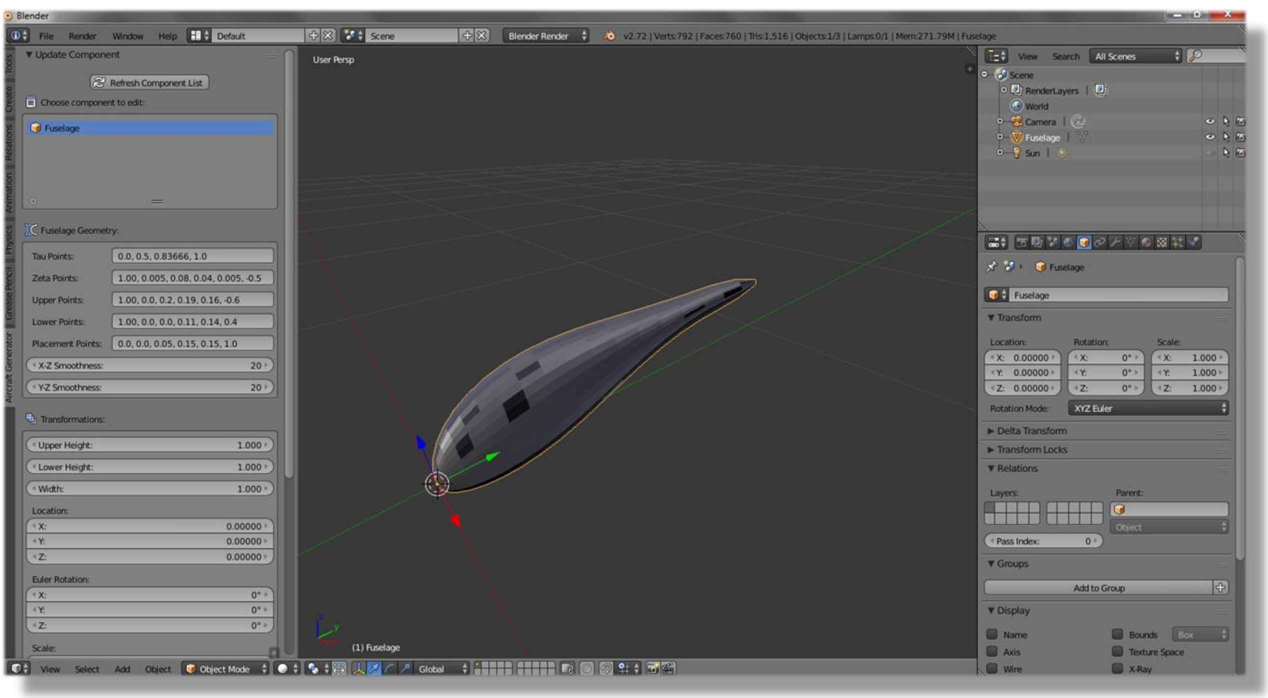
2.4.2 User Screens/Dialog



User screen upon first installing the addon



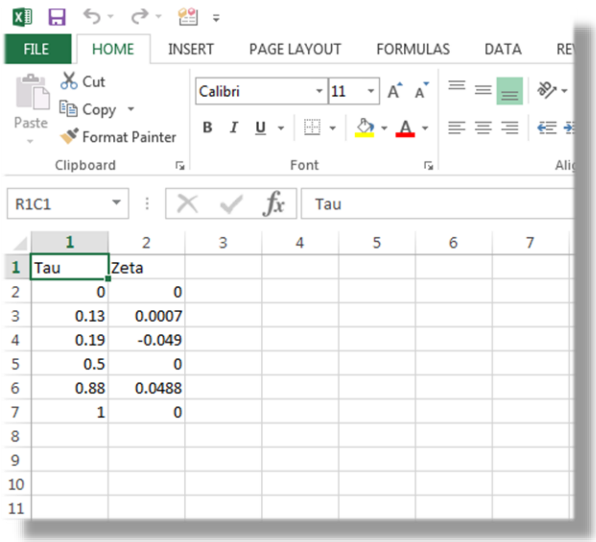
A dialog box for adding a fuselage. This dialog box allows you to input properties before generating the component.



After generating an object, the user can edit properties and update the component on the left.

2.4.3 Report Formats/Sample Data

Any reports from the system can be seen through the system console. Here is an example of importing points from a CSV sheet and having the console report a successful wing generation:



Sample data from a CSV spreadsheet

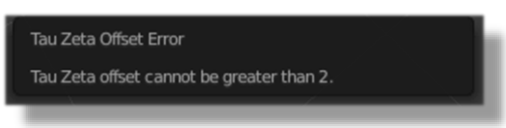
Successful report

2.4.4 On-line Help Material

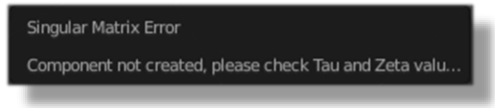
For installing an addon or additional help, visit our website which includes a user manual: www.csulb-ngcproject.me

2.4.5 Error Conditions and System Messages

An error message will pop up should there be any errors. Examples:



Error dialog message when Tau and Zeta are incompatible



Error dialog message when the points generate s singular matrix

System errors and messages can also be checked using the system console:

Sample system message when '0.19d' is input as a point

2.4.6 Control Functions

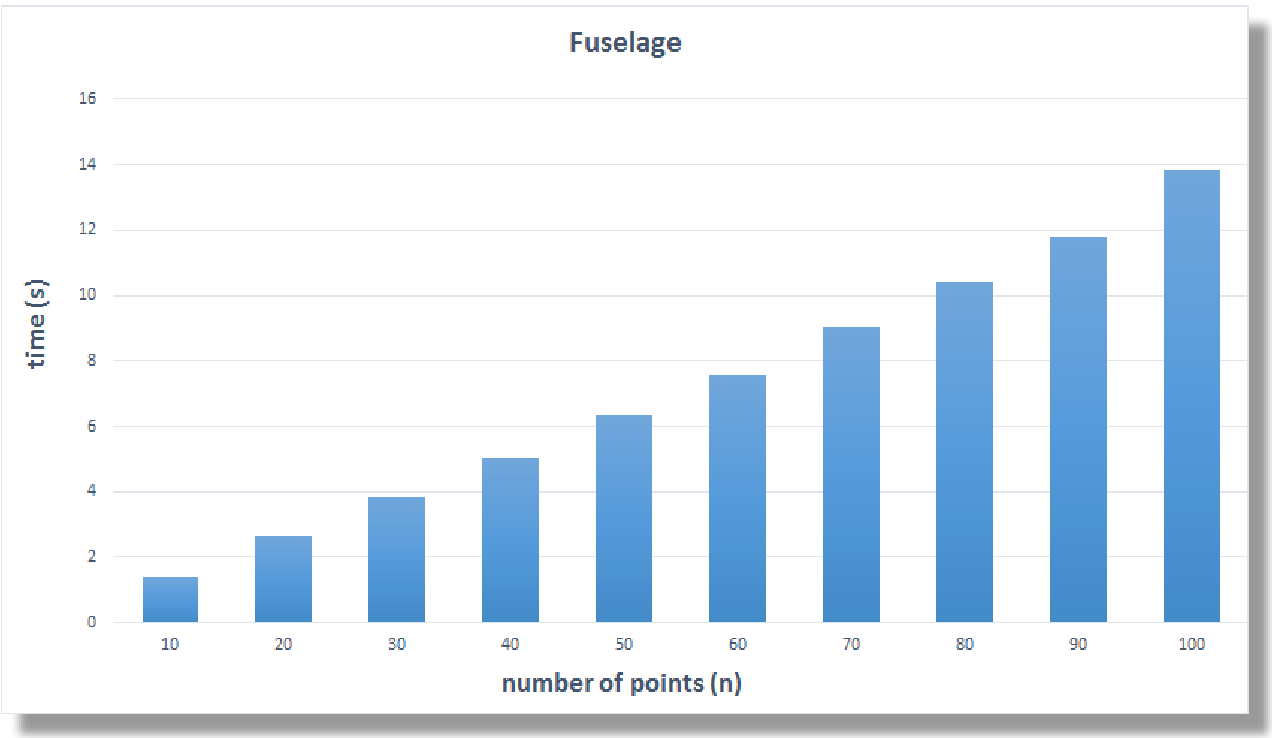
The user interface allows for the control of components through one easy to navigate GUI, on the left hand side of Blender, where all of our functions can be controlled from.

There are also special control functions in Blender:

- Right click to select and drag an object
- Left click to confirm
- Ctrl + z to undo
- F12 to render the scene

3. System Performance Requirements

3.1 Efficiency
While there were no explicit requirements in terms of the efficiency of the creation of the aircraft components, some degree of optimization was implicit. Aircraft components such as the wing, symmetrical wing, and pod are generated within one to three seconds, however, fuselage may take longer depending on the smoothness value input (albeit resulting in a higher quality 3D model).



3.2 Reliability
The aircraft addon package is designed to never cause a fatal error through the use of several try-except blocks (similar to C++/Java style try-catch blocks). The user is notified in the event of any type of error through the error dialog popup or the Blender console (see 2.4.5 Error Conditions and System Messages). Granted that an execution is user-error free, an aircraft shape will generate with 100% accuracy - that is, with N number of aircraft component creations with settings S will always produce the same shape.

3.2.1 Description of Reliability Measures
Having valid points (valid in the sense that the points are parseable floats or integers) isn't enough to guarantee a proper shape generation; a singular matrix error can occur. A singular matrix error is raised when the determinant of a matrix is zero; these types of errors are rare, and are usually raised when the user arbitrarily sets input points. User inputs Tau, Zeta, Upper, Lower, and Placement are validated explicitly via a validation module, and thus an invalid point

within any set of Tau, Zeta, Upper, Lower, or Placement is caught before Blender tries to create the aircraft shape. Other user input points are automatically validated by Blender.

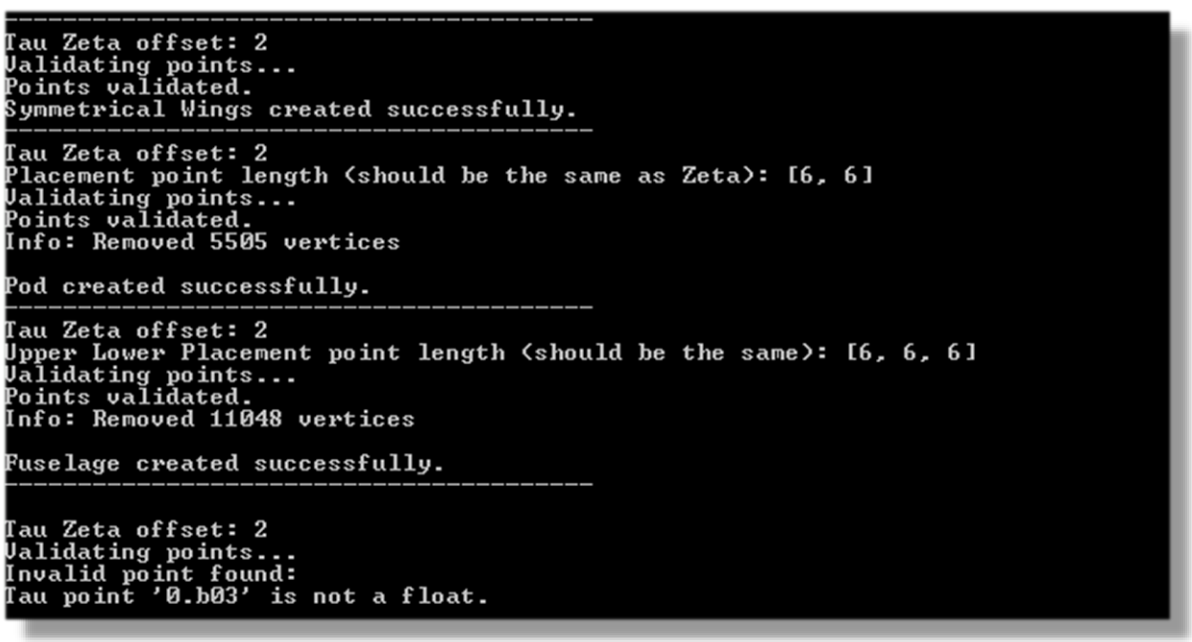
3.2.2 Error/Failure Detection and Recovery

Aircraft properties such as smoothness, base washout, tip washout, sweep, etc. are all either Blender float or integer properties, and thus Blender handles the error handling. For instance, in the event that a user sets Fuselage “X-Z Smoothness” to an incorrect value of “34bad”, Blender will catch the error and revert to the programmed default. Unlike the aforementioned properties, Tau, Zeta, Upper, Lower, and Placement points are all Blender string properties; Blender does not handle error checking on string properties, and thus the aircraft addon package includes Python modules to handle error checking via float/integer parsing.

The process of error detection is as follows (see Blender console image below):

1. Check to see if Tau/Zeta offset is 2 (Zeta points must always have 2 more points than Tau, these are constraint values)
    a. If creating a Fuselage or Pod, check to see if Upper/Lower/Placement has the same number of points as Zeta
2. Validate Tau/Zeta points by parsing each string (a ValueError exception will signify an invalid value)
    a. If creating a Fuselage or Pod, validate Upper/Lower/Placement points
3. Check matrices for a singular matrix error by a simple try-except block.

In the event that any of these checks return false, the appropriate error message will be displayed to the user, and the module will immediately return without creating the aircraft shape.



```
----------------------------------------
Tau Zeta offset: 2
Validating points...
Points validated.
Symmetrical Wings created successfully.
----------------------------------------
Tau Zeta offset: 2
Placement point length (should be the same as Zeta): [6, 6]
Validating points...
Points validated.
Info: Removed 5505 vertices

Pod created successfully.
----------------------------------------
Tau Zeta offset: 2
Upper Lower Placement point length (should be the same): [6, 6, 6]
Validating points...
Points validated.
Info: Removed 11048 vertices

Fuselage created successfully.
----------------------------------------

Tau Zeta offset: 2
Validating points...
Invalid point found:
Tau point '0.b03' is not a float.
```

The Blender console image shows the order of error checking; the last aircraft component had an invalid float Tau point and thus never reached component generation.

3.2.3 Allowable/Acceptable Error/Failure Rate

If an error does occur, it is most likely a result of either/a combination of:

a. Invalid Tau and Zeta points (i.e. ‘0.b3’ is not a float).
b. Incorrect number of Tau and Zeta points (i.e. 4 Tau points and 8 Zeta points which breaks the offset rule 3.2.2 #1 with a difference of 4).
c. Invalid property values for aircraft shape specific properties (i.e. wing: aft thickness, airfoil smoothness, base washout, tip washout, sweep, adjust wing length).
d. Invalid file location.
e. Singular matrix error (which would be caused by incorrect but valid user input points)

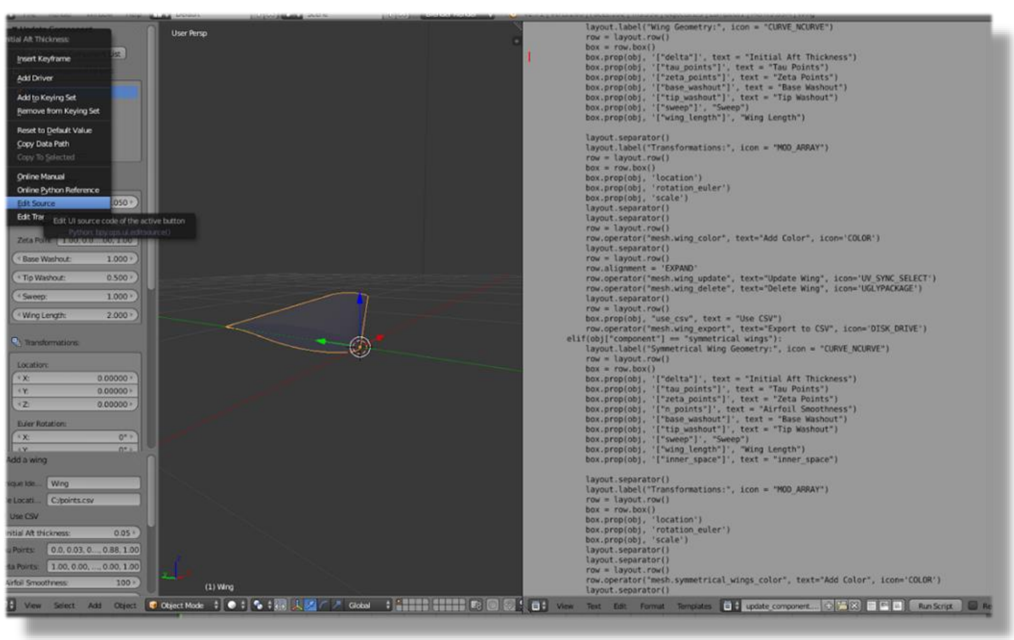
Each error would cause a failure in the generation of any aircraft shape, and thus are handled appropriately *before* the aircraft component is created.

3.3 Maintainability

Each aircraft shape has its own Python file (add_mesh_wing.py, add_mesh_pod.py, etc.), and within that are several Python modules that handle the generation of the aircraft shape. Due to the modularity of each Python script as well as internal documentation of each sub-routine, the code is readable and highly maintainable. See 3.4 Modifiability on how to modify source.

3.4 Modifiability

The Blender 3D computer graphics software is open-source, and thus, all addons and addon packages that are runnable by a user are also viewable, and modifiable by a user. Right-clicking any Blender component (i.e. “Export to CSV” button within the “Update Component” UI view) and selecting “Edit Source” will either open the source code for that particular UI Blender element or identify the specific Python script related to the particular Blender element. An example is shown below:



The aircraft addon package is designed in a way that the scripts responsible for adding the different aircraft shapes to the scene are internal and unseen by the user within Blender. However, the user is still able to navigate to the Blender addon folder (C:\Program Files\Blender Foundation\Blender\2.71\scripts\addons\aeromaster) and modify any of the source code at their will.

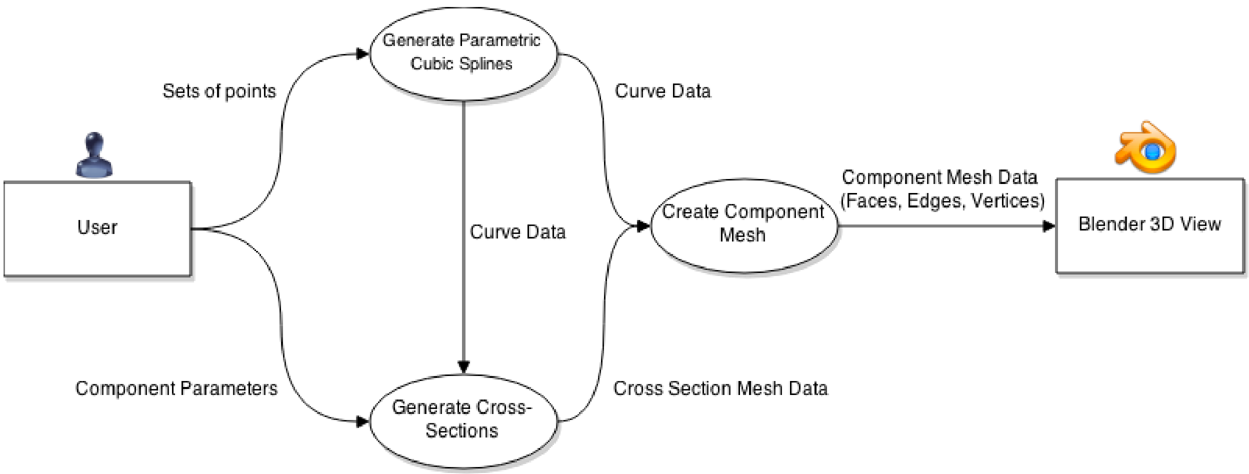
3.5 Portability

Supported Operating Systems: *Microsoft Windows, Mac OS X, GNU/Linux, FreeBSD*

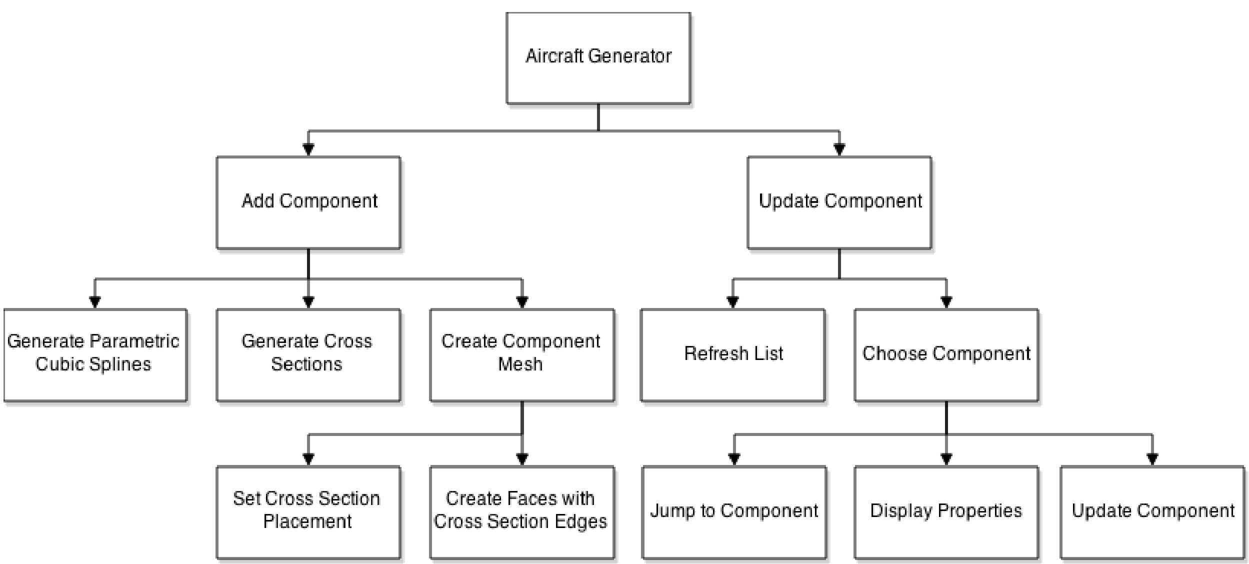
The Blender open-source application is runnable on Microsoft Windows, Mac OS X, GNU/Linux, and FreeBSD. The Aircraft Generator addon package contains Python scripts runnable within Blender, and thus the Aircraft Generator addon package is portable to all platforms that Blender can run on.

4. System Design Overview

4.1 System Data Flow Diagram



4.2 System Structure Charts



4.3 System Data Dictionary

Name	Descriptions
delta	Float value that controls the initial thickness of components (wing, symmetrical wings, pod)
tau_points	Set of points in a String format, “time”, used to parametrically characterize the curve
zeta_points	Set of points in a String format, used to parametrically characterize (x-axis) the curve
upper_points	Set of points in a String format, used to parametrically characterize (x-axis) the upper half of the fuselage component
lower_points	Set of points in a String format, used to parametrically characterize (x-axis) the lower half of the fuselage component
placement_points	Set of points in a String format, used to parametrically characterize the positions of a component’s cross sections
base_washout	Float value used to dictate the size of the wing’s washout at the base of the wing
tip_washout	Float value used to dictate the size of the wing’s washout at the tip of the wing
sweep	Float value that controls the angle of the wing from base to tip
wing_length	A float value used to dictate the length of the wing
inner_space	A float value used to dictate the amount of space in between two symmetrical wings
n_points	Integer value that controls the smoothness of the curvature on the Y-Z axis
smoothness	Integer value that controls the smoothness of the curvature on the X-Z axis
upper	A float value used to control the upper half of the fuselage
lower	A float value used to control the lower half of the fuselage
width	A float value used to control the width of the fuselage
location	A float vector that is native to a Blender object, controls the object’s location
rotation	A float vector that is native to a Blender object, controls the object’s rotation
scale	A float vector that is native to a Blender object, controls the object’s scale
colorwheel	A float vector that describes RGB values, used to set the color of an aircraft component

4.4 System Internal Data Structure Preview

Within Blender, objects are generated using faces, vertices, and (optional) edge data. Additionally, each aircraft component object is attached a number of parameters that describe how to define its geometries. Such parameters include washout, sweep, wing length,

4.5 Description of System Operation

4.5.1 Adding a component

Sets of input points are supplemented by the user. A parametric cubic spline is then generated around these each of these sets of points. The system then uses the parametric cubic spline data to place cross sections of the component along its length and scale each cross section appropriately. Once this step is complete, the system can bridge the edges and vertices together, creating the faces of the new mesh object.

4.5.2 Importing / Exporting CSV

The system uses a CSV object to read from or write to a CSV file. For reading, the systems reads the columns (amount of columns depending on the component), taking in the first and last values as end constraints and the middle values as points. For writing, the systems formats the output so that each set of points goes into the appropriate column (amount of columns depending on the component).

4.5.3 Editing a component

A number of parameters can be adjusted for each component. After setting the desired parameters, the user selects the update button. The system then takes these parameters and redraws the model (deletes the old model and draws the new model) onto the 3D view.

4.5.4 Deleting a component

After selecting a component from the component list, the system sets that object to the “active object.” Selecting the delete button deletes this selected active object.

4.6 Equipment Configuration

The Aircraft Generator addon runs directly within Blender, which is runnable on a compatible computer. After installing the addon from a zip file and activating it through the check box, the user can access all of the functionalities of the Aircraft Generator.

4.7 Implementation Languages

Python is the sole language utilized in the development of this application. It is the native programming language utilized within Blender. The use of Python within Blender allows for the scripting of Blender functionalities, the creation of new operators to be utilized within Blender, and the manipulation of the Blender GUI.

4.8 Required Support Software

Name	Relation
Numpy	This third party Python library was used for mathematical computations, arrays, and matrices, specifically solving systems of equations with matrices
CSV	This Python library was used to interact with CSV files, such as importing and exporting data formats for spreadsheets
Add Mesh - Extra Objects	This addon was used to create mathematical equations describing the cross sections of certain components

5. System Data Structure Specifications

5.1 Other User Input Specification

5.1.1 Identification of Input Data

These aircraft components have the following input data:

Wing: Unique Identifier, File Location, Use CSV toggle, Initial Aft Thickness, Tau Points, Zeta Points, Airfoil Smoothness, Base Washout, Tip Washout, Sweep, Wing Length, Location, Rotation, Scale, Color

Symmetrical Wings: Unique Identifier, File Location, Use CSV toggle, Initial Aft Thickness, Tau Points, Zeta Points, Airfoil Smoothness, Base Washout, Tip Washout, Sweep, Wing Length, Inner Space, Location, Rotation, Scale, Color

Pod: Unique Identifier, File Location, Use CSV toggle, Initial Aft Thickness, Tau Points, Zeta Points, Placement Points, Y-Z Smoothness, X-Z Smoothness, Location, Rotation, Scale, Color

Fuselage: Unique Identifier, File Location, Tau Points, Zeta Points, Upper Points, Lower Points, Placement Points, Y-Z Smoothness, X-Z Smoothness, Location, Rotation, Scale, Upper Height, Lower Height, Width, Color.

5.1.2 Source of Input Data

Default data points are included in the code but the user may enter in their own.

5.1.3 Input Medium

There are two methods for users to input data:

1. Typing in values in the *"Add Aircraft Component"* UI in *Blender*
2. Importing data points from CSV

5.1.4 Data Format/Syntax

Data Name	Format
Unique Identifier	String value
File Location	String value for file location
Use CSV	Boolean value
Initial Aft Thickness	Float value
Tau Points	String of numbers
Zeta Points	String of numbers
Airfoil Smoothness	Integer value
Base Washout	Float value
Tip Washout	Float value
Sweep	Float value
Wing Length	Float value
Location	Float vector
Rotation	Float vector
Scale	Float vector
Color	Integer vector
Inner Space	Float value
Upper Points	String of numbers
Lower Points	String of numbers
Placement Points	String of numbers
X-Z Smoothness	Integer value
Y-Z Smoothness	Integer value
Upper Height	Float value
Lower Height	Float value
Width	Float value

5.1.5 Legal Value Specification

All float values will accept any positive values, however, some values may result in components being generated with strange shapes. Additionally, some float values have no legal negative values, so a dialog box is prompted upon inputting a negative value. Tau and Zeta string values must be properly separated by a comma when input through the UI else an error will occur. When importing CSV, if non float values are imported the object will not be generated. Integer values for Smoothness will accept any positive value however the higher the value the longer time required to generate the component.

5.1.6 Examples

Data Name	Format
Unique Identifier	Wing
File Location	C://Users/Name/Desktop/file.csv
Use CSV	True
Initial Aft Thickness	0.05
Tau Points	0.0, 0.03, 0.19, 0.50, 0.88, 1.00
Zeta Points	1.00, 0.00, 0.0007, -0.049, 0.00, 0.0488, 0.00, 1.00
Airfoil Smoothness	20
Base Washout	1.00
Tip Washout	0.5
Sweep	1.00
Wing Length	2.00
Location	0.00, 0.00, 0.00
Rotation	0.00, 0.00, 0.00
Scale	1.00, 1.00, 1.000
Color	0, 0, 0
Inner Space	1.00
Upper Points	1.00, 0.0, 0.2, 0.19, 0.16, -0.6
Lower Points	1.00, 0.0, 0.0, 0.11, 0.14, 0.4
Placement Points	0.0, 0.0, 0.05, 0.15, 0.15, 1.0
X-Z Smoothness	20
Y-Z Smoothness	20
Upper Height	1.00
Lower Height	1.00
Width	1.00

5.2 Other User Output Specification

5.2.1 Identification of Output Data

These aircraft components have the following output data:

Wing: A generated single wing in the Blender 3D viewer according to specifications set by user inputs.

Symmetrical Wings: A set of two generated symmetrical wings in the Blender 3D viewer according to specifications set by user inputs.

Pod: A generated pod in the Blender 3D viewer according to specifications set by user inputs.

Fuselage: A generated fuselage in the Blender 3D viewer according to specifications set by user inputs

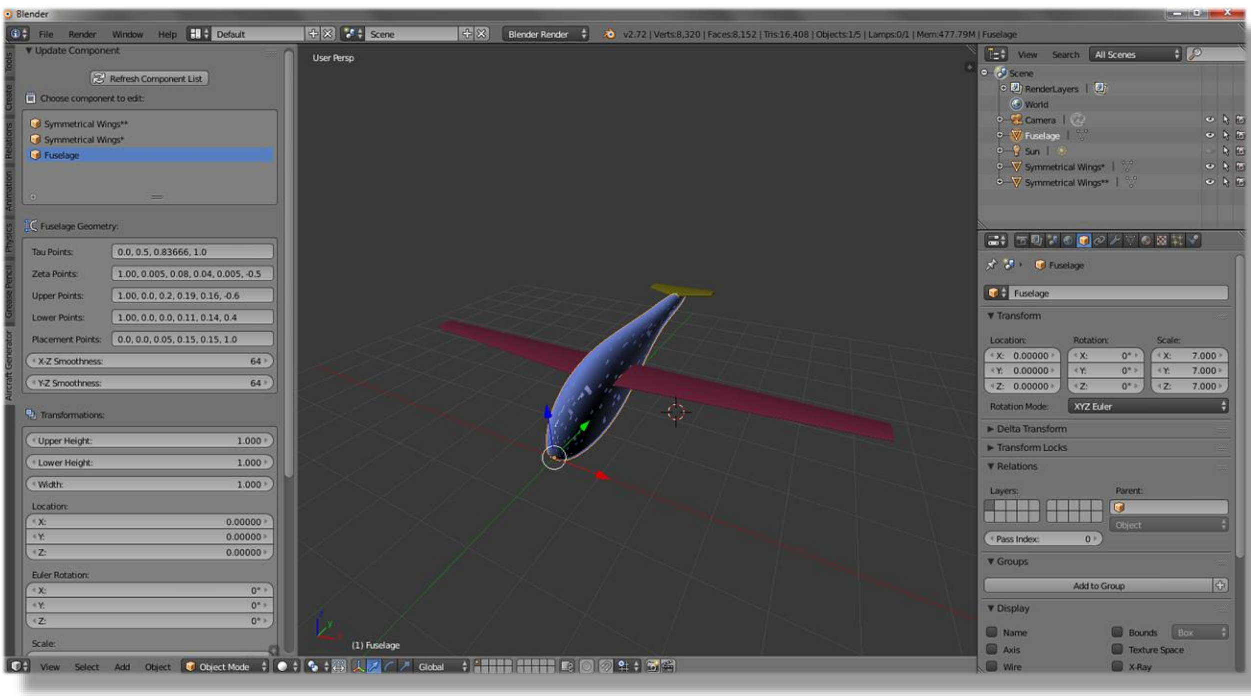
5.2.2 Destination of Output Data
The 3D model data is sent to the Blender Python module (bpy) and displayed in the Blender 3D View.

5.2.3 Output Medium
Aircraft components are output as Blender "meshes." In addition to this mesh data, which contains information on the edges, faces, and vertices, each component also contains values that control its geometries.

5.2.4 Output Format/Syntax
User may save generated components as .blend files to reopen in Blender.

5.2.5 Output Interpretation
Aircraft models may be manipulated in Blender 3D viewer, saved as .blend files, exported as a .stl format for use in 3D printers, or rendered out as an image file.

5.2.6 Example



Output of 3D aircraft model displayed in the 3D View

5.3 System Internal Data Structure Specification

5.3.1 Identification of Data Structures
The data structures in this system consist of the object, which contains data for edges, vertices, and faces, as well as parameters that help define the shape of an aircraft component. These parameters are taken in at the first moment a component is created and can be edited later on. These parameters can be seen in section 4.3 and 5.1.4.

5.3.2 Modules Accessing Structures

The modules accessing the data structures are the following components: Wing, Symmetrical Wing, Pod, and Fuselage. They use functions to access data structures, which are listed below according to their purpose:

Create: add_wing(), add_wings(), add_pod(), add_fuselage(), create_mesh_object(), makeMaterial()

Update: update_objects(), update_obj_search_index(), deselect_all_objects()

Use: setMaterial(), getTauPoints(), getZetaPoints(),

5.3.3 Logical Structure of Data

The data structures are formatted according to the attributes of the properties. They are accessed by the functions listed in section 5.4.2. The following are some examples of data structures listed with their corresponding attributes:

- Unique Identifier: String
- Initial Aft Thickness: float
- Tau Points: six float values
- Location: three float values for X, Y, and Z

6. Module Design Specifications

6.1 Module Functional Specification

6.1.1 Functions Performed

Function	Result
jump_to_object ()	Selects the object and focus the view on the object selected
update_objects()	Gets called when a refresh button or the list was clicked and displays objects as text
makeMaterial()	Creates material for respective object based on color, diffuse, specular, and alpha (Wing, Symmetrical Wings, Pod, Fuselage)
setMaterial()	Sets material to object from makeMaterial()
getTauPoints()	Gets tau points from CSV file to create respective object (Wing, Symmetrical Wings, Pod, Fuselage)
getZetaPoints()	Gets zeta points from CSV file to create respective object (Wing, Symmetrical Wings, Pod, Fuselage)

validateUserPoints()	Validates user point from CSV file or user-passed points to make sure points are valid (Wing, Symmetrical Wings, Pod)
useCSV()	Uses CSV to access a CSV sheet for tau and zeta points for respective object (Wing, Symmetrical Wings, Pod)
add_wing()	Solves and creates a wing based on delta, chi_eq, tau_points, zeta_points, washout, washout_displacement, wing_length, location, rotation, scale, file_location, isUpdate, colorwheel
add_wings()	Solves and creates symmetrical wings based on delta, chi_eq, tau_points, zeta_points, washout, washout_displacement, wing_length, location, rotation, scale, file_location, isUpdate, colorwheel
add_pod()	Solves and creates a pod based on delta, chi_eq, tau_points, zeta_points, smoothness, location, rotation, scale, file_location, isUpdate, colorwheel
add_fuselage()	Solves and creates a fuselage based on tau_points, zeta_points, upper_points, lower_points, placement_points, left_end_constraint, right_end_constraint, upper_left_end_constraint, upper_right_end_constraint, lower_left_end_constraint, lower_right_end_constraint, placement_left_end_constraint, placement_right_end_constraint, smoothness, location, rotation, scale, file_location, n_points, upper, lower, width, colorwheel
create_mesh_object()	Creates a new mesh object based on vertices, edges, faces
createFaces()	Returns a list of new faces
update_obj_search_index (update component)	Searches for index number for object created
deslect_all_objects (update component)	Deselects all objects
draw_add_mesh()	Calls objects and creates button text for respective object such as “Add Wing” (Wing, Symmetrical Wings, Pod, Fuselage)

6.1.2 Module Interface Specifications

The modules that contain input and output arguments can print text to the Python console window. For a module to output an object to the 3D View, it must utilize appropriate Blender data structures, such as a mesh. The modules interface with a CSV file that contains the Tau and Zeta points required to create the respective object. Global variables used by the modules are default values for Tau and Zeta, the conversion value for radians to degrees, and mathematical values such as PI.

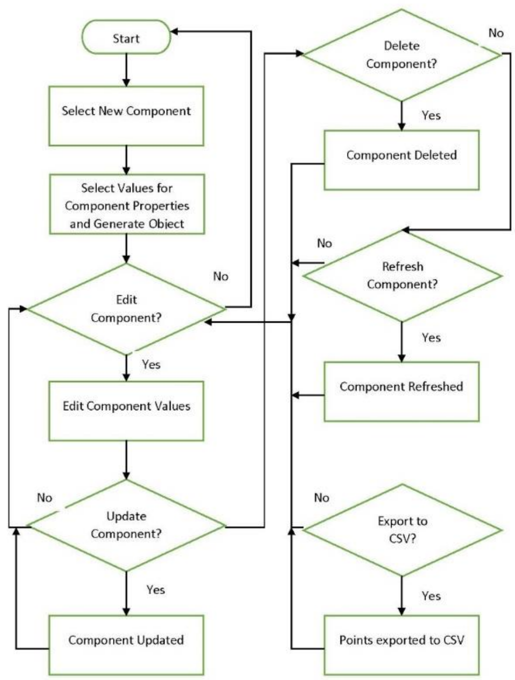
6.1.3 Module Limitations and Restrictions

The modules are limited by the specifications for each Blender data structure. For modules that utilize new Blender data structures, the module may be restricted by the lack of functionality designed for the particular data structure.

6.2 Module Operational Specification

6.2.1 Algorithm Specification

The following flowchart applies to the Wing, Symmetrical Wings, Pod, and Fuselage modules for the operation of the GUI:



For the generation of components, the following algorithm is used:

1. A parametric cubic spline is generated for a scaling and positional curve
2. Cross sections of the object are generated
3. The parametric spline for scaling and position is applied to the cross sections
4. The cross sections are bridged together to create new faces and ultimately, a new mesh object.

For the generation of parametric cubic splines, the following algorithm is used:

1. A set of points for “Tau” and “Zeta” are input.
2. A coefficient matrix is influenced by the input values
3. The system solves for the 2^{nd} derivatives to maintain continuity
4. The system solves for the 1^{st} derivatives to maintain continuity
5. The system interpolates this data to create a new curvature

6.2.3 Description of Module Operation

The following description applies to the Wing, Symmetrical Wings, Pod, and Fuselage modules:

From the Add Aircraft Component panel, the user can create a new component. When a user selects a new component, a dialog box opens with editable fields to input values for the component’s properties. Once the user clicks on the ‘OK’ button, the component is created in the 3D View. The user can now select the newly created component to edit its values. Once selected, the component’s properties are listed in the panel. The user can now edit values in property fields and click the ‘Update’ button to refresh the component. The user can also delete the component by clicking on the ‘Delete’ button in this panel. If the user would like to export the points from the component to a CSV file, the user can click on the ‘Export to CSV’ button.

7. System Verification

7.1 Items/Functions to be Tested

Wing: Generation, all properties, updating object, deleting object, importing from CSV, exporting to CSV

Symmetrical Wings: Generation, all properties, updating object, deleting object, importing from CSV, exporting to CSV

Pod: Generation, all properties, updating object, deleting object, importing from CSV, exporting to CSV

Fuselage: Generation, all properties, updating object, deleting object, importing from CSV, exporting to CSV

7.2 Description of Test Cases

Test Case Number	1
Test Item	Update Component: Refresh Component List Button
Pre-conditions	Have at least one component added in 3D view
Post-conditions	Refreshes the component list if a component is deleted within the 3D view
Input Specifications	User clicks Refresh Component List button
Expected Output Specifications	● Component list should update with the removed component name
Pass/Fail Criteria	● Pass: Component list updates and removes the deleted component name ● Fail: Component list does not remove the deleted component name
Assumptions and Constraints	● Assumption: There is at least one component in 3D view
Dependencies	2. Activating Add Mesh: Aircraft Components

Test Case Number	2
Test Item	Activating Add Mesh: Aircraft Components
Pre-conditions	Aircraft Components packaged folder must be downloaded and saved on the user's computer
Post-conditions	Aircraft Components addon is activated
Input Specifications	User clicks file > user preferences > addons > install from file > user selects the zipped folder > user check marks the addon
Expected Output Specifications	● Addon appears in the UI: user should see "Add Aircraft Component" and "Update Component" in the left module
Pass/Fail Criteria	● Pass: Addon activates and appears in the UI ● Fail: Addon does not appear in the UI
Assumptions and Constraints	● Assumption: User has the Aircraft Components folder downloaded and saved
Dependencies	N/A

Test Case Number	3
Test Item	Add Wing UI button
Pre-conditions	Aircraft Components addon is installed
Post-conditions	Add Wing dialog box pops up
Input Specifications	● User clicks Wing under "Add Aircraft Component"
Expected Output Specifications	● Add Wing dialog box pops up with editable properties
Pass/Fail Criteria	● Pass: Add Wing dialog box pops up ● Fail: Add Wing dialog box does not pop up
Assumptions and Constraints	● Assumption: Aircraft Components addon is installed
Dependencies	2. Activating Add Mesh: Aircraft Components

Test Case Number	4
Test Item	Add Symmetrical Wings UI button
Pre-conditions	Aircraft Components addon is installed
Post-conditions	Add Symmetrical Wings dialog box pops up
Input Specifications	● User clicks Symmetrical Wings under "Add Aircraft Component"
Expected Output Specifications	● Add Symmetrical Wings dialog box pops up with editable properties
Pass/Fail Criteria	● Pass: Add Symmetrical Wings dialog box pops up ● Fail: Add Symmetrical Wings dialog box does not pop up
Assumptions and Constraints	● Assumption: Aircraft Components addon is installed
Dependencies	2. Activating Add Mesh: Aircraft Components

Test Case Number	5
Test Item	Add Fuselage UI button
Pre-conditions	Aircraft Components addon is installed
Post-conditions	Add Fuselage dialog box pops up
Input Specifications	• User clicks Fuselage under "Add Aircraft Component"
Expected Output Specifications	• Add Fuselage dialog box pops up with editable properties
Pass/Fail Criteria	• Pass: Add Fuselage dialog box pops up • Fail: Add Fuselage dialog box does not pop up
Assumptions and Constraints	• Assumption: Aircraft Components addon is installed
Dependencies	2. Activating Add Mesh: Aircraft Components

Test Case Number	6
Test Item	Add Pod UI Button
Pre-conditions	Aircraft Components addon is installed
Post-conditions	Add Pod dialog box pops up
Input Specifications	• User clicks Pod under "Add Aircraft Component"
Expected Output Specifications	• Add Pod dialog box pops up with editable properties
Pass/Fail Criteria	• Pass: Add Pod dialog box pops up • Fail: Add Pod dialog box does not pop up
Assumptions and Constraints	• Assumption: Aircraft Components addon is installed
Dependencies	2. Activating Add Mesh: Aircraft Components

Test Case Number	7
Test Item	Wing Dialog Box: Unique Identifier
Pre-conditions	Wing dialog box is opened
Post-conditions	User inputted unique identifier is assigned or default name is assigned
Input Specifications	● User inputs a unique identifier
Expected Output Specifications	● User inputted unique identifier (i.e. name, e.g. "my wing") is assigned under the components list. Otherwise, the defaulted "Wing" name is assigned, followed by "Wing.001", "Wing.002", etc.
Pass/Fail Criteria	● Pass: User inputted unique identifier is assigned and defaulted names are assigned in sequential order ● Fail: User inputted unique identifier is not assigned and defaulted names are assigned out of order
Assumptions and Constraints	● Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI button

Test Case Number	8
Test Item	Wing Dialog Box: File Location
Pre-conditions	Wing dialog box is opened; a CSV file is saved on a specified file location, e.g. C drive
Post-conditions	File is imported from specified location
Input Specifications	● User types a file location into the input box ● User check marks "Use CSV"
Expected Output Specifications	● File is imported from the specified directory ● Tau and zeta points from CSV file should be displayed ● Wing should be generated according to points
Pass/Fail Criteria	● Pass: File is imported correctly with the right points and wing is generated with those points ● Fail: Points are not imported and wing is not generated with given points
Assumptions and Constraints	● Assumption: A CSV file has been saved; Wing dialog box is already opened
Dependencies	3. Add Wing UI button

Test Case Number	9
Test Item	Wing Dialog Box: Initial Aft Thickness
Pre-conditions	Wing dialog box is opened
Post-conditions	Initial Aft Thickness, the initial thickness of the curve is set
Input Specifications	● User inputs an initial aft thickness value
Expected Output Specifications	● Wing is generated according to the initial aft thickness value or default value of 0.05
Pass/Fail Criteria	● Pass: Initial aft thickness is set and changes model generation accordingly ● Fail: Initial aft thickness is not set and does not change model generation accordingly
Assumptions and Constraints	● Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI button

Test Case Number	10
Test Item	Wing dialog box: Tau Points
Pre-conditions	Wing dialog box is opened
Post-conditions	Wing is generated with the user inputted tau points (independent time variable) or the default points
Input Specifications	● User inputs tau points
Expected Output Specifications	● Wing model is generated according to the inputted tau points or default values of 0.0, 0.03, 0.19, 0.50, 0.88, 1.00
Pass/Fail Criteria	● Pass: Wing is generated according to the user inputted tau points or the default points ● Fail: Wing is not generated correctly with the inputted tau points or the default points
Assumptions and Constraints	● Assumption: Wing dialog box is already opened ● Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	3. Add Wing UI button

Test Case Number	11
Test Item	Wing Dialog Box: Zeta points
Pre-conditions	Wing dialog box is opened
Post-conditions	Wing is generated with the user inputted zeta points (y variables) or the default points
Input Specifications	• User inputs zeta points
Expected Output Specifications	• Wing model is generated according to the inputted zeta points or default values of 0.00, 0.0007, -0.049, 0.00, 0.0488, 0.00
Pass/Fail Criteria	• Pass: Wing is generated according to the user inputted zeta points or the default points • Fail: Wing is not generated correctly with the inputted zeta points or the default points
Assumptions and Constraints	• Assumption: Wing dialog box is already opened • Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	3. Add Wing UI button

Test Case Number	12
Test Item	Wing Dialog Box: Airfoil Smoothness
Pre-conditions	Wing dialog box is opened
Post-conditions	Airfoil smoothness, the number of faces, is set and changes model generation accordingly
Input Specifications	• User inputs airfoil smoothness value
Expected Output Specifications	• Wing is generated according to the user inputted smoothness value or the default value of 20
Pass/Fail Criteria	• Pass: Wing is generated according to the airfoil smoothness • Fail: Wing is not generated correctly with the airfoil smoothness
Assumptions and Constraints	• Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI Button

Test Case Number	13
Test Item	Wing Dialog Box: Base Washout
Pre-conditions	Wing dialog box is opened
Post-conditions	Base washout value, the size of the airfoil across the wing at the base,, is set and changes model generation accordingly
Input Specifications	• User inputs base washout value
Expected Output Specifications	• Wing is generated according to the user inputted base washout value or the default value of 1.00
Pass/Fail Criteria	• Pass: Wing is generated according to the base washout value • Fail: Wing is not generated correctly with the base washout value
Assumptions and Constraints	• Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI Button

Test Case Number	14
Test Item	Wing Dialog Box: Tip Washout
Pre-conditions	Wing dialog box is opened
Post-conditions	Tip washout value, the size of the airfoil across the wing at the tip, is set and changes model generation accordingly
Input Specifications	• User inputs tip washout value
Expected Output Specifications	• Wing is generated according to the user inputted tip washout value or the default value of 1.00
Pass/Fail Criteria	• Pass: Wing is generated according to the tip washout value • Fail: Wing is not generated correctly with the tip washout value
Assumptions and Constraints	• Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI Button

Test Case Number	15
Test Item	Wing Dialog Box: Sweep
Pre-conditions	Wing dialog box is opened
Post-conditions	The sweep angle is set and changes model generation accordingly
Input Specifications	• User inputs sweep value
Expected Output Specifications	• Wing is generated with the user inputted sweep value or the default value of 1.00
Pass/Fail Criteria	• Pass: Wing is generated according to the sweep value • Fail: Wing is not generated correctly with the sweep value
Assumptions and Constraints	• Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI Button

Test Case Number	16
Test Item	Wing Dialog Box: Adjust Wing Length
Pre-conditions	Wing dialog box is opened
Post-conditions	Wing is created with the user specified wing length value
Input Specifications	• User inputs value for wing length
Expected Output Specifications	• Wing is generated according to the user inputted wing length or default value of 2.00
Pass/Fail Criteria	• Pass: Wing is generated with the correct wing length • Fail: Wing is generated with an incorrect wing length
Assumptions and Constraints	• Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI Button

Test Case Number	17
Test Item	Wing Dialog Box: Location
Pre-conditions	Wing dialog box is opened
Post-conditions	Wing is generated at the correct location on the axis
Input Specifications	● User inputs an x, y, z coordinate location
Expected Output Specifications	● Wing is generated at the user inputted x, y, & z coordinates or at the default origin
Pass/Fail Criteria	● Pass: Wing is generated at the user inputted x, y, & z coordinates or at the default origin ● Fail: Wing is not generated at the user inputted x, y, & z coordinates or is not generated at the default origin
Assumptions and Constraints	● Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI Button

Test Case Number	18
Test Item	Wing Dialog Box: Rotation
Pre-conditions	Wing dialog box is opened
Post-conditions	Wing is generated with the correct x, y, & z rotational values
Input Specifications	● User inputs x, y, & z rotational values in degrees
Expected Output Specifications	● Wing is generated according to the user inputted x, y, & z rotational values or the default values of x: 0°, y: 0°, z: 0°
Pass/Fail Criteria	● Pass: Wing is generated correctly with the user inputted x, y, & z rotational values or the default values ● Fail: Wing is not generated correctly with the user inputted x, y, & z rotational values or the wing is not generated with the correct default rotational values
Assumptions and Constraints	● Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI Button

Test Case Number	19
Test Item	Wing Dialog Box: Scale
Pre-conditions	Wing dialog box is opened
Post-conditions	Wing is created with the user specified x, y, & z values or the default values of 1.000
Input Specifications	• User inputs x, y, & z scale values
Expected Output Specifications	• Wing is generated according to the correct x, y, and z scale values
Pass/Fail Criteria	• Pass: Wing is generated according to the user specified values or default scale values • Fail: Wing is not generated according to the user specified values or default scale values
Assumptions and Constraints	• Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI Button

Test Case Number	20
Test Item	Wing Dialog Box: Color
Pre-conditions	Wing dialog box is opened
Post-conditions	Wing is generated with the chosen color
Input Specifications	• User clicks on the colorwheel grey color • User can select a color on the wheel, enter a RGB value, enter a HEX value, and/or enter HSV values
Expected Output Specifications	• Wing is generated according to the chosen color or the default grey color
Pass/Fail Criteria	• Pass: Wing is generated with the correct color • Fail: Wing is generated with an incorrect color
Assumptions and Constraints	• Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI Button

Test Case Number	21
Test Item	Wing Dialog Box: OK
Pre-conditions	Wing dialog box is opened
Post-conditions	Wing is generated and displayed
Input Specifications	• User clicks OK
Expected Output Specifications	• Wing is generated into 3D View with all the correct properties
Pass/Fail Criteria	• Pass: Wing is generated and displayed on the screen • Fail: Wing is not generated and does not display on the screen
Assumptions and Constraints	• Assumption: Wing dialog box is already opened
Dependencies	3. Add Wing UI Button

Test Case Number	22
Test Item	Symmetrical Wings Dialog Box: Unique Identifier
Pre-conditions	Symmetrical Wings dialog box is opened
Post-conditions	User inputted unique identifier is assigned or default name is assigned
Input Specifications	• User inputs a unique identifier
Expected Output Specifications	• User inputted unique identifier (i.e. name, e.g. "my symmetrical wings") is assigned under the components list. Otherwise, the defaulted "Symmetrical Wings" name is assigned, followed by "Symmetrical Wings.001", "Symmetrical Wings.002", etc.
Pass/Fail Criteria	• Pass: User inputted unique identifier is assigned and defaulted names are assigned in sequential order • Fail: User inputted unique identifier is not assigned and defaulted names are assigned out of order
Assumptions and Constraints	• Assumption: Symmetrical Wings dialog box is already opened
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	23
Test Item	Symmetrical Wings Dialog Box: File Location
Pre-conditions	Symmetrical Wings dialog box is opened; a CSV file is saved on a specified file location, e.g. C drive
Post-conditions	File is imported from specified location
Input Specifications	• User types a file location into the input box • User check marks "use CSV"
Expected Output Specifications	• File is imported from the specified directory • Tau and zeta points from CSV file should be displayed • Symmetrical wings should be generated according to points
Pass/Fail Criteria	• Pass: File is imported correctly with the right points and symmetrical wings are generated with those points • Fail: Points are not imported and symmetrical wings are not generated with given points
Assumptions and Constraints	• Assumption: A CSV file has been saved; Symmetrical Wings dialog box is already opened
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	24
Test Item	Symmetrical Wings Dialog Box: Initial Aft Thickness
Pre-conditions	Symmetrical Wings dialog box is opened
Post-conditions	Initial aft thickness, the initial thickness of the curve is set
Input Specifications	• User inputs an initial aft thickness value
Expected Output Specifications	• Symmetrical Wings are generated according to the initial aft thickness value or default value of 0.05
Pass/Fail Criteria	• Pass: Initial aft thickness is set and changes model generation accordingly • Fail: Initial aft thickness is not set and does not change model generation accordingly
Assumptions and Constraints	• Assumption: Symmetrical wings dialog box is already opened
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	25
Test Item	Symmetrical Wings Dialog Box: Tau points
Pre-conditions	Symmetrical wings dialog box is opened
Post-conditions	Symmetrical wings are generated with the user inputted tau points (independent time variable) or the default points
Input Specifications	● User inputs tau points
Expected Output Specifications	● Symmetrical wings model is generated according to the inputted tau points or default values of 0.0, 0.03, 0.19, 0.50, 0.88, 1.00
Pass/Fail Criteria	● Pass: Symmetrical wings are generated according to the user inputted tau points or the default points ● Fail: Symmetrical wings are not generated correctly with the inputted tau points or the default points
Assumptions and Constraints	● Assumption: Symmetrical wings dialog box is already opened ● Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	26
Test Item	Symmetrical Wings Dialog Box: Zeta points
Pre-conditions	Symmetrical wings dialog box is opened
Post-conditions	Symmetrical wings are generated with the user inputted zeta points (y variables) or the default points
Input Specifications	● User inputs zeta points
Expected Output Specifications	● Symmetrical wings model is generated according to the inputted zeta points or default values of 0.00, 0.0007, -0.049, 0.00, 0.0488, 0.00
Pass/Fail Criteria	● Pass: Symmetrical wings are generated according to the user inputted zeta points or the default points ● Fail: Symmetrical wings are not generated correctly with the inputted zeta points or the default points
Assumptions and Constraints	● Assumption: Symmetrical wings dialog box is already opened ● Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	27
Test Item	Symmetrical Wings Dialog Box: Airfoil Smoothness
Pre-conditions	Symmetrical wings dialog box is opened
Post-conditions	Airfoil smoothness, the number of faces, is set and changes model generation accordingly
Input Specifications	• User inputs airfoil smoothness value
Expected Output Specifications	• Symmetrical wings are generated according to the user inputted smoothness value or the default value of 20
Pass/Fail Criteria	• Pass: Symmetrical wings are generated according to the airfoil smoothness • Fail: Symmetrical wings are not generated correctly with the airfoil smoothness
Assumptions and Constraints	• Assumption: Symmetrical Wings dialog box is already opened
Dependencies	4. Add Symmetrical Wings UI Button

Test Case Number	28
Test Item	Symmetrical Wings Dialog Box: Base Washout
Pre-conditions	Symmetrical wings dialog box is opened
Post-conditions	Base washout value, the size of the airfoil across the wing at the base, is set and changes model generation accordingly
Input Specifications	• User inputs base washout value
Expected Output Specifications	• Symmetrical wings are generated according to the user inputted base washout value or the default value of 1.00
Pass/Fail Criteria	• Pass: Symmetrical wings are generated according to the base washout value • Fail: Symmetrical wings are not generated correctly with the base washout value
Assumptions and Constraints	• Assumption: Symmetrical wings dialog box is already opened
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	29
Test Item	Symmetrical Wings Dialog Box: Tip Washout
Pre-conditions	Symmetrical wings dialog box is opened
Post-conditions	Tip washout value, the size of the airfoil across the wing at the tip, is set and changes model generation accordingly
Input Specifications	● User inputs tip washout value
Expected Output Specifications	● Symmetrical wings are generated according to the user inputted tip washout value or the default value of 0.50
Pass/Fail Criteria	● Pass: Symmetrical wings are generated according to the tip washout value ● Fail: Symmetrical wings are not generated correctly with the tip washout value
Assumptions and Constraints	● Assumption: Symmetrical wings dialog box is already opened
Dependencies	5. Add Symmetrical Wings UI button

Test Case Number	30
Test Item	Symmetrical Wings Dialog Box: Sweep
Pre-conditions	Symmetrical wings dialog box is opened
Post-conditions	The sweep angle is set and changes model generation accordingly
Input Specifications	● User inputs sweep angle value
Expected Output Specifications	● Symmetrical wings are generated with the user inputted sweep angle value or the default value of 0.50
Pass/Fail Criteria	● Pass: Symmetrical wings are generated according to the sweep value ● Fail: Symmetrical wings are not generated correctly with the sweep value
Assumptions and Constraints	● Assumption: Symmetrical wings dialog box is already opened
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	31
Test Item	Symmetrical Wings Dialog Box: Adjust Wing Length
Pre-conditions	Symmetrical wings dialog box is opened
Post-conditions	Symmetrical wings are created with the user specified wing length value
Input Specifications	• User inputs value for wing length
Expected Output Specifications	• Symmetrical wings are generated according to the user inputted wing length or default value of 3.20
Pass/Fail Criteria	• Pass: Symmetrical wings are generated with the correct wing length • Fail: Symmetrical wings are generated with an incorrect wing length
Assumptions and Constraints	• Assumption: Symmetrical wings dialog box is already opened
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	32
Test Item	Symmetrical Wings Dialog Box: Inner Space
Pre-conditions	Symmetrical wings dialog box is opened
Post-conditions	Symmetrical wings are created with the user specified inner space value (the space between the two symmetrical wings)
Input Specifications	• User inputs value for inner space
Expected Output Specifications	• Symmetrical wings are generated according to the user inputted inner space or default value of 1.00
Pass/Fail Criteria	• Pass: Symmetrical wings are generated with the correct inner space • Fail: Symmetrical wings are generated with incorrect inner space
Assumptions and Constraints	• Assumption: Symmetrical wings dialog box is already opened
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	33
Test Item	Symmetrical Wings Dialog Box: Location
Pre-conditions	Add Symmetrical Wings dialog box opened
Post-conditions	Symmetrical wings are generated at the correct location
Input Specifications	● Location is entered into input box ● Location changes model generation accordingly
Expected Output Specifications	● Symmetrical wings are generated at the user inputted x, y, & z coordinates or at the default origin
Pass/Fail Criteria	● Pass: Symmetrical wings are generated at the user inputted x, y, & z coordinates or at the default origin ● Fail: Symmetrical wings are not generated at the user inputted x, y, & z coordinates or is not generated at the default origin
Assumptions and Constraints	● Symmetrical wings dialog box already open
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	34
Test Item	Symmetrical Wings Dialog Box: Rotation
Pre-conditions	Add Symmetrical Wings dialog box opened
Post-conditions	Symmetrical wings generated with the correct x, y, & z rotational values
Input Specifications	● Rotation is entered into input box ● Rotation changes model generation accordingly
Expected Output Specifications	● Rotation changes the model accordingly
Pass/Fail Criteria	● Pass: Symmetrical wings is generated with the correct x, y, & z rotational values or the default 0 rotational values (user's perspective should view the wings from the top - should be able to see both ends of the symmetrical wings across the x-axis) ● Fail: Symmetrical wings are not generated with the user inputted x, y, & z rotational values or the wings are not generated with the correct default 0 rotational values
Assumptions and Constraints	● Symmetrical wings dialog box already open
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	35
Test Item	Symmetrical Wings Dialog Box: Scale
Pre-conditions	Add Symmetrical Wings dialog box opened
Post-conditions	Symmetrical wings are created with the user specified x, y, & z values or the default values of 1.000
Input Specifications	● Scale is entered into input box ● Scale changes model generation accordingly
Expected Output Specifications	Symmetrical wings are created and/or adjusted according to the correct x, y, and z scale values
Pass/Fail Criteria	● Pass: Symmetrical wings are created and adjusted according to the user specified values or default scale values ● Fail: Symmetrical wings are not created nor adjusted according to the user specified values or default scale value
Assumptions and Constraints	● Symmetrical wings dialog box already open
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	36
Test Item	Symmetrical Wings Dialog Box: Color
Pre-conditions	Symmetrical wings dialog box is opened
Post-conditions	Symmetrical wings are generated with the chosen color
Input Specifications	● User clicks on the colorwheel grey color ● User can select a color on the wheel, enter a RGB value, enter a HEX value, and/or enter HSV values
Expected Output Specifications	● Symmetrical wings are generated according to the chosen color or the default grey color
Pass/Fail Criteria	● Pass: Symmetrical wings are generated with the correct color ● Fail: Symmetrical wings are generated with an incorrect color
Assumptions and Constraints	● Assumption: Symmetrical wings dialog box is already opened
Dependencies	3. Add Symmetrical Wings UI button

Test Case Number	37
Test Item	Symmetrical Wings Dialog Box: OK
Pre-conditions	Symmetrical wings dialog box is opened
Post-conditions	Symmetrical wings are generated and displayed
Input Specifications	● User clicks OK
Expected Output Specifications	● Symmetrical wings are generated into 3D View with all the correct properties
Pass/Fail Criteria	● Pass: Symmetrical wings are generated and displayed on the screen ● Fail: Symmetrical wings are not generated and do not display on the screen
Assumptions and Constraints	● Assumption: Symmetrical wings dialog box is already opened
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	38
Test Item	Pod Dialog Box: Unique Identifier
Pre-conditions	Pod dialog box is opened
Post-conditions	User inputted unique identifier is assigned or default name is assigned
Input Specifications	● User inputs a unique identifier
Expected Output Specifications	● User inputted unique identifier (i.e. name, e.g. "my pod") is assigned under the components list. Otherwise, the defaulted "Pod" name is assigned, followed by "Pod.001", "Pod.002", etc.
Pass/Fail Criteria	● Pass: User inputted unique identifier is assigned and defaulted names are assigned in sequential order ● Fail: User inputted unique identifier is not assigned and defaulted names are assigned out of order
Assumptions and Constraints	● Assumption: Pod dialog box is already opened
Dependencies	6. Add Pod UI button

Test Case Number	39
Test Item	Pod Dialog Box: File location
Pre-conditions	Pod dialog box is opened; a CSV file is saved on a specified file location, e.g. C drive
Post-conditions	File is imported from specified location
Input Specifications	● User types a file location into the input box ● User check marks "Use CSV"
Expected Output Specifications	● File is imported from the specified directory ● Tau and zeta points from CSV file should be displayed ● Pod should be generated according to points
Pass/Fail Criteria	● Pass: File is imported correctly with the right points and pod is generated with those points ● Fail: Points are not imported and pod is not generated with given points
Assumptions and Constraints	● Assumption: A CSV file has been saved; Pod dialog box is already opened
Dependencies	6. Add Pod UI button

Test Case Number	40
Test Item	Pod Dialog Box: Initial Aft Thickness
Pre-conditions	Pod dialog box is opened
Post-conditions	Initial aft thickness, the initial thickness of the curve is set
Input Specifications	● User inputs an initial thickness value
Expected Output Specifications	● Pod is generated according to the initial aft thickness value or default value of 0.05
Pass/Fail Criteria	● Pass: Initial aft thickness is set and changes model generation accordingly ● Fail: Initial aft thickness is not set and does not change model generation accordingly
Assumptions and Constraints	● Assumption: Pod dialog box is already opened
Dependencies	6. Add Pod UI button

Test Case Number	41
Test Item	Pod Dialog Box: Tau points
Pre-conditions	Pod dialog box is opened
Post-conditions	Pod is generated with the user inputted tau points (independent time variable) or the default points
Input Specifications	• User inputs tau points
Expected Output Specifications	• Pod model is generated according to the inputted tau points or default values of 0.0, 0.03, 0.19, 0.50, 0.88, 1.00
Pass/Fail Criteria	• Pass: Pod is generated according to the user inputted tau points or the default points • Fail: Pod is not generated correctly with the inputted tau points or the default points
Assumptions and Constraints	• Assumption: Pod dialog box is already opened • Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	6. Add Pod UI Button

Test Case Number	42
Test Item	Pod Dialog Box: Zeta points
Pre-conditions	Pod dialog box is opened
Post-conditions	Pod is generated with the user inputted zeta points (y variables) or the default points
Input Specifications	• User inputs zeta points
Expected Output Specifications	• Pod model is generated according to the inputted zeta points or default values of 0.00, 0.0007, -0.049, 0.00, 0.0488, 0.00
Pass/Fail Criteria	• Pass: Pod is generated according to the user inputted zeta points or the default points • Fail: Pod is not generated correctly with the inputted zeta points or the default points
Assumptions and Constraints	• Assumption: Pod dialog box is already opened • Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	6. Add Pod UI Button

Test Case Number	43
Test Item	Pod Dialog Box: Placement Points
Pre-conditions	Pod dialog box is opened
Post-conditions	Pod is generated with the correct placement points (for the cross sections)
Input Specifications	• User inputs placement points
Expected Output Specifications	• Pod model is generated according to the inputted placement points or default points of 0.0, 0.0, 0.05, 0.15, 0.15, 1.0
Pass/Fail Criteria	• Pass: Pod is generated according to the user inputted placement points or the default points • Fail: Pod is not generated correctly with the inputted placement points or the default points
Assumptions and Constraints	• Assumption: Pod dialog box is already opened
Dependencies	6. Add Pod UI Button

Test Case Number	44
Test Item	Pod Dialog Box: Y-Z Smoothness
Pre-conditions	Pod dialog box is opened
Post-conditions	Pod is generated with the set smoothness value, the smoothness on the y-z axis
Input Specifications	• User inputs smoothness value
Expected Output Specifications	• Pod is generated with the set smoothness value or the default value of 20
Pass/Fail Criteria	• Pass: Pod is generated with correct smoothness • Fail: Pod is generated with incorrect smoothness
Assumptions and Constraints	• Assumption: Pod dialog box is already opened
Dependencies	6. Add Pod UI Button

Test Case Number	45
Test Item	Pod Dialog Box: X-Z Smoothness
Pre-conditions	Pod dialog box is opened
Post-conditions	Pod is generated with the set smoothness value, the smoothness on the x-z axis
Input Specifications	● User inputs smoothness value
Expected Output Specifications	● Pod is generated with the set smoothness value or the default value of 20
Pass/Fail Criteria	● Pass: Pod is generated with correct smoothness ● Fail: Pod is generated with incorrect smoothness
Assumptions and Constraints	● Assumption: Pod dialog box is already opened
Dependencies	6. Add Pod UI Button

Test Case Number	46
Test Item	Pod Dialog Box: Location
Pre-conditions	Pod dialog box is opened
Post-conditions	Pod is generated at the correct location on the axis
Input Specifications	● User inputs an x, y, z coordinate location
Expected Output Specifications	● Pod is generated at the user inputted x, y, & z coordinates or at the default origin
Pass/Fail Criteria	● Pass: Pod is generated at the user inputted x, y, & z coordinates or at the default origin ● Fail: Pod is not generated at the user inputted x, y, & z coordinates or is not generated at the default origin
Assumptions and Constraints	● Assumption: Pod dialog box is already opened
Dependencies	6. Add Pod UI Button

Test Case Number	47
Test Item	Pod Dialog Box: Rotation
Pre-conditions	Pod dialog box is opened
Post-conditions	Pod is generated with the correct x, y, & z rotational values
Input Specifications	• User inputs x, y, & z rotational values in degrees
Expected Output Specifications	• Pod is generated according to the user inputted x, y, & z rotational values or the default values of x: 0°, y: 0°, z: 0°
Pass/Fail Criteria	• Pass: Pod is generated correctly with the user inputted x, y, & z rotational values or the default values • Fail: Pod is not generated correctly with the user inputted x, y, & z rotational values or the pod is not generated with the correct default rotational values
Assumptions and Constraints	• Assumption: Pod dialog box is already opened
Dependencies	6. Add Pod UI button

Test Case Number	48
Test Item	Pod Dialog Box: Scale
Pre-conditions	Pod dialog box is opened
Post-conditions	Pod is created with the user specified x, y, & z values or the default values of 1.000
Input Specifications	• User inputs x, y, & z scale values
Expected Output Specifications	• Pod is generated according to the correct x, y, and z scale values
Pass/Fail Criteria	• Pass: Pod is generated according to the user specified values or default scale values • Fail: Pod is not generated according to the user specified values or default scale values
Assumptions and Constraints	• Assumption: Pod dialog box is already opened
Dependencies	6. Add Pod UI Button

Test Case Number	49
Test Item	Pod Dialog Box: Color
Pre-conditions	Pod dialog box is opened
Post-conditions	Pod is generated with the chosen color
Input Specifications	● User clicks on the colorwheel grey color ● User can select a color on the wheel, enter a RGB value, enter a HEX value, and/or enter HSV values
Expected Output Specifications	● Pod is generated according to the chosen color or the default grey color
Pass/Fail Criteria	● Pass: Pod is generated with the correct color ● Fail: Pod is generated with an incorrect color
Assumptions and Constraints	● Assumption: Pod dialog box is already opened
Dependencies	6. Add Pod UI Button

Test Case Number	50
Test Item	Pod Dialog Box: OK
Pre-conditions	Pod dialog box is opened
Post-conditions	Pod is generated and displayed
Input Specifications	● User clicks OK
Expected Output Specifications	● Pod is generated into 3D View with all the correct properties
Pass/Fail Criteria	● Pass: Pod is generated and displayed on the screen ● Fail: Pod is not generated and does not display on the screen
Assumptions and Constraints	● Assumption: Pod dialog box is already opened
Dependencies	6. Add Pod UI Button

Test Case Number	51
Test Item	Fuselage Dialog Box: Unique Identifier
Pre-conditions	Fuselage dialog box is opened
Post-conditions	User inputted unique identifier is assigned or default name is assigned
Input Specifications	● User inputs a unique identifier
Expected Output Specifications	● User inputted unique identifier (i.e. name, e.g. "my fuselage") is assigned under the components list. Otherwise, the defaulted "Fuselage" name is assigned, followed by "Fuselage.001", "Fuselage.002", etc.
Pass/Fail Criteria	● Pass: User inputted unique identifier is assigned and defaulted names are assigned in sequential order ● Fail: User inputted unique identifier is not assigned and defaulted names are assigned out of order
Assumptions and Constraints	● Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI button

Test Case Number	52
Test Item	Fuselage Dialog Box: File location
Pre-conditions	Fuselage dialog box is opened; a CSV file is saved on a specified file location, e.g. C drive
Post-conditions	File is imported from specified location
Input Specifications	● User types a file location into the input box ● User check marks "Use CSV"
Expected Output Specifications	● File is imported from the specified directory ● Tau and zeta points from CSV file should be displayed ● Fuselage should be generated according to points
Pass/Fail Criteria	● Pass: File is imported correctly with the right points and fuselage is generated with those points ● Fail: Points are not imported and fuselage is not generated with given points
Assumptions and Constraints	● Assumption: A CSV file has been saved; Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI button

Test Case Number	53
Test Item	Fuselage Dialog Box: Tau Points
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated with the user inputted tau points (independent time variable) or the default points
Input Specifications	● User inputs tau points
Expected Output Specifications	● Fuselage model is generated according to the inputted tau points or default values of 0.0, 0.5, 0.83666, 1.0
Pass/Fail Criteria	● Pass: Fuselage is generated according to the user inputted tau points or the default points ● Fail: Fuselage is not generated correctly with the inputted tau points or the default points
Assumptions and Constraints	● Assumption: Fuselage dialog box is already opened ● Constraints: If adding more/removing more values than the default number of values, the tau and zeta points must have the same number of values
Dependencies	5. Add Fuselage UI Button

Test Case Number	54
Test Item	Fuselage Dialog Box: Zeta Points
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated with the user inputted zeta points (y variables) or the default points
Input Specifications	● User inputs zeta points
Expected Output Specifications	● Fuselage model is generated according to the inputted zeta points or default values of 0.005, 0.08, 0.04, 0.005
Pass/Fail Criteria	● Pass: Fuselage is generated according to the user inputted zeta points or the default points ● Fail: Fuselage is not generated correctly with the inputted zeta points or the default points
Assumptions and Constraints	● Assumption: Fuselage dialog box is already opened ● Constraints: If adding more/removing more values than the default number of values, the tau and zeta points must have the same number of values
Dependencies	5. Add Fuselage UI button

Test Case Number	55
Test Item	Fuselage Dialog Box: Upper Points
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated with the user inputted upper points (the points for the top curve) or the default points
Input Specifications	• User inputs upper points
Expected Output Specifications	• Fuselage model is generated according to the inputted upper points or default values of 0.0, 0.2, 0.19, 0.16
Pass/Fail Criteria	• Pass: Fuselage is generated according to the user inputted upper points or the default points • Fail: Fuselage is not generated correctly with the inputted upper points or the default points
Assumptions and Constraints	• Assumption: Fuselage dialog box is already opened • Constraints: Upper points, lower points, and placement points should have the same number of points
Dependencies	5. Add Fuselage UI Button

Test Case Number	56
Test Item	Fuselage Dialog Box: Lower Points
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated with the user inputted lower points (the points for the bottom curve) or the default points
Input Specifications	• User inputs lower points
Expected Output Specifications	• Fuselage model is generated according to the inputted lower points or default values of 0.0, 0.4, 0.11, 0.14
Pass/Fail Criteria	• Pass: Fuselage is generated according to the user inputted lower points or the default points • Fail: Fuselage is not generated correctly with the inputted lower points or the default points
Assumptions and Constraints	• Assumption: Fuselage dialog box is already opened • Constraints: Upper points, lower points, and placement points should have the same number of points
Dependencies	5. Add Fuselage UI Button

Test Case Number	57
Test Item	Fuselage Dialog Box: Placement Points
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated with the user inputted placement points (the placement points for the cross sections) or the default points
Input Specifications	● User inputs placement points
Expected Output Specifications	● Fuselage model is generated according to the inputted placement points or default values of 0.0, 0.05, 0.15, 0.15
Pass/Fail Criteria	● Pass: Fuselage is generated according to the user inputted placement points or the default points ● Fail: Fuselage is not generated correctly with the inputted placement points or the default points
Assumptions and Constraints	● Assumption: Fuselage dialog box is already opened ● Constraints: Upper points, lower points, and placement points should have the same number of points
Dependencies	5. Add Fuselage UI Button

Test Case Number	58
Test Item	Fuselage dialog box: X-Z Smoothness
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated with the set smoothness value, the smoothness on the x, z axis
Input Specifications	● User inputs smoothness value
Expected Output Specifications	● Fuselage is generated with the set smoothness value or the default value of 20
Pass/Fail Criteria	● Pass: Fuselage is generated with correct smoothness ● Fail: Fuselage is generated with incorrect smoothness
Assumptions and Constraints	● Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI Button

Test Case Number	59
Test Item	Fuselage Dialog Box: Y-Z Smoothness
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated with the set number of points value, the smoothness on the y, z axis
Input Specifications	• User inputs smoothness value
Expected Output Specifications	• Fuselage is generated with the set value or the default value of 20
Pass/Fail Criteria	• Pass: Fuselage is generated with the correct smoothness • Fail: Fuselage is generated with incorrect smoothness
Assumptions and Constraints	• Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI Button

Test Case Number	60
Test Item	Fuselage Dialog Box: Location
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated at the correct location on the axis
Input Specifications	• User inputs an x, y, z coordinate location
Expected Output Specifications	• Fuselage is generated at the user inputted x, y, & z coordinates or at the default origin
Pass/Fail Criteria	• Pass: Fuselage is generated at the user inputted x, y, & z coordinates or at the default origin • Fail: Fuselage is not generated at the user inputted x, y, & z coordinates or is not generated at the default origin
Assumptions and Constraints	• Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI button

Test Case Number	61
Test Item	Fuselage Dialog Box: Rotation
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated with the correct x, y, & z rotational values
Input Specifications	• User inputs x, y, & z rotational values in degrees
Expected Output Specifications	• Fuselage is generated according to the user inputted x, y, & z rotational values or the default values of x: 0°, y: 0°, z: 0°
Pass/Fail Criteria	• Pass: Fuselage is generated correctly with the user inputted x, y, & z rotational values or the default values • Fail: Fuselage is not generated correctly with the user inputted x, y, & z rotational values or the fuselage is not generated with the correct default rotational values
Assumptions and Constraints	• Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI Button

Test Case Number	62
Test Item	Fuselage dialog box: Scale
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is created with the user specified x, y, & z values or the default values of 1.000
Input Specifications	• User inputs x, y, & z scale values
Expected Output Specifications	• Fuselage is generated according to the correct x, y, and z scale values
Pass/Fail Criteria	• Pass: Fuselage is generated according to the user specified values or default scale values • Fail: Fuselage is not generated according to the user specified values or default scale values
Assumptions and Constraints	• Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI button

Test Case Number	63
Test Item	Fuselage dialog box: Upper Height
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is created with the user specified upper height
Input Specifications	• User inputs upper height value
Expected Output Specifications	• Fuselage is generated according to the user inputted value or default value of 1.00
Pass/Fail Criteria	• Pass: Fuselage is generated according to the user specified value or default upper height value • Fail: Fuselage is not generated according to the user specified value or default upper height value
Assumptions and Constraints	• Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI button

Test Case Number	64
Test Item	Fuselage dialog box: Lower Height
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is created with the user specified lower height
Input Specifications	• User inputs lower height value
Expected Output Specifications	• Fuselage is generated according to the user inputted value or default value of 1.00
Pass/Fail Criteria	• Pass: Fuselage is generated according to the user specified value or default lower height value • Fail: Fuselage is not generated according to the user specified value or default lower height value
Assumptions and Constraints	• Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI button

Test Case Number	65
Test Item	Fuselage dialog box: Width
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is created with the user specified width
Input Specifications	• User inputs width value
Expected Output Specifications	• Fuselage is generated according to the user inputted value or default value of 1.00
Pass/Fail Criteria	• Pass: Fuselage is generated according to the user specified value or default width value • Fail: Fuselage is not generated according to the user specified value or default width value
Assumptions and Constraints	• Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI button

Test Case Number	66
Test Item	Fuselage Dialog Box: Color
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated with the chosen color
Input Specifications	• User clicks on the colorwheel grey color • User can select a color on the wheel, enter a RGB value, enter a HEX value, and/or enter HSV values
Expected Output Specifications	• Fuselage is generated according to the chosen color or the default grey color
Pass/Fail Criteria	• Pass: Fuselage is generated with the correct color • Fail: Fuselage is generated with an incorrect color
Assumptions and Constraints	• Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI Button

Test Case Number	67
Test Item	Fuselage dialog box: OK
Pre-conditions	Fuselage dialog box is opened
Post-conditions	Fuselage is generated and displayed
Input Specifications	● User clicks OK
Expected Output Specifications	● Fuselage is generated into 3D View with all the correct properties
Pass/Fail Criteria	● Pass: Fuselage is generated and displayed on the screen ● Fail: Fuselage is not generated and does not display on the screen
Assumptions and Constraints	● Assumption: Fuselage dialog box is already opened
Dependencies	5. Add Fuselage UI Button

Test Case Number	68
Test Item	Update Component: Wing Geometry: Initial Aft Thickness
Pre-conditions	Wing is generated
Post-conditions	Wing is updated with new aft thickness
Input Specifications	● User inputs aft thickness value
Expected Output Specifications	● Wing is updated based on user inputted value
Pass/Fail Criteria	● Pass: Wing is updated according to the user specified value ● Fail: Wing is not updated according to the user specified value
Assumptions and Constraints	● Assumption: Wing has already been generated
Dependencies	80. Update Component: Update Wing

Test Case Number	69
Test Item	Update Component: Wing Geometry: Tau Points
Pre-conditions	Wing is generated in Blender
Post-conditions	Wing is updated with Tau Points
Input Specifications	• User inputs tau points
Expected Output Specifications	• Wing model is updated according to the inputted tau points
Pass/Fail Criteria	• Pass: Wing is updated according to the user inputted tau points • Fail: Wing is not updated correctly with the inputted tau points
Assumptions and Constraints	• Assumption: Wing has already been generated • Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	80. Update Component: Update Wing

Test Case Number	70
Test Item	Update Component: Wing Geometry: Zeta Points
Pre-conditions	Wing is generated in Blender
Post-conditions	Wing is updated with Zeta Points
Input Specifications	• User inputs zeta points
Expected Output Specifications	• Wing model is updated according to the inputted zeta points
Pass/Fail Criteria	• Pass: Wing is updated according to the user inputted zeta points • Fail: Wing is not updated correctly with the inputted zeta points
Assumptions and Constraints	• Assumption: Wing has already been generated • Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	80. Update Component: Update Wing

Test Case Number	71
Test Item	Update Component: Wing Geometry: Airfoil Smoothness
Pre-conditions	Wing is generated in Blender
Post-conditions	Airfoil smoothness is set and updates object generation accordingly
Input Specifications	• User inputs airfoil smoothness
Expected Output Specifications	• Wing is updated with new airfoil smoothness
Pass/Fail Criteria	• Pass: Wing updates with new airfoil smoothness • Fail: Wing doesn't update with new airfoil smoothness
Assumptions and Constraints	• Assumption: Wing has already been generated
Dependencies	80. Update Component: Update Wing

Test Case Number	72
Test Item	Update Component: Wing Geometry: Base Washout
Pre-conditions	Wing is generated in Blender
Post-conditions	Washout value is set and updates object generation accordingly
Input Specifications	• User inputs washout value
Expected Output Specifications	• Wing is updated with new washout value
Pass/Fail Criteria	• Pass: Wing updates with new washout value • Fail: Wing doesn't update with new washout value
Assumptions and Constraints	• Assumption: Wing has already been generated
Dependencies	80. Update Component: Update Wing

Test Case Number	73
Test Item	Update Component: Wing Geometry: Tip Washout
Pre-conditions	Wing is generated in Blender
Post-conditions	Washout value is set and updates object generation accordingly
Input Specifications	• User inputs washout value
Expected Output Specifications	• Wing is updated with new washout value
Pass/Fail Criteria	• Pass: Wing updates with new washout value • Fail: Wing doesn't update with new washout value
Assumptions and Constraints	• Assumption: Wing has already been generated
Dependencies	80. Update Component: Update Wing

Test Case Number	74
Test Item	Update Component: Wing Geometry: Sweep
Pre-conditions	Wing is generated
Post-conditions	Sweep value is set and updates object generation accordingly
Input Specifications	• User inputs new sweep value
Expected Output Specifications	• Wing is updated with new sweep
Pass/Fail Criteria	• Pass: Wing is updated with new sweep value • Fail: Wing is not updated with new sweep value
Assumptions and Constraints	• Assumption: Wing has already been generated
Dependencies	80. Update Component: Update Wing

Test Case Number	75
Test Item	Update Component: Wing Geometry: Wing Length
Pre-conditions	Wing is generated in Blender
Post-conditions	Wing length value is set and updates object generation accordingly
Input Specifications	● User inputs wing length
Expected Output Specifications	● Wing length updates the object accordingly
Pass/Fail Criteria	● Pass: Wing is updated with new wing length ● Fail: Wing is not updated with new wing length
Assumptions and Constraints	● Assumption: Wing has already been generated
Dependencies	80. Update Component: Update Wing

Test Case Number	76
Test Item	Update Component: Wing: Transformations: Location
Pre-conditions	Wing is generated in Blender
Post-conditions	Wing location value is set and updates object generation accordingly
Input Specifications	● User inputs new wing location
Expected Output Specifications	● Wing location updates the object accordingly
Pass/Fail Criteria	● Pass: Wing is updated with new wing length ● Fail: Wing is not updated with new wing length
Assumptions and Constraints	● Assumption: Wing has already been generated
Dependencies	80. Update Component: Update Wing

Test Case Number	77
Test Item	Update Component: Wing: Transformations: Rotation
Pre-conditions	Wing is generated in Blender
Post-conditions	Euler rotation value is set and updates object generation accordingly
Input Specifications	• User inputs new rotation values
Expected Output Specifications	• Euler rotation updates the object accordingly
Pass/Fail Criteria	• Pass: Wing is updated with new rotation values • Fail: Wing is not updated with new rotation values
Assumptions and Constraints	• Assumption: Wing has already been generated
Dependencies	80. Update Component: Update Wing

Test Case Number	78
Test Item	Update Component: Wing: Transformations: Scale
Pre-conditions	Wing is generated in Blender
Post-conditions	Wing is updated with new scale values
Input Specifications	User inputs new scale values
Expected Output Specifications	• Wing is updated with new scale values
Pass/Fail Criteria	• Pass: Wing is updated with new scale values • Fail: Wing is not updated with new scale values
Assumptions and Constraints	• Assumption: Wing has already been generated
Dependencies	80. Update Component: Update Wing

Test Case Number	79
Test Item	Update Component: Wing: Color
Pre-conditions	Wing is generated
Post-conditions	Wing is updated with new selected color
Input Specifications	User selects a new color
Expected Output Specifications	• Wing is updated with new color
Pass/Fail Criteria	• Pass: Wing is updated with new selected color • Fail: Wing is not updated with new selected color
Assumptions and Constraints	• Assumption: Wing has already been generated
Dependencies	80. Update Component: Update Wing

Test Case Number	80
Test Item	Update Component: Update Wing
Pre-conditions	Wing is generated
Post-conditions	Any property changes will update the wing and display
Input Specifications	User clicks Update Wing
Expected Output Specifications	• Wing will be updated with any new changes made
Pass/Fail Criteria	• Pass: Wing is updated with new changes • Fail: Wing is not updated with new changes
Assumptions and Constraints	• Assumption: Wing has already been generated
Dependencies	3. Add Wing UI Button

Test Case Number	81
Test Item	Update Component: Delete Wing
Pre-conditions	Wing is generated
Post-conditions	Wing is deleted
Input Specifications	User clicks Delete Wing
Expected Output Specifications	• Selected wing is deleted
Pass/Fail Criteria	• Pass: Wing is deleted • Fail: Wing is not deleted
Assumptions and Constraints	• Assumption: Wing has already been generated
Dependencies	3. Add Wing UI Button

Test Case Number	82
Test Item	Update Component: Wing: Export to Excel
Pre-conditions	Wing is generated; Excel file is saved on specified location
Post-conditions	Wing tau and zeta points are saved to Excel file
Input Specifications	User clicks Export to Excel
Expected Output Specifications	• Wing tau and zeta points overwrite and save to the Excel file
Pass/Fail Criteria	• Pass: Wing points are saved correctly on Excel • Fail: Wing points are not saved correctly on Excel
Assumptions and Constraints	• Assumption: Wing has already been generated
Dependencies	3. Add Wing UI Button

Test Case Number	83
Test Item	Update Component: Symmetrical Wings Geometry: Initial Aft Thickness
Pre-conditions	Symmetrical Wings are generated
Post-conditions	Symmetrical wings are updated with new initial aft thickness
Input Specifications	● User enters new initial aft thickness
Expected Output Specifications	● Symmetrical wings are updated based on user input
Pass/Fail Criteria	● Pass: Symmetrical wings are updated with new initial aft thickness ● Fail: Symmetrical wings are not updated with new initial aft thickness
Assumptions and Constraints	● Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	84
Test Item	Update Component: Symmetrical Wings Geometry: Tau Points
Pre-conditions	Symmetrical wings are generated
Post-conditions	Symmetrical wings are updated with new tau points
Input Specifications	● User inputs new tau points
Expected Output Specifications	● Symmetrical wings are updated according to the new tau points
Pass/Fail Criteria	● Pass: Symmetrical wings are updated with the new tau points ● Fail: Symmetrical wings are not updated with new tau points
Assumptions and Constraints	● Assumption: Symmetrical wings have already been generated ● Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	85
Test Item	Update Component: Symmetrical Wings Geometry: Zeta Points
Pre-conditions	Symmetrical wings are generated
Post-conditions	Symmetrical wings are updated with new zeta points
Input Specifications	• User inputs new zeta points
Expected Output Specifications	• Symmetrical wings are updated with new zeta points
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new zeta points • Fail: Symmetrical wings are not updated with new zeta points
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated • Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	86
Test Item	Update Component: Symmetrical Wings Wing Geometry: Airfoil Smoothness
Pre-conditions	Symmetrical wings are generated
Post-conditions	Airfoil smoothness is set and updates object generation accordingly
Input Specifications	• User inputs new airfoil smoothness
Expected Output Specifications	• Symmetrical wings are updated with new airfoil smoothness
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new airfoil smoothness • Fail: Symmetrical wings are not updated with new airfoil smoothness
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	87
Test Item	Update Component: Symmetrical Wings Wing Geometry: Base Washout
Pre-conditions	Symmetrical wings are generated
Post-conditions	Washout value is set and updates object generation accordingly
Input Specifications	• User inputs new washout value
Expected Output Specifications	• Symmetrical wings are updated with new washout value
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new washout value • Fail: Symmetrical wings are not updated with new washout value
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	88
Test Item	Update Component: Symmetrical Wings Wing Geometry: Tip Washout
Pre-conditions	Symmetrical wings are generated
Post-conditions	Washout value is set and updates object generation accordingly
Input Specifications	• User inputs new washout value
Expected Output Specifications	• Symmetrical wings are updated with new washout value
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new washout value • Fail: Symmetrical wings are not updated with new washout value
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	89
Test Item	Update Component: Symmetrical Wings Geometry: Sweep
Pre-conditions	Symmetrical wings are generated
Post-conditions	Sweep angle is set and updates object generation accordingly
Input Specifications	• User inputs new sweep value
Expected Output Specifications	• Washout displacement updates the object accordingly
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new sweep value • Fail: Symmetrical wings are not updated with new sweep value
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	90
Test Item	Update Component: Symmetrical Wings Geometry: Wing Length
Pre-conditions	Symmetrical wings are generated
Post-conditions	Wing length value is set and updates object generation accordingly
Input Specifications	• User inputs new wing length value
Expected Output Specifications	• Wing length updates the object accordingly
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new wing length • Fail: Symmetrical wings are not updated with new wing length
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	91
Test Item	Update Component: Symmetrical Wings Wing Geometry: Inner Space
Pre-conditions	Symmetrical wings are generated
Post-conditions	Inner space is set and updates object generation accordingly
Input Specifications	• User inputs new inner space value
Expected Output Specifications	• Symmetrical wings are updated with new inner space value
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new inner space • Fail: Symmetrical wings are not updated with new inner space
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	92
Test Item	Update Component: Symmetrical Wings: Transformations: Location
Pre-conditions	Symmetrical wings are generated
Post-conditions	Location is set and updates object generation accordingly
Input Specifications	• User inputs new location values
Expected Output Specifications	• Symmetrical wings are updated with new location values
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new location • Fail: Symmetrical wings are not updated with new location
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	93
Test Item	Update Component: Symmetrical Wings: Transformations: Euler Rotation
Pre-conditions	Symmetrical wings are generated
Post-conditions	Euler rotation value is set and updates object generation accordingly
Input Specifications	• User inputs new rotation values
Expected Output Specifications	• Euler rotation updates the object accordingly
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new rotational values • Fail: Symmetrical wings are not updated with new rotational values
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	94
Test Item	Update Component: Symmetrical Wings: Transformations: Scale
Pre-conditions	Symmetrical wings are generated
Post-conditions	Symmetrical wings are updated with new scale values
Input Specifications	User enters new scale values
Expected Output Specifications	• Symmetrical wings are updated with new scale values
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new scale values • Fail: Symmetrical wings are not updated with new scale values
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	95
Test Item	Update Component: Symmetrical Wings: Color
Pre-conditions	Symmetrical Wings are generated
Post-conditions	Symmetrical wings are updated with new color
Input Specifications	User selects a new color
Expected Output Specifications	• Symmetrical wings are updated with new selected color
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new selected color • Fail: Symmetrical wings are not updated with new selected color
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	96. Update Component: Update Symmetrical Wings

Test Case Number	96
Test Item	Update Component: Update Symmetrical Wings
Pre-conditions	Symmetrical wings are generated
Post-conditions	Symmetrical wings are updated with any changes made
Input Specifications	User clicks Update Wings
Expected Output Specifications	• Symmetrical wings are updated with new changes
Pass/Fail Criteria	• Pass: Symmetrical wings are updated with new changes • Fail: Symmetrical wings are not updated with new changes
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	97
Test Item	Update Component: Delete Symmetrical Wings
Pre-conditions	Symmetrical wings are generated
Post-conditions	Symmetrical wings are deleted
Input Specifications	User clicks Delete Wings
Expected Output Specifications	• Symmetrical wings are deleted
Pass/Fail Criteria	• Pass: Symmetrical wings are deleted • Fail: Symmetrical wings are not deleted
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	4. Add Symmetrical Wings UI button

Test Case Number	98
Test Item	Update Component: Symmetrical Wings: Export to Excel
Pre-conditions	Symmetrical wings are generated; Excel file is saved on specified location
Post-conditions	Tau and zeta points are saved to Excel file
Input Specifications	User clicks Export to Excel
Expected Output Specifications	• Tau and zeta points overwrite and save to the Excel file
Pass/Fail Criteria	• Pass: Points are saved correctly on Excel • Fail: Points are not saved correctly on Excel
Assumptions and Constraints	• Assumption: Symmetrical wings have already been generated
Dependencies	4. Add Symmetrical Wings UI Button

Test Case Number	99
Test Item	Update Component: Pod Geometry: Initial Aft Thickness
Pre-conditions	Pod is generated
Post-conditions	Pod is updated with new initial aft thickness
Input Specifications	● User inputs new initial aft thickness
Expected Output Specifications	● Pod is updated with new initial aft thickness value
Pass/Fail Criteria	● Pass: Pod is updated with new initial aft thickness ● Fail: Pod is not updated with new initial aft thickness
Assumptions and Constraints	● Assumption: Pod has already been generated
Dependencies	109. Update Component: Update Pod

Test Case Number	100
Test Item	Update Component: Pod Geometry: Tau Points
Pre-conditions	Pod is generated
Post-conditions	Pod is updated with new tau points
Input Specifications	● User inputs new tau points
Expected Output Specifications	● Pod is updated according to the tau points
Pass/Fail Criteria	● Pass: Pod is updated with new tau points ● Fail: Pod is not updated with new tau points
Assumptions and Constraints	● Assumption: Pod has already been generated ● Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	109. Update Component: Update Pod

Test Case Number	101
Test Item	Update Component: Pod Geometry: Zeta Points
Pre-conditions	Pod is generated
Post-conditions	Pod is updated with new zeta points
Input Specifications	• User inputs zeta points
Expected Output Specifications	• Pod is updated according to the zeta points
Pass/Fail Criteria	• Pass: Pod is updated with new zeta points • Fail: Pod is not updated with new zeta points
Assumptions and Constraints	• Assumption: Pod has already been generated • Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	109. Update Component: Update Pod

Test Case Number	102
Test Item	Update Component: Pod Geometry: Placement Points
Pre-conditions	Pod is generated
Post-conditions	Pod is updated with new placement points
Input Specifications	• User inputs placement points
Expected Output Specifications	• Pod is updated according to the placement points
Pass/Fail Criteria	• Pass: Pod is updated with new placement points • Fail: Pod is not updated with new placement points
Assumptions and Constraints	• Assumption: Pod has already been generated
Dependencies	109. Update Component: Update Pod

Test Case Number	103
Test Item	Update Component: Pod Geometry: X-Z Smoothness
Pre-conditions	Pod is generated
Post-conditions	Pod is updated with new smoothness value
Input Specifications	• User enters smoothness value
Expected Output Specifications	• Pod is updated with new smoothness value
Pass/Fail Criteria	• Pass: Pod is updated with new smoothness value • Fail: Pod is not updated with new smoothness value
Assumptions and Constraints	• Assumption: Pod has already been generated
Dependencies	109. Update Component: Update Pod

Test Case Number	104
Test Item	Update Component: Pod Geometry: Y-Z Smoothness
Pre-conditions	Pod is generated
Post-conditions	Pod is updated with new smoothness value
Input Specifications	• User enters smoothness value
Expected Output Specifications	• Pod is updated with new smoothness value
Pass/Fail Criteria	• Pass: Pod is updated with new smoothness value • Fail: Pod is not updated with new smoothness value
Assumptions and Constraints	• Assumption: Pod has already been generated
Dependencies	109. Update Component: Update Pod

Test Case Number	105
Test Item	Update Component: Pod: Transformations: Location
Pre-conditions	Pod is generated
Post-conditions	Location value is set and updates object generation accordingly
Input Specifications	• User inputs new pod location values
Expected Output Specifications	• Pod is updated with new location values
Pass/Fail Criteria	• Pass: Pod is updated with new location values • Fail: Pod is not updated with new location values
Assumptions and Constraints	• Assumption: Pod has already been generated
Dependencies	109. Update Component: Update Pod

Test Case Number	106
Test Item	Update Component: Pod: Transformations: Euler Rotation
Pre-conditions	Pod is generated
Post-conditions	Euler rotation value is set and updates object generation accordingly
Input Specifications	• User inputs new rotational values
Expected Output Specifications	• Euler rotation updates the object accordingly
Pass/Fail Criteria	• Pass: Pod is updated with new rotational values • Fail: Pod is not updated with new rotational values
Assumptions and Constraints	• Assumption: Pod has already been generated
Dependencies	109. Update Component: Update Pod

Test Case Number	107
Test Item	Update Component: Pod: Transformations: Scale
Pre-conditions	Pod is generated
Post-conditions	Pod is updated with new scale values
Input Specifications	User inputs new scale values
Expected Output Specifications	• Pod is updated with new scale values
Pass/Fail Criteria	• Pass: Pod is updated with new scale values • Fail: Pod is not updated with new scale values
Assumptions and Constraints	• Assumption: Pod has already been generated
Dependencies	109. Update Component: Update Pod

Test Case Number	108
Test Item	Update Component: Pod: Color
Pre-conditions	Pod is generated
Post-conditions	New color is selected and object is updated accordingly
Input Specifications	User selects a new color
Expected Output Specifications	• Pod is updated with new selected color
Pass/Fail Criteria	• Pass: Pod is updated with new selected color • Fail: Pod is not updated with new selected color
Assumptions and Constraints	• Assumption: Pod has already been generated
Dependencies	109. Update Component: Update Pod

Test Case Number	109
Test Item	Update Component: Update Pod
Pre-conditions	Pod is generated
Post-conditions	Pod is updated with any changes made
Input Specifications	User clicks Update Pod
Expected Output Specifications	• Pod is updated with any new changes
Pass/Fail Criteria	• Pass: Pod is updated with all new changes • Fail: Pod is not updated with new changes
Assumptions and Constraints	• Assumption: Pod has already been generated
Dependencies	6. Add Pod UI Button

Test Case Number	110
Test Item	Update Component: Delete Pod
Pre-conditions	Pod is generated
Post-conditions	Pod is deleted
Input Specifications	User clicks Delete Pod
Expected Output Specifications	• Pod is deleted
Pass/Fail Criteria	• Pass: Pod is deleted • Fail: Pod is not deleted
Assumptions and Constraints	• Assumption: Pod has already been generated
Dependencies	6. Add Pod UI Button

Test Case Number	111
Test Item	Update Component: Pod: Export to Excel
Pre-conditions	Pod is generated; Excel file is saved on specified location
Post-conditions	Tau and zeta points are saved to Excel file
Input Specifications	User clicks Export to Excel
Expected Output Specifications	● Tau and zeta points overwrite and save to the Excel file
Pass/Fail Criteria	● Pass: Points are saved correctly on Excel ● Fail: Points are not saved correctly on Excel
Assumptions and Constraints	● Assumption: Pod has already been generated
Dependencies	6. Add Pod UI Button

Test Case Number	112
Test Item	Update Component: Fuselage Geometry: Tau Points
Pre-conditions	Fuselage is generated
Post-conditions	Fuselage is updated with new tau points
Input Specifications	● User enters new tau points
Expected Output Specifications	● Fuselage is updated with new tau points
Pass/Fail Criteria	● Pass: Fuselage is updated with new tau points ● Fail: Object is not updated with new tau points
Assumptions and Constraints	● Assumption: Fuselage has already been generated ● Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	126. Update Component: Update Fuselage

Test Case Number	113
Test Item	Update Component: Fuselage Geometry: Zeta Points
Pre-conditions	Fuselage is generated
Post-conditions	Fuselage is updated with new zeta points
Input Specifications	• User enters new zeta points
Expected Output Specifications	• Fuselage is updated with new zeta points
Pass/Fail Criteria	• Pass: Fuselage is updated with new zeta points • Fail: Fuselage is not updated with new zeta points
Assumptions and Constraints	• Assumption: Fuselage has already been generated • Constraints: If adding more/removing more values than the default number of 6 values, the tau and zeta points must have the same number of values
Dependencies	126. Update Component: Update Fuselage

Test Case Number	114
Test Item	Update Component: Fuselage Geometry: Upper Points
Pre-conditions	Fuselage is generated
Post-conditions	New upper points are inputted and object updates
Input Specifications	• User inputs new upper points
Expected Output Specifications	• Fuselage is updated with new upper points
Pass/Fail Criteria	• Pass: Fuselage is updated with new upper points • Fail: Fuselage is not updated with new upper points
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	115
Test Item	Update Component: Fuselage Geometry: Lower Points
Pre-conditions	Fuselage is generated
Post-conditions	New lower points are set and object updates accordingly
Input Specifications	• User inputs new lower points
Expected Output Specifications	• Fuselage is updated with new lower points
Pass/Fail Criteria	• Pass: Fuselage is updated with new lower points • Fail: Fuselage is not updated with new lower points
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	116
Test Item	Update Component: Fuselage Geometry: Placement Points
Pre-conditions	Fuselage is generated
Post-conditions	New placement points are set and object updates accordingly
Input Specifications	• User enters new placement points
Expected Output Specifications	• Fuselage is updated with new placement points
Pass/Fail Criteria	• Pass: Fuselage is updated with new placement points • Fail: Fuselage is not updated with new placement points
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	117
Test Item	Update Component: Fuselage Geometry: X-Z Smoothness
Pre-conditions	Fuselage is generated
Post-conditions	Fuselage is updated with new smoothness value
Input Specifications	• User enters smoothness value
Expected Output Specifications	• Fuselage is updated with new smoothness value
Pass/Fail Criteria	• Pass: Fuselage is updated with new smoothness value • Fail: Fuselage is not updated with new smoothness value
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	118
Test Item	Update Component: Fuselage Geometry: Y-Z Smoothness
Pre-conditions	Fuselage is generated
Post-conditions	Fuselage is updated with new smoothness value
Input Specifications	• User enters smoothness value
Expected Output Specifications	• Fuselage is updated with new smoothness value
Pass/Fail Criteria	• Pass: Fuselage is updated with new smoothness value • Fail: Fuselage is not updated with new smoothness value
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	119
Test Item	Update Component: Fuselage Transformations: Upper Height
Pre-conditions	Fuselage is generated
Post-conditions	Fuselage is updated with new upper height value
Input Specifications	• User enters upper height value
Expected Output Specifications	• Fuselage is updated with upper height value
Pass/Fail Criteria	• Pass: Fuselage is updated with new upper height value • Fail: Fuselage is not updated with new upper height value
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	120
Test Item	Update Component: Fuselage Transformations: Lower Height
Pre-conditions	Fuselage is generated
Post-conditions	Fuselage is updated with new lower height value
Input Specifications	• User enters lower height value
Expected Output Specifications	• Fuselage is updated with lower height value
Pass/Fail Criteria	• Pass: Fuselage is updated with new lower height value • Fail: Fuselage is not updated with new lower height value
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	121
Test Item	Update Component: Fuselage Transformations: Width
Pre-conditions	Fuselage is generated
Post-conditions	Fuselage is updated with new width value
Input Specifications	• User enters width value
Expected Output Specifications	• Fuselage is updated with new width value
Pass/Fail Criteria	• Pass: Fuselage is updated with new width • Fail: Fuselage is not updated with new width
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	122
Test Item	Update Component: Fuselage: Transformations: Location
Pre-conditions	Fuselage is generated
Post-conditions	Location value is set and updates object generation accordingly
Input Specifications	• User enters new location values
Expected Output Specifications	• Fuselage is updated with new location values
Pass/Fail Criteria	• Pass: Fuselage is updated with new location • Fail: Fuselage is not updated with new location
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	123
Test Item	Update Component: Fuselage: Transformations: Euler Rotation
Pre-conditions	Fuselage is generated
Post-conditions	Euler rotation value is set and updates object generation accordingly
Input Specifications	● User enters new rotational values
Expected Output Specifications	● Fuselage is updated with new rotational values
Pass/Fail Criteria	● Pass: Fuselage is updated with new rotation ● Fail: Fuselage is not updated with new rotation
Assumptions and Constraints	● Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	124
Test Item	Update Component: Fuselage: Transformations: Scale
Pre-conditions	Fuselage is generated
Post-conditions	Scale values are set and object updates accordingly
Input Specifications	User enters new scale values
Expected Output Specifications	● Fuselage is updated with new scale values
Pass/Fail Criteria	● Pass: Fuselage is updated with new scale ● Fail: Fuselage is not updated with new scale
Assumptions and Constraints	● Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	125
Test Item	Update Component: Fuselage: Color
Pre-conditions	Fuselage is generated
Post-conditions	Fuselage is updated with new selected color
Input Specifications	User selects a new color
Expected Output Specifications	• Fuselage is updated with new selected color
Pass/Fail Criteria	• Pass: Fuselage is updated with new color • Fail: Fuselage is not updated with new color
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	126. Update Component: Update Fuselage

Test Case Number	126
Test Item	Update Component: Update Fuselage
Pre-conditions	Fuselage is generated
Post-conditions	Fuselage is updated with any new changes
Input Specifications	User clicks Update Fuselage
Expected Output Specifications	• Fuselage is updated with any newly made changes
Pass/Fail Criteria	• Pass: Fuselage is updated with all new changes • Fail: Fuselage is not updated with all the changes
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	5. Add Fuselage UI button

Test Case Number	127
Test Item	Update Component: Delete Fuselage
Pre-conditions	Fuselage is generated
Post-conditions	Fuselage is deleted
Input Specifications	User clicks Delete Fuselage
Expected Output Specifications	• Fuselage is deleted
Pass/Fail Criteria	• Pass: Fuselage is deleted • Fail: Fuselage is not deleted
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	5. Add Fuselage UI Button

Test Case Number	128
Test Item	Update Component: Fuselage: Export to Excel
Pre-conditions	Fuselage is generated; Excel file is saved on specified location
Post-conditions	Tau and zeta points are saved to Excel file
Input Specifications	User clicks Export to Excel
Expected Output Specifications	• Tau and zeta points overwrite and save to the Excel file
Pass/Fail Criteria	• Pass: Points are saved correctly on Excel • Fail: Points are not saved correctly on Excel
Assumptions and Constraints	• Assumption: Fuselage has already been generated
Dependencies	5. Add Fuselage UI Button

7.3 Justification of Test Cases

For our project, we used unit testing, functional testing, and regression testing. We utilized unit testing by having test cases for all of the smallest testable parts of our project, which were the individual functions. Thus, we also incorporated functional testing. All of our test cases are written to be re-used through regression testing. If we make any changes to our project, all of our test cases will be re-tested. Our test cases are all relevant and necessary because they test each and every functionality of our components: wing, symmetrical wings, pod, and fuselage.

7.4 Test Run Procedures and Results

We manually went through all of the test cases to test each and every function of our application and found no errors. While testing, if something was found to be erroneous, we fixed the problem and re-tested.

7.5 Discussion of Test Results

After thoroughly testing our application, we found that everything works properly. All of our components (wing, symmetrical wings, pod, and fuselage) generate correctly, update correctly, delete correctly, etc. All of the editable properties (tau points, scale, color, etc.) work and update accordingly. Importing from and exporting to CSV also work fine. We found our method of unit, functional, and regression testing very useful because it allowed us to have reusable written tests for every case possible of our application, which allowed us to test every possibility that our project could have an error.

8. Conclusions

8.1 Summary

Our project's main objective was to create a 3D aircraft modeling application to be used for 3D printing. We worked with Blender, an open source 3D graphics software, and used Python to program. The principal focus of our project was utilizing parametric cubic splines for the aircraft geometries. Our project supports full parametric cubic spline generation and allows users to input a set of points, which would then generate appropriate curves and generate a component. The components we implemented are: wing, symmetrical wings, pod, and fuselage.

We created an easy to install Blender packaged addon for our application. Our addon features an easily navigable UI that allows users to easily input and edit geometric properties such as wing length, rotation, color, etc. We implemented support for Excel files so that users can import their own set of points from a spreadsheet and also export their points to the same spreadsheet. Ultimately, the generated aircraft model from our application in Blender can be exported for 3D printing.

8.2 Problems Encountered and Solved

One of our first problems and our main problem was implementing parametric cubic splines. We initially used cubic splines instead but eventually fixed our code to work with parametric cubic splines. A second problem that we encountered was figuring out how to extend the cross-sectional area of the fuselage from 2D to 3D. We solved this by determining equations for 3D coordinates. A third problem was that reading points from an Excel file meant that an object couldn't be modified. This was fixed by implementing the Excel reader and having an update object function. Another big issue we had with our application was that generating a parametric cubic spline took a considerable amount of time. We eventually optimized our code to run faster and more efficiently. Other problems we had were small bugs and errors that were easily fixable.

8.3 Suggestions for Better Approaches

A better approach to our project would have been to gather more requirements and information before starting. We dove into the project without much planning and completed unnecessary things such as translating code we didn't use. We should have laid out a complete plan on how to implement the project before starting.

8.4 Suggestions for Future Extensions to Project

Future extensions for this project would include adding more aircraft components such as an engine, propeller, etc. Also, we could add extra features and properties. A feature that we would have liked to implement if time permitted was adding a texture or selected image to each component. Moreover, we could further optimize our code and parametric cubic spline generation.

Northrop Grumman Student Design Project:

User Manual

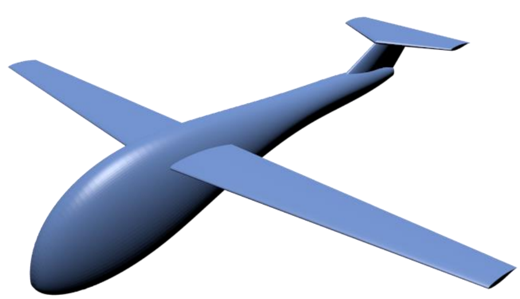


Table of Contents

Installation

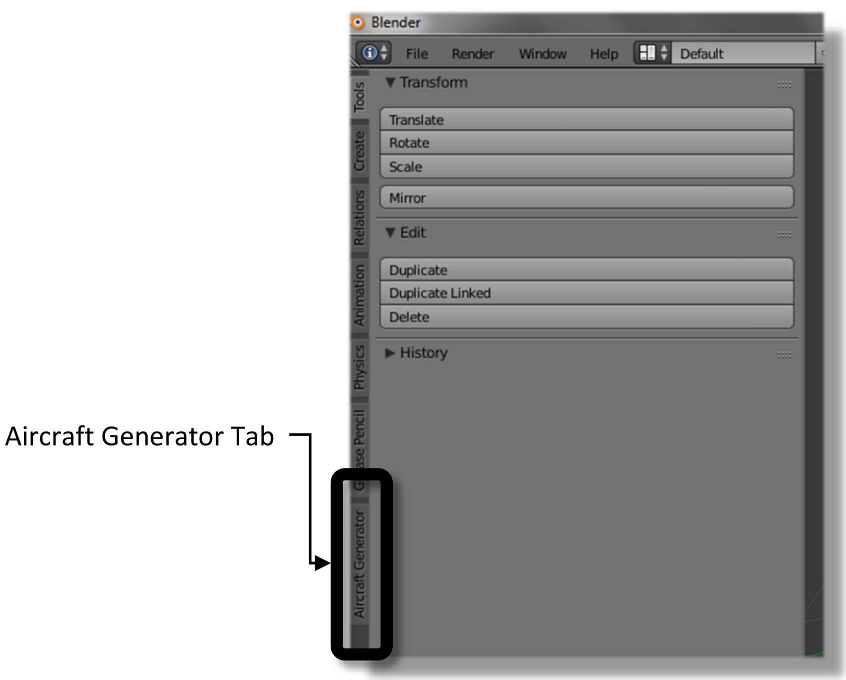
Before beginning the installation process, download the addon from our website at www.csulb-ngcproject.me.

To install the addon, follow these steps:

1. Open the *Blender 3D* application
2. Navigate to *File* > *User Preferences* (Ctrl + Alt + U)
3. Select the *Addons* tab
4. Select *Install from File…*
5. Navigate the file explorer to the target location of the CSULB-NGC Student Design Project addon zip
6. Select the zipped file and push *Install from file…* (or double click the file)
7. *Add Mesh: Aircraft Components* should appear on the dialog box, tick the checkmark box to activate the addon
8. (Optional) Select *Save User Settings* to keep the addon activated on future startups of the Blender application

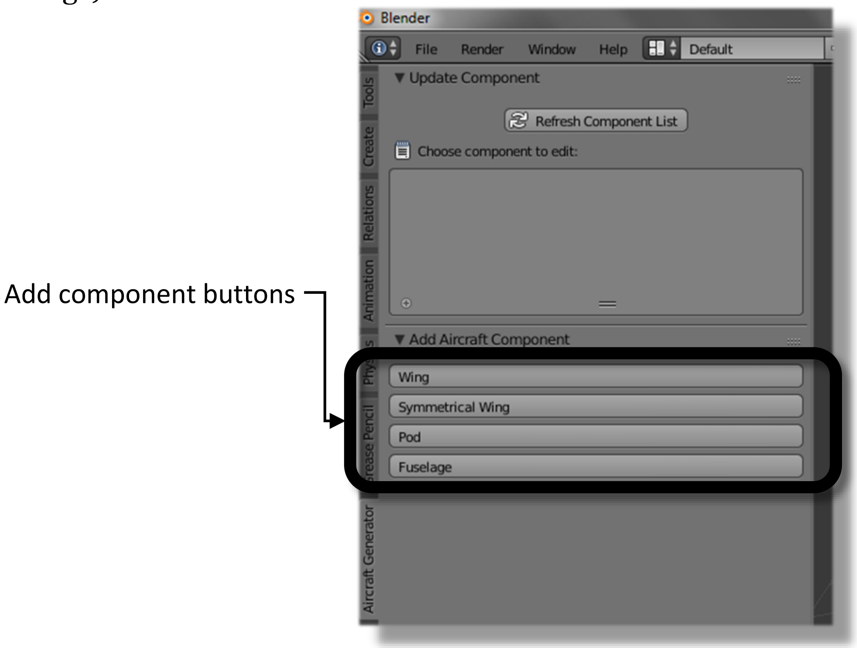
Navigating to the Addon GUI

To navigate into the addon GUI, select the Aircraft Generator tab from the tab bar on the left-hand side.



Adding a Component

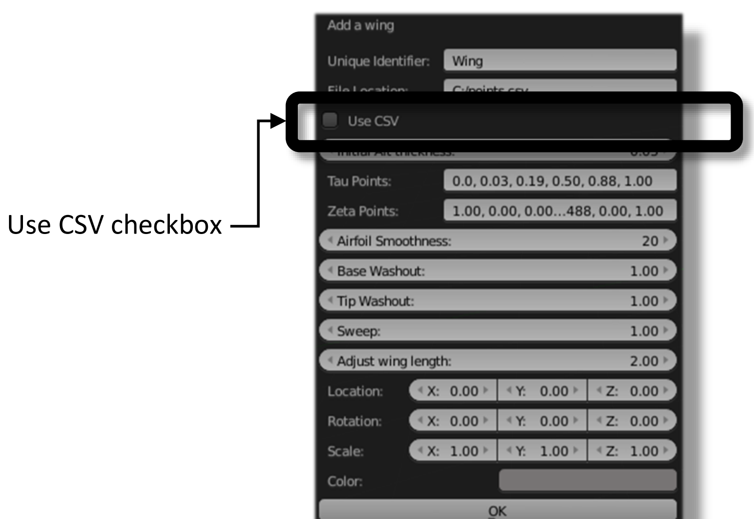
To add a component, simply push the corresponding button (wing, symmetrical wings, pod, fuselage).



Importing CSV

Importing data points for the parametric cubic spline with CSV allows for a faster, more efficient method to generate the aircraft component geometry. See *Troubleshooting Section / FAQ* for CSV format examples. To import a CSV file, follow these steps:

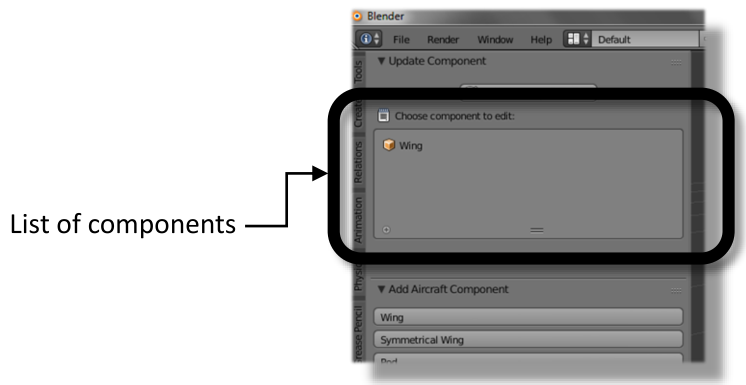
1. Push the corresponding "add button" of the component to import.
2. When the dialog box appears, mark the checkbox labeled *"Use CSV"*.
3. After making any changes to the parameters, select *"OK"*.



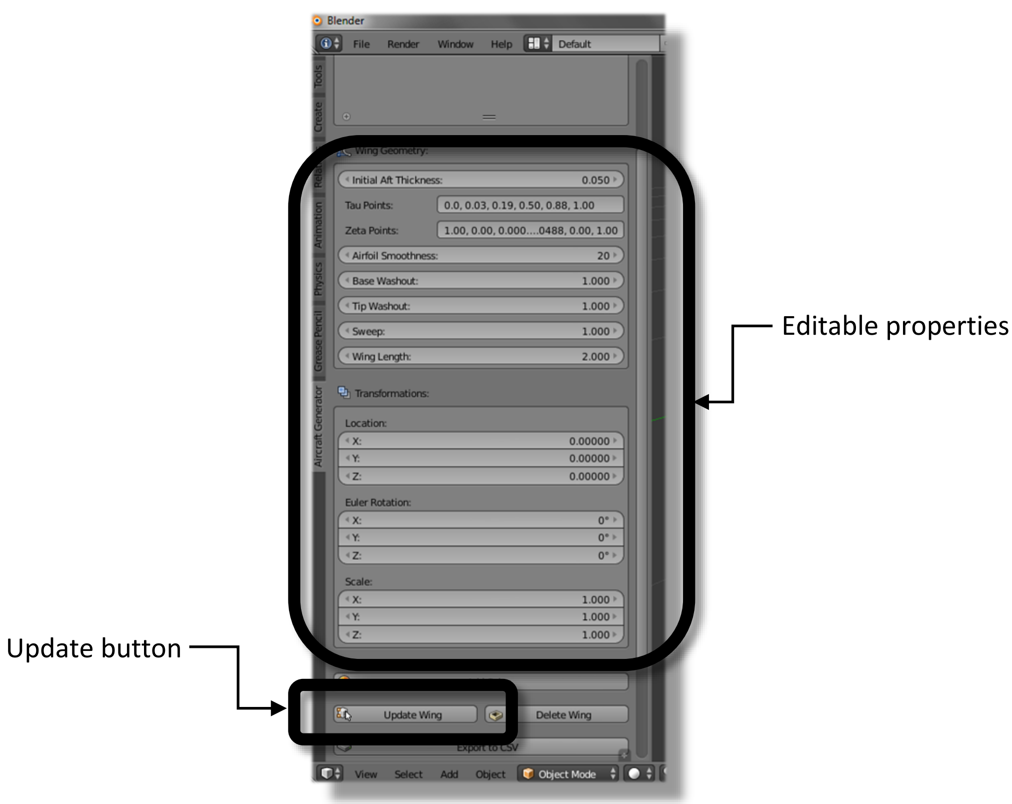
Editing Properties

Each component contains a number of geometry-defining parameters that can be adjusted. To access these parameters, use the following steps:

1. Select the component to edit in the GUI's list of components. *Note this will jump the 3D View's focus to the selected component.
2. Edit any of the parameters that appear.
3. Push the "Update" button at the bottom of the GUI.



Before selecting component



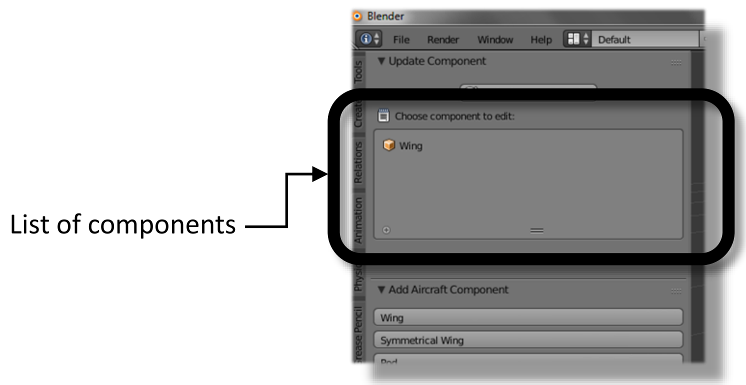
After selecting component

List of Editable Properties and Definitions	
Initial Aft Thickness	Controls the initial thickness of components (wing, symmetrical wings, pod)
Tau Points	Used to parametrically characterize the curvatures
Zeta Points	Used to parametrically characterize the curvatures
Upper Points	Used to parametrically characterize the upper half of a fuselage component
Lower Points	Used to parametrically characterize the lower half of a fuselage component
Placement Points	Used to parametrically characterize the positions of a component's cross sections
Base Washout	Controls the size of the washout at the base of a wing
Tip Washout	Controls the size of the washout at the tip of a wing
Sweep	Controls the angle of the wing from base to tip
Wing Length	Controls the length of the wing
Airfoil Smoothness	Defines the smoothness of the curvature defining an airfoil
X-Z Smoothness	Defines the smoothness of the curvature along the X-Z axes
Y-Z Smoothness	Defines the smoothness of the curvature along the Y-Z axes
Upper Height	Controls the height of the upper half of a fuselage
Lower Height	Controls the height of the lower half of a fuselage
Width	Control the width of a fuselage
Location	A vector that defines an object's location
Rotation	A vector that defines an object's rotation
Scale	A vector that defines an object's scale

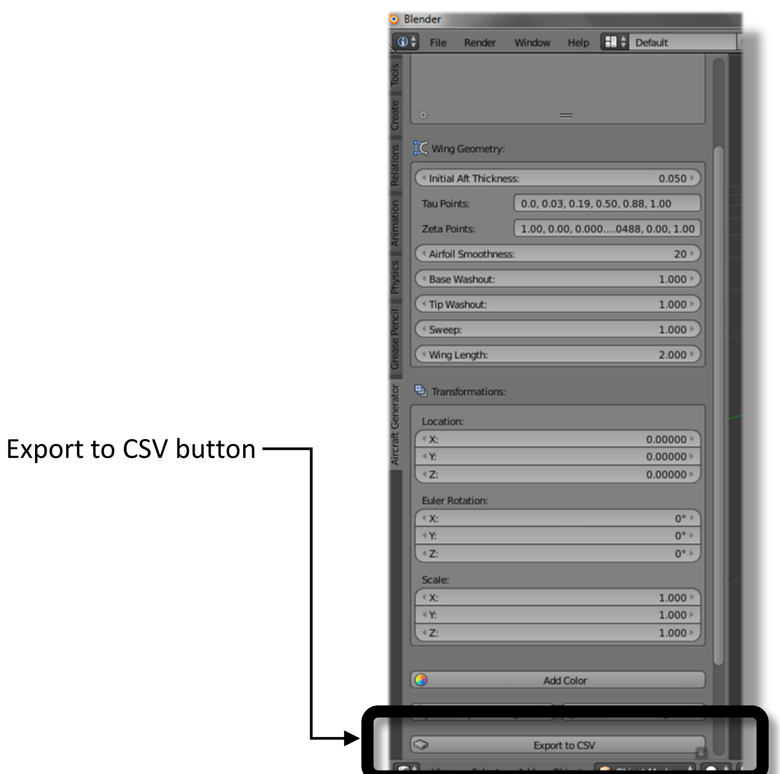
Exporting CSV

The application has the option to export the point data into a CSV file. To do so, follow these steps:

1. Select the component to export in the GUI's list of components. *Note this will jump the 3D View's focus to the selected component.
2. Scroll to the bottom of the GUI and select the "Export to CSV" option.
3. A prompt will appear asking if you wish to continue.
4. The CSV file will be written to either the import file location or the default location (directly to the C drive root folder).



Before selecting component

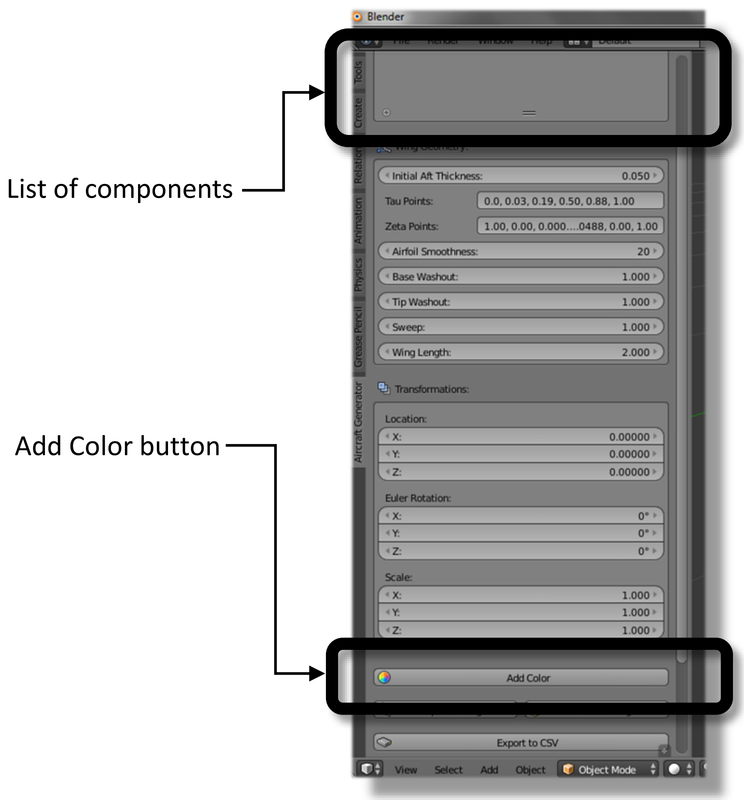


After selecting component

Adding Color

The application has the option to add color to a component. To do so, follow these steps:

1. Select the component to add color to in the GUI's list of components. *Note this will jump the 3D View's focus to the selected component.
2. Scroll to the bottom of the GUI and select the "Add Color" option.
3. A color wheel will appear. Click on the desired color, or enter a value (RGB, HSV, HEX).



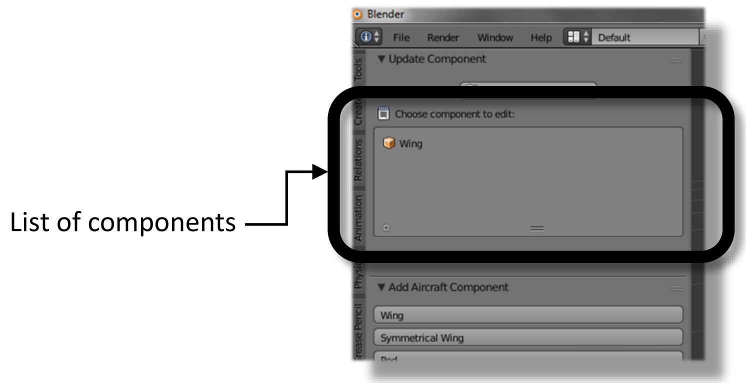
After selecting component

Color Wheel Popup

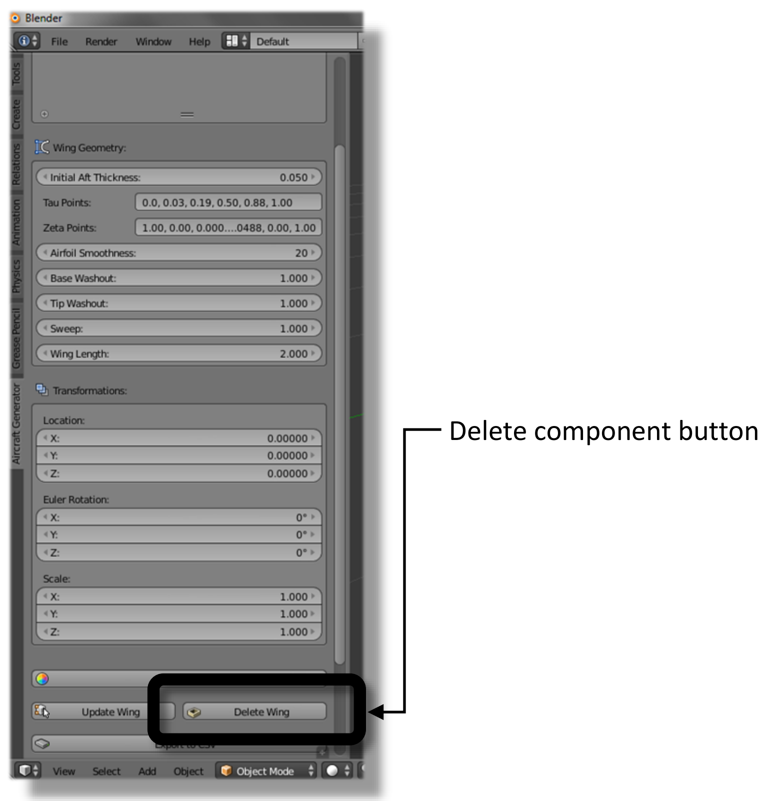
Deleting a Component

To delete a component, simply follow these steps:

1. Select the component to delete from the GUI's list of components. *Note this will jump the 3D View's focus to the selected component.
2. Scroll to the bottom of the GUI and select the "Delete" button.
3. The component will be deleted from the 3D View as well as the list of components.



Before selecting component



After selecting component

Rendering an Image

Rendering an image allows you to see the generated aircraft in higher graphical detail. To render an image of your 3D model, push *F12* on your keyboard. Pushing *ESC* on your keyboard will exit the render view. You can save this image by pushing *F3* on your keyboard, prompting a selector.

A rendered image of a sailplane model

Troubleshooting Section/FAQ

Why does it take long to generate some components?
The amount of points "n" input are used to generate cross sections along the component. While this allows for a smoother curve, it also means for a longer generation time depending on the speed of your computer.

Why does my (component) look strange?
The set of points input control the shape of the parametric cubic spline generated. Because of the nature of parametric cubic splines (maintained continuity on 1st and 2nd derivatives), the shape of curvature generated can make unusual shapes.

Why is the bottom of the fuselage hollow?
The set of points input for the lower half of the fuselage dictates the curvature generated by the parametric cubic spline. Though it looks as if the fuselage has no lower half, what probably occurred is that the curvature shape cuts into the fuselage.

Why is the component still in the component list after deleting?
If you deleted the component utilizing the 3D View or object outliner, the component list will not refresh. Pushing the *"Refresh Component"* List button will properly display the list.

How should the CSV files be formatted for input?
Here are a couple of outline for CSV input format. *Symmetrical wings* will be similar to *Wing*, while *Pod* will only have *Tau*, *Zeta*, and *Placement* points.

Tau	Zeta
0	0
0.13	0.0007
0.19	-0.049
0.5f	0
0.88	0.0488
1	0

Wing CSV

tau	zeta	tau fuselage	zeta fuselage	upper points	lower points	placement p
0	1	0	1	1	1	0
0.03	0	0.5	0.005	0	0	0
0.19	0.0007	0.8366	0.08	0.2	0	0.05
0.5	-0.049	1	0.04	0.19	0.11	0.15
0.88	0		0.005	0.16	0.14	0.15
1	0.0488		-0.5	-0.6	0.4	1
	0					
	1					

Fuselage CSV

Contact Details

For more information, visit our website at www.csulb-ngcproject.me or contact one of the team members at the following emails:

Marvin Trajano marvin.trajano@student.csulb.edu
Lisa Tran lisa.tran02@student.csulb.edu
Benjamin Kray benjamin.kray@student.csulb.edu
Victor Tran victor.tran@student.csulb.edu
Keith Farnham keith.farnham@student.csulb.edu
Bryce Burnett bryce.burnett@student.csulb.edu