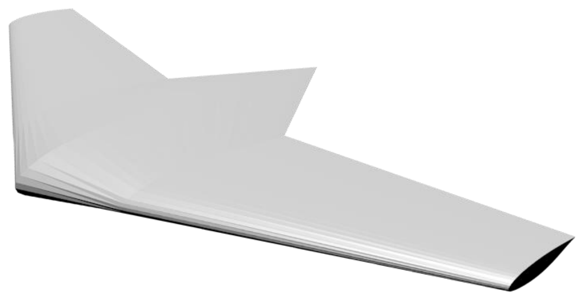
Northrop Grumman Student Design Project



csulb-ngcproject.me

This project features a standalone application for an aircraft wing generation that interacts with the Blender software. It utilizes the parametric polynomial, parametric cubic spline, and parametric Fourier to generate the geometries for a component.

CECS 490B - Spring 2015

Marvin Trajano
Lisa Tran
Victor Tran

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. We looked to utilize an open source software called Blender to create a superior application for generating aircraft models.

1.2 Previous and Current Work
Last semester we created a custom addon for Blender that would allow a user to generate and edit the geometries for various aircraft components, namely the wing and fuselage. This semester we learned new, more efficient, geometric methods and implemented them into a new standalone application.

1.2.1 Parametric Cubic Spline
Last semester we learned about the geometric generation method known as the parametric cubic spline. The way this method works is by specifying a number of points around an airfoil. A spline, or piecewise polynomial function, would then be interpolated so that continuity is maintained up through to the second derivative. This characteristic of the parametric cubic spline meant that a generated airfoil would be smooth to the air flowing by it.

1.2.2 Parametric Polynomial
This semester we were tasked with learning a new geometric method that is much more intuitive than the parametric cubic spline and arguably much more powerful. With the parametric polynomial the user just needs to specify some values at key points (also known as boundary conditions) in the airfoil such as leading edge radius (couldn't do this with the parametric cubic spline), after body angles, and top and bottom maximums. There are a few different boundary condition options, each with its own amount of conditions to generate the airfoil.

As we learned more about the parametric polynomial, we found an instability on high order polynomials. The parametric polynomial method was nearly scrapped, but Phil Barnes found a way around the problem. This involved having separate polynomials for the upper and lower halves of an airfoil.

1.2.3 Parametric Fourier
After the instability with the parametric polynomial was discovered, a new method was introduced to us. This method would be the most efficient of the ones studied so far, as it was much simpler and had more control over the airfoil compared to the other methods. The method involves using a Fourier series instead of a polynomials by taking in a number of values (eight: top point, bottom point, leading edge radius, half trailing edge, beta upper angle, beta lower angle, upper gamma, and lower gamma) to fine tune

the upper and lower halves of an airfoil. The resulting calculations would maintain continuity on all derivatives, an important characteristic in airfoil design.

1.2.4 Aircraft Generator Blender Addon
Last semester our addon had the ability to fully generate a wing and fuselage and supported full parametric cubic spline generation. It featured an easily navigable UI with editable geometric properties. It also featured the ability to import and export Excel files for the aircraft component points. Users were able to specify a set of points, and our application generated functions to visualize curves around those points. However, because of the limited capabilities of designing inputs for addons in Blender, inputting data was awkward and inefficient (mainly inputting a dynamic list of points).

1.3 Project Description
For this semester, our project's goal was to create a standalone application for wing generation. It features a much more easy-to-use interface as opposed to working through the limited addon GUI of the Blender software. Users are able to input data through a GUI application and generate the appropriate curves using parametric polynomials, parametric cubic splines, or the parametric Fourier. Users then have the option of exporting to Blender or saving the data to a text file should they desire.

1.4 Objectives
Our main objective was to create the *easy-to-use, intuitive,* and *efficient* standalone GUI application for wing generation and successfully integrate it with Blender. Our secondary objective was to learn and integrate new methods in designing an airfoil: the parametric polynomial and parametric Fourier.

2. System Design Overview

2.1 Implementation Languages and Libraries

As with last semester, we utilized Python as our programming language as it is the default language within Blender. This makes any interaction between our application and Blender easier to handle. Also returning from the implementation of last semester are the libraries for Blender (bpy) and Numpy. The bpy API allows us to do a number of transformations on the objects and scenes within Blender. Numpy is utilized mainly for matrix calculations, more specifically, solving a system of linear equations.

The first new library that we used this semester was the socket library. Python's socket library allows us to create sockets, giving us the ability for applications to communicate across ports. The main purpose of this library is to allow our standalone application to send data over to the Blender application to generate components.

The second library we've added to the mix is TkInter. TkInter is a graphical user interface API that allows for compatibility over multiple operating systems and is the default GUI API for Python. Because it is bundled with Python, we decided to use this to avoid any dependencies for a user.

This leads us to another set of modules/scripts we utilized this semester: cx_Freeze. To avoid any other dependencies in our program (e.g. numpy) we used cx_Freeze, a method that enabled us to "freeze" our program into an executable file so that the user does not need to install any other programs or libraries.

2.2 Connectivity with Blender

One of the functionalities of our application was to have some sort of connectivity to Blender. The ideal situation for this connectivity was to be able to edit a blend file in "real-time", so that the experience would be efficient and intuitive for a user. We had a couple of options to achieve this functionality, each with their own set of advantages and disadvantages.

Command Prompt
The first method our team found to accomplish this task was running Blender and a Python script through the command prompt. This method was easy to implement, but it came with some characteristics that may not be very user friendly. With this method, Blender would have to be closed for the Python script to generate a component in a .blend file. While Blender could be run in background mode, there was no way to edit a blend file in "real-time" in which the user could see the component being generated.

Open a Generated File from the Application
A second method would be to just have the application save a file with all the data needed to generate the parametric cubic spline. The user would then have to manually open the file in Blender. Obviously, this method would not have the characteristic of "real-time" editing, but it was the easiest to implement.

Client-Server Architecture
The final method, which we ended up using, was having some sort of client-server architecture that will allow for "real-time" editing of the blend file. We used Python sockets to achieve this architecture and allow the application to send data to Blender. However, within Blender a button still had to be pressed to begin the "communication" between the applications.

We implemented this by creating a socket in the Blender addon. This allows Blender to act as the "server", listening to message from the client (our external application), which has its own socket to send messages through. In Blender, we had the portion of the code that listened for data on a separate thread, so the Blender would still be functional. Otherwise, Blender would "freeze" as it waits for the server to finish listening for messages. This has a major drawback in that Blender doesn't cooperate with threads fully. Sometimes Blender will crash due to the threading. Also, threading meant that some of Blender's operators would be unavailable to be called upon through the child thread. This meant that we had to come up with a new way to generate the object compared to last semester, where we used tools inside Blender to generate objects (Bridge Edge Loops, XYZ Function Surfaces, etc.)

2.3 Object Generation Algorithm

Once we were able to send data between the applications, we needed to figure out what to do with this data to create an object in Blender. Because we couldn't use all of last semester's method to generate an object in Blender, the following algorithm was created:

1. Collect input data from the user based on the selected method
2. Begin calculations on the data for the appropriate method
3. These calculations generate a number of points, which are then sent to Blender through a socket
4. The Blender addon appends these points to a list to be used as vertex data for a new mesh
5. Faces or edges are generated and appended to list according to the type of component being generated
    a. For an Airfoil:
       - Edges are generated, connecting each vertex to the previous vertex and the last vertex to the first vertex
    b. For a Wing:
       - Faces are generated, where the first and second vertex of each adjacent airfoil are sent in as a face. Then the sides of the wing are filled with a face, with the vertexes that comprise an airfoil's edges.
6. The vertex and face list are sent to a Blender Operator, which creates a new mesh with the data (*mesh.from_pydata*)

2.4 Data Specifications

With a standalone application, the team wanted to look into some form of file type that we could use instead of the CSV files from last semester. We implemented our application to utilize a plain text file.

The files would be structured with the name of the geometric generation method first, with subsequent lines followed by the settings that go with that method. For example, for the parametric polynomial boundary condition 1:

Parametric Polynomial Boundary Condition 1
Leading Edge
Upper Trailing Edge
Lower Trailing Edge
Radius
Beta Upper Angle
Beta Lower Angle
Gamma

2.5 System Functions Performed

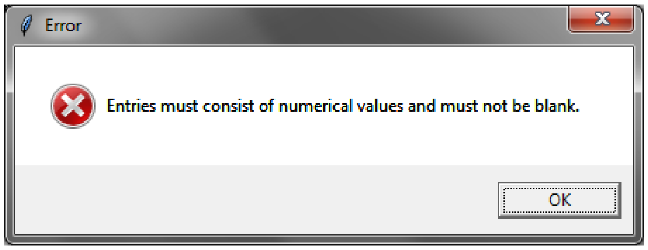
External Application	
Name	**Description**
Apply Settings	Applies the data input by the user and draws the new airfoil in the profile plot
Export to Blender	Takes the data generated by the user and sends information to Blender to create a new object/mesh
New	Clears any data in the generation methods
Open	Opens a .txt file and inserts the data into the appropriate entry fields according to the generation method detailed
Save	Saves the data for a generation method into a .txt file
Save As	Saves the data for a generation method into a .txt file
Change Plot Orientation	Changes the orientation of the profile plot to the left or right side of the application
Help	Opens up a help dialog box with general information on each method
About	Opens up a dialog box with general information on the project
Add Point	Prompts the user to enter a point, then inserts that data into the list-box for the parametric cubic spline
Delete Point	Deletes the selected point, removing it from the list box
Add Station	Prompts the user to enter data for a new station, then inserts that data into the appropriate lists in the application
Edit Station	Retrieves data for the selected station and inserts it into the entry fields, allowing a user to edit data for a station

Delete Station	Deletes the selected station, removing its data from the appropriate lists
Plot Selected Station	Plots the selected station on the profile plot

Blender Addon	
Name	**Description**
Link to Aircraft Generator	Starts the communication between the external application and Blender by creating and listening on a socket
Unlink Aircraft Generator	Stops the communication between the external application and Blender by closing the socket

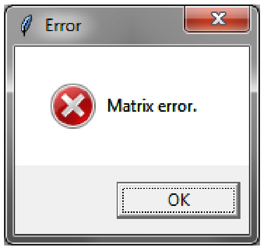
2.6 Error/Failure Detection

Detecting a failure or error produces a dialog box prompting the user to fix the problem. This is done with the TkInter library. We have error checks for empty inputs, non-numerical inputs, and matrix errors. For the parametric cubic spline and 3D parametric Fourier methods, we also have error checks for list-box and drop-down menu selections. All of these errors prompt a dialog box and prevent the user from prematurely applying the settings or exporting to Blender.



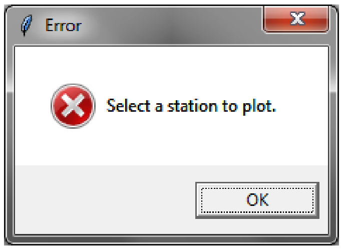
Entries must consist of numerical values and must not be blank.

Example Error Dialog 1



Matrix error.

Example Error Dialog 2



Select a station to plot.

Example Error Dialog 3

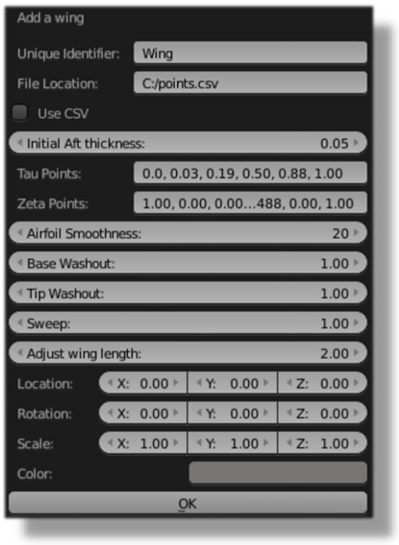
2.7 Portability

Our application can be run straight through Python or through a compiled executable file. This means that our application can be run on any computer capable of using Python. It does not need Blender to design airfoils, however, Blender is needed for 3D models of wings as well as for creating .stl files for 3D printing.

3. User Interface Design

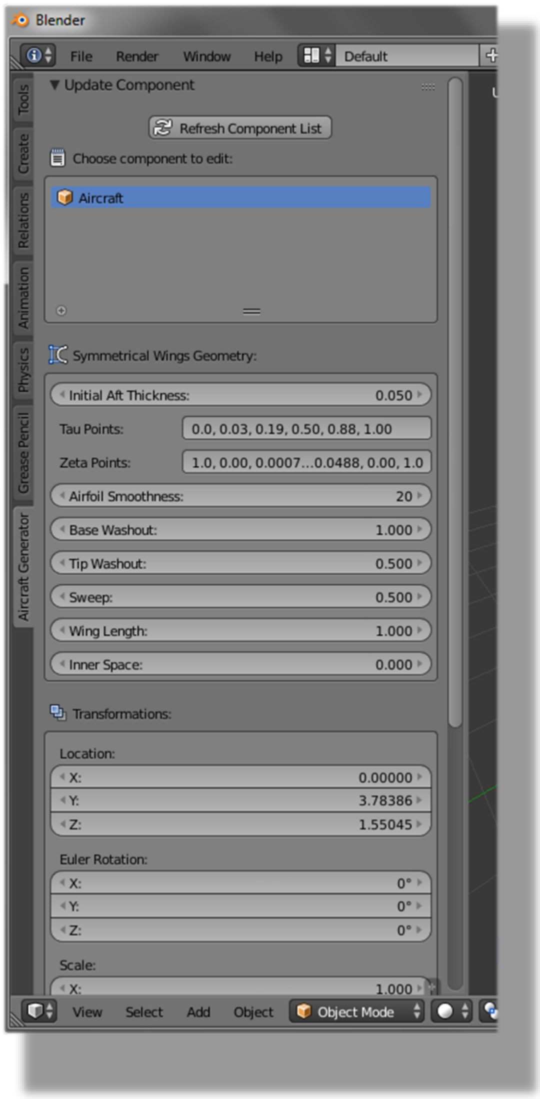
3.1 Previous Designs

The following are screenshots from the previous addon design. Note that Tau and Zeta Points require data to be put on one line separated by commas. The user interface is clunky and hard to navigate. While this addon did have the convenience of being directly in Blender, it was not only inefficient but also very prone to errors. It would have become even more inefficient and hard to navigate as we added more generation methods (only parametric cubic spline was available in this addon). Another problem was that error dialog boxes were not effective. Simply moving the mouse away from the error box would cause it to disappear (again, due to the lack of control over Blender's inferface).


Add a wing

Unique Identifier:	Wing
File Location:	C:/points.csv
Use CSV	
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, 0.00...488, 0.00, 1.00
Airfoil Smoothness:	20
Base Washout:	1.00
Tip Washout:	1.00
Sweep:	1.00
Adjust wing length:	2.00
Location:	X: 0.00 Y: 0.00 Z: 0.00
Rotation:	X: 0.00 Y: 0.00 Z: 0.00
Scale:	X: 1.00 Y: 1.00 Z: 1.00
Color:	

Inputs in Previous Addon



Blender

File Render Window Help Default

Update Component

Refresh Component List

Choose component to edit:

Aircraft

Symmetrical Wings Geometry:

Initial Aft Thickness: 0.050

Tau Points: 0.0, 0.03, 0.19, 0.50, 0.88, 1.00

Zeta Points: 1.0, 0.00, 0.0007...0.0488, 0.00, 1.0

Airfoil Smoothness: 20

Base Washout: 1.000

Tip Washout: 0.500

Sweep: 0.500

Wing Length: 1.000

Inner Space: 0.000

Transformations:

Location:

X: 0.00000

Y: 3.78386

Z: 1.55045

Euler Rotation:

X: 0°

Y: 0°

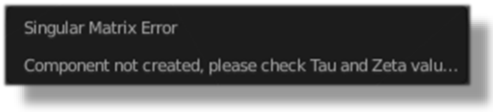
Z: 0°

Scale:

X: 1.000

View Select Add Object Object Mode

Inputs in Previous Addon



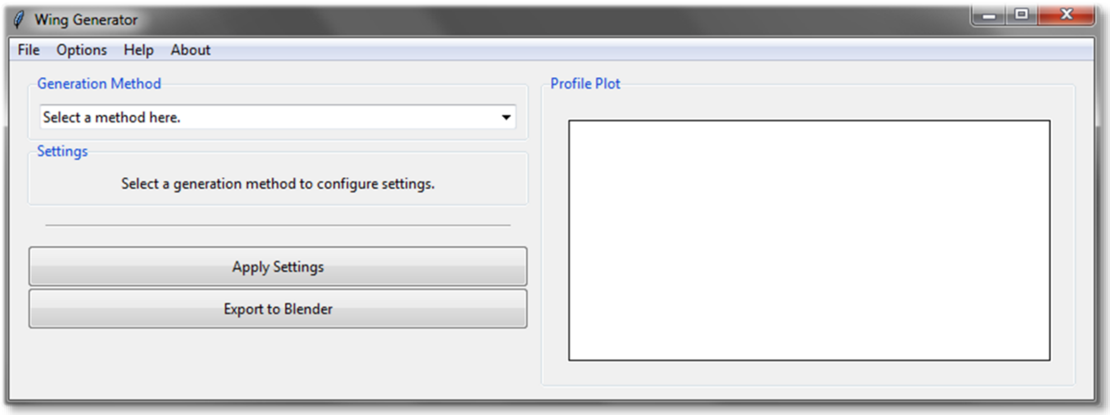
Singular Matrix Error

Component not created, please check Tau and Zeta valu...

Error dialog message when the points generate a singular matrix

3.2 User Input Preview

For the new User Interface, we wanted it to be easy to use as well as efficient for a user. In the Blender addon we had little control over the look and feel of the interface. By making our own application, we have much more power over which components go where.

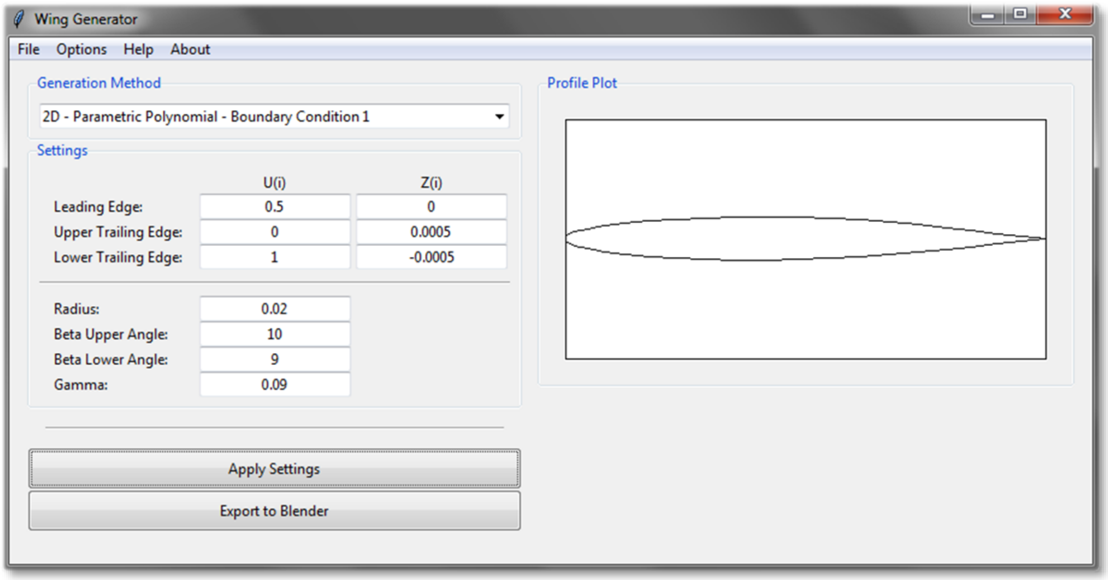


Select a method here.

Select a generation method to configure settings.

Initial GUI

The following screenshots are of the standalone application. Each screenshot has a set of input data filled in with the resulting plot shown (as if the apply settings button was pushed).

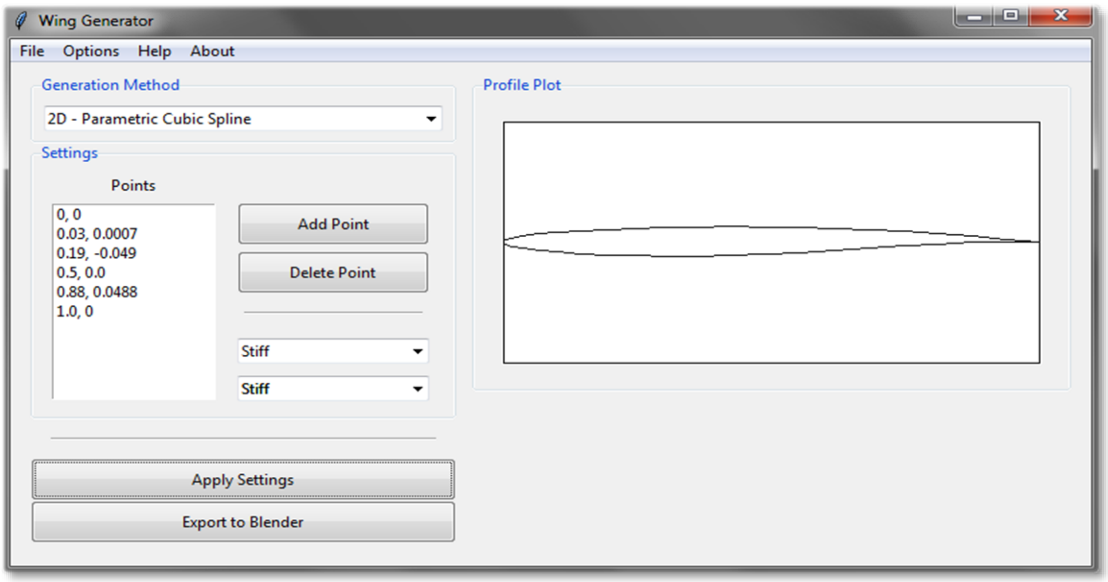


2D - Parametric Polynomial - Boundary Condition 1

	U(i)	Z(i)
Leading Edge:	0.5	0
Upper Trailing Edge:	0	0.0005
Lower Trailing Edge:	1	-0.0005

Radius:	0.02
Beta Upper Angle:	10
Beta Lower Angle:	9
Gamma:	0.09

Input Preview: Parametric Polynomial – Boundary Condition 1 and Resulting Plot



Generation Method: 2D - Parametric Cubic Spline

Settings

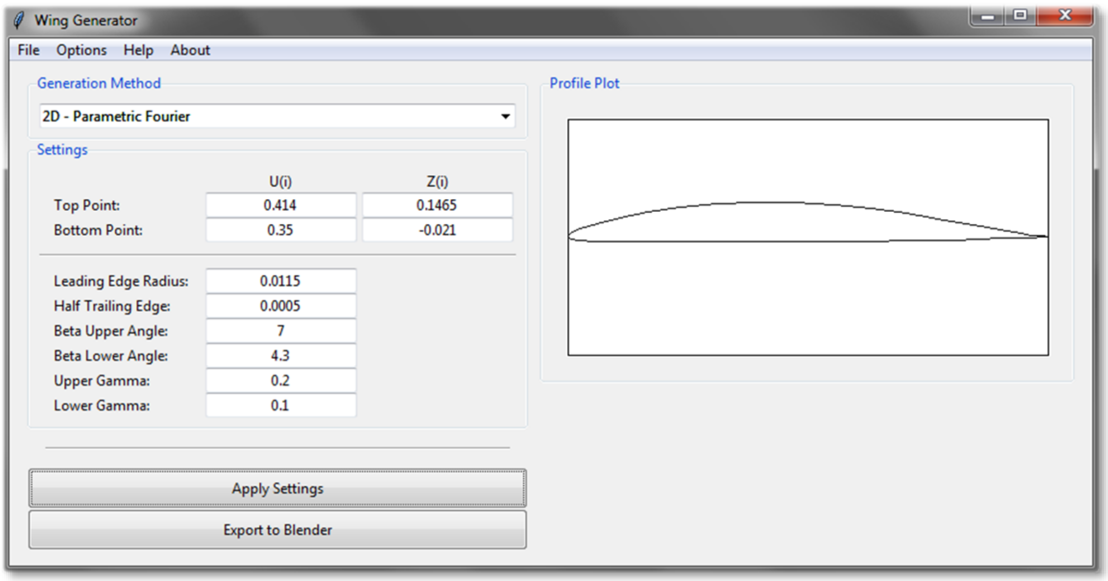
Points

0, 0
0.03, 0.0007
0.19, -0.049
0.5, 0.0
0.88, 0.0488
1.0, 0

Stiff

Stiff

Input Preview: Parametric Cubic Spline and Resulting Plot



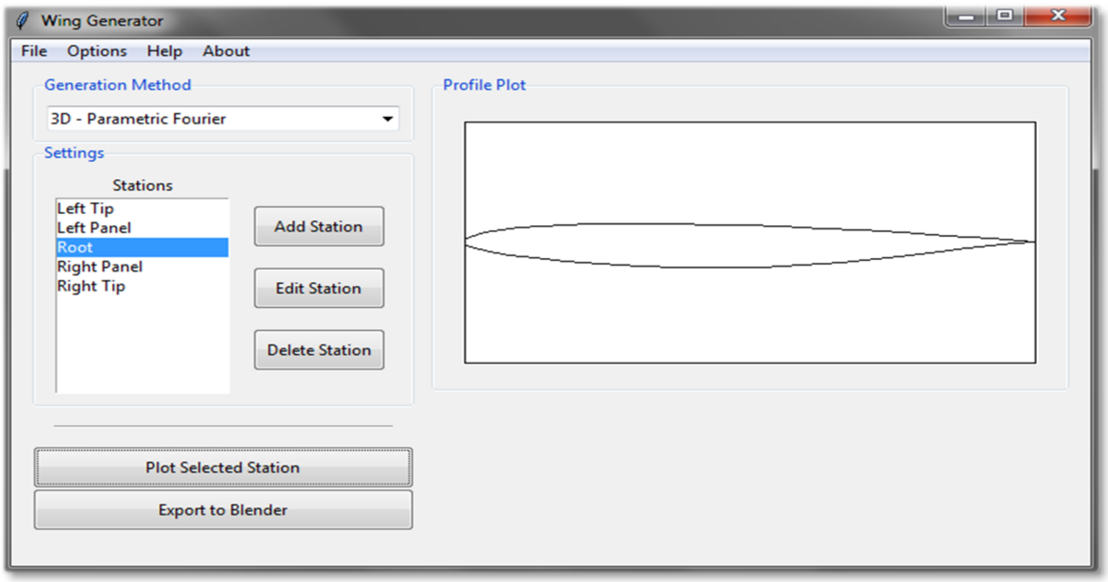
Generation Method: 2D - Parametric Fourier

Settings

	U(i)	Z(i)
Top Point:	0.414	0.1465
Bottom Point:	0.35	-0.021

Leading Edge Radius:	0.0115
Half Trailing Edge:	0.0005
Beta Upper Angle:	7
Beta Lower Angle:	4.3
Upper Gamma:	0.2
Lower Gamma:	0.1

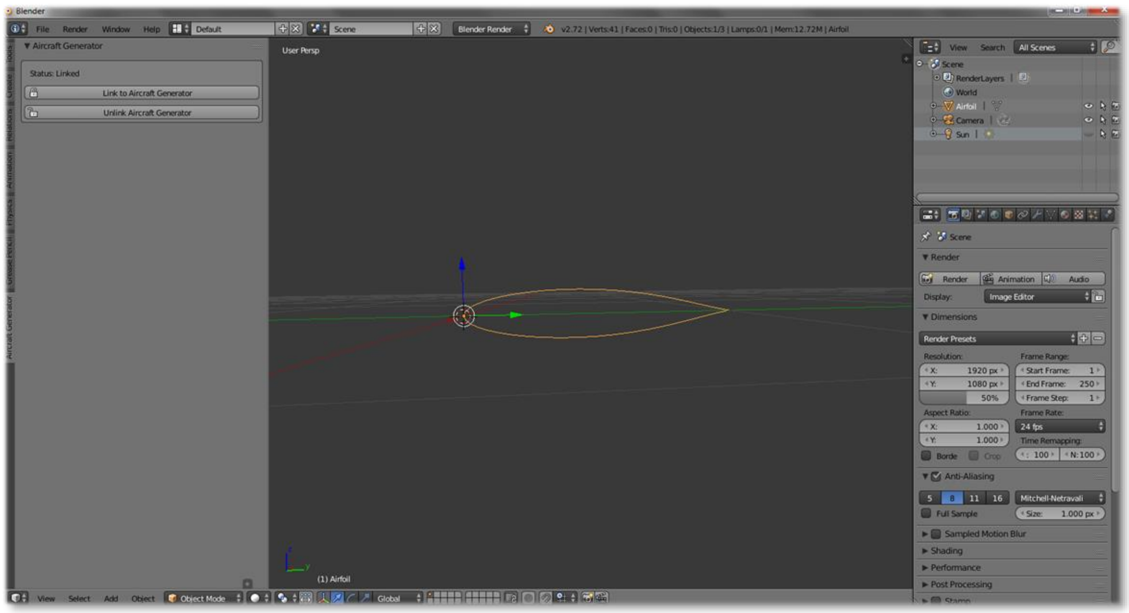
Input Preview: Parametric Fourier and Resulting Plot



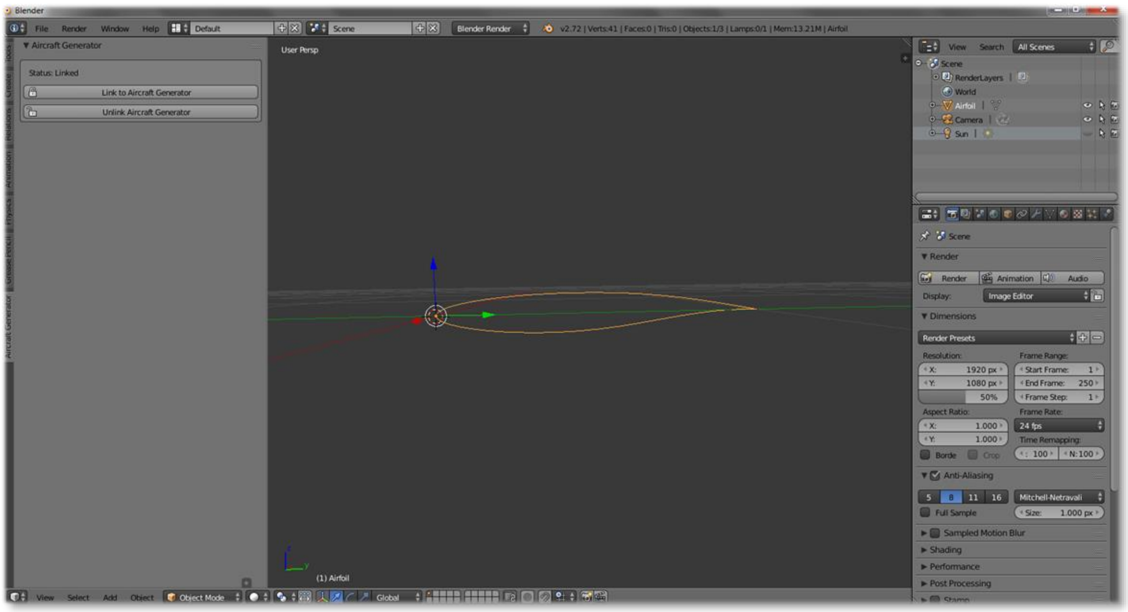
Input Preview: Parametric Fourier (3D) and Resulting Plot

3.3 User Output Preview

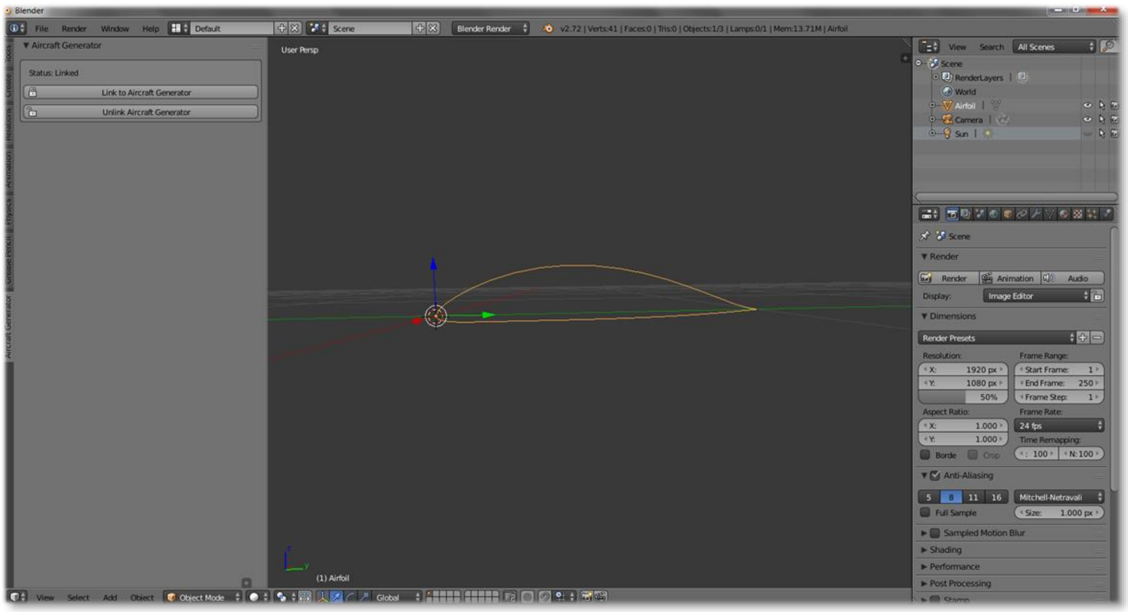
The following screenshots are of the Blender application after linking to our external application and exporting the component.



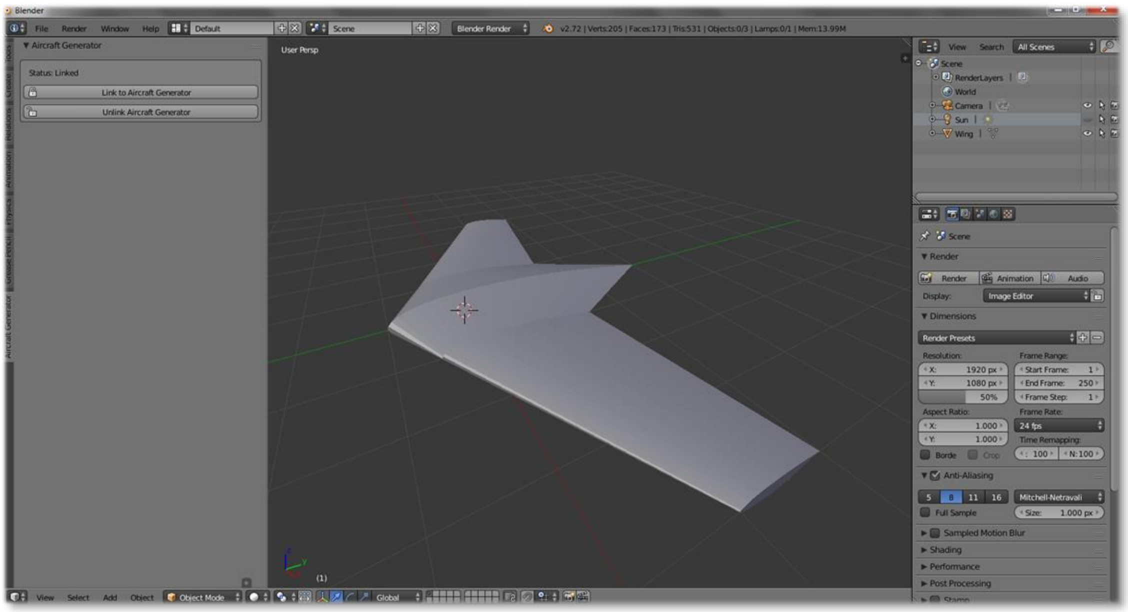
Output Preview: Parametric Polynomial Boundary Condition 1


Aircraft Generator
Status: Linked
Link to Aircraft Generator
Unlink Aircraft Generator
User Persp
(1) Airfoil
v2.72 | Verts:41 | Faces:0 | Tris:0 | Objects:1/3 | Lamps:0/1 | Mem:13.21M | Airfoil

Output Preview: Parametric Cubic Spline


Aircraft Generator
Status: Linked
Link to Aircraft Generator
Unlink Aircraft Generator
User Persp
(1) Airfoil
v2.72 | Verts:41 | Faces:0 | Tris:0 | Objects:1/3 | Lamps:0/1 | Mem:13.71M | Airfoil

Output Preview: Parametric Fourier



v2.72 | Verts:205 | Faces:173 | Tris:531 | Objects:0/3 | Lamps:0/1 | Mem:13.99M

Status: Linked

Display: Image Editor

Resolution: X: 1920 px, Y: 1080 px, 50%

Frame Range: Start Frame: 1, End Frame: 250, Frame Step: 1

Aspect Ratio: X: 1.000, Y: 1.000

Frame Rate: 24 fps

Time Remapping: 100, N:100

Borde, Crop

Anti-Aliasing: 5 [✓ 8] 11 16, Mitchell-Netravali

Full Sample, Size: 1.000 px

Output Preview: Parametric Fourier (3D)

4. Conclusion

4.1 Summary

Our objective this semester was to create a standalone application for wing generation that interacts with Blender. We have successfully accomplished our goal and produced an easy-to-use and user-friendly application. Our application features a generation method selection menu, open/save .txt file support, a drawn plot preview, and exportation to Blender. We successfully implemented two new generation methods compared to last semester's one. The current generation methods supported are: parametric polynomial (with three boundary conditions), parametric cubic spline, and parametric Fourier (2D and 3D).

4.2 Problems Encountered and Solved

The first major problem we encountered starting out this project was the decision of using PyQT versus TkInter for developing the user interface. Both are popular GUI toolkits with their own pros and cons. The main issue we had was selecting the option without licensing conflicts. PyQT required license purchasing while TkInter is the go-to GUI toolkit for Python. Another issue was with the capabilities of each toolkit. PyQT had more capabilities compared to TkInter. Additionally, many people preferred PyQT over TkInter due to TkInter's lack of modern style. For the purpose of our project, we felt that the capabilities of TkInter were just enough to get the job done.

Another major problem we had initially in this project was discovering parametric polynomial instability. We found that the parametric polynomial airfoil geometry method of order 9 was only stable for a small interval of geometries. Outside of that workable range, it "broke" and created undesired curves. We couldn't work with an order of 7 because that wouldn't provide sufficient control of the curve path, even though it does eliminate unwanted oscillations in the shape. We resolved this stability problem by using separate top and bottom polynomials, each with an order of 5.

Connectivity to Blender posed another problem for us. We had to find a way to incorporate our application with Blender to allow the user to edit a blend file in "real-time." We planned to do this by one of three ways: using a command prompt, manually opening a generated file from the application, or implementing a client-server architecture. We ended up implementing a client-server architecture between our application and Blender utilizing Python's socket library.

4.3 Suggestions for Better Approaches

A better approach to this project would have been to thoroughly learn the different generation methods and have a plan ready to implement. We had a delay in our start because of a discovered instability in the parametric polynomial. Had this problem been known prior to our project, we would not have had a problem during implementation and coding.

4.4 Suggestions for Future Extension to Project

Prospective plans for this project should include extending the 3D generation further. The 3D Parametric Fourier is quite limited and could have a lot more control over the shape of the generated wing. Additionally, plans should be made for the rest of the aircraft components such as fuselage, engine, tail, etc., ultimately generating a complete aircraft.

5. Source Code

5.1 Blender Aircraft Generator Module

```python
bl_info = {
    "name": "Aircraft Generator",
    "author": "Marvin Trajano, Lisa Tran, Victor Tran",
    "version": (1, 0),
    "blender": (2, 70, 0),
    "location": "View3D > Left panel ",
    "description": "Links Blender to our external Aircraft Generator application"
}

import bpy
from bpy.props import *
import threading
import socket
import math
import numpy as np

#Global variables
socketThread = 0
s = 0
listenForData = False
unlink = False

#   createThread
#   - Creates the socket server where listening occurs in a new thread, this prevents Blender
from stalling in a loop
def createThread():
    global socketThread, listenForData
    socketThread = threading.Thread(name='socketThread', target=listenToSocket)
    listenForData = True
    createSocketServer()
    socketThread.start()

#   listenToSocket
#   - Checks the listenForData flag, collecting any data received from the application and
parsing it
def listenToSocket():
    global listenForData, s
    while listenForData:
        try:
            data, address = (s.recvfrom(4096))
            message = data.decode('utf-8')
            message = message.split(',')
            type = message.pop()
            if(type == "0"):
                createAirfoil(message, "Airfoil", [0,0,0])
            else:
                createWing(message, "Wing")
        except:
            pass

#   createAirfoil
#   - Uses the message received from the external application (vertices) to create an airfoil
(2D) Blender Mesh Object
def createAirfoil(message, name, location):
    print("Creating airfoil...")
    #Mesh arrays
    verts = []
    edges = []
    faces = []
    e1 = []
    e2 = []
    #Mesh variables
    num = len(message)
    for i in range(0, num):
        if(i%2 == 0):
            e1.append(message[i])
        else:
            e2.append(message[i])
    #Fill vertices array
    for i in range(0, int(num/2)):
        x = 0
        y = float(e1[i])
        z = float(e2[i])
        vert = (x,y,z)
        verts.append(vert)
        if(i == int(num/2)-1):
            edges.append((i, 0))
        else:
            edges.append((i, i+1))
    #Create mesh and object
    mesh = bpy.data.meshes.new(name)
    object = bpy.data.objects.new(name, mesh)
    #Set mesh location
    object.location = location
    bpy.context.scene.objects.link(object)
    #Create mesh from python data
    mesh.from_pydata(verts, edges, faces)
    mesh.update(calc_edges=True)

#   createAirfoil
#   - Uses the message received from the external application (vertices) to create a wing (3D)
Blender Mesh Object
def createWing(message, wingName):
    print("Creating wing...")
    #Mesh arrays
    verts = []
    faces = []
    ug = []
    lg = []
    ler = []
    topPointU = []
    topPointZ = []
    bottomPointU = []
    bottomPointZ = []
    bu = []
    bl = []
    hte = []
    sparX = []
    sparY = []
    sparZ = []
    scale = []
    for i in range(0, len(message), 14):
        ug.append(message[i])
        lg.append(message[i+1])
        ler.append(message[i+2])
        topPointU.append(message[i+3])
        topPointZ.append(message[i+4])
        bottomPointU.append(message[i+5])
        bottomPointZ.append(message[i+6])
        bu.append(message[i+7])
        bl.append(message[i+8])
        hte.append(message[i+9])
        sparX.append(message[i+10])
        sparY.append(message[i+11])
        sparZ.append(message[i+12])
        scale.append(message[i+13])
    data = ""
    for j in range(0, int(len(message)/14)):
        X = parametricFourier(float(ug[j]), float(lg[j]),
            float(ler[j]), float(topPointU[j]), float(topPointZ[j]),
            float(bottomPointU[j]), float(bottomPointZ[j]), float(bu[j]),
            float(bl[j]), float(hte[j]))
        n = 8
        gam = gam_(0, float(ug[j]), float(lg[j]))
        ww = 1 - 2 * 0
        previousX = X_of_W(gam, ww)
        previousY = float(hte[j]) * (1 - 2 * 0)
        data = data + str(previousX) + "," + str(previousY) + ","
        for i in range(1, 41):
            u = i/40
            gam = gam_(u, float(ug[j]), float(lg[j]))
            if(u < 0.5):
                ww = 1 - 2 * u
            else:
                ww = 2 * u - 1
            xx = X_of_W(gam, ww)
            zz = float(hte[j]) * (1 - 2 * u)
            for k in range(0, n):
                remain = 1
                zz = zz + X[k] * math.sin(remain * math.pi * (k+1) * u)
            data = data + str(xx) + "," + str(zz) + ","
    data = data.split(',')
    data.pop()
    airfoils = 0
    for k in range(0, len(data), 82):
        airfoils = airfoils + 1
    vCount = 0
    locIndex = 0
    for i in range(0, len(data), 2):
        x = float(sparX[locIndex])
        y = float(data[i]) * float(scale[locIndex]) + float(sparY[locIndex])
        z = float(data[i+1]) * float(scale[locIndex]) + float(sparZ[locIndex])
        vert = (x,y,z)
        verts.append(vert)
        vCount = vCount + 1
        if(vCount % 41 == 0):
            locIndex = locIndex + 1
    #Side Faces
    for i in range(0, airfoils):
        faces.append((41*i+0, 41*i+1, 41*i+2, 41*i+3, 41*i+4, 41*i+5, 41*i+6, 41*i+7, 41*i+8,
41*i+9, 41*i+10,
                      41*i+11, 41*i+12, 41*i+13, 41*i+14, 41*i+15, 41*i+16, 41*i+17, 41*i+18,
41*i+19, 41*i+20,
                      41*i+21, 41*i+22, 41*i+23, 41*i+24, 41*i+25, 41*i+26, 41*i+27, 41*i+28,
41*i+29, 41*i+30,
                      41*i+31, 41*i+32, 41*i+33, 41*i+34, 41*i+35, 41*i+36, 41*i+37, 41*i+38,
41*i+39, 41*i+40))
    vCount = 0
    for k in range(0, airfoils-1):
        for i in range(0, 41):
            #Wing panel faces
            face1 = (vCount, vCount+40, vCount+41, vCount+1)
            faces.append(face1)
            #Last panel face
            if(vCount % 41 == 0):
                lastFace = (vCount, vCount+41, vCount+81, vCount + 40)
                faces.append(lastFace)
            vCount = vCount + 1
    #Create mesh and object
    mesh = bpy.data.meshes.new(wingName)
    object = bpy.data.objects.new(wingName,mesh)
    #Set mesh location
    object.location = [0,0,0]
    bpy.context.scene.objects.link(object)
    #Create mesh from python data
    mesh.from_pydata(verts,[],faces)
    mesh.update(calc_edges=True)

#   createSocketServer
#   - Creates a socket server to be listened to for data from the external application
def createSocketServer():
    global s
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) #Allows for reuse of the port if
Blender crashes
```

```
    s.bind(('127.0.0.1', 7777))
    print("Server started")

#   parametricFourier
#   - Used to create 3D wings, takes in Fourier values to interpolate points
def parametricFourier(g_t, g_b, r, x_t, z_t, x_b, z_b, b_t, b_b, z):
    n = 8
    FF = np.matrix(np.zeros((n, n)))
    BB = np.array(np.zeros(n))
    for i in range(1,4):
        if(i == 1):
            wt = W_of_X(g_t, x_t)
            wb = W_of_X(g_b, x_b)
        else:
            ut = 0.5 * (1 - wt)
            gam = gam_(ut, g_t, g_b)
            wt = W_of_X(gam, x_t)
            ub = 0.5 * (1 + wb)
            gam = gam_(ub, g_t, g_b)
            wb = W_of_X(gam, x_b)
    gama = 0.5 * (g_t + g_b)
    for i in range(0, n):
        if(i == 0):
            uu = 0
            ww = 1
            zz = 0 + z
            dZdU = math.pi * math.tan(b_t * (math.pi / 180) * (1 - 4 * g_t))
        elif(i == 1 or i == 2):
            uu = ut
            ww = wt
            zz = z_t
            dZdU = 0
        elif(i == 3 or i == 4):
            uu = 0.5
            ww = 0
            zz = 0
            dZdU = -1 * math.sqrt(r * (1+8*gama) * math.pi**2)
        elif(i == 5 or i == 6):
            uu = ub
            ww = wb
            zz = z_b
            dZdU = 0
        elif(i == 7):
            uu = 1
            ww = 1
            zz = 0 - z
            dZdU = math.pi * math.tan(b_b * (math.pi/180)) * (1 - 4 * g_b)
        gam = gam_(uu, g_t, g_b)
        xx = X_of_W(gam, ww)
        if(i != 1 and i != 4 and i != 5):
            BB[i] = dZdU + 2 * z
        else:
            BB[i] = zz - z * (1 - 2 * uu)

        for j in range(0, n):
            remain = 1
            if(i != 1 and i != 4 and i != 5):
                FF[i,j] = math.pi * (j+1) * math.cos(remain * math.pi * (j+1) * uu)
            else:
                FF[i,j] = math.sin(remain * math.pi * (j+1) * uu)
    PP = np.linalg.solve(FF,BB)
    return PP

#Used in Parametric Fourier
def W_of_X(gamma, xx):
    expo = 0.56
    xe = xx ** expo
    w_of_x = xe - 1.3 * gamma * math.sin(math.pi * xe ** 2)
    for i in range(1,5):
        x1 = X_of_W(gamma, w_of_x)
        xe1 = x1 ** expo
        w_of_x = w_of_x + xe - xe1
    return w_of_x

#Used in Parametric Fourier
def X_of_W(gamma, ww):
    x_of_w = 1 - (1 - gamma) * math.cos(ww * math.pi / 2) - gamma * math.cos(ww * 3 * math.pi /
2)
    return x_of_w

#Used in Parametric Fourier
def gam_(uu, g_t, g_b):
    gam = g_b + (g_t - g_b) * (math.cos(uu * (math.pi / 2))) ** 2
    return gam

#   Main Class
#   - Handles the starting and closing of the socket server thread
#   - Registers the operator with Blender, so that it can be used and appears in the left sidebar
class aircraft_generator(bpy.types.Operator):
    bl_idname = "aircraft_generator.modal"
    bl_label = "Link to Aircraft Generator"

    def modal(self, context, event):
        global listenForData, s, unlink
        result = {'PASS_THROUGH'}
        if unlink:
            listenForData = False
            s.close()
            print("Server closed")
            socketThread.join()
            del s
            unlink = False
            result = {'CANCELLED'}
            self.report({'WARNING'}, "Aircraft Generator has been unlinked.")
        return result

    def invoke(self, context, event):
        global listenForData
        if listenForData == False:
            context.window_manager.event_timer_add(0.01, context.window)
            context.window_manager.modal_handler_add(self)
            createThread()
            return {'RUNNING_MODAL'}
        else:
            self.report({'WARNING'}, "Aircraft Generator is already linked.")
            return {'CANCELLED'}

#   Unlink Aircraft Generator
#   - Class to register an unlink button within Blender, this unlinks the connection between
Blender and the external
#     application, closing the socket server and joining the thread
class unlinkAircraftGenerator(bpy.types.Operator):
    """Add a simple box mesh"""
    bl_idname = "aircraft_generator.close"
    bl_label = "Unlink Aircraft Generator"
    bl_options = {'REGISTER', 'UNDO'}

    def execute(self, context):
        global unlink
        unlink = True
        return {'FINISHED'}

#   Aircraft Generator Panel
#   - This class sets up the UI in the left sidebar, linking the previous operators to buttons
class aircraftGeneratorPanel(bpy.types.Panel):
    bl_label = "Aircraft Generator"
    bl_space_type = "VIEW_3D"
    bl_region_type = "TOOLS"
    bl_category = 'Aircraft Generator'

    def draw(self, context):
        global listenForData
        layout = self.layout
        box = layout.box()
        if listenForData:
            box.label(text="Status: Linked")
        else:
            box.label(text="Status: Not Linked")
        box.operator("aircraft_generator.modal", icon='LOCKED')
        box.operator("aircraft_generator.close", icon='UNLOCKED')

def register():
    bpy.utils.register_module(__name__)

def unregister():
    bpy.utils.unregister_module(__name__)

if __name__ == "__main__":
    register()
```

5.2 Standalone Aircraft Generator Application

```
#   CSULB - Northrop Grumman Student Design Project
#   Standalone GUI Application
#   Written by Marvin Trajano, Lisa Tran, Victor Tran
#   Spring 2015

from tkinter import *
from tkinter import ttk
from tkinter import filedialog
import math
import tkinter.messagebox
import ParametricPolynomial as pp
import ParametricCubicSpline as pcs
import ParametricFourier as pf
import socket

#   Main Class
#   - Creates main window frame as well as sets up all the panels and components
class AircraftGenerator(ttk.Frame):
    def __init__(self, parent, isapp=True, name='aircraftgenerator'):
        main = ttk.Frame.__init__(self, name=name, borderwidth=10)
        self.parent = parent
        self.pack(expand=Y, fill=BOTH)
        self.master.title('Wing Generator')
        self.initUI()
        self.grid_columnconfigure(0, weight=1)
        self.centerWindow()
        #Exit check
        parent.protocol("WM_DELETE_WINDOW", self.exitCheck)

    #   initUI
    #   - Initializes the user interface, creating the panels and adds all components to the
panels
    def initUI(self):
        self.tk.call('tk', 'windowingsystem')
        self.option_add("*tearOff", FALSE)
        menubar = Menu(self.parent)
        menu_file = Menu(menubar)
        menubar.add_cascade(menu=menu_file, label='File')
        menu_file.add_command(label='New', command=self.newFile)
        menu_file.add_command(label='Open', command=self.openFile)
        menu_file.add_command(label='Save', command=self.saveFile)
        menu_file.add_command(label='Save As', command=self.saveFileAs)
        menu_file.add_separator()
        menu_file.add_command(label='Export to Blender', command=self.export)
        menu_options = Menu(menubar)
        menubar.add_cascade(menu=menu_options, label='Options')
        menu_options.add_command(label='Change plot orientation', command=self.plotOrientation)
        menu_help = Menu(menubar)
        menubar.add_command(label='Help', command=self.help)
        menubar.add_command(label="About", command=self.about)

        #Station Data
        self.name = []
        self.topPointU = []
        self.topPointZ = []
        self.bottomPointU = []
        self.bottomPointZ = []
        self.ler = []
        self.hte = []
        self.bu = []
        self.bl = []
        self.ug = []
        self.lg = []
        self.sparX = []
        self.sparY = []
        self.sparZ = []
        self.scale = []

        #Main Panel
        self.parent.config(menu=menubar)
        s = ttk.Style()
        s.configure("TButton", padding=6, relief="flat")
        self.inputsF = ttk.Frame(self)
        self.inputsF.pack(side="left", fill="both", expand=True, padx=5)
        self.rowconfigure(1, pad=20)
        self.methodLF = ttk.LabelFrame(self, text='Generation Method', borderwidth=10)
        self.methodLF.grid(in_=self.inputsF, column=0, row=0, sticky=W+E)
        self.methodLF.grid_columnconfigure(0, weight=1)
        self.methodvalue=StringVar()
        self.method = ttk.Combobox(self, textvariable=self.methodvalue, state='readonly')
        self.method.grid(in_=self.methodLF, sticky=W+E)
        self.method['values'] = ('Select a method here.', '2D - Parametric Polynomial - Boundary
Condition 1',
                                 '2D - Parametric Polynomial - Boundary Condition 2',
                                 '2D - Parametric Polynomial - Boundary Condition 3',
                                 '2D - Parametric Cubic Spline',
                                 '2D - Parametric Fourier',
                                 '3D - Parametric Fourier')
        self.method.current(0)
        self.method['values'] = ('2D - Parametric Polynomial - Boundary Condition 1',
                                 '2D - Parametric Polynomial - Boundary Condition 2',
                                 '2D - Parametric Polynomial - Boundary Condition 3',
                                 '2D - Parametric Cubic Spline',
                                 '2D - Parametric Fourier',
                                 '3D - Parametric Fourier')
        self.method.bind("<<ComboboxSelected>>", self.newsettings)
        self.plotLF = ttk.LabelFrame(self, text='Profile Plot', padding=20)
        self.plotLF.pack(side="top", padx=5)
        self.plot = Canvas(self, width=400, height=200, bg='white', highlightthickness=0,
borderwidth=1)
        self.plot.grid(in_=self.plotLF, column=0, row=2)
        self.plot.create_rectangle(1, 1, 400, 200)
        self.settingsLF0 = ttk.LabelFrame(self, text='Settings', borderwidth=10)
        self.settingsLF0.grid(in_=self.inputsF,column=0, row=1, sticky=W+E)
        self.settingsLF0.grid_columnconfigure(0, weight=1, minsize=396)
        placeholder1 = ttk.Label(self, text='Select a generation method to configure settings.',
anchor='center')
        placeholder1.grid(in_=self.settingsLF0, column=0, row=0, sticky=W+E)

        #Parametric Polynomial Boundary Condition 1 Panel
        self.settingsLF1 = ttk.LabelFrame(self, text='Settings', borderwidth=10)
        self.b1_label_u = ttk.Label(self, text='U(i)')
        self.b1_label_u.grid(row=0, column=1, in_=self.settingsLF1)
        self.b1_label_z = ttk.Label(self, text='Z(i)')
        self.b1_label_z.grid(row=0, column=2, in_=self.settingsLF1)
        self.b1_label_e = ttk.Label(self, text='Leading Edge:')
        self.b1_label_e.grid(row=1, column=0, in_=self.settingsLF1, sticky=W, padx=10)
        self.b1_entry_e_u = ttk.Entry(self, justify='center')
        self.b1_entry_e_u.grid(in_=self.settingsLF1, row=1, column=1, padx=2)
        self.b1_entry_e_z = ttk.Entry(self, justify='center')
        self.b1_entry_e_z.grid(in_=self.settingsLF1, row=1, column=2, padx=2)
        self.b1_label_0 = ttk.Label(self, text='Upper Trailing Edge:')
        self.b1_label_0.grid(row=2, column=0, in_=self.settingsLF1, sticky=W, padx=10)
        self.b1_entry_0_u = ttk.Entry(self, justify='center')
        self.b1_entry_0_u.grid(in_=self.settingsLF1, row=2, column=1, padx=2)
        self.b1_entry_0_z = ttk.Entry(self, justify='center')
        self.b1_entry_0_z.grid(in_=self.settingsLF1, row=2, column=2, padx=2)
        self.b1_label_1 = ttk.Label(self, text='Lower Trailing Edge:')
        self.b1_label_1.grid(row=3, column=0, in_=self.settingsLF1, sticky=W, padx=10)
        self.b1_entry_1_u = ttk.Entry(self, justify='center')
        self.b1_entry_1_u.grid(in_=self.settingsLF1, row=3, column=1, padx=2)
        self.b1_entry_1_z = ttk.Entry(self, justify='center')
        self.b1_entry_1_z.grid(in_=self.settingsLF1, row=3, column=2, padx=2)
        ttk.Separator(self.settingsLF1, orient=HORIZONTAL).grid(columnspan=3, sticky=W+E,
pady=10)
        self.b1_label_r = ttk.Label(self, text='Radius:')
        self.b1_label_r.grid(row=5, column=0, in_=self.settingsLF1, sticky=W, padx=10)
        self.b1_entry_r = ttk.Entry(self, justify='center')
        self.b1_entry_r.grid(in_=self.settingsLF1, row=5, column=1, padx=2)
        self.b1_label_b0 = ttk.Label(self, text='Beta Upper Angle:')
        self.b1_label_b0.grid(row=6, column=0, in_=self.settingsLF1, sticky=W, padx=10)
        self.b1_entry_b0 = ttk.Entry(self, justify='center')
        self.b1_entry_b0.grid(in_=self.settingsLF1, row=6, column=1, padx=2)
        self.b1_label_b1 = ttk.Label(self, text='Beta Lower Angle:')
        self.b1_label_b1.grid(row=7, column=0, in_=self.settingsLF1, sticky=W, padx=10)
        self.b1_entry_b1 = ttk.Entry(self, justify='center')
        self.b1_entry_b1.grid(in_=self.settingsLF1, row=7, column=1, padx=2)
        self.b1_label_g = ttk.Label(self, text='Gamma:')
        self.b1_label_g.grid(row=8, column=0, in_=self.settingsLF1, sticky=W, padx=10)
        self.b1_entry_g = ttk.Entry(self, justify='center')
        self.b1_entry_g.grid(in_=self.settingsLF1, row=8, column=1, padx=2)

        #Parametric Polynomial Boundary Condition 2 Panel
        self.settingsLF2 = ttk.LabelFrame(self, text='Settings', borderwidth=10)
        self.b2_label_u = ttk.Label(self, text='U(i)')
        self.b2_label_u.grid(row=0, column=1, in_=self.settingsLF2)
        self.b2_label_z = ttk.Label(self, text='Z(i)')
        self.b2_label_z.grid(row=0, column=2, in_=self.settingsLF2)
        self.b2_label_e = ttk.Label(self, text='Leading Edge:')
        self.b2_label_e.grid(row=1, column=0, in_=self.settingsLF2, sticky=W, padx=10)
        self.b2_entry_e_u = ttk.Entry(self, justify='center')
        self.b2_entry_e_u.grid(in_=self.settingsLF2, row=1, column=1, padx=2)
        self.b2_entry_e_z = ttk.Entry(self, justify='center')
        self.b2_entry_e_z.grid(in_=self.settingsLF2, row=1, column=2, padx=2)
        self.b2_label_0 = ttk.Label(self, text='Upper Trailing Edge:')
        self.b2_label_0.grid(row=2, column=0, in_=self.settingsLF2, sticky=W, padx=10)
        self.b2_entry_0_u = ttk.Entry(self, justify='center')
        self.b2_entry_0_u.grid(in_=self.settingsLF2, row=2, column=1, padx=2)
        self.b2_entry_0_z = ttk.Entry(self, justify='center')
        self.b2_entry_0_z.grid(in_=self.settingsLF2, row=2, column=2, padx=2)
        self.b2_label_1 = ttk.Label(self, text='Lower Trailing Edge:')
        self.b2_label_1.grid(row=3, column=0, in_=self.settingsLF2, sticky=W, padx=10)
        self.b2_entry_1_u = ttk.Entry(self, justify='center')
        self.b2_entry_1_u.grid(in_=self.settingsLF2, row=3, column=1, padx=2)
        self.b2_entry_1_z = ttk.Entry(self, justify='center')
        self.b2_entry_1_z.grid(in_=self.settingsLF2, row=3, column=2, padx=2)
        self.b2_label_t = ttk.Label(self, text='Top Point:')
        self.b2_label_t.grid(row=4, column=0, in_=self.settingsLF2, sticky=W, padx=10)
        self.b2_entry_t_u = ttk.Entry(self, justify='center')
        self.b2_entry_t_u.grid(in_=self.settingsLF2, row=4, column=1, padx=2)
        self.b2_entry_t_z = ttk.Entry(self, justify='center')
        self.b2_entry_t_z.grid(in_=self.settingsLF2, row=4, column=2, padx=2)
        self.b2_label_b = ttk.Label(self, text='Bottom Point:')
        self.b2_label_b.grid(row=5, column=0, in_=self.settingsLF2, sticky=W, padx=10)
        self.b2_entry_b_u = ttk.Entry(self, justify='center')
        self.b2_entry_b_u.grid(in_=self.settingsLF2, row=5, column=1, padx=2)
        self.b2_entry_b_z = ttk.Entry(self, justify='center')
        self.b2_entry_b_z.grid(in_=self.settingsLF2, row=5, column=2, padx=2)
        ttk.Separator(self.settingsLF2, orient=HORIZONTAL).grid(columnspan=3, sticky=W+E,
pady=10)
        self.b2_label_r = ttk.Label(self, text='Radius:')
        self.b2_label_r.grid(row=7, column=0, in_=self.settingsLF2, sticky=W, padx=10)
        self.b2_entry_r = ttk.Entry(self, justify='center')
        self.b2_entry_r.grid(in_=self.settingsLF2, row=7, column=1, padx=2)
        self.b2_label_b0 = ttk.Label(self, text='Beta Upper Angle:')
        self.b2_label_b0.grid(row=8, column=0, in_=self.settingsLF2, sticky=W, padx=10)
        self.b2_entry_b0 = ttk.Entry(self, justify='center')
        self.b2_entry_b0.grid(in_=self.settingsLF2, row=8, column=1, padx=2)
        self.b2_label_b1 = ttk.Label(self, text='Beta Lower Angle:')
        self.b2_label_b1.grid(row=9, column=0, in_=self.settingsLF2, sticky=W, padx=10)
        self.b2_entry_b1 = ttk.Entry(self, justify='center')
        self.b2_entry_b1.grid(in_=self.settingsLF2, row=9, column=1, padx=2)
        self.b2_label_g = ttk.Label(self, text='Gamma:')
        self.b2_label_g.grid(row=10, column=0, in_=self.settingsLF2, sticky=W, padx=10)
        self.b2_entry_g = ttk.Entry(self, justify='center')
        self.b2_entry_g.grid(in_=self.settingsLF2, row=10, column=1, padx=2)

        #Parametric Polynomial Boundary Condition 3 Panel
        self.settingsLF3 = ttk.LabelFrame(self, text='Settings', borderwidth=10)
        self.b3_label_u = ttk.Label(self, text='U(i)')
        self.b3_label_u.grid(row=0, column=1, in_=self.settingsLF3)
        self.b3_label_z = ttk.Label(self, text='Z(i)')
        self.b3_label_z.grid(row=0, column=2, in_=self.settingsLF3)
        self.b3_label_e = ttk.Label(self, text='Leading Edge:')
        self.b3_label_e.grid(row=1, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_e_u = ttk.Entry(self, justify='center')
        self.b3_entry_e_u.grid(in_=self.settingsLF3, row=1, column=1, padx=2)
        self.b3_entry_e_z = ttk.Entry(self, justify='center')
        self.b3_entry_e_z.grid(in_=self.settingsLF3, row=1, column=2, padx=2)
        self.b3_label_0 = ttk.Label(self, text='Upper Trailing Edge:')
        self.b3_label_0.grid(row=2, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_0_u = ttk.Entry(self, justify='center')
        self.b3_entry_0_u.grid(in_=self.settingsLF3, row=2, column=1, padx=2)
        self.b3_entry_0_z = ttk.Entry(self, justify='center')
        self.b3_entry_0_z.grid(in_=self.settingsLF3, row=2, column=2, padx=2)
        self.b3_label_1 = ttk.Label(self, text='Lower Trailing Edge:')
        self.b3_label_1.grid(row=3, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_1_u = ttk.Entry(self, justify='center')
        self.b3_entry_1_u.grid(in_=self.settingsLF3, row=3, column=1, padx=2)
        self.b3_entry_1_z = ttk.Entry(self, justify='center')
        self.b3_entry_1_z.grid(in_=self.settingsLF3, row=3, column=2, padx=2)
        self.b3_label_t = ttk.Label(self, text='Top Point:')
        self.b3_label_t.grid(row=4, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_t_u = ttk.Entry(self, justify='center')
        self.b3_entry_t_u.grid(in_=self.settingsLF3, row=4, column=1, padx=2)
        self.b3_entry_t_z = ttk.Entry(self, justify='center')
        self.b3_entry_t_z.grid(in_=self.settingsLF3, row=4, column=2, padx=2)
        self.b3_label_b = ttk.Label(self, text='Bottom Point:')
        self.b3_label_b.grid(row=5, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_b_u = ttk.Entry(self, justify='center')
        self.b3_entry_b_u.grid(in_=self.settingsLF3, row=5, column=1, padx=2)
        self.b3_entry_b_z = ttk.Entry(self, justify='center')
        self.b3_entry_b_z.grid(in_=self.settingsLF3, row=5, column=2, padx=2)
        ttk.Separator(self.settingsLF3, orient=HORIZONTAL).grid(columnspan=3, sticky=W+E,
pady=10)
        self.b3_label_0_r = ttk.Label(self, text='Upper Radius:')
        self.b3_label_0_r.grid(row=7, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_0_r = ttk.Entry(self, justify='center')
        self.b3_entry_0_r.grid(in_=self.settingsLF3, row=7, column=1, padx=2)
        self.b3_label_1_r = ttk.Label(self, text='Lower Radius:')
        self.b3_label_1_r.grid(row=8, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_1_r = ttk.Entry(self, justify='center')
        self.b3_entry_1_r.grid(in_=self.settingsLF3, row=8, column=1, padx=2)
        self.b3_label_b0 = ttk.Label(self, text='Beta Upper Angle:')
        self.b3_label_b0.grid(row=9, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_b0 = ttk.Entry(self, justify='center')
        self.b3_entry_b0.grid(in_=self.settingsLF3, row=9, column=1, padx=2)
        self.b3_label_b1 = ttk.Label(self, text='Beta Lower Angle:')
        self.b3_label_b1.grid(row=10, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_b1 = ttk.Entry(self, justify='center')
        self.b3_entry_b1.grid(in_=self.settingsLF3, row=10, column=1, padx=2)
        self.b3_label_0_g = ttk.Label(self, text='Upper Gamma:')
        self.b3_label_0_g.grid(row=11, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_0_g = ttk.Entry(self, justify='center')
        self.b3_entry_0_g.grid(in_=self.settingsLF3, row=11, column=1, padx=2)
        self.b3_label_1_g = ttk.Label(self, text='Lower Gamma:')
        self.b3_label_1_g.grid(row=12, column=0, in_=self.settingsLF3, sticky=W, padx=10)
        self.b3_entry_1_g = ttk.Entry(self, justify='center')
        self.b3_entry_1_g.grid(in_=self.settingsLF3, row=12, column=1, padx=2)

        #Parametric Cubic Spline
        self.settingsLF4 = ttk.LabelFrame(self, text='Settings', borderwidth=10)
        pointsLabel = ttk.Label(self, text='Points').grid(in_=self.settingsLF4, row=0, column=0)
        self.listbox = Listbox(self, height=10, width = 20)
        self.listbox.grid(in_=self.settingsLF4, row=1, column=0, rowspan=5, padx=5)
        #0, 0.03, 0.19, 0.5, 0.88, 1.0
        #0, 0.0007, -0.049, 0.0, 0.0488, 0
        #fill the listbox for testing purposes
        self.listbox.insert(END, " 0, 0")
        self.listbox.insert(END, " 0.03, 0.0007")
        self.listbox.insert(END, " 0.19, -0.049")
        self.listbox.insert(END, " 0.5, 0.0")
        self.listbox.insert(END, " 0.88, 0.0488")
        self.listbox.insert(END, " 1.0, 0")
        self.curIndex = None
        self.listbox.bind('<Button-1>', self.setCurrent)
        self.listbox.bind('<B1-Motion>', self.shiftSelection)
        self.addPoint = ttk.Button(self, text='Add Point', command=lambda: self.addListPoint())
        self.addPoint.grid(in_=self.settingsLF4, sticky=W+E, row=1, column=1, padx=10, pady=5)
        self.deletePoint = ttk.Button(self, text='Delete Point', command=lambda:
self.deleteListPoint())
        self.deletePoint.grid(in_=self.settingsLF4, sticky=W+E, row=2, column=1, padx=10)
        self.leftConstraintValue=StringVar()
        self.leftConstraint = ttk.Combobox(self, textvariable=self.leftConstraintValue,
state='readonly')
        self.leftConstraint.grid(in_=self.settingsLF4, sticky=W+E, row=4, column=1, padx=10,
pady=5)
        self.leftConstraint['values'] = ('Left End Constraint', 'Stiff', 'Flat', 'Flexible')
        self.leftConstraint.current(0)
        self.seperatorC = ttk.Frame(self, borderwidth=15)
        self.seperatorC.grid(in_=self.settingsLF4, row=3, column=1, sticky=W+E)
        self.seperatorC.grid_columnconfigure(0, weight=1)
        ttk.Separator(self.seperatorC, orient=HORIZONTAL).grid(sticky=W+E)
        self.rightConstraintValue=StringVar()
        self.rightConstraint = ttk.Combobox(self, textvariable=self.rightConstraintValue,
state='readonly')
        self.rightConstraint.grid(in_=self.settingsLF4, sticky=W+E, row=5, column=1, padx=10,
pady=5)
        self.rightConstraint['values'] = ('Right End Constraint', 'Stiff', 'Flat', 'Flexible')
        self.rightConstraint.current(0)
        self.leftConstraintLabel = ttk.Label(self, text='Left Slope:')
        self.leftConstraintEntry = ttk.Entry(self)
        self.rightConstraintLabel = ttk.Label(self, text='Right Slope:')
        self.rightConstraintEntry = ttk.Entry(self)

        #2D Parametric Fourier Panel
        self.settingsLF5 = ttk.LabelFrame(self, text='Settings', borderwidth=10)
        self.f_label_u = ttk.Label(self, text='U(i)')
        self.f_label_u.grid(row=0, column=1, in_=self.settingsLF5)
        self.f_label_z = ttk.Label(self, text='Z(i)')
        self.f_label_z.grid(row=0, column=2, in_=self.settingsLF5)
        self.f_label_t = ttk.Label(self, text='Top Point:')
        self.f_label_t.grid(row=1, column=0, in_=self.settingsLF5, sticky=W, padx=10)
        self.f_entry_t_u = ttk.Entry(self, justify='center')
        self.f_entry_t_u.grid(in_=self.settingsLF5, row=1, column=1, padx=2)
        self.f_entry_t_z = ttk.Entry(self, justify='center')
        self.f_entry_t_z.grid(in_=self.settingsLF5, row=1, column=2, padx=2)
        self.f_label_b = ttk.Label(self, text='Bottom Point:')
        self.f_label_b.grid(row=3, column=0, in_=self.settingsLF5, sticky=W, padx=10)
        self.f_entry_b_u = ttk.Entry(self, justify='center')
        self.f_entry_b_u.grid(in_=self.settingsLF5, row=3, column=1, padx=2)
        self.f_entry_b_z = ttk.Entry(self, justify='center')
        self.f_entry_b_z.grid(in_=self.settingsLF5, row=3, column=2, padx=2)
        ttk.Separator(self.settingsLF5, orient=HORIZONTAL).grid(columnspan=3, sticky=W+E,
pady=10)
        self.f_label_le_r = ttk.Label(self, text='Leading Edge Radius:')
        self.f_label_le_r.grid(row=5, column=0, in_=self.settingsLF5, sticky=W, padx=10)
        self.f_entry_le_r = ttk.Entry(self, justify='center')
        self.f_entry_le_r.grid(in_=self.settingsLF5, row=5, column=1, padx=2)
        self.f_label_hte = ttk.Label(self, text='Half Trailing Edge:')
        self.f_label_hte.grid(row=6, column=0, in_=self.settingsLF5, sticky=W, padx=10)
        self.f_entry_hte = ttk.Entry(self, justify='center')
        self.f_entry_hte.grid(in_=self.settingsLF5, row=6, column=1, padx=2)
        self.f_label_b0 = ttk.Label(self, text='Beta Upper Angle:')
        self.f_label_b0.grid(row=7, column=0, in_=self.settingsLF5, sticky=W, padx=10)
        self.f_entry_b0 = ttk.Entry(self, justify='center')
        self.f_entry_b0.grid(in_=self.settingsLF5, row=7, column=1, padx=2)
        self.f_label_b1 = ttk.Label(self, text='Beta Lower Angle:')
        self.f_label_b1.grid(row=8, column=0, in_=self.settingsLF5, sticky=W, padx=10)
        self.f_entry_b1 = ttk.Entry(self, justify='center')
        self.f_entry_b1.grid(in_=self.settingsLF5, row=8, column=1, padx=2)
        self.f_label_0_g = ttk.Label(self, text='Upper Gamma:')
        self.f_label_0_g.grid(row=9, column=0, in_=self.settingsLF5, sticky=W, padx=10)
        self.f_entry_0_g = ttk.Entry(self, justify='center')
        self.f_entry_0_g.grid(in_=self.settingsLF5, row=9, column=1, padx=2)
        self.f_label_1_g = ttk.Label(self, text='Lower Gamma:')
        self.f_label_1_g.grid(row=10, column=0, in_=self.settingsLF5, sticky=W, padx=10)
        self.f_entry_1_g = ttk.Entry(self, justify='center')
        self.f_entry_1_g.grid(in_=self.settingsLF5, row=10, column=1, padx=2)

        #3D Parametric Fourier Panel
        self.settingsLF6 = ttk.LabelFrame(self, text='Settings', borderwidth=10)
        pointsLabel = ttk.Label(self, text='Stations').grid(in_=self.settingsLF6, row=0,
column=0)
        self.listboxF = Listbox(self, height=10, width = 20)
        self.listboxF.grid(in_=self.settingsLF6, row=1, column=0, rowspan=5, padx=5)
        #fill the listbox for testing purposes
        self.listboxF.insert(END, "Left Tip")
        self.name.append("Left Tip")
        self.topPointU.append(0.31)
        self.topPointZ.append(0.075)
        self.bottomPointU.append(0.44)
        self.bottomPointZ.append(-0.107)
        self.ler.append(0.02)
        self.hte.append(0.0005)
        self.bu.append(9.5)
        self.bl.append(10.0)
        self.ug.append(0)
        self.lg.append(0.13)
        self.sparX.append(4)
        self.sparY.append(1)
        self.sparZ.append(0)
        self.scale.append(1)
        self.listboxF.insert(END, "Left Panel")
        self.name.append("Left Panel")
        self.topPointU.append(0.31)
        self.topPointZ.append(0.075)
        self.bottomPointU.append(0.44)
        self.bottomPointZ.append(-0.107)
        self.ler.append(0.02)
        self.hte.append(0.0005)
        self.bu.append(9.5)
        self.bl.append(10.0)
        self.ug.append(0)
        self.lg.append(0.13)
        self.sparX.append(1)
        self.sparY.append(-0.5)
        self.sparZ.append(0)
        self.scale.append(2)
        self.listboxF.insert(END, "Root")
        self.name.append("Root")
        self.topPointU.append(0.31)
        self.topPointZ.append(0.075)
        self.bottomPointU.append(0.44)
        self.bottomPointZ.append(-0.107)
        self.ler.append(0.02)
        self.hte.append(0.0005)
        self.bu.append(9.5)
        self.bl.append(10.0)
        self.ug.append(0)
        self.lg.append(0.13)
        self.sparX.append(0)
        self.sparY.append(-1)
        self.sparZ.append(0)
        self.scale.append(4)
        self.listboxF.insert(END, "Right Panel")
        self.name.append("Right Panel")
        self.topPointU.append(0.31)
        self.topPointZ.append(0.075)
        self.bottomPointU.append(0.44)
        self.bottomPointZ.append(-0.107)
        self.ler.append(0.02)
        self.hte.append(0.0005)
        self.bu.append(9.5)
        self.bl.append(10.0)
        self.ug.append(0)
        self.lg.append(0.13)
        self.sparX.append(-1)
        self.sparY.append(-0.5)
        self.sparZ.append(0)
        self.scale.append(2)
        self.listboxF.insert(END, "Right Tip")
        self.name.append("Right Tip")
        self.topPointU.append(0.31)
        self.topPointZ.append(0.075)
        self.bottomPointU.append(0.44)
        self.bottomPointZ.append(-0.107)
        self.ler.append(0.02)
        self.hte.append(0.0005)
        self.bu.append(9.5)
        self.bl.append(10.0)
        self.ug.append(0)
        self.lg.append(0.13)
        self.sparX.append(-4)
        self.sparY.append(1)
        self.sparZ.append(0)
        self.scale.append(1)
        self.curIndexF = None
        self.listboxF.bind('<Button-1>', self.setCurrentF)
        self.addStationF = ttk.Button(self, text='Add Station', command=lambda:
self.addStation())
        self.addStationF.grid(in_=self.settingsLF6, sticky=W+E, row=1, column=1, padx=10, pady=5)
        self.editStationF = ttk.Button(self, text='Edit Station', command=lambda:
self.editStation())
        self.editStationF.grid(in_=self.settingsLF6, sticky=W+E, row=2, column=1, padx=10,
pady=5)
        self.deleteStationF = ttk.Button(self, text='Delete Station', command=lambda:
self.deleteStation())
        self.deleteStationF.grid(in_=self.settingsLF6, sticky=W+E, row=3, column=1, padx=10,
pady=5)
        self.seperatorF = ttk.Frame(self, borderwidth=15)
        self.seperatorF.grid(in_=self.inputsF, column=0, row=3, sticky=W+E)
        self.seperatorF.grid_columnconfigure(0, weight=1)
        ttk.Separator(self.seperatorF, orient=HORIZONTAL).grid(sticky=W+E)
        self.buttonsF = ttk.Frame(self)
        self.buttonsF.grid(in_=self.inputsF, row=4, column=0, sticky=W+E)
        self.buttonsF.grid_columnconfigure(0, weight=1)
        self.applyB = ttk.Button(self, text='Apply Settings', command=lambda: self.apply())
        self.applyB.grid(in_=self.buttonsF, sticky=W+E)
        self.exportB = ttk.Button(self, text='Export to Blender', command=lambda: self.export())
        self.exportB.grid(in_=self.buttonsF, sticky=W+E)

    #   addStation
    #   - Creates a new window to prompt the user for data for a new station
    def addStation(self):
        w = 408
        h = 470
        self.stationWindow = Tk()
        self.stationWindow.wm_title("Add Station")
        nameLabel = ttk.Label(self.stationWindow, text="Name:")
        nameLabel.grid(in_=self.stationWindow, row=0, column=0, pady=10, sticky=E)
        self.nameEntry = ttk.Entry(self.stationWindow, justify='center')
        self.nameEntry.grid(padx=5, row=0, column=1, pady=10)
        u_label = ttk.Label(self.stationWindow, text="U:")
        u_label.grid(in_=self.stationWindow, row=1, column=1, padx=3, pady=3)
        z_label = ttk.Label(self.stationWindow, text="Z:")
        z_label.grid(in_=self.stationWindow, row=1, column=2)
        topPoint = ttk.Label(self.stationWindow, text="Top Point:")
        topPoint.grid(in_=self.stationWindow, row=2, column=0, padx=3, pady=3, sticky=E)
        self.topPointUEntry = ttk.Entry(self.stationWindow, justify='center')
        self.topPointUEntry.grid(padx=5, row=2, column=1, pady=3)
        self.topPointZEntry = ttk.Entry(self.stationWindow, justify='center')
        self.topPointZEntry.grid(padx=5, row=2, column=2, pady=3)
        bottomPoint = ttk.Label(self.stationWindow, text="Bottom Point:")
        bottomPoint.grid(in_=self.stationWindow, row=3, column=0, padx=3, pady=3, sticky=E)
        self.bottomPointUEntry = ttk.Entry(self.stationWindow, justify='center')
        self.bottomPointUEntry.grid(padx=5, row=3, column=1, pady=3)
        self.bottomPointZEntry = ttk.Entry(self.stationWindow, justify='center')
        self.bottomPointZEntry.grid(padx=5, row=3, column=2, pady=3)
        ttk.Separator(self.stationWindow, orient=HORIZONTAL).grid(columnspan=3, sticky=W+E,
pady=10)
        rLabel = ttk.Label(self.stationWindow, text="Leading Edge Radius:")
        rLabel.grid(in_=self.stationWindow, row=5, column=0, padx=3, pady=3, sticky=E)
        self.rEntry = ttk.Entry(self.stationWindow, justify='center')
        self.rEntry.grid(padx=5, row=5, column=1, pady=3)
        hteLabel = ttk.Label(self.stationWindow, text="Half Trailing Edge:")
        hteLabel.grid(in_=self.stationWindow, row=6, column=0, padx=3, pady=3, sticky=E)
        self.hteEntry = ttk.Entry(self.stationWindow, justify='center')
        self.hteEntry.grid(padx=5, row=6, column=1, pady=3)
        buLabel = ttk.Label(self.stationWindow, text="Beta Upper Angle:")
        buLabel.grid(in_=self.stationWindow, row=7, column=0, padx=3, pady=3, sticky=E)
        self.buEntry = ttk.Entry(self.stationWindow, justify='center')
        self.buEntry.grid(padx=5, row=7, column=1, pady=3)
        blLabel = ttk.Label(self.stationWindow, text="Beta Lower Angle:")
        blLabel.grid(in_=self.stationWindow, row=8, column=0, padx=3, pady=3, sticky=E)
        self.blEntry = ttk.Entry(self.stationWindow, justify='center')
        self.blEntry.grid(padx=5, row=8, column=1, pady=3)
        ugLabel = ttk.Label(self.stationWindow, text="Upper Gamma:")
        ugLabel.grid(in_=self.stationWindow, row=9, column=0, padx=3, pady=3, sticky=E)
        self.ugEntry = ttk.Entry(self.stationWindow, justify='center')
        self.ugEntry.grid(padx=5, row=9, column=1, pady=3)
        lgLabel = ttk.Label(self.stationWindow, text="Lower Gamma:")
        lgLabel.grid(in_=self.stationWindow, row=10, column=0, padx=3, pady=3, sticky=E)
        self.lgEntry = ttk.Entry(self.stationWindow, justify='center')
        self.lgEntry.grid(padx=5, row=10, column=1, pady=3)
        ttk.Separator(self.stationWindow, orient=HORIZONTAL).grid(columnspan=3, sticky=W+E,
pady=10)
        sparXLabel = ttk.Label(self.stationWindow, text="Spar Backbone X")
        sparXLabel.grid(in_=self.stationWindow, row=12, column=0, padx=20, pady=3, sticky=W+E)
        sparYLabel = ttk.Label(self.stationWindow, text="Spar Backbone Y")
        sparYLabel.grid(in_=self.stationWindow, row=12, column=1, padx=20, pady=3, sticky=W+E)
        sparZLabel = ttk.Label(self.stationWindow, text="Spar Backbone Z")
        sparZLabel.grid(in_=self.stationWindow, row=12, column=2, padx=20, pady=3, sticky=W+E)
        self.sparXEntry = ttk.Entry(self.stationWindow, justify='center')
        self.sparXEntry.grid(padx=5, row=13, column=0, pady=3)
        self.sparYEntry = ttk.Entry(self.stationWindow, justify='center')
        self.sparYEntry.grid(padx=5, row=13, column=1, pady=3)
        self.sparZEntry = ttk.Entry(self.stationWindow, justify='center')
        self.sparZEntry.grid(padx=5, row=13, column=2, pady=3)
        scaleLabel = ttk.Label(self.stationWindow, text="Scale:")
        scaleLabel.grid(in_=self.stationWindow, row=14, column=0, padx=20, pady=3, sticky=E)
        self.scaleEntry = ttk.Entry(self.stationWindow, justify='center')
        self.scaleEntry.grid(padx=5, row=14, column=1, pady=3)
        add = ttk.Button(self.stationWindow, text="Add", command=lambda: self.addStationInputs())
        add.grid(row=15, column=1, sticky=W+E, padx=10, pady=10)
        cancel = ttk.Button(self.stationWindow, text="Cancel", command=lambda: self.cancelF())
        cancel.grid(row=15, column=2, sticky=W+E, padx=10, pady=10)
        ws = self.stationWindow.winfo_screenwidth()
        hs = self.stationWindow.winfo_screenheight()
        x = (ws - w) / 2
        y = (hs - h) / 2
        self.stationWindow.geometry('%dx%d+%d+%d' % (w, h, x, y))

    #   addStationInputs
    #   - Called when the "Add" button is pressed, takes all the inputs and puts them into the
appropriate arrays used
    #     to store data, throws an Error Dialog if the input values are not acceptable
    def addStationInputs(self):
        if self.nameEntry.get():
            try:
                float(self.topPointUEntry.get())
                float(self.topPointZEntry.get())
                float(self.bottomPointUEntry.get())
                float(self.bottomPointZEntry.get())
                float(self.rEntry.get())
                float(self.hteEntry.get())
                float(self.buEntry.get())
                float(self.blEntry.get())
                float(self.ugEntry.get())
                float(self.lgEntry.get())
                float(self.sparXEntry.get())
                float(self.sparYEntry.get())
                float(self.sparZEntry.get())
                float(self.scaleEntry.get())
                self.name.append(self.nameEntry.get())
                self.topPointU.append(self.topPointUEntry.get())
                self.topPointZ.append(self.topPointZEntry.get())
                self.bottomPointU.append(self.bottomPointUEntry.get())
                self.bottomPointZ.append(self.bottomPointZEntry.get())
                self.ler.append(self.rEntry.get())
                self.hte.append(self.hteEntry.get())
                self.bu.append(self.buEntry.get())
                self.bl.append(self.blEntry.get())
                self.ug.append(self.ugEntry.get())
                self.lg.append(self.lgEntry.get())
                self.sparX.append(self.sparXEntry.get())
                self.sparY.append(self.sparYEntry.get())
                self.sparZ.append(self.sparZEntry.get())
                self.scale.append(self.scaleEntry.get())
                self.listboxF.insert(END, self.nameEntry.get())
            except ValueError:
                tkinter.messagebox.showerror("Error", "Entries must consist of numerical values
and must not be blank.")
        else:
            tkinter.messagebox.showerror("Error", "Enter a name.")
        self.stationWindow.destroy()

    #   deleteStation
    #   - Called then the "Delete Station" button is pressed, removes the data for the selected
station, throws an Error
    #     Dialog if no station is selected
    def deleteStation(self):
        try:
            self.name.pop(self.curIndexF)
            self.topPointU.pop(self.curIndexF)
            self.topPointZ.pop(self.curIndexF)
            self.bottomPointU.pop(self.curIndexF)
            self.bottomPointZ.pop(self.curIndexF)
            self.ler.pop(self.curIndexF)
            self.hte.pop(self.curIndexF)
            self.bu.pop(self.curIndexF)
            self.bl.pop(self.curIndexF)
            self.ug.pop(self.curIndexF)
            self.lg.pop(self.curIndexF)
            self.sparX.pop(self.curIndexF)
            self.sparY.pop(self.curIndexF)
            self.sparZ.pop(self.curIndexF)
            self.scale.pop(self.curIndexF)
            self.listboxF.delete(self.curIndexF)
        except:
            tkinter.messagebox.showwarning("Warning", "Must select to delete!")

    #   editStation
    #   - Called the the "Edit Station" button is pressed, creates a new window to prompt the
user for new data, throws
    #     an Error Dialog if no station is selected
    def editStation(self):
        try:
            w = 408
            h = 470
            self.stationWindow = Tk()
            self.stationWindow.wm_title("Edit Station")
            nameLabel = ttk.Label(self.stationWindow, text="Name:")
            nameLabel.grid(in_=self.stationWindow, row=0, column=0, pady=10, sticky=E)
            self.nameEntry = ttk.Entry(self.stationWindow, justify='center')
            self.nameEntry.grid(padx=5, row=0, column=1, pady=10)
            self.nameEntry.insert(0, self.name[self.curIndexF])
            u_label = ttk.Label(self.stationWindow, text="U:")
            u_label.grid(in_=self.stationWindow, row=1, column=1, padx=3, pady=3)
            z_label = ttk.Label(self.stationWindow, text="Z:")
            z_label.grid(in_=self.stationWindow, row=1, column=2)
            topPoint = ttk.Label(self.stationWindow, text="Top Point:")
            topPoint.grid(in_=self.stationWindow, row=2, column=0, padx=3, pady=3, sticky=E)
            self.topPointUEntry = ttk.Entry(self.stationWindow, justify='center')
            self.topPointUEntry.grid(padx=5, row=2, column=1, pady=3)
            self.topPointUEntry.insert(0, self.topPointU[self.curIndexF])
            self.topPointZEntry = ttk.Entry(self.stationWindow, justify='center')
            self.topPointZEntry.grid(padx=5, row=2, column=2, pady=3)
            self.topPointZEntry.insert(0, self.topPointZ[self.curIndexF])
            bottomPoint = ttk.Label(self.stationWindow, text="Bottom Point:")
            bottomPoint.grid(in_=self.stationWindow, row=3, column=0, padx=3, pady=3, sticky=E)
            self.bottomPointUEntry = ttk.Entry(self.stationWindow, justify='center')
            self.bottomPointUEntry.grid(padx=5, row=3, column=1, pady=3)
            self.bottomPointUEntry.insert(0, self.bottomPointU[self.curIndexF])
            self.bottomPointZEntry = ttk.Entry(self.stationWindow, justify='center')
            self.bottomPointZEntry.grid(padx=5, row=3, column=2, pady=3)
            self.bottomPointZEntry.insert(0, self.bottomPointZ[self.curIndexF])
            ttk.Separator(self.stationWindow, orient=HORIZONTAL).grid(columnspan=3, sticky=W+E,
pady=10)
            rLabel = ttk.Label(self.stationWindow, text="Leading Edge Radius:")
            rLabel.grid(in_=self.stationWindow, row=5, column=0, padx=3, pady=3, sticky=E)
            self.rEntry = ttk.Entry(self.stationWindow, justify='center')
            self.rEntry.grid(padx=5, row=5, column=1, pady=3)
            self.rEntry.insert(0, self.ler[self.curIndexF])
            hteLabel = ttk.Label(self.stationWindow, text="Half Trailing Edge:")
            hteLabel.grid(in_=self.stationWindow, row=6, column=0, padx=3, pady=3, sticky=E)
            self.hteEntry = ttk.Entry(self.stationWindow, justify='center')
            self.hteEntry.grid(padx=5, row=6, column=1, pady=3)
            self.hteEntry.insert(0, self.hte[self.curIndexF])
            buLabel = ttk.Label(self.stationWindow, text="Beta Upper Angle:")
            buLabel.grid(in_=self.stationWindow, row=7, column=0, padx=3, pady=3, sticky=E)
            self.buEntry = ttk.Entry(self.stationWindow, justify='center')
            self.buEntry.grid(padx=5, row=7, column=1, pady=3)
            self.buEntry.insert(0, self.bu[self.curIndexF])
            blLabel = ttk.Label(self.stationWindow, text="Beta Lower Angle:")
            blLabel.grid(in_=self.stationWindow, row=8, column=0, padx=3, pady=3, sticky=E)
            self.blEntry = ttk.Entry(self.stationWindow, justify='center')
            self.blEntry.grid(padx=5, row=8, column=1, pady=3)
            self.blEntry.insert(0, self.bl[self.curIndexF])
            ugLabel = ttk.Label(self.stationWindow, text="Upper Gamma:")
            ugLabel.grid(in_=self.stationWindow, row=9, column=0, padx=3, pady=3, sticky=E)
            self.ugEntry = ttk.Entry(self.stationWindow, justify='center')
            self.ugEntry.grid(padx=5, row=9, column=1, pady=3)
            self.ugEntry.insert(0, self.ug[self.curIndexF])
            lgLabel = ttk.Label(self.stationWindow, text="Lower Gamma:")
            lgLabel.grid(in_=self.stationWindow, row=10, column=0, padx=3, pady=3, sticky=E)
            self.lgEntry = ttk.Entry(self.stationWindow, justify='center')
            self.lgEntry.grid(padx=5, row=10, column=1, pady=3)
            self.lgEntry.insert(0, self.lg[self.curIndexF])
            ttk.Separator(self.stationWindow, orient=HORIZONTAL).grid(columnspan=3, sticky=W+E,
pady=10)
            sparXLabel = ttk.Label(self.stationWindow, text="Spar Backbone X")
            sparXLabel.grid(in_=self.stationWindow, row=12, column=0, padx=20, pady=3,
sticky=W+E)
            sparYLabel = ttk.Label(self.stationWindow, text="Spar Backbone Y")
            sparYLabel.grid(in_=self.stationWindow, row=12, column=1, padx=20, pady=3,
sticky=W+E)
            sparZLabel = ttk.Label(self.stationWindow, text="Spar Backbone Z")
            sparZLabel.grid(in_=self.stationWindow, row=12, column=2, padx=20, pady=3,
sticky=W+E)
            self.sparXEntry = ttk.Entry(self.stationWindow, justify='center')
            self.sparXEntry.grid(padx=5, row=13, column=0, pady=3)
            self.sparXEntry.insert(0, self.sparX[self.curIndexF])
            self.sparYEntry = ttk.Entry(self.stationWindow, justify='center')
            self.sparYEntry.grid(padx=5, row=13, column=1, pady=3)
            self.sparYEntry.insert(0, self.sparY[self.curIndexF])
            self.sparZEntry = ttk.Entry(self.stationWindow, justify='center')
            self.sparZEntry.grid(padx=5, row=13, column=2, pady=3)
            self.sparZEntry.insert(0, self.sparZ[self.curIndexF])
            scaleLabel = ttk.Label(self.stationWindow, text="Scale:")
            scaleLabel.grid(in_=self.stationWindow, row=14, column=0, padx=20, pady=3, sticky=E)
            self.scaleEntry = ttk.Entry(self.stationWindow, justify='center')
            self.scaleEntry.grid(padx=5, row=14, column=1, pady=3)
            self.scaleEntry.insert(0, self.scale[self.curIndexF])
            add = ttk.Button(self.stationWindow, text="OK", command=lambda:
self.editStationInputs())
            add.grid(row=15, column=1, sticky=W+E, padx=10, pady=10)
            cancel = ttk.Button(self.stationWindow, text="Cancel", command=lambda:
self.cancelF())
            cancel.grid(row=15, column=2, sticky=W+E, padx=10, pady=10)
            ws = self.stationWindow.winfo_screenwidth()
            hs = self.stationWindow.winfo_screenheight()
            x = (ws - w) / 2
            y = (hs - h) / 2
            self.stationWindow.geometry('%dx%d+%d+%d' % (w, h, x, y))
        except:
            self.stationWindow.destroy()
            tkinter.messagebox.showwarning("Error", "Select a station to edit.")

    #   editStationInputs
    #   - Called when the "OK" button is pressed, replaces the selected station with the new
data, throws an Error
    #     Dialog if the input values are not acceptable
    def editStationInputs(self):
        if self.nameEntry.get():
            try:
                #Remove current entry
                self.name.pop(self.curIndexF)
                self.topPointU.pop(self.curIndexF)
                self.topPointZ.pop(self.curIndexF)
                self.bottomPointU.pop(self.curIndexF)
                self.bottomPointZ.pop(self.curIndexF)
                self.ler.pop(self.curIndexF)
                self.hte.pop(self.curIndexF)
                self.bu.pop(self.curIndexF)
                self.bl.pop(self.curIndexF)
                self.ug.pop(self.curIndexF)
                self.lg.pop(self.curIndexF)
                self.sparX.pop(self.curIndexF)
                self.sparY.pop(self.curIndexF)
                self.sparZ.pop(self.curIndexF)
                self.scale.pop(self.curIndexF)
                self.listboxF.delete(self.curIndexF)
                #Insert new data into old location
                self.name.insert(self.curIndexF, self.nameEntry.get())
                self.topPointU.insert(self.curIndexF, self.topPointUEntry.get())
                self.topPointZ.insert(self.curIndexF, self.topPointZEntry.get())
                self.bottomPointU.insert(self.curIndexF, self.bottomPointUEntry.get())
                self.bottomPointZ.insert(self.curIndexF, self.bottomPointZEntry.get())
                self.ler.insert(self.curIndexF, self.rEntry.get())
                self.hte.insert(self.curIndexF, self.hteEntry.get())
                self.bu.insert(self.curIndexF, self.buEntry.get())
                self.bl.insert(self.curIndexF, self.blEntry.get())
                self.ug.insert(self.curIndexF, self.ugEntry.get())
                self.lg.insert(self.curIndexF, self.lgEntry.get())
                self.sparX.insert(self.curIndexF, self.sparXEntry.get())
                self.sparY.insert(self.curIndexF, self.sparYEntry.get())
                self.sparZ.insert(self.curIndexF, self.sparZEntry.get())
                self.scale.insert(self.curIndexF, self.scaleEntry.get())
                self.listboxF.insert(self.curIndexF, self.nameEntry.get())
            except ValueError:
                tkinter.messagebox.showerror("Error", "Entries must consist of numerical values 
and must not be blank.")
        else:
            tkinter.messagebox.showerror("Error", "Enter a name.")
        self.stationWindow.destroy()

    #   addListPoint
    #   - Creates a new window to prompt the user for a new point (Parametric Cubic Spline)
    def addListPoint(self):
        w = 275
        h = 100
        self.pointWindow = Tk()
        self.pointWindow.wm_title("Add Point")
        u_label = ttk.Label(self.pointWindow, text="U:")
        u_label.grid(in_=self.pointWindow, row=0, column=0, padx=3, pady=3)
        self.u_entry = ttk.Entry(self.pointWindow, justify='center')
        self.u_entry.grid(padx=5, row=1, column=0, pady=3)
        z_label = ttk.Label(self.pointWindow, text="Z:")
        z_label.grid(in_=self.pointWindow, row=0, column=1)
        self.z_entry = ttk.Entry(self.pointWindow, justify='center')
        self.z_entry.grid(padx=5, row=1, column=1)
        add = ttk.Button(self.pointWindow, text="Add", command=lambda: self.add())
        add.grid(row=2, column=0, sticky=W+E, padx=10, pady=10)
        cancel = ttk.Button(self.pointWindow, text="Cancel", command=lambda: self.cancel())
        cancel.grid(row=2, column=1, sticky=W+E, padx=10, pady=10)
        ws = self.pointWindow.winfo_screenwidth()
        hs = self.pointWindow.winfo_screenheight()
        x = (ws - w) / 2
        y = (hs - h) / 2
        self.pointWindow.geometry('%dx%d+%d+%d' % (w, h, x, y))

    #   add
    #   - Called when the "Add" button is pressed, takes the input values from the Dialog Box and 
inserts them into the
    #     listbox, which is used to display points in a scrollable list area, throws an Error 
Dialog if input values
    #     are not acceptable
    def add(self):
        a = self.u_entry.get()
        b = self.z_entry.get()
        try:
            float(a)
            float(b)
            self.listbox.insert(END, ' ' + self.u_entry.get() + ", " + self.z_entry.get())
        except ValueError:
            tkinter.messagebox.showerror("Error", "Entries must consist of numerical values and 
must not be blank.")
        self.pointWindow.destroy()

    #   cancel
    #   - Destroys the window for adding new points to the listbox
    def cancel(self):
        self.pointWindow.destroy()

    #   cancelF
    #   - Destroys the station window for adding and editing stations for a 3D Parametric Fourier
    def cancelF(self):
        self.stationWindow.destroy()
    #   deleteListPoint
    #   - Called when the "Delete Point" button is pressed, removes the currently selected point
from the point listbox
    def deleteListPoint(self):
        try:
            self.listbox.delete(self.listbox.curselection())
        except:
            tkinter.messagebox.showwarning("Warning", "Must select to delete!")

    #   setCurrent
    #   - Sets the current index value for points to the one selected in the listbox
    def setCurrent(self, event):
        self.curIndex = self.listbox.nearest(event.y)

    #   setCurrentF
    #   - Sets the current index value for 3D Parametric Fourier stations to the one selected in
the listbox
    def setCurrentF(self, event):
        self.curIndexF = self.listboxF.nearest(event.y)

    #   shiftSelection
    #   - Called when an event occurs (mouse left click and drag) on the listbox, this method
allows for the mouse to
    #     move a listbox entry up or down on the listbox
    def shiftSelection(self, event):
        i = self.listbox.nearest(event.y)
        if i < self.curIndex:
            x = self.listbox.get(i)
            self.listbox.delete(i)
            self.listbox.insert(i+1, x)
            self.listboxcurIndex = i
        elif i > self.curIndex:
            x = self.listbox.get(i)
            self.listbox.delete(i)
            self.listbox.insert(i-1, x)
            self.curIndex = i

    #   newFile
    #   - Creates a "new file", clearing all the current entries
    #   - Not applicable for all geometric methods
    def newFile(self):
        if tkinter.messagebox.askyesno("New File?", "Are you sure you want to start a new
file?"):
            self.plot.delete("all")
            self.b1_entry_e_u.delete(0,END)
            self.b1_entry_e_z.delete(0,END)
            self.b1_entry_0_u.delete(0,END)
            self.b1_entry_0_z.delete(0,END)
            self.b1_entry_1_u.delete(0,END)
            self.b1_entry_1_z.delete(0,END)
            self.b1_entry_r.delete(0,END)
            self.b1_entry_b0.delete(0,END)
            self.b1_entry_b1.delete(0,END)
            self.b1_entry_g.delete(0,END)
            self.b2_entry_e_u.delete(0,END)
            self.b2_entry_e_z.delete(0,END)
            self.b2_entry_0_u.delete(0,END)
            self.b2_entry_0_z.delete(0,END)
            self.b2_entry_1_u.delete(0,END)
            self.b2_entry_1_z.delete(0,END)
            self.b2_entry_t_u.delete(0,END)
            self.b2_entry_t_z.delete(0,END)
            self.b2_entry_b_u.delete(0,END)
            self.b2_entry_b_z.delete(0,END)
            self.b2_entry_r.delete(0,END)
            self.b2_entry_b0.delete(0,END)
            self.b2_entry_b1.delete(0,END)
            self.b2_entry_g.delete(0,END)
            self.b3_entry_e_u.delete(0,END)
            self.b3_entry_e_z.delete(0,END)
            self.b3_entry_0_u.delete(0,END)
            self.b3_entry_0_z.delete(0,END)
            self.b3_entry_1_u.delete(0,END)
            self.b3_entry_1_z.delete(0,END)
            self.b3_entry_t_u.delete(0,END)
            self.b3_entry_t_z.delete(0,END)
            self.b3_entry_b_u.delete(0,END)
            self.b3_entry_b_z.delete(0,END)
            self.b3_entry_0_r.delete(0,END)
            self.b3_entry_1_r.delete(0,END)
            self.b3_entry_b0.delete(0,END)
            self.b3_entry_b1.delete(0,END)
            self.b3_entry_0_g.delete(0,END)
            self.b3_entry_1_g.delete(0,END)
            self.f_entry_t_u.delete(0,END)
            self.f_entry_t_z.delete(0,END)
            self.f_entry_b_u.delete(0,END)
            self.f_entry_b_z.delete(0,END)
            self.f_entry_le_r.delete(0,END)
            self.f_entry_hte.delete(0,END)
            self.f_entry_b0.delete(0,END)
            self.f_entry_b1.delete(0,END)
            self.f_entry_0_g.delete(0,END)
            self.f_entry_1_g.delete(0,END)
            self.listbox.delete(0,END)

    #   openFile
    #   - Creates a dialog box to prompt the user for a file name
    #   - Not applicable for all geometric methods
    def openFile(self):
        ftypes = [('All files', '*')]
        dlg = filedialog.Open(self, filetypes = ftypes)
        fl = dlg.show()
        if fl != '':
            text = self.readFile(fl)

    #   readFile
    #   - Opens a text file, reading its data and inserting it into its appropriate field
    #   - Not applicable for all geometric methods
    def readFile(self, filename):
        try:
            f = open(filename, "r")
            text = f.readlines()
            value = []
            for line in text:
                value.append(line.split())
            newvalue = [item[2] for item in value]
            method = self.method.current()
            if(method == 0):
                self.b1_entry_e_u.delete(0,END)
                self.b1_entry_e_u.insert(0,newvalue[0])
                self.b1_entry_e_z.delete(0,END)
                self.b1_entry_e_z.insert(0,newvalue[1])
                self.b1_entry_0_u.delete(0,END)
                self.b1_entry_0_u.insert(0,newvalue[2])
                self.b1_entry_0_z.delete(0,END)
                self.b1_entry_0_z.insert(0,newvalue[3])
                self.b1_entry_1_u.delete(0,END)
                self.b1_entry_1_u.insert(0,newvalue[4])
                self.b1_entry_1_z.delete(0,END)
                self.b1_entry_1_z.insert(0,newvalue[5])
                self.b1_entry_r.delete(0,END)
                self.b1_entry_r.insert(0,newvalue[6])
                self.b1_entry_b0.delete(0,END)
                self.b1_entry_b0.insert(0,newvalue[7])
                self.b1_entry_b1.delete(0,END)
                self.b1_entry_b1.insert(0,newvalue[8])
                self.b1_entry_g.delete(0,END)
                self.b1_entry_g.insert(0,newvalue[9])
            elif(method == 1):
                self.b2_entry_e_u.delete(0,END)
                self.b2_entry_e_u.insert(0,newvalue[0])
                self.b2_entry_e_z.delete(0,END)
                self.b2_entry_e_z.insert(0,newvalue[1])
                self.b2_entry_0_u.delete(0,END)
                self.b2_entry_0_u.insert(0,newvalue[2])
                self.b2_entry_0_z.delete(0,END)
                self.b2_entry_0_z.insert(0,newvalue[3])
                self.b2_entry_1_u.delete(0,END)
                self.b2_entry_1_u.insert(0,newvalue[4])
                self.b2_entry_1_z.delete(0,END)
                self.b2_entry_1_z.insert(0,newvalue[5])
                self.b2_entry_t_u.delete(0,END)
                self.b2_entry_t_u.insert(0,newvalue[6])
                self.b2_entry_t_z.delete(0,END)
                self.b2_entry_t_z.insert(0,newvalue[7])
                self.b2_entry_b_u.delete(0,END)
                self.b2_entry_b_u.insert(0,newvalue[8])
                self.b2_entry_b_z.delete(0,END)
                self.b2_entry_b_z.insert(0,newvalue[9])
                self.b2_entry_r.delete(0,END)
                self.b2_entry_r.insert(0,newvalue[10])
                self.b2_entry_b0.delete(0,END)
                self.b2_entry_b0.insert(0,newvalue[11])
                self.b2_entry_b1.delete(0,END)
                self.b2_entry_b1.insert(0,newvalue[12])
                self.b2_entry_g.delete(0,END)
                self.b2_entry_g.insert(0,newvalue[13])
            elif(method == 2):
                self.b3_entry_e_u.delete(0,END)
                self.b3_entry_e_u.insert(0,newvalue[0])
                self.b3_entry_e_z.delete(0,END)
                self.b3_entry_e_z.insert(0,newvalue[1])
                self.b3_entry_0_u.delete(0,END)
                self.b3_entry_0_u.insert(0,newvalue[2])
                self.b3_entry_0_z.delete(0,END)
                self.b3_entry_0_z.insert(0,newvalue[3])
                self.b3_entry_1_u.delete(0,END)
                self.b3_entry_1_u.insert(0,newvalue[4])
                self.b3_entry_1_z.delete(0,END)
                self.b3_entry_1_z.insert(0,newvalue[5])
                self.b3_entry_t_u.delete(0,END)
                self.b3_entry_t_u.insert(0,newvalue[6])
                self.b3_entry_t_z.delete(0,END)
                self.b3_entry_t_z.insert(0,newvalue[7])
                self.b3_entry_b_u.delete(0,END)
                self.b3_entry_b_u.insert(0,newvalue[8])
                self.b3_entry_b_z.delete(0,END)
                self.b3_entry_b_z.insert(0,newvalue[9])
                self.b3_entry_0_r.delete(0,END)
                self.b3_entry_0_r.insert(0,newvalue[10])
                self.b3_entry_1_r.delete(0,END)
                self.b3_entry_1_r.insert(0,newvalue[11])
                self.b3_entry_b0.delete(0,END)
                self.b3_entry_b0.insert(0,newvalue[12])
                self.b3_entry_b1.delete(0,END)
                self.b3_entry_b1.insert(0,newvalue[13])
                self.b3_entry_0_g.delete(0,END)
                self.b3_entry_0_g.insert(0,newvalue[14])
                self.b3_entry_1_g.delete(0,END)
                self.b3_entry_1_g.insert(0,newvalue[15])
            elif(method == 3):
                tkinter.messagebox.showinfo("Error", "Sorry, open file is not supported for
parametric cubic spline.")
            elif(method == 4):
                self.f_entry_t_u.delete(0,END)
                self.f_entry_t_u.insert(0,newvalue[0])
                self.f_entry_t_z.delete(0,END)
                self.f_entry_t_z.insert(0,newvalue[1])
                self.f_entry_b_u.delete(0,END)
                self.f_entry_b_u.insert(0,newvalue[2])
                self.f_entry_b_z.delete(0,END)
                self.f_entry_b_z.insert(0,newvalue[3])
                self.f_entry_le_r.delete(0,END)
                self.f_entry_le_r.insert(0,newvalue[4])
                self.f_entry_hte.delete(0,END)
                self.f_entry_hte.insert(0,newvalue[5])
                self.f_entry_b0.delete(0,END)
                self.f_entry_b0.insert(0,newvalue[6])
                self.f_entry_b1.delete(0,END)
                self.f_entry_b1.insert(0,newvalue[7])
                self.f_entry_0_g.delete(0,END)
                self.f_entry_0_g.insert(0,newvalue[8])
                self.f_entry_1_g.delete(0,END)
                self.f_entry_1_g.insert(0,newvalue[9])
            elif(method == 5):
                tkinter.messagebox.showinfo("Error", "Sorry, open file is not supported for 3D
parametric fourier.")
        except:
            tkinter.messagebox.showinfo("Error", "Error loading file.")

    #   saveFileAs
    #   - Opens a file dialog to prompt user for the file name, then writes the appropriate data
to a text file
    #   - Not applicable for all geometric methods
    def saveFileAs(self):
        method = self.method.current()
        if(method == 0):
            self.f = filedialog.asksaveasfile(mode='w', defaultextension=".txt")
            self.f.write("leading_edge_u = " + self.b1_entry_e_u.get())
            self.f.write("\nleading_edge_z = " + self.b1_entry_e_z.get())
            self.f.write("\nupper_trailing_edge_u = " + self.b1_entry_0_u.get())
            self.f.write("\nupper_trailing_edge_z = " + self.b1_entry_0_z.get())
            self.f.write("\nlower_trailing_edge_u = " + self.b1_entry_1_u.get())
            self.f.write("\nlower_trailing_edge_z = " + self.b1_entry_1_z.get())
            self.f.write("\nradius = " + self.b1_entry_r.get())
            self.f.write("\nbeta_upper_angle = " + self.b1_entry_b0.get())
            self.f.write("\nbeta_lower_angle = " + self.b1_entry_b1.get())
            self.f.write("\ngamma = " + self.b1_entry_g.get())
            self.f.close()
        elif(method == 1):
            self.f = filedialog.asksaveasfile(mode='w', defaultextension=".txt")
            self.f.write("leading_edge_u = " + self.b2_entry_e_u.get())
            self.f.write("\nleading_edge_z = " + self.b2_entry_e_z.get())
            self.f.write("\nupper_trailing_edge_u = " + self.b2_entry_0_u.get())
            self.f.write("\nupper_trailing_edge_z = " + self.b2_entry_0_z.get())
            self.f.write("\nlower_trailing_edge_u = " + self.b2_entry_1_u.get())
            self.f.write("\nlower_trailing_edge_z = " + self.b2_entry_1_z.get())
            self.f.write("\ntop_point_u = " + self.b2_entry_t_u.get())
            self.f.write("\ntop_point_z = " + self.b2_entry_t_z.get())
            self.f.write("\nbottom_point_u = " + self.b2_entry_b_u.get())
            self.f.write("\nbottom_point_z = " + self.b2_entry_b_z.get())
            self.f.write("\nradius = " + self.b2_entry_r.get())
            self.f.write("\nbeta_upper_angle = " + self.b2_entry_b0.get())
            self.f.write("\nbeta_lower_angle = " + self.b2_entry_b1.get())
            self.f.write("\ngamma = " + self.b2_entry_g.get())
            self.f.close()
        elif(method == 2):
            self.f = filedialog.asksaveasfile(mode='w', defaultextension=".txt")
            self.f.write("leading_edge_u = " + self.b3_entry_e_u.get())
            self.f.write("\nleading_edge_z = " + self.b3_entry_e_z.get())
            self.f.write("\nupper_trailing_edge_u = " + self.b3_entry_0_u.get())
            self.f.write("\nupper_trailing_edge_z = " + self.b3_entry_0_z.get())
            self.f.write("\nlower_trailing_edge_u = " + self.b3_entry_1_u.get())
            self.f.write("\nlower_trailing_edge_z = " + self.b3_entry_1_z.get())
            self.f.write("\ntop_point_u = " + self.b3_entry_t_u.get())
            self.f.write("\ntop_point_z = " + self.b3_entry_t_z.get())
            self.f.write("\nbottom_point_u = " + self.b3_entry_b_u.get())
            self.f.write("\nbottom_point_z = " + self.b3_entry_b_z.get())
            self.f.write("\nupper_radius = " + self.b3_entry_0_r.get())
            self.f.write("\nlower_radius = " + self.b3_entry_1_r.get())
            self.f.write("\nbeta_upper_angle = " + self.b3_entry_b0.get())
            self.f.write("\nbeta_lower_angle = " + self.b3_entry_b1.get())
            self.f.write("\nupper_gamma = " + self.b3_entry_0_g.get())
            self.f.write("\nlower_gamma = " + self.b3_entry_1_g.get())
            self.f.close()
        elif(method == 3):
            tkinter.messagebox.showinfo("Error", "Sorry, save is not supported for parametric
cubic spline.")
        elif(method == 4):
            self.f = filedialog.asksaveasfile(mode='w', defaultextension=".txt")
            self.f.write("\ntop_point_u = " + self.f_entry_t_u.get())
            self.f.write("\ntop_point_z = " +self.f_entry_t_z.get())
            self.f.write("\nbottom_point_u = " + self.f_entry_b_u.get())
            self.f.write("\nbottom_point_z = " + self.f_entry_b_z.get())
            self.f.write("\nleading_edge_radius = " + self.f_entry_le_r.get())
            self.f.write("\nhalf_trailing_edge = " + self.f_entry_hte.get())
            self.f.write("\nbeta_upper_angle = " + self.f_entry_b0.get())
            self.f.write("\nbeta_lower_angle = " + self.f_entry_b1.get())
            self.f.write("\nupper_gamma = " + self.f_entry_0_g.get())
            self.f.write("\nlower_gamma = " + self.f_entry_1_g.get())
            self.f.close()
        elif(method == 5):
            tkinter.messagebox.showinfo("Error", "Sorry, save is not supported for 3D parametric
fourier.")

    #   saveFile
    #   - Opens a file dialog to prompt user for the file name, then writes the appropriate data
to a text file
    #   - Not applicable for all geometric methods
    def saveFile(self):
        method = self.method.current()
        if(method == 0):
            newfile = open(self.f.name, "w")
            newfile.write("leading_edge_u = " + self.b1_entry_e_u.get())
            newfile.write("\nleading_edge_z = " + self.b1_entry_e_z.get())
            newfile.write("\nupper_trailing_edge_u = " + self.b1_entry_0_u.get())
            newfile.write("\nupper_trailing_edge_z = " + self.b1_entry_0_z.get())
            newfile.write("\nlower_trailing_edge_u = " + self.b1_entry_1_u.get())
            newfile.write("\nlower_trailing_edge_z = " + self.b1_entry_1_z.get())
            newfile.write("\nradius = " + self.b1_entry_r.get())
            newfile.write("\nbeta_upper_angle = " + self.b1_entry_b0.get())
            newfile.write("\nbeta_lower_angle = " + self.b1_entry_b1.get())
            newfile.write("\ngamma = " + self.b1_entry_g.get())
            tkinter.messagebox.showinfo("Saved", "Saved successfully.")
            newfile.close()
        elif(method == 1):
            newfile = open(self.f.name, "w")
            newfile.write("leading_edge_u = " + self.b2_entry_e_u.get())
            newfile.write("\nleading_edge_z = " + self.b2_entry_e_z.get())
            newfile.write("\nupper_trailing_edge_u = " + self.b2_entry_0_u.get())
            newfile.write("\nupper_trailing_edge_z = " + self.b2_entry_0_z.get())
            newfile.write("\nlower_trailing_edge_u = " + self.b2_entry_1_u.get())
            newfile.write("\nlower_trailing_edge_z = " + self.b2_entry_1_z.get())
            newfile.write("\ntop_point_u = " + self.b2_entry_t_u.get())
            newfile.write("\ntop_point_z = " + self.b2_entry_t_z.get())
            newfile.write("\nbottom_point_u = " + self.b2_entry_b_u.get())
            newfile.write("\nbottom_point_z = " + self.b2_entry_b_z.get())
            newfile.write("\nradius = " + self.b2_entry_r.get())
            newfile.write("\nbeta_upper_angle = " + self.b2_entry_b0.get())
            newfile.write("\nbeta_lower_angle = " + self.b2_entry_b1.get())
            newfile.write("\ngamma = " + self.b2_entry_g.get())
            tkinter.messagebox.showinfo("Saved", "Saved successfully.")
            newfile.close()
        elif(method == 2):
            newfile = open(self.f.name, "w")
            newfile.write("leading_edge_u = " + self.b3_entry_e_u.get())
            newfile.write("\nleading_edge_z = " + self.b3_entry_e_z.get())
            newfile.write("\nupper_trailing_edge_u = " + self.b3_entry_0_u.get())
            newfile.write("\nupper_trailing_edge_z = " + self.b3_entry_0_z.get())
            newfile.write("\nlower_trailing_edge_u = " + self.b3_entry_1_u.get())
            newfile.write("\nlower_trailing_edge_z = " + self.b3_entry_1_z.get())
            newfile.write("\ntop_point_u = " + self.b3_entry_t_u.get())
            newfile.write("\ntop_point_z = " + self.b3_entry_t_z.get())
            newfile.write("\nbottom_point_u = " + self.b3_entry_b_u.get())
            newfile.write("\nbottom_point_z = " + self.b3_entry_b_z.get())
            newfile.write("\nupper_radius = " + self.b3_entry_0_r.get())
            newfile.write("\nlower_radius = " + self.b3_entry_1_r.get())
            newfile.write("\nbeta_upper_angle = " + self.b3_entry_b0.get())
            newfile.write("\nbeta_lower_angle = " + self.b3_entry_b1.get())
            newfile.write("\nupper_gamma = " + self.b3_entry_0_g.get())
            newfile.write("\nlower_gamma = " + self.b3_entry_1_g.get())
            tkinter.messagebox.showinfo("Saved", "Saved successfully.")
            newfile.close()
        elif(method == 3):
            tkinter.messagebox.showinfo("Error", "Sorry, save is not supported for parametric
cubic spline.")
        elif(method == 4):
            newfile = open(self.f.name, "w")
            newfile.write("\ntop_point_u = " + self.f_entry_t_u.get())
            newfile.write("\ntop_point_z = " +self.f_entry_t_z.get())
            newfile.write("\nbottom_point_u = " + self.f_entry_b_u.get())
            newfile.write("\nbottom_point_z = " + self.f_entry_b_z.get())
            newfile.write("\nleading_edge_radius = " + self.f_entry_le_r.get())
            newfile.write("\nhalf_trailing_edge = " + self.f_entry_hte.get())
            newfile.write("\nbeta_upper_angle = " + self.f_entry_b0.get())
            newfile.write("\nbeta_lower_angle = " + self.f_entry_b1.get())
            newfile.write("\nupper_gamma = " + self.f_entry_0_g.get())
            newfile.write("\nlower_gamma = " + self.f_entry_1_g.get())
            tkinter.messagebox.showinfo("Saved", "Saved successfully.")
            newfile.close()
        elif(method == 5):
            tkinter.messagebox.showinfo("Error", "Sorry, save is not supported for 3D parametric
fourier.")

    #   export
    #   - Checks if a method has been selected to call exportToBlender
    def export(self):
        if(self.method.get()=='Select a method here.'):
            tkinter.messagebox.showinfo("Error", "Choose a method first.")
        else:
            self.exportToBlender()

    #   plotOrientation
    #   - Changes the orientation of the plot (left or right side of the application)
    def plotOrientation(self):
        if(self.inputsF.pack_info()['side'] == 'right'):
            self.inputsF.pack(side="left", fill="both", expand=True)
        else:
            self.inputsF.pack(side="right", fill="both", expand=True)

    #   scrollbar
    #   - Configures a scrollbar to be used on a frame
    def scrollbar(self, event):
        self.canvas.configure(scrollregion = self.canvas.bbox("all"), width = 375, height = 470)

    #   help
    #   - Creates a new scrollable window with information on each geometric methods
    def help(self):
        w = 397
        h = 475
        newWindow = Tk()
        newWindow.wm_title("Help")
        ws = newWindow.winfo_screenwidth()
        hs = newWindow.winfo_screenheight()
        x = (ws - w) / 2
        y = (hs - h) / 2
        newWindow.geometry('%dx%d+%d+%d' % (w, h, x, y))
        myframe = Frame(newWindow, relief = GROOVE, width = 100, height = 100, bd = 1)
        myframe.place(x = 0, y = 0)
        self.canvas = Canvas(myframe, background = "#ffffff")
        frame = Frame(self.canvas, background = "#ffffff")
        myscrollbar = ttk.Scrollbar(myframe, orient="vertical", command=self.canvas.yview)
        self.canvas.configure(yscrollcommand = myscrollbar.set)
        myscrollbar.pack(side = "right", fill = "y")
        self.canvas.pack(side = "left")
        self.canvas.create_window((0,0), window = frame, anchor = 'nw')
        Label(frame, justify = "center", font = "Helvetica 24 bold underline", background =
"#ffffff",
              text = "Help").pack(side = "top", padx = 10, pady= 10)
        Label(frame, justify = "center", background = "#ffffff", fg = "#336699",
              font = "Helvetica 12 bold", text = "\nParametric Polynomial").pack(side = "top",
padx = 10, pady= 10)
        Label(frame, justify = "left", background = "#ffffff",
              text="The parametric polynomial allows you to specify values at key"
                   "\npoints in the airfoil such as leading edge radius, afterbody   "
                   "\nangles, and top and bottom point maximums. There are 3   "
                   "\ndifferent boundary conditions. Selecting different boundary   "
                   "\nconditions will allow you more and different key point options.  "
                   "\nEnter your values and click 'Apply Settings' to"
                   " generate the plot. ").pack(side = "top", padx = 10, pady= 10)
        Label(frame, justify = "center", background = "#ffffff", fg = "#336699",
              font = "Helvetica 12 bold", text = "\nParametric Cubic Spline").pack(side = "top",
padx = 10, pady= 10)
        Label(frame, justify = "left", background = "#ffffff",
              text="The parametric cubic spline allows you to specify a number of  "
                   "\npoints around the airfoil. There are already default points "
                   "\nprogrammed in for you to use. First you need to choose the "
                   "\nshape of the left and right end constraints of the airfoil by"
                   "\nselecting from the drop-down menu. You can add more points "
                   "\nby clicking the 'Add Point' button. You can delete points by "
                   "\nhighlighting the set of points you want to delete and clicking "
                   "\nthe 'Delete Point' button. Once you have the points you want to "
                   "\nuse, click 'Apply Settings' to generate the plot.").pack(side = "top",
padx = 10, pady= 10)
        Label(frame, justify = "center", background = "#ffffff", fg = "#336699",
              font = "Helvetica 12 bold", text = "\nParametric Fourier").pack(side = "top", padx
= 10, pady= 10)
        Label(frame, justify = "left", background = "#ffffff",
              text="The 2D parametric fourier allows you to define 8 values: "
                   "\n top point, bottom point, leading edge radius, half trailing edge,"
                   "\nbeta upper angle, beta lower angle, upper gamma, and lower "
                   "\ngamma to fine tune the upper and lower halves of the airfoil. "
                   "\nEnter your values and click ‘Apply Settings’ to generate the plot."
                   "\n\nThe 3D parametric Fourier allows you to define the values for "
                   "\nmultiple airfoils (stations), across the span of a wing. An object "
                   "\nis then generated with these airfoils connected. Note that the "
                   "\nairfoil is connected to the next airfoil as it goes down the list, so "
                   "\nthe order in which you add the stations to the list will determine"
                   "\nthe order in which they are generated and connected. There is a "
                   "\nset of preset stations as an example. You can add stations by "
                   "\n clicking ‘add station’, edit stations by clicking ‘edit station’, and "
                   "\ndelete stations by selecting a station and clicking ‘delete station'. "
                   " \nClick ‘Plot Selected Airfoil’ to plot the "
                   "selected airfoil.").pack(side = "top", padx = 10, pady= 10)
        Label(frame, justify = "center", background = "#ffffff", fg = "#336699",
              font = "Helvetica 12 bold", text = "\nOpening/Saving .txt Files").pack(side =
"top", padx = 10, pady= 10)
        Label(frame, justify = "left", background = "#ffffff",
              text="Opening/saving .txt files is only supported for the parametric  "
                   "\npolynomial and the 2D parametric fourier. Download the 4 "
                   "\n.txt files from our website."
                   "\n\nTo open:"
                   "\n1.) Select the generation method you want to use."
                   "\n2.) Go to ‘File’."
                   "\n3.) Select ‘Open’."
                   "\n4.) Select the corresponding .txt file from wherever you have it "
                   "\nsaved and click ‘Open’. The values from the file should be "
                   "\nimported."
                   "\n5.) Click ‘Apply Settings’ to generate the plot."
                   "\n\nTo save any points you have entered:"
                   "\n1.) Go to ‘File’."
                   "\n2.) Select ‘Save As’."
                   "\n3.) Enter a name and click ‘Save’."
                   " \n\nOnce you have created your .txt file, you can continue working."
                   "\nIf you want to save again, go to ‘File’ "
                   "and select ‘Save’ this time.").pack(side = "top", padx = 10, pady= 10)
        Label(frame, justify = "center", background = "#ffffff", fg = "#336699",
              font = "Helvetica 12 bold", text = "\nConnecting/Exporting"
                                                " to Blender").pack(side = "top", padx = 10,
pady= 10)
        Label(frame, justify = "left", background = "#ffffff",
              text="Once you have your airfoil generated with your preferred  "
                "\ngeneration method and you want to export to Blender, follow "
                "\nthese steps before hitting ‘Export to Blender’:"
                "\n\n1.) Download the ‘testserver.py’ file from our website. "
                "\n2.) Open up Blender"
                "\n3.) Click on the 'Choose Screen Layout' button and select"
                "\n‘Scripting’."
                "\n4.) Click ‘New’ to create a new text data block."
                "\n5.) Paste the ‘testserver.py’ code into the space. "
                "\n6.) Click ‘Run Script’. "
                "\nNote: If you are using a small computer screen, you may "
                "\nhave to drag the scripting window to see the ‘Run Script’"
                "\nbutton. "
                "\n7.) Click on the ‘Choose Screen Layout’ button again and select "
                " \n‘Default’. You should now see ‘Aircraft Generator’ in the left "
                "\nmodule. It should say ‘Status: Not Linked’."
                "\n8.) Click ‘Link to Aircraft Generator’. It should change to"
                "\n‘Status: Linked’. "
                "\n9.) Come back to this Wing Generator application and click"
                "\n‘Export to Blender’. "
                "\n10.) Go back to Blender. You should "
                "now see the generated model.").pack(side = "top", padx = 10, pady= 10)
        Label(frame, justify = "center", background = "#ffffff", fg = "#336699",
              text = "\nFor more detailed help, refer to our user manual.").pack(side = "top",
padx = 10, pady= 10)
        frame.bind("<Configure>", self.scrollbar)

    #   about
    #   - Creates a new window with general information on the project
    def about(self):
        w = 300
        h = 150
        newWindow = Tk()
        newWindow.wm_title("About")
        l = Label(newWindow, text="CSULB Northrop Grumman Student Design Project\n"
                                  "Wing Generator v. 1.0 \nwww.csulb-ngcproject.me\nSpring 2015")
        l.pack(side = "left", fill = BOTH, padx = 10)
        ws = newWindow.winfo_screenwidth()
        hs = newWindow.winfo_screenheight()
        x = (ws - w) / 2
        y = (hs - h) / 2
        newWindow.geometry('%dx%d+%d+%d' % (w, h, x, y))

    #   newsettings
    #   - Checks the current value of the method combobox, changing the panel to appropriately
    def newsettings(self, event):
        value = self.method.current()
        self.resizeWindow()
        if(value == 0):
            self.settingsLF0.grid_forget()
            self.settingsLF2.grid_forget()
            self.settingsLF3.grid_forget()
            self.settingsLF4.grid_forget()
            self.settingsLF5.grid_forget()
            self.settingsLF6.grid_forget()
            self.settingsLF1.grid(in_=self.inputsF, column=0, row=1, sticky=W+E)
            self.settingsLF1.grid_columnconfigure(0, weight=1)
            self.applyB.configure(text="Apply Settings")
        elif(value == 1):
            self.settingsLF0.grid_forget()
            self.settingsLF1.grid_forget()
            self.settingsLF3.grid_forget()
            self.settingsLF4.grid_forget()
            self.settingsLF5.grid_forget()
            self.settingsLF6.grid_forget()
            self.settingsLF2.grid(in_=self.inputsF, column=0, row=1, sticky=W+E)
            self.settingsLF2.grid_columnconfigure(0, weight=1)
            self.applyB.configure(text="Apply Settings")
        elif(value == 2):
            self.settingsLF0.grid_forget()
            self.settingsLF1.grid_forget()
            self.settingsLF2.grid_forget()
            self.settingsLF4.grid_forget()
            self.settingsLF5.grid_forget()
            self.settingsLF6.grid_forget()
            self.settingsLF3.grid(in_=self.inputsF, column=0, row=1, sticky=W+E)
            self.settingsLF3.grid_columnconfigure(0, weight=1)
            self.applyB.configure(text="Apply Settings")
        elif(value == 3):
            self.settingsLF0.grid_forget()
            self.settingsLF1.grid_forget()
            self.settingsLF2.grid_forget()
            self.settingsLF3.grid_forget()
            self.settingsLF5.grid_forget()
            self.settingsLF6.grid_forget()
            self.settingsLF4.grid(in_=self.inputsF, column=0, row=2, sticky=W+E)
            self.settingsLF4.grid_columnconfigure(0, weight=1)
            self.applyB.configure(text="Apply Settings")
        elif(value == 4):
            self.settingsLF0.grid_forget()
            self.settingsLF1.grid_forget()
            self.settingsLF2.grid_forget()
            self.settingsLF3.grid_forget()
            self.settingsLF4.grid_forget()
            self.settingsLF6.grid_forget()
            self.settingsLF5.grid(in_=self.inputsF, column=0, row=2, sticky=W+E)
            self.settingsLF5.grid_columnconfigure(0, weight=1)
            self.applyB.configure(text="Apply Settings")
        elif(value == 5):
            self.settingsLF0.grid_forget()
            self.settingsLF1.grid_forget()
            self.settingsLF2.grid_forget()
            self.settingsLF3.grid_forget()
            self.settingsLF4.grid_forget()
            self.settingsLF5.grid_forget()
            self.settingsLF6.grid(in_=self.inputsF, column=0, row=2, sticky=W+E)
            self.settingsLF6.grid_columnconfigure(0, weight=1)
            self.applyB.configure(text="Plot Selected Station")

    #   apply
    #   - Checks the current value of the method combobox, calling the appropriate geometric
generation method and
    #     drawing the results, throws an Error Dialog if the input values are not acceptable
    def apply(self):
        if(self.method.current()=='Select a method here.'):
            tkinter.messagebox.showerror("Error", "Choose a method first.")
        elif(self.method.get()=='2D - Parametric Polynomial - Boundary Condition 1'):
            try:
                X =pp.boundaryOne(float(self.b1_entry_e_u.get()), float(self.b1_entry_e_z.get()),
                float(self.b1_entry_0_u.get()), float(self.b1_entry_0_z.get()),
float(self.b1_entry_1_u.get()),
                float(self.b1_entry_1_z.get()), float(self.b1_entry_r.get()),
float(self.b1_entry_b0.get()),
                float(self.b1_entry_b1.get()), float(self.b1_entry_g.get()))
                self.draw(X)
            except ValueError:
                tkinter.messagebox.showerror("Error", "Entries must consist of numerical values
and must not be blank.")
        elif(self.method.get()=='2D - Parametric Polynomial - Boundary Condition 2'):
            try:
                X =pp.boundaryTwo(float(self.b2_entry_e_u.get()), float(self.b2_entry_e_z.get()),
                float(self.b2_entry_0_u.get()), float(self.b2_entry_0_z.get()),
float(self.b2_entry_1_u.get()),
                float(self.b2_entry_1_z.get()), float(self.b2_entry_r.get()),
float(self.b2_entry_b0.get()),
                float(self.b2_entry_b1.get()), float(self.b2_entry_g.get()),
float(self.b2_entry_t_u.get()),
                float(self.b2_entry_t_z.get()), float(self.b2_entry_b_u.get()),
float(self.b2_entry_b_z.get()) )
                self.draw(X)
            except ValueError:
                tkinter.messagebox.showerror("Error", "Entries must consist of numerical values
and must not be blank.")
        elif(self.method.get()=='2D - Parametric Polynomial - Boundary Condition 3'):
            try:
                X = pp.boundaryThree(float(self.b3_entry_e_u.get()),
float(self.b3_entry_e_z.get()),
                float(self.b3_entry_0_u.get()), float(self.b3_entry_0_z.get()),
float(self.b3_entry_1_u.get()),
                float(self.b3_entry_1_z.get()), float(self.b3_entry_t_u.get()),
float(self.b3_entry_t_z.get()),
                float(self.b3_entry_b_u.get()), float(self.b3_entry_b_z.get()),
float(self.b3_entry_0_r.get()),
                float(self.b3_entry_1_r.get()), float(self.b3_entry_b0.get()),
float(self.b3_entry_b1.get()),
                float(self.b3_entry_0_g.get()), float(self.b3_entry_1_g.get()))
                self.draw(X)
            except ValueError:
                tkinter.messagebox.showerror("Error", "Entries must consist of numerical values
and must not be blank.")
        elif(self.method.get()=='2D - Parametric Cubic Spline'):
            if (self.leftConstraint.current() == 0 or self.rightConstraint.current() == 0):
                tkinter.messagebox.showerror("Error", "Select constraints.")
            else:
                try:
                    points = []
                    tau = ""
                    zeta = ""
                    points = self.listbox.get(0,END)
                    for i in range(0, len(points)):
                        temp = points[i].split(',')
                        if(i < len(points)-1):
                            tau = tau + (temp[0]) + ','
                            zeta = zeta + (temp[1]) + ','
                        else:
                            tau = tau + (temp[0])
                            zeta = zeta + (temp[1])
                    X = pcs.parametricCubicSpline(tau, zeta, self.leftConstraint.current(),
                                                  self.rightConstraint.current())
                    self.draw(X)
                except:
                    tkinter.messagebox.showerror("Error", "Add points.")
        elif(self.method.get()=='2D - Parametric Fourier'):
            try:
                X = pf.parametricFourier(float(self.f_entry_0_g.get()),
float(self.f_entry_1_g.get()),
                float(self.f_entry_le_r.get()), float(self.f_entry_t_u.get()),
float(self.f_entry_t_z.get()),
                float(self.f_entry_b_u.get()), float(self.f_entry_b_z.get()),
float(self.f_entry_b0.get()),
                float(self.f_entry_b1.get()), float(self.f_entry_hte.get()))
                self.draw(X)
            except ValueError:
                tkinter.messagebox.showerror("Error", "Entries must consist of numerical values
and must not be blank.")
        elif(self.method.get()=='3D - Parametric Fourier'):
            try:
                X = pf.parametricFourier(float(self.ug[self.curIndexF]),
float(self.lg[self.curIndexF]),
                float(self.ler[self.curIndexF]), float(self.topPointU[self.curIndexF]),
                float(self.topPointZ[self.curIndexF]),
                float(self.bottomPointU[self.curIndexF]),
float(self.bottomPointZ[self.curIndexF]),
                float(self.bu[self.curIndexF]),
                float(self.bl[self.curIndexF]), float(self.hte[self.curIndexF]))
                self.draw(X)
            except:
                tkinter.messagebox.showerror("Error", "Select a station to plot.")
```

```
    #   draw
    #   - Checks the current value of the method combobox, doing the appropriate calculations for
coordinates for
    #     the corresponding method, coordinates must be changed from cartesian to the graphical
window
    def draw(self, X):
        value = self.method.current()
        if(value == 0):
            self.plot.delete("all")
            self.plot.create_rectangle(1, 1, 400, 200)
            previousX = 1-(1-float(self.b1_entry_g.get()))*math.sin(math.pi*0)\
                            +float(self.b1_entry_g.get())*math.sin(3*math.pi*0)
            previousY = float(self.b1_entry_0_z.get())*(1-2*0) + X[0]*0 + X[1]*math.pow(0,2) +
X[2]*math.pow(0,3) \
                            + X[3]*math.pow(0,4) + X[4]*math.pow(0,5)
            previousX = 400 * previousX
            previousY = -200 * previousY + 100
            for i in range(1, 41):
                u = i/40
                myCanvasX = 1-(1-float(self.b1_entry_g.get()))*math.sin(math.pi*u)\
                            +float(self.b1_entry_g.get())*math.sin(3*math.pi*u)
                myCanvasY = float(self.b1_entry_0_z.get())*(1-2*u) + X[0]*u + X[1]*math.pow(u,2)
+ X[2]*math.pow(u,3) \
                            + X[3]*math.pow(u,4) + X[4]*math.pow(u,5)
                #Change from MyCanvas coordinate to Window coordinate
                windowX = 400 * myCanvasX
                windowY = -200 * myCanvasY + 100
                self.plot.create_line(windowX, windowY, previousX, previousY)
                previousX = windowX
                previousY = windowY
        elif(value == 1):
            self.plot.delete("all")
            self.plot.create_rectangle(1, 1, 400, 200)
            previousX = 1-(1-float(self.b2_entry_g.get()))*math.sin(math.pi*0) \
                        + float(self.b2_entry_g.get())*math.sin(3*math.pi*0)
            previousY = float(self.b2_entry_0_z.get())*(1-2*0) + X[0]*0 + X[1]*math.pow(0,2) \
                        + X[2]*math.pow(0,3) + X[3]*math.pow(0,4) + X[4]*math.pow(0,5) +
X[5]*math.pow(0,6) \
                        + X[6]*math.pow(0,7)
            previousX = 400 * previousX
            previousY = -200 * previousY + 100
            for i in range(1, 41):
                u = i/40
                myCanvasX = 1-(1-float(self.b2_entry_g.get()))*math.sin(math.pi*u) \
                            + float(self.b2_entry_g.get())*math.sin(3*math.pi*u)
                myCanvasY = float(self.b2_entry_0_z.get())*(1-2*u) + X[0]*u + X[1]*math.pow(u,2)
\
                            + X[2]*math.pow(u,3) + X[3]*math.pow(u,4) + X[4]*math.pow(u,5) +
X[5]*math.pow(u,6) \
                            + X[6]*math.pow(u,7)
                #Change from MyCanvas coordinate to Window coordinate
                windowX = 400 * myCanvasX
                windowY = -200 * myCanvasY + 100
                self.plot.create_line(windowX, windowY, previousX, previousY)
                previousX = windowX
                previousY = windowY
        elif(value == 2):
            self.plot.delete("all")
            self.plot.create_rectangle(1, 1, 400, 200)
            w = 1
            previousX = 1 - (1 - float(self.b3_entry_0_g.get())) * math.cos(w*math.pi/2) \
                        - float(self.b3_entry_0_g.get()) * math.cos(w*3*math.pi/2)
            previousY = float(X[0,0]*w + X[0,1]*math.pow(w,2) + X[0,2]*math.pow(w,3) +
X[0,3]*math.pow(w,4)
                                  + X[0,4]*math.pow(w,5))
            previousX = 400 * previousX
            previousY = -200 * previousY + 100
            for i in range(0, 20):
                u = 0.5*i/19
                w=1-2*u
                myCanvasX = 1 - (1 - float(self.b3_entry_0_g.get())) * math.cos(w*math.pi/2) \
                            - float(self.b3_entry_0_g.get()) * math.cos(w*3*math.pi/2)
                myCanvasY = float(X[0,0]*w + X[0,1]*math.pow(w,2) + X[0,2]*math.pow(w,3) +
X[0,3]*math.pow(w,4)
                                  + X[0,4]*math.pow(w,5))
                #Change from MyCanvas coordinate to Window coordinate
                windowX = 400 * myCanvasX
                windowY = -200 * myCanvasY + 100
                self.plot.create_line(windowX, windowY, previousX, previousY)
                previousX = windowX
                previousY = windowY
            for i in range(0, 20):
                u = 0.5+0.5*i/19
                w=2*u-1
                myCanvasX = 1 - (1 - float(self.b3_entry_1_g.get())) * math.cos(w*math.pi/2) \
                            - float(self.b3_entry_1_g.get()) * math.cos(w*3*math.pi/2)
                myCanvasY = float(X[1,0]*w + X[1,1]*math.pow(w,2) + X[1,2]*math.pow(w,3) +
X[1,3]*math.pow(w,4)
                                  + X[1,4]*math.pow(w,5))
                #Change from MyCanvas coordinate to Window coordinate
                windowX = 400 * myCanvasX
                windowY = -200 * myCanvasY + 100
                self.plot.create_line(windowX, windowY, previousX, previousY)
                previousX = windowX
                previousY = windowY
        elif(value == 3):
            self.plot.delete("all")
            self.plot.create_rectangle(1, 1, 400, 200)
            previousX = 1-(1-0.05)*math.sin(math.pi*0) \
                        + 0.05*math.sin(3*math.pi*0)
            previousY = X[0]
            previousX = 400 * previousX
            previousY = -200 * previousY + 100
            for i in range(1, 41):
                u = i/40
                myCanvasX = 1-(1-0.05)*math.sin(math.pi*u) \
                            + 0.05*math.sin(3*math.pi*u)
                myCanvasY = X[i]

                #Change from MyCanvas coordinate to Window coordinate
                windowX = 400 * myCanvasX
                windowY = -200 * myCanvasY + 100
                self.plot.create_line(windowX, windowY, previousX, previousY)
                previousX = windowX
                previousY = windowY
        elif(value == 4):
            self.plot.delete("all")
            self.plot.create_rectangle(1, 1, 400, 200)
            n = 8
            gam = pf.gam_(0, float(self.f_entry_0_g.get()), float(self.f_entry_1_g.get()))
            ww = 1 - 2 * 0
            previousX = pf.X_of_W(gam, ww)
            previousY = float(self.f_entry_hte.get()) * (1 - 2 * 0)
            previousX = 400 * previousX
            previousY = -200 * previousY + 100
            for i in range(1, 41):
                u = i/40
                gam = pf.gam_(u, float(self.f_entry_0_g.get()), float(self.f_entry_1_g.get()))
                if(u < 0.5):
                    ww = 1 - 2 * u
                else:
                    ww = 2 * u - 1
                xx = pf.X_of_W(gam, ww)
                zz = float(self.f_entry_hte.get()) * (1 - 2 * u)
                for j in range(0, n):
                    remain = 1
                    zz = zz + X[j] * math.sin(remain * math.pi * (j+1) * u)
                windowX = 400 * xx
                windowY = -200 * zz + 100
                self.plot.create_line(windowX, windowY, previousX, previousY)
                previousX = windowX
                previousY = windowY
        elif(value == 5):
            self.plot.delete("all")
            self.plot.create_rectangle(1, 1, 400, 200)
            n = 8
            gam = pf.gam_(0, float(self.ug[self.curIndexF]), float(self.lg[self.curIndexF]))
            ww = 1 - 2 * 0
            previousX = pf.X_of_W(gam, ww)
            previousY = float(self.hte[self.curIndexF]) * (1 - 2 * 0)
            previousX = 400 * previousX
            previousY = -200 * previousY + 100
            for i in range(1, 41):
                u = i/40
                gam = pf.gam_(u, float(self.ug[self.curIndexF]), float(self.lg[self.curIndexF]))
                if(u < 0.5):
                    ww = 1 - 2 * u
                else:
                    ww = 2 * u - 1
                xx = pf.X_of_W(gam, ww)
                zz = float(self.hte[self.curIndexF]) * (1 - 2 * u)
                for j in range(0, n):
                    remain = 1
                    zz = zz + X[j] * math.sin(remain * math.pi * (j+1) * u)
                windowX = 400 * xx
                windowY = -200 * zz + 100
                self.plot.create_line(windowX, windowY, previousX, previousY)
                previousX = windowX
                previousY = windowY

    #   exportToBlender
    #   - Checks the current value of the method combobox, doing the appropriate calculations and
calling the sendData
    #     function
    #   - Note that coordinates to not have to be converted to the window coordinates, as Blender
uses Cartesian
    def exportToBlender(self):
        data = ""
        value = self.method.current()
        if(value == 0):
            try:
                X = pp.boundaryOne(float(self.b1_entry_e_u.get()),
float(self.b1_entry_e_z.get()),
                    float(self.b1_entry_0_u.get()), float(self.b1_entry_0_z.get()),
float(self.b1_entry_1_u.get()),
                    float(self.b1_entry_1_z.get()), float(self.b1_entry_r.get()),
float(self.b1_entry_b0.get()),
                    float(self.b1_entry_b1.get()), float(self.b1_entry_g.get()))
                previousX = 1-(1-float(self.b1_entry_g.get()))*math.sin(math.pi*0)\
                            +float(self.b1_entry_g.get())*math.sin(3*math.pi*0)
                previousY = float(self.b1_entry_0_z.get())*(1-2*0) + X[0]*0 + X[1]*math.pow(0,2)
+ X[2]*math.pow(0,3) \
                            + X[3]*math.pow(0,4) + X[4]*math.pow(0,5)
                data = data + str(previousX) + "," + str(previousY) + ","
                for i in range(1, 41):
                    u = i/40
                    myCanvasX = 1-(1-float(self.b1_entry_g.get()))*math.sin(math.pi*u)\
                                +float(self.b1_entry_g.get())*math.sin(3*math.pi*u)
                    myCanvasY = float(self.b1_entry_0_z.get())*(1-2*u) + X[0]*u \
                                + X[1]*math.pow(u,2) + X[2]*math.pow(u,3) \
                                + X[3]*math.pow(u,4) + X[4]*math.pow(u,5)
                    data = data + str(myCanvasX) + "," + str(myCanvasY) + ","
                data = data + "0" + ","
                self.sendData(data)
            except ValueError:
                tkinter.messagebox.showerror("Error", "Entries must consist of numerical values
and must not be blank.")
        elif(value == 1):
            try:
                X = pp.boundaryTwo(float(self.b2_entry_e_u.get()),
float(self.b2_entry_e_z.get()),
                    float(self.b2_entry_0_u.get()), float(self.b2_entry_0_z.get()),
float(self.b2_entry_1_u.get()),
                    float(self.b2_entry_1_z.get()), float(self.b2_entry_r.get()),
float(self.b2_entry_b0.get()),
                    float(self.b2_entry_b1.get()), float(self.b2_entry_g.get()),
float(self.b2_entry_t_u.get()),
                    float(self.b2_entry_t_z.get()), float(self.b2_entry_b_u.get()),
float(self.b2_entry_b_z.get()) )
                previousX = 1-(1-float(self.b2_entry_g.get()))*math.sin(math.pi*0) \
                            + float(self.b2_entry_g.get())*math.sin(3*math.pi*0)
                previousY = float(self.b2_entry_0_z.get())*(1-2*0) + X[0]*0 + X[1]*math.pow(0,2)
\
                            + X[2]*math.pow(0,3) + X[3]*math.pow(0,4) + X[4]*math.pow(0,5) +
X[5]*math.pow(0,6) \
                            + X[6]*math.pow(0,7)
                data = data + str(previousX) + "," + str(previousY) + ","
                for i in range(1, 41):
                    u = i/40
                    myCanvasX = 1-(1-float(self.b2_entry_g.get()))*math.sin(math.pi*u) \
                                + float(self.b2_entry_g.get())*math.sin(3*math.pi*u)
                    myCanvasY = float(self.b2_entry_0_z.get())*(1-2*u) + X[0]*u +
X[1]*math.pow(u,2) \
                                + X[2]*math.pow(u,3) + X[3]*math.pow(u,4) + X[4]*math.pow(u,5) +
X[5]*math.pow(u,6) \
                                + X[6]*math.pow(u,7)
                    data = data + str(myCanvasX) + "," + str(myCanvasY) + ","
                data = data + "0" + ","
                self.sendData(data)
            except ValueError:
                tkinter.messagebox.showerror("Error", "Entries must consist of numerical values
and must not be blank.")
        elif(value==2):
            try:
                X = pp.boundaryThree(float(self.b3_entry_e_u.get()),
float(self.b3_entry_e_z.get()),
                    float(self.b3_entry_0_u.get()), float(self.b3_entry_0_z.get()),
float(self.b3_entry_1_u.get()),
                    float(self.b3_entry_1_z.get()), float(self.b3_entry_t_u.get()),
float(self.b3_entry_t_z.get()),
                    float(self.b3_entry_b_u.get()), float(self.b3_entry_b_z.get()),
float(self.b3_entry_0_r.get()),
                    float(self.b3_entry_1_r.get()), float(self.b3_entry_b0.get()),
float(self.b3_entry_b1.get()),
                    float(self.b3_entry_0_g.get()), float(self.b3_entry_1_g.get()))
                w = 1
                previousX = 1 - (1 - float(self.b3_entry_0_g.get())) * math.cos(w*math.pi/2) \
                            - float(self.b3_entry_0_g.get()) * math.cos(w*3*math.pi/2)
                previousY = float(X[0,0]*w + X[0,1]*math.pow(w,2) + X[0,2]*math.pow(w,3) +
X[0,3]*math.pow(w,4)
                                  + X[0,4]*math.pow(w,5))
                data = data + str(previousX) + "," + str(previousY) + ","
                for i in range(0, 20):
                    u = 0.5*i/19
                    w=1-2*u
                    myCanvasX = 1 - (1 - float(self.b3_entry_0_g.get())) * math.cos(w*math.pi/2)
\
                                - float(self.b3_entry_0_g.get()) * math.cos(w*3*math.pi/2)
                    myCanvasY = float(X[0,0]*w + X[0,1]*math.pow(w,2) + X[0,2]*math.pow(w,3) +
X[0,3]*math.pow(w,4)
                                      + X[0,4]*math.pow(w,5))
                    data = data + str(myCanvasX) + "," + str(myCanvasY) + ","
                for i in range(0, 20):
                    u = 0.5+0.5*i/19
                    w=2*u-1
                    myCanvasX = 1 - (1 - float(self.b3_entry_1_g.get())) * math.cos(w*math.pi/2)
\
                                - float(self.b3_entry_1_g.get()) * math.cos(w*3*math.pi/2)
                    myCanvasY = float(X[1,0]*w + X[1,1]*math.pow(w,2) + X[1,2]*math.pow(w,3) +
X[1,3]*math.pow(w,4)
                                      + X[1,4]*math.pow(w,5))
                    data = data + str(myCanvasX) + "," + str(myCanvasY) + ","
                data = data + "0" + ","
                self.sendData(data)
            except ValueError:
                tkinter.messagebox.showerror("Error", "Entries must consist of numerical values
and must not be blank.")
        elif(value==3):
            if (self.leftConstraint.current() == 0 or self.rightConstraint.current() == 0):
                tkinter.messagebox.showerror("Error", "Select constraints.")
            else:
                try:
                    points = []
                    tau = ""
                    zeta = ""
                    points = self.listbox.get(0,END)
                    for i in range(0, len(points)):
                        temp = points[i].split(',')
                        if(i < len(points)-1):
                            tau = tau + (temp[0]) + ','
                            zeta = zeta + (temp[1]) + ','
                        else:
                            tau = tau + (temp[0])
                            zeta = zeta + (temp[1])
                    X = pcs.parametricCubicSpline(tau, zeta, self.leftConstraint.current(),
                                                  self.rightConstraint.current())
                    previousX = 1-(1-0.05)*math.sin(math.pi*0) \
                                + 0.05*math.sin(3*math.pi*0)
                    previousY = X[0]
                    data = data + str(previousX) + "," + str(previousY) + ","
                    for i in range(1, 41):
                        u = i/40
                        myCanvasX = 1-(1-0.05)*math.sin(math.pi*u) \
                                    + 0.05*math.sin(3*math.pi*u)
                        myCanvasY = X[i]
                        data = data + str(myCanvasX) + "," + str(myCanvasY) + ","
                    data = data + "0" + ","
                    self.sendData(data)
                except:
                    tkinter.messagebox.showerror("Error", "Add points.")
        elif(value==4):
            try:
                X = pf.parametricFourier(float(self.f_entry_0_g.get()),
float(self.f_entry_1_g.get()),
                    float(self.f_entry_le_r.get()), float(self.f_entry_t_u.get()),
float(self.f_entry_t_z.get()),
                    float(self.f_entry_b_u.get()), float(self.f_entry_b_z.get()),
float(self.f_entry_b0.get()),
                    float(self.f_entry_b1.get()), float(self.f_entry_hte.get()))
                n = 8
                gam = pf.gam_(0, float(self.f_entry_0_g.get()), float(self.f_entry_1_g.get()))
                ww = 1 - 2 * 0
                previousX = pf.X_of_W(gam, ww)
                previousY = float(self.f_entry_hte.get()) * (1 - 2 * 0)
                data = data + str(previousX) + "," + str(previousY) + ","
                for i in range(1, 41):
                    u = i/40
                    gam = pf.gam_(u, float(self.f_entry_0_g.get()),
float(self.f_entry_1_g.get()))
                    if(u < 0.5):
                        ww = 1 - 2 * u
                    else:
                        ww = 2 * u - 1
                    xx = pf.X_of_W(gam, ww)
                    zz = float(self.f_entry_hte.get()) * (1 - 2 * u)
                    for j in range(0, n):
                        remain = 1
                        zz = zz + X[j] * math.sin(remain * math.pi * (j+1) * u)
                    data = data + str(xx) + "," + str(zz) + ","
                data = data + "0" + ","
                self.sendData(data)
            except ValueError:
                tkinter.messagebox.showerror("Error", "Entries must consist of numerical values
and must not be blank.")
```

```
        elif(value==5):
            try:
                for j in range(0, len(self.name)):
                    data = data + str(self.ug[j]) + ","
                    data = data + str(self.lg[j]) + ","
                    data = data + str(self.ler[j]) + ","
                    data = data + str(self.topPointU[j]) + ","
                    data = data + str(self.topPointZ[j]) + ","
                    data = data + str(self.bottomPointU[j]) + ","
                    data = data + str(self.bottomPointZ[j]) + ","
                    data = data + str(self.bu[j]) + ","
                    data = data + str(self.bl[j]) + ","
                    data = data + str(self.hte[j]) + ","
                    data = data + str(self.sparX[j]) + ","
                    data = data + str(self.sparY[j]) + ","
                    data = data + str(self.sparZ[j]) + ","
                    data = data + str(self.scale[j]) + ","
                data = data + "1" + ","
                self.sendData(data)
            except:
                tkinter.messagebox.showerror("Error", "Select a station to plot.")

    #   sendData
    #   - Creates a new socket to send data to Blender
    #   - The Application acts as the client, with Blender the Server, to send data over a port
    def sendData(self, data):
        host = '127.0.0.1'
        port = 7777
        server = ('127.0.0.1', 7777)
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect((host, port))
        s.sendto(str.encode(data[:-1]), server)
        s.close()

    #   resizeWindow
    #   - Checks the current value of the method combobox, appropriately resizing and cenetering
the window
    def resizeWindow(self):
        value = self.method.current()
        if(value == 0):
            w = 900
            h = 410
            screenW = self.parent.winfo_screenwidth()
            screenH = self.parent.winfo_screenheight()
            x = (screenW - w) / 2
            y = (screenH - h) / 2
            self.parent.geometry('%dx%d+%d+%d' % (w, h, x, y))
        elif(value == 1):
            w = 900
            h = 450
            screenW = self.parent.winfo_screenwidth()
            screenH = self.parent.winfo_screenheight()
            x = (screenW - w) / 2
            y = (screenH - h) / 2
            self.parent.geometry('%dx%d+%d+%d' % (w, h, x, y))
        elif(value == 2):
            w = 900
            h = 495
            screenW = self.parent.winfo_screenwidth()
            screenH = self.parent.winfo_screenheight()
            x = (screenW - w) / 2
            y = (screenH - h) / 2
            self.parent.geometry('%dx%d+%d+%d' % (w, h, x, y))
        elif(value == 3):
            w = 805
            h = 415
            screenW = self.parent.winfo_screenwidth()
            screenH = self.parent.winfo_screenheight()
            x = (screenW - w) / 2
            y = (screenH - h) / 2
            self.parent.geometry('%dx%d+%d+%d' % (w, h, x, y))
        elif(value == 4):
            w = 900
            h = 425
            screenW = self.parent.winfo_screenwidth()
            screenH = self.parent.winfo_screenheight()
            x = (screenW - w) / 2
            y = (screenH - h) / 2
            self.parent.geometry('%dx%d+%d+%d' % (w, h, x, y))
        elif(value == 5):
            w = 755
            h = 415
            screenW = self.parent.winfo_screenwidth()
            screenH = self.parent.winfo_screenheight()
            x = (screenW - w) / 2
            y = (screenH - h) / 2
            self.parent.geometry('%dx%d+%d+%d' % (w, h, x, y))

    #   centerWindow
    #   - Centers the original frame
    def centerWindow(self):
        w = 900
        h = 280
        screenW = self.parent.winfo_screenwidth()
        screenH = self.parent.winfo_screenheight()
        x = (screenW - w) / 2
        y = (screenH - h) / 2
        self.parent.geometry('%dx%d+%d+%d' % (w, h, x, y))

    def exitCheck(self):
        if tkinter.messagebox.askyesno("Confirm Exit", "Are you sure you want to exit the
program?"):
            self.parent.destroy()

#   Main Loop
#   - Calls tKinter and the Application Class
def main():
    root = Tk()
    app = AircraftGenerator(root)
    app.pack(side="top", fill="both", expand=True)
    root.mainloop()

if __name__ == '__main__':
    main()
```

5.3 Parametric Polynomial

```
#   CSULB - Northrop Grumman Student Design Project
#   Parametric Polynomial
#   Written by Marvin Trajano, Lisa Tran, Victor Tran
#   Spring 2015

import numpy as np
import math
import tkinter.messagebox

#   boundaryOne
#   - Takes in a leading edge points, upper trailing edge point, lower trailing edge point,
leading edge radius, beta
#     upper angle, beta lower angle, and gamma to calculate a polynomial for an airfoil
def boundaryOne(u_e, z_e, u_0, z_0, u_1, z_1, r, b0, b1, gamma):
    d2xdu2 = math.pi*math.pi*math.sin(math.pi*u_e)*(1-gamma)-
9*math.pi*math.pi*gamma*math.sin(3*math.pi*u_e)
    #Set up matrices
    A = np.matrix(np.zeros((5,5)))
    for i in range(0, 5):
        A[0, i] = math.pow(u_e, i+1)
    for i in range(0, 5):
        A[1, i] = math.pow(u_1, i+1)
    for i in range(0, 5):
        A[2, i] = math.pow(u_0, i)*(i+1)
    for i in range(0, 5):
        A[3, i] = math.pow(u_1, i)*(i+1)
    for i in range(0, 5):
        A[4, i] = math.pow(u_e, i)*(i+1)

    B = np.array(np.zeros(5))
    B[0] = z_e - z_0*(1-2*u_e)
    B[1] = z_1 - z_0*(1-2*u_1)
    B[2] = math.pi*math.tan(b0*math.pi/180)*(1-4*gamma) + 2*z_0
    B[3] = math.pi*math.tan(b1*math.pi/180)*(1-4*gamma) + 2*z_0
    B[4] = -1*math.pow((r*(d2xdu2)), 0.5) + 2*z_0
    #Solve matrices for polynomial
    try:
        X = np.linalg.solve(A, B)
        return X
    except:
        tkinter.messagebox.showerror("Error", "Matrix error.")

#   boundaryTwo
#   - Takes in a leading edge points, upper trailing edge point, lower trailing edge point, top
point, bottom point,
#     leading edge radius, beta upper angle, beta lower angle, and gamma to calculate a
polynomial for an airfoil
def boundaryTwo(u_e, z_e, u_0, z_0, u_1, z_1, r, b0, b1, gamma, u_t, z_t, u_b, z_b):
    d2xdu2 = math.pi*math.pi*math.sin(math.pi*u_e)*(1-gamma)-
9*math.pi*math.pi*gamma*math.sin(3*math.pi*u_e)
    #Set up matrices
    A = np.matrix(np.zeros((7,7)))
    for i in range(0, 7):
        A[0, i] = math.pow(u_e, i+1)
    for i in range(0, 7):
        A[1, i] = math.pow(u_1, i+1)
    for i in range(0, 7):
        A[2, i] = math.pow(u_0, i)*(i+1)
    for i in range(0, 7):
        A[3, i] = math.pow(u_1, i)*(i+1)
    for i in range(0, 7):
        A[4, i] = math.pow(u_e, i)*(i+1)
    for i in range(0, 7):
        A[5, i] = math.pow(u_t, i+1)
    for i in range(0, 7):
        A[6, i] = math.pow(u_b, i+1)
    B = np.array(np.zeros(7))
    B[0] = z_e - z_0*(1-2*u_e)
    B[1] = z_1 - z_0*(1-2*u_1)
    B[2] = math.pi*math.tan(b0*math.pi/180)*(1-4*gamma) + 2*z_0
    B[3] = math.pi*math.tan(b1*math.pi/180)*(1-4*gamma) + 2*z_0
    B[4] = -1*math.pow((r*(d2xdu2)), 0.5) + 2*z_0
    B[5] = z_t - (z_0*(1-2*u_t))
    B[6] = z_b - (z_0*(1-2*u_b))
    #Solve matrices for polynomial
    try:
        X = np.linalg.solve(A, B)
        return X
    except:
        tkinter.messagebox.showerror("Error", "Matrix error.")

#   boundaryThree
#   - Takes in a leading edge points, upper trailing edge point, lower trailing edge point, top
point, bottom point,
#     upper leading edge radius, lower leading edge radius, beta upper angle, beta lower angle,
upper gamma, and lower
#     gamma to calculate a polynomial for an airfoil
def boundaryThree(u_e, z_e, u_0, z_0, u_1, z_1, u_t, z_t, u_b, z_b, r_0, r_1, b0, b1, g0, g1):
    d2xdu2 = math.pi*math.pi*math.sin(math.pi*u_e)*(1-g0)-
9*math.pi*math.pi*g0*math.sin(3*math.pi*u_e)
    #Upper half matrices
    A = np.matrix(np.zeros((5,5)))
    for i in range(0, 5):
        A[0, i] = math.pow(0, i)*(i+1)
    for i in range(0, 5):
        A[1, i] = math.pow(u_t, i+1)
    for i in range(0, 5):
        A[2, i] = math.pow(u_t, i)*(i+1)
    for i in range(0, 5):
        A[3, i] = math.pow(1, i)
    for i in range(0, 5):
        A[4, i] = math.pow(1, i)*(i+1)
    B = np.array(np.zeros(5))
    B[0] = math.pow((r_0*(d2xdu2)), 0.5) / 2
    B[1] = z_t
    B[2] = 0
    B[3] = z_0
    B[4] = -1*math.tan(b0*math.pi/180)*(math.pi/2)*(1-4*g0)
    try:
        x_upper = np.linalg.solve(A, B)
    except:
        tkinter.messagebox.showerror("Error", "Matrix error.")
    #Lower Matrices
    d2xdu2 = math.pi*math.pi*math.sin(math.pi*u_e)*(1-g1)-
9*math.pi*math.pi*g1*math.sin(3*math.pi*u_e)
    for i in range(0, 5):
        A[0, i] = math.pow(0, i)*(i+1)
    for i in range(0, 5):
        A[1, i] = math.pow(u_b, i+1)
    for i in range(0, 5):
        A[2, i] = math.pow(u_b, i)*(i+1)
    for i in range(0, 5):
        A[3, i] = math.pow(1, i)
    for i in range(0, 5):
        A[4, i] = math.pow(1, i)*(i+1)
    B = np.array(np.zeros(5))
    B[0] = -1*math.pow((r_1*(d2xdu2)), 0.5) / 2
    B[1] = z_b
    B[2] = 0
    B[3] = z_1
    B[4] = math.tan(b1*math.pi/180)*(math.pi/2)*(1-4*g1)
    x_lower = np.linalg.solve(A, B)
    X = np.matrix(np.zeros((2,5)))
    for i in range(0,5):
        X[0, i] = x_upper[i]
    for i in range(0, 5):
        X[1, i] = x_lower[i]
    return X
```

5.4 Parametric Fourier

```
#   CSULB - Northrop Grumman Student Design Project
#   Parametric Fourier
#   Originally Written By Phil Barnes of NGC
#   Slight Modifications by Marvin Trajano, Lisa Tran, Victor Tran
#   Spring 2015

import math
import numpy as np
import tkinter.messagebox

#   parametricFourier
#   - Takes in a number of values (8 if points are considered as one) to fine tune the upper and
lower halves of an
#     airfoil, the result maintains continuity on all derivatives, an important characteristic in
airfoil design
def parametricFourier(g_t, g_b, r, x_t, z_t, x_b, z_b, b_t, b_b, z):
    n = 8
    FF = np.matrix(np.zeros((n, n)))
    BB = np.array(np.zeros(n))
    for i in range(1,4):
        if(i == 1):
            wt = W_of_X(g_t, x_t)
            wb = W_of_X(g_b, x_b)
        else:
            ut = 0.5 * (1 - wt)
            gam = gam_(ut, g_t, g_b)
            wt = W_of_X(gam, x_t)
            ub = 0.5 * (1 + wb)
            gam = gam_(ub, g_t, g_b)
            wb = W_of_X(gam, x_b)

    gama = 0.5 * (g_t + g_b)
    for i in range(0, n):
        if(i == 0):
            uu = 0
            ww = 1
            zz = 0 + z
            dZdU = math.pi * math.tan(b_t * (math.pi / 180) * (1 - 4 * g_t))
        elif(i == 1 or i == 2):
            uu = ut
            ww = wt
            zz = z_t
            dZdU = 0
        elif(i == 3 or i == 4):
            uu = 0.5
            ww = 0
            zz = 0
            dZdU = -1 * math.sqrt(r * (1+8*gama) * math.pi**2)
        elif(i == 5 or i == 6):
            uu = ub
            ww = wb
            zz = z_b
            dZdU = 0
        elif(i == 7):
            uu = 1
            ww = 1
            zz = 0 - z
            dZdU = math.pi * math.tan(b_b * (math.pi/180)) * (1 - 4 * g_b)
        gam = gam_(uu, g_t, g_b)
        xx = X_of_W(gam, ww)
        if(i != 1 and i != 4 and i != 5):
            BB[i] = dZdU + 2 * z
        else:
            BB[i] = zz - z * (1 - 2 * uu)
        for j in range(0, n):
            remain = 1
            if(i != 1 and i != 4 and i != 5):
                FF[i,j] = math.pi * (j+1) * math.cos(remain * math.pi * (j+1) * uu)
            else:
                FF[i,j] = math.sin(remain * math.pi * (j+1) * uu)
    try:
        PP = np.linalg.solve(FF,BB)
        return PP
    except:
        tkinter.messagebox.showerror("Error", "Matrix error.")

def W_of_X(gamma, xx):
    expo = 0.56
    xe = xx ** expo
    w_of_x = xe - 1.3 * gamma * math.sin(math.pi * xe ** 2)
    for i in range(1,5):
        x1 = X_of_W(gamma, w_of_x)
        xe1 = x1 ** expo
        w_of_x = w_of_x + xe - xe1
    return w_of_x

def X_of_W(gamma, ww):
    x_of_w = 1 - (1 - gamma) * math.cos(ww * math.pi / 2) - gamma * math.cos(ww * 3 * math.pi /
2)
    return x_of_w

def gam_(uu, g_t, g_b):
    gam = g_b + (g_t - g_b) * (math.cos(uu * (math.pi / 2))) ** 2
    return gam
```

5.5 Parametric Cubic Spline

```
#   CSULB - Northrop Grumman Student Design Project
#   Parametric Cubic Spline
#   Originally Written By Phil Barnes of NGC
#   Slight Modifications by Marvin Trajano, Lisa Tran, Victor Tran
#   Spring 2015

import numpy as np
import tkinter.messagebox

def parametricCubicSpline(tau_points, zeta_points, lc, rc):
    tau_points = tau_points.split(',')
    x_points_string = zeta_points.split(',')
    number_of_points = len(tau_points)
    time = np.array(np.zeros(number_of_points))
    xx = np.array(np.zeros(number_of_points))
    left_end_constraint = int(lc)
    right_end_constraint = int(rc)
    for i in range(0, len(tau_points)):
        time[i] = tau_points[i]
        xx[i] = x_points_string[i]
    #Call CS_solve(np, t_s(), x_s(), dxdt_s(), d2xdt2_s(), exL, exR) ' solve for 2nd derivatives
x(t)
    velocity = np.array(np.zeros(number_of_points))
    acceleration = np.array(np.zeros(number_of_points))
    number_of_splines = number_of_points - 1
    DEL = np.array(np.zeros(number_of_splines))
    eps = np.array(np.zeros(number_of_splines))
    for i in range(0, number_of_splines):
        DEL[i] = time[i+1] - time[i]
        eps[i] = xx[i+1] - xx[i]
    FF = np.matrix(np.zeros((number_of_points,number_of_points)))
    BB = np.array(np.zeros(number_of_points))
    if(number_of_points < 4 and left_end_constraint == 2):
        left_end_constraint = 1
    if(number_of_points < 4 and right_end_constraint == 2):
        right_end_constraint = 1
    vL = 0
    vR = 0
    #INFLUENCE COEFFICIENT MATRIX
    if(left_end_constraint != 0 and left_end_constraint != 1 and left_end_constraint != 2):
        vL = left_end_constraint
        left_end_constraint = 3
    if(right_end_constraint != 0 and right_end_constraint != 1 and right_end_constraint != 2):
        vR = right_end_constraint
        right_end_constraint = 3
    for i in range(0, number_of_points):
        for j in range(0, number_of_points):
            FF[i,j] = 0
            if(left_end_constraint == 1 and i == 0 and j == 0):
                FF[i,j] = 1
            elif((left_end_constraint == 3 or left_end_constraint == 0) and i == 0 and j == 0):
                FF[i,j] = DEL[0] / 3
            elif((left_end_constraint == 3 or left_end_constraint == 0) and i == 0 and j == 1):
                FF[i,j] = DEL[0] / 6
            elif(left_end_constraint == 2 and i == 0 and j == 0):
                FF[i,j] = 1
            elif(left_end_constraint == 2 and i == 0 and j == 1):
                FF[i,j] = -1 - DEL[0] / DEL[1]
            elif(left_end_constraint == 2 and i == 0 and j == 2):
                FF[i,j] = DEL[0] / DEL[1]
            elif(i > 0 and i < number_of_points - 1):
                if(j == i-1):
                    FF[i,j] = DEL[i-1] / 6
                elif(j == i):
                    FF[i,j] = (DEL[i-1] + DEL[i]) / 3
                elif(j == i+1):
                    FF[i,j] = DEL[i] / 6
            elif(right_end_constraint == 1 and i == number_of_points - 1 and j ==
number_of_points - 1):
                FF[i,j] = 1
            elif((right_end_constraint == 3 or right_end_constraint == 0) and i ==
number_of_points - 1
                 and j == number_of_points - 2):
                FF[i,j] = -DEL[number_of_splines - 1] / 6
            elif((right_end_constraint == 3 or right_end_constraint == 0) and i ==
number_of_points - 1
                 and j == number_of_points - 1):
                FF[i,j] = -DEL[number_of_splines - 1] / 3
            elif(right_end_constraint == 2 and i == number_of_points - 1 and j ==
number_of_points - 1):
                FF[i,j] = 1
            elif(right_end_constraint == 2 and i == number_of_points - 1 and j ==
number_of_points - 2):
                FF[i,j] = -1 - DEL[number_of_points - 2] / DEL[number_of_points - 3]
            elif(right_end_constraint == 2 and i == number_of_points - 1 and j ==
number_of_points - 3):
                FF[i,j] = DEL[number_of_points - 2] / DEL[number_of_points - 3]
        BB[i] = 0
        if(left_end_constraint == 1 and i == 0):
            BB[i] = 0
        elif((left_end_constraint == 3 or left_end_constraint == 0) and i == 0):
            BB[i] = eps[0] / DEL[0] - vL
        elif(left_end_constraint == 2 and i == 0):
            BB[i] = 0
        elif(i > 0 and i < number_of_points - 1):
            BB[i] = eps[i] / DEL[i] - eps[i - 1] / DEL[i - 1]
        elif(right_end_constraint == 1 and i == number_of_points - 1):
            BB[i] = 0
        elif((right_end_constraint == 3 or right_end_constraint == 0) and i == number_of_points -
1):
            BB[i] = eps[number_of_splines - 1] / DEL[number_of_splines - 1] - vR
        elif(right_end_constraint == 2 and i == number_of_splines - 1):
            BB[i] = 0
    try:
        acceleration = np.linalg.solve(FF,BB)
    except:
        tkinter.messagebox.showerror("Error", "Matrix error.")
    #1st derivatives
    for i in range(0, number_of_points):
        if(i < number_of_points - 1):
            velocity[i] = eps[i] / DEL[i] - acceleration[i] * DEL[i] / 3 - acceleration[i+1] *
DEL[i] / 6
        else:
            velocity[i] = velocity[number_of_splines-1] + acceleration[number_of_splines-1] \
                * DEL[number_of_splines-1] / 2 + acceleration[number_of_points-1] *
DEL[number_of_splines-1] / 2
    #interp
    vert = 41
    xArray = np.array(np.zeros(vert))
    for i in range(0, vert):
        to_ = i / (vert - 1)
        if (to_ < time[0]):
            xo_ = xx[0] + velocity[0] * (to_ - time[0])
            vvo = velocity[0] + acceleration[0] * (to_ - time[0])
            d3xdt3 = (acceleration[1] - acceleration[0]) / DEL[0]
            aao = acceleration[0] + d3xdt3 * (to_ - time[0])
        elif(to_ > time[number_of_points-1]):
            xo_ = xx[number_of_points-1] + velocity[number_of_points-1] * (to_ -
time[number_of_points - 1])
            vvo = velocity[number_of_points - 1] + acceleration[number_of_points-1] * (to_ -
time[number_of_points - 1])
            d3xdt3 = (acceleration[number_of_points - 1] - acceleration[number_of_points - 2])\
                / DEL[number_of_splines-1]
            aao = acceleration[0] + d3xdt3 * (to_ - time[0])
        else:
            for j in range(0, number_of_splines):
                if(time[j+1] >= to_):
                    tmti = to_ - time[j]
                    epsi = eps[j]
                    DELI = DEL[j]
                    xxi = xx[j]
                    aai = acceleration[j]
                    dxdti = velocity[j]
                    aip1 = acceleration[j+1]
                    xo_ = xxi + dxdti * tmti + aai * tmti ** 2 / 2 + (aip1 - aai) * tmti ** 3 /
(6 * DELI)
                    vvo = dxdti + aai * tmti + (aip1 - aai) * tmti ** 2 / (2 * DELI)
                    aao = aai + (aip1 - aai) * tmti / DELI
                    break
        xArray[i] = xo_
    return xArray
```

Northrop Grumman Student Design Project Spring 2015

User Manual

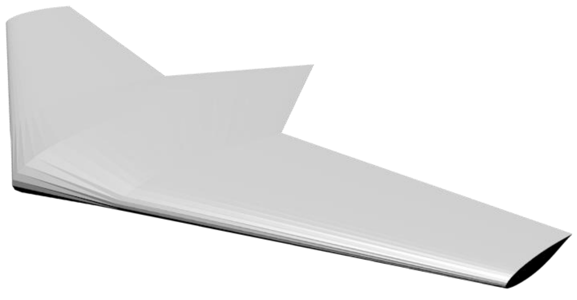


Table of Contents

Running the Application

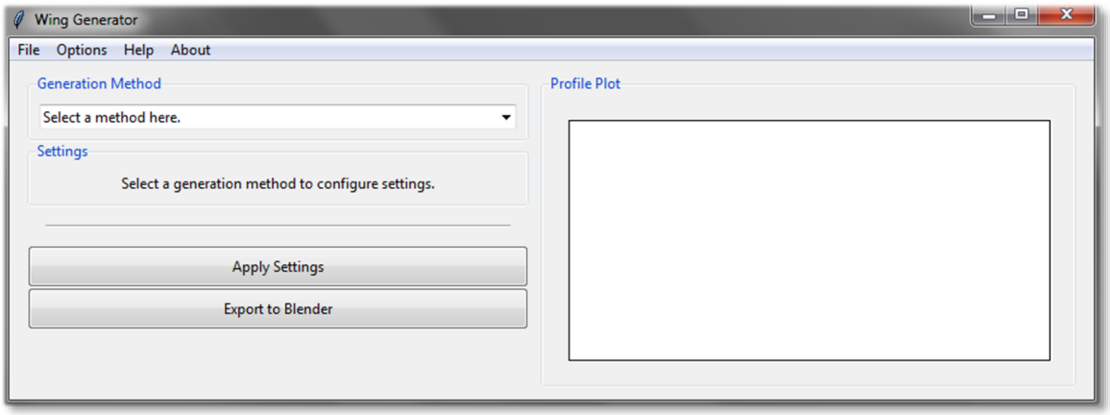
Download the application from our website at www.csulb-ngcproject.me.

Option 1:
You can download the following source code directly and run the application straight through Python:

> AircraftGenerator.py
> ParametricPolynomial.py
> ParametricCubicSpline.py
> ParametricFourier.py

Option 2:
You can download the zipped folder which contains an executable that can be ran without Python installed (Windows only).

Once you run the application, the Wing Generator GUI should appear:

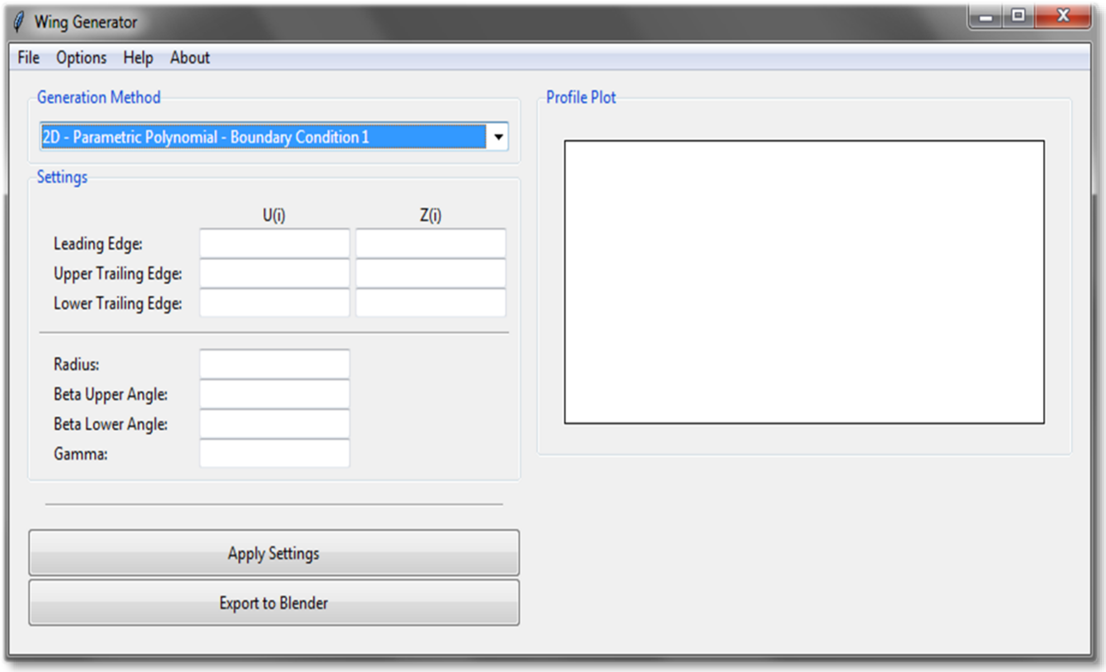


Initial GUI

Using the Different Generation Methods

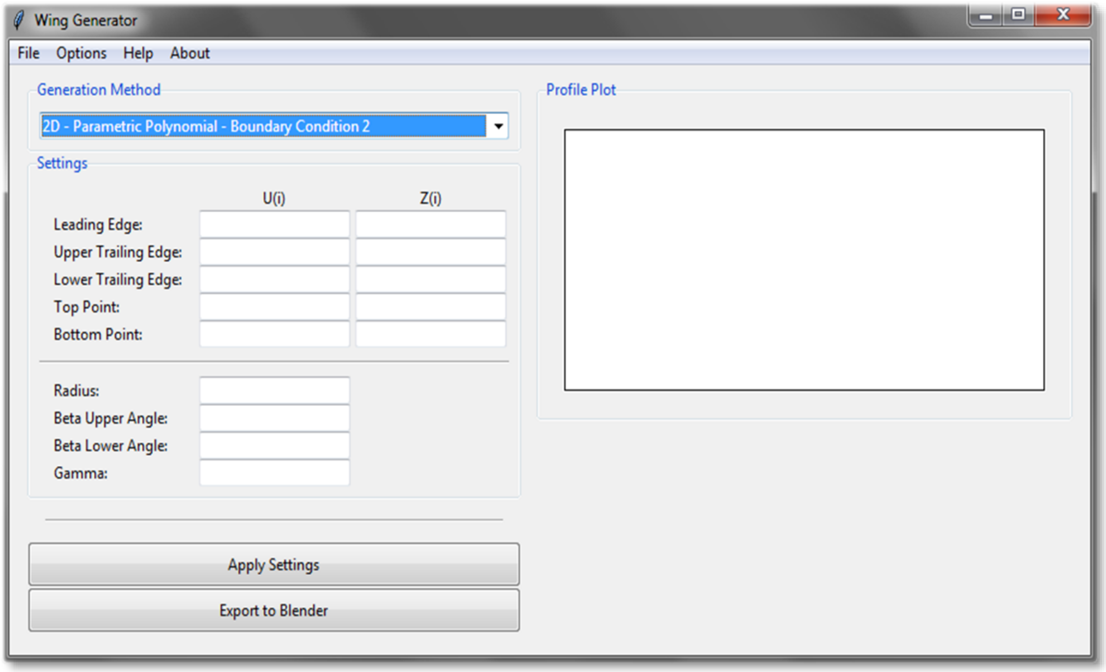
Parametric Polynomial

The parametric polynomial allows you to specify values at key points in the airfoil such as leading edge radius, after body angles, and top and bottom point maximums. There are 3 different boundary conditions. Selecting different boundary conditions will allow you more options for key points.



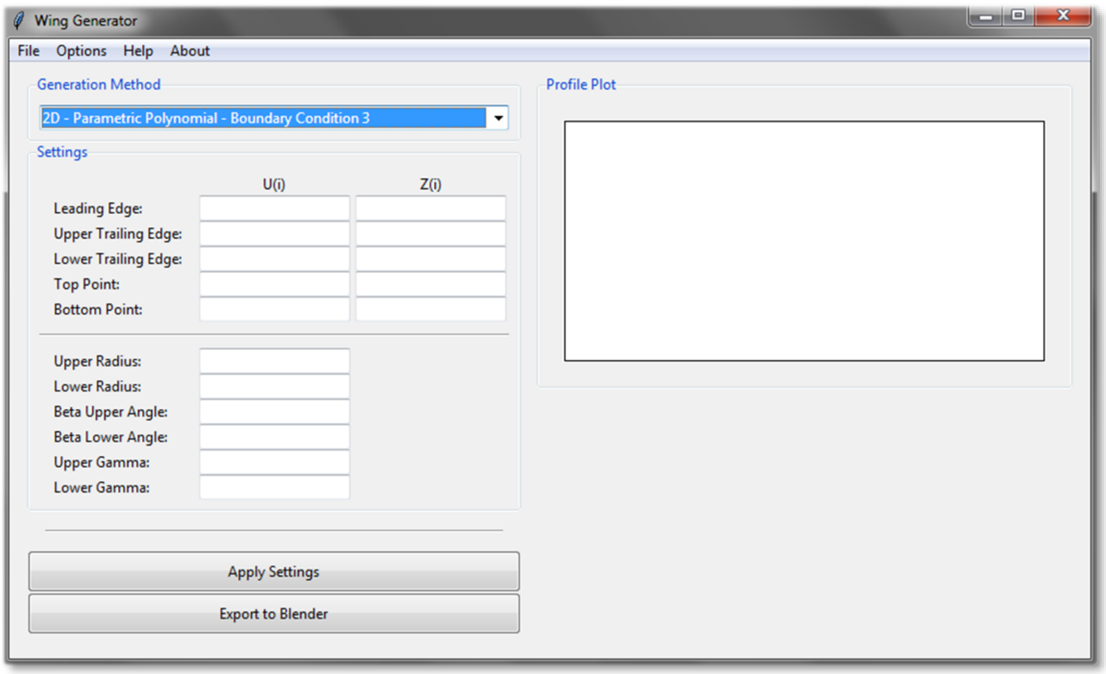
Generation Method: 2D - Parametric Polynomial - Boundary Condition 1

Parametric Polynomial – Boundary Condition 1



Generation Method: 2D - Parametric Polynomial - Boundary Condition 2

Parametric Polynomial – Boundary Condition 2



2D - Parametric Polynomial - Boundary Condition 3

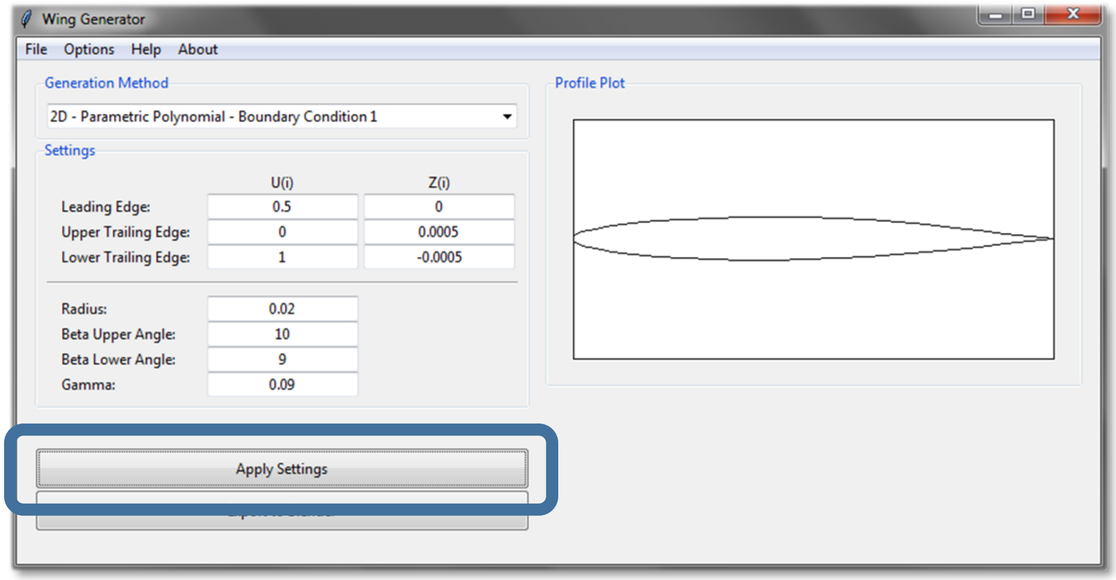
Settings

	U(i)	Z(i)
Leading Edge:		
Upper Trailing Edge:		
Lower Trailing Edge:		
Top Point:		
Bottom Point:		

Upper Radius:	
Lower Radius:	
Beta Upper Angle:	
Beta Lower Angle:	
Upper Gamma:	
Lower Gamma:	

Parametric Polynomial – Boundary Condition 3

Enter your values and click "Apply Settings" to generate the plot.



2D - Parametric Polynomial - Boundary Condition 1

Settings

	U(i)	Z(i)
Leading Edge:	0.5	0
Upper Trailing Edge:	0	0.0005
Lower Trailing Edge:	1	-0.0005

Radius:	0.02
Beta Upper Angle:	10
Beta Lower Angle:	9
Gamma:	0.09

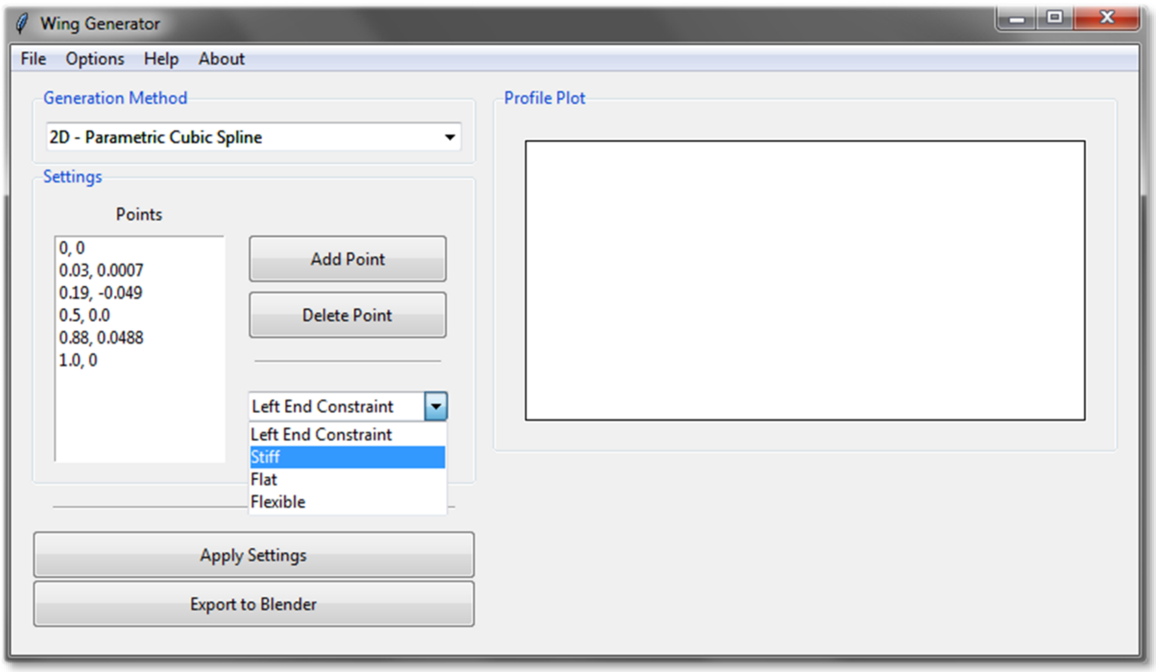
Input Preview: Parametric Polynomial – Boundary Condition 1 and Resulting Plot

Parametric Cubic Spline

The parametric polynomial allows you to specify a number of points around the airfoil. There are already default points to use an as example.

> Note: For the parametric cubic spline, the order of the points matters. You can adjust the points by using your mouse to drag them up or down appropriately.

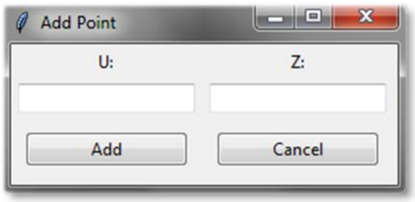
First you need to choose the shape of the left and right end constraints of the airfoil by selecting from the drop-down menu:



```
0, 0
0.03, 0.0007
0.19, -0.049
0.5, 0.0
0.88, 0.0488
1.0, 0
```

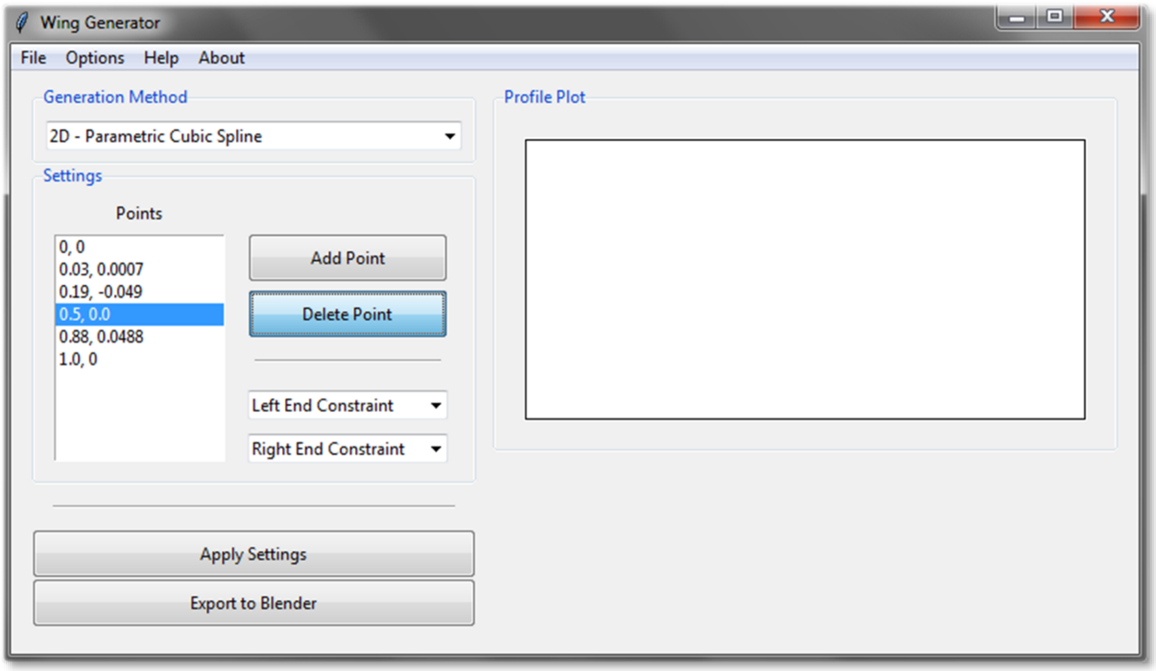
Selecting the End Constraints

You can add more points by clicking the "add point" button:



Adding a New Point

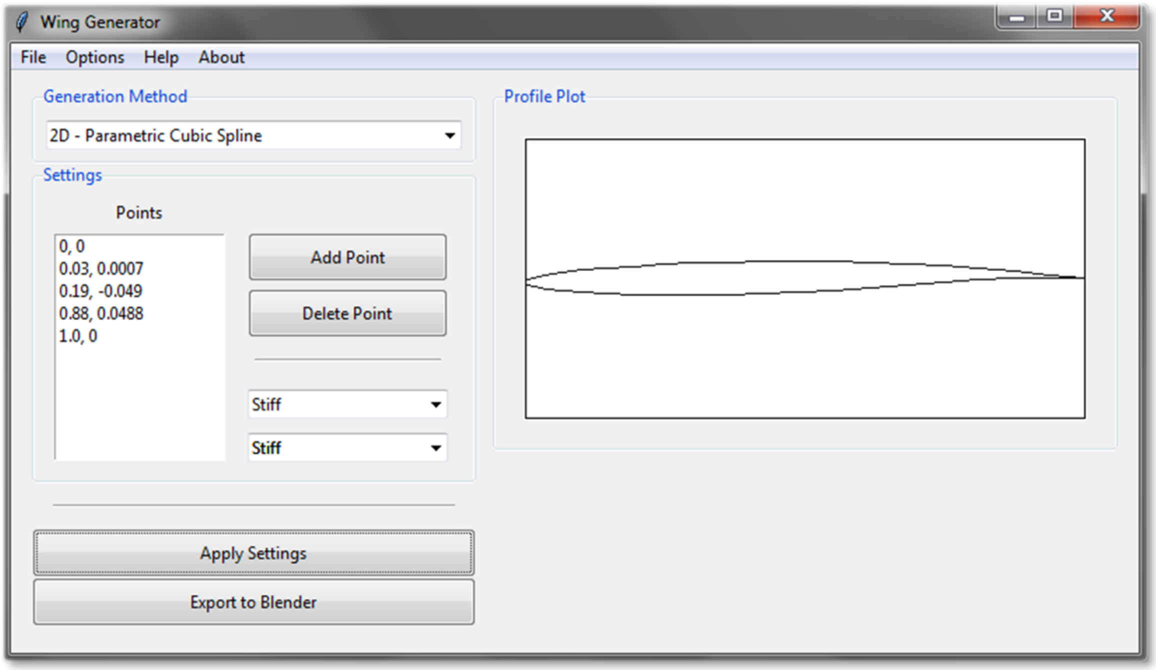
You can delete points by highlighting the set of points you want to delete and clicking the "delete point" button:



0, 0
0.03, 0.0007
0.19, -0.049
0.5, 0.0
0.88, 0.0488
1.0, 0

Deleting a Point

Once you have the points you want to use, click "apply settings" to generate the plot.



0, 0
0.03, 0.0007
0.19, -0.049
0.88, 0.0488
1.0, 0

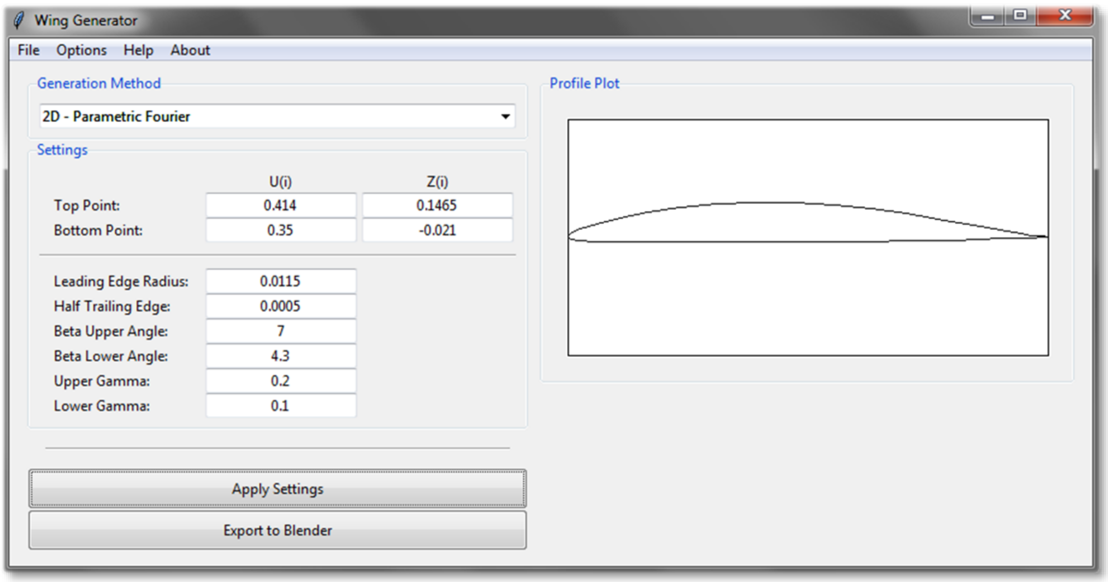
Stiff

Stiff

Generated Plot

Parametric Fourier 2D

The 2D parametric Fourier allows you to define 8 values: top point, bottom point, leading edge radius, half trailing edge, beta upper angle, beta lower angle, upper gamma, and lower gamma to fine tune the upper and lower halves of the airfoil. Enter your values and click "apply settings" to generate the plot.



Generation Method: 2D - Parametric Fourier

	U(i)	Z(i)
Top Point:	0.414	0.1465
Bottom Point:	0.35	-0.021

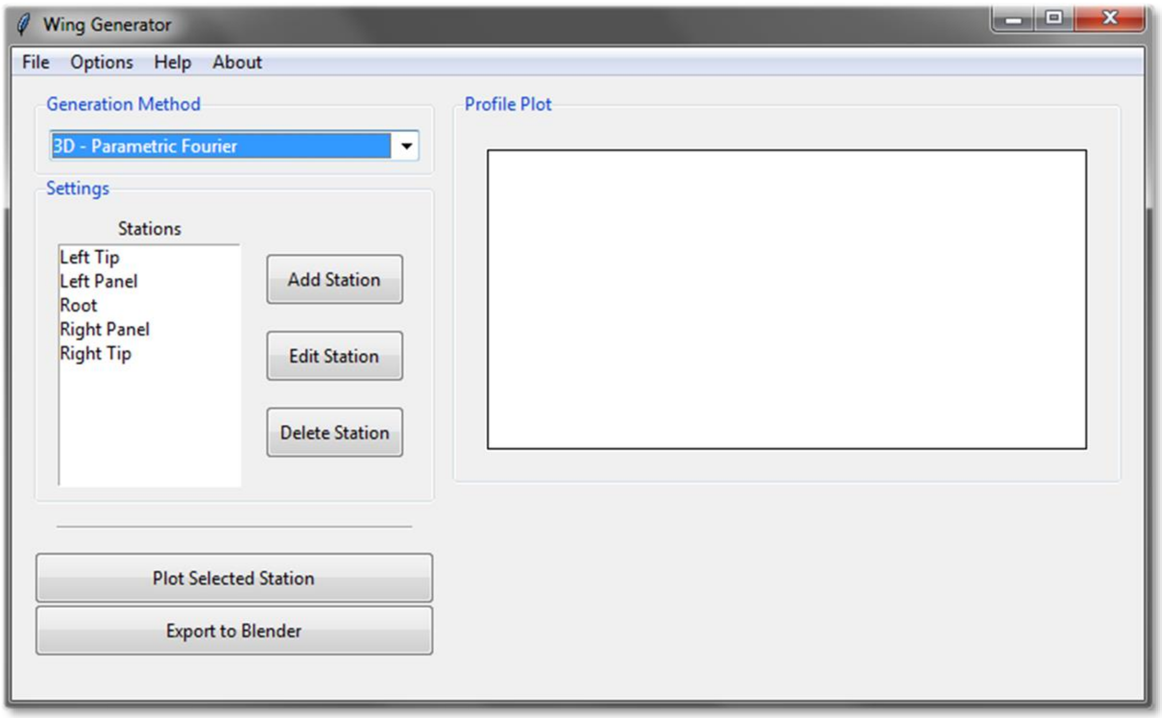
Leading Edge Radius:	0.0115
Half Trailing Edge:	0.0005
Beta Upper Angle:	7
Beta Lower Angle:	4.3
Upper Gamma:	0.2
Lower Gamma:	0.1

Parametric Fourier Settings and Generated Plot

Parametric Fourier 3D

The 3D parametric Fourier allows you to define the values for multiple airfoils (stations), across the span of a wing. An object is then generated with these airfoils connected. Note that the airfoil is connected to the next airfoil as it goes down the list, so the order in which you add the stations to the list will determine the order in which they are generated and connected.

Note: There is a set of preset stations as an example.

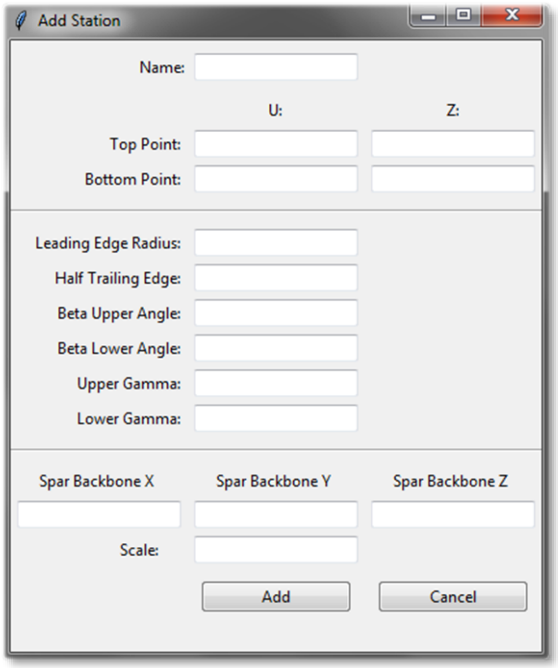


Generation Method: 3D - Parametric Fourier

Stations
Left Tip
Left Panel
Root
Right Panel
Right Tip

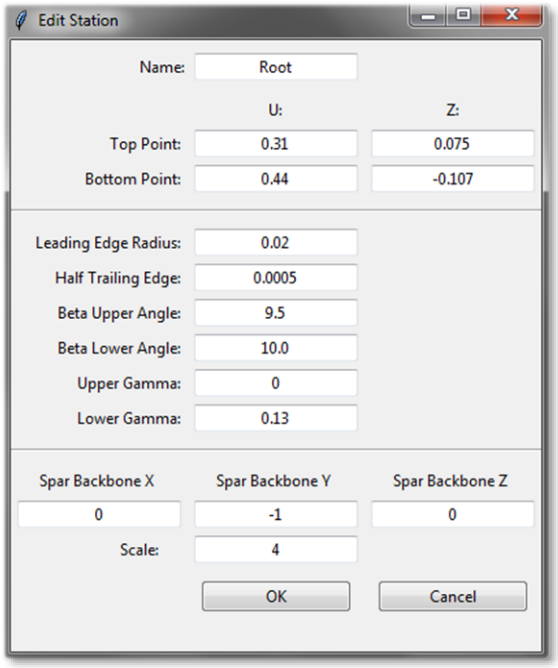
Default Interface for 3D Parametric Fourier

You can click “Add Station”, which will prompt a dialog box to appear with entries for inputs.



Dialog Box for Adding a Station

Clicking “Edit Station” after selecting one of the stations from the list will allow you to change the values of a station.



Name: Root

	U:	Z:
Top Point:	0.31	0.075
Bottom Point:	0.44	-0.107

Leading Edge Radius:	0.02
Half Trailing Edge:	0.0005
Beta Upper Angle:	9.5
Beta Lower Angle:	10.0
Upper Gamma:	0
Lower Gamma:	0.13

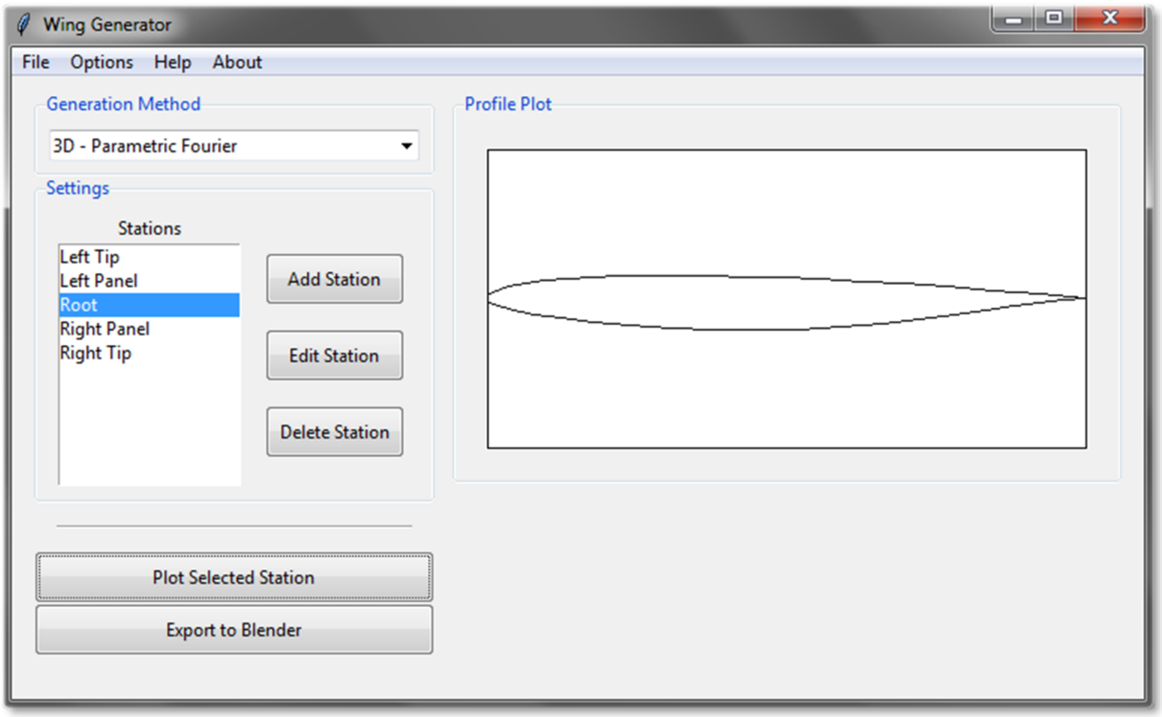
Spar Backbone X	Spar Backbone Y	Spar Backbone Z
0	-1	0

Scale: 4

Dialog Box for Editing a Station

Clicking “Delete Station” after selecting one of the stations from the list will delete a station.

You can click “Plot Selected Airfoil” to plot the selected airfoil.



Plot after pushing “Plot Selected Station”

Opening and Saving .txt Files

Opening and saving .txt files is only supported for the parametric polynomial and the 2D parametric Fourier.

Download these .txt files from our website for examples:

> BoundaryCondition1.txt
> BoundaryCondition2.txt
> BoundaryCondition3.txt
> 2DFourier.txt

Opening

We will use the parametric polynomial, boundary condition 1 generation method for our example.

1. Select the generation method you want to use, in our case "2D - Parametric Polynomial - Boundary Condition 1".
2. Go to "File".
3. Select "Open".
4. Select "BoundaryCondition1.txt" from wherever you have it saved and click "Open." The values from the file should be imported into the application.
5. Click "Apply settings" to generate the plot.

Saving

To save any data you have entered:

1. Go to "File".
2. Select "Save As".
3. Enter a name and click "Save".

Once you have created your .txt file, you can continue working. If you want to save again, you can go to "File" and select "Save" instead.

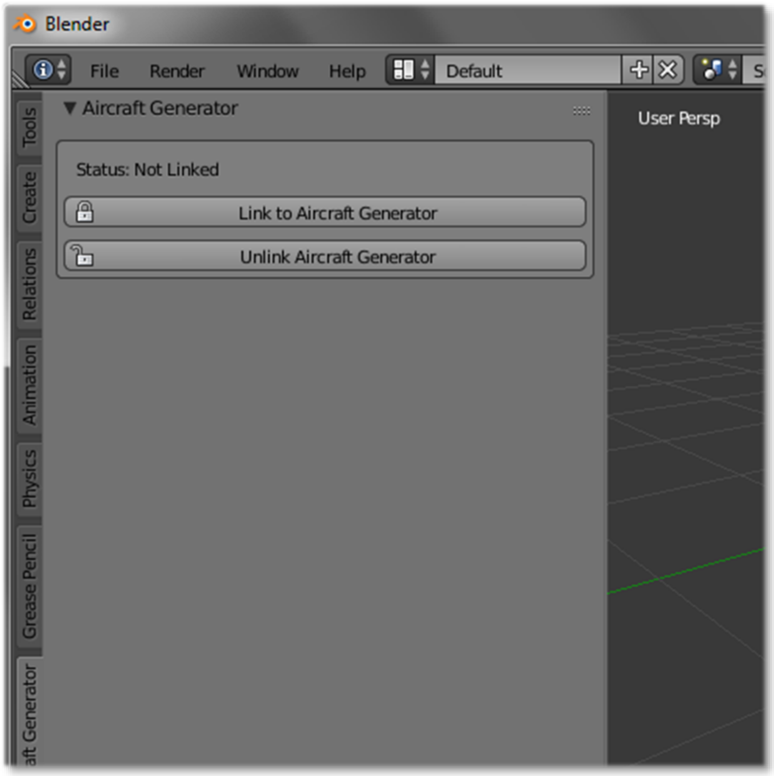
Connecting and Exporting to Blender

Once you have your airfoil generated with your preferred generation method and you want to export to Blender, follow these steps before hitting "Export to Blender":

Installing

1. Download the "AircraftServer.py" file from our website.
2. Open up Blender.
3. Go to File > User Preferences > Addons or Ctrl-Alt-U > Addons
4. Click “Install from File...”
5. Navigate to the “AircraftServer.py” file and click on it
6. Search for “Aircraft Server” and tick the checkbox to activate the Addon
7. Click “Save User Settings” and Restart Blender

You should see the Addon on the left sidebar if you installed and activated it correctly.



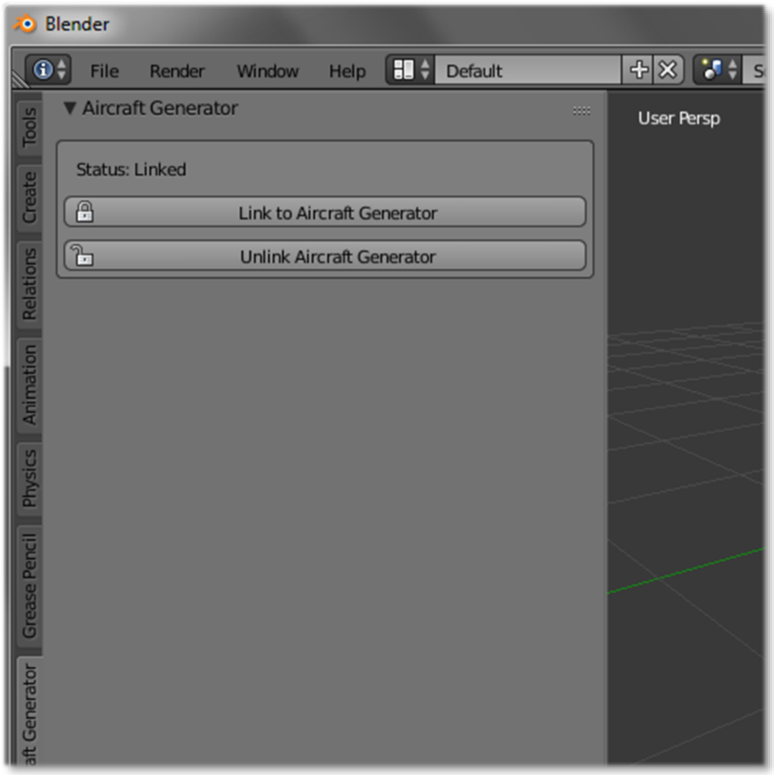
Status: Not Linked

Link to Aircraft Generator

Unlink Aircraft Generator

Addon After Installing and Activating

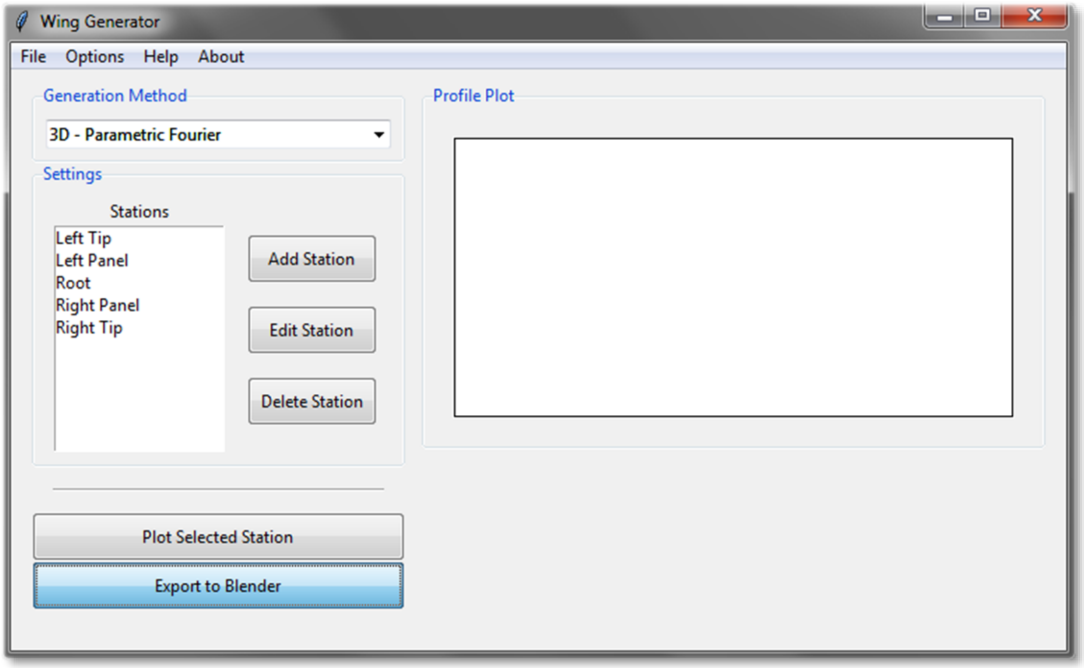
Click "Link to Aircraft Generator". It should change to "Status: Linked".


Blender
File Render Window Help Default
Tools Create Relations Animation Physics Grease Pencil aft Generator
Aircraft Generator
User Persp
Link to Aircraft Generator
Unlink Aircraft Generator

Status: Linked

Status after Linking

Go back to the Wing Generator application and click "Export to Blender".

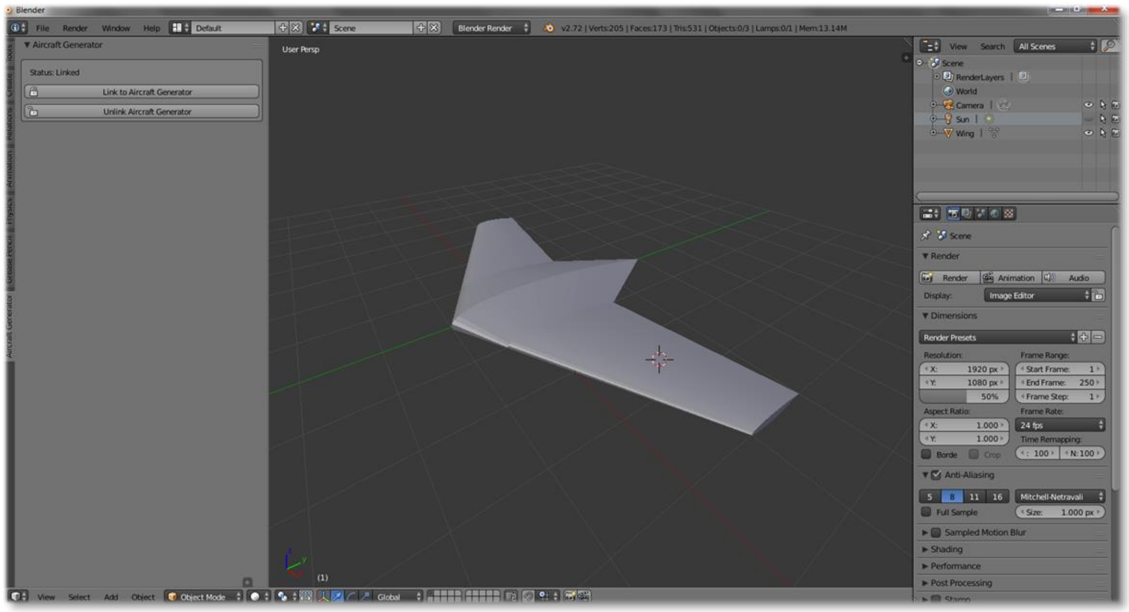


Generation Method: 3D - Parametric Fourier

Stations
Left Tip
Left Panel
Root
Right Panel
Right Tip

Export to Blender Button being Pressed

Go back to Blender. You should now see the generated model.



```
v2.72 | Verts:205 | Faces:173 | Tris:531 | Objects:0/3 | Lamps:0/1 | Mem:13.14M
```

Resolution: X: 1920 px, Y: 1080 px, 50%
Frame Range: Start Frame: 1, End Frame: 250, Frame Step: 1
Aspect Ratio: X: 1.000, Y: 1.000
Frame Rate: 24 fps
Time Remapping: 100, N: 100
Anti-Aliasing: 5 8 11 16, Size: 1.000 px

Exported Model

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
Victor Tran	victor.tran@student.csulb.edu