Calculating Fin Flutter Velocity for Complex Fin Shapes

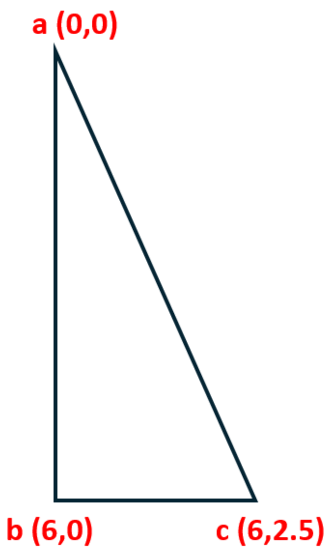
© John K Bennett, 2024
jkb@colorado.edu
January 2024

In *Peak of Flight* Issue #615[1], I described how to estimate the fin flutter velocity for trapezoidal and elliptical fin shapes. This follow-on article describes how to handle triangular, multi-sided, and other more complex fin shapes. The original article should be read for this one to make better sense. In most of the examples shown, we will use arbitrary units of length, without concerning ourselves about the unit system in use.

When estimating flutter velocity for a complex fin design, the fin properties that are most likely to be challenging to calculate include **Fin Area**, **Cx** (the axial distance from the front of the fin to the fin centroid), and possibly **tip chord length**. Let's examine how these values can be calculated for fin shapes that are not trapezoidal or elliptical.

Triangular Fins

Although there are explicit formulae for the area and centroid of a triangle (see below), we don't have to use them. Since a triangle can be considered as a degenerate form of trapezoid, the trapezoidal equations in the original article and spreadsheet work fine for triangular fins; we just need to set the tip chord length to zero. This will also make **λ** equal to zero. The trapezoidal equations will produce correct results for triangular fins once this is done. Let's prove this to ourselves. Consider the following triangular fin:



The area of a triangle = **½ base * height**, or in this case, **3*2.5 = 7.5**. The area of a trapezoid with a zero-length tip chord reduces to the same formula:

$$Area = Height \times \frac{\cancel{Tip\,Chord\,Length} + Root\,Chord\,Length}{2}$$

As we will see below, the centroid of a triangle given its vertices is just the average of each of the vertex coordinates, i.e.,

$$C_x = \frac{(V1_x + V2_x + V3_x)}{3}$$

Cx = **(0 + 6 + 6) / 3 = 4.**

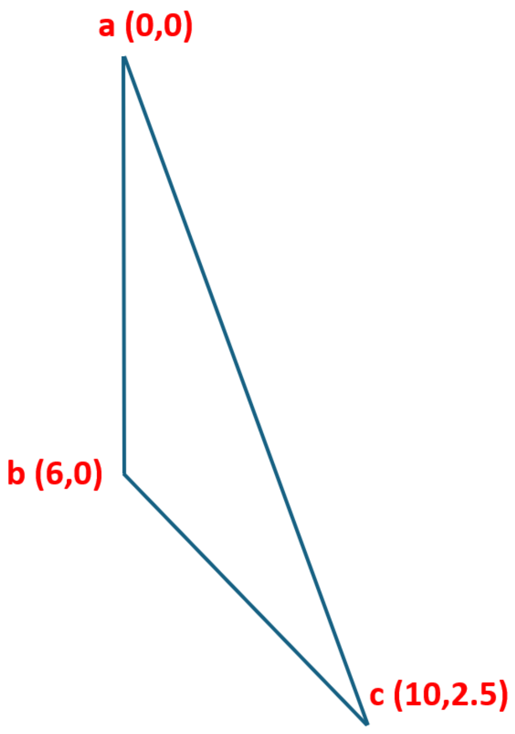
[1] https://www.apogeerockets.com/education/downloads/Newsletter615.pdf

If we use our formula for the centroid of a trapezoid with the tip chord set to zero (note that m, the sweep length, is the **x** coordinate of the **c** vertex in the triangle shown above),

$$C_x = \frac{(2 \times TC \times m) + TC^2 + (m \times RC) + (TC \times RC) + RC^2}{3(TC + RC)}$$

we get **Cx** = **((0) + (0) + (6*6) + (0) + (6*6)) / (3*6) = 4**, the same answer we obtained above.

This method also works when the triangle is irregular. Consider the following (probably not very good) fin design:



In this case, **Cx** calculated using the triangle formula **is (0 + 6 + 10) / 3 = 5.33. Cx** calculated using the trapezoid formula (with TC set to zero) is = **((0) + (0) + (10*6) + (0) + (6*6)) / (3*6) = 5.33.**

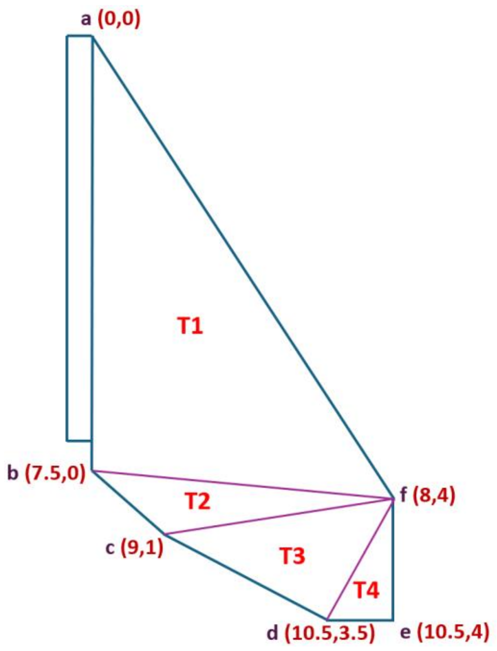
To summarize, for triangular fins we can calculate **V_f** using the fin flutter velocity spreadsheet[2] in the usual way. We only need to set the tip chord length to zero. The fin sweep length is just the **x** coordinate of vertex **c** (6 and 10 in our two examples).

Triangular fins such as these are also easy to simulate. RockSim and OpenRocket both provide the means to create custom fins by entering a set of points. In this case, we would enter the vertices **a**, **b**, and **c**. See *Peak of Flight* Issue #488 (February 5, 2019)[3] for an example of how to do this in RockSim.

Other Simple Polygon-Based Fin Shapes

For more complex fin shapes, we can use a method called "geometric decomposition." This method divides the fin into a set of geometric objects (typically triangles, rectangles, or circles) that we know how to handle, and then combines the results. The easiest way to employ geometric decomposition is to assign **(x, y)** coordinates to each vertex of the fin polygon. For example, consider the swept fin pictured below.

[2] https://github.com/jkb-git/Fin-Flutter-Velocity-Calculator
[3] https://www.apogeerockets.com/education/downloads/Newsletter488_Large.pdf



The fin polygon vertices are labeled **a**, **b**, **c**, **d**, **e**, and **f**. We first determine the coordinates for each vertex (assigning **(0, 0)** to the forward most vertex toward the nosecone to simplify the calculation). The vertex coordinates come from the fin design itself. Then we divide the fin polygon into a small number of triangles.[4] The resulting four triangles: **abf**, **bcf**, **cdf**, and **def,** are demarked in the figure with purple lines. Let's label these triangles **T1**, **T2**, **T3**, and **T4**, respectively, as shown on the figure. If we calculate the areas of the four triangles, the area of the entire fin is just the sum of these areas. Let's see how this works. From geometry, we know that the area of any triangle, given its three vertex coordinates **(Ax, Ay)**, **(Bx, By)**, and **(Cx, Cy)** is:

$$Area = \left|\frac{A_x(B_y - C_y) + B_x(C_y - A_y) + C_x(A_y - B_y)}{2}\right|$$

Using this formula, we can calculate the areas of each of the four triangles are as follows:

$$Area(T_1) = \left|\frac{a_x(b_y - f_y) + b_x(f_y - a_y) + f_x(a_y - b_y)}{2}\right|$$

Area of T1 (abf) = **(0(0-4) + 7.5(4-0) + 8(0-0))/2** = **15**

$$Area(T_2) = \left|\frac{b_x(c_y - f_y) + c_x(f_y - b_y) + f_x(b_y - c_y)}{2}\right|$$

Area of T2 (bcf) = **(7.5(1-4) + 9(4-0) + 8(0-1))/2 = (-22.5+36-8)/2** = **2.75**

$$Area(T_3) = \left|\frac{c_x(d_y - f_y) + d_x(f_y - c_y) + f_x(c_y - d_y)}{2}\right|$$

Area of T3 (cdf) = **(9(3.5-4) + 10.5(4-1) + 8(1-3.5))/2 = (-4.5+31.5-20)/2** = **3.5**

$$Area(T_4) = \left|\frac{d_x(e_y - f_y) + e_x(f_y - d_y) + f_x(d_y - e_y)}{2}\right|$$

[4] Polygon triangulation for an arbitrary polygon is an interesting problem in computational geometry. For our purposes, all we need to know is that the minimum number of triangles for an n-sided polygon is (n-2). And it really does not matter if we have an extra triangle or two when we partition our fin. We will revisit this issue later in the article.

Area of T4 (def) = **(10.5(4-4) + 10.5(4-3.5) + 8(3.5-4))/2 = (0+5.25-4)/2** = **0.625**

The total fin area is therefore **15 + 2.75 + 3.5 + 0.625 = 21.875**. Now that we have the area, we can calculate **Cx**. To do this, we need to find the centroid of each of our triangles (recall that we only need the axial - parallel to the direction of flight - dimension of the centroid), and then calculate their weighted average to determine the fin centroid. Let's start with the triangle centroids. **Cx** of any triangle, given its coordinates, is just the average of the **x** coordinates of its vertices, i.e.,

$$C_x = \frac{(V1_x + V2_x + V3_x)}{3}$$

Therefore:

T1_Cx = (0 + 7.5 + 8)/3 = 5.17

T2_Cx = (7.5 + 9 + 8)/3 = 8.17

T3_Cx = (9 + 10.5 + 8)/3 = 9.17

T4_Cx = (10.5 + 10.5 + 8)/3 = 9.67

To calculate **Cx** of the entire fin, we calculate a weighted average by adding the products of each triangle's **Cx** and the area of the that triangle, and then divide the result by the total fin area, as follows:

$$C_x = \frac{((T1\_Cx \times T1\_Area) + (T2\_Cx \times T2\_Area) + (T3\_Cx \times T3\_Area) + (T4\_Cx \times T4\_Area))}{TotalFinArea}$$

Therefore, **Cx** = **((5.17 * 15) + (8.17 * 2.75) + (9.17 * 3.5) + (9.67 * 0.625)) / 21.875**

= **((77.55) + (22.47) + (32.1) + (6.04)) / 21.875**

= **6.31**

Now we can calculate ***ε*** using the formula (recalling that ***ε*** is a measure of distance expressed as a fraction of the whole root chord):

$$\varepsilon = (\frac{C_x}{RC}) - 0.25$$

ε = 6.31/7.5 – 0.25 = **0.59**

Armed with these values, we can now calculate $\mathbf{V_f}$ in the usual way. Recall that the fin sweep length is just the **x** coordinate of vertex **f** (8 in this case).

Entering these data into the fin flutter velocity spreadsheet[2] (overriding the values for **Cx** and **Fin Area** to use the values calculated above) allows us to estimate the flutter velocity for our polygonal fin.

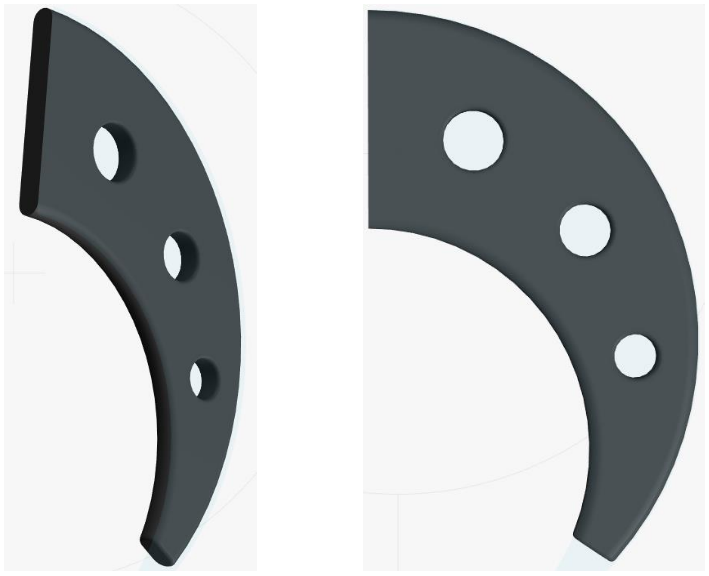
Polygonal fins such as these are also easy to simulate. As before, RockSim and OpenRocket both provide the means to create custom fins by entering a set of points. In this case, we would enter the vertices **a**, **b**, **c**, **d**, **e**, and **f**. Again, see *Peak of Flight* Issue #488 (February 5, 2019)[3] for an example of how to do this in RockSim.

<u>Complex Fins with Conic Shapes</u>

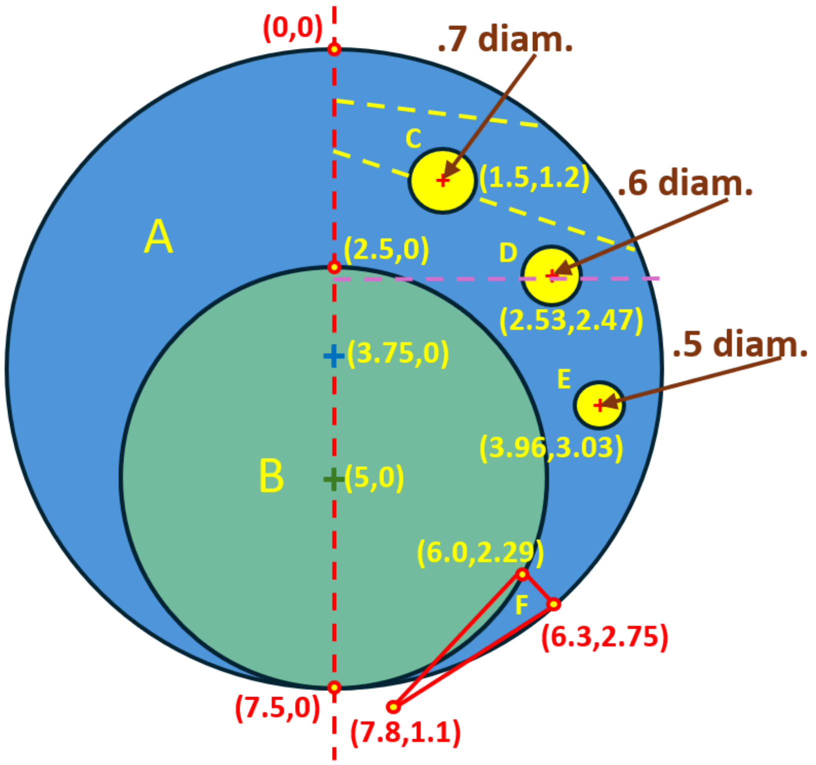
Geometric decomposition is a powerful tool, which can be applied in either an additive (as in the previous example) or a subtractive manner, as we will see in the next example. If you have ever designed a part to be 3D printed by joining and cutting different geometric shapes, that is the same process we will use to obtain the area and centroid of a complex fin.

This example will explore how to estimate the flutter velocity of a "bat wing" fin with holes. For purposes of this example, we will ignore the question of the flight-appropriateness of such a fin, and just go with "hey, they look really cool." We will also ignore any aerodynamic concerns other than estimation of flutter velocity.

Consider the following fin shape:



Let's see how we can analyze this fin using geometric decomposition. We start by looking closely at how the fin image might be constructed (this approach was, in fact, the one used in Fusion 360 to design the fin depicted above), as shown in the (not quite to scale) figure below.



This figure shows two large circles (**A** enclosing **B**), three small circles (**C**, **D**, and **E**), and triangle **F**. The red dashed line bisects the two large circles. The two dashed yellow lines represent the quarter chord and half chord lines. We will come back to the dashed purple line in a bit. The coordinates show the location of all relevant points (color has no significance other than visibility). Triangle **F** is intended to represent the "straightened out" small, curved triangle at the bottom of the figure with an equivalent area triangle. This is just to save us a bunch of tedious math for a very small part of the problem. After we size triangle **F**, we can "cut off" the blue arced segment from our fin (and from further consideration).

Calculation of Fin Area

Upon examination, we see that our bat fin image can be constructed by taking the right half of circle **A**, and subtracting the right half of circle **B**, as well as circles **C**, **D**, and **E**, and the triangle **F**. This is exactly how we can calculate the area of our fin, as follows:

The area of the right half of circle **A** is **(πr^2)/2 = (π*3.75*3.75) / 2** = **22.1**

The area of the right half of circle **B** is **(πr^2)/2 = (π*2.5*2.5) / 2** = **9.82**

The area of circle **C** is **(πr^2) = π*0.35*0.35** = **.38**

The area of circle **D** is **(πr^2) = π*0.3*0.3** = **.28**

The area of circle **E** is **(πr^2) = π*0.25*0.25** = **.2**

We use the formula for the area of a triangle given its vertices to calculate the area of triangle **F**:

$$Area = \left| \frac{A_x(B_y - C_y) + B_x(C_y - A_y) + C_x(A_y - B_y)}{2} \right|$$

The area of triangle **F** is **(7.8(2.75-2.29) + 6.3(2.29-1.1) + 6(1.1-2.75))/2 = (3.59+7.5-9.9)/2** = **0.6**

Combining these terms, the **total area of the fin = 22.1 - 9.82 - .38 - .28 - .2 - 0.6** = **10.82**

Calculation of Cx

To calculate **x** dimension of the fin centroid, we need to first calculate **Cx** (the **x** coordinate of the centroid) of each component for which we just found the area. For all the circles, **Cx** is just the midpoint value, so:

A_Cx = **3.75**

B_Cx = **5**

C_Cx = **1.5**

D_Cx = **2.53**

E_Cx = **3.96**

For the triangle **Cx**, we average the **x** coordinates of its vertices:

$$C_x = \frac{(V1_x + V2_x + V3_x)}{3}$$

F_Cx = (7.8 + 6.3 + 6.0) / 3 = **6.7**

Cx of the fin is therefore (note that we are subtracting instead of adding):

$$Cx = \frac{((A\_Cx \times Half\_A\_Area) - (B\_Cx \times Half\_B\_Area) - (C\_Cx \times C\_Area) - (D\_Cx \times D\_Area) - (E\_Cx \times E\_Area) - (F\_Cx \times F\_Area))}{TotalFinArea}$$

= ((3.75*22.1) - (5*9.82) - (1.5*.38) - (2.53*.28) - (3.96*.2) - (6.7*.6)) / 10.82

= ((82.88) - (49.1) - (.57) - (.71) - (.79) - (4.02)) / 10.82

= **2.56**

The dashed purple line on the colored figure above shows the location of **Cx**. Now we can calculate ***ε*** using the formula we have seen before:

$$\varepsilon = (\frac{C_x}{RC}) - 0.25$$

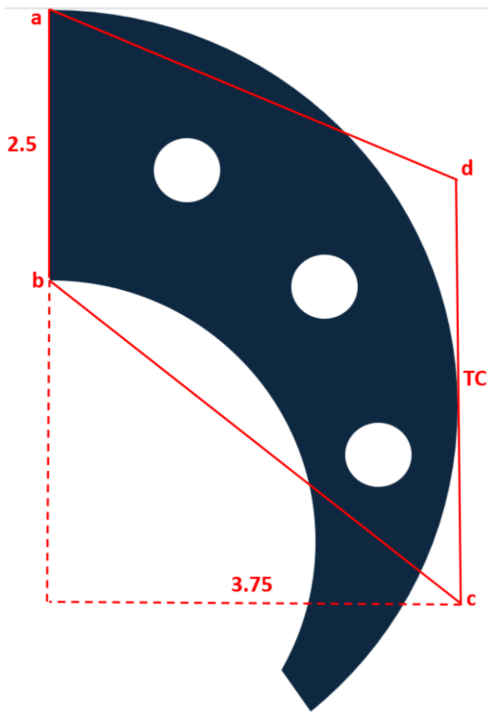
(recalling that ***ε*** is a measure of distance expressed as a fraction of the whole root chord).

ε = **2.56/2.5 – 0.25** = **0.77**

Now let's calculate this fin's flutter velocity. Since we do not need sweep length (we have already calculated **Cx**), the only fin parameter we don't have at this point is the tip chord length. We could use the width of the end of the fin arc, which is the square root of the sum of the squares of the vertex coordinate differences:

$\sqrt{((6.3-6)^2 + (2.75-2.29)^2)}$ = **0.55**.

However, the curved sweep of the fin to the rear suggests a different approach would be a better choice. As we did for elliptical fins, we will find a "pseudo" tip chord length of a trapezoidal fin that gives the same area as the bat wing fin. We do this by setting the value of the area of the bat wing fin equal to the area of a trapezoidal fin with the same root chord and semi-span length (height), and then solving for tip chord length. This process is depicted in the figure below. Trapezoid ***abcd*** has the same area as the bat wing, and the "pseudo" tip chord length is ***dc***, calculated as shown.



$$TC = dc = (\frac{Bat\,Wing\,Fin\,Area}{Height} \times 2) - Root\,Chord\,Length$$

TC = (((10.82 / 3.75) * 2) - 2.5) = 3.27. This is the value we will use to calculate **V_f**.

Now let's assume we are designing a rocket with bat wing fins of this shape whose dimension are in inches, and that we have chosen to use balsa fins that are 1/16" in thickness. For this exercise, assume that the launch altitude is 400 ft, launch temperature is 65 deg. F, and that the predicted height of our rocket at maximum velocity is 1000 ft. Summarizing the inputs to the fin flutter velocity calculator spreadsheet (overridden values shown in **red**):

		Data to be Entered	**Imperial Units**	
Launch Site Data				
MaxV	=	maximum predicted rocket velocity	250	ft/sec
AMaxV	=	predicted altitude at predicted maximum rocket velocity	1000	ft
LSA	=	launch site altitude (ASL)	400	ft
TLS	=	temperature at launch site	65	deg F
Use Default Temp?	=	Use Default Sea-Level Temp (**DST**) or Launch Site Temp (**LST**)	DST	DST or LST
Fin Geometry Data				
t	=	fin thickness	0.0625	in
m	=	fin sweep length	Don't Care	in
TC	=	tip chord length	3.27	in
RC	=	root chord length	2.5	in
SSL	=	semi span length (height)	3.75	in
G_E (shear modulus)		Shear Modulus (doubled in calculation if tip-to-tip reinforcement)	33359	psi
T2T	=	Tip-to-Tip reinforcing present?	NO	YES or NO
		Calculated Values		
Cx (for trapezoidal fins)		$C_x = \frac{(2 \times TC \times m) + TC^2 + (m \times RC) + (TC \times RC) + RC^2}{3(TC + RC)}$	2.56	in
(edit formula for other fin shapes)				
t/c (thickness ratio)		**fin thickness / root chord length**	0.0250	
λ (lambda) (taper ratio)		**tip chord length / root chord length**	1.3080	
(create "pseudo" tip chord if nec.)				
Fin Area (trapezoidal fin)		$Area = Height \times \frac{Tip\,Chord\,Length + Root\,Chord\,Length}{2}$	10.820	in sq
(edit formula for other fin shapes)				
A (aspect ratio)		**(semi-span length or height)^2 / fin area**	1.2997	

The $\mathbf{V}_f$ result given these inputs is 85 ft/sec (about 58 mph). That's very low. If we double the balsa thickness (to 1/8"), $\mathbf{V}_f$ increases to 240 fps (163 mph). If we change the material to 1/16" birch aircraft plywood, $\mathbf{V}_f$ is 138 ft/sec (94 mph). Doubling the plywood thickness to 1/8" increases $\mathbf{V}_f$ to 392 ft/sec (267 mph). Small model rockets rarely exceed 250 mph, but larger and higher power rockets can approach or exceed Mach 1. Therefore, in order make the best choices for best fin material and thickness, we need to know how fast our rocket is expected to fly on the largest motor we intend to use. See the original article (*PoF* #615[1]) for a discussion of how we might make these choices.

OpenRocket and RockSim both support custom fin shapes, but to my knowledge, neither simulator considers fin holes to have aerodynamic significance (if fin holes are allowed at all). We can manually adjust fin weight in the simulator to account for the holes, but that's about it. Also, the aerodynamic effect of fin holes is unpredictable using simple analysis tools like those employed here. In general, unusual fin shapes may have unexpected flight results. Novel fin designs should be flown with caution, and only when it is safe to do so. Also, it is a good idea to alert the RSO when testing a new fin design for the first time.

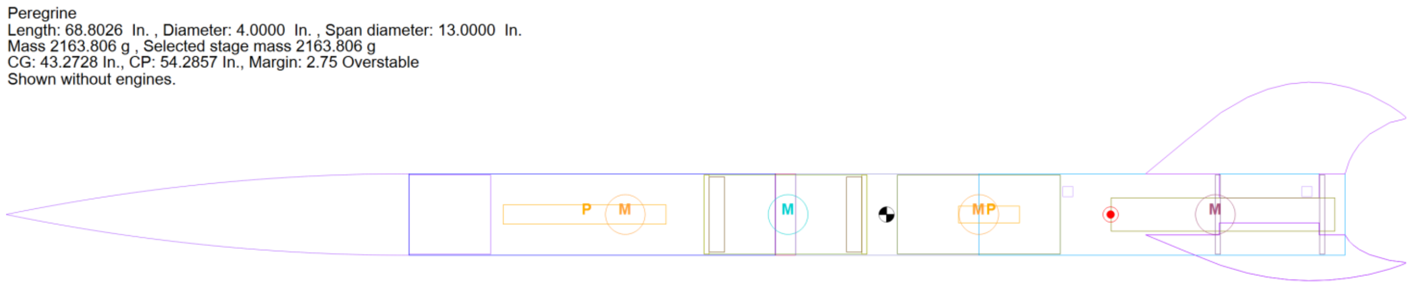
In this article, we calculated the centroids using first principles, but a lot of helpful guidance can be found at Wikipedia's "List of Centroids" URL: https://en.wikipedia.org/wiki/List_of_centroids.

Custom fins are readily designed using tools like Fusion 360, which can export either DXF (for laser cutting) or STL (for 3D printing) files. A lot of guidance on this, material selection, and related subjects can be found online.

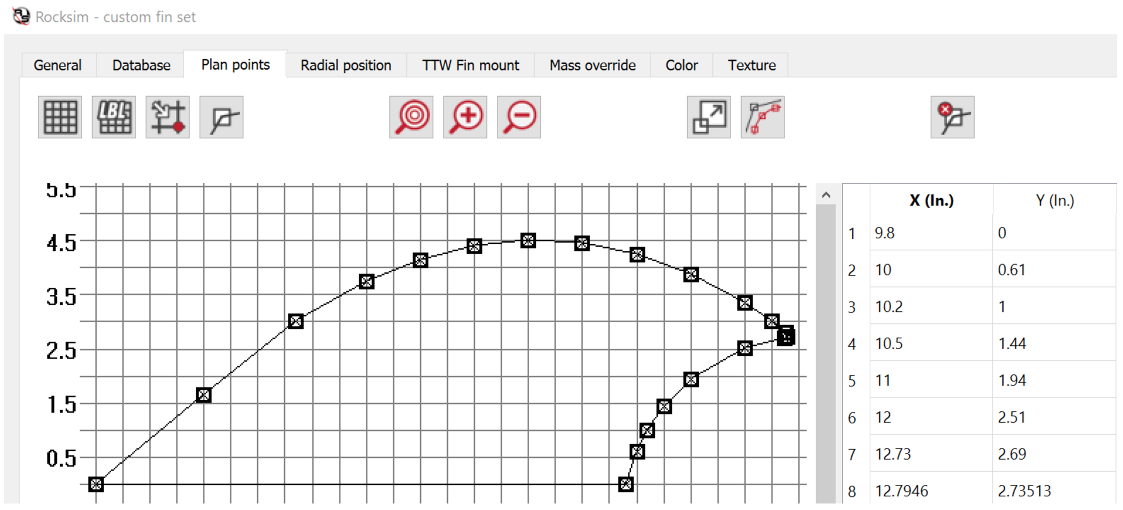
Analyzing Arbitrary Fin Shapes with Multi-Sided Polygon Triangulation

Note: This section will ultimately involve some computer programming. Even if that is not your cup of tea, I hope that you will find useful information here.

Some fins designs may be sufficiently complex that their analysis is impractical using the methods we have discussed so far. Consider, for example, the fin design on the Apogee Peregrine[5] rocket. This is a commercially available 4-inch rocket kit with an unusual fin design, as shown below:


Peregrine
Length: 68.8026 In. , Diameter: 4.0000 In. , Span diameter: 13.0000 In.
Mass 2163.806 g , Selected stage mass 2163.806 g
CG: 43.2728 In., CP: 54.2857 In., Margin: 2.75 Overstable
Shown without engines.

In *Peak of Flight* #488[3], Tim Van Milligan demonstrated how complex curved fins such as these could be entered into RockSim by selecting vertices on a grid. These vertices are shown in the figure below. OpenRocket provides a similar mechanism for creating fins from a set of vertices.



	X (In.)	Y (In.)
1	9.8	0
2	10	0.61
3	10.2	1
4	10.5	1.44
5	11	1.94
6	12	2.51
7	12.73	2.69
8	12.7946	2.73513

This fin design may not correspond to easily manipulated geometric elements like circles, triangles, rectangles, or ellipses. So, how do we calculate the area and centroid of fins like these? It turns out that we can generalize geometric decomposition to handle almost any fin design, from simple trapezoidal fins to the complex fin shown here. The basic idea follows the manual process we used for simple polygons:

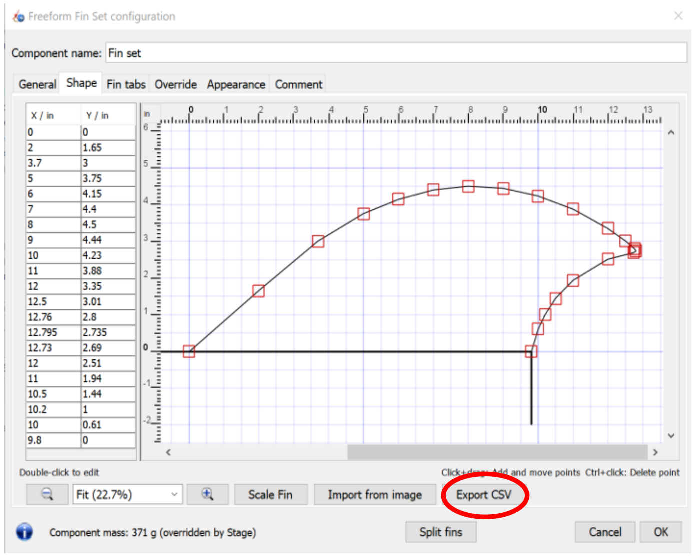
1. Define the fin outline with a set of vertices.
2. Find a set of triangles within the fin that use these vertices.
3. Calculate the area of each triangle from its vertices. The total fin area is the sum of the areas of the triangles.
4. Calculate **Cx** of each triangle. The **Cx** of the entire fin is the weighted average of the **Cx's** of each triangle.

Let's take each of these steps one-by-one.

Obtaining the Vertices

The vertices come straight from our simulator. For fins already defined by vertices, OpenRocket has an "**Export CSV**" button on its Freeform Fin Set Configuration page, as shown below.

[5] https://www.apogeerockets.com/Rocket-Kits/Skill-Level-3-Model-Rocket-Kits/Peregrine



Freeform Fin Set configuration

Component name: Fin set

General | Shape | Fin tabs | Override | Appearance | Comment

X / in	Y / in
0	0
2	1.65
3.7	3
5	3.75
6	4.15
7	4.4
8	4.5
9	4.44
10	4.23
11	3.88
12	3.35
12.5	3.01
12.76	2.8
12.795	2.735
12.73	2.69
12	2.51
11	1.94
10.5	1.44
10.2	1
10	0.61
9.8	0

Double-click to edit

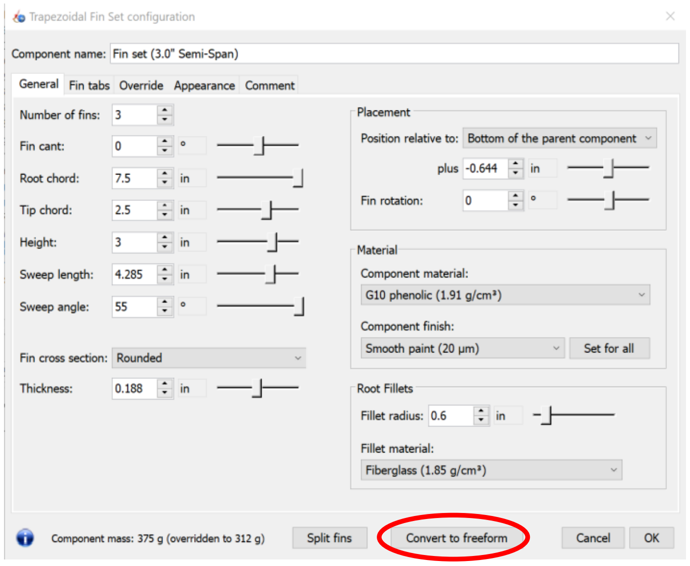
Click+drag: Add and move points Ctrl+click: Delete point

Fit (22.7%) | Scale Fin | Import from image | Export CSV

Component mass: 371 g (overridden by Stage)

Split fins | Cancel | OK

For trapezoidal fins, OpenRocket has a “**Convert to Freeform**” button on its Trapezoidal Fin Set Configuration page:



Trapezoidal Fin Set configuration

Component name: Fin set (3.0" Semi-Span)

General | Fin tabs | Override | Appearance | Comment

Number of fins:	3
Fin cant:	0 °
Root chord:	7.5 in
Tip chord:	2.5 in
Height:	3 in
Sweep length:	4.285 in
Sweep angle:	55 °
Fin cross section:	Rounded
Thickness:	0.188 in

Placement

Position relative to:	Bottom of the parent component
plus	-0.644 in
Fin rotation:	0 °

Material

Component material: G10 phenolic (1.91 g/cm³)

Component finish: Smooth paint (20 μm) | Set for all

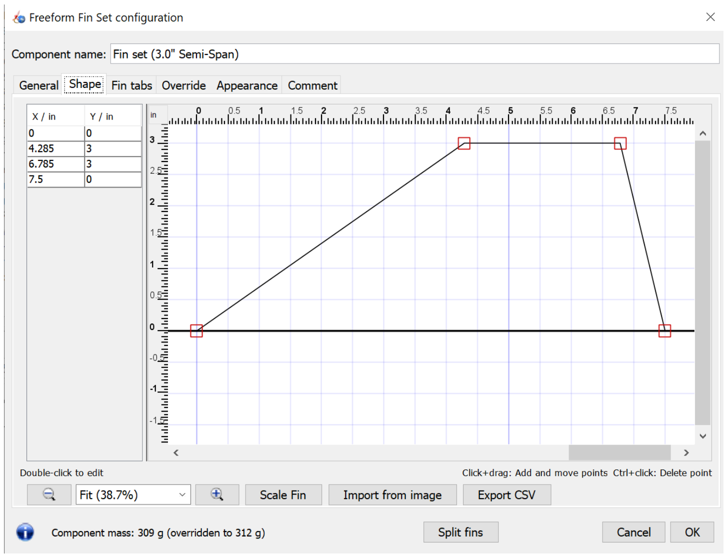
Root Fillets

Fillet radius:	0.6 in
Fillet material:	Fiberglass (1.85 g/cm³)

Component mass: 375 g (overridden to 312 g)

Split fins | Convert to freeform | Cancel | OK

which will produce a freeform page from which the vertices can be exported as above:



Freeform Fin Set configuration

Component name: Fin set (3.0" Semi-Span)

General | Shape | Fin tabs | Override | Appearance | Comment

X / in	Y / in
0	0
4.285	3
6.785	3
7.5	0

Double-click to edit

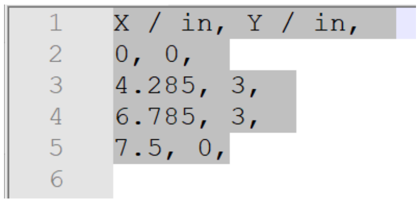
Click+drag: Add and move points Ctrl+click: Delete point

Fit (38.7%) | Scale Fin | Import from image | Export CSV

Component mass: 309 g (overridden to 312 g)

Split fins | Cancel | OK

RockSim does not make things quite this easy, but since OpenRocket can open a RockSim file to perform the export, you can use either simulator to design your fins. The vertex file exported by OpenRocket looks like this (for the four-vertex simple fin shown above):



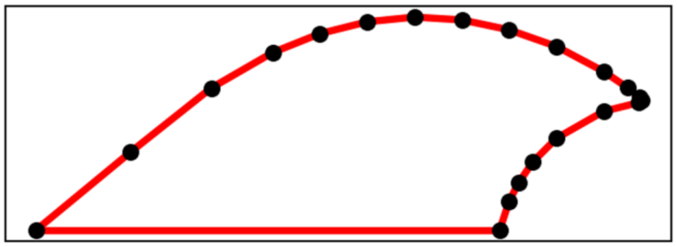
```
X / in, Y / in,
0, 0,
4.285, 3,
6.785, 3,
7.5, 0,

```

This text file has a header row that identifies the X and Y coordinate columns, as well as the unit system. Each row contains one vertex (with the x and y coordinates separated by a comma), and for some reason, an extra comma at the end of the row.

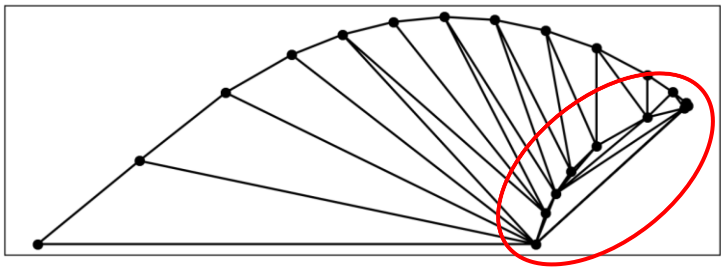
Creating the Triangles

The exported fin vertices should define a closed polygon. The more vertices there are, the smoother any fin edge curve will be. For example, the Peregrine rocket fins define twenty-one vertices:



As we saw earlier, for small sets of vertices, it is possible to manually construct a set of triangles within the fin polygon. However, as the number of vertices grows, the manual process becomes tedious at best, and at some point, overwhelming. Fortunately, triangulation of polygons is a well-studied area of computational geometry. Much of that work revolves around the inherent complexity of the problem (how fast we can generate the triangles). As it turns out, modern computer graphics is based on creating and processing millions of triangles many times per second.

One popular algorithm for polygon triangulation was created by Boris Delaunay in 1934[6]. A desirable aspect of Delaunay's approach is that it tries to minimize the number of triangles with very small angles, so called "sliver triangles." However, the basic Delaunay algorithm treats the input vertices as just a set of vertices. This results in the algorithm generating extra triangles outside the fin boundary when it encounters a concave shape, as demonstrated below:

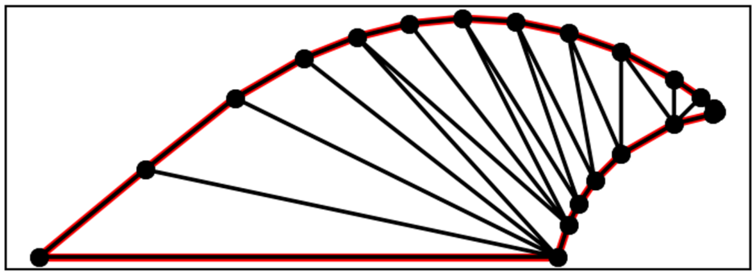


To address this problem, the "constrained" Delaunay triangulation algorithm[7] was developed. Constrained Delaunay triangulation assumes that the input vertices form a "planar straight-line graph", or PSLG, that defines a closed

[6] https://www.mathnet.ru/links/a0aeb3483e137913db801557182e6e35/im4937.pdf (in French)

[7] L.P. Chew, SCG '87: Proceedings of the Third Annual Symposium on Computational Geometry, October 1987, Pages 215–222

boundary, and requires additional input in the form of non-crossing line segments built from the vertex set, all in one plane, that define that boundary. If we apply the constrained Delaunay triangulation algorithm to our set of vertices and segments, we get the following set of triangles:



This is the result that we want, but we are going to have to do a bit of computer programming to get there. The good news is that well-developed (and free) software packages exist to perform most of what we need to accomplish. Let's work in Python, which has a rich set of software packages for scientific and engineering programming. You can download Python here[8]. A basic Python tutorial can be found here[9]. Help on installing Python packages can be found here[10]. There are also a wide selection of books and online resources for all things Python. Two of my two favorites among the many Python introductory books would be these[11]. We will initially use three Python software packages: `numpy` (a library for working with arrays), `matplotlib` (a library for creating graphs and charts), and `triangle` (the package that will perform the actual triangulation[12]). By using these packages, our (heavily commented) code to perform the triangulation is short:

```
import numpy as np
import matplotlib.pyplot as plt
import triangle as tr

# Read in the vertex file exported by OpenRocket
verts = np.genfromtxt("peregrinefins.csv", delimiter=",")
verts = np.delete(verts, 0, 0)  # delete the header row of the csv file
verts = np.delete(verts, 2, 1)  # delete third column of csv file (created by extra comma)
segs = []                       # must specify segments as well as verts to create a PSLG
for i in range (len(verts)):    # create segments of a closed PSLG from list of vertices
    seg = []
    seg.append(i)
    if (i == (len(verts)-1)):
        seg.append(0)
    else:
        seg.append(i+1)
    segs.append(seg)
tr_input = dict(vertices=verts, segments=segs)  # triangle expects Python dict as input
tr_output = tr.triangulate(tr_input,'p')        # make triangles. 'p' = input is a PSLG
tris = tr_output['triangles'].tolist()          # convert output of triangle to Python list
tr.compare(plt, tr_input, tr_output)            # set up what we are going to plot
plt.show()
```

If we run this code, we get the following output:

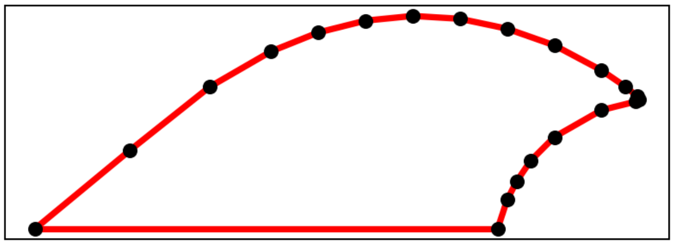
[8] https://www.python.org/downloads/

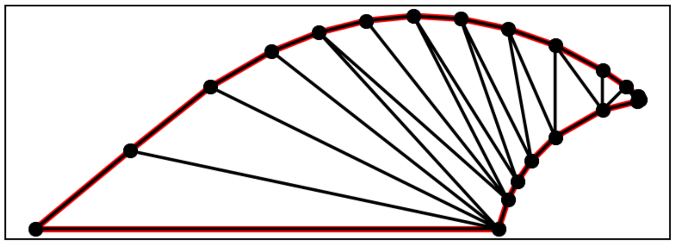
[9] https://www.w3schools.com/python/python_intro.asp

[10] https://python.land/virtual-environments/installing-packages-with-pip#Python_Install_Pip

[11] https://www.amazon.com/gp/product/B071Z2Q6TQ and https://www.amazon.com/Python-Crash-Course-Eric-Matthes/dp/1718502702

[12] The Python triangle package is actually a Python "wrapper" around Jonathan Shewchuk's two-dimensional mesh generator and Delaunay triangulator library, written in C (see https://www.cs.cmu.edu/~quake/triangle.html).





It’s always a good idea to display the triangles that are produced as a check that everything went according to plan. If we are curious, we can see how many triangles were produced, and what the list of triangles looks like, by adding two lines (the last two lines shown below) to our code:

```
tr_input = dict(vertices=verts, segments=segs)  # triangle expects a Python dict as input
tr_output = tr.triangulate(tr_input,'p')        # make triangles. 'p' = input is a PSLG
tris = tr_output['triangles'].tolist()          # convert output of triangle to Python list
# tr.compare(plt, tr_input, tr_output)          # set up what we are going to plot
# plt.show()
print ("Number of Triangles Produced = ", len(tris))
print (tris)
```

When we run the revised code, we get the following output (in addition to the plot above):

```
Number of Triangles Produced =  19
[[1, 0, 20], [3, 2, 20], [20, 2, 1], [4, 20, 19], [3, 20, 4], [19, 6, 5], [5, 4,
 19], [7, 6, 18], [6, 19, 18], [17, 8, 7], [15, 14, 11], [16, 15, 9], [17, 16, 8
], [15, 10, 9], [9, 8, 16], [14, 12, 11], [11, 10, 15], [14, 13, 12], [17, 7, 18
]]
```

Each of the nineteen triangles is defined by three integers. These integers represent indices into the vertex array that we used as input to the triangulation process. For example, adding the code below will print the vertices of the first triangle (whose vertex indices are: [**1, 0, 20**]).

```
print("Number of Triangles Produced = ", len(tris))
print(tris)
print("First Triangle Vertices = (", \
       verts[1].tolist(), ", ", verts[0].tolist(), ", " , verts[20].tolist(),")")
```

Running this code prints (in addition to the previous output) the vertices of our first triangle (the one in the bottom left):

```
First Triangle Vertices:( [2.0, 1.65] ,  [0.0, 0.0] ,  [9.8, 0.0]
```

Calculating Fin Area and Cx

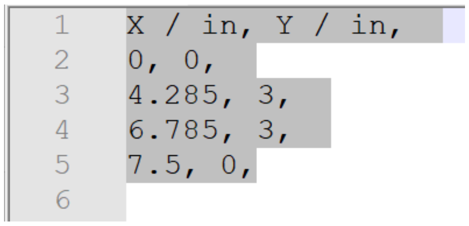
Now that we have our triangles, we already know how to calculate fin area and **Cx,** given each triangle’s vertices. We just need to write the code to do this. Let’s add this code to what we already have:

```
# for each of our triangles, compute an area and a Cx
tri_areas = []      # this list will hold the areas of each triangle
tri_Cxs = []        # this list will hold the Cx’s of each triangle
fin_area = 0
fin_Cx = 0
for tri in tris:
    A = verts[tri[0]]       #extract the vertices of this triangle
    B = verts[tri[1]]
    C = verts[tri[2]]
    # compute triangle area; 0 is the index of the x coord; 1 is the index of the y coord
    ar = ((A[0]*(B[1]-C[1])) + (B[0]*(C[1] - A[1])) + (C[0]*(A[1] - B[1])))/2
    tri_areas.append(ar)    # add the area of this triangle to our list of areas
    fin_area += ar          # add the area of this triangle to the total fin area
    # compute triangle Cx; 0 is the index of the x coord
    tri_Cx = (A[0] + B[0] + C[0]) / 3
    tri_Cxs.append(tri_Cx)  # add the Cx of this triangle to our list of Cx's
# compute Fin Cx by taking the weighted average of all the triangle Cx's
for i in range(len(tri_areas)):             # first get the numerator
    fin_Cx += (tri_Cxs[i] * tri_areas[i])   # use fin_Cx to hold intermediate result
fin_Cx = (fin_Cx / fin_area)                # now divide by the total fin area
print ("Fin Area = ", "{:2.2f}".format(fin_area))
print ("Fin Cx = ", "{:2.2f}".format(fin_Cx))
```

When we run this code, we get:

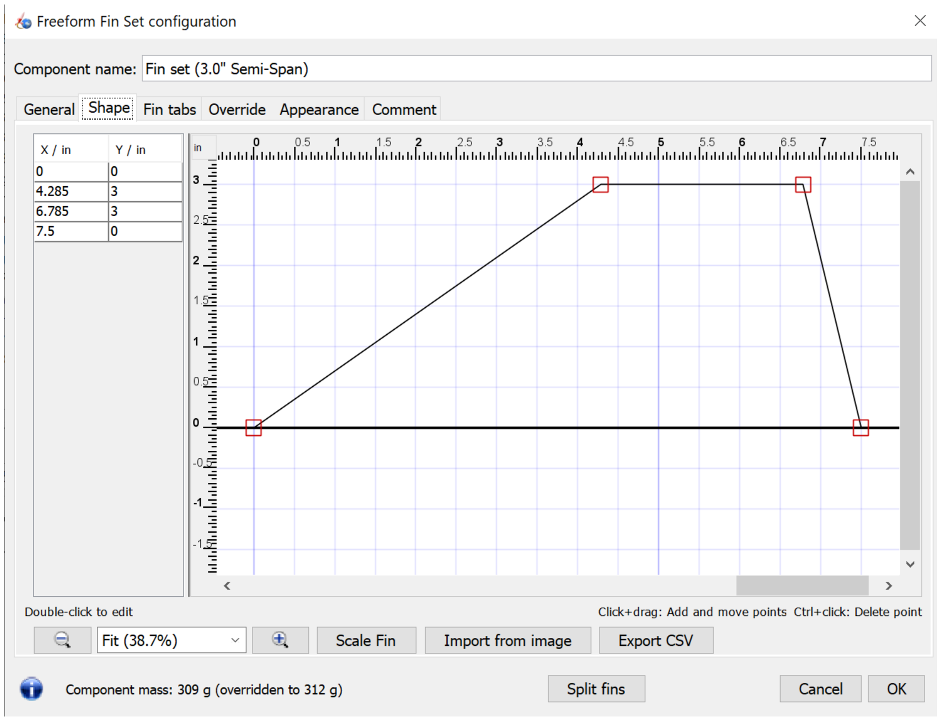
```
Fin Area =  35.85
Fin Cx =  6.78
```

How cool is that? This program (**Fin_Area_and_Cx.py**) is available for download from GitHub[2]. It is worth noting that this program can analyze any fin set for which we can produce a set of vertices, from very simple trapezoidal fins to curved fins with holes. Let's convince ourselves that this is true, starting with the original rocket/fin "worked example" in *Peak of Flight* #615[1]. The vertex file produced by that fin is:



```
X / in, Y / in,
0, 0,
4.285, 3,
6.785, 3,
7.5, 0,
```

The fin configuration that produced this file is:



Freeform Fin Set configuration

Component name: Fin set (3.0" Semi-Span)

X / in	Y / in
0	0
4.285	3
6.785	3
7.5	0

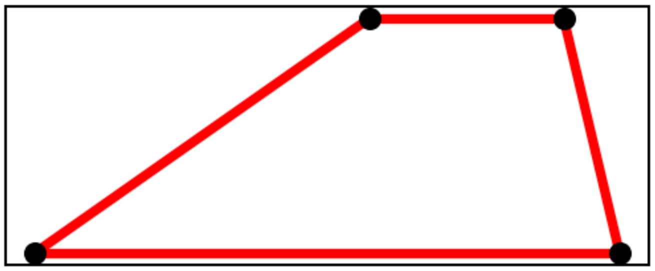
Double-click to edit

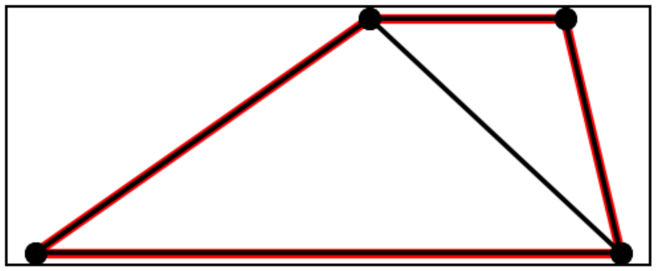
Click+drag: Add and move points Ctrl+click: Delete point

Fit (38.7%)

Component mass: 309 g (overridden to 312 g)

We only need two triangles to characterize this fin, as shown below:





To compute **V***f* , we need to augment our program (using the same rocket and motor selections as the original *PoF* article). The entire program so far is shown below. Notice that we have imported another package, “`math`”, which we need to use square root and the constant pi. Also, in addition to computing **V***f* and **Cx**, we have added an inputs section, as well as ensuring that the program prints out the correct units for each result.

```
import numpy as np
import matplotlib.pyplot as plt
import triangle as tr
import math

# Start of Inputs
VERBOSE = 0    # '0' only prints final results; 1 prints intermediate results
# Inputs to be provided
Units = "Imperial" # "Imperial" = inches,feet,psi,deg.F; "SI" = cm,meters,KPa,deg.C
# Input provided below must match selection above
# Rocket and Launch Site Specs
MaxV = 1500          # maximum predicted rocket velocity
AMaxV = 14000        # predicted altitude at predicted maximum rocket velocity (AGL)
LSA = 4500           # launch site altitude (ASL)
TLS = 65             # Temp (Fahrenheit or Centigrade, depending upon selected units)
DEF = "DST"          # Use Default Sea-Level Temp ("DST") or Launch Site Temp ("LST")
#Fin Specs
Thickness = 0.1875   # Fin Thickness
TC = 2.5             # if TC == -1, calculate TC (or a Pseudo TC)
RC = 7.5             # Root Chord length
Height = 3           # Fin Height
GE = 600000          # Shear Modulus
T2T  = "NO"          # Tip-to-Tip reinforcing present? "YES" or "NO"
Fin_Vertex_File_Name = "trap.csv" # Name of csv file output from OpenRocket
# End of Inputs

# Read in the vertex file exported by OpenRocket (see articles for explanation)
verts = np.genfromtxt(Fin_Vertex_File_Name, delimiter=",")
verts = np.delete(verts, 0, 0)  # delete the header row of the csv file
verts = np.delete(verts, 2, 1)  # delete third column of csv file (created by extra comma)
segs = []                       # must specify segments and vertices to create a PSLG
for i in range (len(verts)):    # create segments of a closed PSLG from list of vertices
    seg = []
    seg.append(i)
    if (i == (len(verts)-1)):
        seg.append(0)
    else:
        seg.append(i+1)
    segs.append(seg)
tr_input = dict(vertices=verts, segments=segs)  # triangle expects Python dict as input
tr_output = tr.triangulate(tr_input,'p')        # make triangles. 'p' says input is a PSLG
tris = tr_output['triangles'].tolist()          # convert output of triangle to Python list
# print("Number of Triangles Produced = ", len(tris))
# print (tris)
# print ("First Triangle Vertices = (", \
#        verts[1].tolist(), ", " , verts[0].tolist(), ", " , verts[20].tolist(),")")
# next two lines moved to end of file so plot can stay visible until closed
# tr.compare(plt, tr_input, tr_output)            # set up what we are going to plot
# plt.show()                                      # create the plot

# for each generated triangle, compute an area and a Cx
tri_areas = []      # this list will hold the area of each triangle
tri_Cxs = []        # this list will hold the Cx of each triangle
fin_area = 0
fin_Cx = 0
for tri in tris:          # for every generated triangle
    A = verts[tri[0]]     # extract the vertices of this triangle
    B = verts[tri[1]]
    C = verts[tri[2]]
    # compute triangle area; 0 is index of vertex x coord; 1 is index of vertex y coord
    ar = ((A[0]*(B[1]-C[1])) + (B[0]*(C[1] - A[1])) + (C[0]*(A[1] - B[1])))/2
    tri_areas.append(ar)    # add the area of this triangle to our list of areas
    fin_area += ar          # add the area of this triangle to the total fin area
    # compute triangle Cx; 0 is the index of the vertex x coord
    tri_Cx = (A[0] + B[0] + C[0]) / 3
    tri_Cxs.append(tri_Cx)  # add the Cx of this triangle to our list of Cx's
# compute Fin Cx by taking the weighted average of all the triangle Cx's
for i in range(len(tri_areas)):  # first get the numerator (weighted sum of areas)
    fin_Cx += (tri_Cxs[i] * tri_areas[i])
fin_Cx = (fin_Cx / fin_area)     # now divide by the total fin area to get Cx

# Compute Vf; first compute all intermediate values
Fin_Eps = (fin_Cx / RC) - 0.25   # compute epsilon (see article for definition)
if (TC < 0): # if TC is entered as -1, compute TC or pseudo TC
    TC = (((fin_area / Height) * 2) - RC)
ThicknessRatio = (Thickness / RC)      # compute three fin ratios
Lambda = (TC / RC) # if TC = 0, Lambda will also be zero (triangular fin)
AspectRatio = ((Height * Height) / fin_area)
if (Units == "Imperial"): # set constants and suffices to Imperial Units
    p0 = 14.696
    DST_Base_Temp = 59
    Temp_Dec_Per_Unit = 0.00356
    SoS_Mult = 49.03
    Low_Temp = 459.7
    T0 = 518.7
    velsuf = "ft/sec"
    finsuf = "in"
    altsuf = "feet"
    tempsuf = "deg F"
    GEsuf = "psi"
    areasuf = "sq in"
else: # set constants and suffices to SI Units
    p0 = 101.325
    DST_Base_Temp = 15
    Temp_Dec_Per_Unit = .0065
    SoS_Mult = 20.05
    Low_Temp = 273.16
    T0 = 288.16
    velsuf = "meters/sec"
    finsuf = "cm"
    altsuf = "meters"
    tempsuf = "deg C"
    GEsuf = "KPa"
    areasuf = "sq cm"
DN = (24 * Fin_Eps * 1.4 * p0) / math.pi      # compute "denominator constant"
```

```
if(DEF=="DST"):     # Use Default Sea-Level Temp ("DST") as base
    Temp = (DST_Base_Temp - (Temp_Dec_Per_Unit * (LSA + AMaxV)))
else:  # Use Launch Site Temp ("LST") as base
    Temp = (TLS-(Temp_Dec_Per_Unit * (AMaxV)))
Spd_of_Sound = SoS_Mult * math.sqrt(Low_Temp + Temp) # Compute speed of sound
airP = p0 * pow(((Temp  + Low_Temp)/T0),5.256)             # Compute air pressure
Term1 = (DN * pow(AspectRatio, 3)) / (pow(ThicknessRatio, 3) * (AspectRatio + 2))
Term2 = (Lambda + 1)/2
Term3 = (airP / p0)
if (T2T == "YES"): # if tip to tip reinforcing present, double GE
      GE = 2 * GE
Vf = Spd_of_Sound * math.sqrt(GE/(Term1 * Term2 * Term3))
Margin = Vf - MaxV  # compute safety margin (see article)
Margin_Pct = 100 * ((Vf-MaxV)/MaxV)

# print the input for verification
print ("Unit System = ", Units)
# Rocket and Launch Site Specs, as provided
print ("****Rocket and Launch Site Specs****")
print ("MaxV = ", "{:2.1f}".format(MaxV), velsuf)
print ("AMaxV = ", "{:2.1f}".format(AMaxV), altsuf)
print ("LSA = ", "{:2.1f}".format(LSA), altsuf)
print ("TLS = ", "{:2.1f}".format(TLS), tempsuf)
print ("DEF = ", DEF)
print ("****Fin Specs****")
print ("RC = ", "{:2.3f}".format(RC), finsuf)
print ("Height = ", "{:2.3f}".format(Height), finsuf)
print ("Thickness = ", "{:2.4f}".format(Thickness), finsuf)
print ("TC = ", "{:2.3f}".format(TC), finsuf)
if (T2T == "YES"):
    print ("Tip-to-tip reinforcing present; GE doubled to: ","{:2.1f}".format(GE),GEsuf)
else:
    print ("GE = ","{:2.1f}".format(GE),GEsuf)
print ("T2T = ", T2T)
print ("Fin_Vertex_File_Name = ", Fin_Vertex_File_Name)
print()
if (VERBOSE): # print all the intermediate values, if VERBOSE is true
    print ("Fin Eps = ", "{:2.3f}".format(Fin_Eps))
    print ("TC or Pseudo TC = ", "{:2.2f}".format(TC), finsuf)
    print ("Thickness Ratio = ", "{:2.3f}".format(ThicknessRatio))
    print ("Lambda = ", "{:2.3f}".format(Lambda))
    print ("Aspect Ratio = ", "{:2.3f}".format(AspectRatio))
    print ("DN = ", "{:2.2f}".format(DN), GEsuf)
    print ("Temp at MaxV = ", "{:2.2f}".format(Temp), tempsuf)
    print ("Spd_of_Sound = ", "{:2.2f}".format(Spd_of_Sound), velsuf)
    print ("airP = ", "{:2.2f}".format(airP), GEsuf)
    print ("Term1 = ", "{:2.2f}".format(Term1), GEsuf)
    print ("Term2 = ", "{:2.2f}".format(Term2))
    print ("Term3 = ", "{:2.2f}".format(Term3))
# always print the important stuff
print ("Fin Area = ", "{:2.2f}".format(fin_area), areasuf)
print ("Fin Cx = ", "{:2.2f}".format(fin_Cx), finsuf)
print ("Vf = ", "{:2.1f}".format(Vf), velsuf)
print ("Margin = ", "{:2.1f}".format(Margin), velsuf)
print ("Margin% = ", "{:2.1f}%".format(Margin_Pct))
# plot moved to here so we wouldn't have to close plot to see results
tr.compare(plt, tr_input, tr_output)            # set up what we are going to plot
plt.show()                                      # create the plot
```

When we run this code, we obtain the following output, confirming that we get the same answer as before:

```
Number of Triangles Produced =  2
[[1, 0, 3], [3, 2, 1]]
Unit System =  Imperial
****Rocket and Launch Site Specs****
MaxV =  1500.0 ft/sec
AMaxV =  14000.0 feet
LSA =  4500.0 feet
TLS =  65.0 deg F
DEF =  DST
****Fin Specs****
RC =  7.500 in
Height =  3.000 in
Thickness =  0.1875 in
TC =  2.500 in
GE =  600000.0 psi
T2T =  NO
Fin_Vertex_File_Name =  trap.csv

Fin Area =  15.00 sq ft
Fin Cx =  4.49 in
Vf =  2618.1 ft/sec
Margin =  1118.1 ft/sec
Margin% =  74.5%
```

Note that printing of the intermediate calculations can be turned off by setting the constant “VERBOSE” to 0.

Now, let’s go back and run this code on the Apogee Peregrine fin set. Do to this, we need to pick a motor. The motor mount tube of the Peregrine is 38mm, and 11 inches long. The Peregrine is intended for dual deployment, so we will be using a plugged motor. The highest total impulse 38mm motors that fit within the motor mount length are the Cesaroni I236-17A Blue Streak and the AeroTech I59WN-P. If we simulate a launch (using RockSim) with these motors, we get the following results:

Engines loaded	**Optimal delay**	**Max. altitude Feet**	**Max. velocity Feet / Sec**	**Max. acceleration Gees**
[I236BS-0]	10.53	2544.09	463.44	11.44
[I59WN-0]	6.58	2849.46	285.61	5.68

If we enter the highest velocity data (the I236BS) into our program (again turning off the “VERBOSE” switch in the code) and calculate **V*f***, we get:

```
Unit System =  Imperial
****Rocket and Launch Site Specs****
MaxV =  464.0 ft/sec
AMaxV =  2544.0 feet
LSA =  4500.0 feet
TLS =  65.0 deg F
DEF =  DST
****Fin Specs****
RC =  9.800 in
Height =  4.500 in
Thickness =  0.2500 in
TC =  6.133 in
GE =  89000.0 psi
T2T =  NO
Fin_Vertex_File_Name =  peregrinefins.csv

Fin Area =  35.85 sq ft
Fin Cx =  6.78 in
Vf =  757.6 ft/sec
Margin =  293.6 ft/sec
Margin% =  63.3%
```

These data tell us that any 38mm H or I motor that fits within the motor mount dimensions will probably fly well in the Peregrine. If we are willing for the motor case to extend past the motor mount by up to an inch, these motors have the highest total impulse (RockSim launch simulation data also shown):

Engines loaded	Optimal delay	Max. altitude Feet	Max. velocity Feet / Sec	Max. acceleration Gees
[I242WH-0]	11.82	3470.01	565.26	9.77
[I303BS-0]	12.01	3466.34	587.80	14.94
[I435T-0]	12.36	3580.77	623.42	28.52
[I284W-0]	12.10	3527.40	595.57	17.17

If we enter the highest velocity data for these motors (the I435T) into our program and calculate $\mathbf{V}_f$, we get:

```
Unit System =  Imperial
****Rocket and Launch Site Specs****
MaxV =  623.5 ft/sec
AMaxV =  3581.0 feet
LSA =  4500.0 feet
TLS =  65.0 deg F
DEF =  DST
****Fin Specs****
RC =  9.800 in
Height =  4.500 in
Thickness =  0.2500 in
TC =  6.133 in
GE =  89000.0 psi
T2T =  NO
Fin_Vertex_File_Name =  peregrinefins.csv

Fin Area =  35.85 sq ft
Fin Cx =  6.78 in
Vf =  769.8 ft/sec
Margin =  146.3 ft/sec
Margin% =  23.5%
```

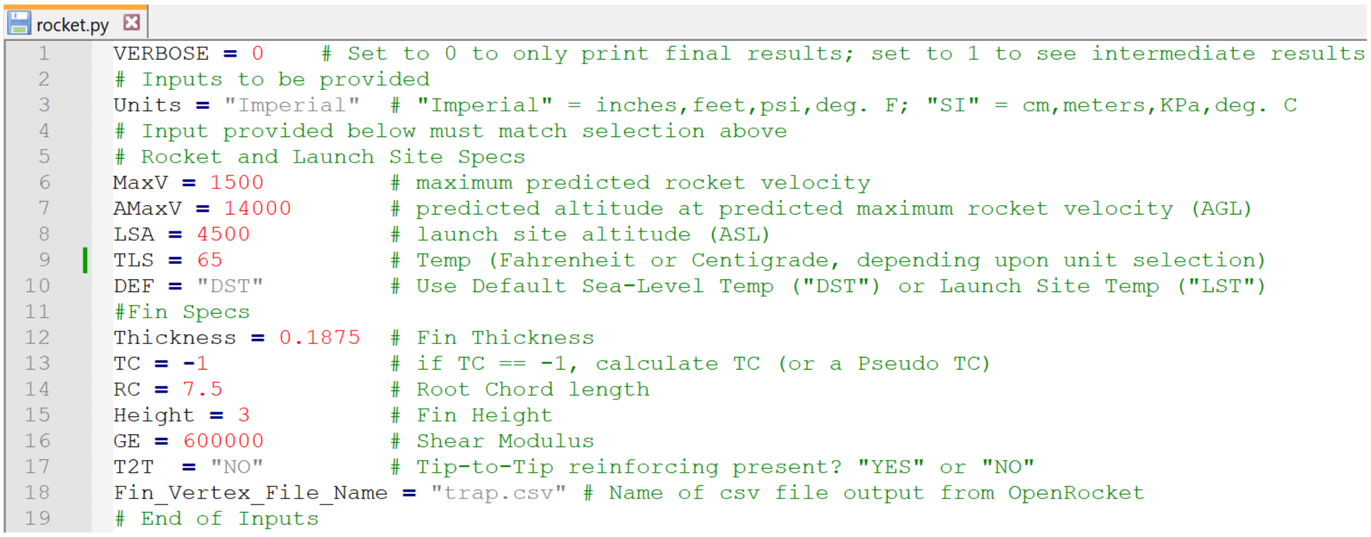
This is right on the edge of a safe flight, even on a cool day. The other motors are likely fine under most flight conditions, unless it's a super-hot day. If we want to stretch to a J motor, we will need to carefully select one with a slow burn rate at the low end of the J total impulse range, e.g., the Cesaroni P38-5G Mellow (J94), which simulates as follows:

Engines loaded	Optimal delay	Max. altitude Feet	Max. velocity Feet / Sec	Max. acceleration Gees
[J94MY-0]	10.07	4066.90	469.28	5.06

```
Unit System =  Imperial
****Rocket and Launch Site Specs****
MaxV =  469.3 ft/sec
AMaxV =  4067.0 feet
LSA =  4500.0 feet
TLS =  65.0 deg F
DEF =  DST
****Fin Specs****
RC =  9.800 in
Height =  4.500 in
Thickness =  0.2500 in
TC =  6.133 in
GE =  89000.0 psi
T2T =  NO
Fin_Vertex_File_Name =  peregrinefins.csv

Fin Area =  35.85 sq ft
Fin Cx =  6.78 in
Vf =  775.6 ft/sec
Margin =  306.3 ft/sec
Margin% =  65.3%
```

The final program code that performs these calculations (**Fin_Area_and_Cx_and_Input.py**) can be downloaded from GitHub[2]. This downloadable version of the code can also optionally read a configuration file (e.g., "rocket.py") that looks just like the input section of the code:



rocket.py

```
VERBOSE = 0    # Set to 0 to only print final results; set to 1 to see intermediate results
# Inputs to be provided
Units = "Imperial"  # "Imperial" = inches,feet,psi,deg. F; "SI" = cm,meters,KPa,deg. C
# Input provided below must match selection above
# Rocket and Launch Site Specs
MaxV = 1500           # maximum predicted rocket velocity
AMaxV = 14000         # predicted altitude at predicted maximum rocket velocity (AGL)
LSA = 4500            # launch site altitude (ASL)
TLS = 65              # Temp (Fahrenheit or Centigrade, depending upon unit selection)
DEF = "DST"           # Use Default Sea-Level Temp ("DST") or Launch Site Temp ("LST")
#Fin Specs
Thickness = 0.1875    # Fin Thickness
TC = -1               # if TC == -1, calculate TC (or a Pseudo TC)
RC = 7.5              # Root Chord length
Height = 3            # Fin Height
GE = 600000           # Shear Modulus
T2T  = "NO"           # Tip-to-Tip reinforcing present? "YES" or "NO"
Fin_Vertex_File_Name = "trap.csv" # Name of csv file output from OpenRocket
# End of Inputs
```

This change allows us to create a separate file for each rocket/fin/launch site combination, without having to significantly edit the actual program text itself. We just need to name each file appropriately, and change "rocket" in the four lines of the program code shown below to be the chosen filename prefix. Alternatively, we could leave the program code untouched, and just change each configuration's file name to "rocket.py" (only one at a time):

```
rocket = Path("rocket.py")
if rocket.is_file():      # if input file exists for initialization, read it
    # Read in the config file
    from rocket import *
```

The program code must change slightly to support this optional input file, as shown below (this change has been incorporated in the downloadable version of the code):

```
# start of change
from pathlib import Path  # we need the pathlib package to manage filename paths
# *** you must use the same name in the next four lines. i.e.,
    # peregrine_rocket = Path("peregrine_rocket.py")
    # if peregrine_rocket.is_file():   # if input file exists for initialization, read it
    #  # Read in the config file
    #  from peregrine_rocket import *
rocket = Path("rocket.py")
if rocket.is_file():      # if input file exists for initialization, read it
    # Read in the config file
    from rocket import *
else:                       # otherwise, initialize variables as follows
      VERBOSE = 0 # Set to 0 to only see final results; set to 1 to see full results
      # Inputs to be provided
      Units = "Imperial"  # "Imperial" = inches,feet,psi,deg.F; "SI" = cm,meters,KPa,deg.C
      # Input provided below must match selection above
      # Rocket and Launch Site Specs
      MaxV = 1500         # maximum predicted rocket velocity
      AMaxV = 14000       # predicted altitude at predicted maximum rocket velocity (AGL)
      LSA = 4500          # launch site altitude (ASL)
      TLS = 65            # Temp (Fahrenheit or Centigrade, depending upon unit selection)
      DEF = "DST"         # Use Default Sea-Level Temp ("DST") or Launch Site Temp ("LST")
      #Fin Specs
      Thickness = 0.1875  # Fin Thickness
      TC = -1             # if TC == -1, calculate TC or Pseudo TC
      RC = 7.5            # Root chord length
      Height = 3          # Fin Height
      GE = 600000         # Shear Modulus
      T2T = "NO"          # Tip-to-Tip reinforcing present? "YES" or "NO"
      Fin_Vertex_File_Name = "trap.csv" # Name of csv file output from OpenRocket
      # End of Inputs
# end of change
```

Finally, it is possible to edit the triangle package input to include holes in the fins, e.g.:

```
      tr_input = dict(vertices=verts, segments=segs, holes=holes)
```

This technique requires us to provide vertices and segments that define each hole, as well as a point within each hole. Completing this modification (more fully described in the Python triangle package documentation[13]) is left to the intrepid reader as an exercise.

End Note

To reiterate what I said in *PoF* #615, these methods only estimate the fin flutter velocity. The choice of input values (altitudes, temperatures, fin dimensions, and in particular, fin material shear modulus) will have a significant impact on the results. We will always strive to use the best data available, but it is important to remember that the final result is only an estimate.

About the Author

John Bennett is an engineering Professor Emeritus at the University of Colorado Boulder, and a TRA and NAR member in OregonRocketry. Using Brinley's book as a guide, he built and flew many Zn/S and KN0$_3$/sugar rockets as a teenager. In retirement, he is now rediscovering his love of amateur rocketry, this time with better supervision.

[13] https://rufat.be/triangle/examples.html