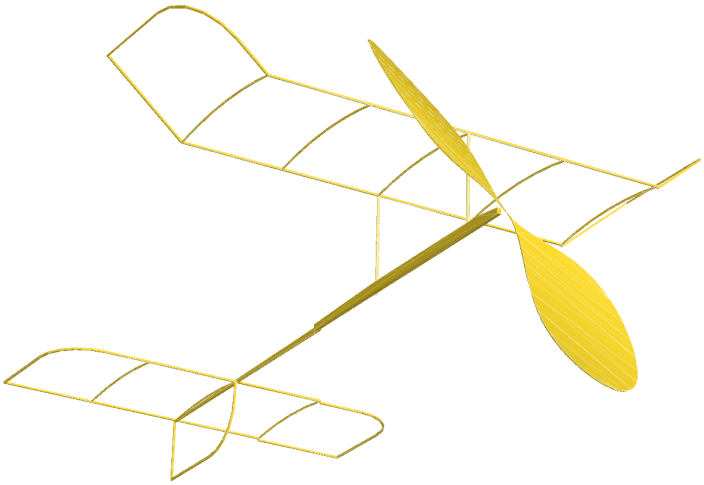
Designing an Indoor Model using OpenSCAD

Roie R. Black

January 13, 2021



Designing a new model airplane usually involves generating a plan of some sort, then constructing a prototype model from that plan. Of course you can use a pencil and paper to generate your plans, but if you think you might want to publish the plan, you will need to use some form of *Computer Aided Design* tool to produce your final plan. Unfortunately many popular CAD tools are complex, and often too expensive for the average modeler.

Having recently retired from teaching Computer Science, and finally getting back into model building, I decided to design a new indoor model for the *Limited Pennyplane* class. As part of the design process, I wanted to see that airplane in 3D even before I built the first prototype. I decided to use a different form of CAD tool: OpenSCAD [5], a tool designed for computer programmers!

While that description may discourage some folks from reading further, rest assured that this particular tool is simple enough that non-programmers can certainly master it. In fact, some teachers have successfully managed to get elementary school kids to use OpenSCAD to design simple 3D models.

OpenSCAD is an open-source (meaning free) 3D modeling program, available on all major platforms. It is commonly used by folks designing parts to be printed on 3D printers. What makes OpenScad different is how you generate the design. Instead of using your mouse to drag things around on the screen, you describe your model in a simple programming language. Formally, OpenSCAD uses something called *Constructive Solid Geometry* to construct your model, then gives you a visual interface you can use to examine your 3D model in detail.

I will only show example code from the project so you can get a feel for the design process. You are encouraged to explore the project website for much better documentation and complete source code. [3].

OpenSCAD

Installing OpenSCAD is oretty simple using instructions found on the projects website. Once it is installed, open it up and take a look at the basic interface:

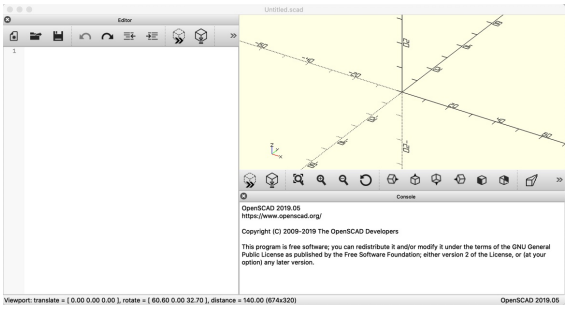


Figure 1: OpenSCAd interface

There are three areas we will be using in this view:

- Editor - the left panel where you type in your code.
- Preview - the top right panel will be where your model is displayed.
- Messages - the bottom right panel is where error messages will be shown when processing your code.

Building a model is a trial and error process. You type in or modify your code, then click on a command to process that code. You look at the preview window to see your model, and search the message area for hints about what went wrong. Non-programmers will find this a bit frustrating, but this takes practice to master, so do not get discouraged. My advice is to always take small steps. That limits the number of problems you face in getting things to work.

The best way to learn anything new is to experiment. Beginning programmers are always searching the Internet for solutions to their problems, but the only thing you actually learn when copying and pasting stuff is how to copy and paste. You will learn far more by thing in code yourself - at least until you get more proficient at this. There is nothing wrong with looking at code written by others. However studying that code will teach you how better to write your own code. I will show you enough code in this design to gve you a feel for how you do things using OpenSCAD. The actual code I generated for this design is on the project Github account (citeblackr.

Constructive Solid Geometry

OpenSCAD builds 3D models using a small set of primitive shapes, and a set of movement and combining operations to create more complex models.

Primitive Shapes

Openscad supports both 2D and 3D shapes. We will be using some simple 2D shapes, like circles and rectangles, and more complex 2D shapes like a polygon. The 3D shapes we will use include spheres, cylinders, and cubes. All of these shapes can be scaled and moved around using simple movement operations.

3D Primitives

For our first look at how you do things in OpenSCAD, here is a piece of code that will show the three basic 3D shapes:

Listing 1: scad/demo1.scad

```
cube();
translate([3,0,0])
  sphere();
translate([6,0,0])
  cylinder();
```

Figure 2 shows the result.

Each primitive shape is created at the origin. Cubes are created in the region where all three coordinates are positive. Spheres are created with the center of

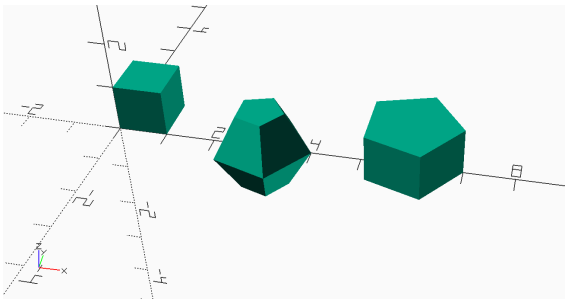


Figure 2: Demo 1

the sphere at the origin. The cylinder is centered along the **Z** axis. If you look closely, you will see a small representation of the coordinate directions at the lower left of this image.

These shapes do not look quite right. The problem is that OpenSCAD generates approximations to the rounded shaped, using a set of small polygons to build up the model. If we make these polygons smaller, things look better. All we need to do to fix this is change the code so it looks like this:

Listing 2: scad/demo2.scad

```
cube();
translate([3,0,0])
  sphere($fn=100);
translate([6,0,0])
  cylinder($fn=100);
```

Figure 3 shows a much better result.

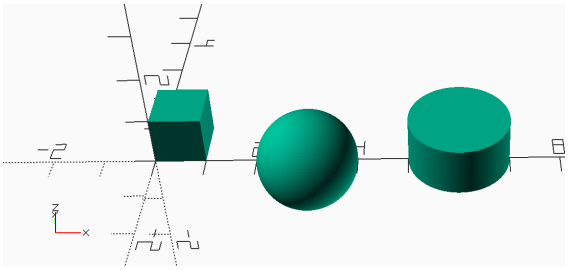


Figure 3: Demo 2

Some shapes are smart and can form different versions of themselves:

Listing 3: scad/demo3.scad

```
cube([5,1,1]);
translate([3,0,0])
  sphere(r1=1, r2 = 0.25, $fn=100);
translate([6,0,0])
  cylinder($fn=100);
```

Figure 4 shows a warped cube and cylinder. Spheres are not so smart, they stay spheres unless we warp them with external commands.

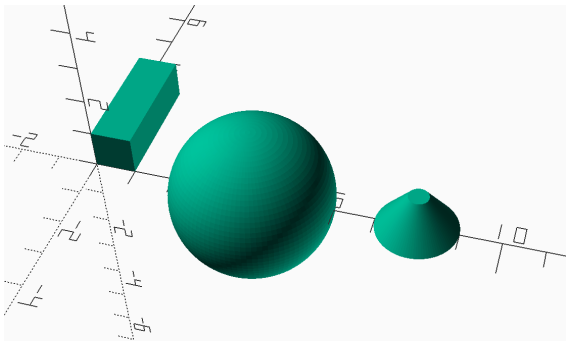


Figure 4: Demo 3

Specifying things in 3D is often done using the notation seen in the *cube()* command. The numbers are the size we want in the **[x,y,z]** directions. This is a *vector* which we will use a lot in our work.

2D primatives

2D shapes include the circle and square, which act much like their 3D counterparts. A more interesting 2D shape is the *polygon*.

Listing 4: scad/polygon-demo1.scad

```
triangle_points =[
  [0,0],[100,0],[0,100],[
  10,10],[80,10],[10,80]
];
triangle_paths =[
  [0,1,2],[3,4,5]
];
```

```
polygon(triangle_points,triangle_paths,10);
```

Here, we create *variables* and set them equal to a list of vectors. 2D vectors have only 2 numbers, for the **X** and **Y** coordinate values. The first list defines a set of six points: three for the outer triangle, and three more for the inner triangle. The second list identifies *paths* meaning a continuous line that makes up a closed circuit, one for the outer triangle, and one for the inner triangle. (I know this is confusing, but we will not need much of this kind of code in our design work.)

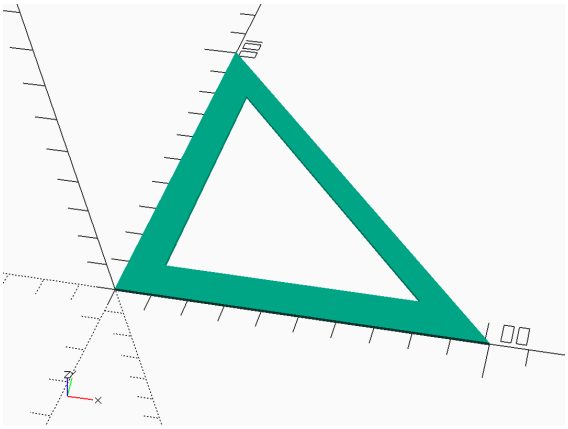


Figure 5: Polygon Demo 1

Figure 5shows a 2D shape with no thickness, although OpenSACD gives it enough of a thickness to show up on the screen.

We can use this 2D shape to create a 3D object by *extruding it* in the **Z** direction:

Listing 5: scad/polygon-demo2.scad

```
triangle_points =[
  [0,0],[100,0],[0,100],
  [10,10],[80,10],[10,80]
];
triangle_paths =[
  [0,1,2],[3,4,5]
];
linear_extrude(h=1)
    polygon(
      points = triangle_points,
      paths = triangle_paths,
      convexity=10
  );
```

Figure 6 definitely shows an interesting shape.

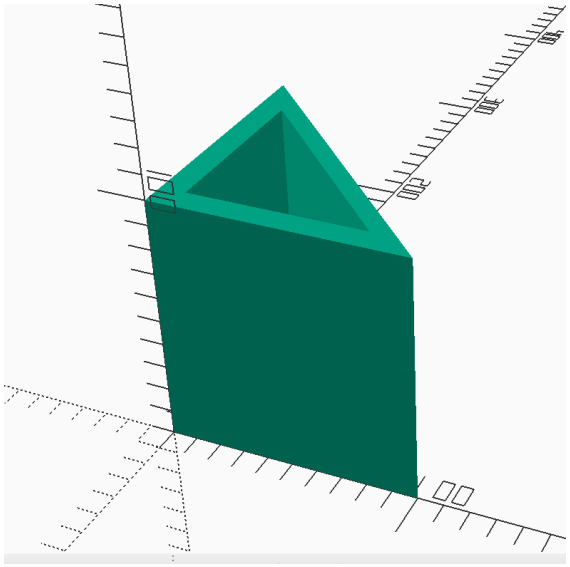


Figure 6: Polygon Demo 2

Obviously, we can form some interesting things with OpenSCAD¿ But things get even more interesting when we start combining multiple shapes to form even more complex objects.

0.1 Combining Operations

We form more complex objects by moving thngs around and combining them to form other objects. An example found in the *Wikipedia* article on CSG demonstrates these operations.

Suppose you wanted to build something that looks like Figure 7.

This shape was inspired from an example in the Wikipedia article on CSG [1].

We can form this shape using three cylinders, a sphere, and a cube. We use all three basic combining

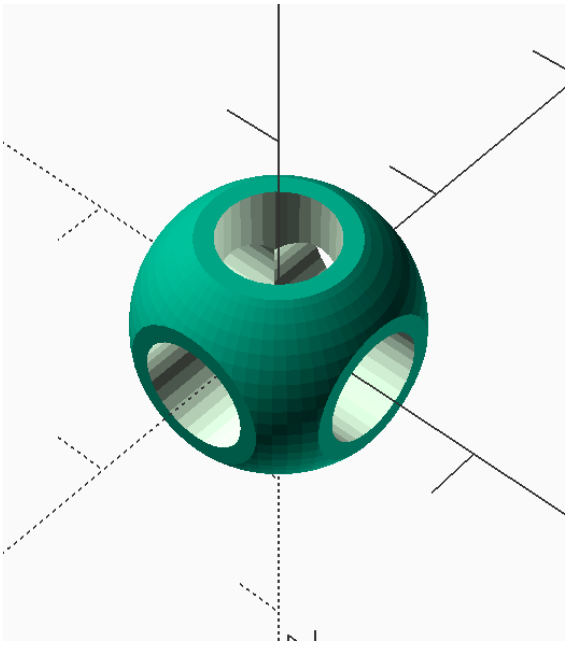


Figure 7: CSG Example Shape

operations to construct the final shape.

Here is the OpenSCAD code used to generate this shape:

Listing 6: scad/csg-demo.scad

```
module core() {
    union() {
        cylinder(
          r=0.25,
          h=2,
          center=true,
          $fn=32
        );
        rotate([90,0,0])
            cylinder(
              r=0.25,
              h=2,
              center=true,
              $fn=32
            );
        rotate([0,90,0])
            cylinder(
              r=0.25,
              h=2,
              center=true,
              $fn=32
            );
    }
}

module round_cube() {
    intersection() {
        # cube(
            [1,1,1],
            center=true
          );
        sphere(
          r=0.6,
          $fn=64,
          center=true
        );
    }
}

module part() {
    difference() {
        round_cube();
        # core();
    }
}

core();
```

There is a lot to absorb here, but taken a piece at a time, it is not so bad.

Modules

OpenSCAD lets you package a number of operations in a *module* that you can activate later, one or more times. The module can have parameters, which makes this a powerful way to manage shapes that are similar, but differ in some set of parameters. We will create a basic rib module for this model, and use parameters to control the exact rib we want.

The name we choose for each module should help you remember what the module is all about. In this

example, we are interested in the final **part** shape, which is constructed using the difference operation. This final module uses two supporting modules to build the part. Note that the code does not care about how you write your code, but it is common to use indentation to show that operations are being applied to one of more groups of operations.Also, we surround a sequence of individual operations inside of curly braces when needed.

To build this part, we first set up three cylinders, aligned along each coordinate axis. The bf center parameter, sets each cylinder up with the origin of the coordinate system at the exact center of the cylinder. That strange **$fn=32** parameter is needed ot make OpenSCAD generate cylinders that actually look round. Inside, OpenSCAD uses small straight line segments to generate a curve, and by default the size of these lines is pretty large, so we raise the number of segments we want with this number. Too many and rendering our model will be slow.

Notice that all three of these cylinders occupy the same space. In the real world, we could not do that, but in our modeling world this is common. We form the **union** of these three overlapping cylinders to form one merged shape. Many times, we could skip the **union** operation and just place the shapes where we want. This would be an important issue if we were actually going to 3D print an object, but it is not really important when all we want to do is see our design.

The outer shell of our part is made up of the **intersection** of a sphere and a cube. We size the cube shape so it trims off the six sides of the sphere where holes will end up. Finally, we use the **difference** operator to carve out the inside of the part, using our three-cylinder shape.

Successfully building 3D models involves visualizing what you want, then arranging simple shapes as needed and performing these simple combining operations to generate the gadget you want! It takes practice! The more you experiment the better you will get!

I encourage you to fire up OpenSCAD and type in this code. You will be better able to see how things work by doing this!

Building the Wing

Let's start off this design by building the wing.

Design Constraints

The class rules specify two constraints on the design of the wing. The projected span is limited to 18 inches, nd the maximum chord is limited to five inches. Based on a survey of indoor models, we will use a flat center section with wing tips angled upward, providing the needed dihedral for stability. Here is our starting geometry:

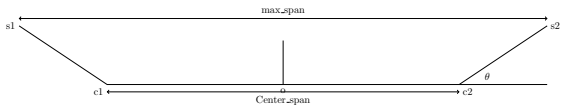


Figure 8: Basic Wing Layout

The constraint in this figure is **max_span**. We are free to choose the size of either the dihedral angle, or offset of the tip section.

Circular Arc Airfoils

Many indoor model airplanes use a simple circular arc airfoil.

Usually, these models have some specified chord for either the wing or stabilizer, and the airfoil is specified as an arc with a maximum height that is some percentage of that chord. So, we are given the chord, and thickness as a percentage value. The challenge is to figure out the radius of the arc that satisfies these values.

Arc Geometry

Since we start off with the chord and thickness specified as a percentage, we need to get rid of the percentage:

Given:

- **c** - the chord of the wing
- **T** - the camber as a percentage of **c**

$$t = Tc/100 \tag{1}$$

Here is a diagram showing what we are dealing with:

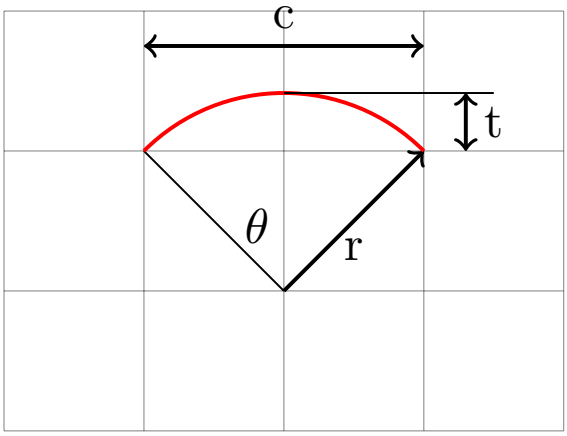


Figure 9: Circular arc Geometry

From this figure, we can write two equations:

$$r\sin(\theta) = \frac{c}{2} \tag{2}$$

$$r - r\cos(\theta) = t \tag{3}$$

The unknown variables are now:

- **r** - the radius of the circular arc
- θ - one half of the total angle swept by the arc

While we could drag out our old math books from high school, I would rather let my computer do the hard work. Time for SymPy [4]!

0.1.1 SymPy

SymPy is a Python program that knows how to do a lot of math. It manipulates *symbols*, not numbers, and that is a lot of what you should have learned in a trigonometry classes.

Here is how we get SymPy to solve our equations:

Listing 7: python/circular-arc.py

```
import sympy

r,c,t,theta = sympy.symbols('r c t theta')

eq1 = 2 * r * sympy.cos(theta) - c
eq2 = r -r * sympy.sin(theta) -t

sol = sympy.solve([eq1,eq2],[r,theta])
print(sympy.latex(sol))
```

Running this code, we find our solution:

```
r,c,t,theta = sympy.symbols('r c t theta')

eq1 = 2 * r * sympy.cos(theta) - c
eq2 = r - r * sympy.sin(theta) - t
sol = sympy.solve([eq1,eq2],[r,theta])
```

Here are the solutions:

$$\left[\left(c^2/8t + t/2,\ 2\,\mathrm{atan}\left(\frac{c-2t}{c+2t}\right)\right)\right] \tag{4}$$

Here, I asked SymPy to solve the two equations for the two *symbols* **r** and θ. The two solutions are shown. The first one is the **radius** we need to our code. We do not really need to worry about the angle part.

We will use these result to set up an OpenSCAD module that will build our ribs.

Warning! If there are any parents reading this, do not let your kids know about SymPy. On the other hand, if you cannot solve your kids math homework, you might try SymPy on the sly! YMMV!

0.2 Ribs

The ribs for this model will be simple circular arcs. We need to specify the chord, and the camber thickness as a percentage of the chord, we also need to

specify how thick the rib will be and how deep the rib will be. We will create a simple module to let the user generate as many ribs as needed, adjusting each with the needed parameters:

```
rib(
  chord=5,
  camber=6,
  thickness=1/32,
  height=1/16
);
```

When I defined the rib module, I set un default values as shown above, so to activate a standard rib, I just need to write **rib()**.

The code needed to generate a rib is a bit involved, but basically involves creating a short cylinder, cutting out the center of that cylinder leaving a tube, then slicing off the part of the ring that leaves just the rib. We then rotate it and align it for use later.

Here is an example rib, ready for use:

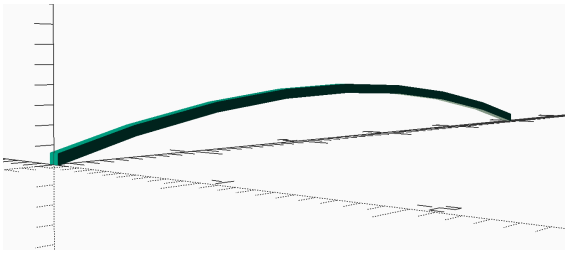


Figure 10: Basic rib

0.3 Wing Center Section

The leading and trailing edges of this model will be simple 1/16" balsa sticks. These are easy to model oin OpenSCAD as really skinny cubes. The center section of the wing holds five equally spaced ribs. This code is pretty simple, so I packaged it in a simple module:

```
wing_center_section(
  chord=5, nribs= 5, span=12
);
```

Here is this code:

This code uses a loop to place the ribs. Details on all of these language constructs can be found in the *Users Manual* **??**

Figure 11Wing Center Section.

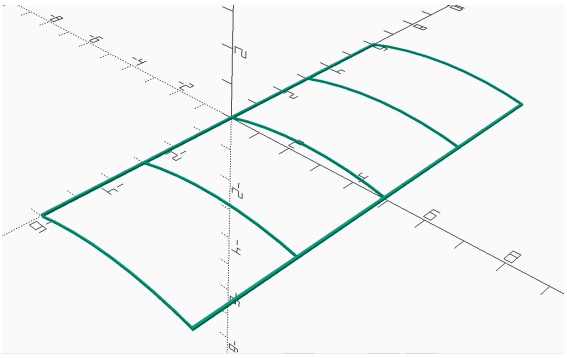


Figure 11: Wing Center Section

There is one detail in this image worth examining closer. Figure 13 shows the joint at the outer rib. This allows the tip section to mate at this joint.

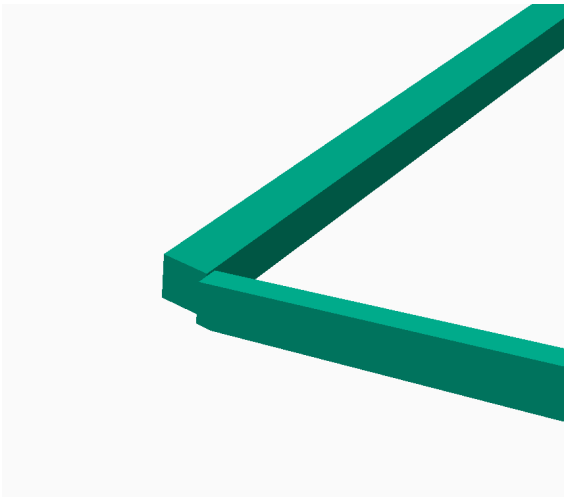


Figure 12: Tip Joint Detail

Being able to zoom in on details like this is handy for making sure your design is clean.

Tip Design

The tip is another rectangular section, except I decided to round off the leading edge corner. There are no ribs in this section, except for a flat rib at the tip. Here is the geometry I decided on:

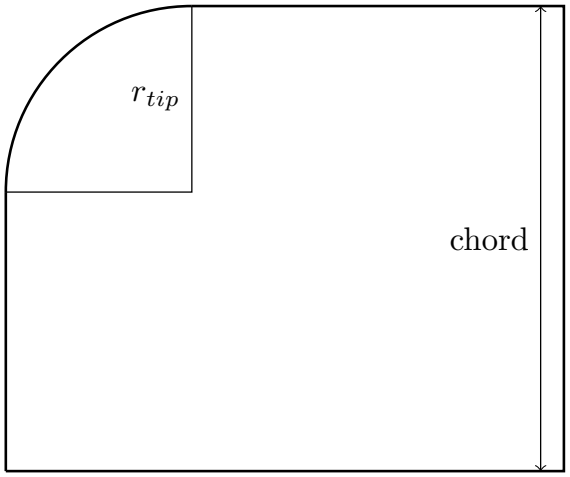


Figure 13: Tip Design

The tip module uses a simple supporting module that generates the curved section. This will be a balsa stick formed over a template. By writing the tip as a module, I will be able to reuse this design on the stabilizer and the fin!

```
tip_section(span=3, chord=5, radius=2);
```

There is one interesting issue in the tip design. Many builders taper the leading and trailing edges so the tip is lighter. In this design, I decided to taper just the leading and trailing edge tip spars and make the circular arc and the tip thinner square stock. The puzzle is how to do this in OpenSCAD.

It turns out to be easy. Once again, we use the *difference* operation. We generate a square strip the size we want, then set up a block longer than the strip and place it at a slight angle so we end up chopping off the unwanted material leaving a tapered strip. It is easy enough to do this twice to get the result we are after. This is sanding with no mess!

Figure 14 an image showing the basic idea

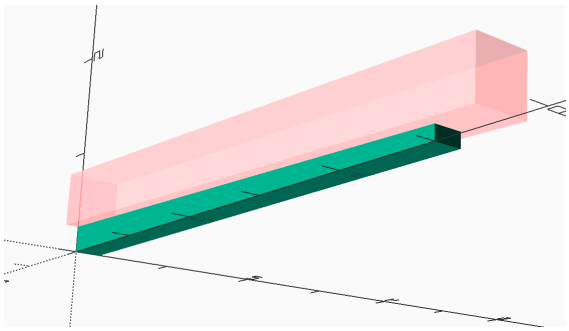


Figure 14: Trimming Block

FIgurewing-tip.pngWing Tip shows the final tip section.

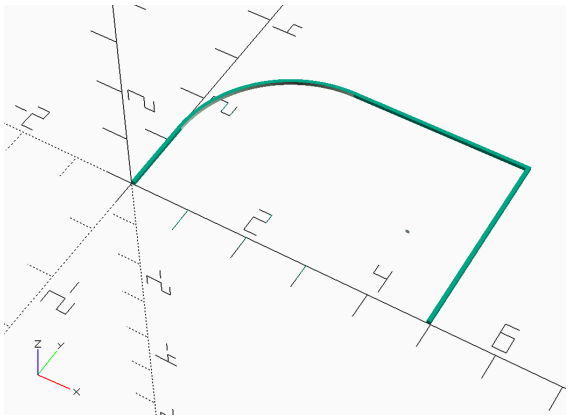


Figure 15: Wing Tip

0.4 Tip Templates

I kept this design simple. The only templates needed will be used for forming the curved segments at the tips. Fortunately, OpenSCAD can help generate printed templates we can use!

Basically, we set up code that generates a tip section, then flatten it back into a 2D drawing. The operation that does this is called a **projection**. Once that is set up, we will ask OpenSCAD to export this model as an SVG file which can be printed. I use my Chrome

browser to do the printing on my home printer. Figure **??** shows what the template looks like in Chrome.

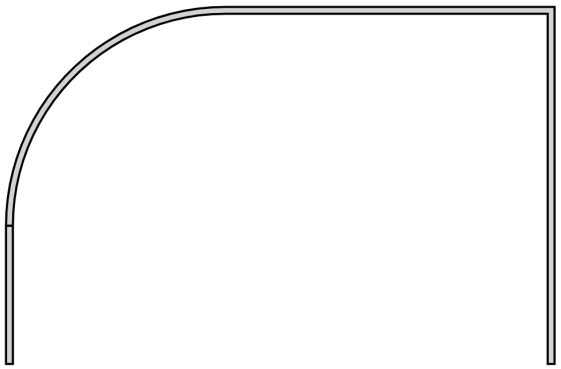


Figure 16: Tip Template

0.5 Wing Assembly

All we need to to to complete the wing is generate the center section, then generate the wing tips and rotate them into place at the proper dihedral angle. The only thing tricky here is generating the two tip sections.

Since the tip is flat, I can generate two of then, and rotate one 180 degrees so it will fit on the opposite side. I do need to do some minor translating to make sure things line up, but by now, this is getting easy!

Figurefig:wing.pngFinal Wing shows the completed wing.

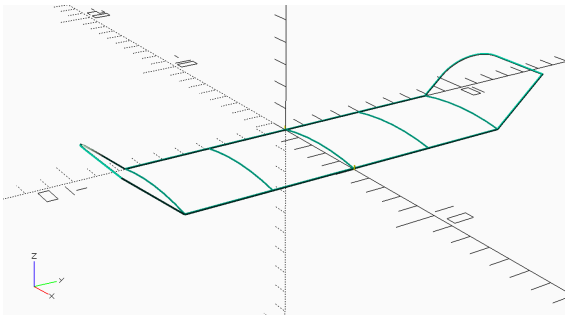


Figure 17: Wing Assembly

1 Stabilizer and Fin

The code we created to build the wing provides everything needed to build the stabilizer and fin.

1.1 stabilizer

The stab is identical to the wing, except in dimension ans number of ribs. The modules used for the wing give us everything needed to build the stabilizer.

Figure 18 shows the result.

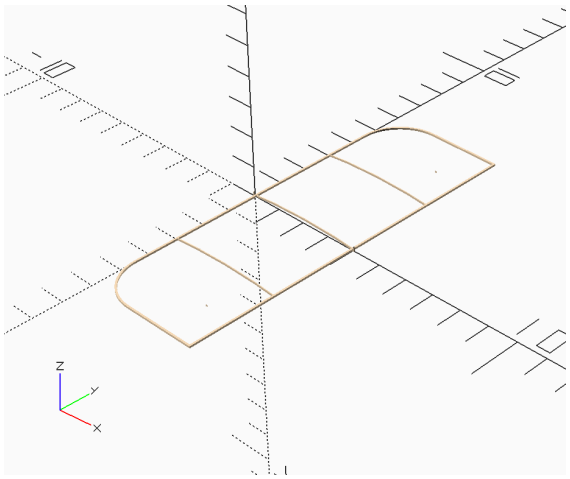


Figure 18: Stabilizer

1.2 Vertical Fin

The fin is a slightly different version of the tips. The only addition here is a square spar at the base of the fin. As we will see, we will mount the fin at an offset, and not glue it directly to the tail boom.

Figure 19 shows the fin.

Generating STL Files

Many folks who use OpenSCAD generate *STL* files from their design. These files are used as part of the 3D printing process. Although I wish we could 3D print flyable model airplanes, that is not what we will do with the STL files in this project.

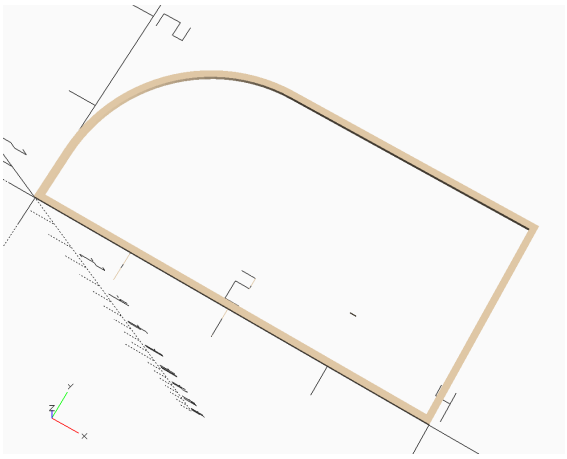


Figure 19: Vertical Fin

STL stands for *Standard Tessellation Library* [2]. Translated, that means an STL file is a list of triangular objects that describe the surface of a 3D object. These triangles cover the surface making a "water tight" approximation to the real surface. Triangles are guaranteed to be flat in the 3D space, and if they are tiny enough, they can be rendered to create a 3D display of the object, or sent to a 3D printer after suitable processing of all the triangles so the printer knows where the surface is (that is a topic for another article!)

My purpose in introducing these triangles is simple. I want to know how much my model will weigh, and I want to know where the center of gravity will be when the design is constructed. Figuring out these two details is impossible using conventional building techniques: you build the model, weight it, anf figure out the center of gravity manually. Computer geeks never do anything manually if they can get their computer to do the work.

I found a nice Python library that is all set up to figure out the volume of a 3D shape defined in an STL file, and return it's center of gravity location. It is simple enough to add in the predicted wood density for the design and come up with a weight estimate for each part. A little post-processing of all this data will give us some estimate of the total designs weight.

Unfortunately, figuring out the weight of the glue is not so simple. For that we need to know where each glue joint will be and the surface area of the glue joint. I am working on that problem now, but am not ready to show any results yet.

Knowing where each part will be placed in the final design, and the CG data for that part will let us predict the CG for the complete model as well!

Biography

In the Summer of 1955 I was delivering the evening newspaper in Falls Church, Virginia, when I rounded the corner of an apartment building and saw a man release the propeller on a rubber-powered model airplane. The plane circled in front of this man's home for several minutes, and magically landed where it had started. The airplane was a Henderson Gadfly, published in Model Airplane News that year. I was

fascinated by that sight, and decided to figure out how the airplane managed to do that. I talked the man into giving me the plans he used to build the model, traced from the magazine. I still has those plans to this day!) Soon, a couple of my friends and I decided to start building model airplanes of our own. We all took a bus to downtown Washington, D.C (kids could do that back then), and joined the Academy of Model Aeronautics. We also joined the Fairfax Model Associates and began competing in a variety of events, mostly control line and gas free flight. At one meeting, Bill Bigge, an internationally known indoor model builder, was the guest speaker. I got my first look at a new form of model airplane. The indoor models Bill brought to the meeting were fascinating, and cheap enough even a kid with a limited allowance could build one. Bill became my mentor, and I managed to build an ornithopter and helicopter and set two national records! I spent 20 years as an officer in the USAF, then another 17 years teaching college-level Computer Science. I finally retired for good in 2018, and moved with my wife to Kansas City, where I joined the Heart of America Free Flight Association, and again began flying model airplanes, this time focusing on rubber and electric powered outdoor free flight, and indoor events.When not building model airplanes, I am active in Amateur Radio and am currently authoring a book on Computer Architecture.

References

[1] Constructive solid geometry - Wikipedia, 2021. URL `https://en.wikipedia.org/wiki/Constructive{_}solid{_}geometry`.

[2] STL (file format) - Wikipedia, 2021. URL `https://en.wikipedia.org/wiki/STL{_}(file{_}format)`.

[3] R. Black. Math Magik LPP, 2021. URL `https://rblack42.github.io/math-magik-lpp/`.

[4] S. Developers. SymPy, 2021. URL `https://www.sympy.org/en/index.html`.

[5] M. Kintel. OpenSCAD - The Programmers Solid 3D CAD Modeller, 2021. URL `https://www.openscad.org/`.