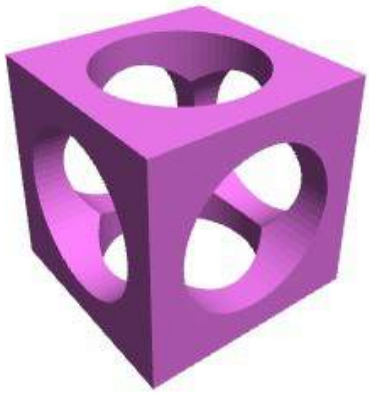
From Dreams to Designs:
The Polar3D Introduction to BlocksCAD

David D. Thornburg, PhD
Dthornburg@polar3d.com



The images in this document are produced by the author.

The author can be reached at: Dthornburg@polar3d.com

The programmer, like the poet, works only slightly removed from pure thought stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures.

— Frederick P. Brooks Jr., Mythical Man Month, 1975.

Table of Contents

Introduction 5
Chapter 1 — Looking Around BlocksCAD 9
Chapter 2 — Shapes 12
Chapter 3 — Transforms 21
Chapter 4 — Set Operations 43
Chapter 5 — Math 55
Chapter 6 — Logic 58
Chapter 7 — Variables 62
Chapter 8 — Loops 65
Chapter 9 — Text 71
Chapter 10 — Modules 77
Chapter 11 — Combining Shapes into Complete Objects 87
Chapter 12 — Dreaming and Doing on Your Own 103
Acknowledgements 107
About the Author 108

Introduction

BlocksCAD is a free cloud-based programming language that is used to design and render 3D objects that can be printed on a 3D printer. Unlike traditional programming environments, BlocksCAD is easily accessible to school-age children, and is also capable of creating complex objects using advanced programming concepts like recursion.

This book started out as a BlocksCAD manual, but was expanded to provide projects of varying complexity to help cement the developing mastery of this language in anyone who wants to explore it.

Teaching students to program is now a hot topic — once again. In the early days of personal computing, there was little to no commercial software available. If you wanted to run a program, you had to write it first. The debates in the 70's were not about teaching programming, but about which language to use. The choices at that time boiled down to BASIC and Logo for most teachers, with Logo having an elegance that made it (in my view) the logical choice. But then commercial software came into the marketplace, with shrink wrapped applications ranging from word processors to animation tools, driving the need for programming out of the classroom.

Recently, we have come to realize the price that was paid by making this choice. We have a generation of students who know how to use computers, but know nothing about how to make them do new things — to run programs they have written by themselves. Aside from the financial benefits that come from knowing how to program, the satisfaction and general problem solving skills associated with programming are denied to those who have never created even the simplest of programs on their own.

For example, Burning Glass (http://burning-glass.com/research/coding-skills/) wrote a report on the topic, *Beyond Point and Click: The Expanding Demand for Coding Skills* that shows:

1. **Coding skills are in high demand**
   Seven million job openings in 2015 were in occupations that value coding skills. This corresponds to 20% of "career track" jobs, defined as those which pay a national living wage of at least $15/hour.

2. **Coding skills are of value to candidates across major job categories**
   • Information Technology (IT) workers
   • Data Analysts
   • Artists and Designers
   • Engineers
   • Scientists

3. **Coding Jobs Pay More**
   Jobs requiring coding skills pay $22,000 per year more than jobs that don't: $84,000 vs $62,000 per year. (This analysis includes only "career track" jobs.)

4. **Coding skills Provide an Avenue to High-Income Jobs**
   Half of jobs in the top income quartile (>$57,000 per year) are in occupations that commonly require coding skills from job applicants.

5. **Coding Jobs are Growing Faster than the Job Market**
   Programming jobs are growing fastest, 50% faster than the market overall. In general, programming jobs are growing 12% faster than the market average.

The data speaks for itself. Tools like BlocksCAD that combine the development of coding skills with the

design of 3D printable objects has great value in education.

Why I wrote this book

I've written numerous books on Logo — a powerful language that (among many other things) lets you create graphical objects of great complexity. But whether simple or complex, these objects all have two things in common: they reside on the computer screen (or as a printed image), and they are two dimensional, even when they are used to represent three dimensional objects.

When I wrote those books, there were no 3D printers — computer peripherals that let you build tangible physical objects from plastic and other materials. That has all changed. Good 3D printers can be purchased today for less than what I spent on my first laser printer, and the performance continues to advance. The printers made by Polar3D, for example, make plastic parts from the extrusion of filament onto a glass build plate. These cloud-based printers have a small form factor perfect for classroom use, yet allow the building of larger objects than one might imagine, since they operate using polar rather than cartesian coordinates. Because each printer has a built-in camera, print progress can be monitored anywhere in the world one has Internet access. From a pragmatic perspective, a school might have several printers in a central location, each of which can be monitored from anywhere.

Traditionally, objects made on these printers are designed in a graphic modeling program like Tinkercad (www.tinkercad.com). The designer creates a design on the screen by drawing it, and the resulting three-dimensional drawing is then sent to the printer for printing. There are a great many things that can be designed this way with ease, and our previous book on the topic of 3D printing in the classroom explores some of these (http://amzn.to/1pyeaqk).

In fact, there are three general ways to develop 3D models. First, you can manipulate 3D building blocks on the screen with tools like Tinkercad. You can also "sculpt" a model using virtual clay that can be manipulated to form very organic shapes using programs like Sculptris or Meshmixer. The third way is to build your model through a set of instructions written in a programming tool like OpenSCAD (www.openscad.org). This third approach is perfect for those models that are very simple, or very difficult, if not impossible, to build using tools in the other two classes, or to build models that rely on variable parameters to change size. For example, a model of a napkin ring can be written so it works with different sizes, even down to the size of a ring on your finger, This versatility is very useful. And that is where this book comes in.

In the hands of people willing to learn OpenSCAD, amazing things are possible. However, this language has a steep learning curve, especially for younger users and programming neophytes in general.

The problem with OpenSCAD

While everything in this book can be written using OpenSCAD, our focus is on BlocksCAD, a language derived from OpenSCAD, because OpenSCAD is tricky to master.

For example, here is the OpenSCAD code for a simple (but beautiful) geometric shape:

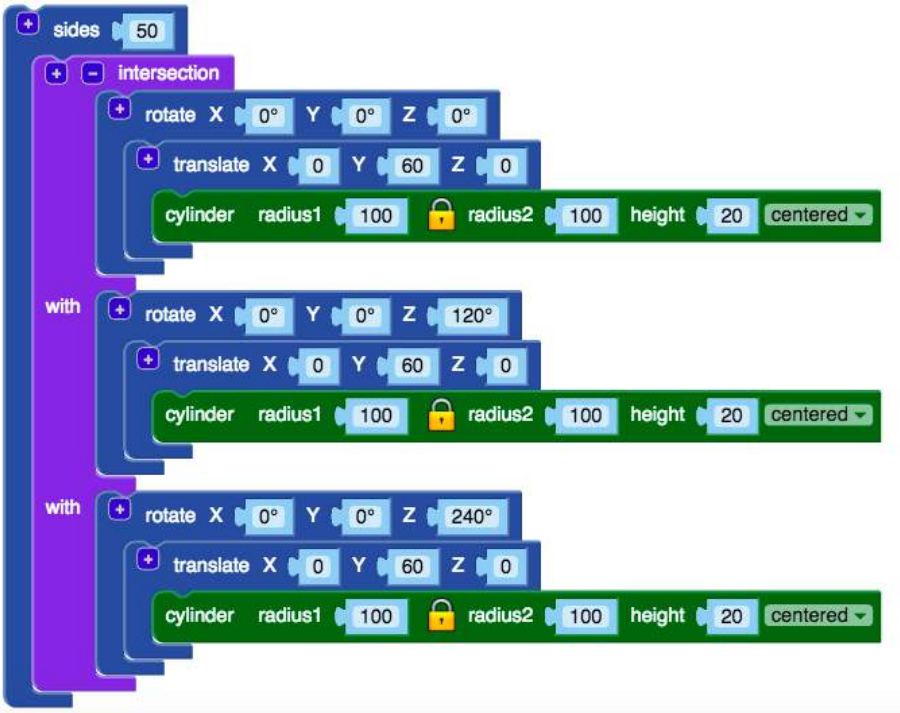
```
//Reuleaux Triangle

{
 $fn=50;  //set sides to 50
 intersection(){
  rotate([0, 0, 0]){
   translate([0, 60, 0]){
    cylinder(r1=100, r2=100, h=20, center=true);
   }
  }
  rotate([0, 0, 120]){
   translate([0, 60, 0]){
```

```
    cylinder(r1=100, r2=100, h=20, center=true);
   }
  }
  rotate([0, 0, 240]){
   translate([0, 60, 0]){
    cylinder(r1=100, r2=100, h=20, center=true);
   }
  }
 }
}
```

Notice all the curly brackets. If one of these is left out, the program will not compile, and you won't get your final shape. Also, you'll see that some of the lines end with semicolons — essential punctuation marks that are easy to forget.

On the other hand, BlocksCAD, which generates OpenSCAD code, is far easier to learn. This book focuses on BlocksCAD for this reason.

The BlocksCAD code for the same shape is shown below:



This program is easy to read (and create) and all the needed punctuation marks are put in the right places automatically in the equivalent OpenSCAD code.

After designing a part with BlocksCAD, you can save it as an STL file for your printer, send it to your printer on the Polar Cloud (www.polar3d.com) and you can also usually export the OpenSCAD code if you want. This can be helpful if you are placing a parametric model on Thingiverse or another sharing site since it lets others tailor your design to meet their needs. We'll have more to say about that later.

The main point is that BlocksCAD lets you create amazing projects without the burden of having to master OpenSCAD.

The book proceeds from the simple to the complex, with each new feature explored in depth so you can see how it might be useful in your own designs. On our journey you will want to 3D print some of

the designs you are making, although they can be seen (and rotated) inside BlocksCAD or with another program I like, MeshLab (www.meshlab.net). While screen images are nice to have, they pale in comparison to holding an object you have made on a 3D printer.

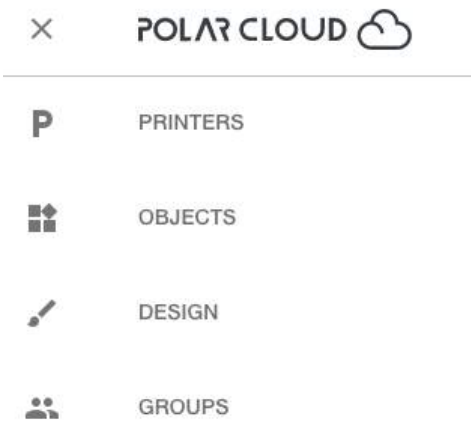
Your projects can range from the practical to the artistic — and everything in between.

Chapter 1 — Looking Around BlocksCAD

The first thing to do is launch BlocksCAD by going to the Polar Cloud (www.polar3d.com). In the upper left, next to the words Polar Cloud, you'll see a menu item with three horizontal lines.

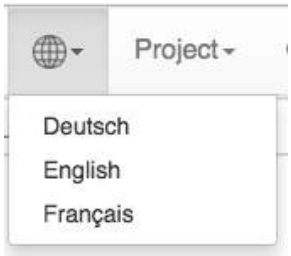
When you click on this button, it drops down a menu with several choices.

Select DESIGN which brings up two options, BlocksCAD and Tinkercad. We'll be using BlocksCAD.



When you launch BlocksCAD, register for your free account. BlocksCAD runs in the cloud, making it perfect for use with any device with Internet access from computers to tablets and Chromebooks. You should note that BlocksCAD works best with Chrome and Firefox browsers.

The menu bar to the right of the BlocksCAD logo contains some important items. The first item looks like a globe and lets you choose the language for your project:



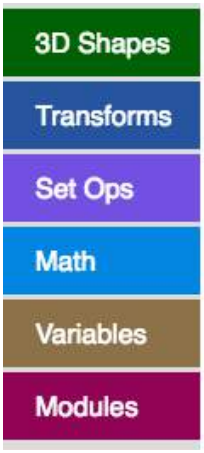
For this book, we'll do everything in English, but its is nice to see other options.

The **Project** menu has several options:

My Projects — Displays a list of saved projects
New — Creates a new project
Save — Saves the current project
Save As A Copy — Saves the current project to a new project
Save Blocks to your Computer — Exports the current project to your computer as an XML file
Load Blocks from your Computer — Loads a BlocksCAD XML file
Import Blocks into Current Project — Imports a BlocksCAD XML file into the current project

The next menu item, Options, lets you change the color scheme of the blocks and, more importantly,

work with a simpler set of blocks suitable for younger users who are just getting started!



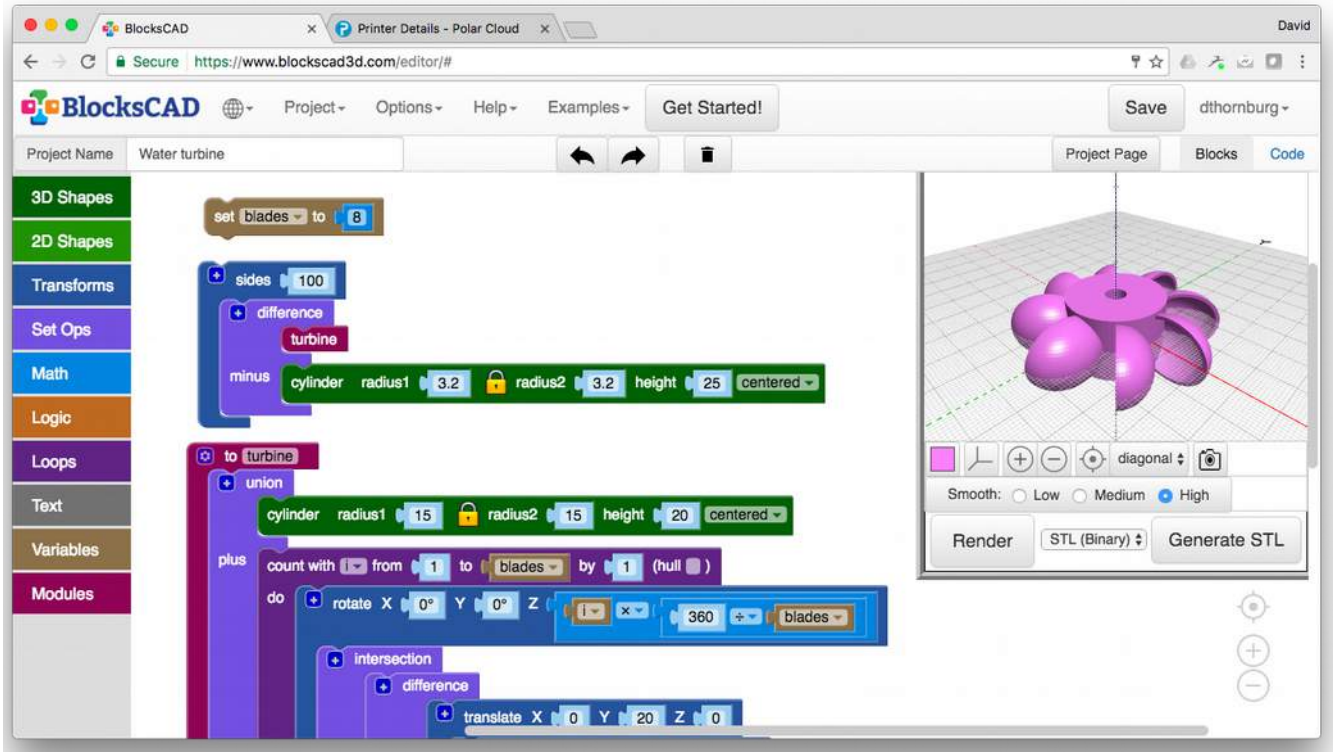
We will work with the entire set of block types in this book, but it is nice to know that those starting out have the option to work with just a basic set of block types if they want.

The remaining menu choices provide help and show examples of projects. The Get Started button shows a short video on how to use BlocksCAD.

If you've logged in to your BlocksCAD account, the Save button will save your model using the project name you can enter in the text window in the next line down from the Menu bar. If you haven't logged in, you can always save your files on your computer's drive instead of in the cloud. The option is yours! I typically save my files on my computer as well as in the cloud — kind of like wearing both a belt and suspenders.

This line with your project name also lets you back up or advance your model based on your edits, and lets you delete selected items from your model. The right side of this line provides two buttons. The first of these gives the default view of your model as a collection of programming blocks. The second shows what the model looks like in the OpenSCAD language. The OpenSCAD view is read-only. All edits need to be done in the Blocks view (which is the point of BlocksCAD in the first place!).

Directly below this line is the main part of the BlocksCAD interface. The image below shows part of a water turbine model, just to highlight the three major areas of the screen.



There are ten color-coded buttons representing block categories on the left side of the screen. Each of these categories brings up the different kinds of objects used in a BlocksCAD program. These include two kinds of shapes (both 3D and 2D), various transformations, setting different operations on our shapes, math operations, logic operations, loops, text, variables, and the definition and use of modules you can build to collapse large program segments you use over again into a single command.

The middle of the window shows the BlocksCAD code for your project, and the graphics window on the right provides a picture of your model that can be expanded and rotated to confirm that it accurately reflects your design. This window also contains a Render button needed to compile and display your design, along with some image quality settings, and the ability to generate a 3D graphics file for your printer. This is likely to be an STL file, although other options are available.

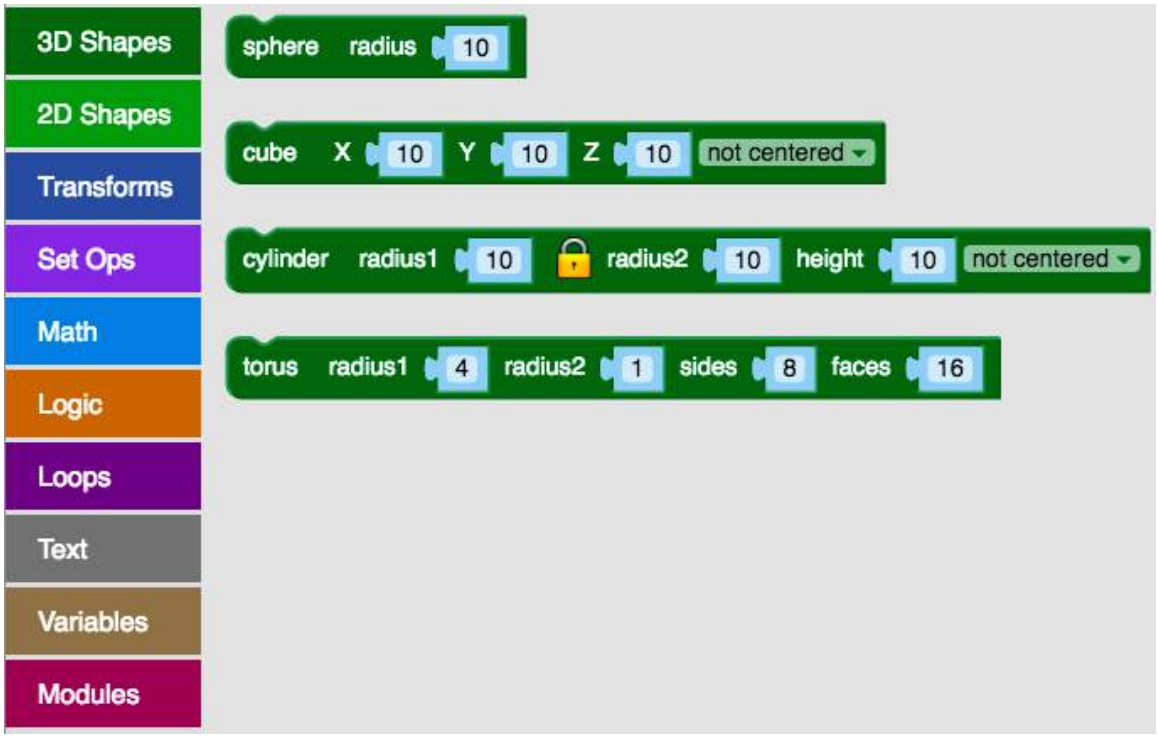
Your shapes can be saved in the BlocksCAD cloud, as well as on your computer's hard drive. BlocksCAD files are saved in the XML format. They can also be exported as OpenSCAD files. Unfortunately there is no current way to import OpenSCAD files into BlocksCAD, but this may be possible in the future.

As with any creative tool, be sure to save your work frequently.

The next chapters take you through the various commands that make up a BlocksCAD program. As we make progress, you'll get involved with the design of some things you will want to print!

Chapter 2 — Shapes

BlocksCAD lets you build and work with all kinds of shapes — even those you import from your own STL files! The most common shapes are three dimensional, so we'll start there. A list of these shapes appears when you click on the 3D Shapes button. To add a shape to your project, click on its name and it will show up in your main window.



You can build quite complex models in BlocksCAD using these four basic shapes: Spheres, Cubes (which can be boxes), Cylinders (which can be cones) and Toruses. We'll also show how to design and import other basic shapes (like gears) you can make with other tools like Inkscape (www.inkscape.org).

In this chapter we will explore these shapes separately, and then we'll learn how to assemble them into complex objects in the next chapter.

The dimensions of all your shapes are in millimeters. Most 3D printers can handle shapes up to 100 mm in each dimension, and some (like the Polar3D) can work with larger shapes than that!

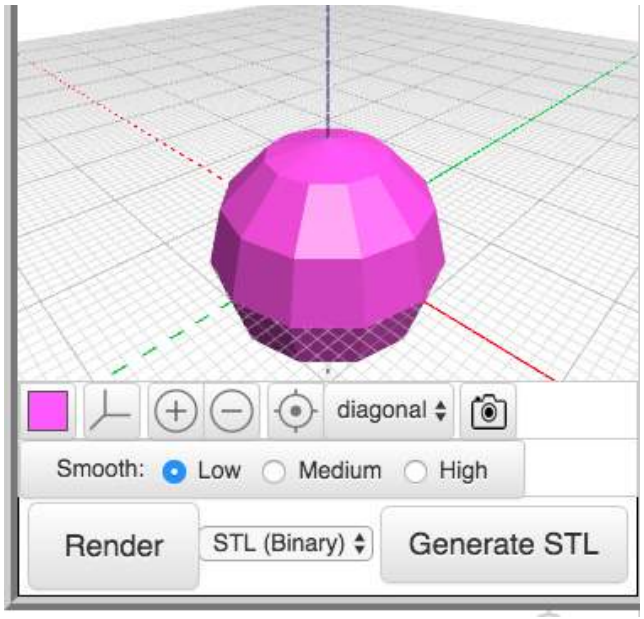
Spheres

Spheres are handy shapes with which to build things, and the BlocksCAD command for drawing spheres looks like this

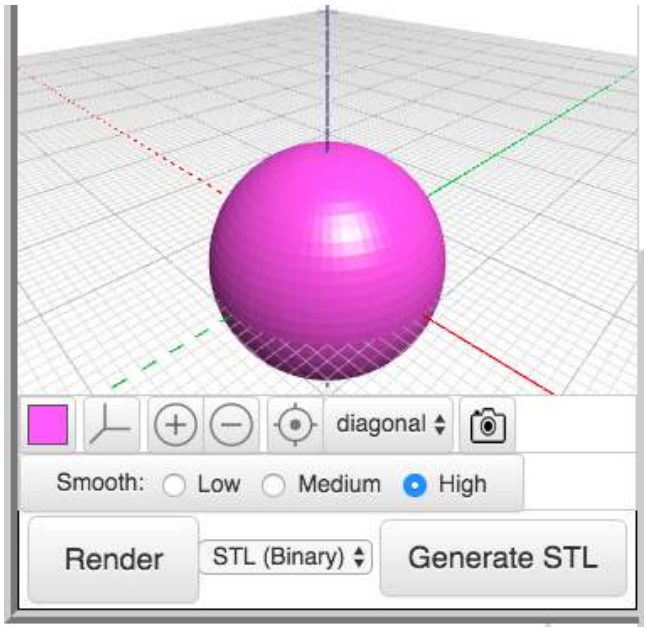
The default value for the sphere is 10 mm and the sphere is centered at the origin of the graphics coordinates. To change the sphere size, replace the 10 with the number you wish — for example, 31.456. The value is up to you and it can be any decimal number in mm that you want.

If you render the original sphere with a smoothness value of "low" and magnify it a bit, you get the

following shape after rendering it:



This is not a sphere at all, but we can improve things by changing the smoothness to “high” and rendering it again:



That looks much better.

The smoothness option is only needed when you have curved shapes in your design, but you should probably use it all the time. The only downside is that the high setting results in somewhat larger STL files for your printer, and the image might take a bit longer to render. We'll show another way to achieve this smooth curved shapes in a later chapter.

Before moving on to Cubes, I want to mention that you'll learn how to transform your sphere into other rounded shapes — egg shapes, for example. This gets covered in another chapter.

Cubes

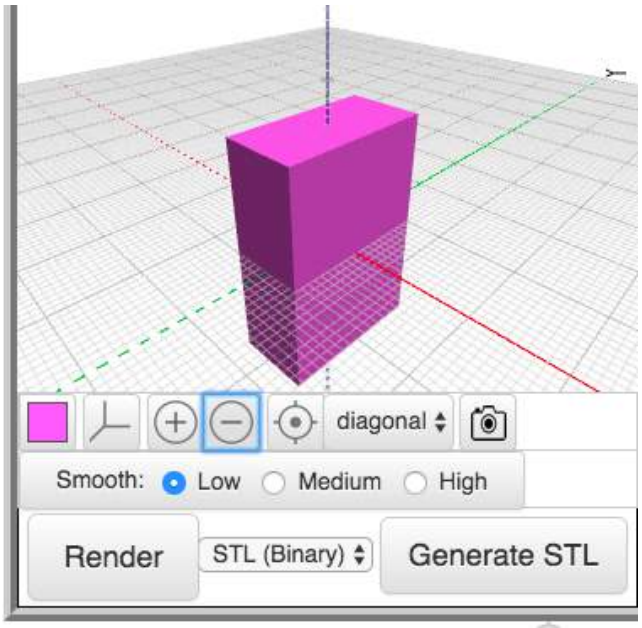
Next we'll explore the somewhat misnamed Cube. This shape is used to make boxes with any dimensions. Technically, these shapes are rectangular prisms. The “cube” name is easier to type if you are using OpenSCAD, so it was kept in BlocksCAD.

Start by deleting the sphere from your screen and then choosing the Cube command from the 3D Shapes button.

The default is a cube whose lower left corner is at the origin with dimensions of 10, 10, and 10 mm where x, y and z are the respective values for the size of the cube in millimeters.

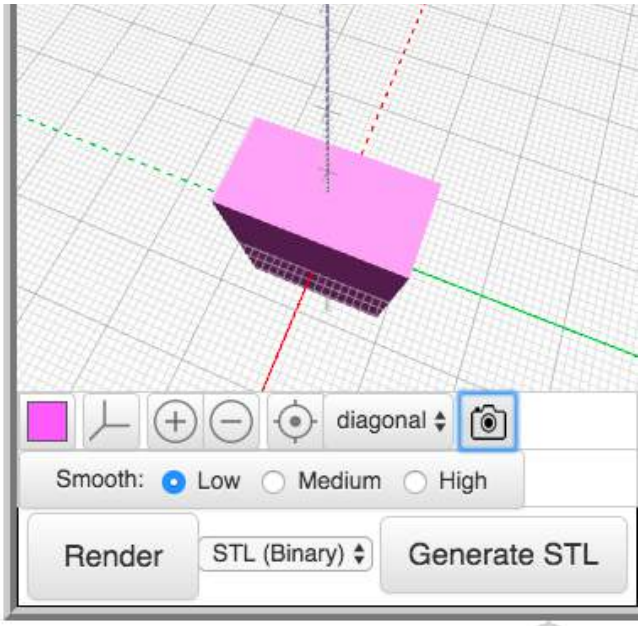
We can change the cube to a different box shape centered on the origin by changing our numbers and changing “not centered” to “centered” by clicking on the tab to the right of the words “not centered.”

This (when rendered) gives this image:



This is obviously not a cube anymore, but that is the name given to these shapes in OpenSCAD.

If you want, you can click the mouse anywhere in the graphics window and rotate your view of the shape to see it from a different angle.



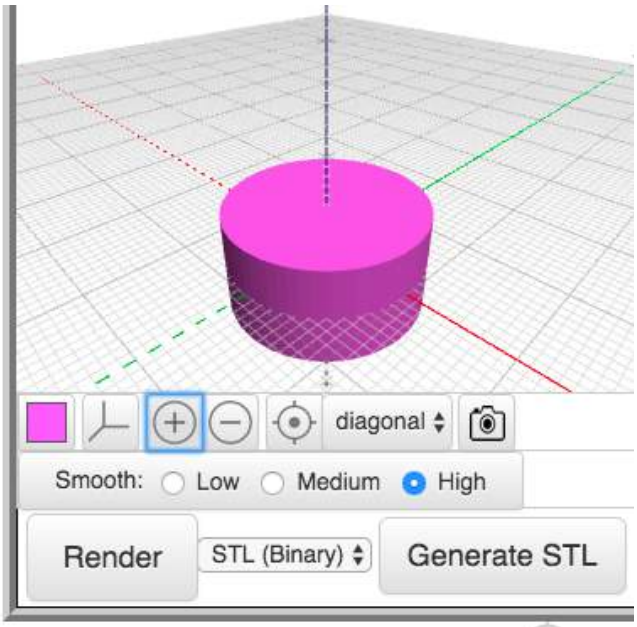
Note that you've only changed the view of the shape, not its actual orientation. You'll learn how to rotate shapes later in this book

Cylinders

Our next shape in this chapter is the cylinder. As with the cube, this name is not exactly correct since, in addition to cylinders, it draws truncated cones and cones. Now, since a cylinder is just a cone with the same radius for the top and bottom, why not call it a cone? Beats me.

The basic command looks like this:

And produces the following cylinder.

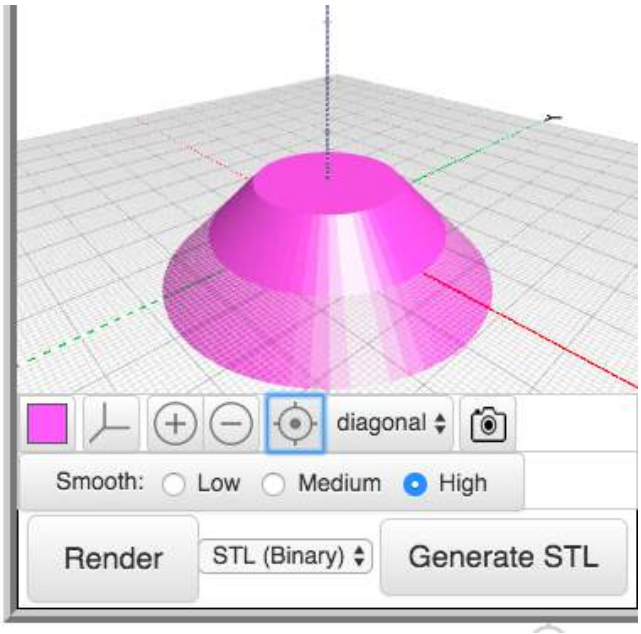


There are two things to notice. First, I set the render position to “centered.”

I almost always do that so I know exactly where my parts are. When I move them later, I'm always moving the center position. This way, if I change the size of a shape, the center always stays in the same place.

Second, notice the lock before the second radius. When this lock is on, the number you put in radius1 automatically is repeated for radius2, giving us a cylinder. To generate a conical shape, click on the lock to unlock it. Now you can have a different radius for the bottom than you have for the top. For example, this command gives us a cone.

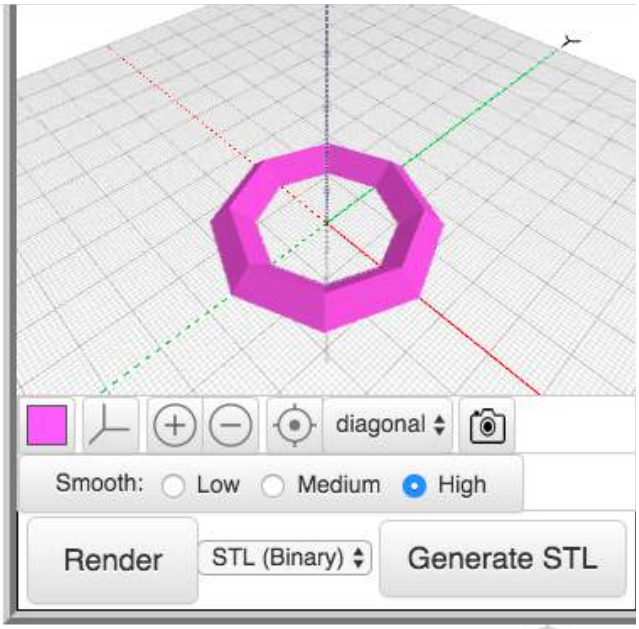
The cone looks like this:



Torus

The remaining built-in shape is the torus. This is used to make tubular shapes that are bent so the ends connect like a ring. The torus has several parameters:

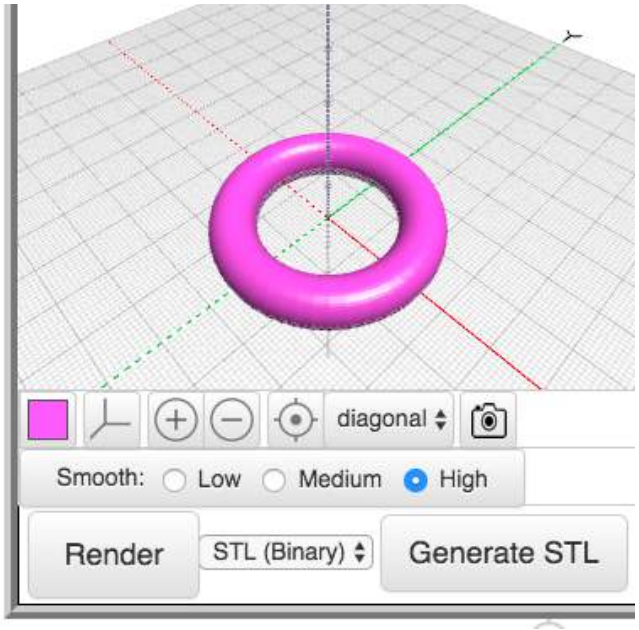
Before going into the details, notice there is no centering command. Like the sphere, the torus shape is always centered at the origin.



Now let's look at the parameters for this shape. Radius1 is the radius of the torus itself — the distance from the origin to the center of the shape along the X-axis. Radius2 is the radius of the tube itself. Sides represent the number of sides in our ring (in our case, 8). Faces represent the number of faces in the tube itself — in our case, 4.

To make a traditional circular torus, change the number of sides and faces to a larger number — for example, 50.

This gives the following shape.

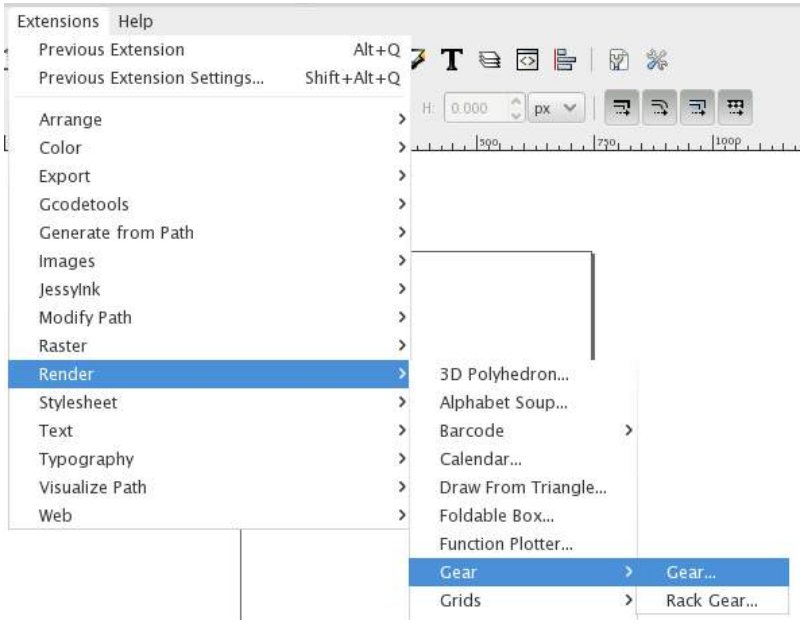


Importing other shapes

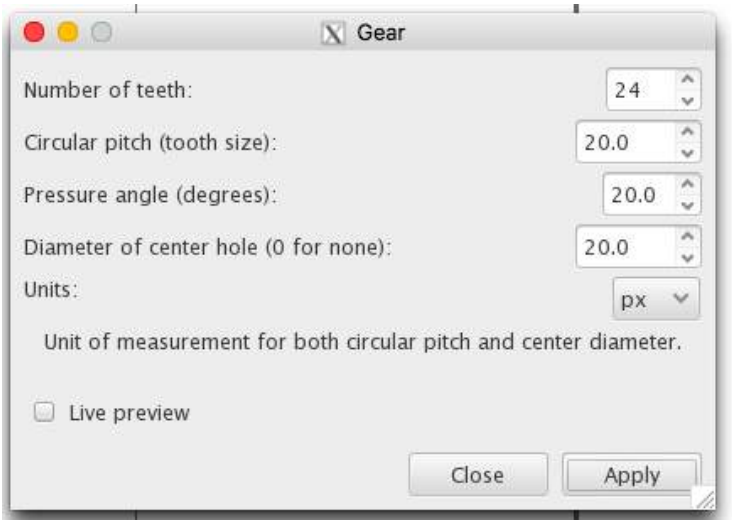
As powerful as these four shapes are, there are lots of shapes you'll want to use that can be made more easily with other tools — gears, for example.

The tool I use most is a program called Inkscape (www.inkscape.org) which lets you design and export images as two-dimensional SVG (scalable vector graphic) files. For example, gears are easy to make. (Inkscape is worth learning by itself, and a full tutorial goes beyond the scope of this book, so we'll just demonstrate one aspect of this powerful tool). Note that it is important, when installing Inkscape on a Macintosh, to follow the directions exactly!

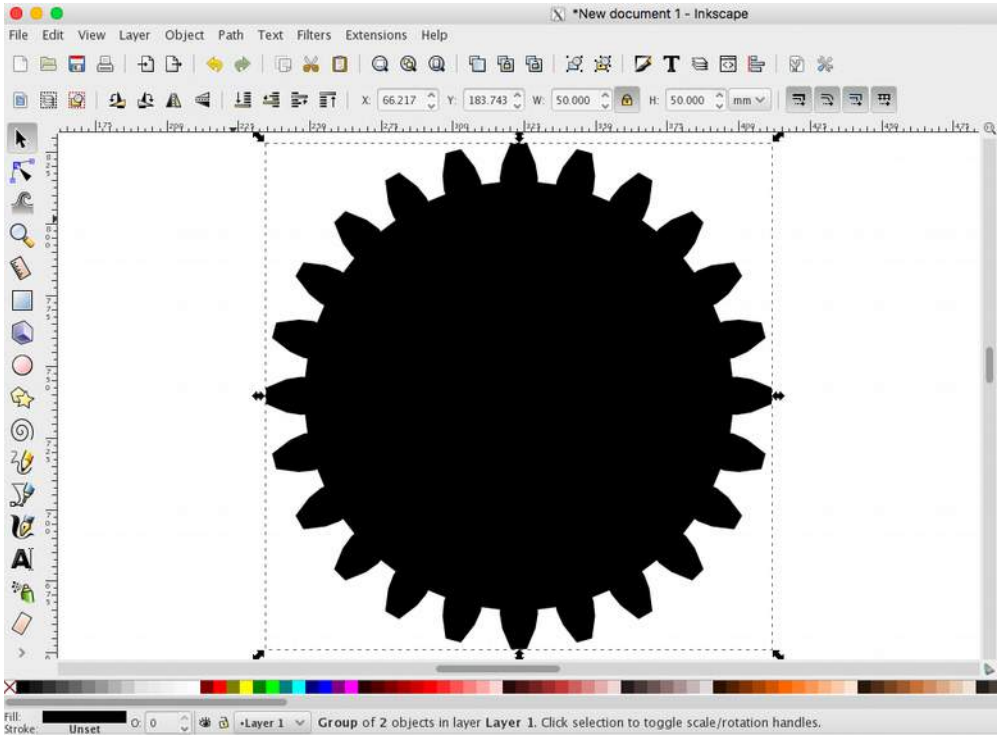
After installing, launch Inkscape and under the Extensions menu choose Gear and Gear...



This brings up the following box containing the details of your gear.



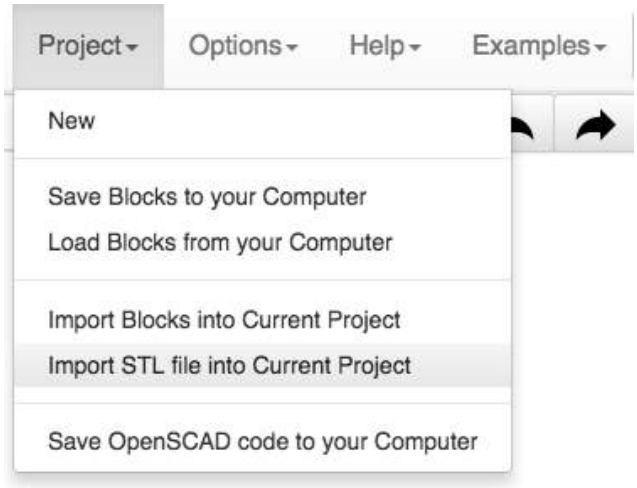
I chose a gear with 24 teeth, and left the other parameters the same. When you apply these settings, the outline of a gear appears. I then filled the gear with black, and set the size to a diameter of 50 mm.



After saving the gear as an SVG file, I next used an online tool — svg2stl.com — where I uploaded the SVG file and set the extrusion depth to 5 mm. You can choose other values if you want, but this is good for now. Also, BlocksCAD lets you scale along any axis later if you wish. (We'll show how to do this in the next chapter.)

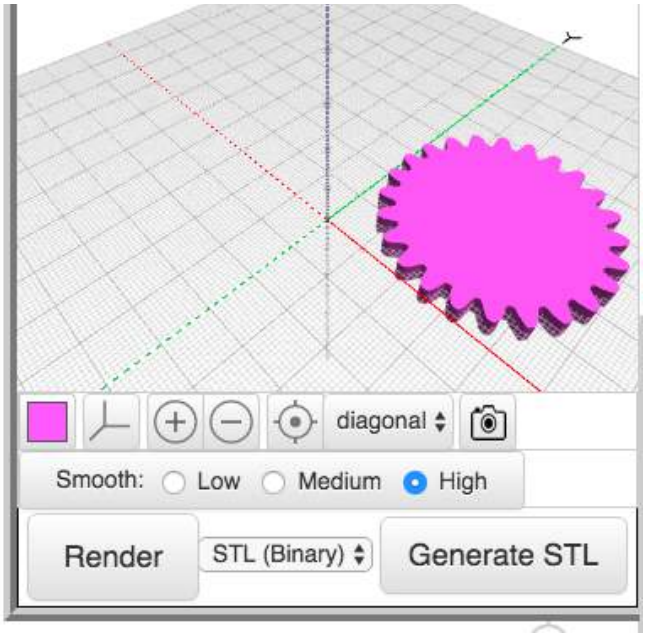
Once you download the converted file, it is given a new name. In our case, it is called gear.svg_5mm.stl.

Now it is time to go back to BlocksCAD where, under the Project menu you'll see the command to import an STL file into your current project.



Choose the new STL file you just made, and the following 3D shape block will appear on the screen.

When you render the gear, you'll see that it is not centered on the screen. We'll show how to fix this later and how to put a mounting hole in the center of your gear.

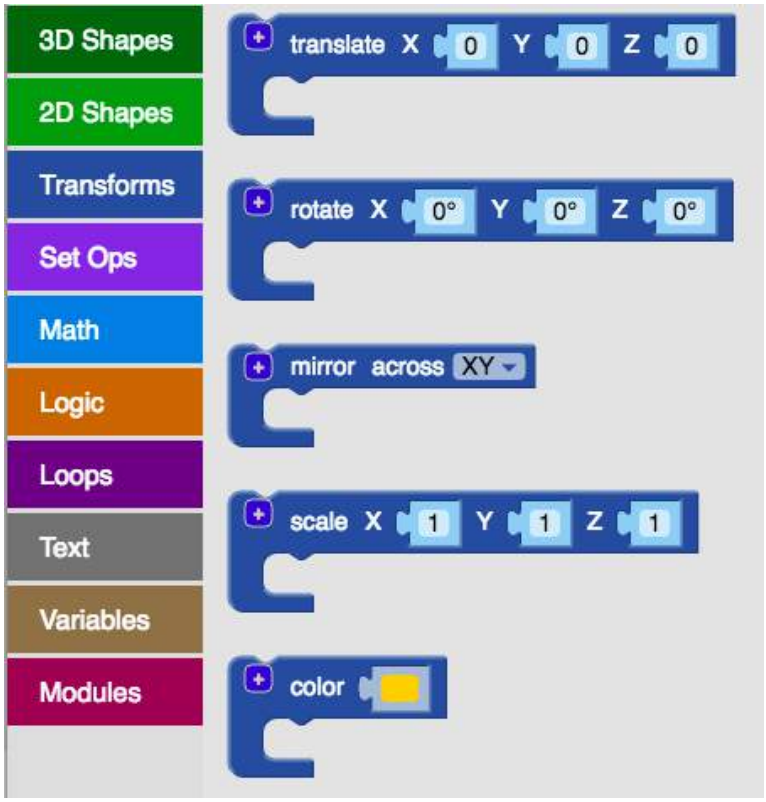


At this point, you have 3D shapes pretty much under control. BlocksCAD also lets you create a few 2D shapes that can be extruded into 3D models, but these are rarely if ever needed, so we'll address them when we need to.

Now it is time to learn how to transform the shapes you've already made.

Chapter 3 — Transforms

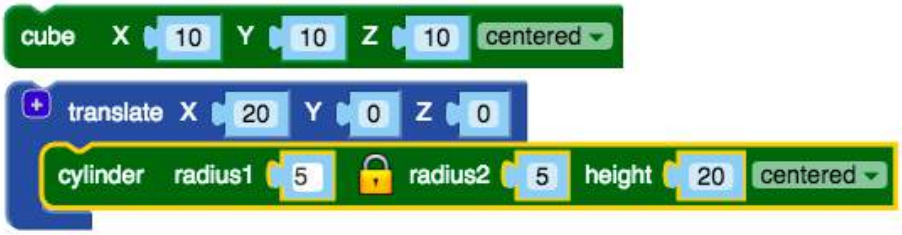
BlocksCAD supports twelve types of transformations of your shapes. Some of these are going to get used a lot, and others may never get used. Just the same, you should know about all of them. In fact, when combined with what you learned in the previous chapter, you'll be able to make some cool designs!



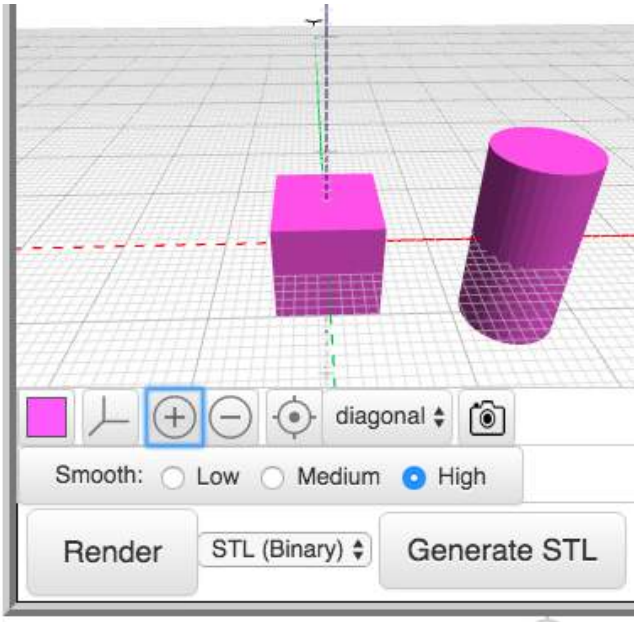
Translate
This transformation moves an object to any coordinate you want, no matter how it was drawn in the first place. If you used “centered” when you drew the object, then the transformation will move the center to another location. Otherwise, it will move the object relative to the origin of the axes. (This is why I almost always use “centered” when I draw something.) The command has the basic form:

The object to be moved is placed inside the Translate block and the translation distance along the X-, Y- and Z-axes is (as always) in millimeters. The numbers you use can be either positive or negative. Negative numbers will translate an object in the opposite direction.

To show this in action, I'll put a cube at the origin and a cylinder 20 mm away along the X-axis:



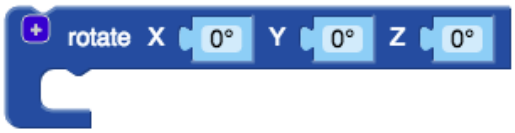
This gives us the following two shapes.



Rotate

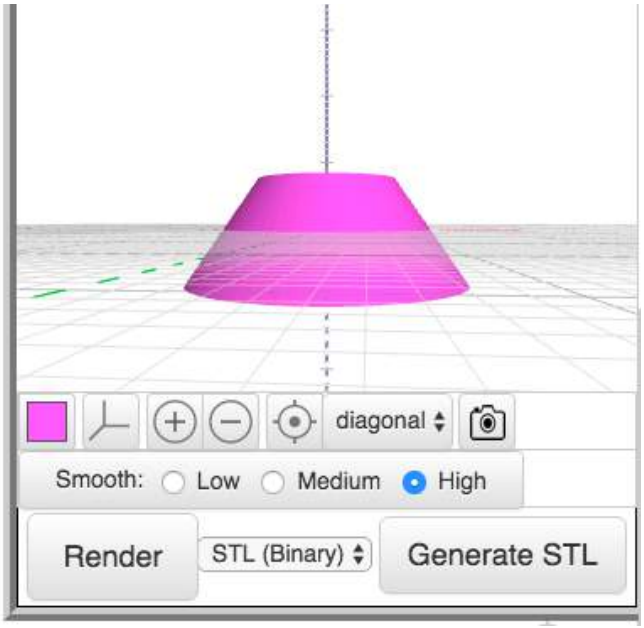
This transformation can be a bit tricky to use. Basically, it rotates your object by a specified amount (in degrees) around the X- Y-, and Z-axes. It does not do this all at once, however. First it rotates around the X-axis, then the new position is rotated around the Y-axis, and finally, the resulting position is rotated around the Z-axis. If you forget this, you may find yourself with a stranger overall rotation than the one you desired. But, fear not, this is fixable.

The basic form is:



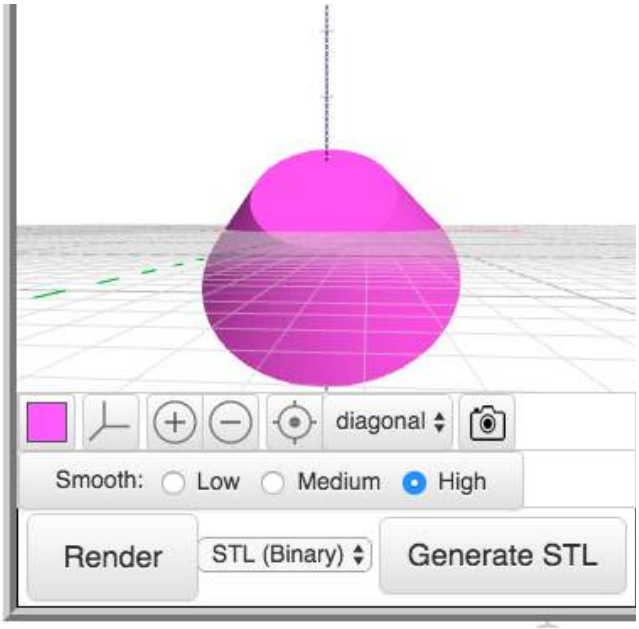
To see how this works, let's work with a cone: First, we will just rotate around one axis at a time so you can see exactly what is happening:

With all the angles set to 0°, we get the unrotated image of our cylinder.



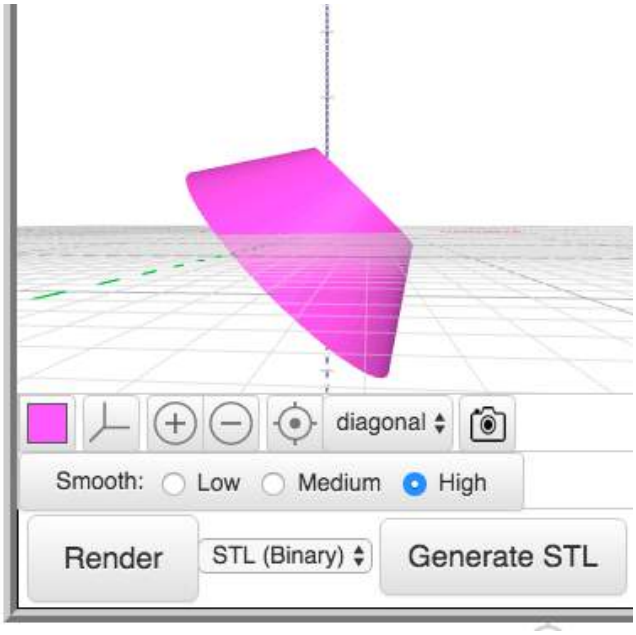
Next, we rotate around the X-axis by 45°

This tilts the cone forward in our view.



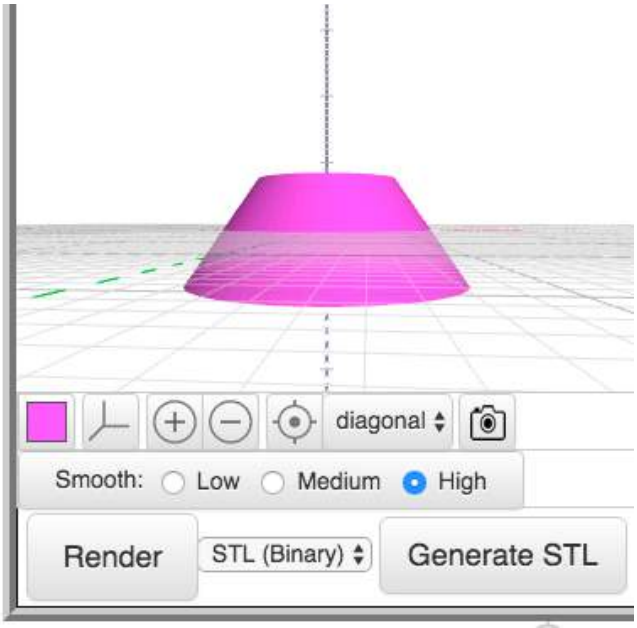
When we rotate around the Y-axis we get the cone tilted to another direction.

This rotation started from the original cone position.



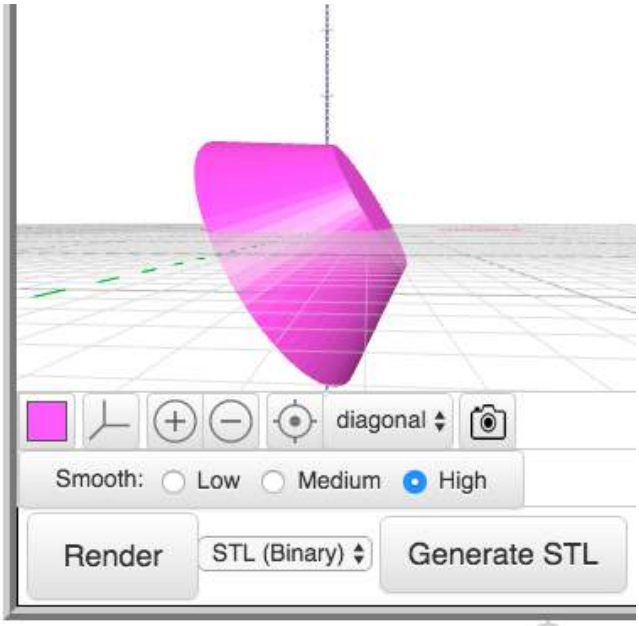
Finally we can see how things look when we rotate around the Z-axis.

This gives us the original shape because the cone is rotationally symmetric around the Z-axis.



But if we rotate around all three axes with one command, we get another position for the cone.

This orientation comes because all three rotations were done in sequence (first the rotation around X, then around Y, and then around Z) without resetting the cone position between each one.



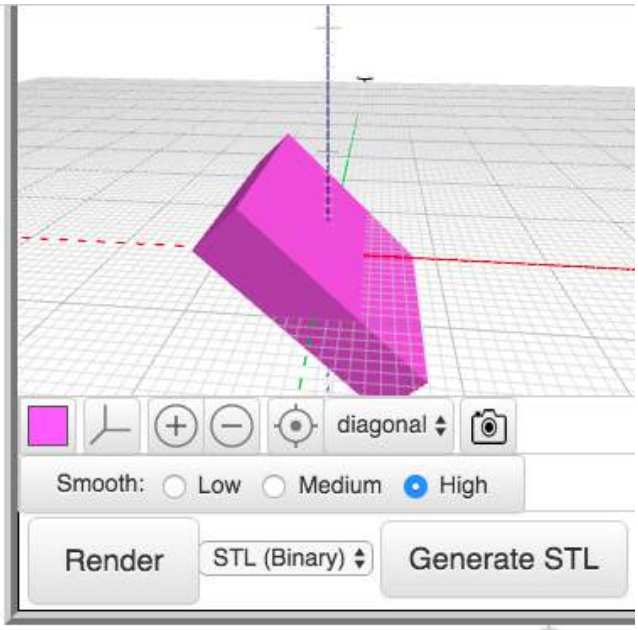
Because this can be confusing, I recommend that you only rotate around one axis at a time, and use multiple Rotation blocks as needed. This way, if your final orientation isn't what you had in mind, it is easy to fix.

Mirror

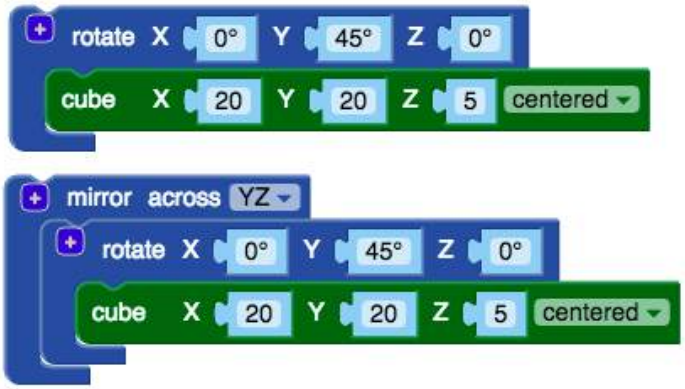
The mirror transformation creates a reflection (mirror image) of the object. This is what the Mirror block looks like.

To illustrate this, we will draw two boxes, one rotated 40° around the Y-axis, and the second as a mirror image of this box reflected across the YZ-plane. First, we draw the rotated box.

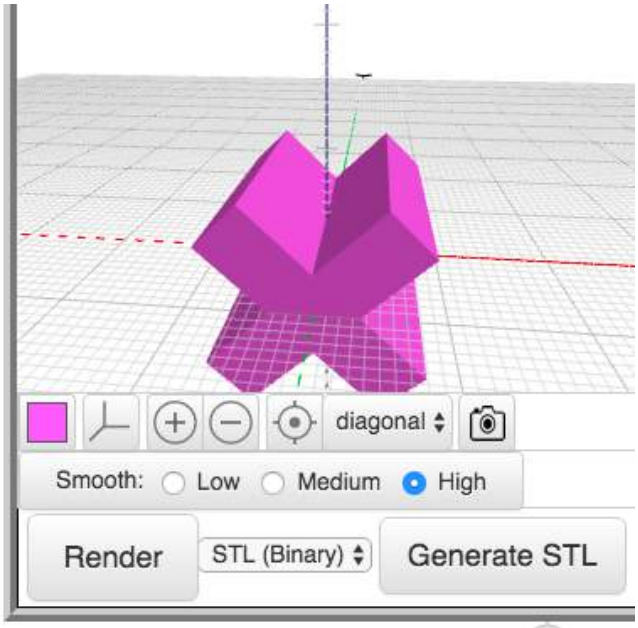
This box shows the original shape



Next we draw the mirror image after setting the mirror plane to YZ.. The plane you choose is the one holding the mirror. The Mirror block provides three options for the mirror placement, the XY, XZ, and YZ planes. If you want to reflect around another plane, you can use the Rotate block before using the Mirror across block. The choice of mirroring plane is completely up to you.



The mirrored image and the original one look like this.

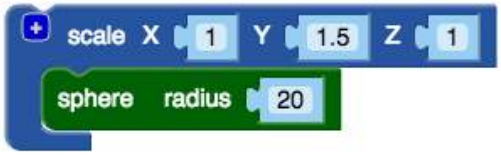


Because the original rotation was 45°, the resulting shape is the kind of “X” normally associated with the multiplication sign.

Scale

The Scale block is perfect for simple changes to an existing shape. The command looks like this:

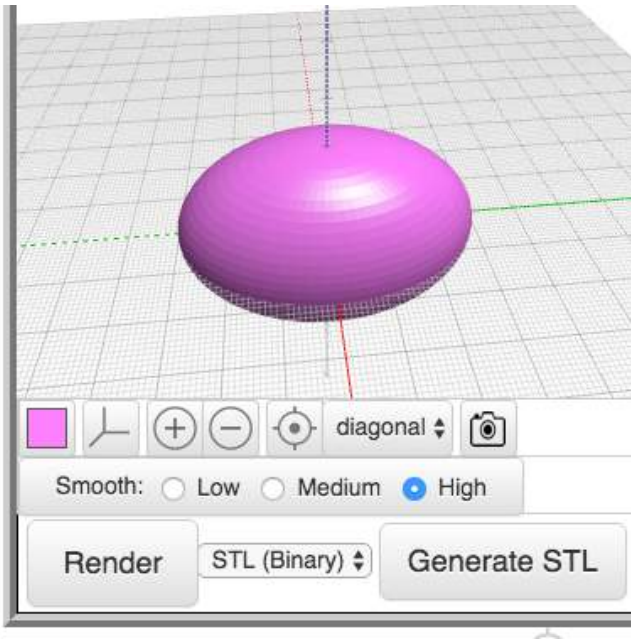
The values of X, Y, and Z determine how much a shape will be scaled along each axis. Suppose, for example, we want to turn a sphere into an egg shape. Here's how you'd do that:



Scale is multiplicative, so the final shape will be the same size in the X and Z directions, but will be 1.5

times longer along the Y-axis.

The resulting shape looks pretty good!

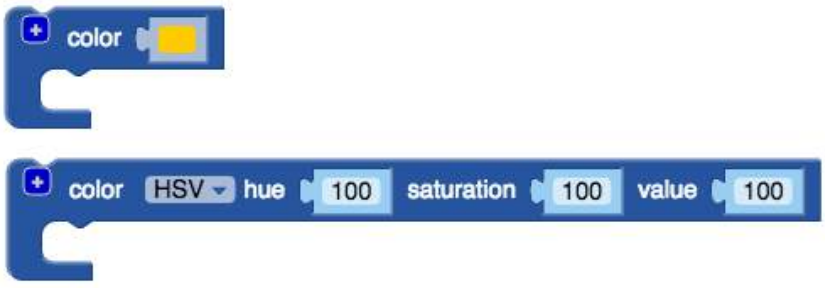


The Scale block is quite handy when the shape you want can be made by stretching or pushing an existing shape along the three axes.

Color and **Color HSV**

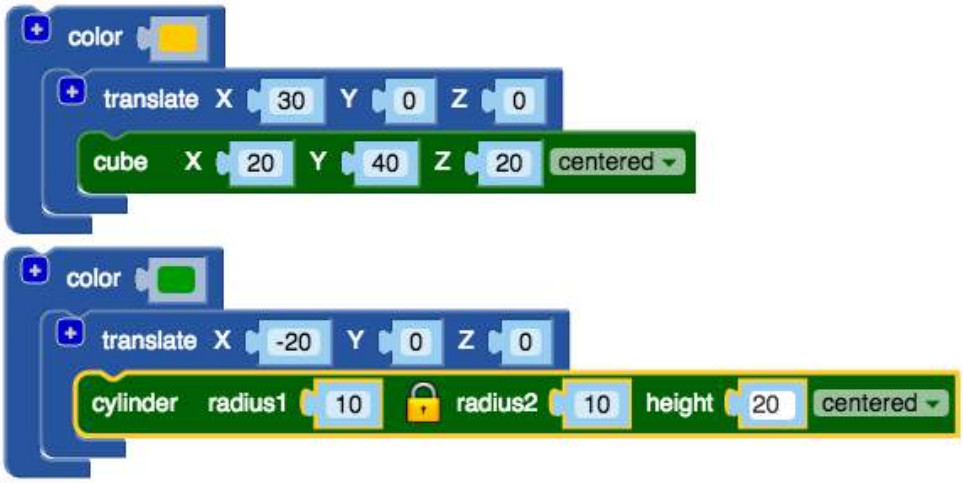
Most likely, your object will be printed in only one color. There are some 3D printers that print in multiple colors, typically by having more than one extruder. In that case, the object file has a different format from STL so it can include color information. The color of your final print is determined by the color of the filament you use. Color can be useful in your model if you have a complex shape and you want to highlight it on your screen by showing it in a different color.

BlocksCAD has two color blocks, Color and Color HSV

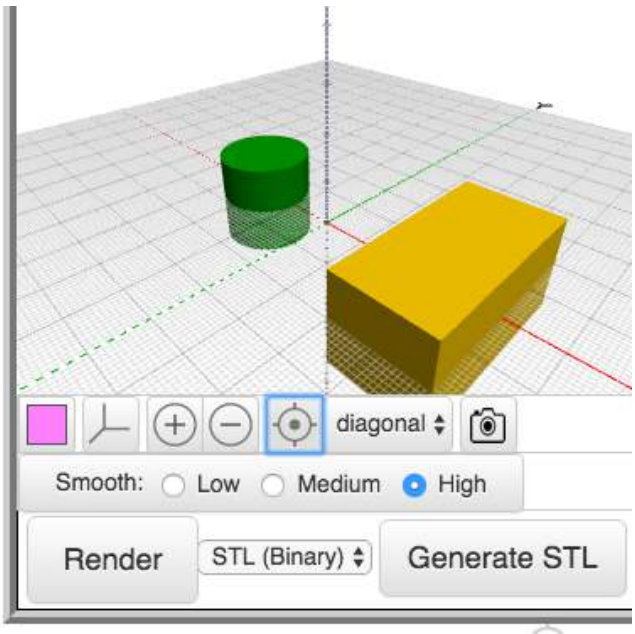


The first of these lets you choose a color from a palette by clicking on the color box, and the second from a set of numbers representing Hue, Saturation, and Value. Alternatively, the second Color block lets you use values for the thee primary colors: Red, Green, and Blue. The only advantage of this Color block is that it allows the displayed color to be calculated, not just chosen by hand.

Here's how the basic command can be used to draw a block and a cylinder with different colors:



Here's how the shapes look when rendered:

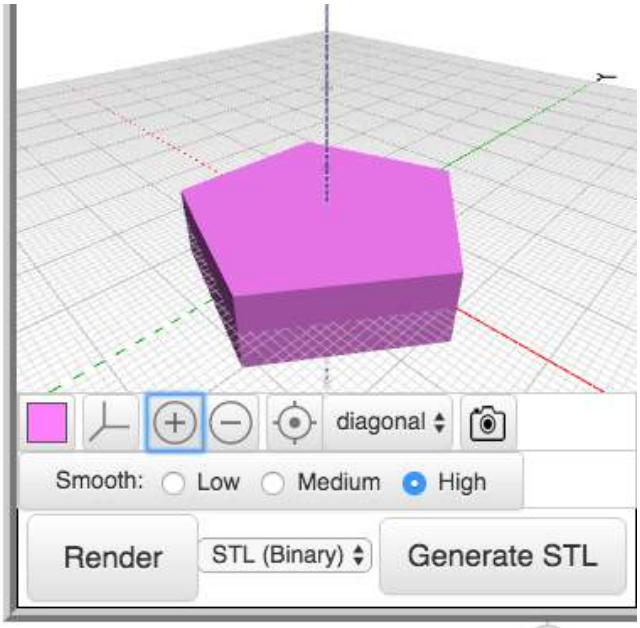


Sides

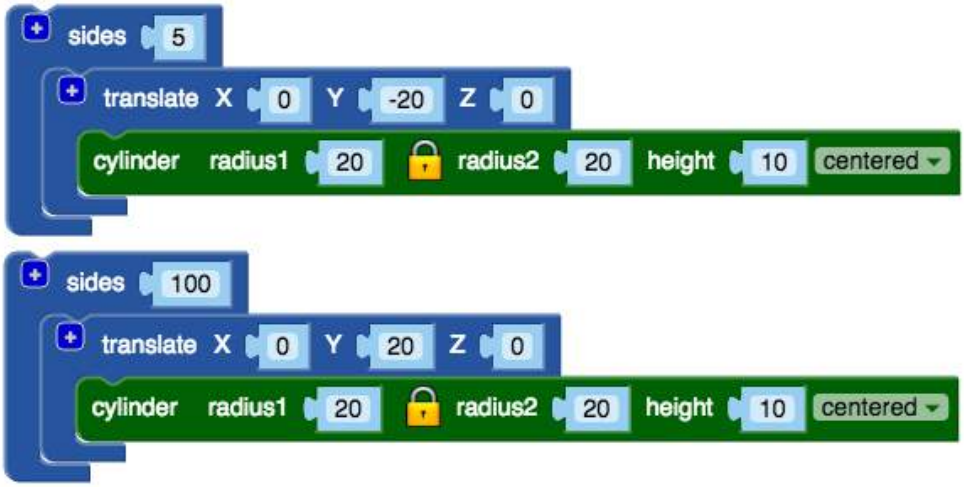
In the previous chapter we showed how to smooth the surfaces of curved shapes using the Smooth command under the box showing the rendered shape. There is a separate command for this operation that is fully compatible with OpenSCAD, and that is the Sides block.

For example, if we wanted to make a cylinder with a pentagonal cross section, we could do it this way.

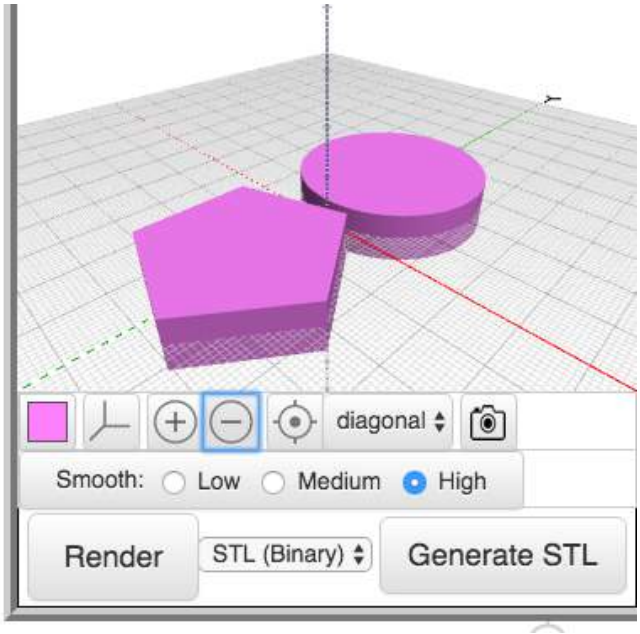
This gives us the following shape.



Also, you can use this command with different values for different numbers of sides:

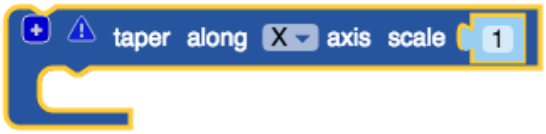


This gives us two very different shapes. The one with 100 sides is a very good cylinder.



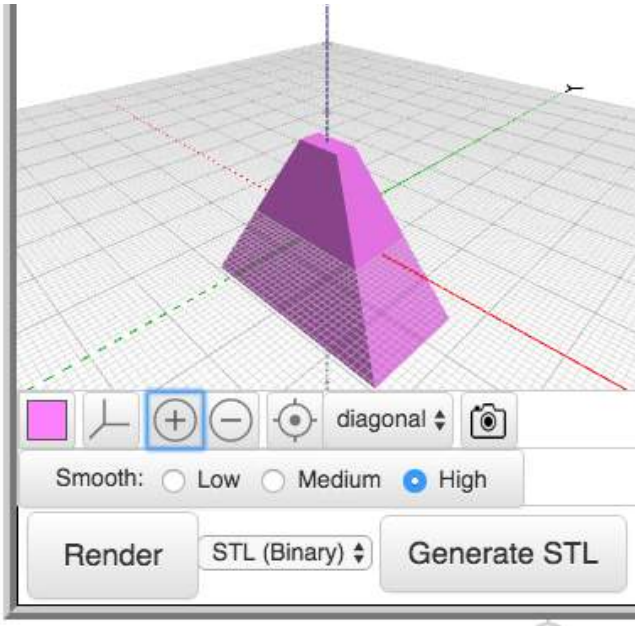
Taper along

The Taper along block is very elegant. It lets you take any shape and taper it along one of the three axes:



For example, we can take a simple block and make it into a tapered shape:

This gives us the following design:



This is a very handy tool to have. There is only one problem with it. If you click on the exclamation sign for this block you'll see the following:

This message lets you know that the block is not compatible with the OpenSCAD language, which is only important if you are exporting your designs to that format for posting on a site like (for example) Thingiverse.

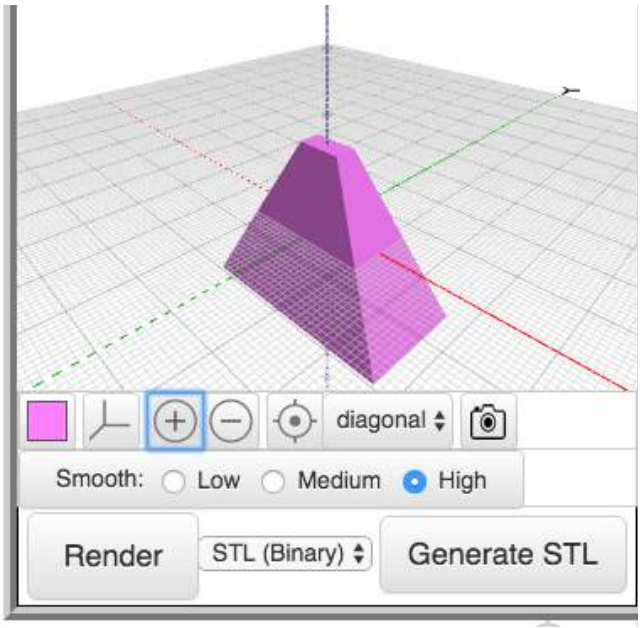
Don't worry, though, there is another (less elegant) way to achieve the same result that is compatible with OpenSCAD.

Linear extrude

The main tool compatible with OpenSCAD for creating tapered shapes is the Linear extrude block.

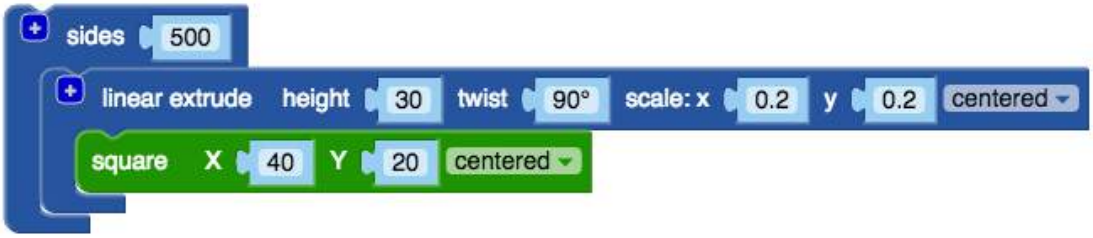
This block lets you extrude a shape to any height you wish, with different scaling values for the X- and Y-axes, and with a twist along the way. Instead of working with a 3D shape, this command works with a 2D shape — in our case, a quadrilateral.

This command gives us the same shape we just made with the Taper along block:

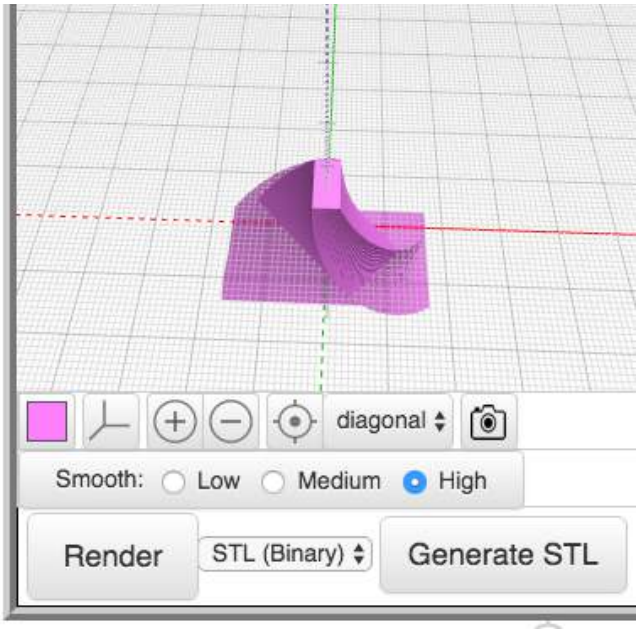


Note: Since the Taper along block is not compatible with OpenSCAD, you should use the Linear extrude block if you plan to make objects that others might edit or render using OpenSCAD themselves.

As mentioned, Linear extrude also lets you produce a twist along the Z-axis. For example, a tapered block with a 90° twist is made this way:



I set the number of sides to 500 to produce a smooth twist. You should experiment with different values to see what you get. By the way, any time you render with a lot of sides, the rendering time goes up. I usually experiment with low values of the Sides block until I get the right overall shape, and then set the final value for the rendered shape.



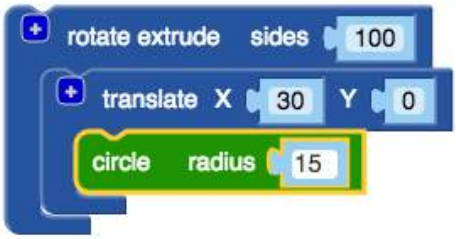
Rotate extrude

The Rotate extrude block is not very intuitive, but it can sure make some interesting shapes.

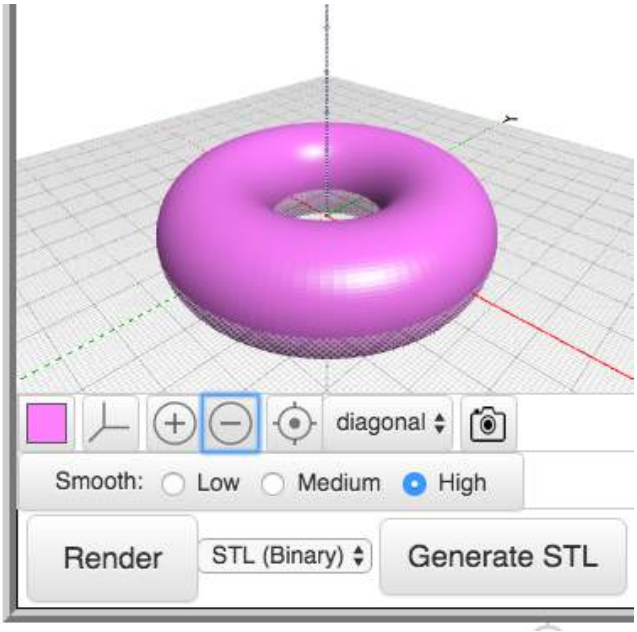
Rotational extrusion spins a 2D shape around the Z-axis to form a solid which has rotational symmetry. One way to think of this operation is to imagine a potter's wheel placed on the X-Y plane with its axis of rotation pointing up towards the positive Z-axis. Then place the to-be-made object on this virtual potter's wheel, take the cross-section of this object on the X-Z plane, but keep only the right half (X >= 0).

Since a 2D shape is rendered by BlocksCAD on the X-Y plane, an alternative way to think of this operation is as follows: spin a 2D shape around the Y-axis to form a solid. The resultant solid is placed so that its axis of rotation lies along the Z-axis.

For example, suppose we offset a circle with a radius of 15 mm along the X-axis by 30 mm:

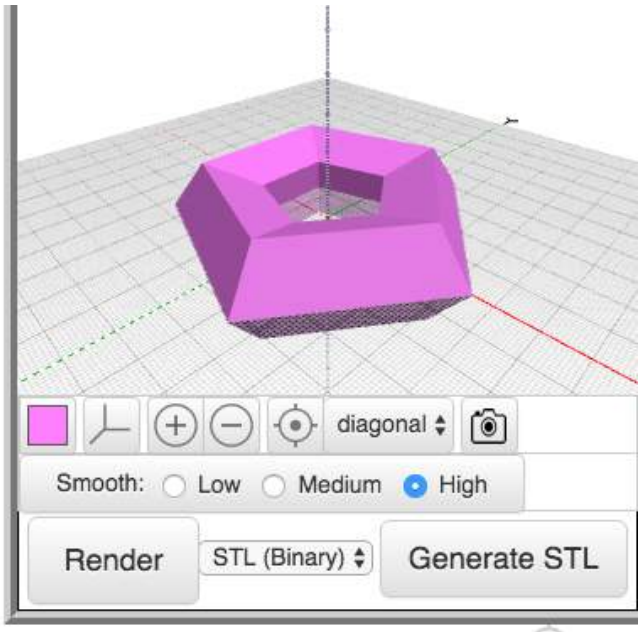


The result is a smooth, fat toroid.



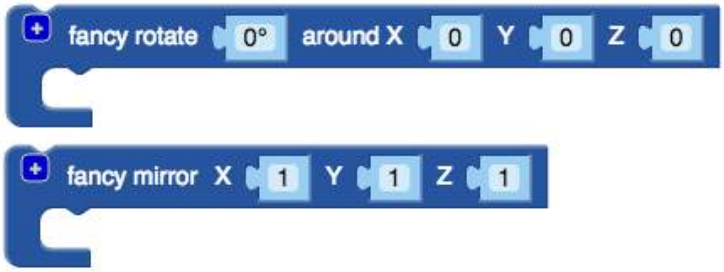
Next let's look at a shape with only 5 sides:

The number of sides is the same for both the cross section of the toroid as well as for the toroid itself:



If you wanted the cross section to have more sides, you can use the Sides block around the circle.

Fancy rotate and Fancy mirror
We've already explored the basic Rotate and Mirror blocks. The two remaining operations in the Transforms menu are based on these.



Because we already have explored ways to rotate and mirror objects, I think we can safely ignore these fancy operations for now.

Print orientation
When you design an object to print, it is always a good idea to plan ahead so your part will print properly and efficiently. For example, a part with sloping sides often needs additional support materials to hold everything in the right position as it is being built. While this support can be removed and discarded later. It is a waste of filament, and adds time to the the print job. Sometimes a simple rotation of the part by 90° is all you need to completely eliminate supports, making the print work far more efficiently. For example, the simple business card holder shown below requires 17.5 m of filament when printed the way it will be positioned when it is sitting on a desk, but only requires 9.5 m of filament if it is first rotated by 90° on the X-axis.

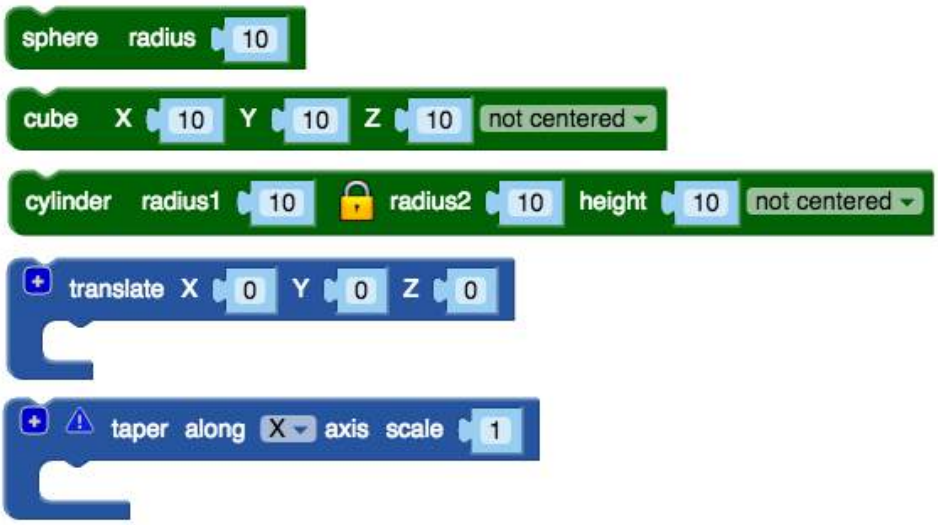
The holder on the left was printed in the normal position it would have when sitting on a desk. Because of the sloping sides, internal supports are needed to hold the sides up while they are being printed. These supports need to be removed when you are done, and they waste filament. The card holder on the right was rotated by 90° before printing. This made all the sides vertical, eliminating the need for supports. No clean up was required, no filament was wasted, and the holder printed faster.

Of course, some complex designs will always need supports, but careful planning can minimize these, leading to cleaner prints.

The more designs you print, the easier it will be for you to find the best orientation to use during printing, and to factor this into your design before it is rendered.

A Project for You

The tools described in the past two chapters let you create some neat shapes! Let's experiment by using what we already know to build a model of an R2-class robot using three of our basic shapes and two Transformation blocks:

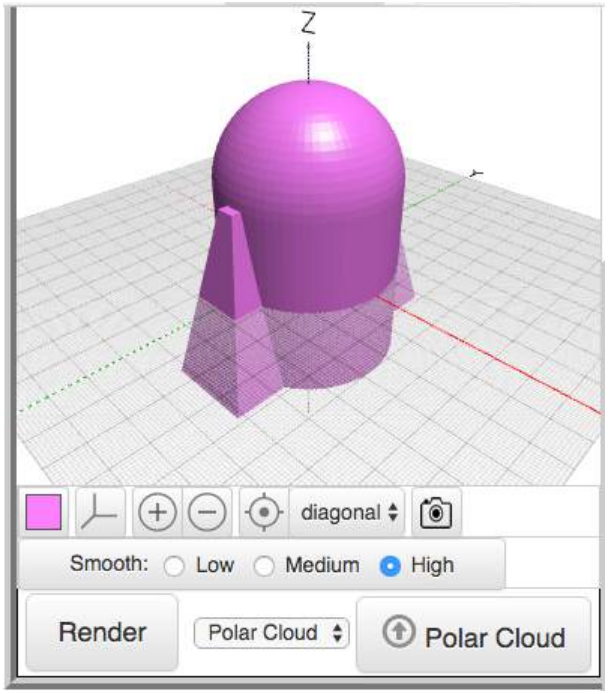


Some of these blocks will be used more than once. When you are done, your robot might look like this:

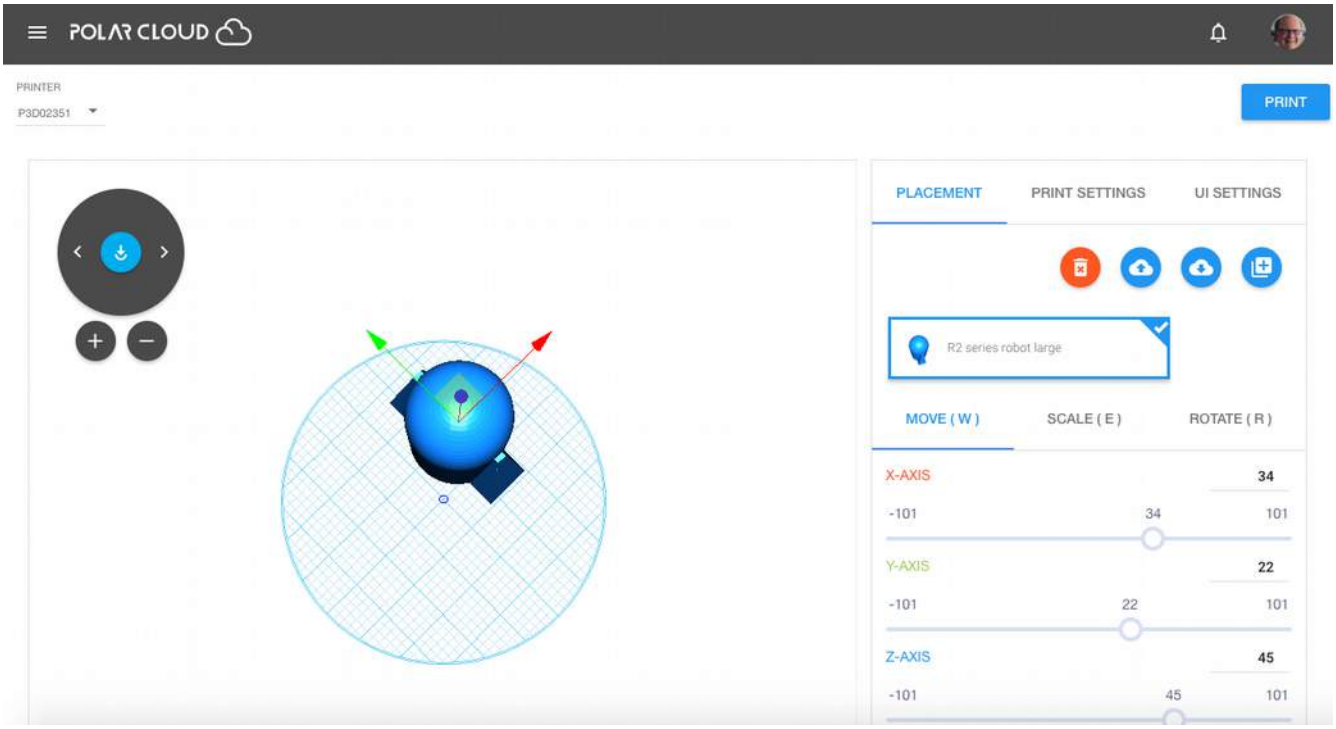
When finished, render the shape and generate the STL file for your printer. While you can make any size robot you wish, I made mine 90 mm high, with a radius of 30 mm.

Important note!

While you can export your model to your computer as an STL file, it is far better to save it directly to the Polar Cloud. This is done by choosing Polar Cloud next to the Render button.



Once you click on Polar Cloud (the one with the arrow pointing upward) your model will get sent to the build plate of the default printer and saved in your Objects library in the cloud.



Once the file is uploaded, you can choose the printer you want to use, adjust the position of the object on the build plate, and change the print quality — all without having to go through the intermediate step of saving your file to your computer. While handy in general, this feature is perfect for those using Chromebooks!

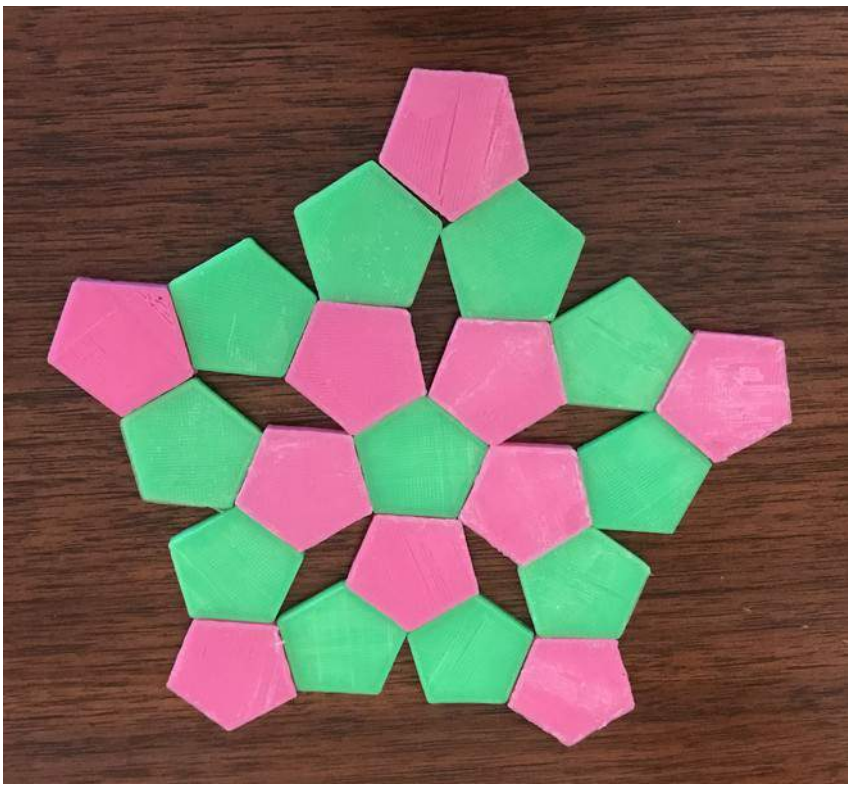
Here's what the robot looks like when it is printed.

At this point, go ahead and make a few more objects before going to the next chapter. Here are some ideas to get you started:

Pentagonal tiles
While you can cover a flat surface with equilateral triangles, squares, and regular hexagons, the same can not be said for regular pentagons. Test this out by making a bunch of pentagons and seeing what kinds of patterns you can make.

A cylinder can be turned into a pentagon if you set the number of sides to 5.

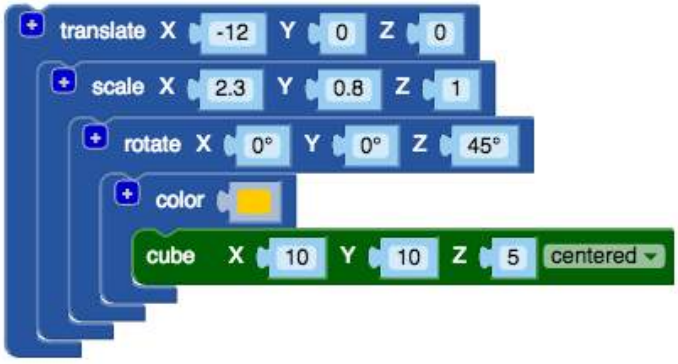
Make and print a bunch of pentagons and experiment with the patterns you can create.



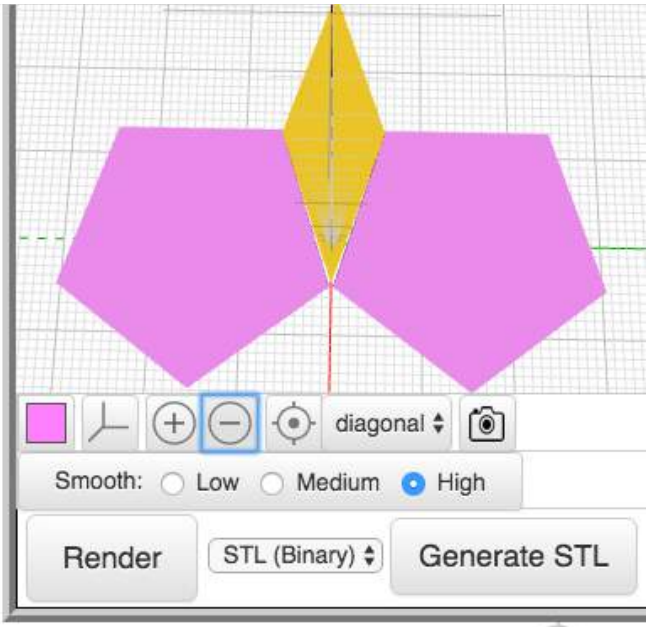
I made two sets of pentagons in different colors. They can be arranged into many interesting patterns, but they will never tile a flat surface without gaps. As a young student once said to me, “You will always need grout.” Of course, the addition of one more tile shape (a diamond quadrilateral) eliminates the need for grout. You should make some of these as a neat project and explore the different kinds of tiling patterns you can make.

Diamond tiles

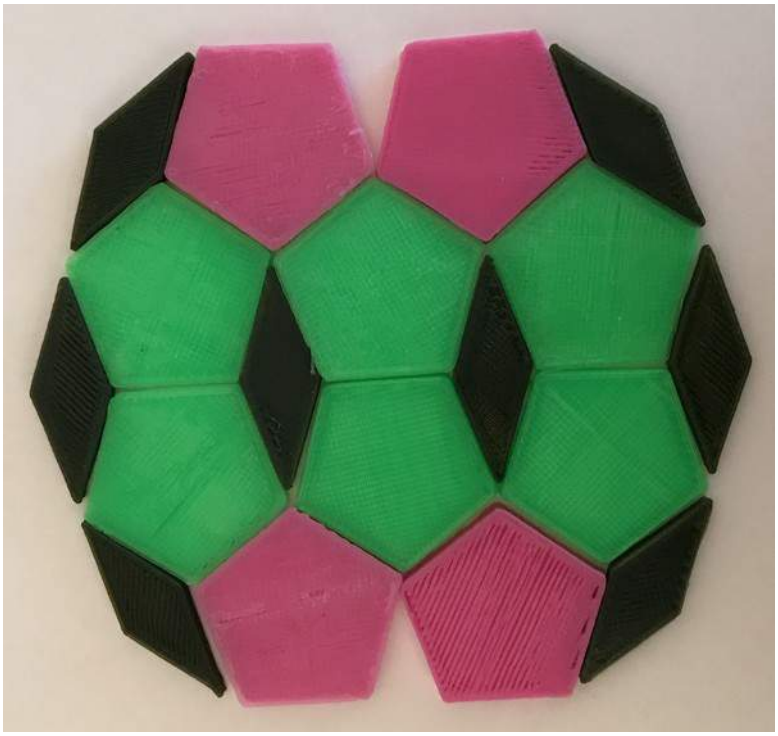
The gaps in the pentagonal tiling patterns can be filled with quadrilateral shapes. To make this shape, make two pentagons in BlocksCAD that just touch at the points. Next, use a transformed Cube block to make a diamond pattern that just fits the gap.



By tweaking the numbers in the Translate and Scale blocks, you can make a shape that just fits.

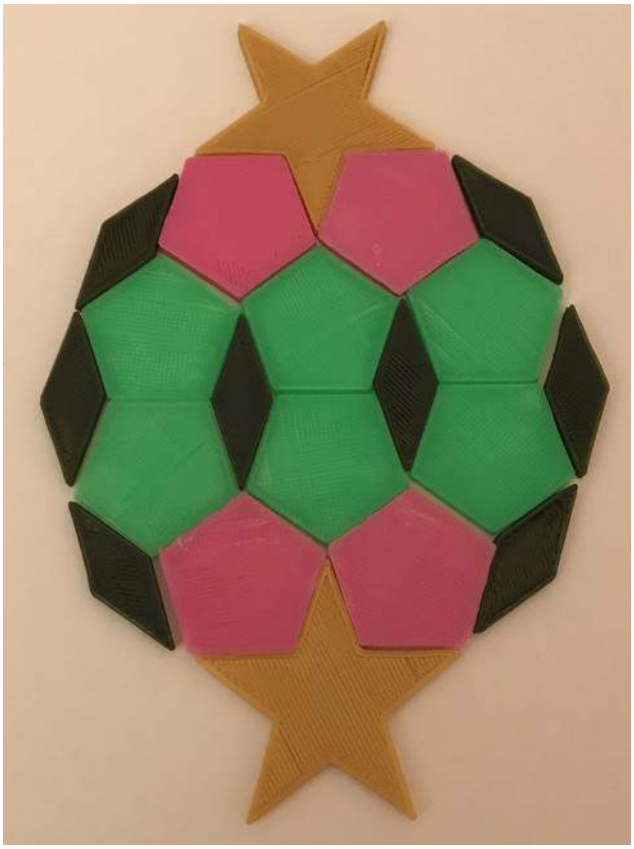


Once you print these shapes, add them to the pentagonal tiles you already printed and explore the ways you can tile a surface with these two tile shapes.



Stars

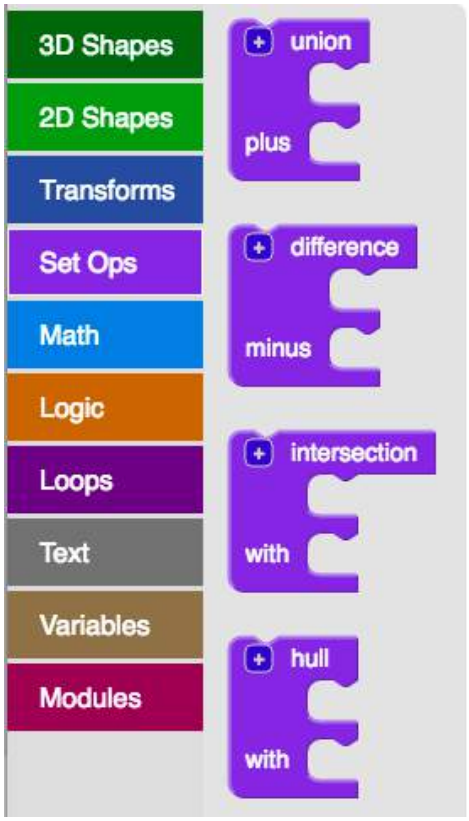
If you build this pattern, you will likely discover that you are going to hit a snag that can be fixed by adding a new “star” shape. I'll let you design this new shape on your own if you are so inclined. Once you do, print a bunch in a different color and keep on building.



Pretty cool, eh? The nice thing about tiling patterns is that they are artistic and mathematical at the same time. If you get bitten by the tiling bug, you'll start seeing nice patterns everywhere you look. Floors, walls, counter tops, quilts, honeycombs; the list goes on and on. Because they are fairly thin (3 mm is a good thickness) they print pretty fast, making it easy to experiment with lots of patterns. The mathematics of tiling (tessellation theory) is fascinating, and I encourage you to explore it.

Chapter 4 — Set Operations

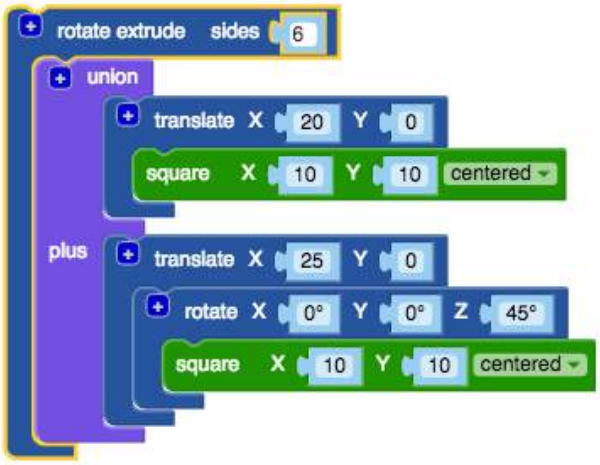
In this chapter, we'll explore four basic operations that can be used to combine several shapes into one. These operations can be used by themselves, or combined with other operations to create a wide variety of interesting and useful shapes. Our goal is to add some more useful tools for your kit to make your BlocksCAD experience even more powerful.



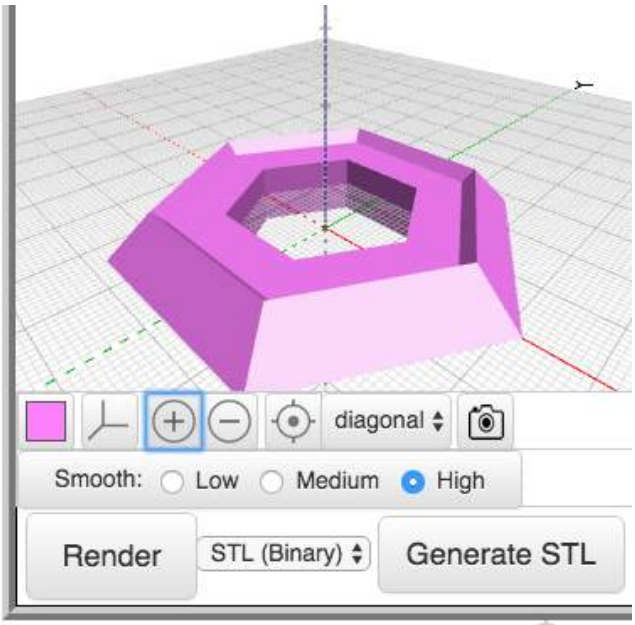
Union

The Union operation is the simplest of the four we will explore. All it does is combine several shapes into one. The basic format for this operation is:

Union just combines a bunch of shapes into one. For example, the Rotate extrude block expects to have one 2D object with which to work. The Union block lets us combine several 2D shapes into one, which allows us to make more complex shapes with the Rotate extrude block:



This produces the following shape:



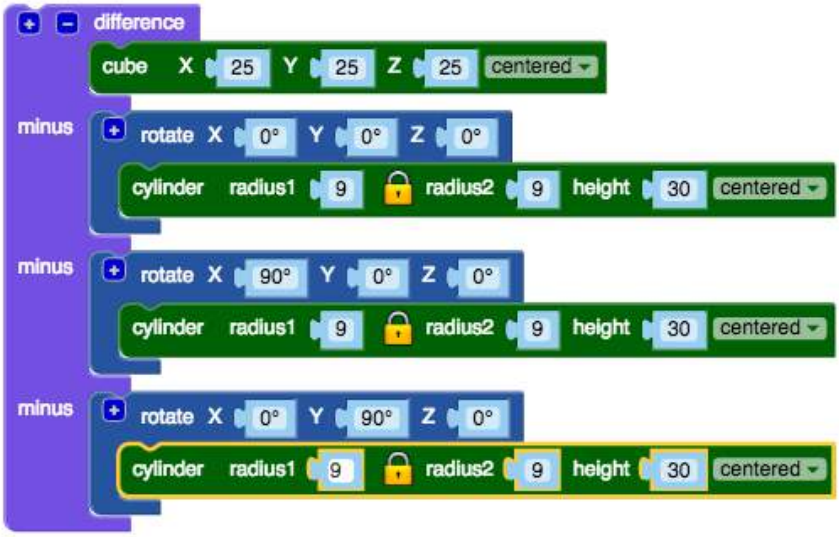
The Union block can work with any number of shapes (2D or 3D, but only one kind at a time). To add more items to the block, just click on the + sign at the upper left of the block.

In most cases, if all you are doing is combining some shapes together, the Union operation is not really needed. The value of this operation comes when we combine it with other operations. It is also handy to make the listing of your program pleasing to the eyes. Another benefit of the Union block is that it can make the shape easier to print by eliminating hidden boundaries of overlapping shapes. Historically, I've been pretty sloppy when it comes to using the Union command, but after some bad experiences with complex models, I'm getting to the point where I use it a lot.

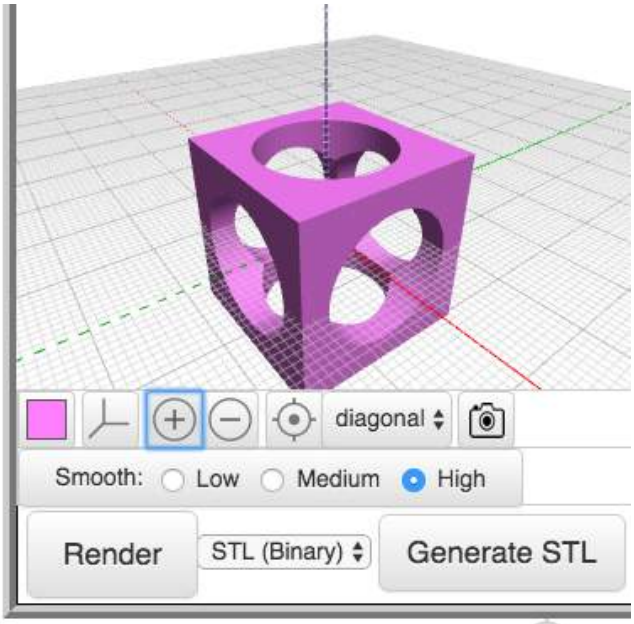
Difference

This operation is one my favorites for creating nice complex shapes. What it does is subtract one shape from another one. The basic form for this operation is:

As with the other blocks in the Set Ops category, you can add more items to be subtracted by clicking on the + sign. For example, consider the following shape

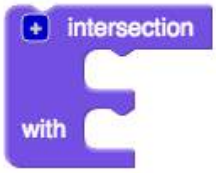


This makes the following complex shape:

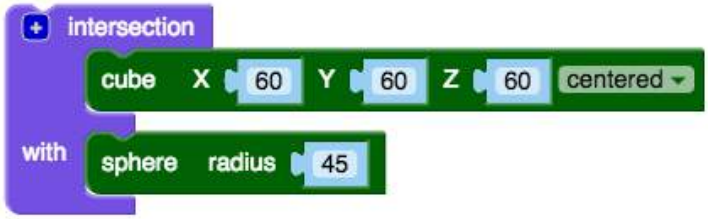


Unlike the Union operation, this one works a bit differently. The first shape in the list is the object from which the remaining shapes are subtracted. If you add more spaces for items to subtract than you need, you can remove them (starting at the bottom of the block) by clicking on the – (minus) sign at the top of the block.

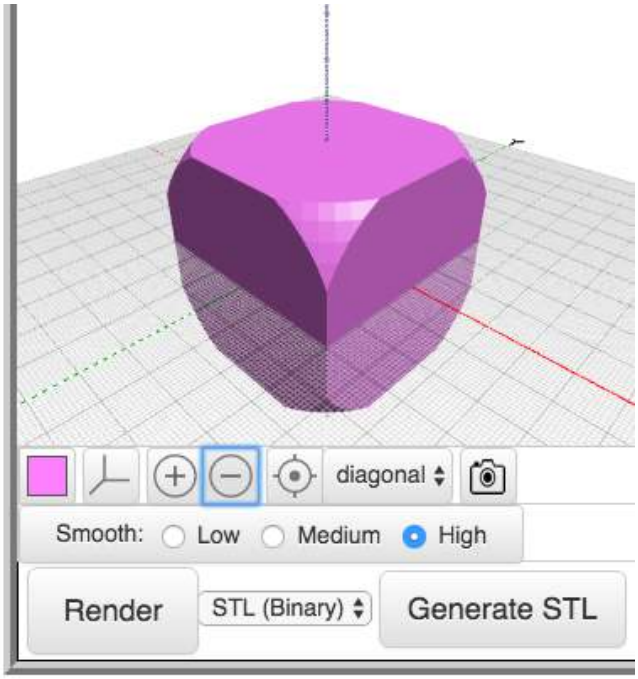
Intersection
Our next operation creates a final shape based on the intersection or overlap of various shapes. It has the following basic form:



Basically, this command is like a special version of the Union block: it adds shapes together. The difference is that the resultant shape only includes those parts of the original shapes that intersect or overlap with each other. For example, suppose you wanted to make a cube with rounded corners (for a game die, perhaps). The command might look something like this:



This produces the following shape:



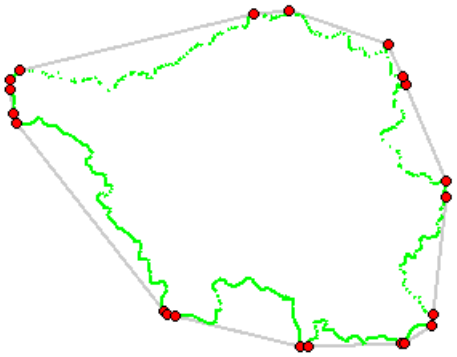
Because large parts of the sphere fall outside the shape of the cube, they don't appear in the final part. Only the areas that intersect show up in the final shape.

As with the other operations in this chapter, you can add more shapes to this command.

Hull
Our final tool for this chapter, the Hull block, lets you put a convex hull around a collection of shapes.

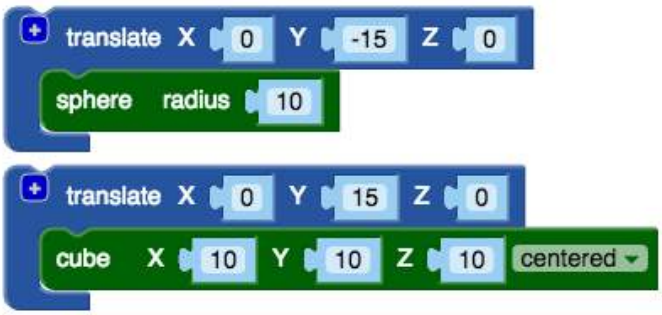
A convex hull can be thought of as the smallest enclosure of a shape so that no parts of the shape poke

through, and all parts (including disconnected ones) are enclosed. In other words, it is as though your final shape is a shrink-wrapped version of the original.

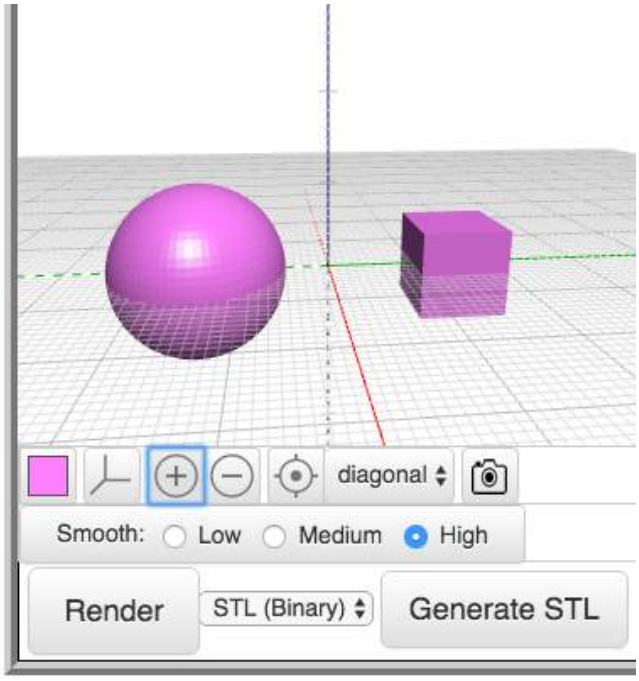


Hulls can be found for both 2D and 3D objects. The basic command is:

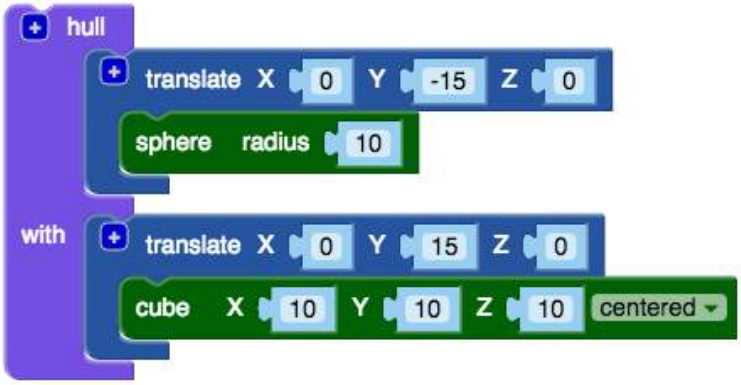
Suppose we start with two shapes — a sphere and a cube:



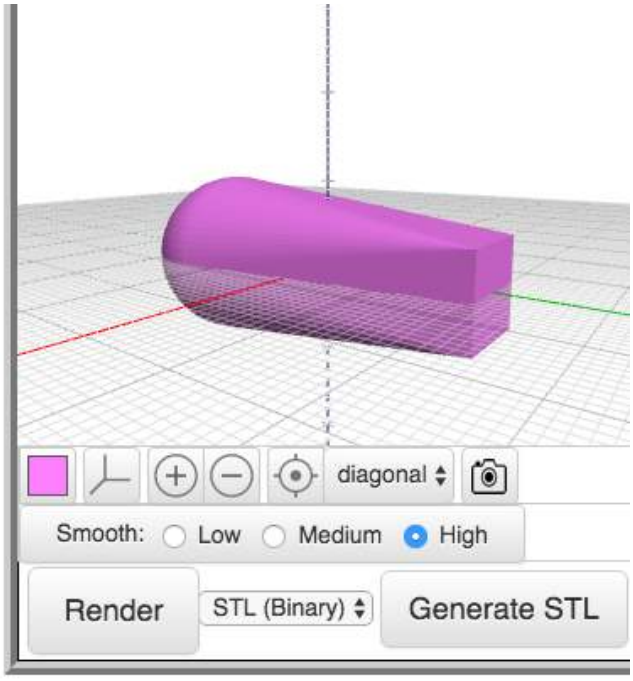
This gives us the following:



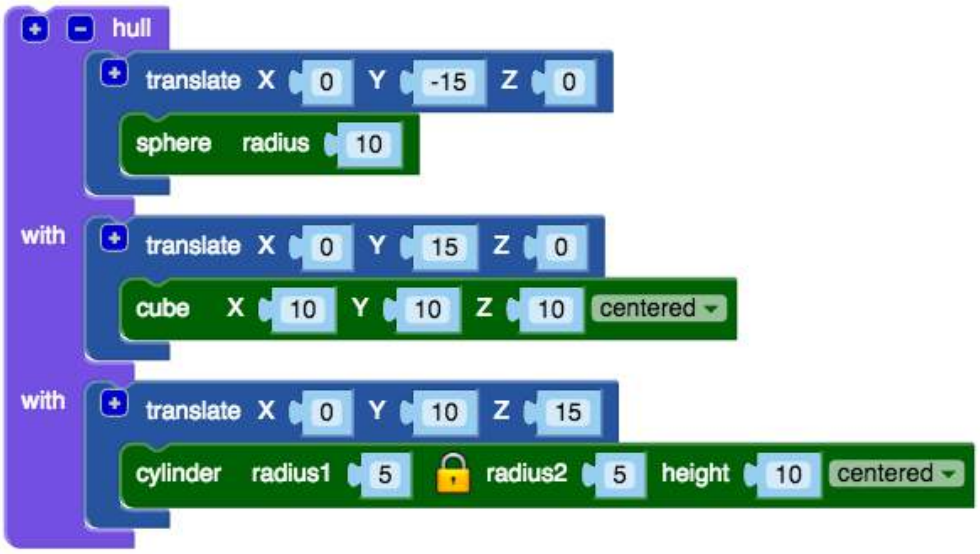
Now we'll see what happens when we make a hull from these two shapes:



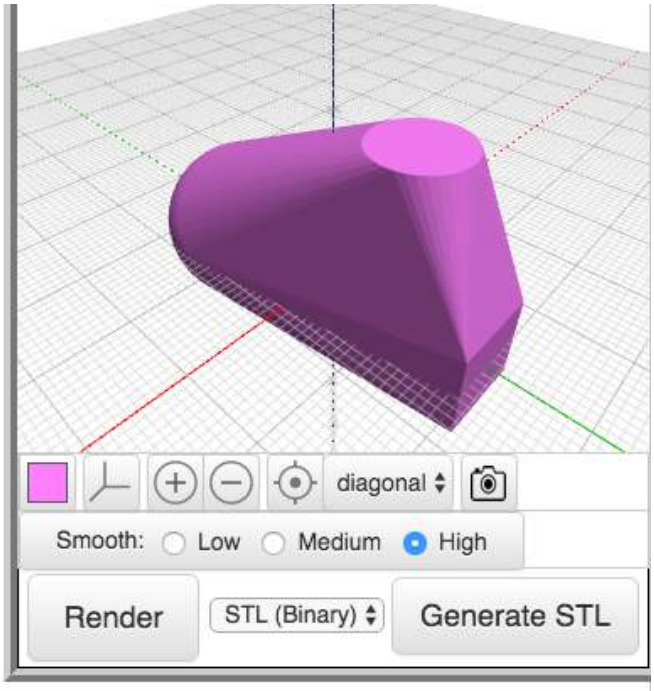
This is our result:



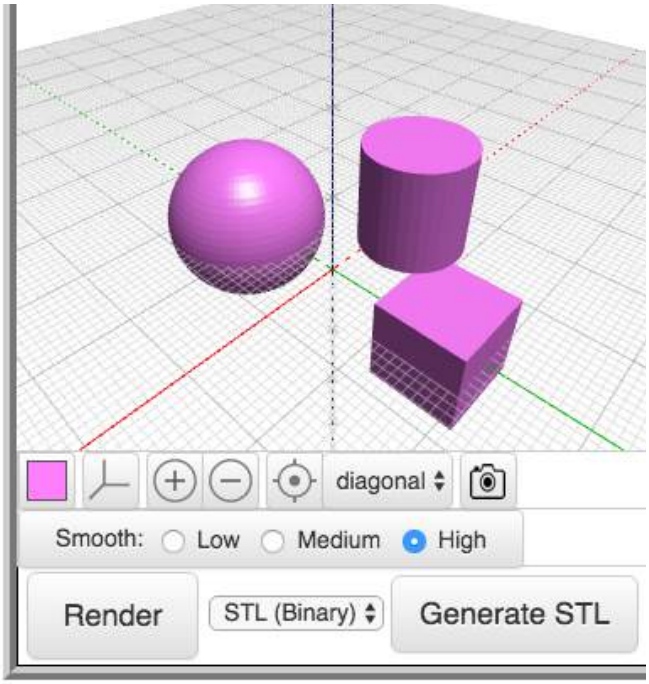
Now let's add one more shape to our project.



Here's the result:



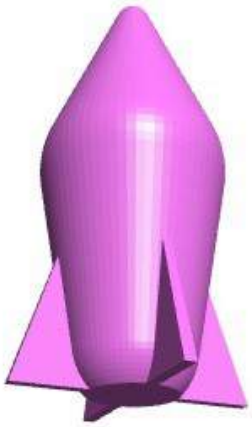
If we render the shapes without the Hull block, we just get three separate shapes:



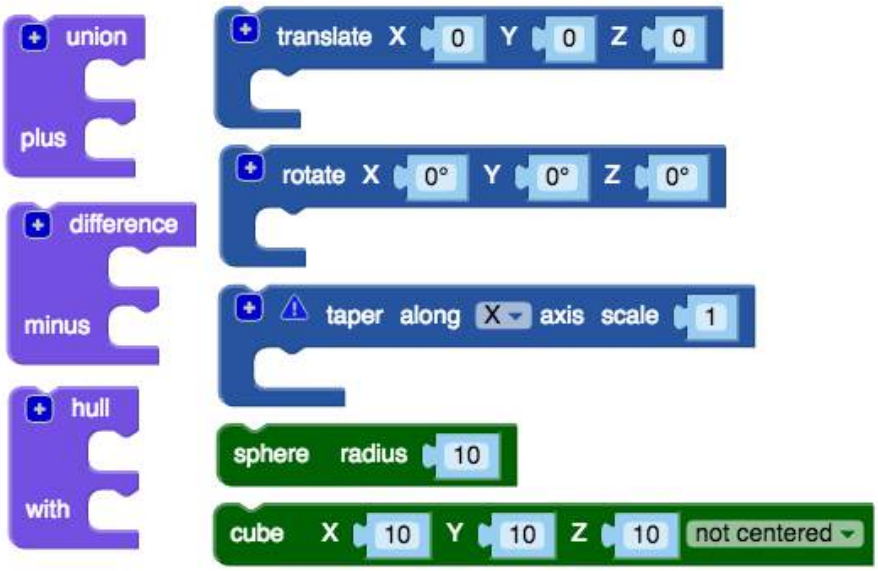
These four simple operations will get a lot of use, so the more you know about them, the better.

A project

Your task for now is to experiment with what you've learned and come up with some interesting shapes of your own. For example, you might want to make a model of a rocket:

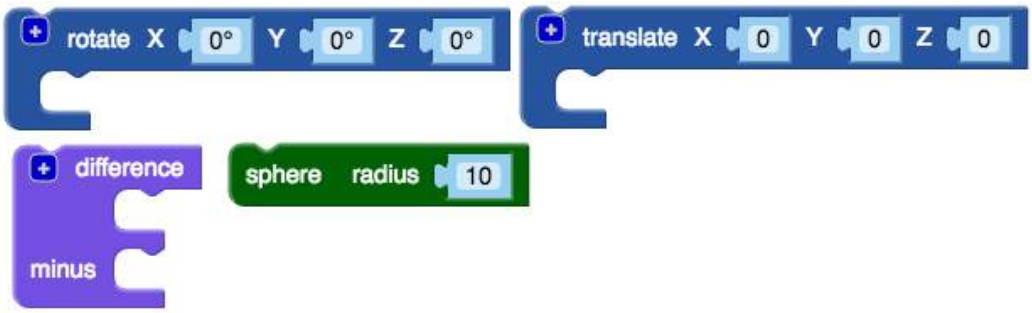


Your challenge is to build this model (or one of your own design) using only these operations and shapes:

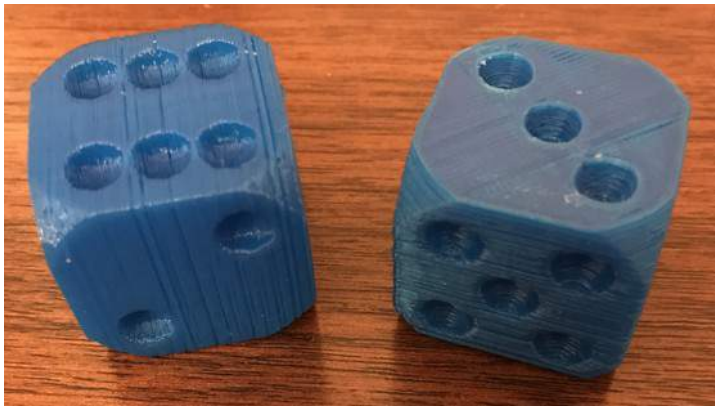


Here's what our model looked like when it was printed.

Another project
Earlier on this chapter we made the block of a gaming die. Now, adding these blocks, make a real set of dice.

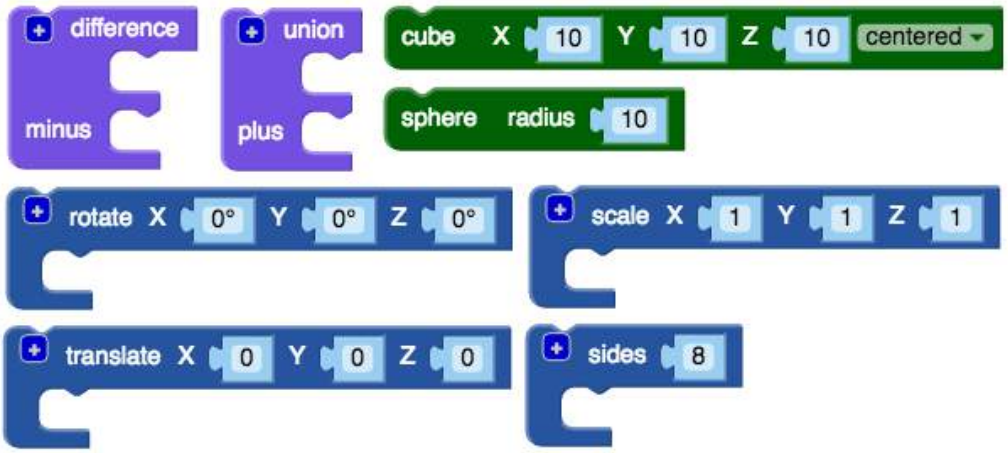


The dots are made with small hemispheres and are placed so the sum of the numbers on opposite faces equals seven. Here's the final set. Note we made two dice because of the rule that says “Never say die.”

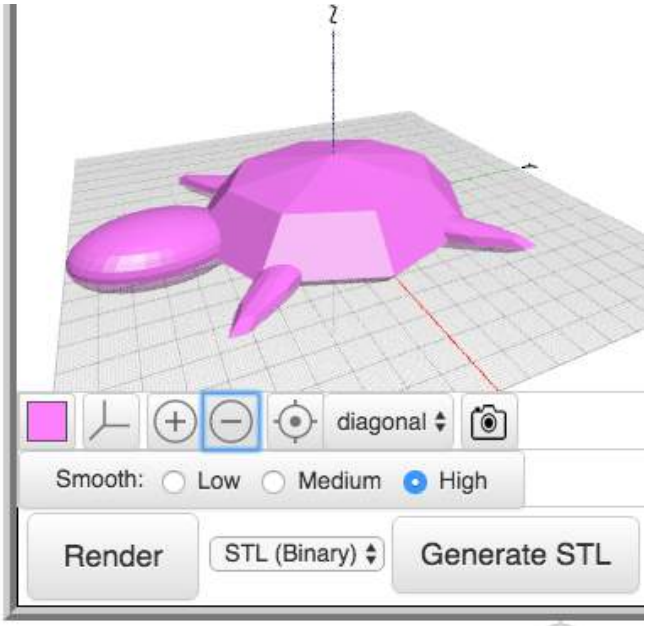


Design your own turtle
Turtles have shells, legs, a head, and a tail. Can you design your own turtle?

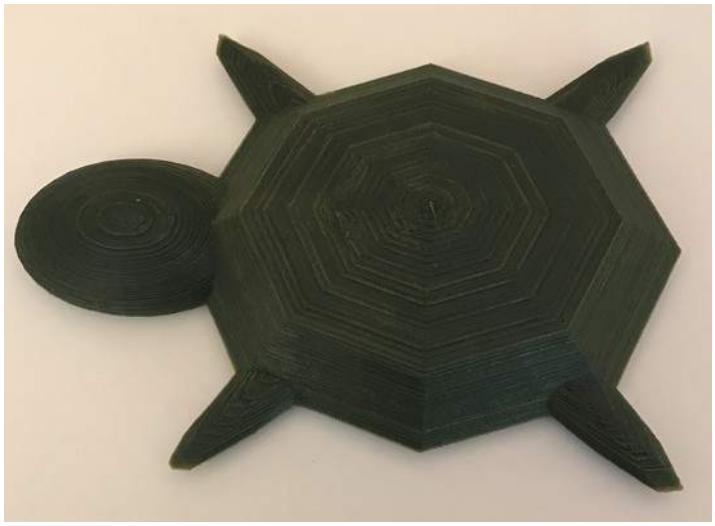
Here are some blocks to use:



Hint: the rough shell was made by applying the Sides block with a value of 8.



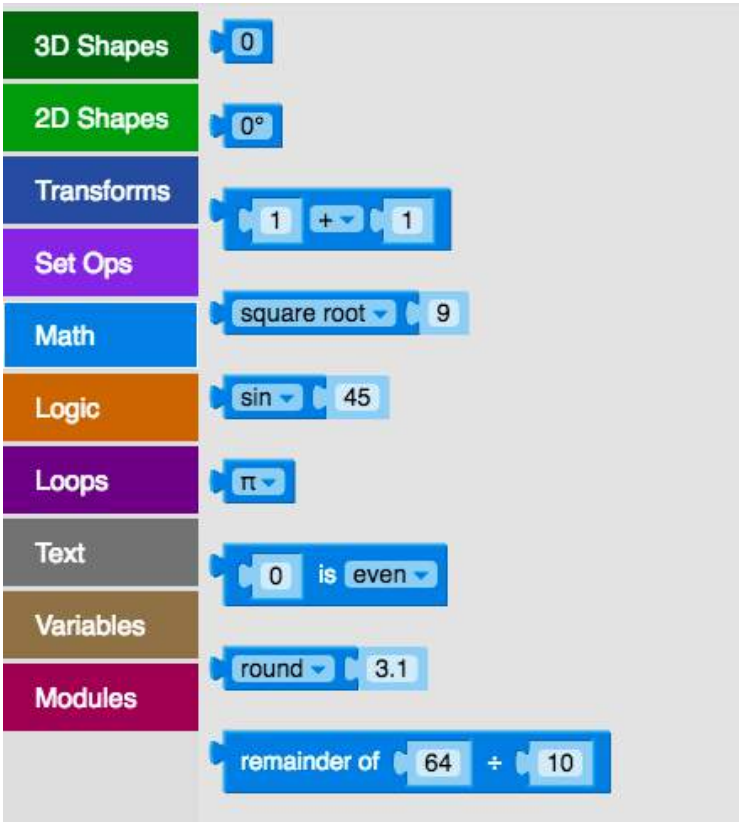
Here's how the finished turtle printed.



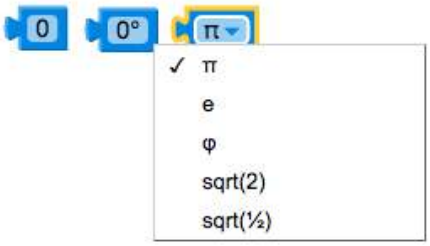
Next we'll explore how BlocksCAD handles mathematics.

Chapter 5 — Math

BlocksCAD handles a wide variety of mathematical operations. These operations can include numbers and variables. Numerical values can have decimal values (3.1415, for example) and, as we've seen, can represent sizes in mm, angles in degrees, and numbers in general.

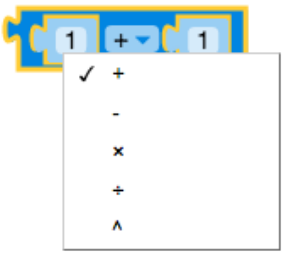


Let's start by exploring the numbers themselves:



The first two items on the screen are numerical values representing (for example) a length in mm, and an angle in degrees. The third item produces the value of some commonly used constants, pi, e (the base of natural logarithms), phi (the golden mean – roughly 1.618), and the square roots of 2 and ½.

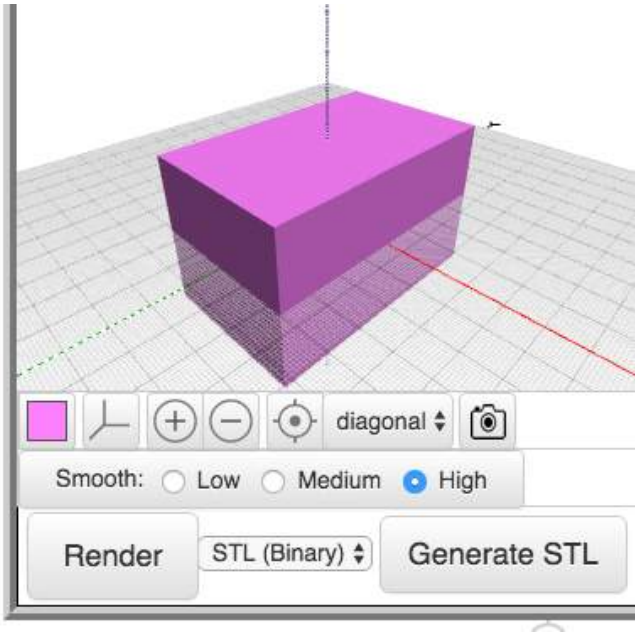
Simple arithmetic operations on numbers can be performed with this block:



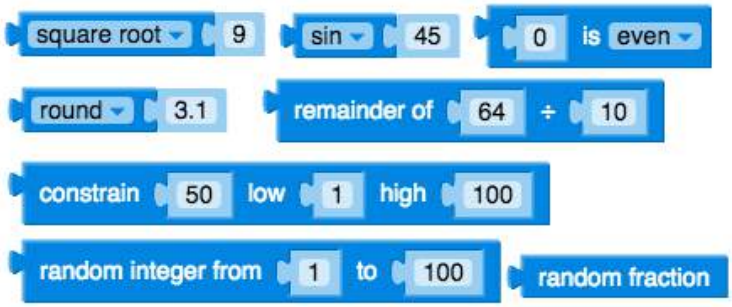
The result of this block is the combination of two numbers through either addition, subtraction, multiplication, division, or exponentiation (raising the value of the first number to the power of the second number).

For example, if you wanted to make a rectangular block 50 mm on one edge whose cross section was the ratio of the golden mean, you could create the following:

You'll see that I plugged in a combination block for the Y-value so I could set it to 50 x *phi*. The resulting shape looks like this:



In addition to arithmetical calculations, BlocksCAD includes other operations.

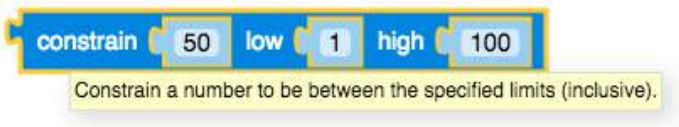


The Square Root block contains the following operations: square root, absolute (value), - (negation) , ln, log 10, e^x, and 10^x,

Trigonometric functions are found in the Sin (sine) block, including sin, cos, tan, asin, acos, and atan. Remember that trigonometric functions work with angles in degrees, not radians.

The Round block rounds a number to the nearest integer. It also has settings to round up to the next highest integer, and to the lower integer. The remaining functions are pretty self-explanatory.

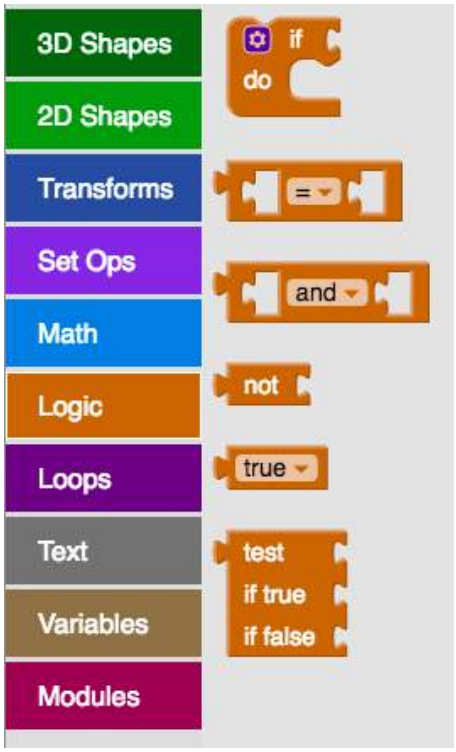
That said, if you ever want to know what a block does, just hold your mouse over it and an explanation will appear!



This is a great feature of BlocksCAD. For example, if you hold your mouse over the Cylinder block, the various dimensions are explained:

Chapter 6 — Logic

Before getting to the application of all this material, there is another set of operations that BlocksCAD uses. These are statements that evaluate to one of two values: true or false. They are named “Boolean” operations after the British mathematician George Boole, who pioneered the field of mathematical logic (without which digital computers would not be possible).



Here are the basic Boolean operations produced by this block:

Each of the two spaces holds an expression, “a” in the first space and “b” in the second blank space. You determine the operation by selecting the sign in the middle from the pull down menu.

(a>b) is true if a is greater than b
(a≥b) is true if a is greater than or equal to b
(a<b) is true if a is less than b
(a≤b) is true if a is less than or equal to b
(a=b) is true if a and b have the same value
(a≠b) is true if a and b have different values

To see how these work, here are some examples:

(4>7) Is false
(5≥5) is true
(3<29) is true
(15.2654≤15) is false

(9.5=9.5001) is false

While this block can work with numbers, it produces a logical result (true or false.)

On the other hand, there is another block that works with two logical expressions:

The result is also logical – true or false. The options for this block are “and” and “or”. For the “and” setting, the expression evaluates to true if both a and b are true. If “or” is selected, the result is true if either a or b is true. For example:

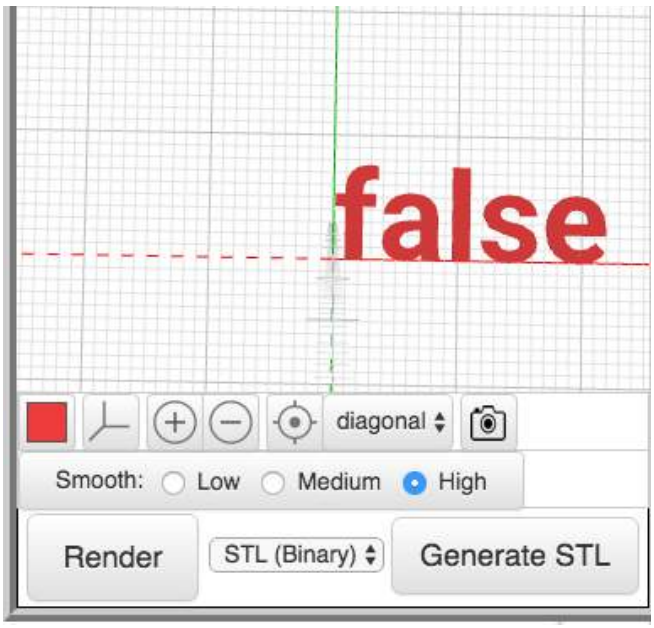
is true and

is false.

Handy tip

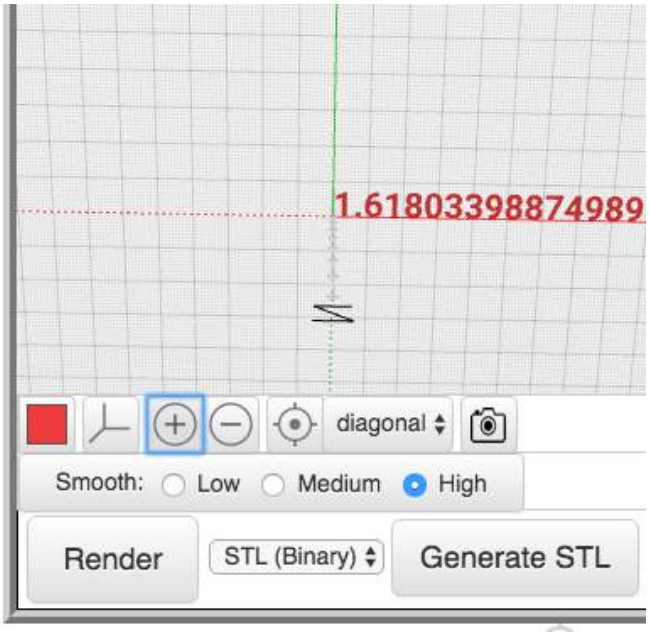
You can tell what a logical expression evaluates to by plopping it into a Text block in BlocksCAD. For example, putting (3>2) into a Text block will print "true." We'll explore text in another chapter, but for what we want to do, choose a 2D Text block and replace the “abc” entry with a logical expression.

When you render this expression, this shows up on the screen:



You can do this with mathematical expressions, too — if you want to see what the Golden Mean actually is, you can plop that arithmetic block into a text block and it will print out the answer to 15 digits.

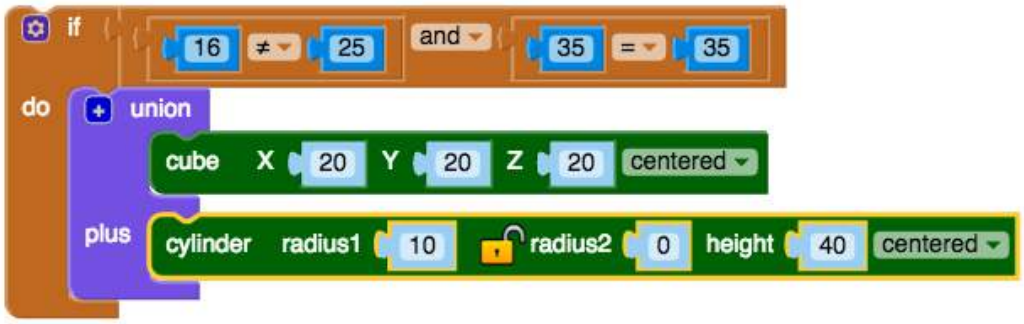
This gives the following value.



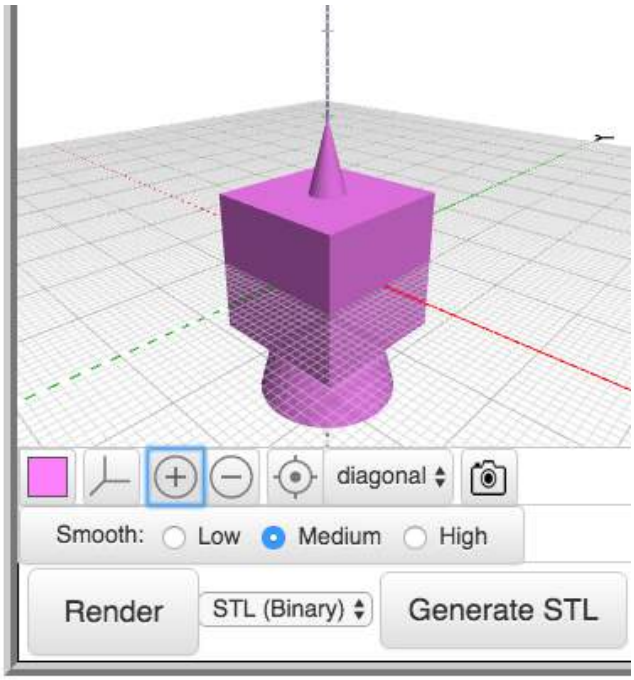
Conditional blocks

The next block performs an action when something is true.

For example, because the following logical statement is true, the 3D object is created,



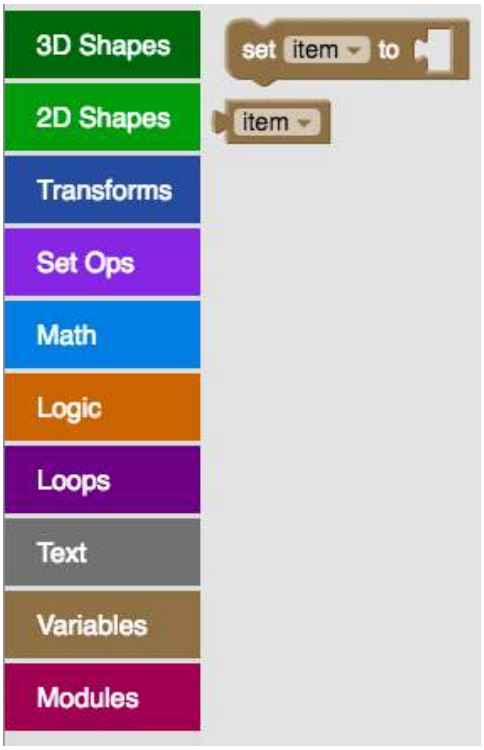
The shape looks like this:



We'll explore the “If block” in more depth in another chapter.

Chapter 7 — Variables

Variables are powerful tools in BlocksCAD for many reasons. In this chapter, we'll explore the use of variables to let us design parametric shapes — shapes that can be modified with the simple change of a numerical parameter. For example, if you're designing an object and aren't sure how big to make it, you can define a single variable that can be changed to give you different sized parts.

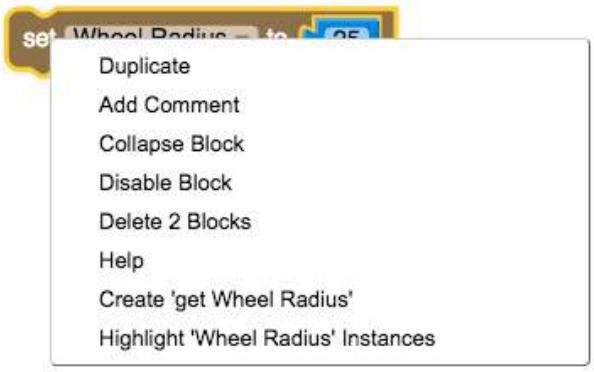


Suppose you want to design some wheels for a project. You may have many projects that use wheels. Instead of designing different wheels for different projects, you can design one wheel and then use a single variable to set the wheel radius.

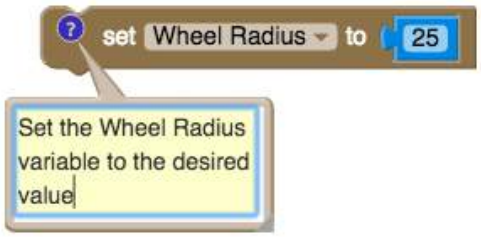
Our wheel will be made of three cylinders. The outer ones are the wheel rims, and the center one is a little bit smaller so the wheel can hold an O-ring used as the tire. Let's start by defining a variable called Wheel Radius.

The radius value is in mm and the variable value can be inserted wherever you would have entered a number.

Another good thing to use with variables is a comment describing the variable in more detail. To create a comment, right-click on the Set block containing the variable name and choose Add Comment.

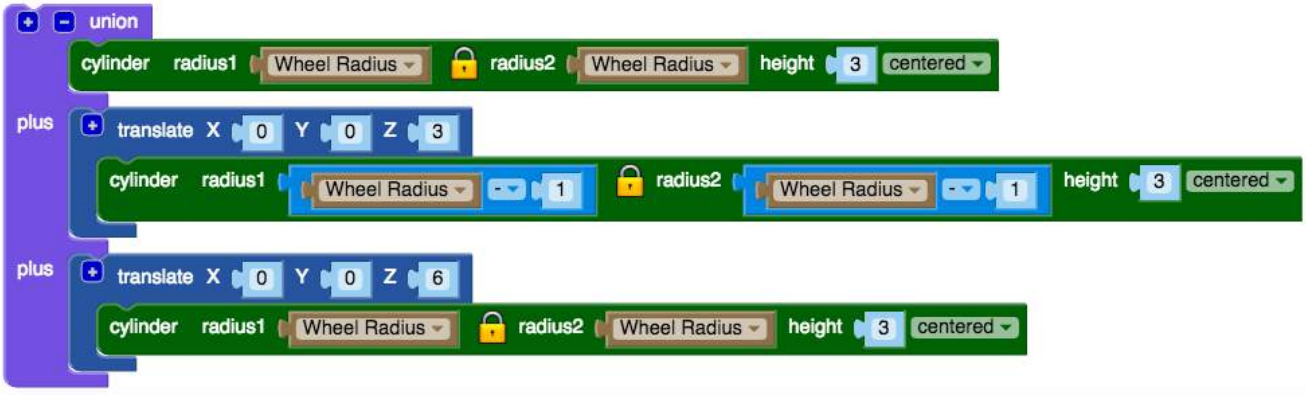


This puts a small question mark at the upper left of the Set block that opens a text window where you can add your comment. While this may not be important if you only have one variable, if you have a lot of them, it helps you keep track of what each variable means. To close the text window, click on the question mark again.



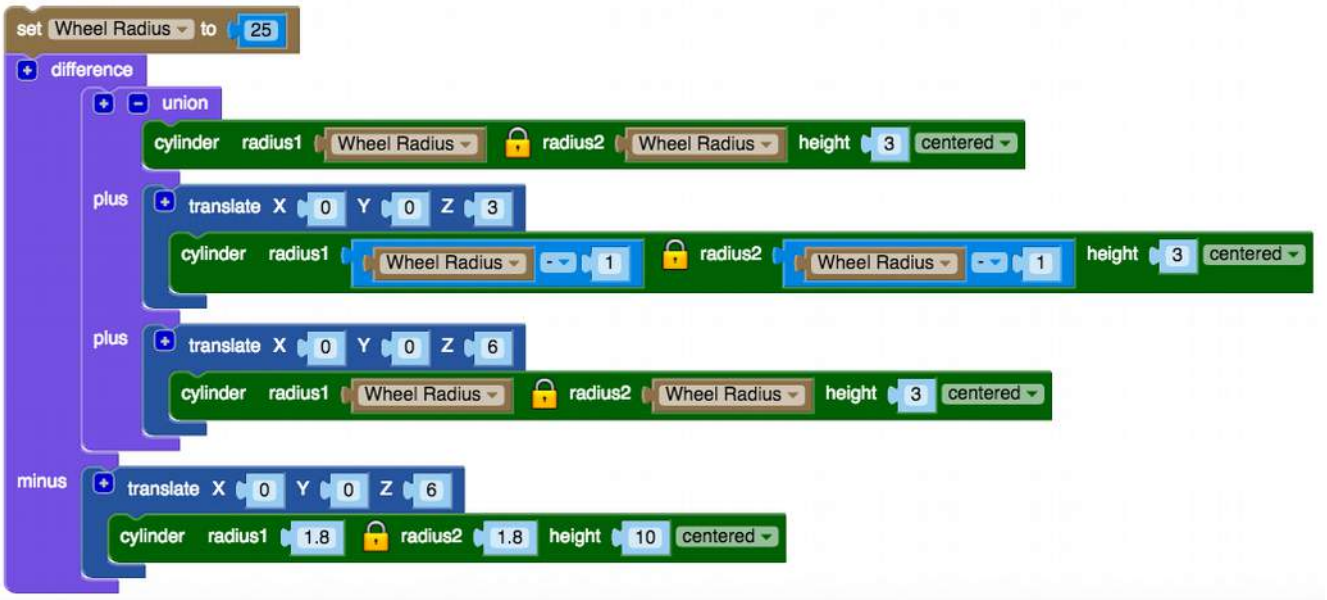
Comments can be added to any block in your design.

Now, for the wheel itself, we add three cylinders together, each of which is 3 mm thick. The middle one is 1 mm smaller in radius than the outer two so that when an O-ring is placed around the wheel to make a tire, it is held in place.

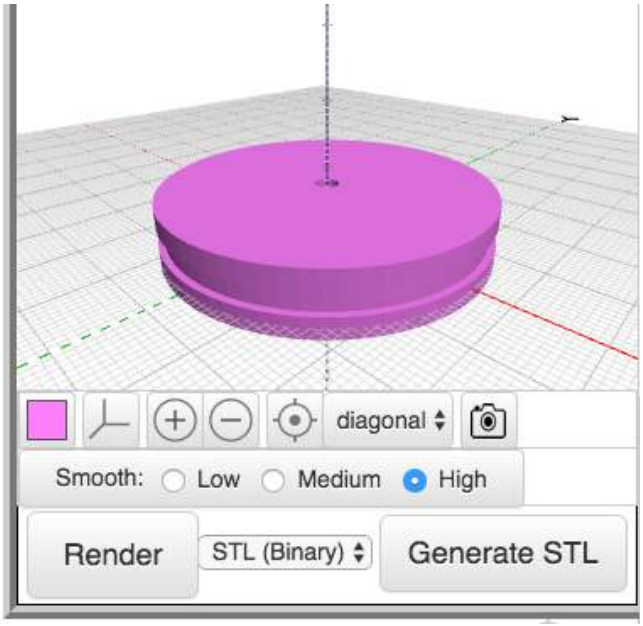


Finally, we use the Difference block to make a hole in the center of the wheel for the axle. I often use wooden skewers for axles. The ones I use have a radius of about 1.9 mm. By making the center hole in the wheel 1.8 mm, the axle will fit in the hole tightly.

You should change the wheel radius to other values and see that, while the overall radius of the wheel changes, the 1 mm rim stays the same to hold the tire in place.



The final design looks like this.



The printed wheel looks like this with the O-ring snapped in place as the tire.

Assortments of O-rings can be purchased from Amazon.

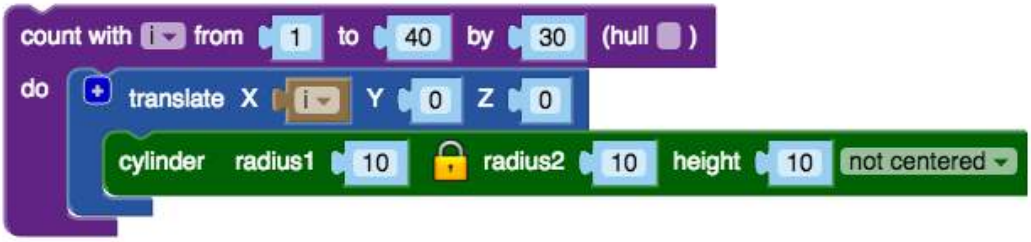
Chapter 8 — Loops

This chapter illustrates how easy it is to have BlocksCAD do a series of operations multiple times. The first of these applies to the situation where we know in advance the number of times we want something to happen (the Count with block) and the second applies when we want to continue doing an operation until a specific value is reached (the If block from the Logic group of blocks). Every decent programming language has these two kinds of operations, although the syntax of the statements might change from one language to another.

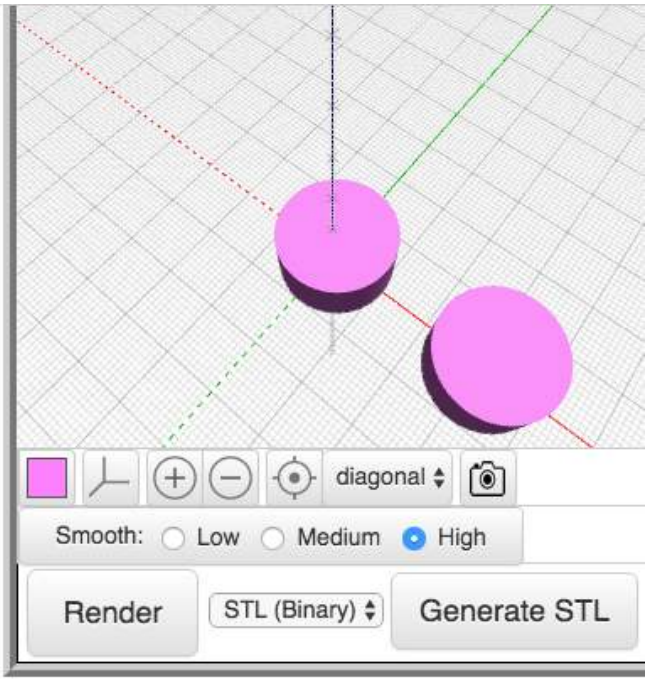
Count with
This block works with a range of numbers and has the following appearance:

This loop has a built-in variable "i". It starts with the value 1, counts by 1, and stops when it reaches the value 10. For each different value of i (1,2,3,...,10) the code inside the loop is run.

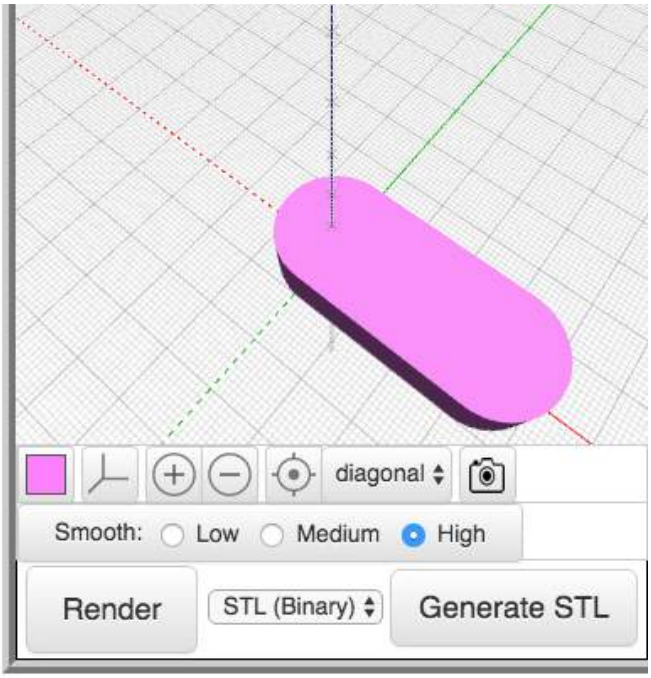
For example, the following loop draws two cylinders:



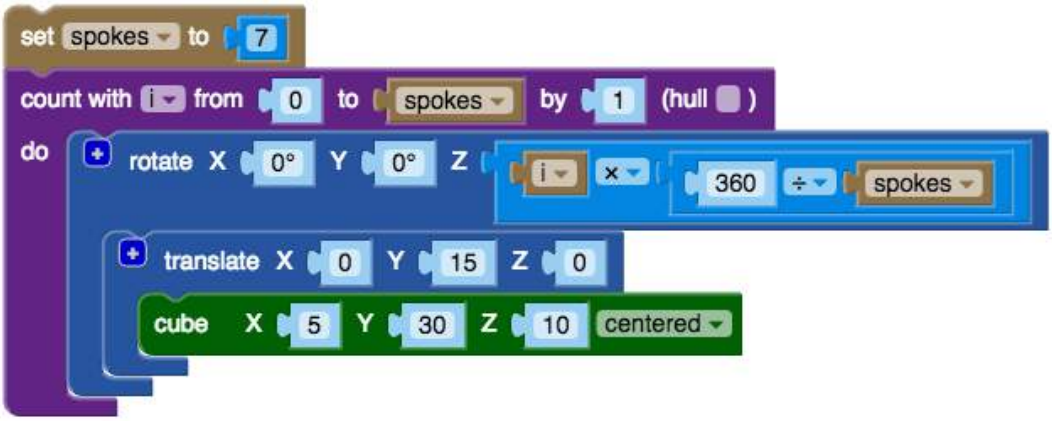
In this loop, “i” will have the values 1 and 31. Here are the shapes produced:



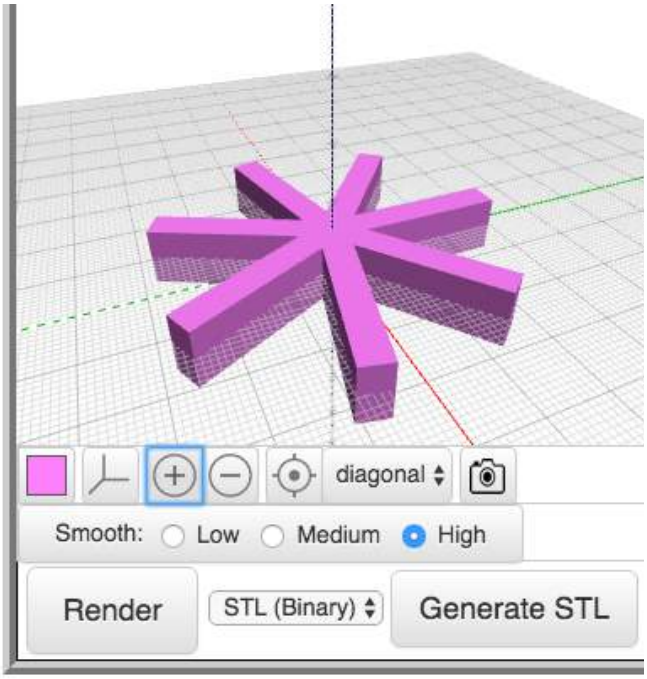
If you check the “hull” box in the block, your final shape will be the convex hull of our two cylinders.



The default increment is 1, and this is the number you are likely to use most for this value. The actual value for application in your object will likely be made using a math command. For example.



Here, we divide a circle into a number of equal sized pieces. The variable "spokes" defines how many pieces to divide the circle into. The angle of these pieces is calculated in the "360 / spokes" math block. The variable "i" counts through each piece, and you can multiply i X (360 / spokes) to get all the different orientations for the spokes. This makes the following shape.



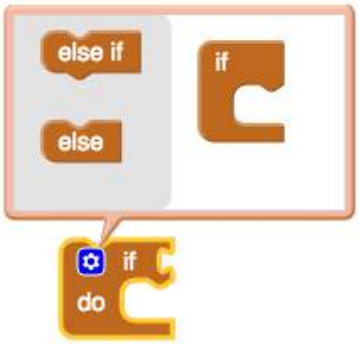
By changing the value of the number of spokes, you can make different arrangements. Notice that this object uses two variables, one for the total number of spokes, and another for the Count with block.

If block

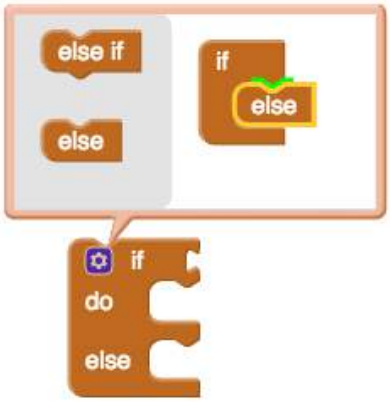
We'll wrap up this chapter with our one remaining logic command, the If block.

This block looks to see if the expression plugged in next to the word “If” is true, in which case it does the commands plugged into the space next to the word “Do.”

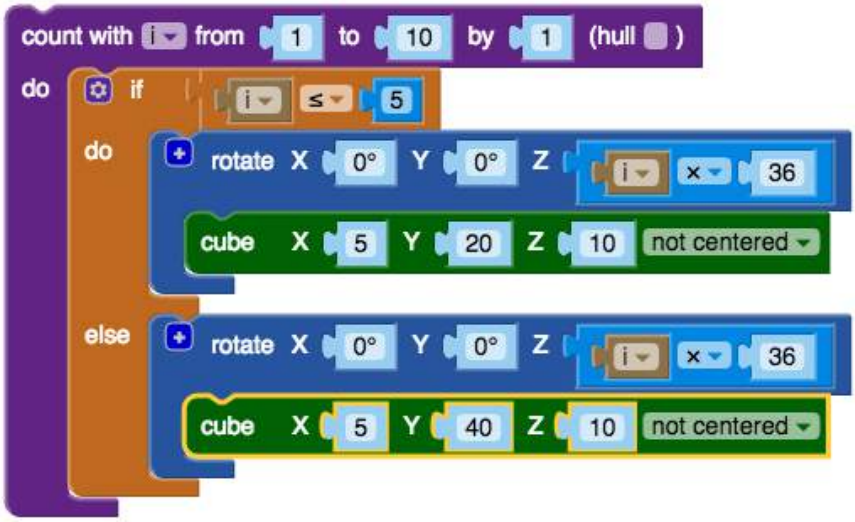
This command has options that can be accessed by clicking on the gear at the upper left of the block.



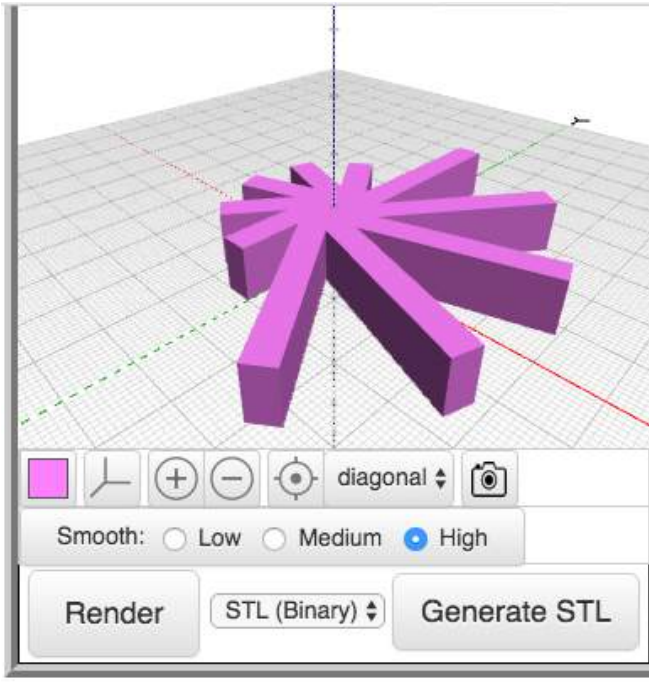
The most common modification is the addition of the Else command created by dragging the Else block inside the If block in the upper window. Once it is in place, click on the gear to collapse the modification window.



Here is a simple example that uses the If-Else configuration of this block.



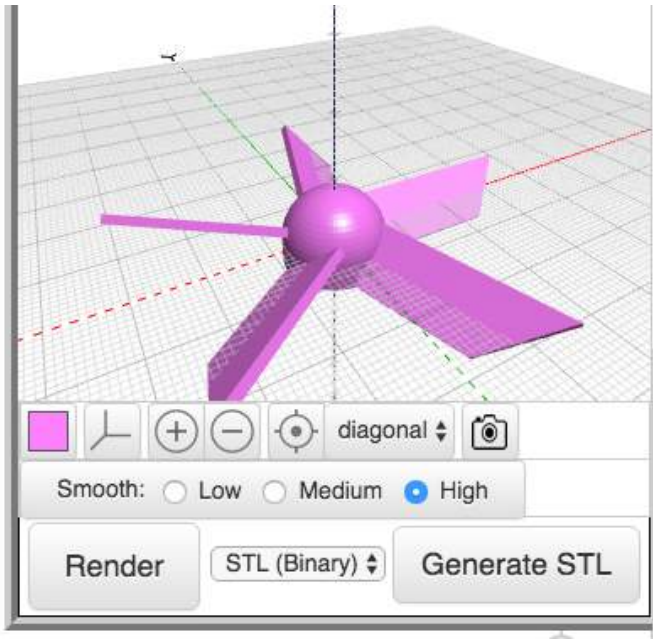
This creates an object with five short spokes and five longer ones.



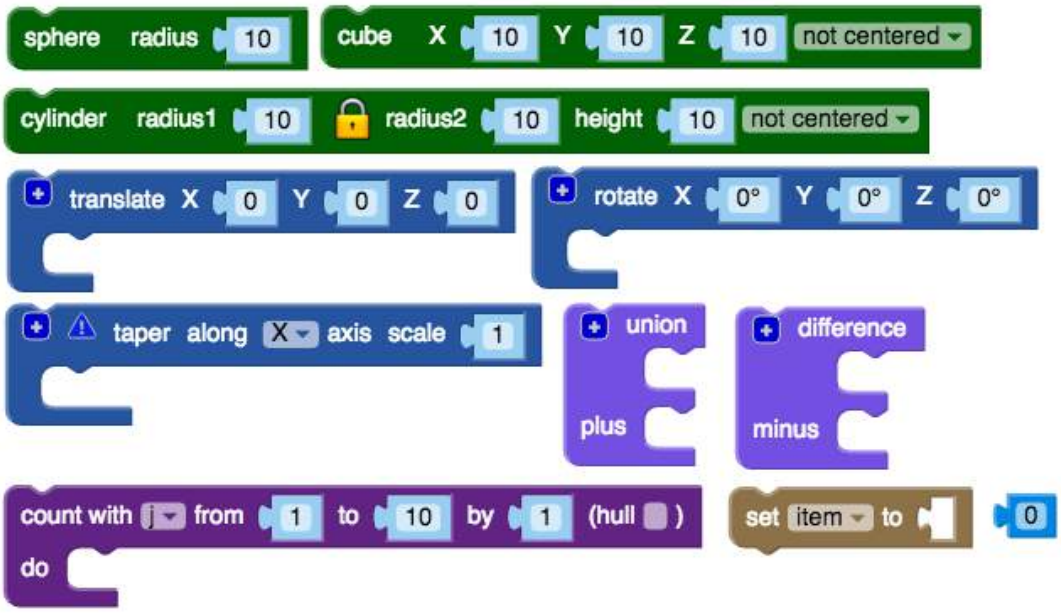
While it may not be obvious now, If-Else commands are important in all programming languages, including BlocksCAD.

Propeller project

For this chapter, you're challenged to design a complex shape — a propeller that can be rendered with as many or as few blades as you wish.



I tapered my blades to make them a bit more efficient, but you are free to modify the design as you choose. Here are the blocks I used. You will likely use some of these blocks more than once.



The main thing I'd like you to focus on is the use of variables and the Count with block to let your design be used to make propellers with different numbers of blades.

If you put your propeller on a motor, you'll want to wear safety goggles to protect your eyes in case it flies off. Also, you might want to place a toroidal ring around the blade to help protect your fingers from a rapidly rotating blade.

Chapter 9 — Text

One of the fairly recent additions to OpenSCAD is the ability to create text, and this feature has been added to BlocksCAD as well.

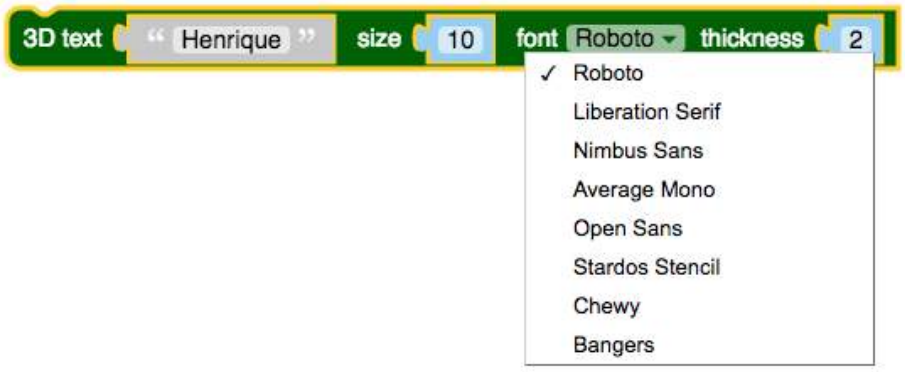
BlocksCAD supports both 2D and 3D text, and the resulting text can be used anywhere you use the original 2D and 3D graphic objects. The 3D text block looks like this:

The text you want to render gets typed into the first placeholder (the one with the letters abc). The next field is the height (size) of a capital letter (H, for example) multiplied by 0.75 to convert the measurement to points, followed by a menu showing a limited choice of fonts, and finished off with the thickness of the text along the Z-axis measured in mm. At this time there is no option for centering the text, so you'll have to do that yourself.

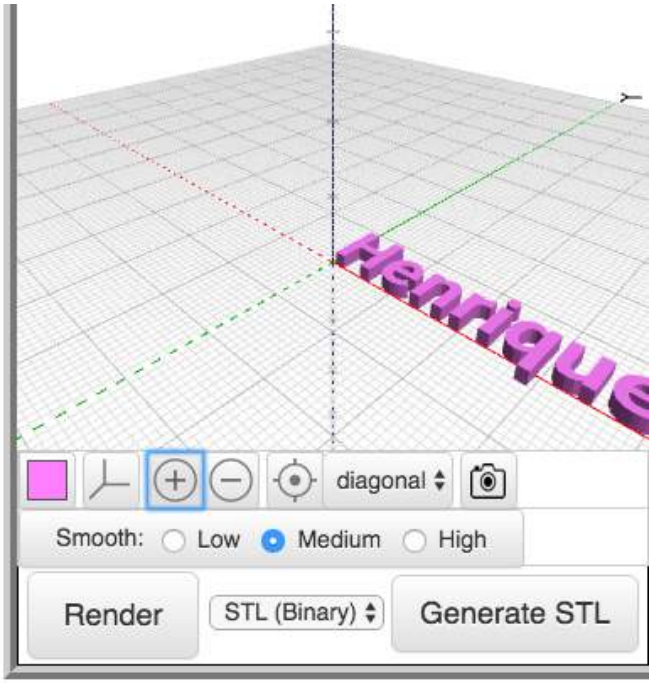
Font size is a very complicated thing. If you put a font size of 10 in lots of different fonts an "H" will likely be a slightly different size in all of them.

As for the fonts themselves, BlocksCAD uses its own set in the cloud since it does not have permission to look at the fonts you have on your computer. This is a web security precaution. It also means you don't have to worry about sharing your designs with others without having to check to see what fonts others have on their computers.

You'll have to tinker with your designs to get the text just the way you want it.



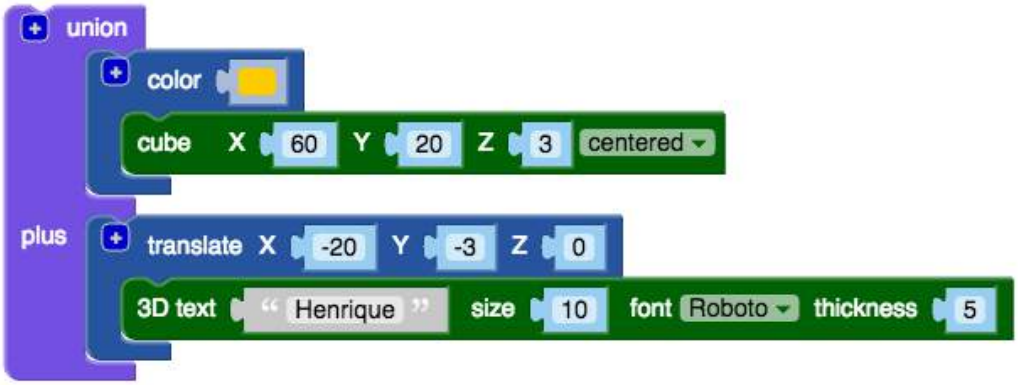
This collection has both serif and sans serif fonts from which you can choose. When you render the text, it looks like this on the screen.



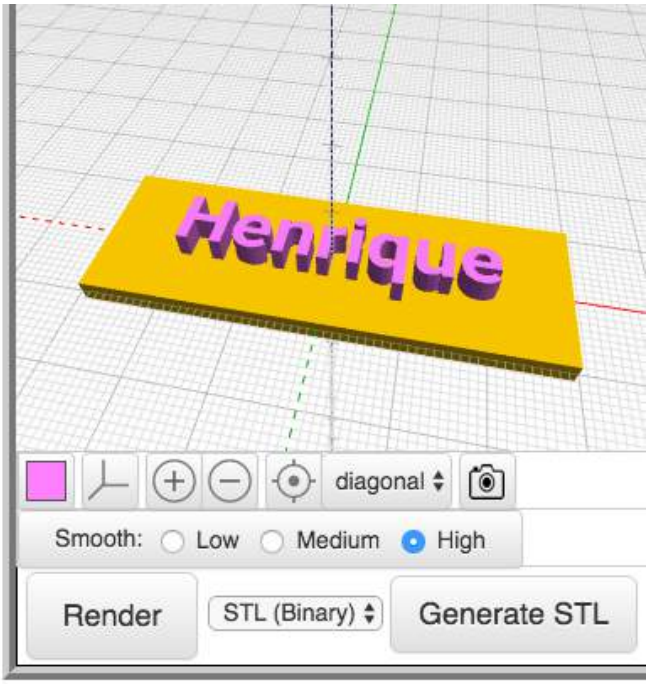
The text is drawn along the X-axis, with the lower left corner of the first character located at the origin.

If you want to use your own fonts, you'll need to export your design to OpenSCAD and change the font to one that is on your computer. Hopefully, the BlocksCAD font library will grow over time to give you more options.

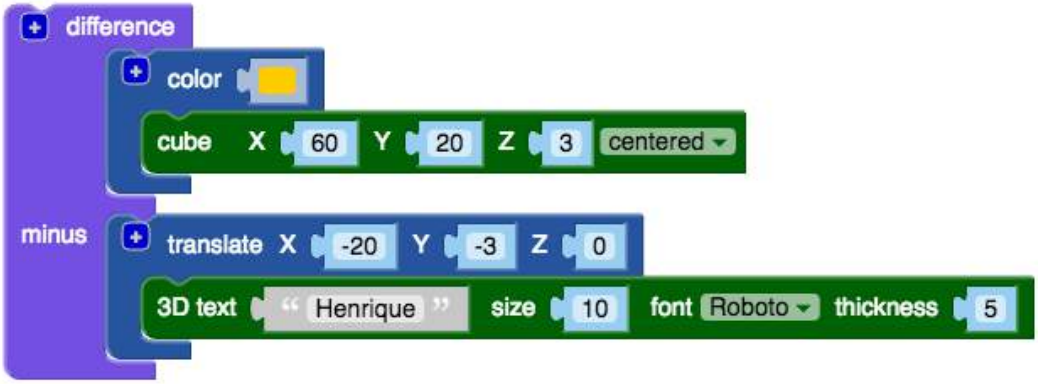
If you render and print your 3D text at this point, you'll end up with a collection of separate letters. Generally, you'll want your text to be embossed or recessed on a surface.



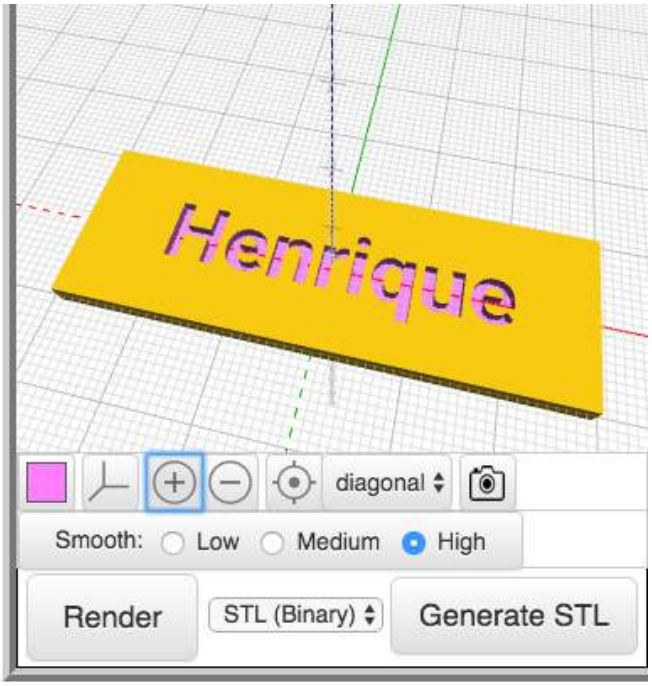
This combination of blocks gives embossed text. I used a different color for the base to make it easy to see. Of course, the color information does nothing with a single-color printer.



If we use the Difference block, our text can be subtracted from the base.

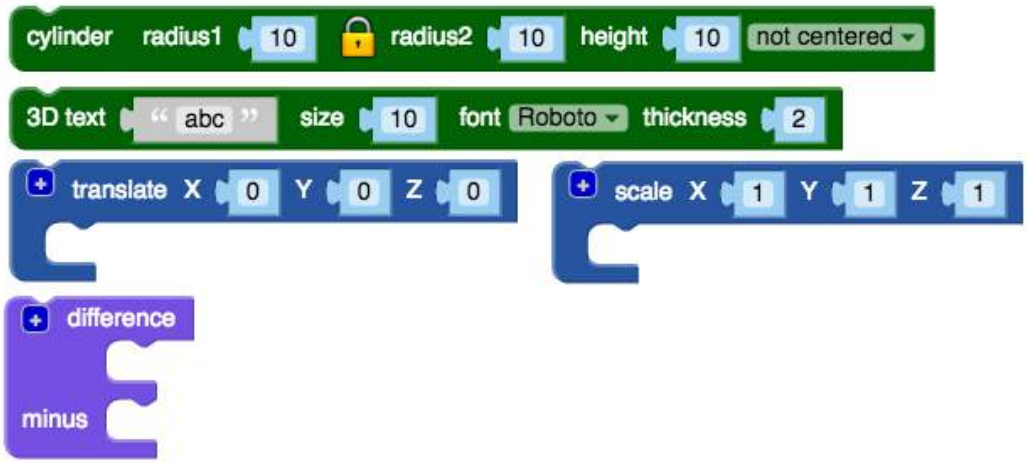


The result looks like this:

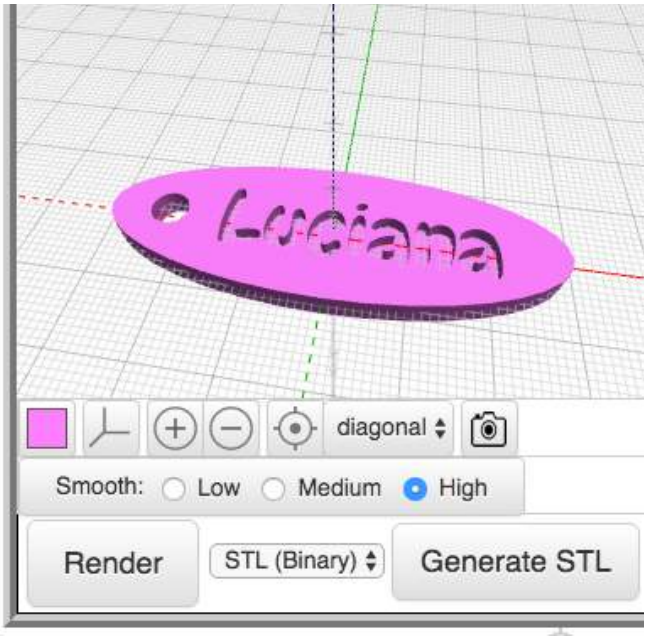


Name tag

The project for this chapter is practical. Use the following blocks to make a name tag that can be used on a backpack or piece of luggage.



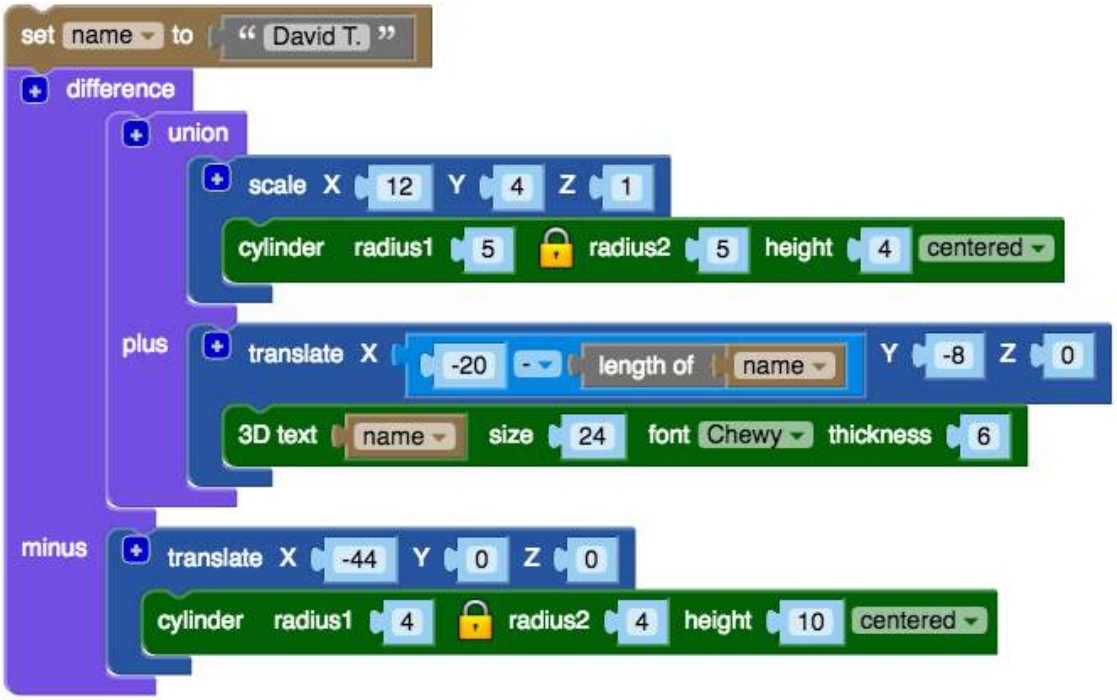
Your tag might look like this:



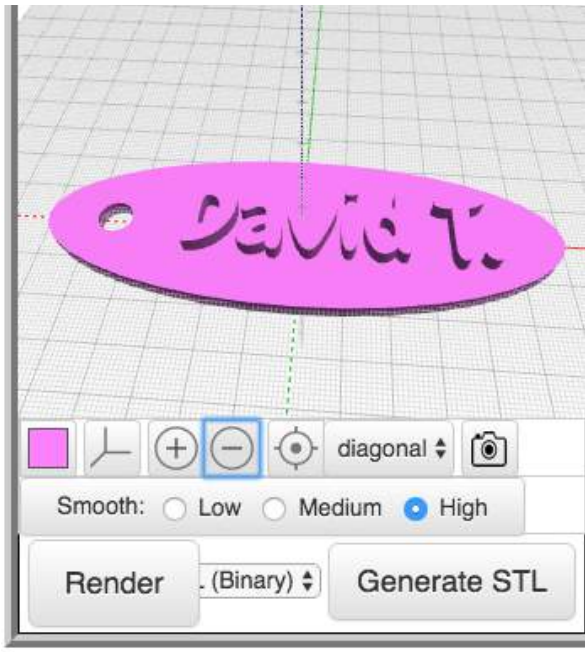
The final tag can be attached to luggage with a plastic cable tie.

Use of variables in text

The text blocks let you work with variables as well, although you may have to tweak your designs that use them. Here's how the types of tags we've made look when using variables.



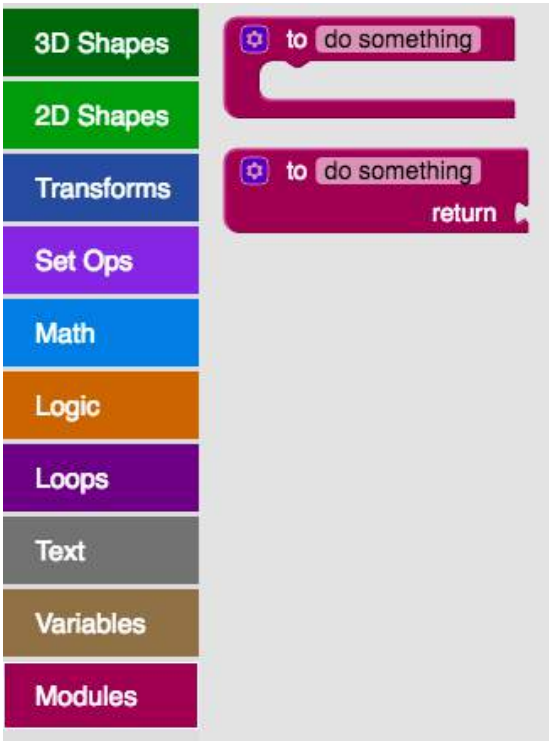
Note in the place where text is added that we offset the text along the X-axis by -20 mm minus the length of the name. The Length block produces a number equal to the number of characters in the text string. This gives us the following image when rendered.



The final print looks like this:

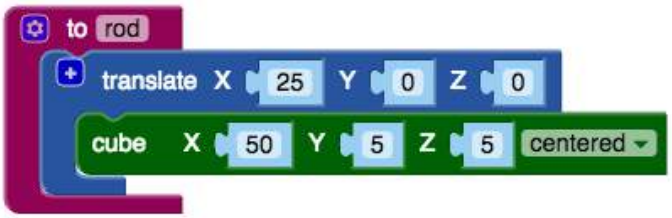
Chapter 10 — Modules

Modules have several uses in BlocksCAD. The most common use is to combine a set of frequently used blocks into a new command that can be redefined as a single block using a name of your choice,

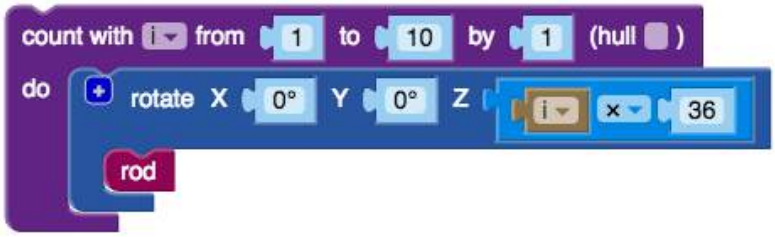


The basic block looks like this:

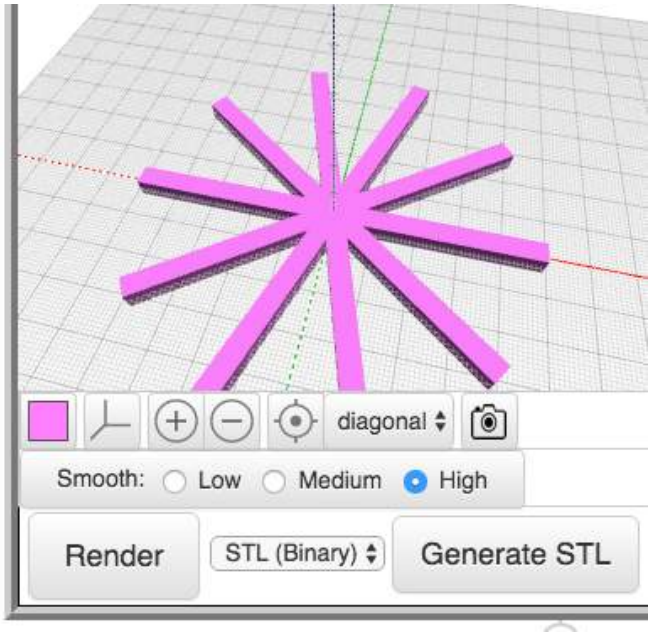
You will replace the words, “do something” with a descriptive word reflecting your new module. For example, while it is so simple that a module is hardly needed, suppose you want to define a simple rod:



Now once a module is defined, you need to use the module's executable block to actually build anything. These new blocks get added to the Module button automatically as modules are defined. So, for example, a set of ten rotated rods would be made this way:



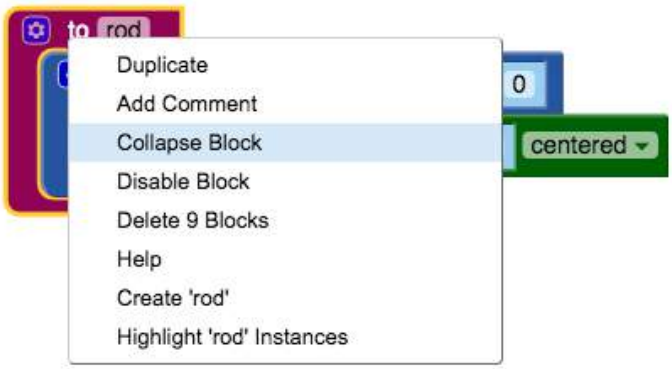
The resulting shape looks like this.



As mentioned, the real power of module definitions comes when they are used to define a block with many elements. You can "label" other blocks of code and collapse them too (not just modules). You make a label by adding a comment to a block. When you collapse the stack of blocks, it looks for comments within the stack of blocks and uses those to label the collapsed block.

Collapsing code makes room. It also encourages programmers to add comments to blocks of code, which makes debugging so much easier because you know which set of code was used to produce which shape in the project.

If you right-click your mouse on the module definition, you'll see a list of options, including one that collapses the block to reduce screen clutter.

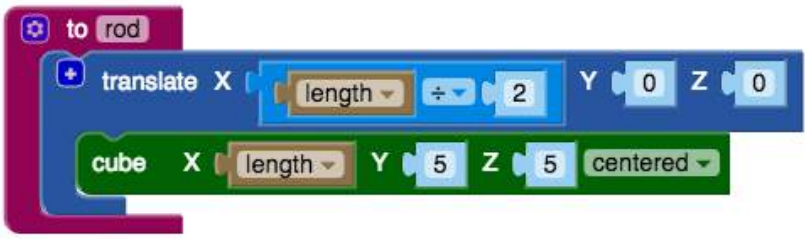


A collapsed block looks like this:

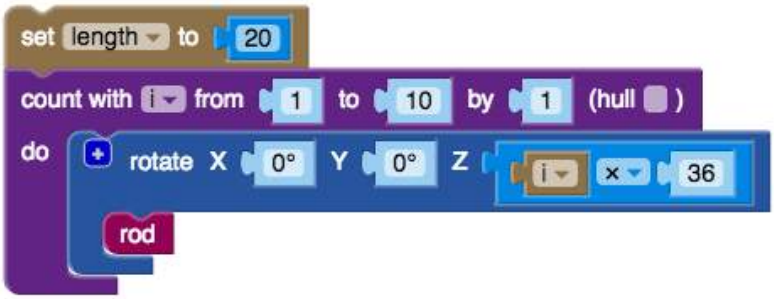
This doesn't take up much screen space, and it can always be expanded (using the right click process) if it needs to be edited.

Using modules with global variables

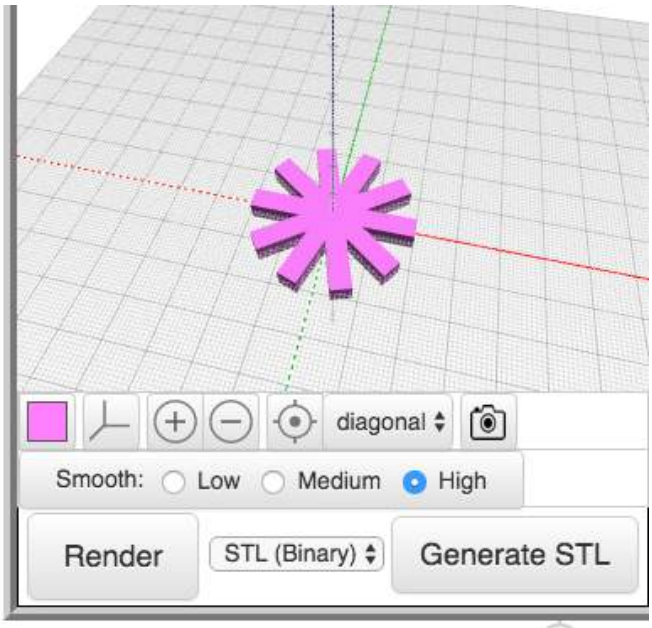
Suppose we wanted to choose the value of our rod length after it was defined. One way to do this is to create a global variable named length, and redefine our module to incorporate it.



In the application area using this module, we start by giving length a value:



This gives us a collection of smaller rods than we had before.

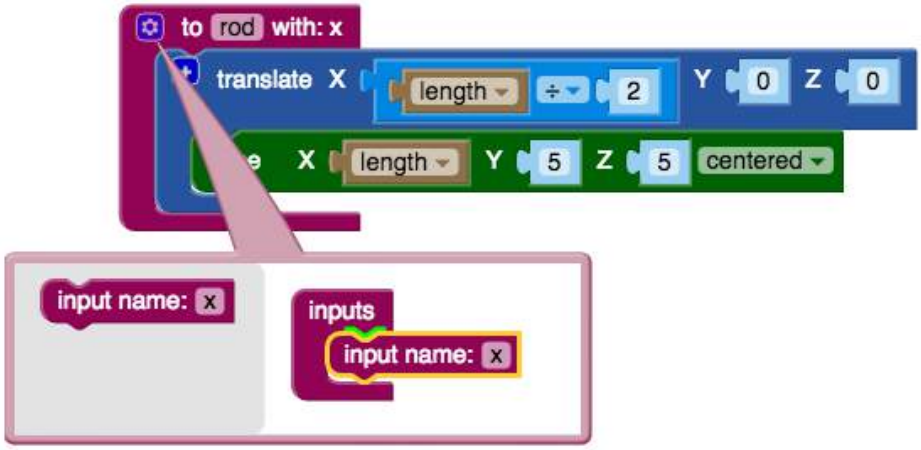


Once set, the values of global variables apply to anywhere they are used in the entire program. If there are several places in your program where you have given different values to the same global variable,

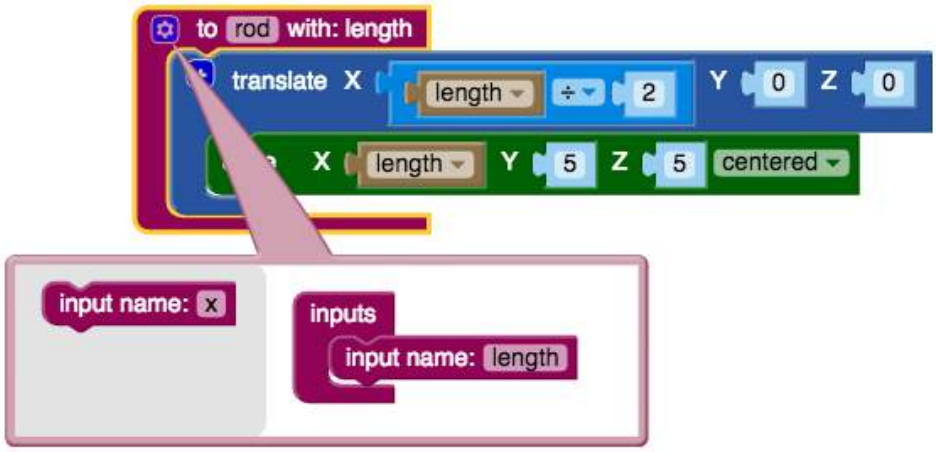
the value closest to the end of the program is the one that is used everywhere. Choosing the value based on physical location comes from the way that BlocksCAD (i.e., OpenSCAD) programs are compiled — details of which are outside the scope of this book.

But what if you want to use the same variable name to apply to different values in different instances of a module? This is done by using local variables.

Local variables
To illustrate this, click on the gear icon at the upper left of the Rod module definition.

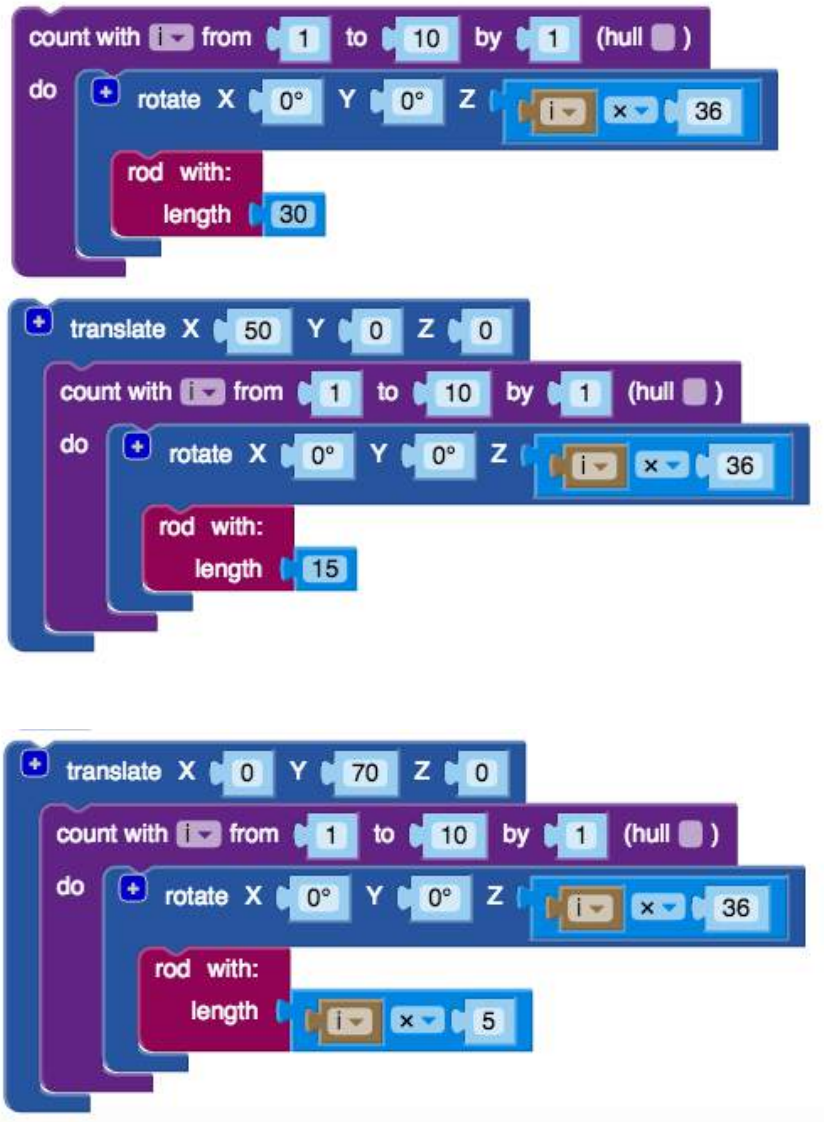


This opens a box with inputs on the right, and a block called Input name on the left with an initial variable named x. Drag this Input name block to the Inputs block and rename x with any name you wish (I used length since we used that for the rod length before).



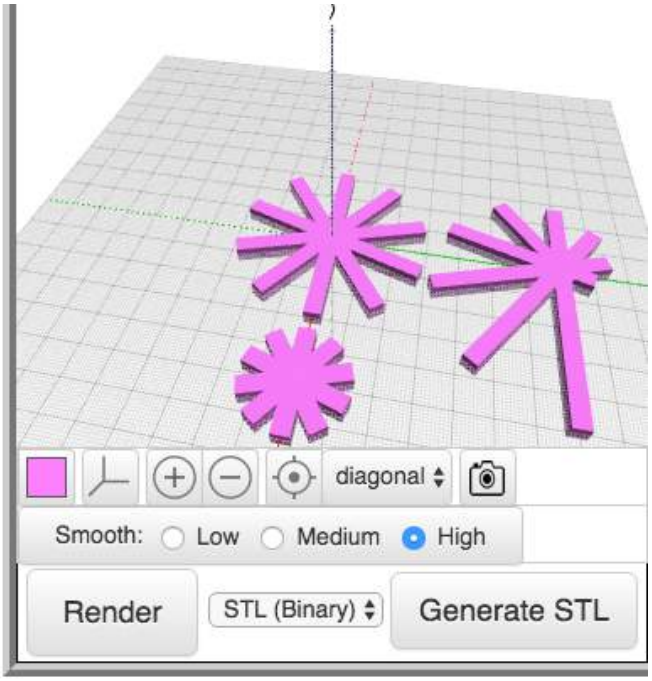
The difference from the previous examples is that the value of length applies only to applications inside the module we defined. Each time we use the same module in our program, we can have different values for the input, which is why it is called a “local variable.”

Here's an example showing how this works:



The third shape uses “i” as a variable as part of the calculation for the rod length, showing that you can use variables, not just numbers, in your modules.

Notice that the Rod block is now called Rod with, followed by any local variables you've defined. As you plug in values for these variables, you can see that these values are only applied inside your defined modules.

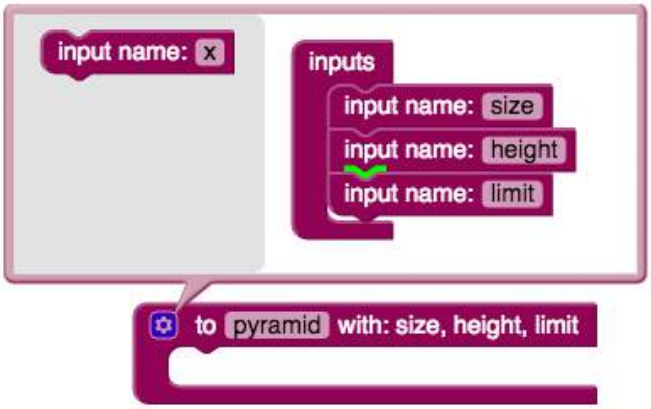


Recursion

We are now ready to introduce one of the most powerful (and tricky) concepts in programming — recursion. A module that uses recursion makes use of the module itself within its definition. This amazingly powerful feature is made easier through the use of local variables, whose values are "passed" to other parts of the module in its definition.

While this topic is quite challenging for the beginning programmer, it lets you create shapes that would be tedious, if not impossible, to make any other way. Understanding recursion is probably one of the harder concepts you'll encounter as you learn coding.

Our first example makes use of a module that draws a box, and then uses itself again to draw a smaller but higher box, and continues this process until the X- and Y-values of the box are less than a certain amount. This gives us a stepped pyramid shape. We'll start with the module definition incorporating three local variables representing the size of the cube, its height, and the limit for our smallest size.

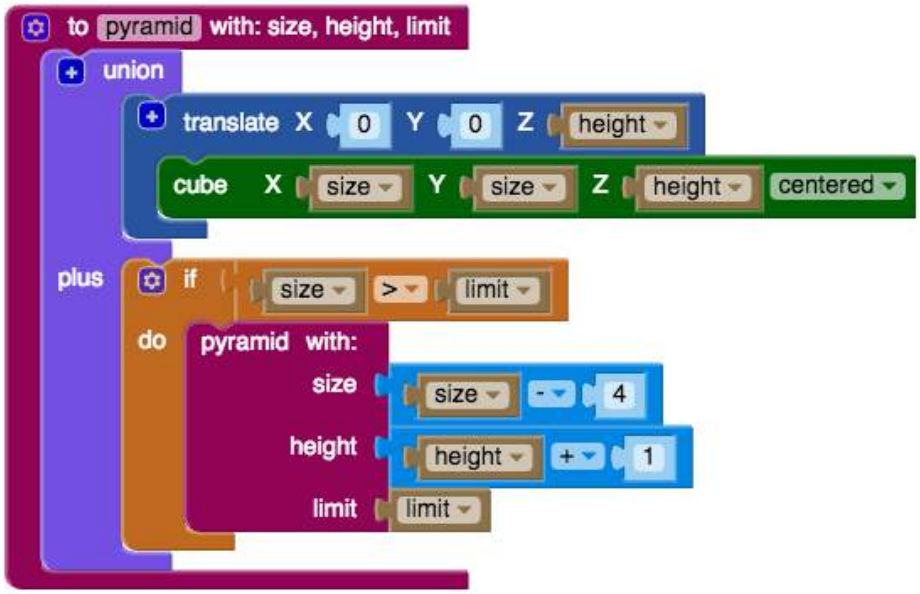


Our first example defines a module that draws a single box, with position and size parameters being sent to the module.

Inside this module, there is an if-statement that asks "Is the size of the box I'm drawing bigger than the limit parameter?" If that is true, then we aren't done, and need to draw another, smaller, higher box. So the module calls itself, with a higher position and a smaller size. The effect is that the module keeps drawing one box, checking to see if there is room for another box, and if so, calls itself to draw another (smaller, higher) box. Once we draw a box that is smaller than the limit parameter, the if statement will return false, and the module stops calling itself (so the pyramid is done).

This example could also easily be done with a loop (can you figure out how?). But the next recursion example will show a shape that is very hard to define in another way. (See Chapter 11.)

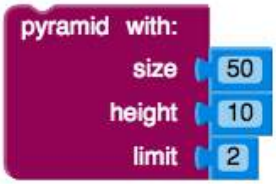
Our first recursive module definition looks like this:



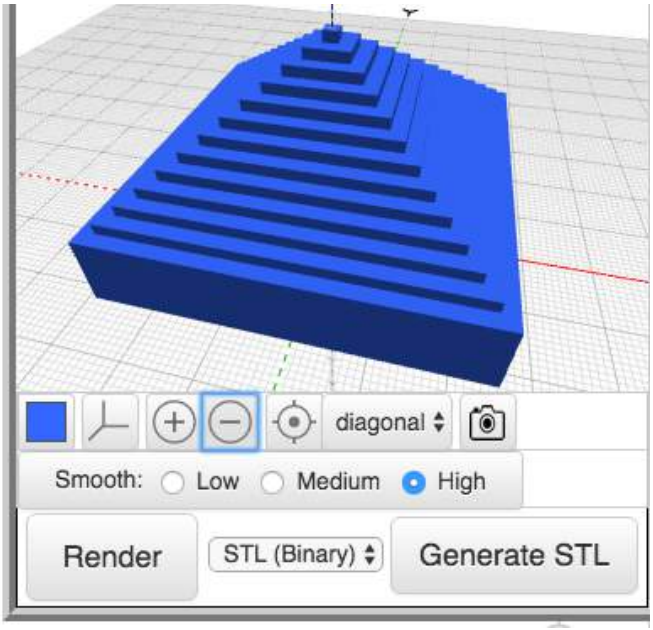
The first part draws our “cube” translated along the Z-axis. Next, we use the If block to check to see if the size is greater than our limit, in which case it executes the module again using modified values for the size and height variables.

The idea of including a module in its own definition may seem bizarre to you, but it is the essence of recursive programming — sort of like picking yourself up by your own bootstraps. While this seems impossible in the physical world, it is often used in programming since it lets you create programs that would be very hard to write otherwise.

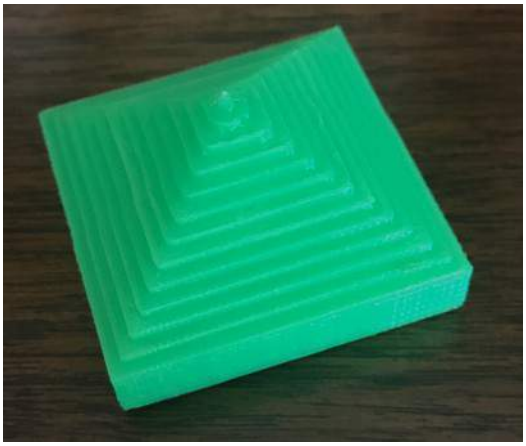
To see your module in action, try this:



This is the shape you'll get when you execute this command.



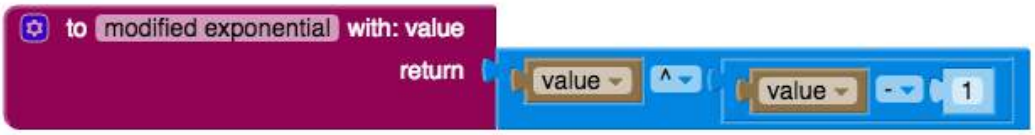
Printed, the shape looks like this:



There is another operation in the Modules area of BlocksCAD. Unlike the one we explored, the second one lets you define mathematical functions. For example, suppose you wanted a function that performs this operation:

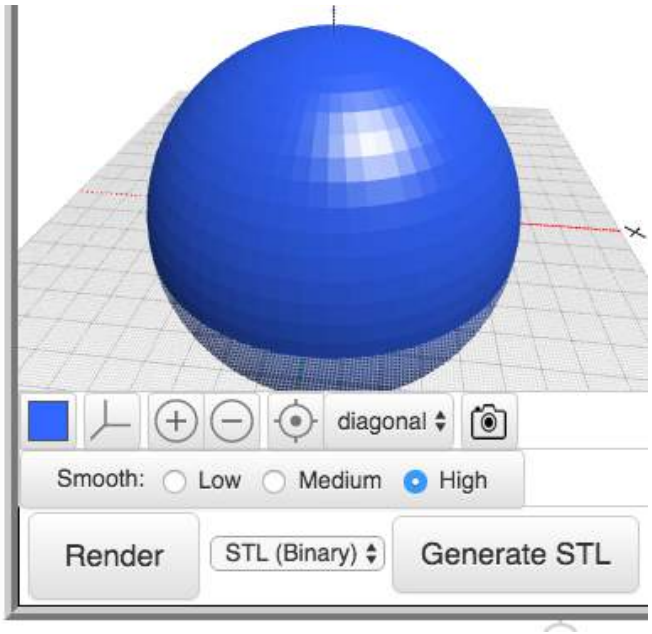
$$value^{(value-1)}$$

Here's what the function would look like in BlocksCAD:

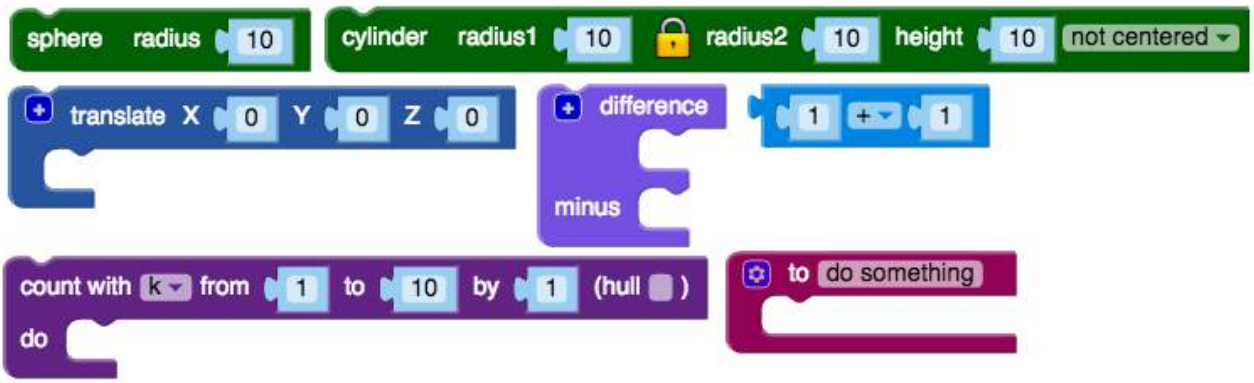


This function can then be used like any other mathematical operation.

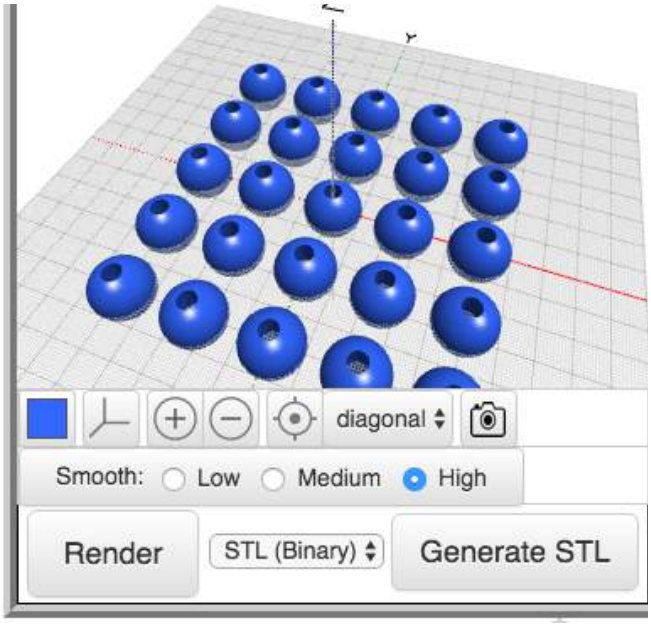
This generates the following image.



Your project
You can make a collection of beads for a bracelet or necklace using the following blocks.



Your beads should have a hole in them for the string. Also, experiment by using a module with a local variable setting the bead radius.


diagonal
Smooth:
Low
Medium
High
Render
STL (Binary)
Generate STL

Chapter 11 — Combining Shapes into Complete Objects

BlocksCAD has even more features than we've explored so far, but the ones we've described should let you design just about anything you'd like to build. In this chapter we'll explore some cool designs to trigger your imagination to make even better ones on your own!

A good starting point for any design project is a paper and pencil sketch of the thing you want to make. There is special isometric graph paper you can use to make it easier to sketch 3D designs, but any kind of sketch will be useful.

The next step is to look at your design to see if there are some parts that are used several times in designing your final project. These parts are likely to make good modules since they can be refined to be sure they look like what you want before incorporating them in the final project. We did this in the previous chapter.

As for writing the program, there are a few things to remember. First, there is no such thing as too many comments. We haven't used comments much because our shapes were easy enough to decode by looking at them. If you collapse your modules, have comments showing how they are used and what they do can be very helpful. Another great use of comments is to explain the default values of variables used in your designs. If one general design is used to create simple variations later on, you should identify and define the parameters (global variables) you will use and put them near the front of the program where they are easy to find and change as needed.

In this chapter we'll show a few projects I designed to show off a few special features. I didn't put any projects in this chapter for you to do because, by this time, you are probably designing and printing all kinds of cool stuff on your own.

We'll start with the most complex project. Once you have this mastered, the rest will look easy.

Recursion revisited
When we introduced recursion in the previous chapter, our example looked like something that could be made with a simple Count with block. In many languages, this is also called a For...Next command. By using the value of the index (the counting parameter), you can modify the shapes of the things you are creating.

Good programming languages (like BlocksCAD) support recursive programming. Instead of looping a single set of commands, a recursive module uses replicas of itself as a part of the module definition. Once a module is run, the values of variables are local to that instance of the module. The result is the ability to create complex objects that can't be made with simple loop instructions.

In celebration of this capability, we'll create a famous recursive structure, a fractal tree. Fractals are complex structures in which any part of the structure is a replica of the structure as a whole. An entire branch of mathematics (chaos and complexity theory) is devoted to the exploration of these structures.

Consider a real tree, for example. The branches are similar to each other, but their size changes as smaller branches grow out of larger ones.

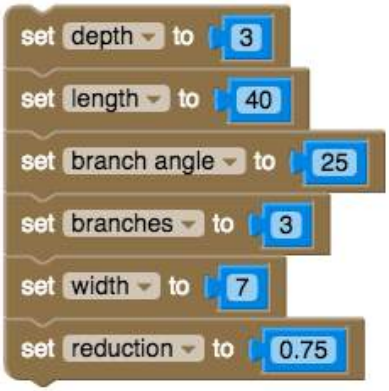
A similar pattern is found in ferns where the whole structure is similar to the smaller parts that make up the plant.

I became so engaged by these structures that I wrote a books on the topic back in the 1980's: *Discovering Apple Logo : An Invitation to the Art and Pattern of Nature*, Addison Wesley, 1983).

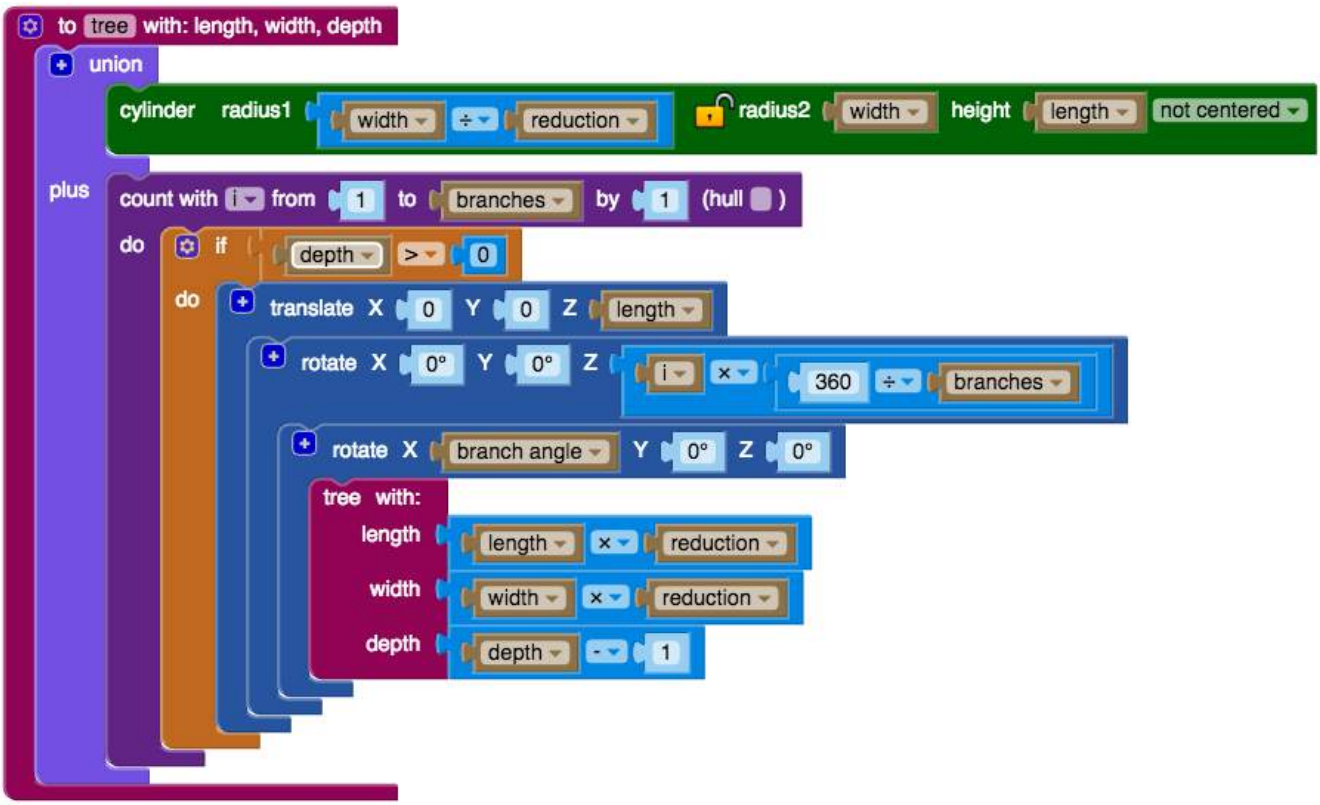
The language I wrote about, Logo, supported recursion, so it was a natural choice. Of course, in those days, the graphics were low resolution 2D images on an Apple II computer — a far cry from what we can do with BlocksCAD and a 3D printer today.

Building our tree

As our tree is being built, the size and length of branches will change, just as they do with real trees. We start by creating a set of global variables used in our structure.



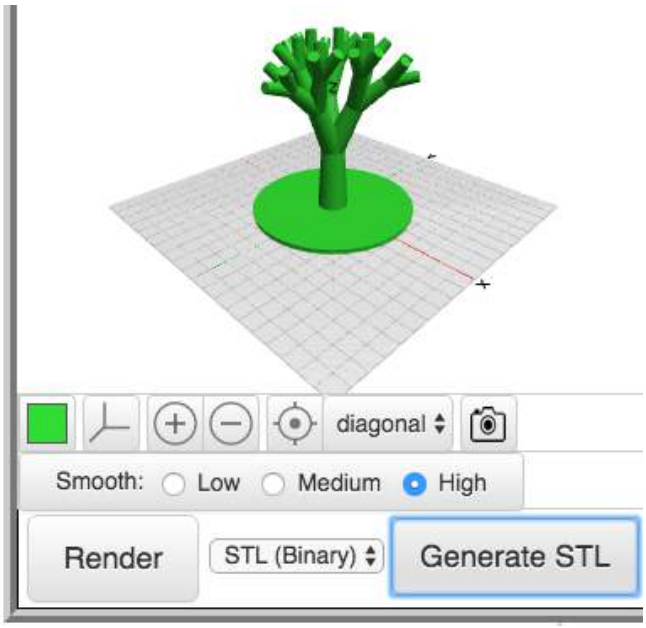
Next we define a recursive module called “tree”. This module uses three local variables — the starting branch length and width and the depth (number of levels) of the tree. These three variables will have their local values changed during use of the module. All other variables will keep their global values.



There are two things to notice in this module. First, you'll see that we don't use the Centered option for the cylinder. This makes it easy to get all the branches connected. Second, the cylinder is tapered from its previous radius at the bottom to the new one at the top — just like the branches of many real trees.

Once this module has been created, we can create our final instructions.

When this BlocksCAD program is run, it produces the following tree. I changed the rendering color by clicking on the Render window box with the color swatch and choosing green as the new color since it looks more treelike. Of course the color of your printed object will be determined by the filament you use.



Here's our printed tree.

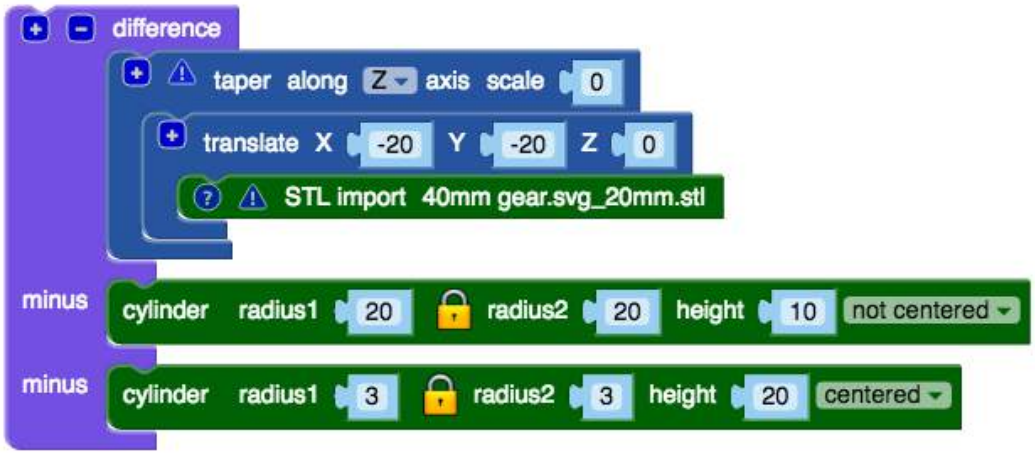
This tree also looks like a stalk of broccoli. You should do some research to see why this is.

Beveled gears

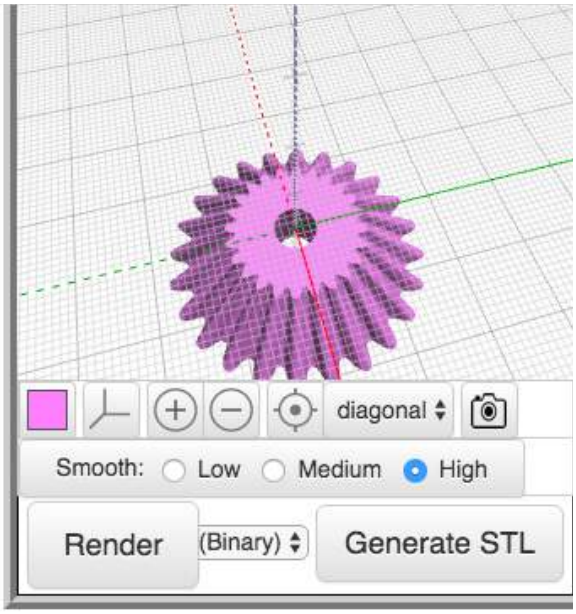
Earlier, we gave you a project to use Inkscape to create gears that can be extruded with BlocksCAD. Sometimes, you may want to have a pair of gears that mesh at 90 degrees to each other, so turning one gear causes the other one to turn along another axis. Here's how to do this.

In Inkscape, choose Gear... from Render under the Extensions menu. This brings up a window where you can set some of the elements of your gear. We'll just stay with the default gear with 24 teeth, although you can change this based on your need. In Inkscape, set the diameter of the gear to 40 mm and use the SVG2STL website to make a gear 20 mm thick.

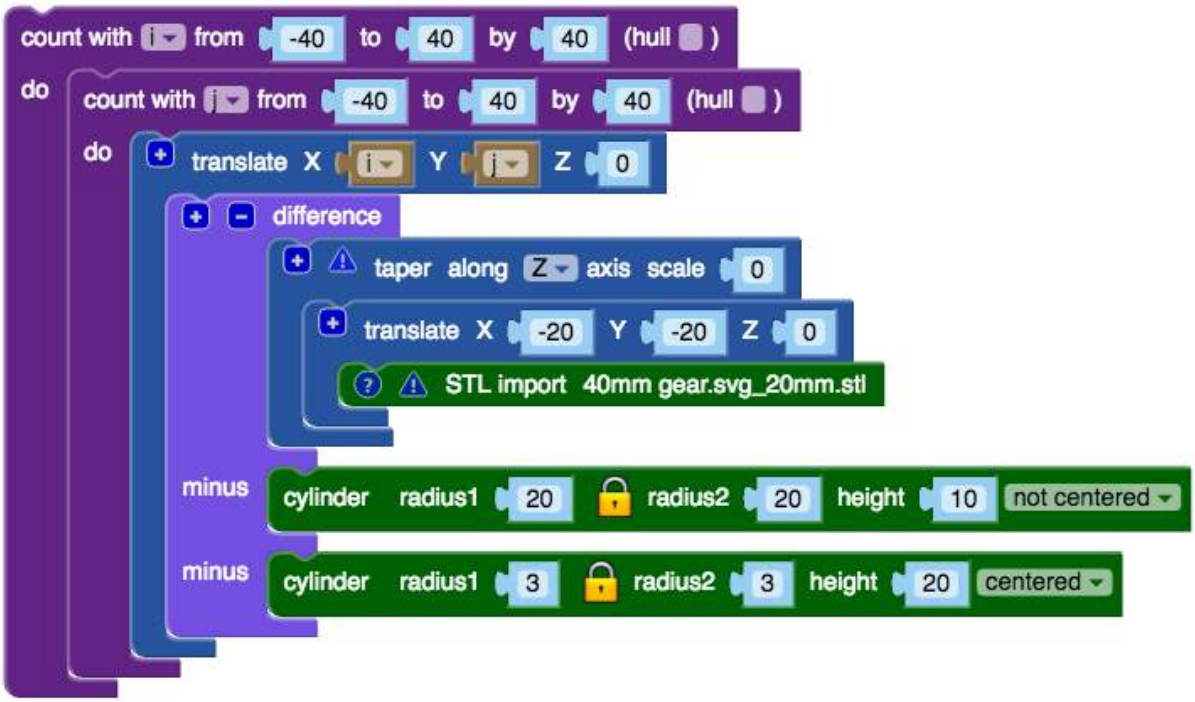
The following BlocksCAD program draws a gear with a bevel of 45 degrees and a hole in the center. You'll want to set the hole diameter to a value of your choosing. Because we are using the Taper along block, it is important that the gear is centered at the origin, which is done with the Translate block.



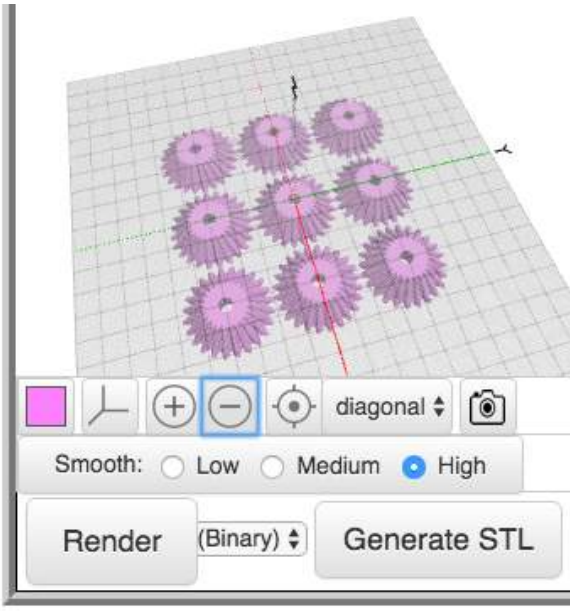
The final gear looks likes this.



You can use the Count with block to set up a collection of gears to print at the same time.



Here is the array of gears ready to print.



Here's how two meshed gears look.

Cable holder

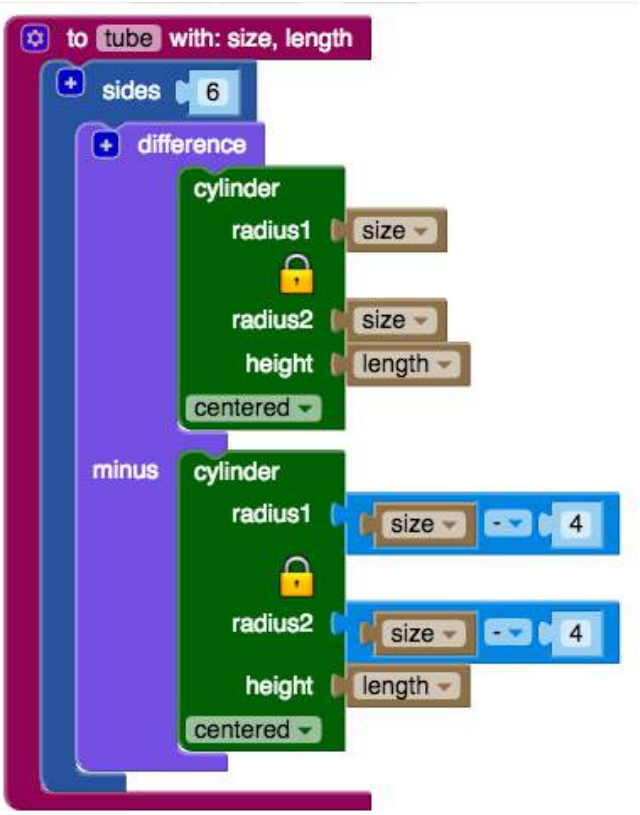
If you're like me, you have a bunch of USB cables lying around for your tablets, smartphones, and other devices. These cables get tangled together and are also easily misplaced. This demo project makes a pretty cable holder you can put on your desk to keep your cables neatly stored. The project starts with two variables representing the size and length of each hexagonal tube. Think of the size as being more like a radius than as a diameter.

The module defining each tube is pretty easy to design. I used hollow cylinders for the tubes and by setting the number of sides to 6, these cylinders became hexagonal tubes. This is a neat trick since the Sides block can be used with cylinders to create any regular polygon.

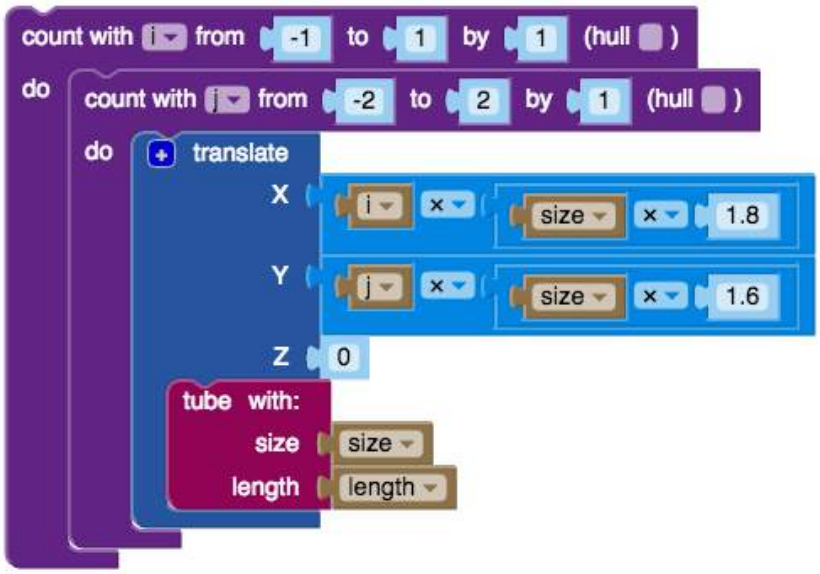
To keep the code horizontally compact, I have the inputs lined up vertically, not horizontally. This makes

the code a bit easier to read without having to scroll the page horizontally. To make this change, right-click your mouse on the Cylinder block and choose External Inputs from the menu that pops up.

You can switch back to Inline Inputs any time you want through the same process. Here's what the final module looks like.

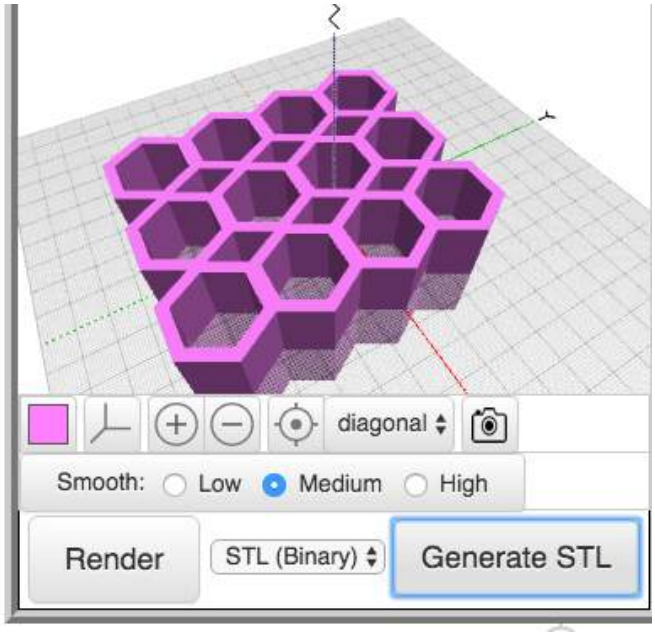


Once the module is defined, we can use it to make our final shape.

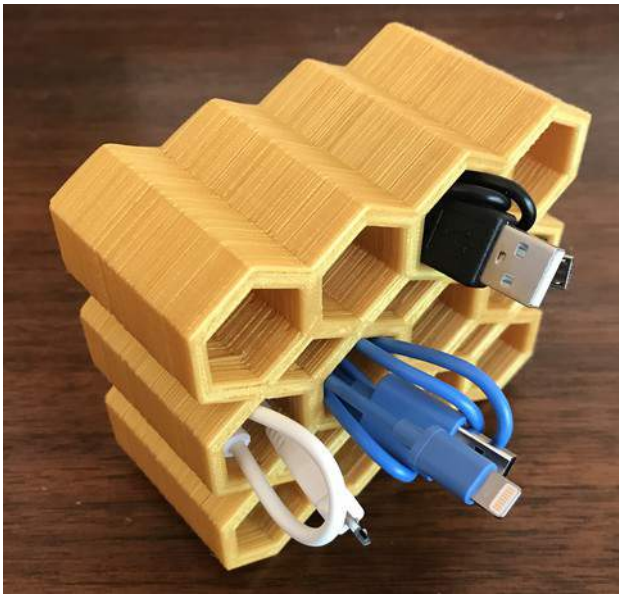


As you see, we used the External Inputs option with the Translate block just to keep the code horizontally compact. You don't need to do any of this — the program works the same no matter how the inputs are displayed.

As for the design itself, I chose the shape of this holder because I find it pleasant to look at.



Your final object is ready to print!



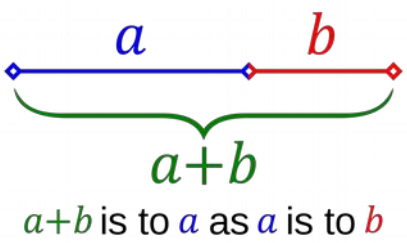
Phylotaxis

First, a word of caution. This project is for those willing to be enchanted by one of the most beautiful concepts in mathematics. If that is not your interest right now, feel free to move on. You can come back to this activity any time you wish!

Pineapples, pine cones, sunflower seed pods and other plant-based structures have what appears to be a set of two intersecting spirals.

In botany, this kind of structure is called phylotaxis. This project builds a 3D model of this structure that lets us explore a bit of the mathematics behind the shape.

The cornerstone of the math we'll use is the Golden Mean, a number that shows up often in nature. This number is defined by dividing a line into two segments. The ratio of the entire line length to that of the longer segment equals the ratio of the longer segment to the shorter one. Mathematically, consider the line shown below:



The Golden Mean (represented by the Greek letter *phi*) is defined by

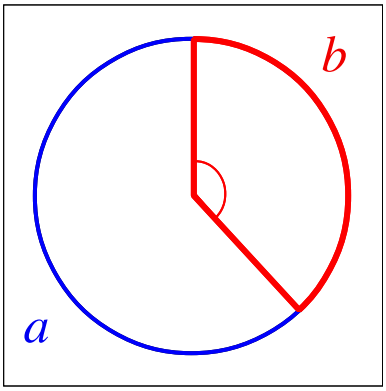
$$\frac{a+b}{a}=\frac{a}{b}$$

The solution to this quadratic equation yields:

$$\phi=\frac{1+\sqrt{5}}{2}\approx 1.6180339887\ldots$$

The golden mean has been known since antiquity and, in addition to nature, it appears in everything from ancient temples to corporate logos.

In our case, we'll explore a related number — the Golden Angle. This is found when a circle is divided into two arcs so the ratio of the circumference to the longer arc equals the ratio of the longer arc to the shorter one. The Golden Angle is the angle subtended by the shorter arc (*b*).

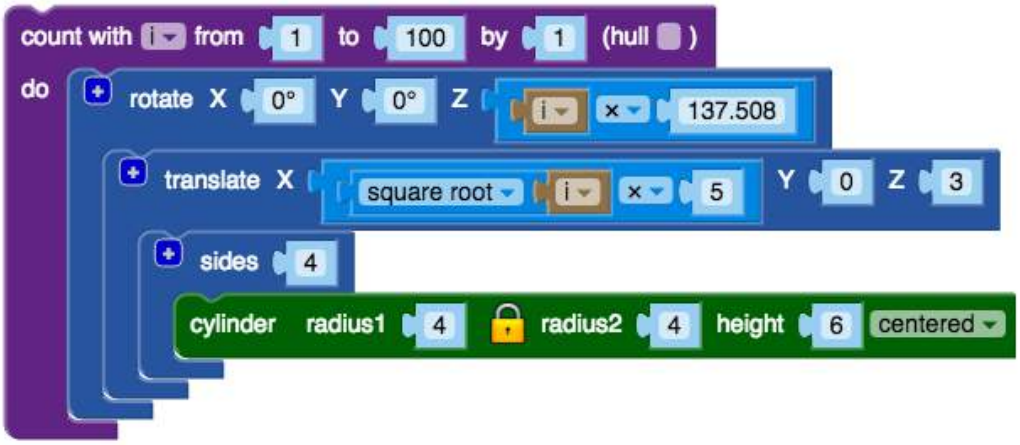


$$360\left(1-\tfrac{1}{\phi}\right)\approx 137.508^{o}$$

Our next task is to create a BlocksCAD program to make a bunch of “sunflower seeds” as they might appear in a flower. This angle will play a critical role!

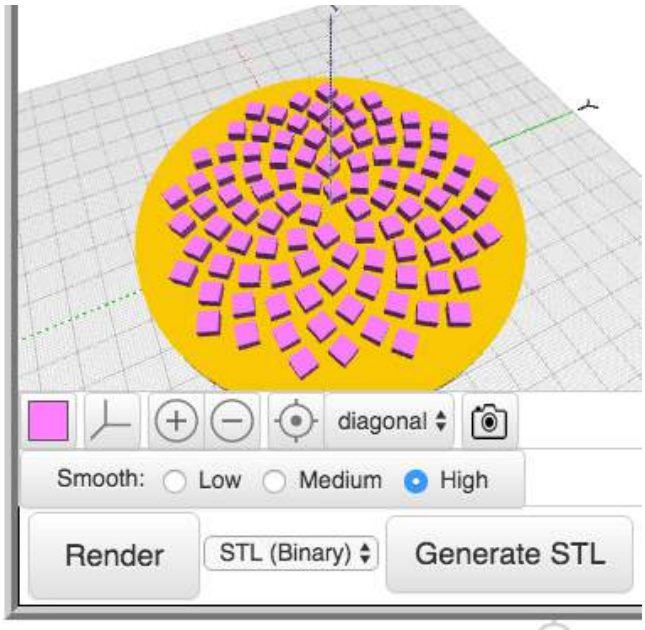
Start by making the base to hold the seeds. I did this with a 5 mm thick disk 60 mm in radius and colored it yellow to contrast with the seeds themselves.

Next, we make the seed array in a different color.



This instruction set puts 100 seeds on the disk. Each seed is made from a quadrilateral using a cylinder with four sides. Each seed is positioned along the X-axis by the square root of the count value (i) so they will be pretty evenly spaced along the radius of the disk. This number is multiplied by 5 to keep the seeds from overlapping. The translated seed is then rotated around the Z-axis by our Golden Angle times the index value.

The result looks a lot like the seed pod for a sunflower.



We just created one spiral in our design, but there appear to be two sets of spirals in our model, one twisting clockwise, and one twisting counter clockwise.

And this is where life gets even more interesting. If you count from the end of the spirals, you'll see that there are 21 counter clockwise spirals, and 13 clockwise ones. What makes this interesting is that 13 and 21 are examples of Fibonacci numbers, named after an Italian mathematician who discovered this series where each successive number is the sum of the previous two numbers in the series.

Fibonacci (also known as Leonardo of Pisa, or by his family name, Leonardo Bonacci) travelled to the Arab world with his father and studied mathematics while there. He popularized the Arabic number system in Europe — the system we use today — through a book he wrote in 1202 (*Liber Abaci).* He became known as the most talented European mathematician of the Middle Ages.

The number series that bears his name (Fibonacci numbers) was described in an example in his book, where he applied it to the propagation rate of rabbits. While the series was known before his time, his work popularized it because it had strong connection to living organisms. As already mentioned, each number in the series is the sum of the previous two numbers, starting with the numbers 1 and 1.

$$F_{n+1}=F_n+F_{n-1}$$

$$F_0=F_1=1$$

The series starts out with 1, 1, 2, 3, 5, 8, 13, 21, and so on. There is nothing in our BlocksCAD code that explicitly uses these numbers, yet they show up anyway. And (here's the next gem), if you look at the ratio of successive Fibonacci numbers, they turn up an interesting result. For example, 21/13 is 1.61538... a number that should look a bit familiar. In fact, for sufficiently large Fibonacci numbers, you'll discover a ratio equal to the Golden Mean.

Even though Fibonacci himself certainly knew about the Golden Ratio, he failed to make the connection between his numbers and the Golden Mean. It wasn't until Johannes Kepler explored these numbers in 1611 that a proof was offered that the ratio of successive Fibonacci numbers approached the Golden Mean. As Kepler said,

> *Geometry has two great treasures; one is the Theorem of Pythagoras; the other, the division of a line into extreme and mean ratio. The first we may compare to a measure of gold, the second we may name a precious jewel.*
> — Johannes Kepler

The proof is fairly simple. The goal is to show that:

$$\lim_{n\to\infty}\frac{F_n}{F_{n-1}}=\phi$$

Since the ratio approaches a constant, then it is also the case that (for sufficiently large values of *n*):

$$\frac{F_n}{F_{n-1}}\approx\frac{F_{n-1}}{F_{n-2}}$$

Since

$$F_n=F_{n-1}+F_{n-2}$$

then

$$\frac{F_{n-1}+F_{n-2}}{F_{n-1}}\approx\frac{F_{n-1}}{F_{n-2}}$$

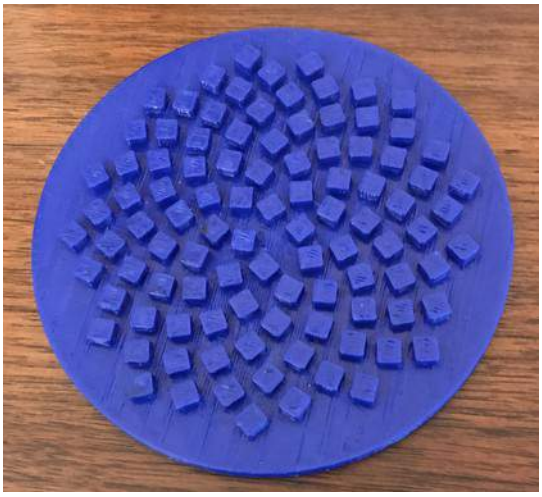
Does this look familiar? Remember that the definition of the Golden Mean is:

$$\frac{a+b}{a}=\frac{a}{b}$$

For large values of *n* these two ratios are nearly the same, proving that the ratio of two successive Fibonacci numbers approaches the Golden Mean as *n* approaches ∞.

Now *phi* is a number that shows up in the Golden Angle we used in our model, so the connection is there at some level. What is amazing is how common the intersecting spirals are in plants of all kinds.

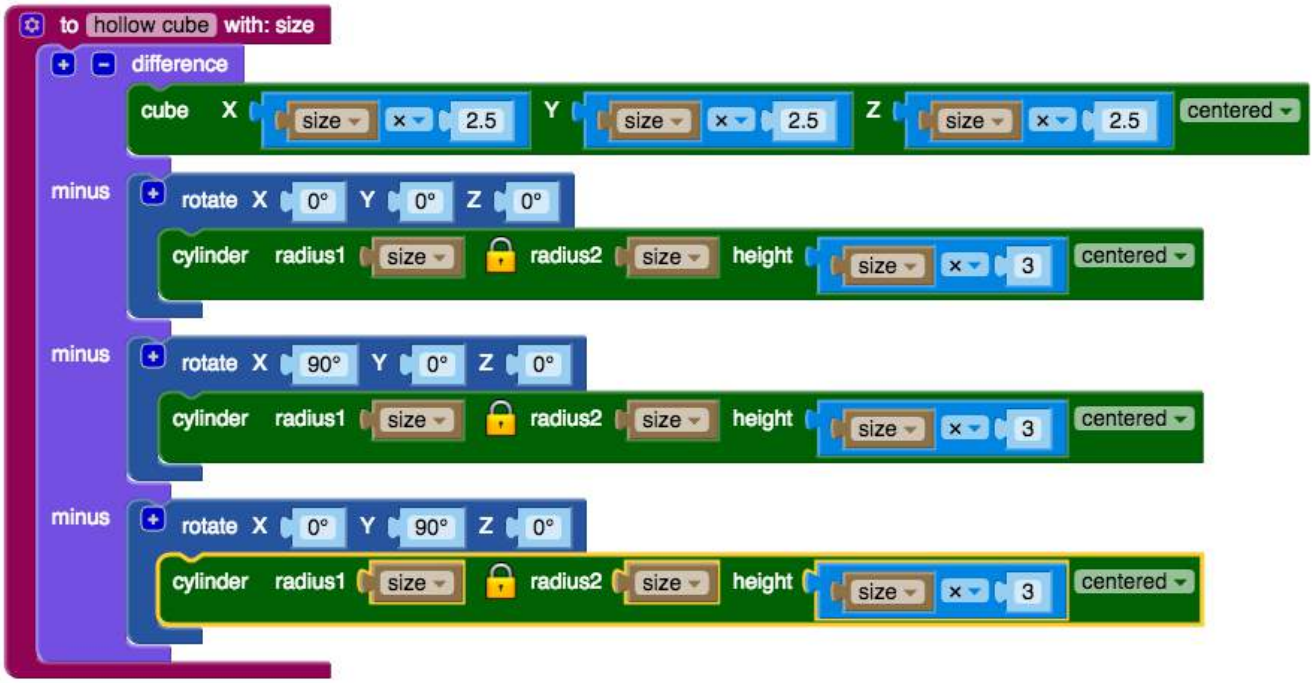
I invite you to explore this some more on your own. There are some good books on the topic, and you can design and build models of other complex plant structures.



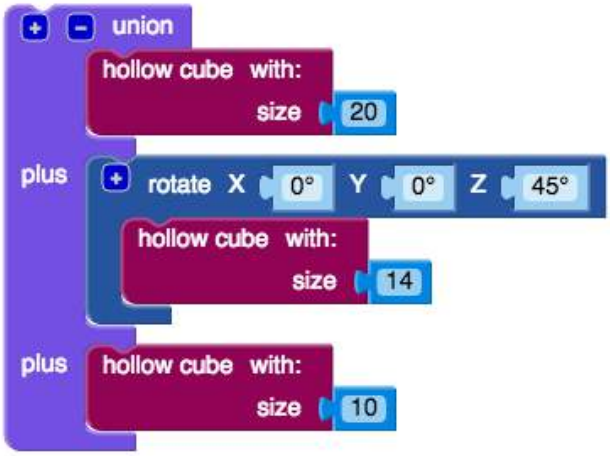
Nested cubes

Our last project for this chapter is an outgrowth of the cube with holes on each axis that we made in Chapter 4. Instead of making one cube, we'll nest three of them together. Each one will be free to move a bit, but they will not come apart. This particular shape, while not of any practical utility, is extremely difficult to make using a subtractive process, but is easy to make with an additive process like 3D printing.

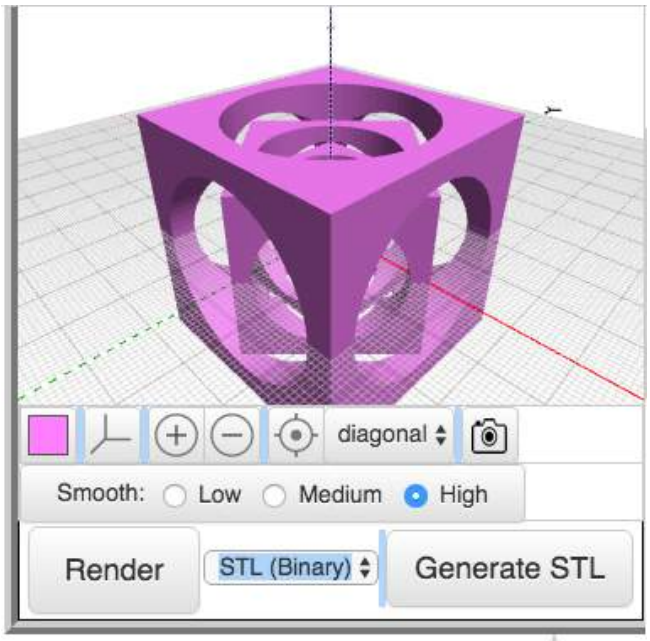
We start by defining our basic cube that incorporates a local variable for the size.



Next, we draw three of them with different sizes. The middle one is rotated around the Z-axis by 45° so everything just fits. The sizes were chosen so that, when finished, each hollow cube is not touching either of the others.

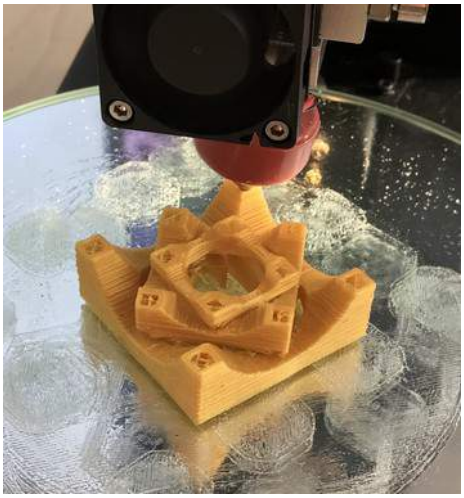


Here's the final shape:

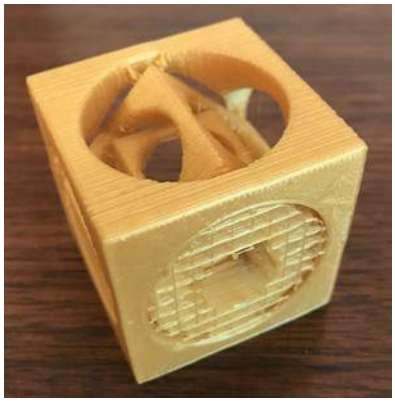


You should magnify and rotate the shape to be sure no two cubes are touching.

Of course, when you print the shape, support structures will be added so nothing floats around during the printing process. Once the printing is done, these supports can be cut out, allowing each cube to rattle around independently.



When finished, carefully remove the cube from the printer.



The support structures are holding the three cubes together. You can peel the supports away with needle nose pliers and dental picks, allowing the three cubes to wiggle around the structure independently.

This completes your project,

Chapter 12 — Dreaming and Doing on Your Own

It has been quite a journey thus far, but you now have played with the basic elements of BlocksCAD. At this point you should experiment on your own, perhaps just by tinkering, and then by designing and building complex parts of your own design. As you've seen, the real power of BlocksCAD comes from making shapes that would be much harder to make with other 3D design tools. Also, you have the added benefit of learning the basics of a programming language that has lots in common with other high-end languages.

While you will have memorized some of the features of this language, I expect you will want to keep this book nearby as you create your own programs. It will take time for you to memorize all the commands and features. That said, there are a few things to remember:

1. There can never be too many comments to improve readability of your programs.

2. BlocksCAD is a compiled language, which means that all global variables are set when the program is compiled.

3. Be prepared to tinker with your design. Virtually every project in this book was tweaked over and over again to get the desired result. Because BlocksCAD programs compile and render pretty quickly, there is no penalty for tinkering.

4. Remember to set the smoothness of your rendering to high when you do your final rendering if you have any curved shapes.

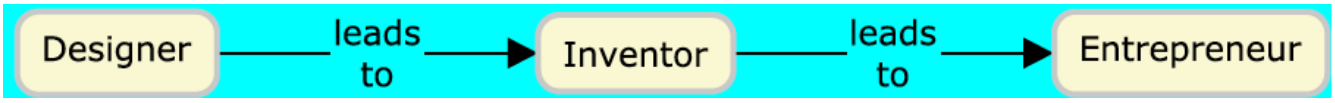
What to make
So far, I've been the one suggesting projects for you to design and build. Now it is time for you to develop strategies to invent your own projects.

There is plenty of low hanging fruit for you to harvest — new switch wall plate covers, replacement shelf brackets, holders for paper towels, and tons of things at home that you can replace with 3D-printed versions. For example, our Christmas tree last year had over a dozen ornaments (including the tree topper) that were printed on my Polar3D printer at home.

If you have a locker at school, I'm sure you'll think of some cool things to build that will help you keep everything organized and easy to find. Some of these things are perfect for you to design with BlocksCAD, and others can be found by searching for STL files online through sites like www.myminifactory.com — a site that links to lots of repositories of 3D designs done by others.

But these activities, as useful as they might be, scarcely hint at the real power you now have.

With everything you've learned thus far, you are ready to invent completely new things, and improve the design of objects that already exist. To move in this direction, you head into the realm of invention. If your invention meets a real need, it might even form the basis of a company! This moves you from the realm of inventor to that of the entrepreneur.



Now you might be thinking, “Hey, how can kids come up with a company to start?” Well, you might be surprised to learn that there are plenty of companies that were started by kids who had great ideas, and the courage to pursue them. A quick web search will reveal a lot. Some high schools, like the Illinois Mathematics and Science Academy have a summer program on entrepreneurship they created to help students design, and start their own companies.

What is an invention?
When Alan Kay and I worked together at the Xerox Palo Alto Research Center in the 1970's, he was fond of saying, “The best way to predict the future is to invent it.” Among other things, we were quite an inventive group of people. The modern graphical user interface for computers, the Ethernet, and the laser printer are just a few of the amazing things that came out of the fertile minds gathered at that location. My own invention of the touch screen, along with myriad other developments, made that facility one of the most creative places on the planet. In fact, our Alto computer was arguably the first personal computer and was developed in 1973. It is not an overstatement to suggest that our collective contributions invented the future.

Building a better mousetrap.
3D printers are powerful tools for inventors to have. If you can design it, you can build it.

According to the US patent system there are three elements of a patentable invention.

1. Novelty
2. Non-obviousness
3. Utility

In order to be patented, your invention needs to meet all of these criteria as well as being a processe, machine, article of manufacture, or composition of matter. (This is called the Statutory requirement). Novelty simply means that it is a new invention — one that has not already been made by someone else. Non-obviousness means that your invention would not be obvious to other people skilled in the general area of your invention. For example, if you invented a new kind of water pump, in order to be patented it would have to have features in its design that are not obvious to existing water pump designers. This can be an area of contention as you go through the patent process, but your patent attorney can help you here. Believe it or not, there are many very simple inventions that are patentable because they never occurred to others “versed in the art” (to use a phrase you may hear in the future). The result is that you should never think that your inventions need to be complex!

Utility simply means that your invention does something useful.

How does the inventive process work?
Sometimes the idea for an invention just pops into your mind. Other times, you might have a general idea or goal, but the final invention will be the product of several people working together. There are various ideation strategies with names like Brainstorming, SCAMPER, etc. that you can use to gather ideas from which your final project can emerge.

Throughout the inventive process, it is important to keep a notebook filled with sketches and notes, and to have each page of this notebook signed and dated by a witness to confirm when the invention was conceived. This becomes handy if someone else files for a similar patent, since it can be used to confirm when you did the work — and timing is everything.

Also, build (and keep) any 3D printed prototypes you make, even if they don't work properly. They should also be signed and dated if possible. Alternatively, you can glue photos of your models in your notebook.

How do I get my ideas supported?
If you need outside support to bring your invention to market, you'll need to have a short but compelling story to share with potential investors. This is called your “elevator pitch,” because it is the sort of thing you can do in a hypothetical elevator ride with a potential investor. You can find how to create effective elevator pitches by searching this term on the Internet.

Before you start selling your invention, it needs to have a name. Product naming is an art. Again, help can be found online to get you started.

Making a better soap dish.
Instead of making a better mouse trap, suppose you wanted to make a better soap dish.

Start by asking what's wrong with current designs? For example, does the soap slide out of the dish too easily? Does soap residue create lots of gunk at the bottom of the dish? Spend some time looking at current soap dish designs to see if you have novel ideas for a better design.

Two aspects of your design are engineering and esthetics. In addition to the engineering aspect of your design, it should also be esthetically pleasing. The nice thing about BlocksCAD is that it lets you modify designs quickly, so you can test out different designs and get feedback from potential users of your new soap dish.

Once you've completed your design and built your prototypes, you might decide to start your own company. Alternatively, you might try selling your idea to a company who is already in the business. In either case, having good legal advice is strongly advised. The last thing you want is for someone to copy your design and leave you with nothing to show for your effort.

To finish off this book, let's change topics and explore ways of sharing your designs with others.

Sharing your designs
While you might design things that are just for your own use, you may also want to share some of your designs with others. There are quite a few sites with rich repositories of 3D printable files, many of which can be downloaded for free.

www.myminifactory.com/
This site hosts over 20,000 models sorted into different categories. Designers can reach potential customers, and the collection is curated, insuring that posted objects will print properly. In fact, they post photos of actual prints for each of the models they host. Their collection is growing rapidly, and is broken down into categories, making it easy to browse for interesting models.

http://www.yeggi.com/
Yeggi is a search engine for posted 3D shapes that looks at numerous sites where objects can be posted and downloaded. It is a great first stop for those times you're looking for something specific and lack the time to design it yourself.

Each of these sites is perfect for loading your own models. If you are looking for models to print or edit, you should look at all of them. And, there are other places where people upload models to be downloaded.

goo.gl/BHqNcK
If you are an educator, you will definitely want to check this out. I've created a template you can use for STEAM-related projects that entail 3D models. Once you create a piece of curriculum using this template, it will be professionally mentored and the final lesson posted on the Teacher Idea Center as part of STEAMtrax Plus, which can be searched by subject area and grade level. This is a tremendous resource for educators, and students alike!

No matter what site you use for posting models, there are some important things to do:

1. Be sure your design can be printed. It is easy to design 3D shapes that are hard, if not impossible, for some printers to make. If you can't build it with your own printer, don't post it.

2. If you want people to have your BlocksCAD and related files, be sure these are put in a zipped folder along with your sample STL file so people have access to them. You probably should include the OpenSCAD files as well since this is a popular tool with the more technically-inclined.

3. Choose the licensing for your copyright (and respect the copyrights of anyone whose design you have modified).

Creative Commons
Most people choose to release their designs using the Creative Commons Copyright licenses (creativecommons.org). Their free, easy-to-use copyright licenses provide a simple, standardized way to give the public permission to share and use your creative work — on conditions of your choice. Creative Commons licenses let you easily change your copyright terms from the default of "all rights reserved" to "some rights reserved." These licenses are not an alternative to copyright. They work alongside traditional copyrights and enable you to modify your copyright terms to best suit your needs.

It is recommended that you visit their site to see the various licenses they have created so you can choose the one that is best suited for each project you post.

Most of all, let your creativity run free. Don't be afraid to experiment. If your design doesn't work, tinker with it until you get what you want. Also, be prepared to find something unexpected in your designs that you can turn into a nifty project.

Acknowledgements

This book is the outgrowth of work done by many people all over the world. First, because BlocksCAD is based on OpenSCAD, thanks have to be given to the entire OpenSCAD community, especially Marius Kintel, the developer and maintainer of OpenSCAD.

The block-like programming metaphor used in BlocksCAD is an outgrowth of the programming metaphor used by Scratch, developed in the Lifelong Kindergarten group at MIT's Media Lab under the leadership of Professor Mitch Resnick. BlocksCAD uses the Blockly interface first proposed by Neil Fraser at Google.

BlocksCAD itself was developed by Jennie Yoder under the leadership of Einstein's Workshop.

The insights of many people contributed to some of the ideas in this book. Feedback and interactions with these amazing people helped turn what started as an instruction manual into a far more useful book. Special thanks go to Dan Newman, Sara Armstrong, and others who shared their comments along the way.

About the Author

Dr. David Thornburg is the Director of Education for Polar3D and has written numerous books on programming languages for education — especially on Logo. He is also the co-author of the first book on 3D printing in the classroom, and is a lifelong proponent of the constructionist model of education based on inquiry-driven project-based learning.

With his strong interdisciplinary background, David applies his strengths in the STEAM fields to all his work, and is a highly respected presenter at education conferences worldwide.