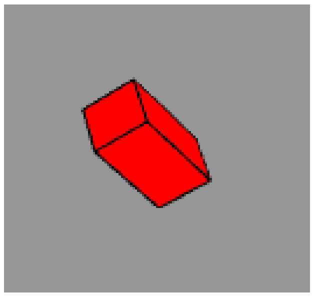
# The 3rd Dimension!

Up to this point we have been working in two dimensions; the x direction (width) and the y directions (height). We are now going to take a simple look at the the third dimension or the z direction which is (depth). There are a few 3D shapes that we will put to good use in this HotSheet.



**PROCESSING**
**SKILL REQUIREMENTS**
DIFFICULTY 1-5

| | |
|---|---|
| ELECTRICAL PROTOTYPING | ○○○○○ |
| ROBOTICS | ○○○○○ |
| SOLDERING | ○○○○○ |
| PROGRAMMING | ●●○○○ |
| DIY | ○○○○○ |

## MATERIALS LIST

- Computer
- Processing
- Graph Paper
- Colored Pencils
- Super fine tip sharpie

## STEP 1: The 3rd Dimension

We are really familiar with X and Y, width and height in Processing. Going into the 3rd dimension takes a little more work but it is well worth it. When we look at our screens we know that the width increases as we move from left to right. The height increases from the top down. For depth, or the z direction, it decreases as it moves further into the screen and 0 is at the screen itself.

## STEP 2: Setup?!

If we are going to be dealing with 3D shapes we have to add something to our setup code. In our 2D sketches we only have width and height in our `size()` function. In 3D we add a third value to `size()`, we add P3D after width and height. Our basic 3D `size()` setup looks like this.

```
void setup()
{
  size(600,600,P3D);
}
```

## STEP 3: Basic Shapes

Now that we have our basic setup we can now draw a few basic 3D shapes. Here are the basic 3D functions that we can play with:

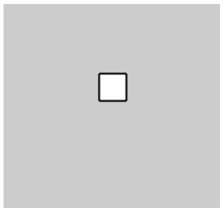
`box(size);` - produces a cube with each with a length of "size"

`box(w,h,d);`- produces a box with a width (w), a height (h), and depth(d). it is always placed at 0,0,0!

`sphere(radius);` - produces a sphere with a radius of "radius". It is always placed at 0,0,0!

## STEP 4: Drawing Shapes

We have one more thing that we have to add for each 3D shape. Since all 3D shapes are placed at 0,0,0. We need to place them in a matrix so we can move it around using translate() and the other transformation functions we have used with the `pushMatrix()` and `popMatrix()`. For a basic cube we start by creating a matrix using `pushMatrix()` and `popMatrix()`. Between the two we will place our `box()` function with a dimension of 25 pixels. We can now use `translate()` to move the box to the center of the window.



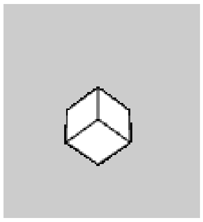
```
void draw()
{
 pushMatrix();
 translate(width/2,height/2);
 box(25);
 popMatrix();

}
```

We created a box but we are looking directly at it from one side. We need to rotate the box, but if we use our basic rotate() function we will only rotate on the X,Y plane and will not rotate away or towards us. 3D shapes have their own transformation functions.

## STEP 4: 3D Transformation

Now that we have everything up and running lets take a look at some 3D transformation functions. We want to rotate that box so we can see that it is 3D! Here are a few:

`rotateX(R);` - rotates the 3D shape on the X axis in R radians.
`rotateY(R);` - rotates the 3D shape on the Y axis in R radians.
`rotateZ(R);` - rotates the 3D shape on the Z axis in R radians.

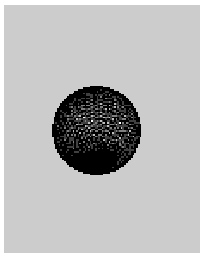


```
void draw()
{
 pushMatrix();
 translate(width/2,height/2);
 rotateX(QUARTER_PI);
 rotateY(QUARTER_PI);
 box(25);

 popMatrix();

}
```

So, we add `rotateX(QUARTER_PI);` and `rotateY(QUARTER_PI);` to our code and we get a box that has been rotated 45 degrees on the X axis and 45 degrees on the Y axis.

## STEP 5: Sphere

We can replace box with sphere(25) and get a sphere that has a radius of 25 pixels. The sphere is made up of triangles that fit together. We can also fill() these shapes with color we can also use all of our stroke modifiers to make changes to our 3D shapes.



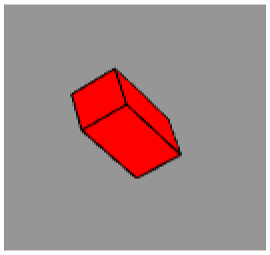
```
void draw()
{
 pushMatrix();
 translate(width/2,height/2);
 rotateX(QUARTER_PI);
 rotateY(QUARTER_PI);
 sphere(25);

 popMatrix();

}
```

## STEP 6: Motion

We can also set any value that we use to any previous built in variables that we have used in previous HotSheets. For example: `mouseX`, `mouseY`, `millis()`, `seconds()`, etc. We are going to have the cube bounce back and forth on the and grow based on time.

As an example of this we took the previous motion HotSheet and applied it to a box. Notice that some of the code that we have been working with is there still, we have just added a few built in variables and also added an X variable that we can increment and a grow variable that we change based on a bounce. We also filled the box with red and changed its height by `second()*2` which causes it to grow over one minute and then shrink to flat when a new minute starts.

Try it out, maybe add a second shape in a different matrix, have that one bounce up and down. What happens when you change the variable rotations to different axis?



```
int x=300;
int grow= 1;
void setup()
{
  size(600,600,P3D);
}

void draw()
{
 background(150);
 pushMatrix();
 fill(255,0,0);
 translate(x,height/2);
 rotateX(millis()*.001);
 rotateY(QUARTER_PI);
 box(25,25,second()*2);
 popMatrix();

 x=x+grow;

 if(x<=0||x>=width)
 {
   grow= grow*-1;
 }
}
```

## STEP 8: Share your work!

OpenProcessing

If you would like to share your work...which you do! You can sign up at openprocessing.org. Open Processing allows you to share your code, view the drawing online, as well as allows others to fork or borrow your code to manipulate and change it on their own while still keeping your sketch intact for others to check out.

### TAKING IT FURTHER

- Make a pyramid of boxes stacked one on top of the other
- Make a 3D snowman!
- Combine 3D and 2D and see what happens!