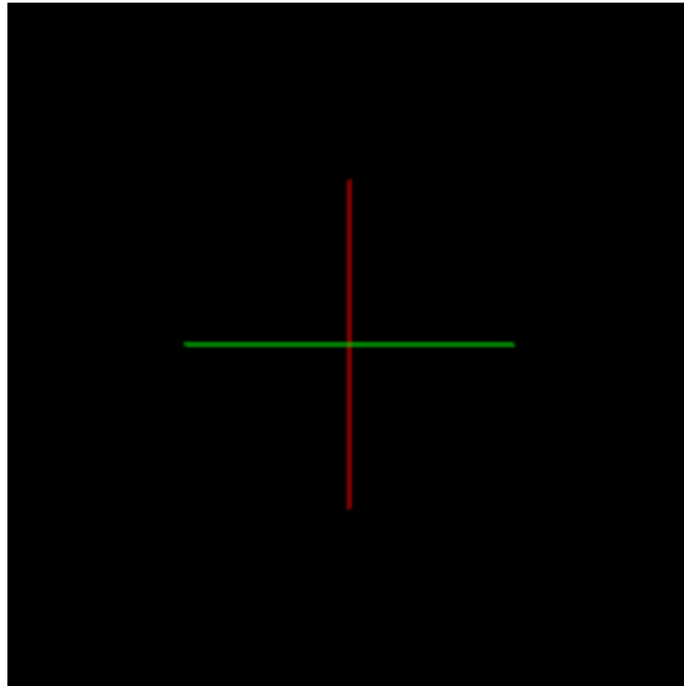
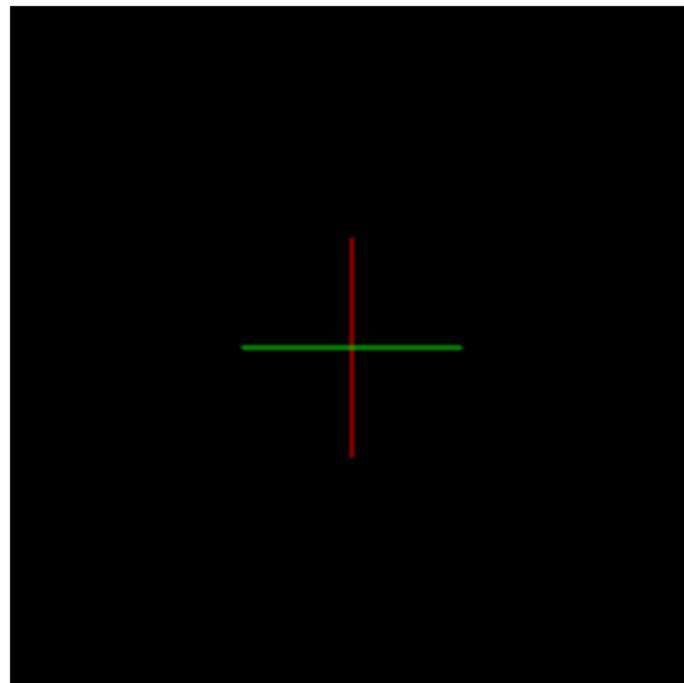


0. The Initial Form



1. Comment out the `lookAt()` call and replace it with `translate()` with parameters `(0,0,-15)`. Is there any change in the display? Why ? Why not? Please explain in detailed and add image. Explain your process.



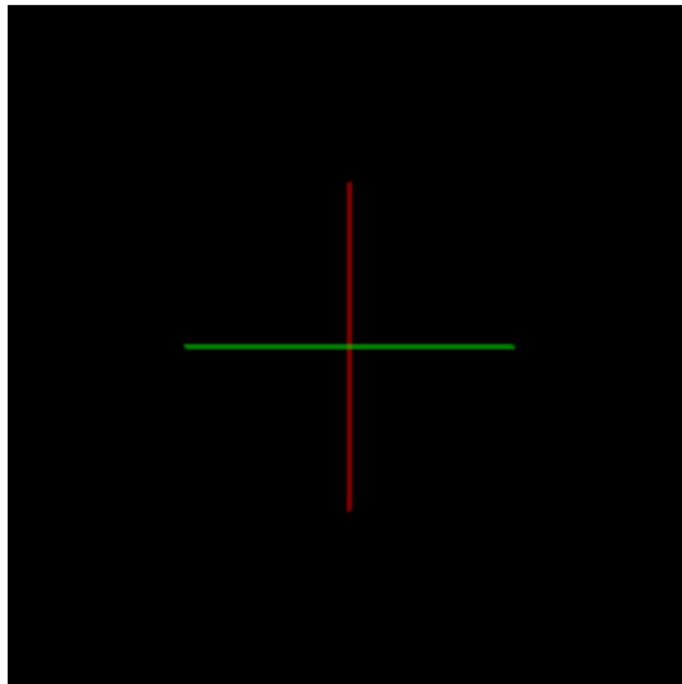
Yes, there is a change. It is because when we transform in the z axis, we (the camera and so our view) are moving away, that is why the cross looks smaller.

- 2. Comment out both the `lookAt()` and `translate()` lines. What happens? Why? Explain your process.**

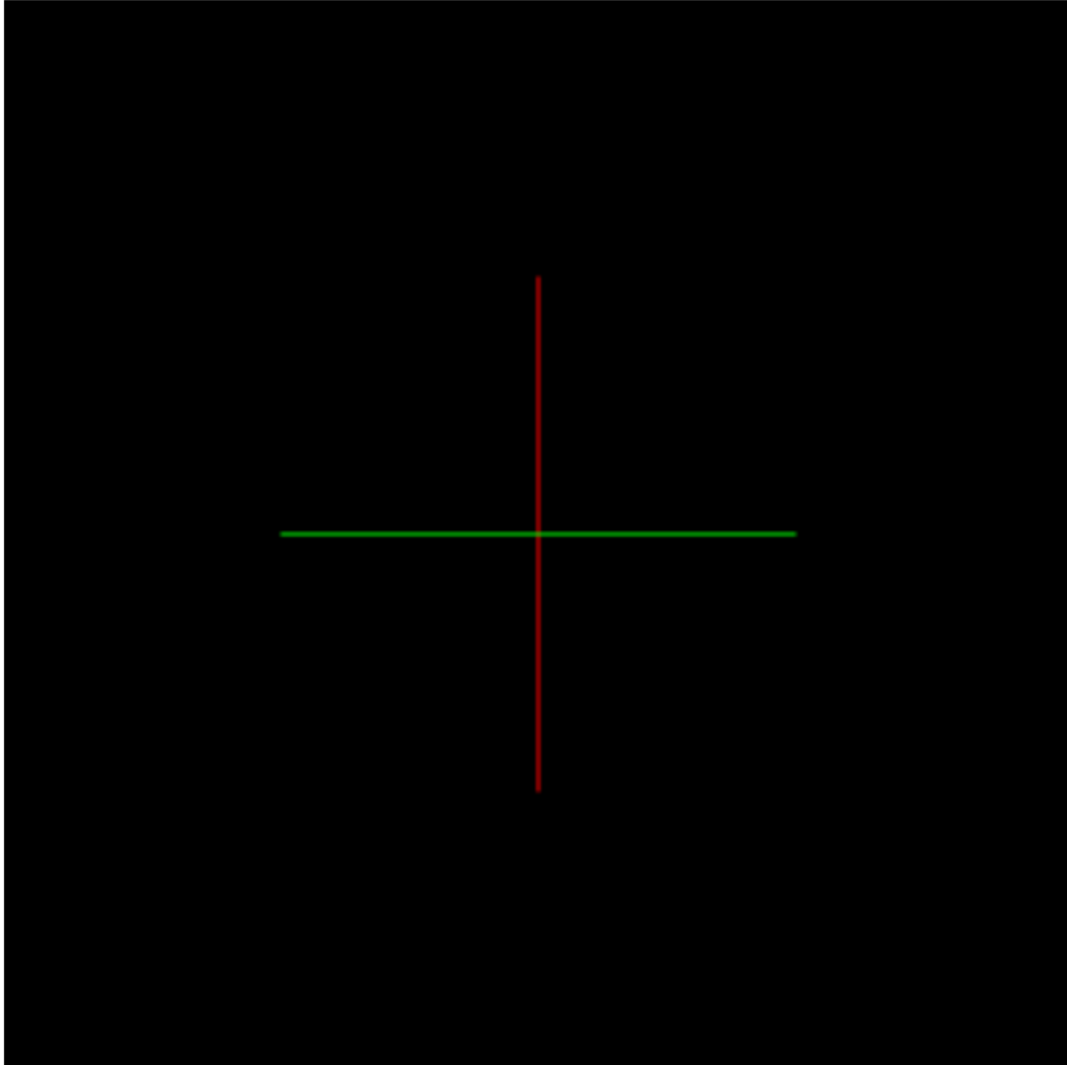


The cross is not seen. That does not mean that it disappears we just do not see it.

- 3. Restore the `lookAt()` call.**

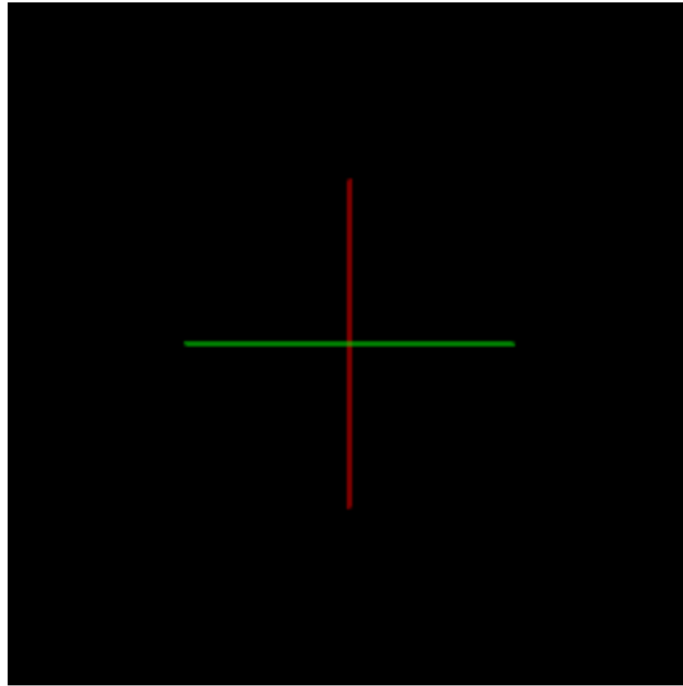


4. Take a look at the perspective() call. The aspect ratio you were originally given was 1.0. Explain your process.
- 4.1. What happens when the aspect ratio is 1.0 and you change the canvas dimensions in boxes.html to width="400", height="400"? Add the result image.



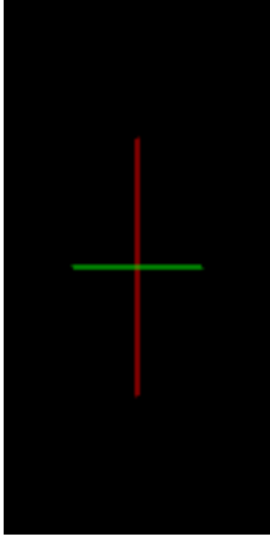
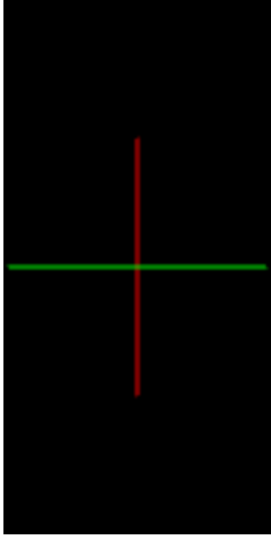
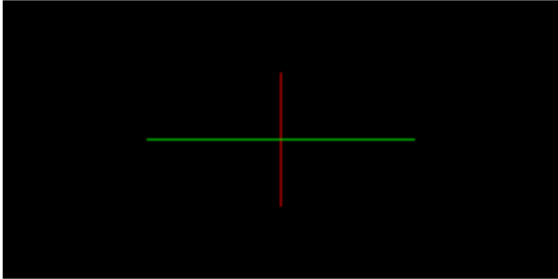
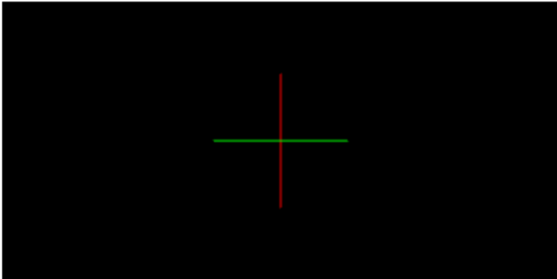
The canvas gets bigger.

4.2. How about width="256" and height="256". Add the result image.

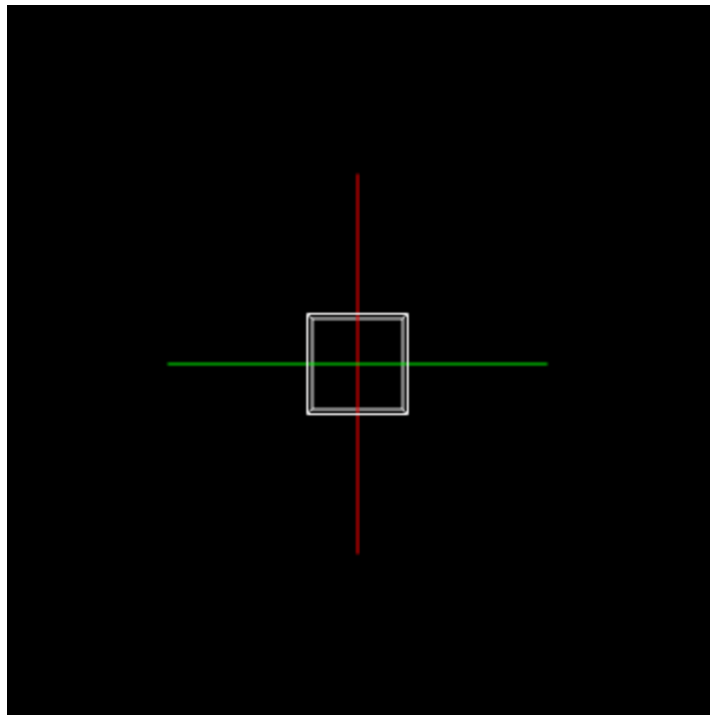


The canvas gets back its old shape and size.

4.3. The aspect ratio allows us to compensate for different shapes of canvas. Study there commendations in point 2. of WebGLAnti-Patterns, then modify the aspect ratio in your perspective call so that it is an appropriate ratio of width to height based on the actual dimensions of the viewing area. Test the result with the two suggested canvas shapes to be sure you got it right.

Before changing the code	After changing the code
 <p>Width = 100, Height = 200</p>	 <p>Width = 100, Height = 200</p>
 <p>Width = 400, Height = 200</p>	 <p>Width = 400, Height = 200</p>

5. Draw a wireCube centered (0,0,0) relative to the axes. You can use the provided buffers and related shape data. Do this in the render function. Explain your process.



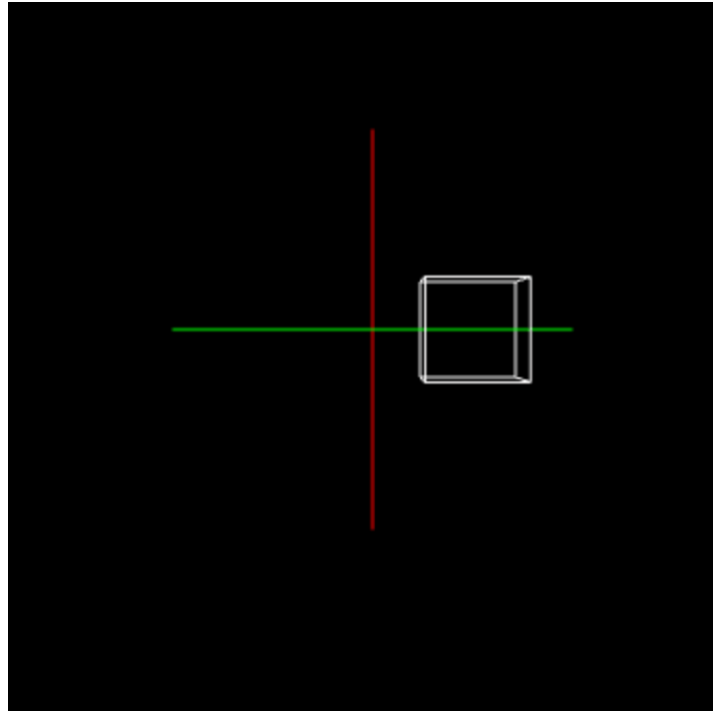
```
//wire Cube 1
var Cube1Matrix = translate(1.0, 0.0, 0.0); // Identity matrix initially

mv1 = mult(mv, Cube1Matrix); // Update the model-view matrix
gl.uniformMatrix4fv(mvLoc, gl.TRUE, flatten(transpose(mv1)));

// Draw the wire cube
gl.drawArrays(shapes.wireCube.type, shapes.wireCube.start,
shapes.wireCube.size);
```

First, I found the translation matrix and then I multiplied the model-view matrix with the translation matrix. After that, I send model-view matrix to the shader program and draw the wire cube.

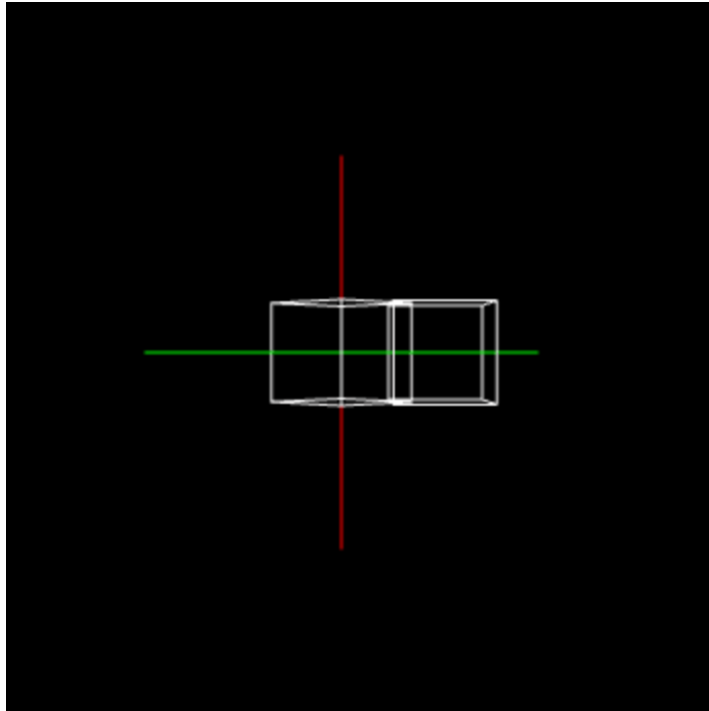
6. Move this cube so that it is centered as $(1,0,0)$ relative to the axes. Explain your process.



```
var centeredCubeMatrix = translate(1.0, 0.0, 0.0);
```

To make the cube centered as $(1,0,0)$, I kept the code as 5th stage and just changed the transformation matrix from $(0,0,0)$ to $(1,0,0)$.

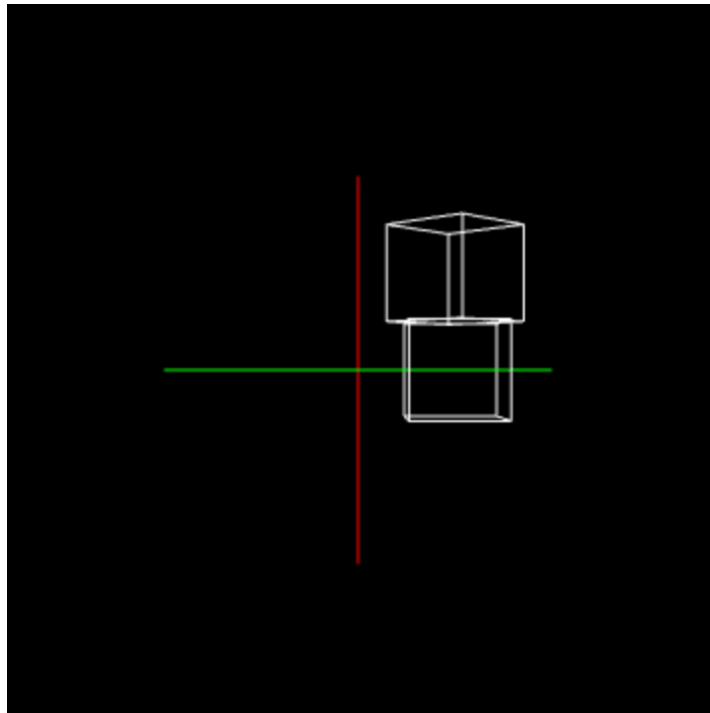
7. Draw a second cube after the first –in a new colour(**blue**) if you can- and rotate it 45 degrees around the y-axis. Explain your process.



```
var Cube2RotationMatrix = rotate(45.0, 0.0, 1.0, 0.0);  
  
mv2 = mult(mv, Cube2RotationMatrix);  
  
gl.uniformMatrix4fv(mvLoc, gl.TRUE, flatten(transpose(mv2)));  
  
gl.drawArrays(shapes.wireCube.type, shapes.wireCube.start,  
shapes.wireCube.size);
```

To create a second cube, I first created the rotation matrix. then I multiplied the model-view matrix with the rotation matrix. After that, I send model-view matrix to the shader program and draw the wire cube just like I did in the first wire cube. However, I could not find a way to the cube blue.

8. Place this rotated cube directly above the first cube. It will be centered at (1,0,0) relative to the axes. Be careful of the order of transformation. Explain your process.



```
//wire Cube 2
var Cube2RotationMatrix = rotate(45.0, 0.0, 1.0, 0.0);
var Cube2TranslationMatrix1 = translate(1.0, 0.0, 0.0);
var Cube2TranslationMatrix2 = translate(0.0, 1.0, 0.0);

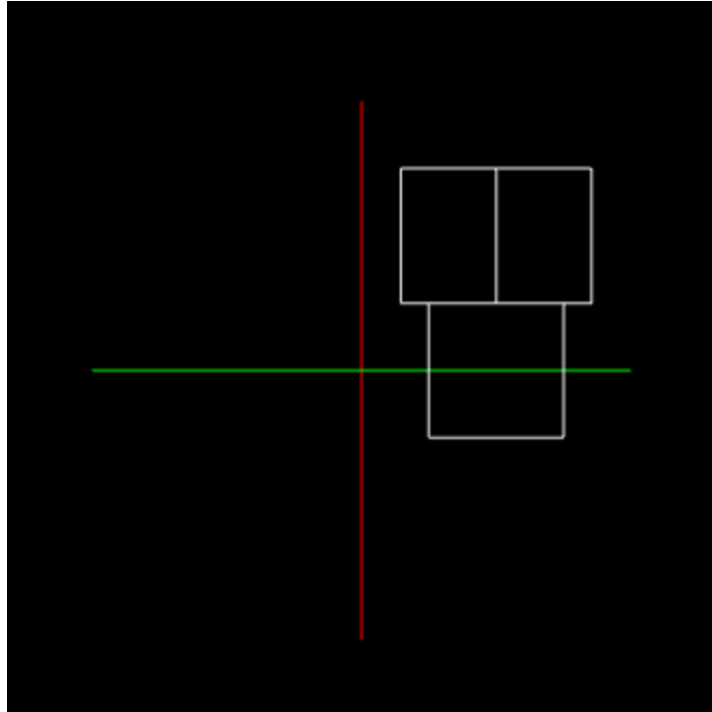
mv2 = mult(mv, Cube2TranslationMatrix1);
mv2 = mult(mv2, Cube2RotationMatrix);
mv2 = mult(mv2, Cube2TranslationMatrix2);

gl.uniformMatrix4fv(mvLoc, gl.TRUE, flatten(transpose(mv2)));

gl.drawArrays(shapes.wireCube.type, shapes.wireCube.start,
shapes.wireCube.size);
```

I firstly created rotation and transformation matrices for cube 2. As the next step, I multiplied these matrices with the model-view matrix. I have already explained the rest, so I will not explain it again. As a result, I moved this cube to the desired location.

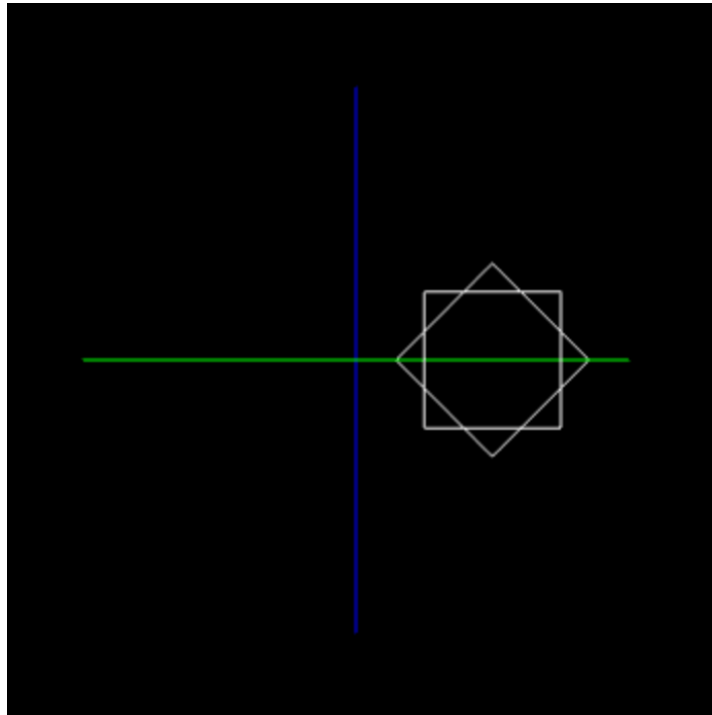
9. The perspective view makes the two cubes look a little awkward. Try using orthographic projection instead of the perspective call. The function for that is: `ortho`. Use left, right, bottom, top, near and far values that include the whole scene and not much more. See the picture for expected results. Explain your process.



```
p = ortho(-3, 3, -3, 3, 0.1, 100.0);
```

I used the `ortho` function to change our code from a perspective view to an orthographic view. I also commented out “`p = perspective(45.0, aspect, 0.1, 100.0);`” part.

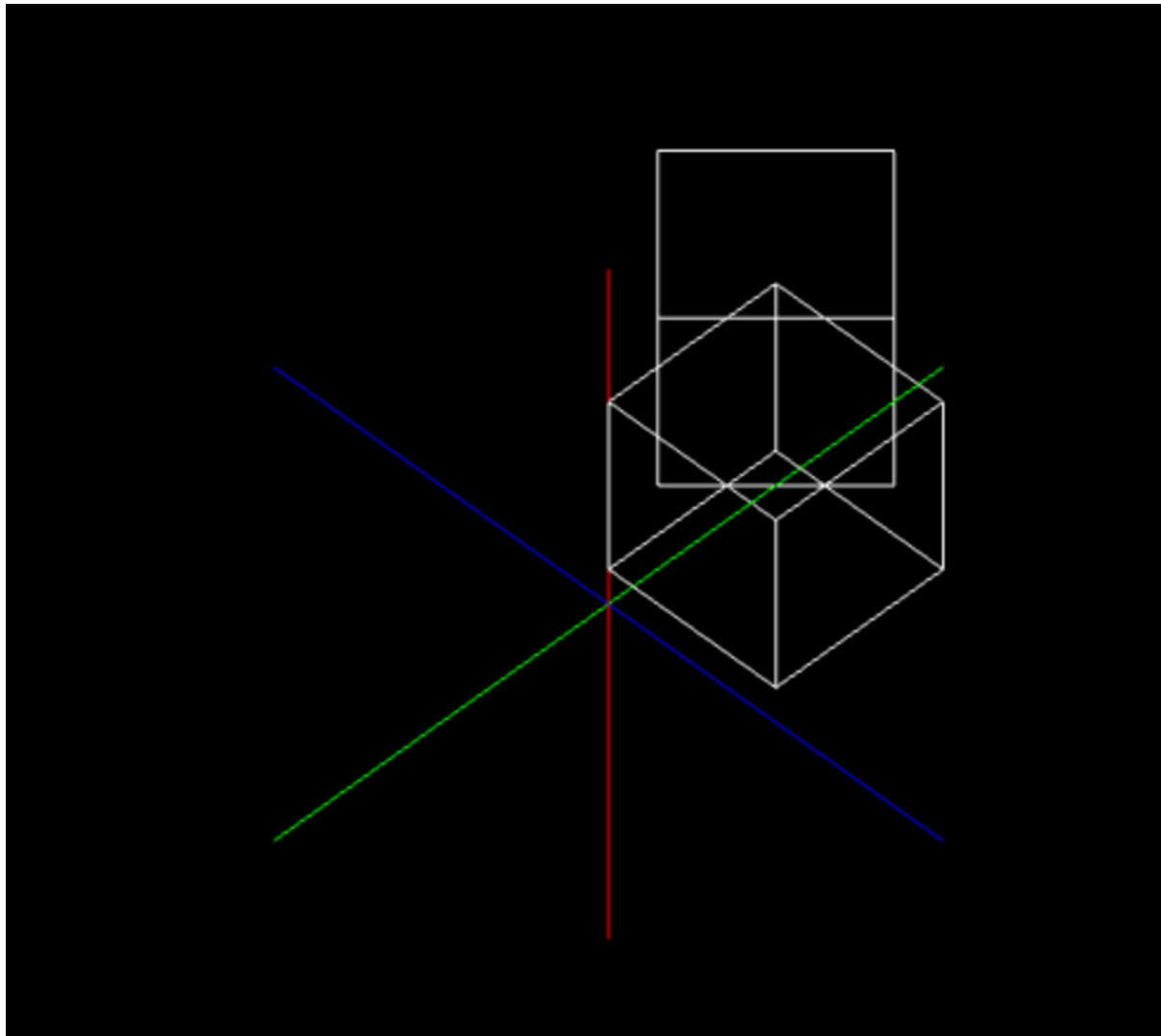
- 10. Rotate everything (using modelling transformations NOT lookAt) so that looking down at the top of the boxes and seeing the blue z-axis (and no red y-axis). See the picture for expected results. Explain your process.**



```
mv = lookAt(eye, at, up);  
// Define the rotation matrix  
var xRotationMatrix = rotate(90.0, 1.0, 0.0, 0.0);  
  
mv = mult(mv, xRotationMatrix);
```

In order to change our view direction without changing the lookat function, we had to multiply the model-view matrix by a rotation matrix. Using the 90 degree x-axis transformation matrix, I obtained the desired result.

11. Rotate everything so that you can see all three axes along with the cubes. See the picture for expected results. Explain your process.



```
var xRotationMatrix = rotate(45.0, 1.0, 0.0, 0.0);  
var xRotationMatrix2 = rotate(45.0, 0.0, 1.0, 0.0);  
  
// Update the model-view matrix for the entire scene (e.g., before drawing the  
axes)  
mv = mult(mv, xRotationMatrix);  
mv = mult(mv, xRotationMatrix2);
```

I rotated it 45 degrees on the x and y axes to see the shape more easily. To do this, I had to create a matrix that rotates on the x-axis and a matrix that rotates 45 degrees on the y-axis. Then I multiplied these matrices by the model-view matrix respectively and got the result above.