CSCI 4810 Assignment 3

Demonstrating Perspective Projection and 3D Geometric Transformations with Code

Ellemieke Van Kints, ejv88036@uga.edu
Hamid Arabnia, hra@uga.edu

October 31, 2022

1 Introduction

This report begins with an overview of my algorithm to perform perspective projection and 3D geometric transformations. After the algorithm overview, I test my program using the parameters provided in the project description. Then, I experiment with different parameter values and discuss the impact of each parameter on the resulting image. A cube is used by default for all tests throughout this report, but other shapes, including a pyramid and rectangular prism, are also tested to show that my program is generalizable to any 3D object made up of lines. This is followed by various other experiments to demonstrate that perspective projection and the 3D geometric transformations work correctly. The output for all tests are provided.

2 Program Overview

My algorithm begins by reading in 3D coordinate points stored in a text file. These points represent the line endpoints of a 3D object, and they are defined in the World-Coordinate System (WCS). In order to perform a geometric transformation on the 3D object, a transformation matrix is applied each coordinate point. For the scale and rotate operations in particular, intermediate transformations are concatenated into a singular transformation matrix, which is then applied to the coordinate points.

These transformed coordinate values are then mapped to the Eye-Coordinate System (ECS) using a viewing transformation matrix, which is calculated by lining up the axes of each coordinate system. This calculation involves multiple steps, but they are all concatenated into one overall transformation matrix, V, which is then applied to the coordinate points. Then, perspective projection is used to transform the 3D coordinates to their 2D locations on the screen. In my implementation, a 1024 x 1024 screen resolution is used, and the resulting 2D coordinates are checked to ensure they lie within the bounds of the screen. Only completely visible images are supported in my implementation. Finally, if the object is completely visible, it is scan-converted and displayed on the screen.

3 Perspective Projection

To demonstrate that perspective projection works correctly, I will first run a test using the parameter values specified in the project description. The *viewpoint* is set to (6, 8, 7.5), with the viewing axis z_e pointed directly at the origin of the WCS. The screen is assumed to be a square for all tests. The *screen size* is set to 30 cm, and d, the distance that a viewer should sit away from the screen for optimal viewing, is set to 60 cm. The output from this test can be seen in *Figure 1.*

I will now run tests with varying *viewpoint* coordinates. All other parameters

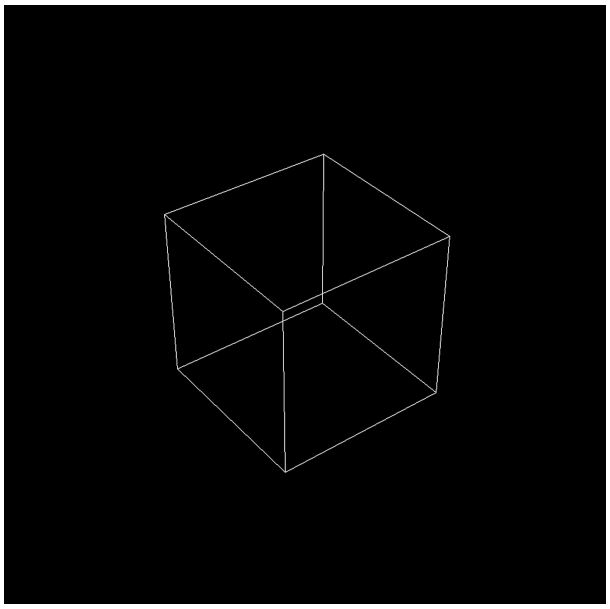

Figure 1: Output from perspective projection, where $viewpoint = (6, 8, 7.5)$, $screen\ size = 30$ cm, and $d = 60$ cm.

will be kept the same. First, I will set *viewpoint* to (6, 8, 30). The output from this test can be seen in *Figure 2*. Here, we can see that increasing the z viewpoint coordinate, or z_e, causes the object to move further away from us. We also appear to be viewing the object from a steeper angle, but this is expected as we move further away from the origin of the WCS. Next, I will set *viewpoint* to (6, 8, 1). The output from this test can be seen in *Figure 3*. Here, we can see that decreasing the z viewpoint coordinate causes the object to move closer to us. We also appear to be viewing the object straight-on, which is expected, as z_e points directly at the origin of the WCS.

I will now run tests with varying *screen size* values. For these tests, *viewpoint* will be set to (6, 8, 20) and d will remain the same. First, I will set *screen size* to 10. The output from this test can be seen in *Figure 4*. Here, we can see that decreasing the screen size causes the object to appear closer to us. This is expected, as a smaller screen size means the object will take up more space on the screen. Next, I will set *screen size* to 60. The output from this test can be seen in *Figure 5*. Here, we can see that increasing the screen size causes the object to appear further away. This is also expected, as a larger screen size means the object will take up less space on the screen. For both tests, we can see that the angle at which we are viewing the object remains the same. This makes sense, as our *viewpoint* coordinates are the same for the both tests.

I will now run tests with varying d values. All other parameters will be kept the same. First, I will set d to 10. The output from this test can be seen in *Figure 6*.

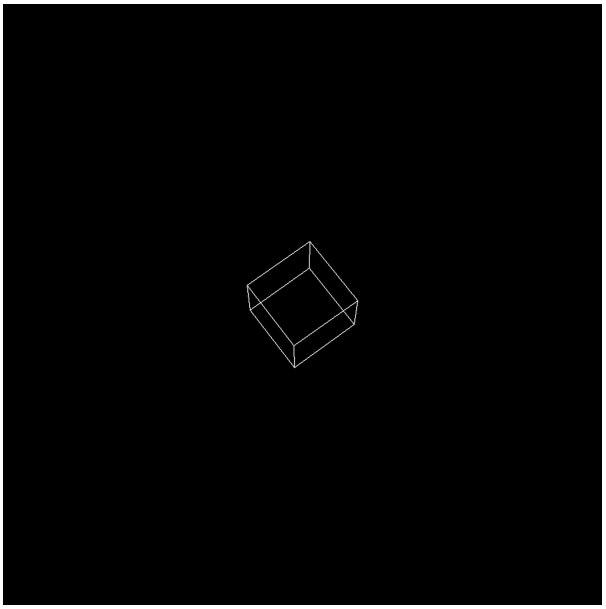

Figure 2: Output from perspective projection, where $viewpoint = (6, 8, 30)$, $screen\ size = 30$ cm, and $d = 60$ cm.

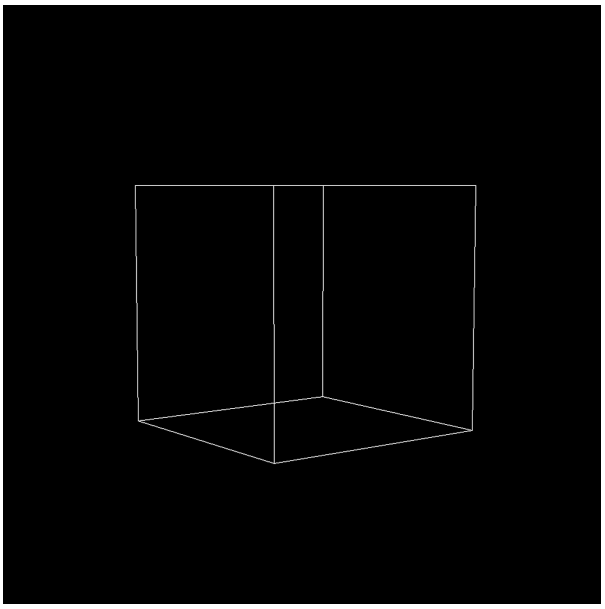

Figure 3: Output from perspective projection, where $viewpoint = (6, 8, 1)$, $screen\ size = 30$ cm, and $d = 60$ cm.

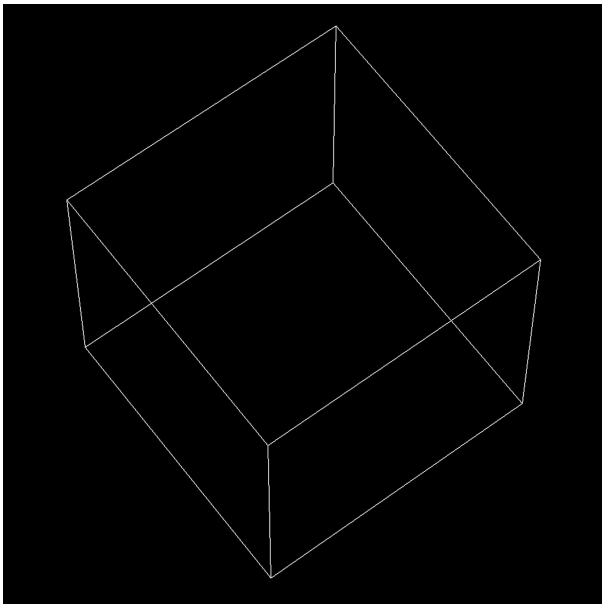

Figure 4: Output from perspective projection, where $viewpoint = (6, 8, 20)$, $screen\ size = 10$ cm, and $d = 60$ cm.

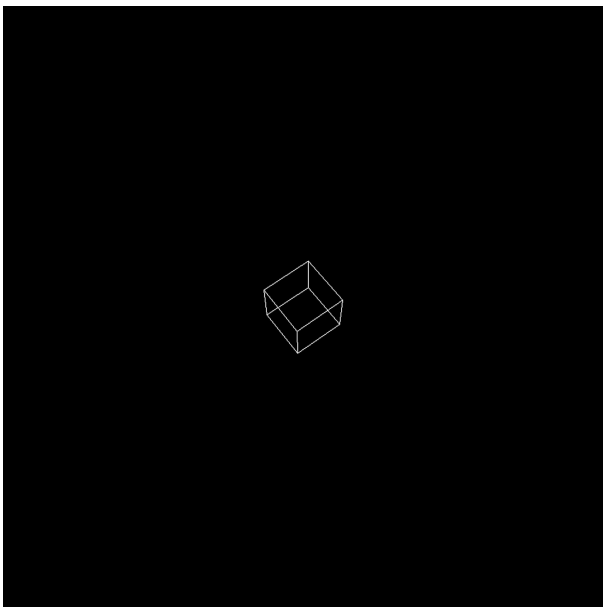

Figure 5: Output from perspective projection, where $viewpoint = (6, 8, 20)$, $screen\ size = 60$ cm, and $d = 60$ cm.

Here, we can see that decreasing the distance that a viewer should sit away from the screen for optimal viewing causes the object to appear further away. This is expected, as a smaller d value indicates that object will appear small on our screen, because the viewer will be be sitting very close to the screen. Next, I will set d to 100. The output from this test can be seen in *Figure 7*. Here, we can see that the distance that a viewer should sit away from the screen for optimal viewing causes the object to appear closer to us. This is also expected, as a larger d value indicates that object must appear large on the screen, because the viewer will be be sitting far away from the screen. The value of d differs from device to device, so when we are looking at the same object on various devices, it will be displayed exactly the same on our retina.

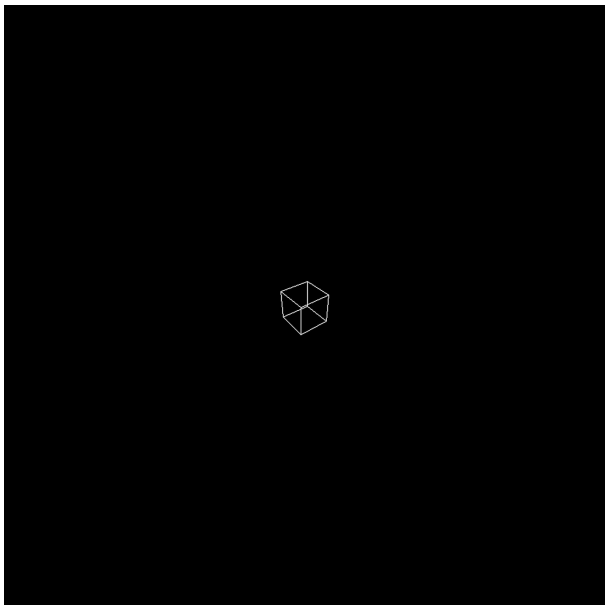

Figure 6: Output from perspective projection, where $viewpoint = (6, 8, 7.5)$, $screen\ size = 30$ cm, and $d = 10$ cm.

I will now run tests with varying screen resolutions. All other parameters will be kept the same. First, I will set the resolution to 960 x 600. The output from this test can be seen in *Figure 8*. Here, we can see that increasing the width of the display also increases the width of the object being displayed. Likewise, when the resolution is set to 600 x 960, we can see that increasing the length of the display also increases the length of the object being displayed. The output from this test can be seen *Figure 9*.

3.1 Using Other Shapes

To demonstrate that my program is generalizable to other 3D geometric shapes, I will test perspective projection with both a pyramid and a rectangular prism.

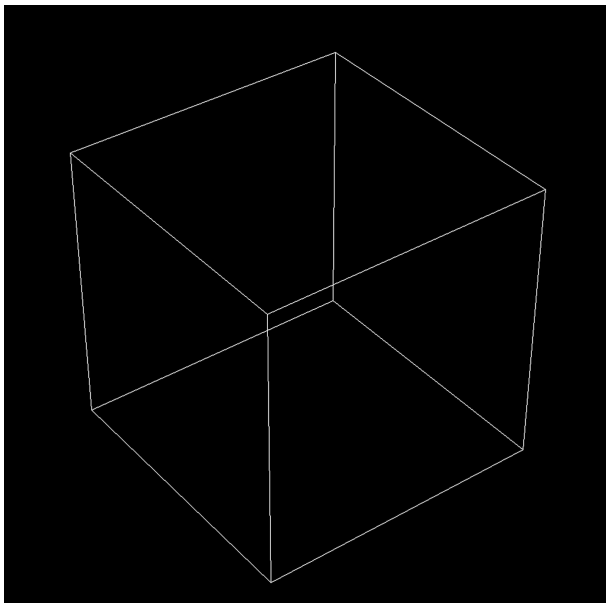

Figure 7: Output from perspective projection, where $viewpoint = (6, 8, 7.5)$, $screen\ size = 30$ cm, and $d = 100$ cm.

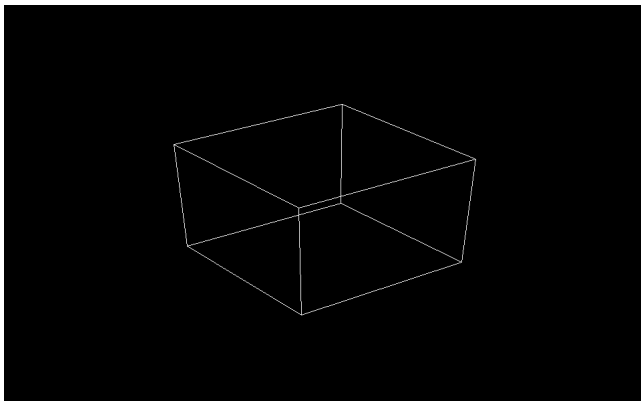

Figure 8: Output from perspective projection, where $viewpoint = (6, 8, 7.5)$, $screen\ size = 30$ cm, $d = 60$ cm, and the screen resolution is 960x600.

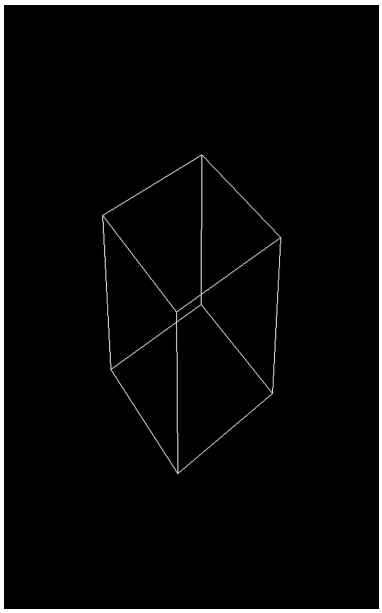

Figure 9: Output from perspective projection, where $viewpoint = (6, 8, 7.5)$, $screen\ size = 30$ cm, $d = 60$ cm, and the screen resolution is 600x960

The tests will use the parameter values specified in the project description. The perspective projection of the pyramid can be seen in *Figure 10*, and the perspective projection of the rectangular prism can be see in *Figure 11*.

4 3D Geometric Transformations

I will now run various tests to demonstrate that the 3D geometric transformations *translate*, *scale*, and *rotate* all work correctly. The cube, pyramid, and rectangular prism will be used in each test to show that my program is generalizable to any 3D object made up of lines. Each test is described in detail below.

4.1 3D Translate

To demonstrate that 3D translation works correctly, I will run tests for each translation in the x, y, and z direction. For each test, the *viewpoint* will be set to (6, 8, 20), the *screen size* will be set to 30 cm, and d will be set to 60 cm. I will begin with translations in the x direction. First, I will set the x displacement value to 2. The y and z displacement values will be set to 0. The output for this test can be seen in *Figure 12*. Here, we can see that a translation by a positive value in the x direction causes the object to shift down and to the left. This behavior is opposite of what we would expect, but it is correct. Our

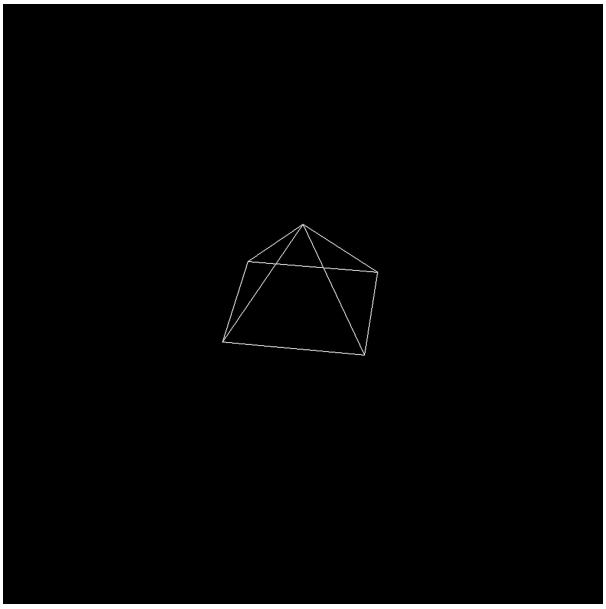

Figure 10: The perspective projection of a pyramid, where *viewpoint* $= (6, 8, 7.5)$, *screen size* $= 30$ cm, $d = 60$ cm.

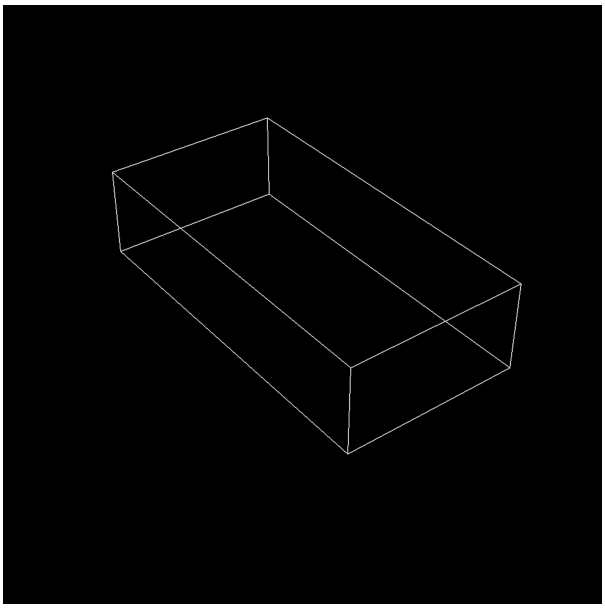

Figure 11: The perspective projection of a rectangular prism, where *viewpoint* $= (6, 8, 7.5)$, *screen size* $= 30$ cm, $d = 60$ cm.

point of view is in the virtual world, so the flipped directions are due to the perspective projection of the image. Next, I will set the x displacement value to -2. The y and z displacement values will be set to 0. The output for this test can be seen in *Figure 13*. Here, we can see that a translation by a negative value in the x direction causes the object to shift up and to the right. The behavior is opposite of what we would expect, but it is also correct, for the same reason as before. In both tests, our line of gaze z_e remains pointing towards the origin of the WCS.

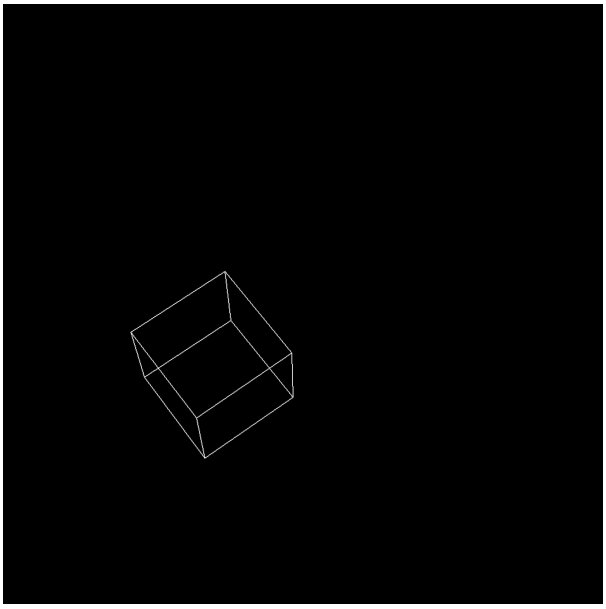

Figure 12: Output of 3D Translation, where $x = 2$, $y = 0$, and $z = 0$.

I will now run tests with various translations in the y direction. All other parameters will be kept the same. The pyramid will be used to demonstrate this test. First, I will set the y displacement value to 2. The x and z displacement values will be set to 0. The output for this test can be seen in *Figure 14*. Here, we can see that a translation by a positive value in the y direction causes the object to shift down and to the right. This behavior is opposite of what we would expect, but it is correct. Our point of view is in the virtual world, so the flipped directions are due to the perspective projection of the image. Next, I will set the y displacement value to -2. The x and z displacement values will be set to 0. The output for this test can be seen in *Figure 15*. Here, we can see that a translation by a negative value in the y direction causes the object to shift up and to the left. The behavior is opposite of what we would expect, but it is correct, for the same reason as before. In both tests, our line of gaze z_e remains pointing towards the origin of the WCS.

I will now run tests with various translations in the z direction. All other param-

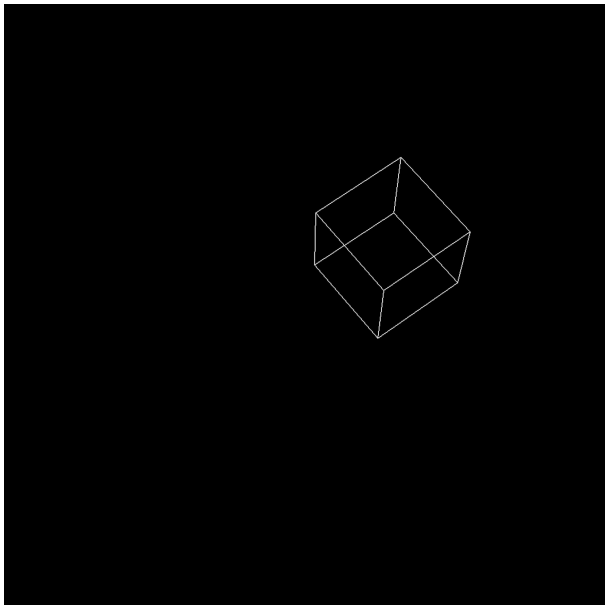

Figure 13: Output of 3D Translation, where $x = -2$, $y = 0$, and $z = 0$.

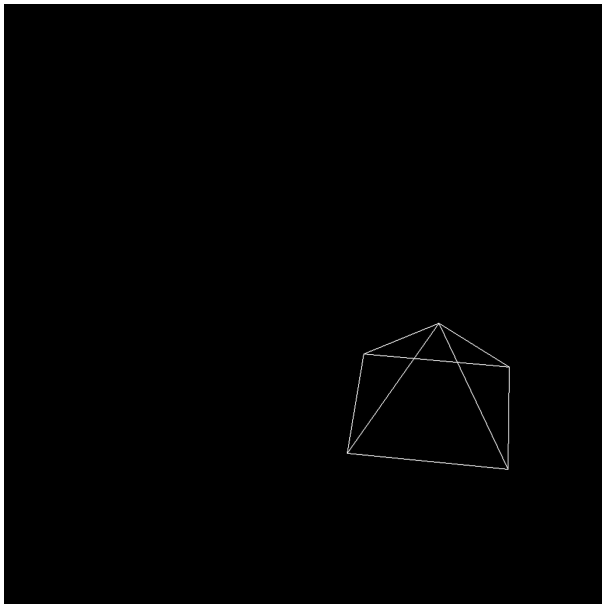

Figure 14: Output of 3D Translation, where $x = 0$, $y = 2$, and $z = 0$.

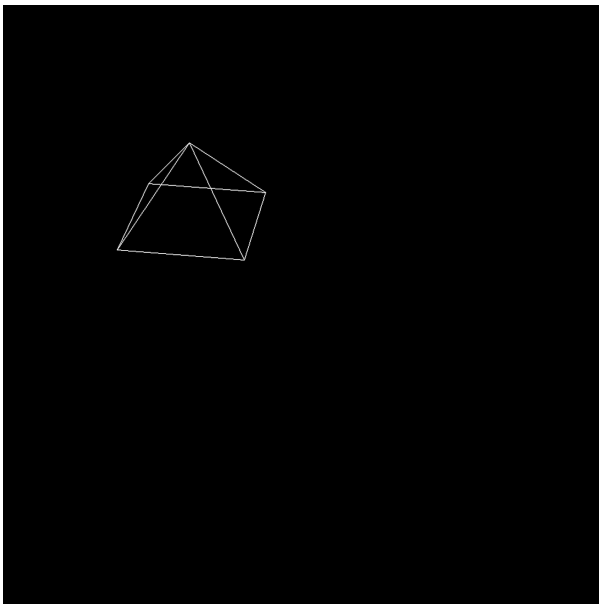

Figure 15: Output of 3D Translation, where $x = 0$, $y = -2$, and $z = 0$.

eters will be kept the same. The rectangular prism will be used to demonstrate this test. First, I will set the z displacement value to 3. The x and y displacement values will be set to 0. The output for this test can be seen in *Figure 16*. Here, we can see that a translation by a positive value in the z direction causes the object to appear larger and shift slightly up. This behavior is correct, and it is due to the perspective projection of the image. Next, I will set the z displacement value to -3. The x and z displacement values will be set to 0. The output for this test can be seen in *Figure 17*. Here, we can see that a translation by a negative value in the z direction causes the object to appear smaller and shift slightly down. The behavior is correct, and it is due to the perspective projection of the image. In both tests, our line of gaze z_e remains pointing towards the origin of the WCS.

4.2 3D Scale

To demonstrate that 3D scale works correctly, I will apply scaling transformations that both enlarge and shrink the cube. First, I will apply a horizontal, vertical, and depth scaling factor of 0.1 to the cube, with the center of scale located at (0, 0, 0). It should be noted that for this test, the *viewpoint* will be set to (6, 8, 7.5), the *screen size* will be set to 30 cm, and d will be set to 60 cm. The output can be seen in *Figure 18*. Here, we can see that applying a scaling factor less than one in each dimension shrinks the object. This behavior is what we would expect. Next, I will apply a horizontal, vertical, and depth scaling factor of 2 to the cube, with the center of scale located at (0, 0, 0). It

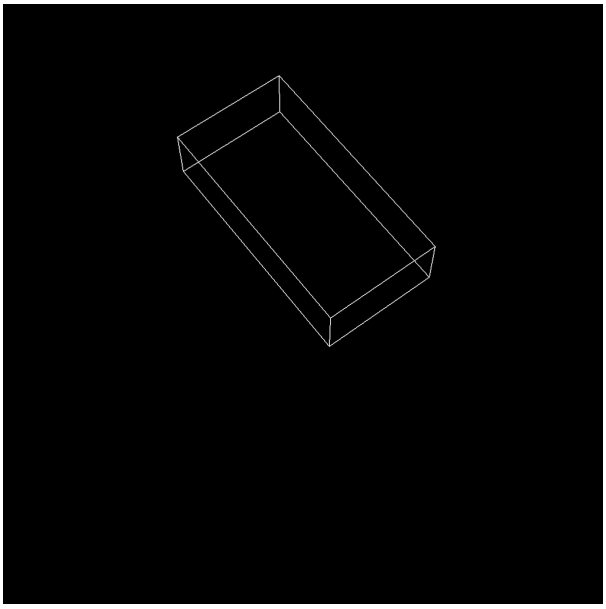

Figure 16: Output of 3D Translation, where $x = 0$, $y = 0$, and $z = 3$.

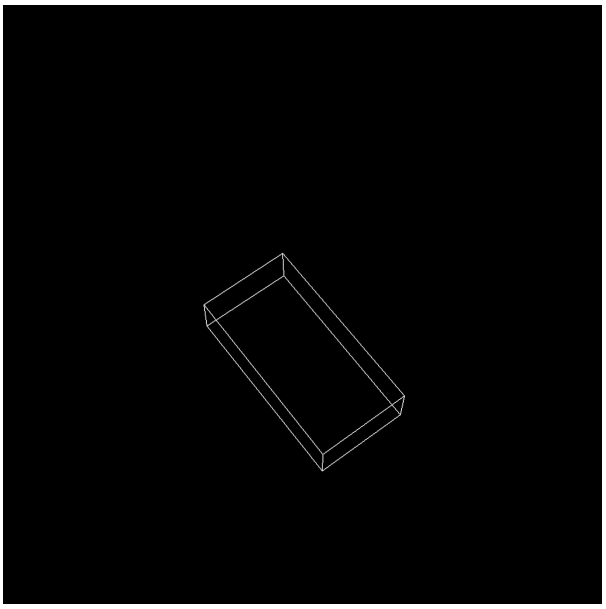

Figure 17: Output of 3D Translation, where $x = 0$, $y = 0$, and $z = -3$.

should be noted that for this test, the *viewpoint* will be set to (6, 8, 20), the *screen size* will be set to 30 cm, and d will be set to 60 cm. The output can be seen in *Figure 19*. Here, we can see that applying a scaling factor greater than one in each dimension enlarges the object. This behavior is also what we would expect.

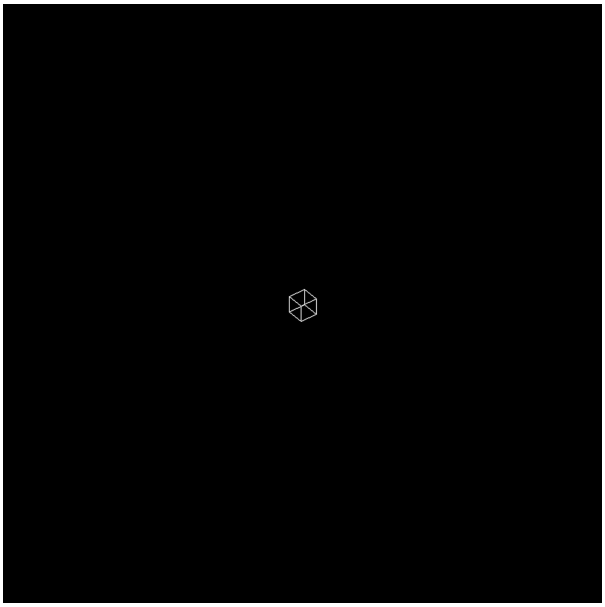

Figure 18: Output of 3D Scale, where a horizontal, vertical, and depth scaling factor of 0.1 is applied, with the center of scale located at (0, 0, 0).

4.3 3D Rotations

To demonstrate that 3D rotation works correctly, I will run tests for each counter-clockwise rotation about the fixed x, y, and z axis. For each test, the *viewpoint* will be set to (6, 8, 20), the *screen size* will be set to 30 cm, and d will be set to 60 cm. I will begin by rotating the cube counter-clockwise about the x-axis. First, I will set the rotation angle to 45 degrees and the center of rotation to (0, 0, 0). The output for this test can be seen in *Figure 20*. Next, I will set the rotation angle to -20 degrees and the center of rotation to (0, 0, 0). The output for this test can be seen in *Figure 21*.

I will now rotate the pyramid counter-clockwise about the fixed y-axis. The rotation angle will be set to to 45 degrees with the center of rotation at (0, 0, 0). All other parameters will be kept the same. The output for this test can be seen in *Figure 22*. Next, the rotation angle will be set to -45 degrees with the center of rotation at (0, 0, 0). Again, all other parameters will be kept the same. The output for this test can be seen in *Figure 23*. It should be noted that the center of pyramid is not located at (0, 0, 0).

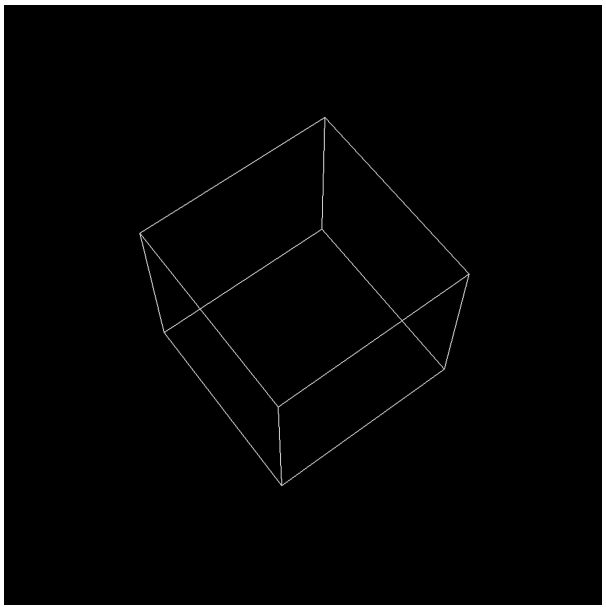

Figure 19: Output of 3D Scale, where a horizontal, vertical, and depth scaling factor of 2 is applied, with the center of scale located at (0, 0, 0).

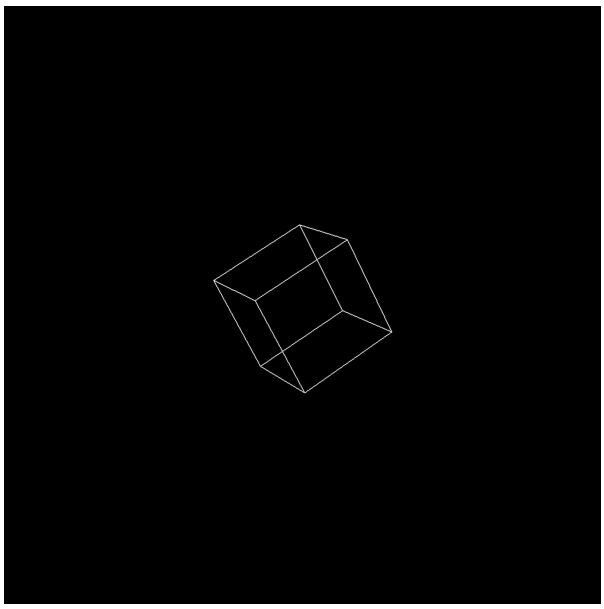

Figure 20: Output of 3D Rotation about the x-axis, with the rotation angle of 45 degrees and the center of rotation located at (0, 0, 0).

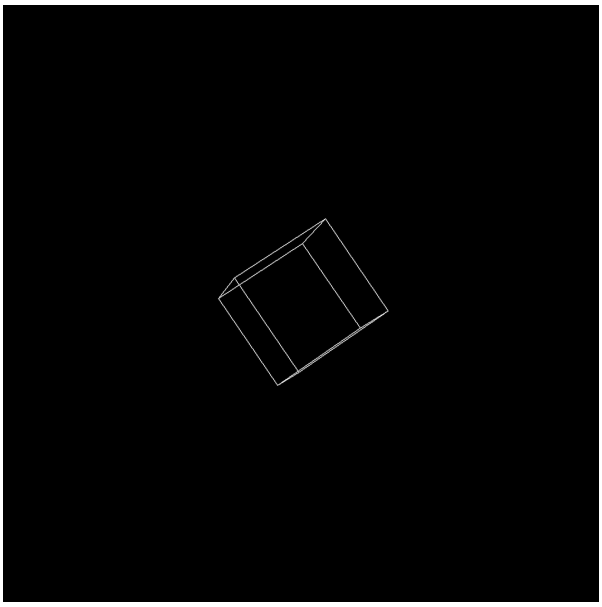

Figure 21: Output of 3D Rotation about the x-axis, with the rotation angle at -20 degrees and the center of rotation located at (0, 0, 0).

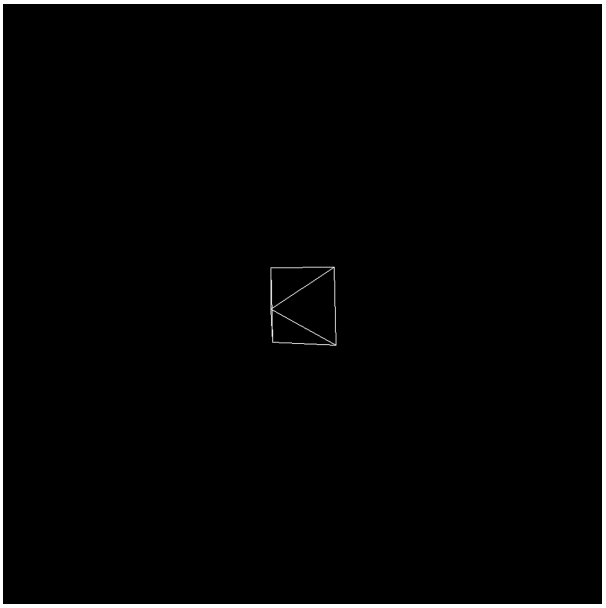

Figure 22: Output of 3D Rotation about the y-axis, with the rotation angle of 45 degrees and the center of rotation located at (0, 0, 0).

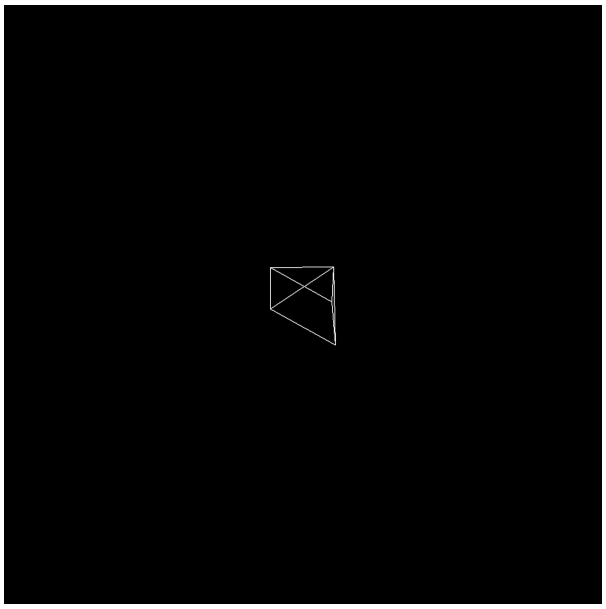

Figure 23: Output of 3D Rotation about the y-axis, with the rotation angle at -45 degrees and the center of rotation located at (0, 0, 0).

I will now rotate the rectangular prism counter-clockwise about the fixed z-axis. The rotation angle will be set to 45 degrees with the center of rotation at (0, 0, 0). All other parameters will be kept the same. The output for this test can be seen in *Figure 24*. Next, the rotation angle will be set to -45 degrees with the center of rotation at (0, 0, 0). Again, all other parameters will be kept the same. The output for this test can be seen in *Figure 25*. It should be noted that the center of rectangular prism is not located at (0, 0, 0).

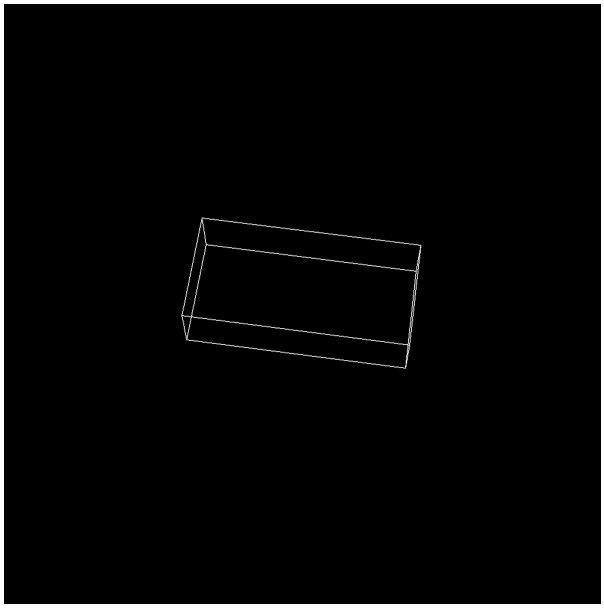

Figure 24: Output of 3D Rotation about the z-axis, with the rotation angle of 45 degrees and the center of rotation located at (0, 0, 0).

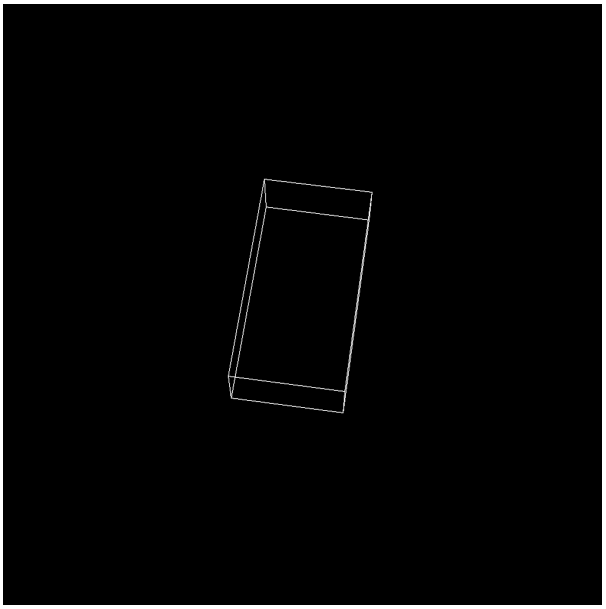

Figure 25: Output of 3D Rotation about the z-axis, with the rotation angle at -45 degrees and the center of rotation located at (0, 0, 0).