## CA1 Implementation Report

**Introduction**

The purpose of this CA is to create a 3D wireframe scene based on the theme ‘Start with Nothing’ using wireframe models and the processing language. I had to make 4 different shapes for my scene with one being static, and the other 3 each displaying a different form of transformation. For my scene, I chose a simple player model for my static shape. My other shapes include a translating gun, a rotating sign, and a scaling cube enemy.

A screenshot of a computer code

Description automatically generated

Figure Snippet of plotted points code

To start, I plotted out every point for the 2D versions of my shapes using float variables. Once the 2D versions are complete, making the 3D points is as simple as copy pasting all the points of a shape and changing their Z values. I separated the two sets of points with differing z values with comments as seen in figure 1 on lines 54 and 62. This process ended out with a total of 78 different points.

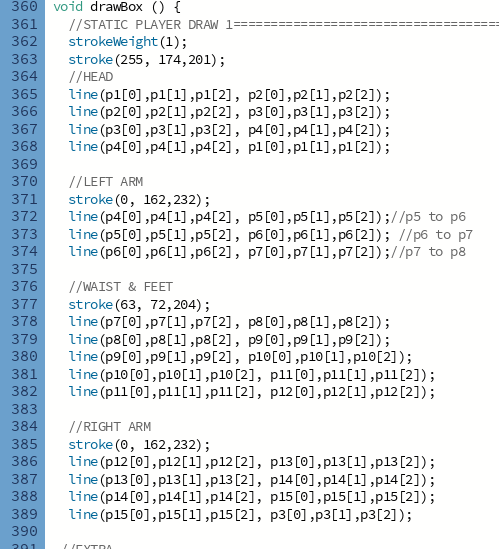


Figure Joining points.

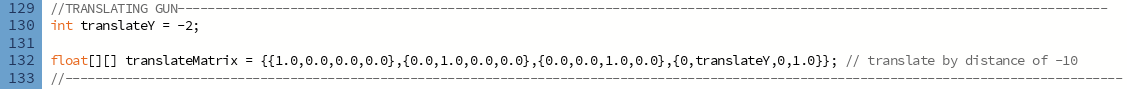
The next step was to join all these points which is done in my drawBox method. I also separated the line joinings with comments which made it easier to add color to specific parts of the shapes afterwards.

A screenshot of a computer

Description automatically generated

Figure Points plotted and joined only

After all the points are plotted and appropriately joined to one another, figure 3 shows that result.



With a base now having been drawn, all that was left was to apply my transforms where necessary. The first transform I did was translating for my gun shape. In my translate matrix on line 132, the first 3 sets of numbers are left mostly 0, with x, y, and z in each set being set to 1.0 as to not change them in any way. The fourth set of numbers also has zeroes except for the y value. The Y value is instead equal to a variable I declared just above that on line 130 called ‘int translateY’. This variable is equal to -2 so shape will be translated by -2 on only its Y value every time it loops. The reason I have the Y value as a variable is because I will be changing this value.

A screenshot of a computer code

Description automatically generated

Figure Translate Loop

In my draw function, I have an if statement to check the value Y value of the lowest Y-valued point on my gun shape. If that Y value because less than 40, the translate variable is set to 2. If its greater than 80, its set to -2. Then on line 190, the translate value is set to equal the Y position for the translation matrix. This effectively loops the gun translation and makes then gun go up and down.

A close-up of a text

Description automatically generated

The next shape I moved onto was my sign shape. This is my shape that rotates. The first matrix on line 136 makes the shape translate back to point 0,0 on the axis. This is due to the fourth set of numbers in the matrix. The reason the numbers are -50,-55,-5 is because I had to get the centre coordinate for the x, y, and z values. Taking away the centre point values from x, y, and z translates the shape to 0,0. The second matrix on line 137 is done to bring the shape back to its original position. This is much the same as the first matrix except the previous negative values are set to positive. Next, a float called theta on line 138 is made which is equal to 10 radians. This is used in the rotYMatrix on line 141. Theta is incorporated into the matrix for the x and the z value. The values representing Y are left at 0 because when the shape is rotating around the Y axis, these values remain as 0.

A screenshot of a computer code

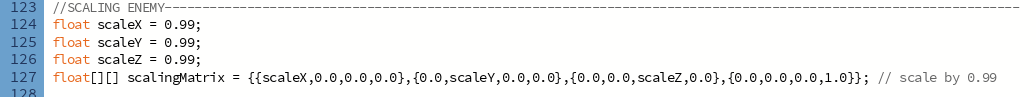
Description automatically generated

For my code to apply transforms, I have float variables for each transform; m is for scaling, n is for translating, and z is for rotating. These variables are made equal to the appropriate matrices. Specifically for the rotation matrix, z is made equal the translateToOrigin matrix to move the shape’s position to 0,0. Then z is made equal rotYMatrix, which rotates the shape. Finally, z is made equal translateBackFromOrigin which moves the shape back to its original position.

A screenshot of a computer program

Description automatically generated

Each variable m, n, and z are all sent to a different method which will handle 4 elements and its 3D calculations. A explain of how this works is that in the image, the first number of the p array, p\_new[0], is used to hold the new value of x after it has been transformed. This applies to the y value of p\_new[1], and the z value of p\_new[2]. P\_new is then returned as the new position of the shape after the transform. I made a new method for all 3 transforms although I believe I could have gotten the same final result for my project ad I only done one method to handle the 3D calculations.



The last shape to be transformed is my enemy shape which will be scaled. I started by changing the appropriate x, y, and z values in the matrix into variables which hold a value of 0.99. This means the shape will be scaled by 0.99.

A screenshot of a computer program

Description automatically generated

In my draw function, I track one of the points on my enemy. If the Y value of that point becomes less than 50, the scaleX, Y, and Z all get 0.0099 added to them. This makes the shape increase in size. Also, if the Y value of that point becomes greater than 80, the values have 0.0099 taken away from them, decreasing the size of the shape. After the if statements, the changed x, y, and z values are set to the newly changed values. This essentially makes the shape increase in size to a certain point and decrease in size to a certain point.

A screenshot of a computer code

Description automatically generated

The applyTransform method then takes place which applies the new changes of the coordinates to all the points of my shapes. In the image above, p33 through p44 use the transform\_pointScale method which passes in the value on n and the related points. The same is done for m and z where they are passed into the appropriate transform methods.



This matrix here works that the x, y, and z values are scaling factors. These values are less than 1, which means that the points of this matrix will be scaled down by 97% in all three dimensions. This is what applies the scaling transformation to the shape.

A white background with black text

Description automatically generated

The rotation transformation is a bit more complicated. It first scales the shape up by a factor of 1.1 in the x, y, and z, direction, increasing it by 10%. The rotateZMatrix makes the object rotate around the z axis. The shape rotates in the x and y plane at an angle according to the variable ‘theta’ which is stated above as being 10 radians. The z value is left at 1.0 which means it is unchanged.

A close-up of a computer code

Description automatically generated

Inside the setup class is another matrix. This matrix overrides the previous matrix. The previous matrix rotated at 10 radians where as this matrix changes the values to rotate at 2 radians. The rotation transformation is the more complicated of the 3 with the inclusion of the theta variable and the radians.

The user can view the rotation from different angles using the mouse with the help of the peasy cam library.

A close-up of words

Description automatically generated

The user can view the rotation from different angles using the mouse with the help of the peasy cam library. At the top of the code, the peasy library is imported and a peasyCam variable is made called cam.

A close-up of a computer code

Description automatically generated

Then in the setup method, the cam variable is made equal a new peasyCam object . The values set are related to the minimum and maximum distance of the camera from the shape drawn. The user can then change the view of the rotation by clicking and dragging the mouse around the screen.

A black and white image of a sign

Description automatically generated

This is the final result.

**Conclusion**

Having completed 8 labs before this project, I was very prepared upon starting. The longest part of this project was having to plot out all 78 of my different points and then join the correct points to one another. The only difficulties I had with this project were making the scaling and translating shapes loop by increasing and decreasing their values. To solve this problem, I used debug statements to check what values were changing while the shape was transforming. Once I knew what values were changing and how they were changing, I could make appropriate if statements to make a max and minimum range of the shape’s points can increase or decrease to during its transformation. Like I stated earlier, the project could have been shortened down had I used only one transform method to handle 3D calculations rather than making 3 separate methods for the 3 different transforms.