**Three-Dimensional Graphics**

This is a very optional, unnecessarily advanced lesson. Please do not go any further unless you have finished all other projects.

This lesson looks at building rudimentary 3D images on a 2D plane.

In this lesson you will:

* Create a function that converts a 3D point (x,y,z) into 2D (x,y)
* Create an array of 3D points to represent a cube in 3D (8 vertices).
* Draw the cube on the screen
* Create a function that rotates a 3D point about three axes.
* Write a pygame loop that animates/rotates the cube using a timer and keystrokes.

We start by importing the math and pygame packages:

import math

import pygame

The math package is needed for the trigonometry we will need to do. We will be using the draw methods a lot, so let's import them as well:

from pygame.draw import \*

Set up our screen size:

screen\_width = 640

screen\_height = 520

size = screen\_width, screen\_height # tuple for the screen size

screen = pygame.display.set\_mode(size) # set the dimensions of screen

Now we define some functions we will need. The first function changes the coordinates so that (0,0) is no longer at the top left corner, but is moved to the center of the screen.

# Functions go here:

def center(x,y):

# converts normal screen format (origin at top left corner)

# to cartesian (origin at center of screen)

a = int (x + screen\_width/2) # shift right half a screen

b = int(screen\_height/2 - y) # shift up half a screen

return (a,b )

Now our rotating cube will be centred at the origin - which will be in the centre of your screen instead of up at the top left corner of the screen where it defaults.

**Converting from 3D to 2D**

Now we write the most important function called **convert().** This function converts a point from 3D to 2D. Unfortunately, it is empty right now.

def convert (x,y,z):

# converts a point in 3-space to a point in 2-space

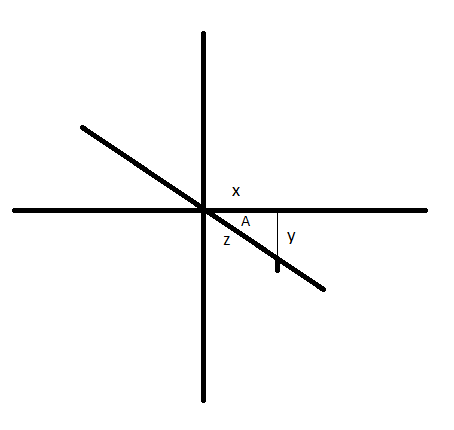
# put stuff in here!

return (x,y)

How do we convert?

To appear “3D” we want the Z-axis to appear to come out of the page. The Z-axis is what makes a picture look 3D.

For this exercise, we give the viewer a “birds-eye” point of view, slightly above and to the left of the grid. To make this effect, we draw the z axis as a line going diagonally down and to the right. Here is a representation of the point (0,0,50).



You can see the point as a little tick mark on the Z axis. In 2D it looks like the point is (x, -y). If A is the angle between the X axis and the Z axis, then you can figure out the value of x and y using trigonometry.

In Pygame, the trig formulas are:

math.sin(A), math.cos(A) and math.tan(A)

We can set the angle A to whatever perspective we want. **Note: A is an angle in radians, not degrees**. Radians are in fractions of 2\*pi . To convert:

Angle in radians = Angle in degrees \* 2 \* pi / 360

(or, simplified, Angle in degrees \* pi/180)

To get pi, you can write 3.1415 or math.pi.

If A = 0 then the viewer is looking straight down the Z axis. In the picture above, A is about 30 degrees (or pi/6 radians).

The picture above shows the 3D point (0,0,Z). However, in two dimensions, it looks like a triangle. The horizontal side of the triangle is x. The vertical side is y. The hypotenuse is z. To figure out what x and y values are:

x = z \* cos(angle)

y = -z \* cos(angle) (negative because it goes down the y axis)

In your convert function, we do essentially the same as above. Add this to your code (where it says “Put stuff here”)

A = math.pi/6

x = x + z\*math.cos(A)

y = y - z\*math.sin(A)

**The Cube Array**

Let's create a cube. First we create an array of vertices for our cube. Here is the first vertex:

# Main Program Starts Here (after your functions):

cube = [] # create an empty array of points

a = 50

b = 50

c = 50 # create x, y and z coord.

(x,y) = convert (a,b,c) # convert to 2D point

cube.append(center(x,y)) # add to our cube list of points

# create rest of cube yourself !

This adds one point to our array. You must create the other 7 points yourself. **Don't do this now.**  For now, let's just draw the screen.

Create a black screen and draw some Cartesian axes in green.

# Main Program Starts Here:

green = 50, 255, 50 # some RGB tuples

white = 255,255,255

black = 0,0,0

screen.fill(black) # start with a black background

line (screen, green, (0,screen\_height/2), (screen\_width, screen\_height/2))

line (screen, green, (screen\_width/2, 0), (screen\_width/2, screen\_height))

pygame.display.flip() # flip screen on to window

Try running our program so far and making sure it works. You should see two green lines.

Now let's add our vertex. We will draw a small circle to represent the corner of the cube. Insert the line in black in the place indicated:

line (screen, green, (0,screen\_height/2), (screen\_width, screen\_height/2))

line (screen, green, (screen\_width/2, 0), (screen\_width/2, screen\_height))

circle(screen, white, cube[0], 3, 1) # draw a small circle.

pygame.display.flip() # flip screen on to window

The circle function looks like this:

circle (screen, colour, center, radius, line width)

In our case above, the center of the circle is the vertex of the cube.

**Exercise:**

Create all 8 vertices of the cube, convert and draw them.

**Rotating our Cube**

You don't really get much of a perspective unless you rotate the cube. Add this new function to the function section near the top of your code.

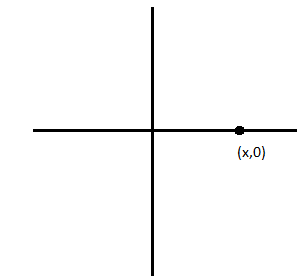
def rotate (point3D, xRot, yRot, zRot ):

# rotates a 3D point (x,y,z) about each axis

# put stuff in here!

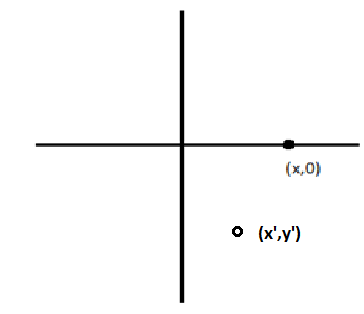
return (x, y, z)

To understand rotation, we'll take a simple 2D example. First we start with a point on the x axis:



Think of this being like a clock pointing to “3 O’Clock”. The arrow is X units long (this is the radius). Now suppose we rotate the hand clockwise. This would be rotating it about the Z axis (the Z axis is sticking straight out at you).

The point is rotated an angle A, and ends up in a new position, say “5 O’Clock”:



The point moves in a circle. The radius of the circle is still X. But now our point is at a new value of x we’ll call x’. This means, our new coordinates are given by:

x' = +x cos( A)

y' = **-**x sin (A)

But what if the point starts on the y axis?





Rotating it the same way, we end up with this:









Now the original length (y) becomes our radius, and the new values of x and y are:

x' = y sin(A)

y' = y cos(A)

Putting both sets of equations together, if your point is at any point (x,y) and we wish to rotated it by an angle A, the new coordinates would be :

**x' = x cos(A) + y sin(A)**

**y' = y cos(A) - x sin(A)**

The above only rotates our cube about the Z axis. You would need to write a separate version of these transformation equations to rotate about the X and Y axes.

**Exercises:**

1. Adapt the above formulas to your rotate() function and see if it works by calling it in your main program.
2. Create the formulas to rotate about the X axis and Y axis. Verify that they work.
3. Create a game loop that rotates the cube in any direction automatically.
4. Use the keyboard to control the direction of rotation.
5. Use the line function to draw the cube as a wireframe (lines between points instead of just points).