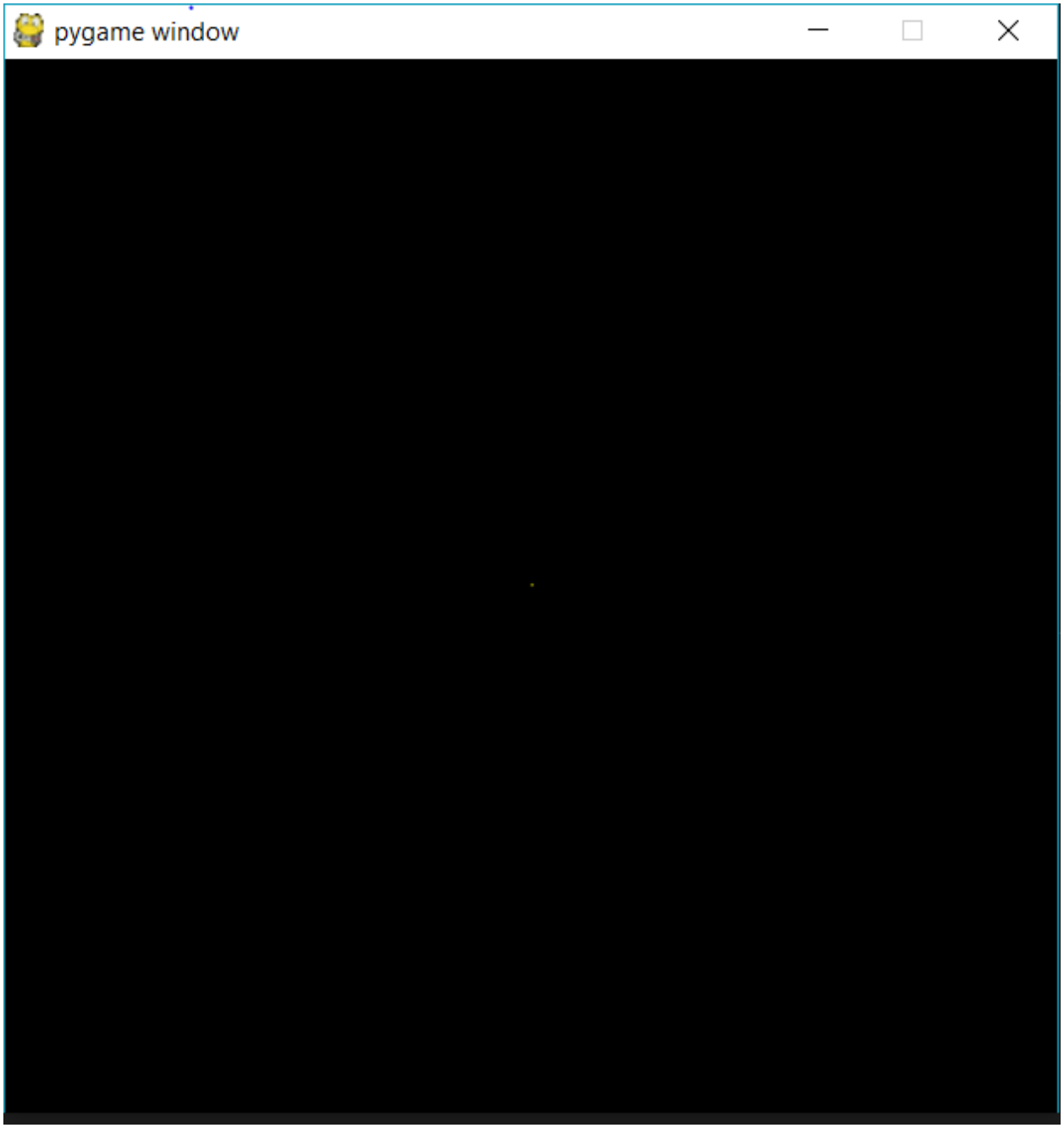
Basic Drawing

Get the basic.py
Run the python file. You will see the window below.



Now let’s understand what this simple code says.

1. **Importing statements**

```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
```

The first and the second lines import pygame and its functions. We are not definitely working on pygame. But we would like to use its window management. OpenGL draws but it doesn’t have the window which it can display the images on. Then we have the from OpenGL.GL import * line which basically imports all OpenGL Functions. These are functions that directly act on drawing. from OpenGL.GLU import * imports utility functions that help OpenGL in some way. GLU is Graphics Library Utilities.

2. Initialization

```
def init():
pygame.init()
display = (500, 500)
pygame.display.set_mode(display, DOUBLEBUF|OPENGL)
glClearColor(0.0, 0.0, 0.0, 1.0)
gluOrtho2D(-1.0, 1.0, -1.0, 1.0)
```

pygame.init() initializes a pygame window which displays what we draw using OpenGL display = (500, 500) makes our pygame window width and height 500 pixels each. pygame.display.set_mode(display, DOUBLEBUF|OPENGL) this function tells pygame that we will be displaying graphics made with OpenGL. DUBLEBUF is a ‘short’ version for double buffer which tells how the screen should be refreshed when

displaying OpenGL graphics. glClearColor(0.0, 0.0, 0.0, 1.0) This will set our window black. It takes four parameters which represent color. RGBA – Red, Green, Blue and Alpha. We can have any color that we want combining Red, Green and Blue. Alpha tells the window how much an object is transparent. 0.0 means 100% transparent and 1.0 means 100% opaque. gluOrtho2D(-1.0, 1.0, -1.0, 1.0) This sets the OpenGL that we are working on 2D space and the coordinates are from -1.0 to 1.0 on X-axis and same for Y-axis

3. **Let's jump to main function and we will get back to draw() function.**

```
def main():
    init()
    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
        draw()
        pygame.display.flip()
        pygame.time.wait(10)
    main()
```

On the first line we called the init() function which will initialize our window.
A window only displays something until displaying is over and closes soon after. One way of keeping the window open is creating an endless while loop and forcing it to stay. The while loop ensures that.
When we want to close a window, we click the close button. That clicking event is sensed in the for-loop statement.

Finally, there is the draw() function, which draws what we want.
pygame.display.flip() simply updates our window.
pygame.time.wait(10) delays execution of our program in 10 seconds, which defines how often we update the window.

4. **Draw Function**

```
def draw():
     glClear(GL_COLOR_BUFFER_BIT)
     glColor3f(1.0, 1.0, 0.0)
     glBegin(GL_POINTS)

     glVertex2f(0.0, 0.0)
     glEnd()
     glFlush()
```

Finally, what we were waiting for, the drawing function.
glClear(GL_COLOR_BUFFER_BIT) clears the window with the color you set in init() function
glColor3f(1.0, 1.0, 0.0) is a function that sets color for what is drawn next. The 3f is a reminder that the function takes 3 floating points, Red, Green, and Blue. In this case, we gave it a full red color and none for the others.
glBegin(GL_POINTS) tells OpenGL that we are drawing points and to start drawing them.
glVertex2f(0.0, 0.0) tells OpenGL that the point we draw is on the location of (0, 0)
glEnd() tells OpenGL that there is no more drawing
glFlush() tells OpenGL that the above commands to be executed as quickly as possible on one run without executing other drawing commands.

Now… Let's get to work

1.

Now let’s make our point a little bigger to see. Add **glPointSize** just before **glBegin**. Now the point is 10X bigger.

```
...
glColor3f(1.0, 0.0, 0.0)
glPointSize(10)
glBegin(GL_POINTS)
```

2.

Let’s add some more points.

```
glBegin(GL_POINTS)
glVertex2f(0.5, 0.0)
glVertex2f(-0.5, 0.0)
glVertex2f(0.0, 0.5)
glEnd()
```

3.

That’s great but, why would I care about many points?
Well, let’s change the **glBegin** function a bit.

```
glBegin(GL_LINES)
```

This tells OpenGL that we intend to draw lines instead of points.
Here is our line. But wait, where is the third point?
glBegin(GL_LINES) tells OpenGL to draw a line on every pair of consecutive points. Being alone, the last point is left alone.
Try glBegin(GL_LINE_STRIP) and glBegin(GL_LINE_LOOP)

Question 1. What did you get in both case?

Using Numpy

It's lame to manually give our graphics system the vertices that we need. We know better, let’s use Numpy.
Let’s draw Y = X2
1. Generate numbers with numpy for X and Y and give every pair of X and Y to OpenGL

```
def draw():
     glClear(GL_COLOR_BUFFER_BIT)
     glColor3f(1.0, 0.0, 0.0)
     # generate 100 points within -1 to 1 range
     x = np.linspace(-1, 1, 100)
     # rais every point in x by 2
     y = np.power(x, 2) glPointSize(10)
     glBegin(GL_LINE_STRIP)
     # for every pair (a, b) of the numbers in x, y
     for a, b in zip(x, y):
          # give (a, b) to OpenGL to draw
          glVertex2f(a, b)
     glEnd()
     glFlush()
```

Question 2: Try to change the function to log x.