## Pygame tutorial #2: drawing lines

Welcome to part 2 in my [pygame](http://pygame.org/) tutorial. This time we will build upon what you learned last time as we start drawing onto surfaces. We will continue to move very slowly so that everybody gets a chance to learn.

**Drawing a line**

You draw a line by using pygame.draw.line. You can also use pygame.draw.aaline which draws an anti-aliased line. Using anti-aliasing can make the line appear less jagged in some cases at the expense of the function call being much slower.

To draw a blue line from *(0, 0)* to *(200, 100)* (note: we are measuring in pixels) onto the surface *screen* you do:

pygame.draw.line(screen, (0, 0, 255), (0, 0), (200, 100))

You could also do this:

blue = 0, 0, 255  
point1 = 0, 0  
point2 = 200, 100  
pygame.draw.line(screen, blue, point1, point2)

This is will do exactly the same thing as the previous example, but is (possibly) more readable. Let’s try to put this together to a little program. We will draw two diagonal lines: one going from the top left-hand corner to the bottom right-hand corner and one from the top right-hand corner to the bottom left-hand corner. We will use pygame.draw.line for one of the lines and pygame.draw.aaline for the other.

3 import pygame

4

5 screen = pygame.display.set\_mode((640, 480))

6 running = 1

7

8 **while** running:

9 event = pygame.event.poll()

10 **if** event.type == pygame.QUIT:

11 running = 0

12

13 screen.fill((0, 0, 0))

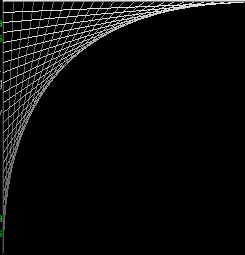
14 pygame.draw.line(screen, (0, 0, 255), (0, 0), (639, 479))

15 pygame.draw.aaline(screen, (0, 0, 255), (639, 0), (0, 479))

16 pygame.display.flip()

I hope you recognize most of the code from the first tutorial. If you run the program you should see a big blue ‘x’ across the window.

**Exercises**

1. Improve the program by getting rid of “magic numbers”. Declare two variables width and height and initialize them to the width and the height of the window. Then modify the rest of the program to use these variables instead of actual numbers. To verify that you have made all the required changes, try setting width and height to different values and confirm that you still get two diagonal lines and that each line’s end points are at one of the corners of the window.
2. Improve the program further by declaring variables called linecolor, topleft, bottomright and so on. Modify the program to make use of the variables.
3. Modify the program to draw the two lines in different colors.
4. Write a program that instead draws one horizontal and one vertical line. The lines should both be centered.
5. Write a program that draws four lines: the diagonal lines as in the example as well has the horizontal and vertical lines as in the last exercise. Each line should be in a different color. The program should take the height and width of the window as command-line arguments.
6. Write a program that draws the following pattern:  
     
   Once you have solved it for the upper left hand corner, repeat it so that the same pattern is drawn in all four corners of the screen. You should be able to draw all in all corners with a single loop and four calls to pygame.draw.line.

**Moving things around**

Now that we know how to draw lines on the screen with pygame, let’s start moving them around. Moving lines around is simple. we store the line coordinates in variables. Inside the event loop we modify the values of those variables. Let’s draw a line that jumps up and down.

3 import pygame

4

5 y = 0

6 dir = 1

7 running = 1

8 width = 800

9 height = 600

10 screen = pygame.display.set\_mode((width, height))

11 linecolor = 255, 0, 0

12 bgcolor = 0, 0, 0

13

14 **while** running:

15 event = pygame.event.poll()

16 **if** event.type == pygame.QUIT:

17 running = 0

18

19 screen.fill(bgcolor)

20 pygame.draw.line(screen, linecolor, (0, y), (width-1, y))

21

22 y += dir

23 **if** y == 0 **or** y == height-1: dir \*= -1

24

25 pygame.display.flip()

There we go. The y-position of the line is determined by the variable y which increased by dir in each iteration of the event loop. The value of dir is 1 when the line is moving downwards, or -1 when the line is moving upwards. Simple, isn’t it?

**Exercises**

1. Comment out the line screen.fill(bgcolor) and run the program. What happens?
2. Extend the program to also include a vertical line that moves across the screen.

**Drawing a color bar**

Our final example for today will draw a single color bar (“copper bar” if you’ve ever owned an Amiga). Just like our last example, this one will also jump.

3 import pygame

4

5 y = 0

6 dir = 1

7 running = 1

8 barheight = 124

9 screen = pygame.display.set\_mode((800, 600));

10

11 barcolor = []

12 **for** i **in** range(1, 63):

13 barcolor.append((0, 0, i\*4))

14 **for** i **in** range(1, 63):

15 barcolor.append((0, 0, 255 - i\*4))

16

17 **while** running:

18 event = pygame.event.poll()

19 **if** event.type == pygame.QUIT:

20 running = 0

21

22 screen.fill((0, 0, 0))

23 **for** i **in** range(0, barheight):

24 pygame.draw.line(screen, barcolor[i], (0, y+i), (799, y+i))

25

26 y += dir

27 **if** y + barheight > 599 **or** y < 0:

28 dir \*= -1

29

30 pygame.display.flip()

In this example I create an array called colorbar (for the lack of a better name). This will hold values for the colorbar as it shifts from black to bright blue and back to black. Keep in mind that a color is composed of red, green and blue. Each one of these can be a value between 0 and 255.

If I change the blue value by one for each new line, I would get a really smooth gradient bar. But, the height would be 256 pixels from black to blue and another 256 pixels from blue back to black = 512 pixels, which is too high. There would hardly be enough space in our window to see the bar moving up and down.

So I have to decide on a bar height that I find acceptable. I settled on 124, which means I have 62 pixels from the black to blue gradient and another 62 for the blue to black gradient. This also means that I have to move from a blue value of 1 to a blue value of 255 in 62 pixels. The change in blue per line must be 4 per line.

I use two for loops to populate the barcolor array. The first loop pushes increasing values of blue to the array and the second one decreasing values of blue. It is important to notice that this array is nothing other than a lookup. It doesn’t contain the bar as you see it on the screen. It just contains color values.

Run it and see what it looks like. Once you’ve seen it run, analyze the code and make sure you understand *exactly* how it works. Then start experimenting with it.

**Exercises**

1. The color bar example code is horrible. Full of “magic numbers” everywhere! Fix it.
2. Try using pygame.draw.aaline for drawing the lines. Run the program. Do you notice any difference in speed?
3. Change the color of the color bar.
4. Write a program that draws three color bars in different colors. Take note of the height per bar, so that all three can fit and have some space for movement.

**Conclusion**

In this tutorial you have learned to draw lines. We will be staying with methods of drawing to the screen a little while longer, before we move on to other interesting things like loading and displaying images. In the next tutorial you will also learn a bit about mouse events.