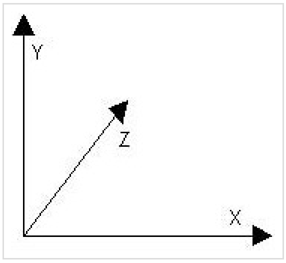
Simulation of 3D Point Rotation with Python and Pygame

Posted by lefam on April 20, 2011 Tweet 1 赞 7

In this tutorial I will show you how to build a simulation of 3D point rotation using Python and Pygame. I will begin by giving you a brief background on 3d computer graphics theory. It will help you understand the code that will be presented afterwards. I assume you have a basic knowledge of Pygame. If you are new to Pygame consider reading first the article A Fast Introduction to Pygame. The video below shows the simulation in action.

Background

To make 3D simulations, first of all, we must describe the objects in 3D space. The most common way of describing 3D objects is through the use of a 3D cartesian coordinate system. We will use the system shown in the figure below, where Y increases going up, X increases going right, and Z increases going into the screen. This is called a *left handed system*. An alternative system, is the *right handed system* where Z increases in the opposite direction.



Next, with points defined in 3D cartesian coordinate system, we can apply transformations such as translation, rotation, and scaling. Note that unlike 2D rotations that occur in one plane, rotations in 3D space occur along an arbitrary axis. Below I present the formulas for 3D rotation along the X, Y, and Z axes.

```
Rotation along X:
y' = y*cos(a) - z*sin(a)
z' = y*sin(a) + z*cos(a)
x' = x

Rotation along Y:
z' = z*cos(a) - x*sin(a)
x' = z*sin(a) + x*cos(a)
```

Follow Me:

Recent Posts
- HTML5 Canvas Crazy Balls
- First Android App: Touch Controlled Cube on Samsung Galaxy S2
- 5 Cool OpenGL Demos in VB.NET
- 3D Starfield in HTML5 Canvas
- A Simple 3D Room made with DirectX and C++

Categories
- .NET
- Android
- Arduino
- C++
- Computer Graphics
- DirectX
- Electronics
- Game Programming
- HTML5
- JavaScript
- Lab
- OpenGL
- PHP
- Programming
- Pygame
- Python
- VB.NET
- Web Development

Archives
- January 2012
- October 2011
- July 2011
- June 2011
- May 2011
- April 2011

Tags
3d 3d engine 4017 android arduino breakout c++ css directx experiment game loop game programming GD GDI+ html html5 demo javascript LED OpenGL parallax particles php php tips potentiometer pygame python samsung galaxy s2 simulation starfield vb.net visual basic

```
y' = y

Rotation along Z:
x' = x*cos(a) - y*sin(a)
y' = x*sin(a) + y*cos(a)
z' = z
```

When we are finished applying the transformations, we must draw the 3D object into the screen. But, since the screen is 2D we will have to convert the 3D coordinates to 2D coordinates. This operation is called *3D projection*, and there are many ways of doing it. I will cover just the common one that is the *perspective projection.* Below I present the formula.

```
3D Perspective Projection
x' = x * fov / (z + viewer_distance) + half_screen_width
y' = -y * fov / (z + viewer_distance) + half_screen_height
z' -> for now, z is useless
```

In the formula, *fov* is a constant value that defines the "field of vision". I like to use values between 128 and 256. *view_distance* is the distance from the object to the viewer.

With all the points converted to 2D space, we can finally draw the object using Pygame 2D drawing functions.

The Code

I present below the code to simulate the rotation of 8 points/vertices of a cube. To start with, let's define the Point3D class to represent points in 3D space:

```python
import sys, math, pygame

class Point3D:
    def __init__(self, x = 0, y = 0, z = 0):
        self.x, self.y, self.z = float(x), float(y), float(z)

    def rotateX(self, angle):
        """ Rotates the point around the X axis by the given angle in degrees. ""
        rad = angle * math.pi / 180
        cosa = math.cos(rad)
        sina = math.sin(rad)
        y = self.y * cosa - self.z * sina
        z = self.y * sina + self.z * cosa
        return Point3D(self.x, y, z)

    def rotateY(self, angle):
        """ Rotates the point around the Y axis by the given angle in degrees. ""
        rad = angle * math.pi / 180
        cosa = math.cos(rad)
        sina = math.sin(rad)
        z = self.z * cosa - self.x * sina
        x = self.z * sina + self.x * cosa
        return Point3D(x, self.y, z)

    def rotateZ(self, angle):
        """ Rotates the point around the Z axis by the given angle in degrees. ""
        rad = angle * math.pi / 180
        cosa = math.cos(rad)
        sina = math.sin(rad)
        x = self.x * cosa - self.y * sina
        y = self.x * sina + self.y * cosa
        return Point3D(x, y, self.z)

    def project(self, win_width, win_height, fov, viewer_distance):
        """ Transforms this 3D point to 2D using a perspective projection. """
        factor = fov / (viewer_distance + self.z)
        x = self.x * factor + win_width / 2
        y = -self.y * factor + win_height / 2
        return Point3D(x, y, 1)
```

First,we have the constructor that initializes instances of the class. Next, we have methods to rotate a point around X, Y, and Z axes. These functions return the rotated point. Finally, we have the function that projects a point from 3D to 2D space.

Next, we define the Simulation class that will do the rest of the job:

```python
class Simulation:
    def __init__(self, win_width = 640, win_height = 480):
        pygame.init()

        self.screen = pygame.display.set_mode((win_width, win_height))
        pygame.display.set_caption("Simulation of 3D Point Rotation (http://codent

        self.clock = pygame.time.Clock()

        self.vertices = [
            Point3D(-1,1,-1),
            Point3D(1,1,-1),
            Point3D(1,-1,-1),
            Point3D(-1,-1,-1),
            Point3D(-1,1,1),
            Point3D(1,1,1),
            Point3D(1,-1,1),
            Point3D(-1,-1,1)
        ]

        self.angleX, self.angleY, self.angleZ = 0, 0, 0

    def run(self):
        while 1:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    sys.exit()

            self.clock.tick(50)
            self.screen.fill((0,0,0))

            for v in self.vertices:
                # Rotate the point around X axis, then around Y axis, and finally
                r = v.rotateX(self.angleX).rotateY(self.angleY).rotateZ(self.angle
                # Transform the point from 3D to 2D
                p = r.project(self.screen.get_width(), self.screen.get_height(),
                x, y = int(p.x), int(p.y)
                self.screen.fill((255,255,255),(x,y,2,2))

            self.angleX += 1
            self.angleY += 1
            self.angleZ += 1

            pygame.display.flip()

if __name__ == "__main__":
    Simulation().run()
```

When this class is instantiated, it starts with initializing Pygame, and then setting up a window. Next, we create a clock object to lock the frame rate of the animation. Then, we define 8 points in 3D space. After that, we define the variables that will hold angles of rotation around each axis.

The *run()* function runs the main loop where the points are rotated and drawn. First, it checks if a QUIT event is pending, and if so quits the application. Then, it locks the frame rate to 50 FPS, and after that clears the screen. Then, it traverses the vertex list, rotating each one around X, Y, and Z axes. Then, it projects the point, and draws it into the screen. After traversing the vertex list, finally, it increases the angle variables, and then updates the screen.

The last line of code runs the simulation.

Click here to download the full source code.

Conclusion

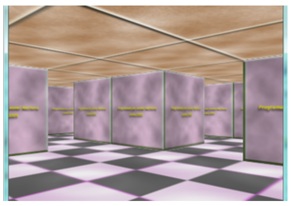
With a little bit of 3d computer graphics theory, we were able to make a simple 3d simulation using Pygame. In the next tutorial we will expand on the theory and code from this tutorial to make a rotating wireframe cube.

To stay updated, make sure you subscribe to this blog, follow us on twitter, or follow us on facebook.

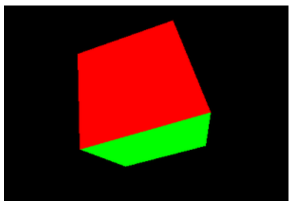
Share the knowledge!

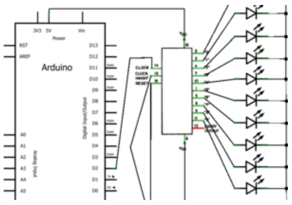
- Post to Delicious
- Post to Digg
- Post to Facebook

You may also like:


A Simple 3D Room made with DirectX and C++

3D Starfield in HTML5 Canvas


How to Draw A Cube Using PHP


Arduino LED Bar Graph Driven by a 4017 Counter

Manage subscriptions

First Android App: Touch Controlled Cube on Samsung Galaxy S2

See Also:

- Rotating 3D Wireframe Cube with Python
- 3D Starfield made using Python and Pygame
- Rotating 3D Cube using Python and Pygame
- A Cool Parallax Starfield Simulation using Python
- Simple Starfield with Python and Pygame

Computer Graphics, Pygame, Python 3d, pygame, python, simulation

← A Fast Introduction to Pygame Rotating 3D Wireframe Cube with Python →

Leave a comment ? 7 Comments.

Nik May 29, 2012 at 17:42 Reply

If you're running Python via IDLE, display window hangs on exit unless you do pygame.QUIT per…
def run(self):
while 1:
for event in pygame.event.get():
if event.type == pygame.QUIT:
pygame.QUIT
sys.exit()

lefam May 30, 2012 at 12:10 Reply

Hello Nik!
You are right. I will update the code to include pygame.quit()
Attention that in your snippet you used pygame.QUIT when it should be pygame.quit()

Thank you for pointing out this fact.

Nik May 29, 2012 at 17:44 Reply

Drat, it ate the white-space…

Nik May 30, 2012 at 01:53 Reply

ps: could you mention how you're drawing boxes without any rectangle command ??

lefam May 30, 2012 at 12:15 Reply

The statement below draws a 2×2 white rectangle:

self.screen.fill((255,255,255),(x,y,2,2))

(255, 255, 255) is an RGB tuple defining WHITE color.

(x, y, 2, 2) defines the rect where (x,y) are the coordinates of the upper left corner and (2,2) are width and height of the rect respectively.

Nik May 30, 2012 at 16:59 Reply

Thank you !!

davidh October 17, 2013 at 02:24 Reply

under def rotateX if you change it to 90 it will display all six sides, where as now it only displays 5

rad = angle * math.pi / 90

Leave a Comment

NAME

EMAIL

Website URL

SUBMIT

Notify me of followup comments via e-mail. You can also subscribe without commenting.

Δ Top