Using matrices

Rotation in 3D | 3D Graphics with Pygame | Matrix transformations

Dec. 19, 2011

Code on Github

Introduction

Our wireframe object is currently defined by a list of Node objects and a list of Edge objects, which hopefully makes things easy to understand, but it isn't very efficient. That's not a problem if we're just making cubes, but will be if we want to create more complex objects. In this tutorial, we will:

- Convert our list of nodes to a numpy array
- Simplify how edges are stored
- Create a cube using the new system

By the end of this and the following tutorial our program should function exactly the same as before, but will be more efficient.

NumPy

If you're not familiar with NumPy, then this tutorial might take a bit of work to understand, but I think it's worth the effort. You can do everything without using matrices, but it does actually simplify things in the long run and your program will be a lot quicker. I'll do my best to explain NumPy, but you might also want to look at the official tutorial.

The first thing is to download NumPy if you haven't already done so.Then import it in our wireframe.py module (the **as np** part is a common shortcut which saves a bit of typing later):

```
1.  import numpy as np
```

Since NumPy includes a lot of mathematical functions, we can use it to replace the math module, thus replace **math.sin()** with **np.sin()**.

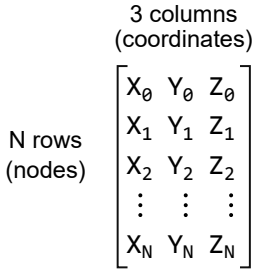
Our program currently defines nodes using the **Node** object, which is fine when you only have eight, but if you want thousands then it will quickly become very time and memory inefficient. A node is really just three numbers, so we could convert the list of nodes to a list of lists, each containing three numbers. However, if we use a NumPy array, we get a lot of built-in mathematical functions which will prove useful later.

So, we can delete our **Node** object and change the **Wireframe** class nodes attribute to:

```
16.  self.nodes = np.zeros((0, 4))
```

This creates a NumPy array with 0 row and 4 columns. This would be filled with zeros, but since there are no rows, there are no values. There are no rows because, to start with there are no nodes. There are four rather than three columns because it makes some transformations easier as I'll explain when we come to them. Note that we are using the NumPy array class and not the matrix class because it's easier to work with and does everything that matrices do. From a mathematical point of view they can still be considered matrices.

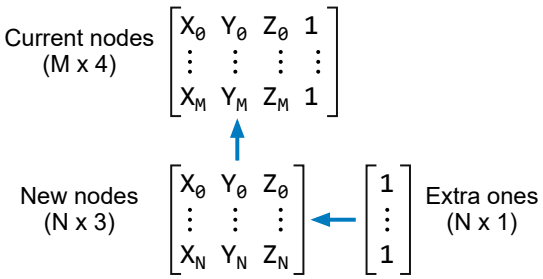
Next we need to change the **Wireframe** class **addNodes()** function because it currently takes a list of 3-tuples and converts each into a **Node** object. We want to change it to take a N x 3 NumPy array, in which each of the N rows is a vector of 3 coordinates (x, y and z).



For example, we would define the nodes of a unit square like this:

```
1.  square = Wireframe()
2.  nodes = np.array([[0, 0, 0],
3.                    [1, 0, 0],
4.                    [1, 1, 0],
```

In order to add this N x 3 array of nodes to the array of current nodes, we first need to add an N x 1 column of ones to get an N x 4 array. Then we add the new nodes as additional rows to the current node array.



We create a N x 1 array of ones by using np.ones(N, 1). We could work out the number of rows we need (N) by looking at the shape attribute of the new node array. For example, you can try:

```
6.  print nodes.shape
7.  >>> (4, 3)
```

Alternatively we can use the **len()** function, which returns the number of rows of an array, as though it were a list. Once we have a column of ones we horizontally stack onto the array of nodes, using **np.hstack()**. We then vertically stack that array onto the array of current nodes with **np.vstack()**. So we change our **Wireframe addNodes()** method to:

```
13.  def addNodes(self, node_array):
14.      ones_column = np.ones((len(node_array), 1))
15.      ones_added = np.hstack((node_array, ones_column))
16.      self.nodes = np.vstack((self.nodes, ones_added))
```

Simplifying edges

Just as the Node object as basically three numbers, the Edge object is basically two numbers. We could also replace all the edges with a simple NumPy array, but in this case, I think it's easier to use a list of lists. Once we've defined the edges we never need to change the values, so we don't need the matrix functions available for working with arrays.

We can therefore remove the Edge object simplify the **addEdges()** method to:

```
14.      self.edges += edgeList
```

Testing the new system

To check that our **addNodes()** and **addEdges()** methods are working as we expect, we should update the **outputNodes()** and **outputEdges()** methods.

```
16.  def outputNodes(self):
17.      print "\n --- Nodes --- "
18.      for i, (x, y, z, _) in enumerate(self.nodes):
19.          print "   %d: (%d, %d, %d)" % (i, x, y, z)
```

Here we loop through the nodes, getting their x, y and z coordinates. We can ignore the final value as this will always be 1.

We should update **outputEdges()** too.

```
21.  def outputEdges(self):
22.      print "\n --- Edges --- "
23.      for i, (node1, node2) in enumerate(self.edges):
24.          print "   %d: %d -> %d" % (i, node1, node2)
```

We can now create a cube object in a similar way as before:

```
36.  if __name__ == "__main__":
37.      cube = Wireframe()
38.      cube_nodes = [(x, y, z) for x in (0, 1) for y in (0, 1) for z in (0, 1)]
39.      cube.addNodes(np.array(cube_nodes))
40.      cube.addEdges([(n, n + 4) for n in range(0, 4)])
41.      cube.addEdges([(n, n + 1) for n in range(0, 8, 2)])
42.      cube.addEdges([(n, n + 2) for n in (0, 1, 4, 5)])
43.      cube.outputNodes()
44.      cube.outputEdges()
```

Fixing the display

node.x to **node[0]** and **node.y** to **node[1]**:

```
67.  if self.displayNodes:
68.      for node in wireframe.nodes:
69.          pygame.draw.circle(self.screen, self.nodeColour, (int(node[0]),
     int(node[1])), self.nodeRadius, 0)
```

To fix how the edges are displayed we change the code to:

```
63.  if self.displayEdges:
64.      for n1, n2 in wireframe.edges:
65.          pygame.draw.aaline(self.screen, self.edgeColour, wireframe.nodes[n1][:2],
     wireframe.nodes[n2][:2], 1)
```

This loops through the edges, which are a list of lists containing two numbers, which we call n1, and n2. These refer to the start and end nodes of the edges, so we get those nodes, and then extract their x- and y-coordinates, which are the first two, hence the **[:2]** index.

To test the code, we have to import numpy as np like before:

```
3.  import numpy as np
```

Then change the **addNodes()** call to:

```
99.  cube.addNodes(np.array(cube_nodes))
```

The one difference between creating a cube with this new **Wireframe** object, is that we must first convert the list comprehension into a NumPy array. Now we should find that we have successfully created a cube object. In the next tutorial we'll fix the transformation functions.

Comments (14)

Andy on May 16, 2013, 11:46 p.m.

Truly excellent tutorials! Everything I've spent the last tew months trying to do in Python and Pygame, you've managed to implement in a clear, simple and extensible way. Encouragingly, in many cases I wasn't far off the solution, I just needed a little help.

But where's the next tutorial? Just as things were hotting up! :)

Andy on May 16, 2013, 11:46 p.m.

Hi Peter,

Truly excellent tutorials! Everything I've spent the last few months trying to do in Python and Pygame, you've managed to implement in a clear, simple and extensible way. Encouragingly, in many cases I wasn't far off the solution, I just needed a little help.

But where's the next tutorial? Just as things were hotting up! :)

Peter on May 21, 2013, 2:34 p.m.

Thanks Andy. I have been ridiculously slow in writing this tutorial, I think the next one has been half-written for over six months. I've published it now as it is, but I will try to finish it soon. And then actually get to the interest part of shading things.

Peter on May 21, 2013, 2:34 p.m.

Thanks Andy. I have been ridiculously slow in writing this tutorial, I think the next one has been half-written for over six months. I've published it now as it is, but I will try to finish it soon. And then actually get to the interest part of shading things.

Anonymous on May 23, 2013, 2:34 a.m.

Great tutorials! Keep them coming! I'm doing a school project and am trying to create something similar to what you made in "Pygame physics simuation in 3D." Do you think you could post the code for that somewhere?

Great tutorials! Keep them coming! I'm doing a school project and am trying to create something similar to what you made in "Pygame physics simuation in 3D." Do you think you could post the code for that somewhere?

Peter on May 24, 2013, 2:45 p.m.

If you email me via the Contact me button on the left, I can email you the program. I was planning on writing about it in a tutorial, but it will probably take me a long time to get around to that.

Andy on June 10, 2013, 7:15 p.m.

Thank you Peter!! Sorry I didn't notice your reply until now. Can't wait to try out the things you mention in your unfinished article, I've been struggling with making the switch to numpy for ages.

Andy on June 10, 2013, 7:15 p.m.

Thank you Peter!! Sorry I didn't notice your reply until now. Can't wait to try out the things you mention in your unfinished article, I've been struggling with making the switch to numpy for ages.

---

justu on April 14, 2014, 7:26 a.m.

thanks for the share

Tom on Oct. 18, 2014, 6:58 p.m.

Hi Peter

Im just wondering where the next tutorial is? Im currently building a game in python and am interested in using this method for display. However I need to be able to display faces. You said in previous comments that you had posted the half finnised version of the next tutorial. Could you please send me a link to it.

Hi Peter,

Im just wondering where the next tutorial is? Im currently building a game in python and am interested in using this method for display. However I need to be able to display faces. You said in previous comments that you had posted the half finnised version of the next tutorial. Could you please send me a link to it.

Aaron Carlton on July 28, 2015, 11:52 p.m.

You are awesome sir! Keep up the great work!

Anonymous on Feb. 10, 2017, 4 a.m.

Tom, I believe you could implement faces simply by drawing polygons whose points are nodes in the wireframe's list of nodes (if you are using Peter's wireframe class or a similar class.)

Leave a comment

Name:

Comment: