# CS251 Proj3

## CS251 Project 3 - 3D Viewing

The purpose of this project was to build a class to handle the viewing aspect of 3D Data Visualization. The goal was to build a class for building and managing all the viewing parameters and view transformation matrix. To start, the first part of the project was the lab which had us create a View class that holds the current viewing parameter and can build a view transformation matrix[VTM] based on the parameters. Then, after we have a working capability to generate a VTM base on parameters, the project 3 tasks ask us to use the View matrix to create a set of axes that respond to user input to translate, scale, and rotate. Overall, the main outcome of this project was a 3D set of Data Axes that could translate, zoom in/out, and rotate with interaction from the mouse buttons and the user.

**Task 1.** This task asks us to start by creating a set of axes in our visualization program. To do this, I created a numpy matrix in my Display.py *_init_* method that stores 6 endpoints(2 for each axis) and the homogenous coordinate. I called this field axesEndpoints. Task1 also asked us to create another field that stores a list of actual graphic line objects, so I simply called this field 'axes'. Next, Task1 asked us to create two new functions. The purpose of first function, called buildAxes(), is to build the VTM, multiply the axesEndpoints by the VTM, and then create three new line objects, one for each axes. To do this, I first built a new VTM, transposed the points in my axesEndpoints matrix, and then multiplied VTM by axesEndpoints and transposed the result. Finally, I append three 'canvas.create_line' objects to my axes[] list and constructed the axes. Finally, I created the second function, called updateAxes(). This function looks very similar to my buildAxes() method, but instead of creating lines, it simply executes 'canvas.coords' that updates the coordinates of each line object. This is what one of the three canvas.coords lines looked like: "**self.canvas.coords(self.axes[1], pts[2,0], pts[2,1], pts[3,0], pts[3,1])"**

**Task 2.** This part of the project implements the user interaction to control the axes. There are 4 main steps to this task. Step 1 of Task2 implemented code for "Button1 Motion" and controlled translating the axes around the canvas. This first part of Task2 was the *process for panning*. Next, Step 2 of Task2 implemented code for "Button3 Motion" and controlled scaling the axes by zooming in and out. This second part of Task2 was the *process for scaling*. Here was my code for these two pieces:

```
#----------------------------------#Project3, Task2
    #called if the first button of a real mouse is pressed and the mouse is moving
    #-----Process for Panning (notes from Class w/Stephanie)
    #task2.a
    def handleMouseButton1Motion(self,event):

        #calculate the difference
        dx = float(event.x - self.baseClick[0])
        dy = float(event.y - self.baseClick[1])

        # dx = dx / self.canvas.winfo_width()
        dx = dx / self.view.screen[0,0]
        dy = dy / self.view.screen[0,1]
        dx = dx * self.view.extent[0,0]
        dy = dy * self.view.extent[0,1]

        delta0 = dx
        delta1 = dy

        self.view.vrp = self.view.vrp + delta0*self.view.u + delta1*self.view.vup

        self.updateAxes()

        self.baseClick = ( event.x, event.y)
```

```
#----------------------------------#Project3, Task2
    #called if the third button of a real mouse is pressed
    #-----Process for Scaling (notes from Class w/Stephanie)
    #task2.b
    def handleMouseButton3(self, event):
        self.baseClick = (event.x, event.y)
        self.baseExtent = self.view.extent.copy() #help from Theo S
        #print 'hwllo'
        #help from Stephanie during office hours

    def handleMouseButton3Motion(self, event):
        #print "got here"
        dy = event.y - self.baseClick[1]
        k = 1.0 / self.canvas.winfo_height()
        print k
        f = 1.0 + k * dy
        f = max( min(f, 3.0), 0.1)
        self.view.extent = self.baseExtent * f
        self.updateAxes()
```

Next, the second half of Task 2 involves the rotation of the axes. Step 3 of Task2 asks me to create a rotateVRC function in my View.py class. This class makes a pipeline of translation or alignment matrices that come together to handle the result of rotating. Then, it normalizes the new VRC values from the rotation. Finally, Step 4 of Task2 asks me to bing the rotation action the "Button 2 Motion." To do this, I create a new field called BaseClick2 that allows me to calculate the difference between where I first click and where I drag the mouse to. Then, I can use that value to calculate a degree and rotate by that. Here is my rotateVRC method and my handleMouseButton2Motion(...**i know** this is a lot of code... but i like being super organized lol):

```
#=========================================================
#-----------------------------#Project3, Task2
    def rotateVRC(self, angleVUP, angleU):
        #task2.c.1
        val = self.vrp + self.vpn * self.extent[0,2] * 0.5
        t1 = np.matrix( [[1, 0, 0, -val[0,0]],
                         [0, 1, 0, -val[0,1]],
                         [0, 0, 1, -val[0,2]],
                         [0, 0, 0, 1] ] )
        #task2.c.2
        Rxyz = np.matrix([[self.u[0, 0], self.u[0, 1], self.u[0, 2], 0.0],
                          [self.vup[0, 0], self.vup[0, 1], self.vup[0, 2], 0.0],
                          [self.vpn[0, 0], self.vpn[0, 1], self.vpn[0, 2], 0.0 ],
                          [0, 0, 0, 1] ] )
        #task2.c.3
        r1 = np.matrix( [[math.cos(angleVUP), 0, math.sin(angleVUP), 0],
                         [0, 1, 0, 0],
                         [-math.sin(angleVUP), 0,math.cos(angleVUP), 0],
                         [0, 0, 0, 1] ] )
        #task2.c.4
        r2 =  np.matrix( [[1, 0, 0, 0],
                         [0, math.cos(angleU), -math.sin(angleU), 0],
                         [0, math.sin(angleU), math.cos(angleU), 0],
                         [0, 0, 0, 1] ] )
        #task2.c.5
        t2 = np.linalg.inv(t1) #help from Theo S.
        #task2.c.6
        tvrc = np.matrix([[ self.vrp[0, 0], self.vrp[0, 1], self.vrp[0,2], 1.0],
                          [ self.u[0, 0], self.u[0, 1], self.u[0, 2], 0.0 ],
                          [ self.vup[0, 0], self.vup[0, 1], self.vup[0, 2], 0.0 ],
                          [ self.vpn[0, 0], self.vpn[0, 1], self.vpn[0, 2], 0.0 ]] )
        #task2.c.7
        tvrc = (t2*Rxyz.T*r2*r1*Rxyz*t1*tvrc.T).T
        #print tvrc
        #task2.c.8
        self.vrp = tvrc[0, :3]
        self.u = self.normalize(tvrc[1, :3])
        self.vup = self.normalize(tvrc[2, :3])
        self.vpn = self.normalize(tvrc[3, :3])
        #print self.u, self.vup, self.vpn
```

```
#-----------------------------------#Project3, Task2
    #task2.d
    def handleMouseButton2(self, event):
        self.baseClick2 = (event.x, event.y)
        self.cloneView = self.view.clone()
        print 'handle mouse button 2: %d %d' % (event.x, event.y)

    #called if the second button of mouse has been pressed and the mouse is moving
    def handleMouseButton2Motion(self, event):
        diff = ( event.x - self.baseClick2[0], event.y - self.baseClick2[1])

        delta0 = (-diff[0])/(1.0*self.canvas.winfo_height())*math.pi
        delta1 = (diff[1])/(1.0*self.canvas.winfo_width())*math.pi

        degx = delta0*(180/math.pi)
        degy = delta1*(180/math.pi)

        self.view = self.cloneView.clone()
        self.view.rotateVRC(delta0, delta1)
        self.updateAxes()
```

==========

## Data API and Functions ~~ View.py Class

**View:** class that holds the current viewing parameter and can build a view transformation matrix[VTM] based on the parameters. We can use the View matrix to create a set of axes that respond to user input to translate, scale, and rotate. Overall, View helps Display create 3D set of Data Axes that could translate, zoom in/out, and rotate with interaction from the mouse buttons and the user.

*Here are my View class methods:*

**reset():** this method assigns default *_init_* values to a number of fields ===> vpr, vpn, vup, u extent, screen, offset

**normalize():** takes in a vector parameter and then executes math to normalize the vector

**build():** this is the main chunk of the view class that we created in lab. This build method uses the current viewing parameters to return a view matrix.

**clone():** this method returns a copy of View and all of its fields.

**rotateVRC():** this method handles rotating the axes. It takes two angle arguments ===> how much to rotate about VUP axis and how much to rotate about U axis. This class makes a pipeline of translation or alignment matrices that come together to handle the result of rotating. Then, it normalizes the new VRC values from the rotation.

==========

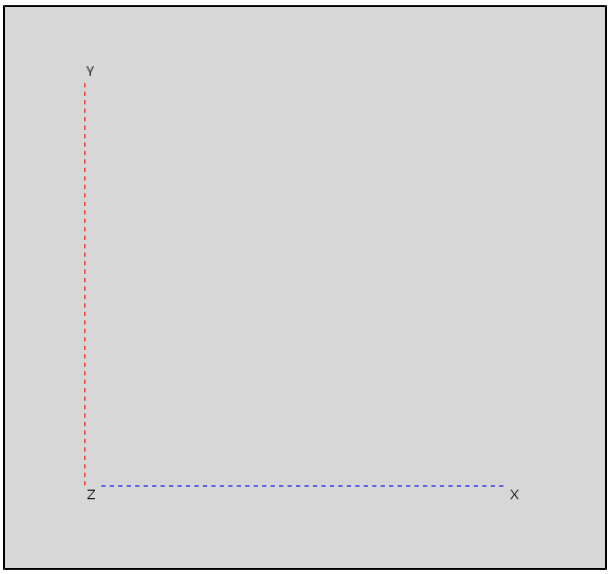
**Extension1.** My first extension was simply to add color to my axis. To do this, I added a " fill='color' " parameter for create_lines in my buildAxes() code. I also made these lines dashed with the help of Theo Slatloff.

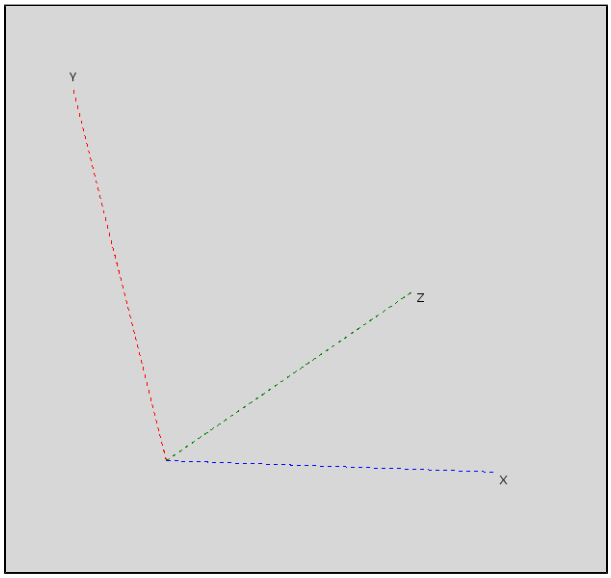
**Extension2.** Next, I added axis labels to my 3D graph. To do this, I added these 6 lines of code to my buildAxes() method:

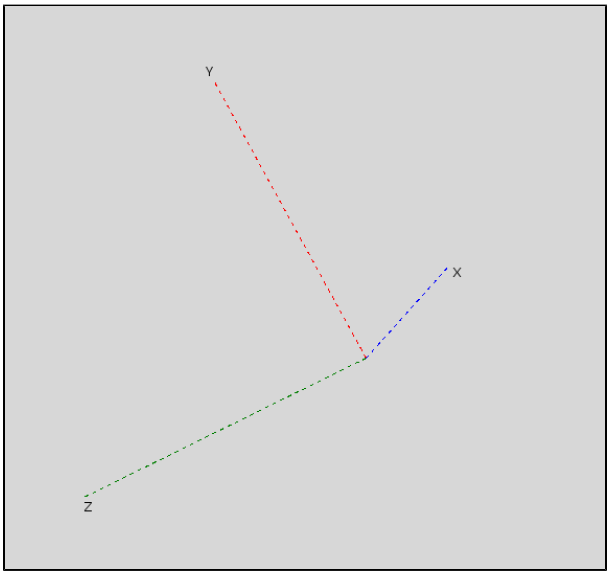
```
    #~~~~~~~~~~~~~Project3, Extension 2~~~~~~~~~~~~~~~~~~~
        #EXTENSION2(P3), Axes Labels
        self.xLabel = tk.Label(self.canvas, text = "X")
        self.xLabel.place(x=pts[1,0], y=pts[1,1])

        self.yLabel = tk.Label(self.canvas, text = "Y")
        self.yLabel.place(x=pts[3,0], y=pts[3,1])

        self.zLabel = tk.Label(self.canvas, text = "Z")
        self.zLabel.place(x=pts[5,0], y=pts[5,1])
```

**Extension3.** Finally, I created a menu command that allows the user to reset the canvas to the original view. To do this, I simply created a method in my Display class that calls the reset() method from View.py. Then, I updated my menucmd and menutext code in buildMenus(). It successfully works and resets the 3D axes to their original placement!

***~~~CHECK OUT SOME PICTURES OF THE FINAL RESULTS OF MY AXES:~~~~***




**What I learned.** I learned how to construct a View class that can build a 3D set of axes by using a VTM and an endpoints matrix to create lines on a canvas. I also learned how to create some awesome user interaction with my axes such as panning, scaling, and rotation. Through this project, I became more comfortable with numpy! Also, I became more comfortable with working with user interaction and GUI knowledge in my Display.py class. Finally, I am understanding how to visualize the work more clearly to better understand the goal with data analysis and visualization.

**Who helped me.** I received of help from the amazing Bruce Maxwell and classmates Theo and Walker. Also, I collaborated with Julia Saul, Steven Parrott, and Brendan Doyle.