Prakhar Bhardwaj
MS Mechanical – Research
CMU

Language used – Python

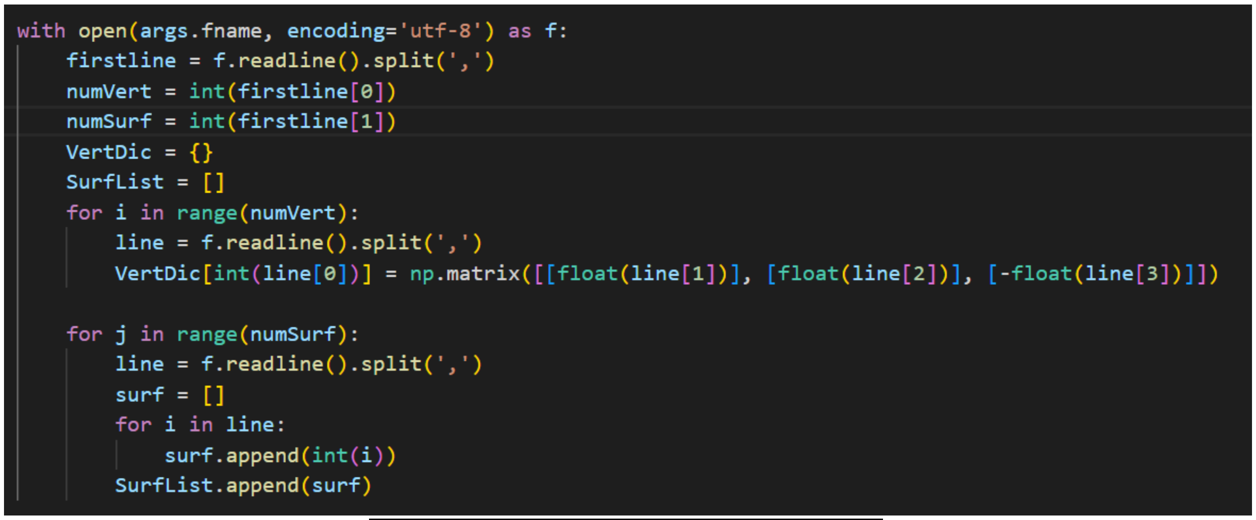
Libraries used –

1	Pygame – for creating a window and handling mouse inputs
2	Numpy – for performing mathematical operations on matrices
3	Math – for mathematical operations such as cosine, sine, and radians
4	openGL – for providing different utility functions, such as setting the camera view

Part 1:

Converting 3D coordinates to 2D –

I have done the conversion from 3D coordinates to 2D using matrix operations. The 3D coordinates are initially stored in a dictionary called VertDic, where the keys are integers representing each vertex, and the values are numpy matrices representing their x, y, and z coordinates. SurfList is a list that will store the IDs of the vertices that make up the surface.

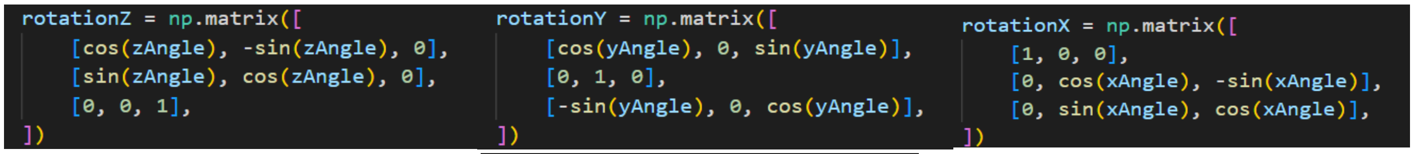


```
with open(args.fname, encoding='utf-8') as f:
    firstline = f.readline().split(',')
    numVert = int(firstline[0])
    numSurf = int(firstline[1])
    VertDic = {}
    SurfList = []
    for i in range(numVert):
        line = f.readline().split(',')
        VertDic[int(line[0])] = np.matrix([[float(line[1])], [float(line[2])], [-float(line[3])]])

    for j in range(numSurf):
        line = f.readline().split(',')
        surf = []
        for i in line:
            surf.append(int(i))
        SurfList.append(surf)
```

Fig. Implementation of VertDic and SurfList

These coordinates are then rotated around the z, y, and x axes according to the user's input using 3 separate rotation matrices.



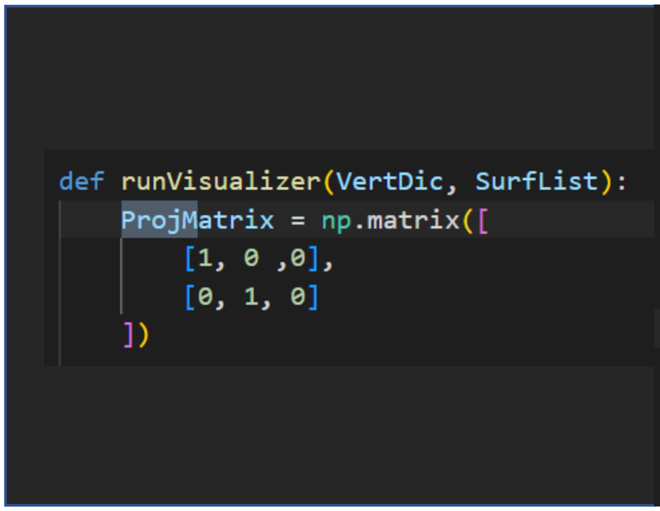
```
rotationZ = np.matrix([
    [cos(zAngle), -sin(zAngle), 0],
    [sin(zAngle), cos(zAngle), 0],
    [0, 0, 1],
])
```

```
rotationY = np.matrix([
    [cos(yAngle), 0, sin(yAngle)],
    [0, 1, 0],
    [-sin(yAngle), 0, cos(yAngle)],
])
```

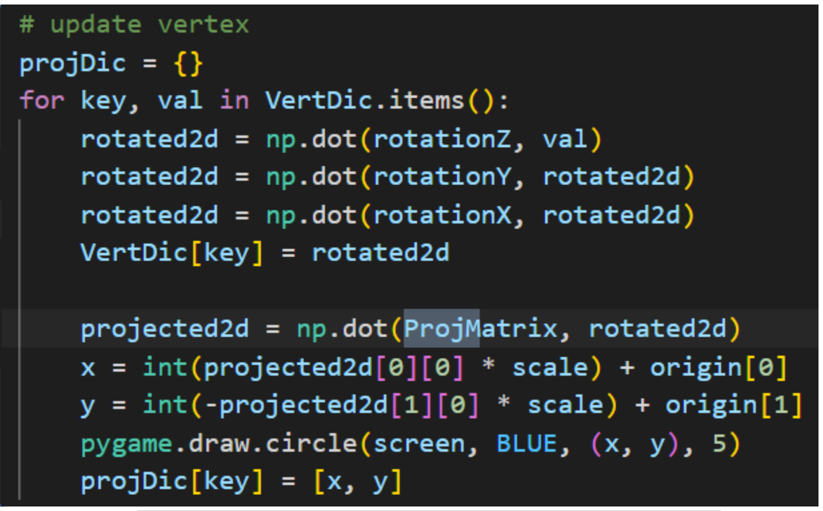
```
rotationX = np.matrix([
    [1, 0, 0],
    [0, cos(xAngle), -sin(xAngle)],
    [0, sin(xAngle), cos(xAngle)],
])
```

Fig. Definition of rotation matrices

The rotated coordinates are then projected onto the 2D screen using a 2x3 projection matrix called ProjMatrix, which is multiplied with the rotated 3D coordinates to obtain their 2D screen coordinates. These screen coordinates are then drawn using Pygame's drawing functions.



```
def runVisualizer(VertDic, SurfList):
    ProjMatrix = np.matrix([
        [1, 0 ,0],
        [0, 1, 0]
    ])
```



```
# update vertex
projDic = {}
for key, val in VertDic.items():
    rotated2d = np.dot(rotationZ, val)
    rotated2d = np.dot(rotationY, rotated2d)
    rotated2d = np.dot(rotationX, rotated2d)
    VertDic[key] = rotated2d

    projected2d = np.dot(ProjMatrix, rotated2d)
    x = int(projected2d[0][0] * scale) + origin[0]
    y = int(-projected2d[1][0] * scale) + origin[1]
    pygame.draw.circle(screen, BLUE, (x, y), 5)
    projDic[key] = [x, y]
```

Fig. Defining the projection matrix

Fig. Implementation of 2D projected coordinates

Reason for choosing the projection matrix –

The matrix is used to convert the 3D coordinates into 2D using matrix multiplication. By multiplying the 3D point (x,y,z) by this projection matrix, the resulting 2D point will have the same x and y coordinates as the original 3D point, but with a z coordinate of zero. This effectively projects the 3D point onto the x-y plane.

Part 1 – Final Output

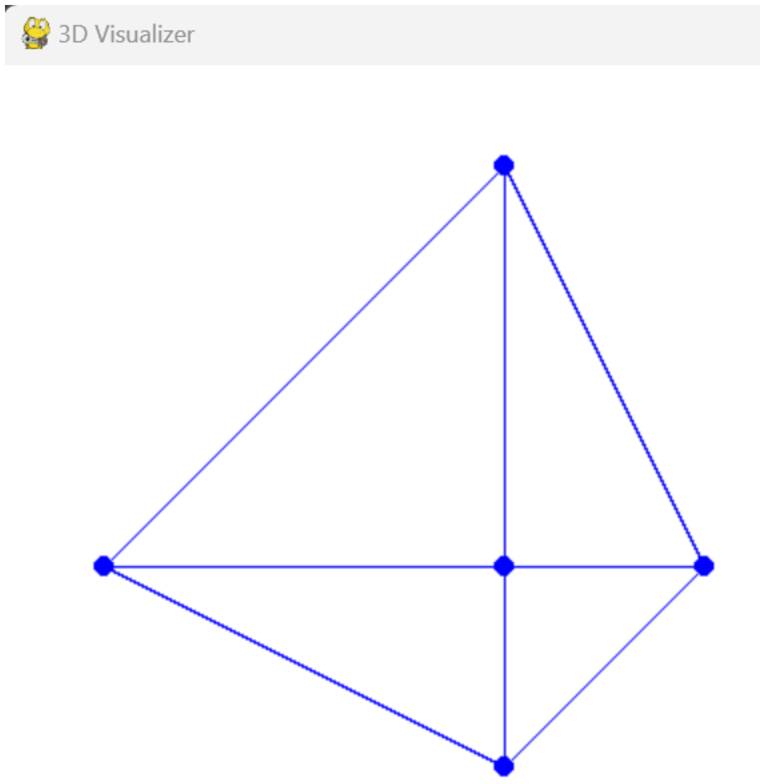
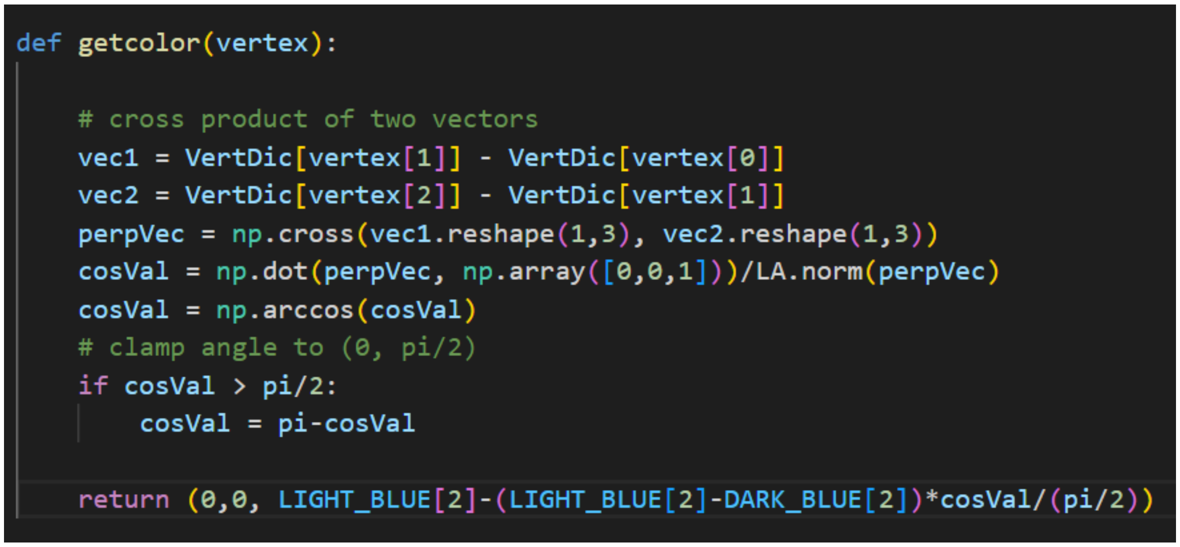


Fig. Output for part 1 of the assignment

Part 2:

Smooth color variation based on the angle with the Z-axis –

I have made a getcolor function to calculate the shading color of each polygon surface based on the angle between the surface normal and the Z-axis. This function takes a list of vertex indices that define the polygon's surface as input. It first calculates the cross product of two vectors formed by taking the difference between adjacent vertices. Then it calculates the angle between the surface normal and the Z-axis using the dot product and the norms of the two vectors. The angle is then normalized to the range [0, pi/2] and mapped to color between LIGHT_BLUE and DARK_BLUE.



```python
def getcolor(vertex):

    # cross product of two vectors
    vec1 = VertDic[vertex[1]] - VertDic[vertex[0]]
    vec2 = VertDic[vertex[2]] - VertDic[vertex[1]]
    perpVec = np.cross(vec1.reshape(1,3), vec2.reshape(1,3))
    cosVal = np.dot(perpVec, np.array([0,0,1]))/LA.norm(perpVec)
    cosVal = np.arccos(cosVal)
    # clamp angle to (0, pi/2)
    if cosVal > pi/2:
        cosVal = pi-cosVal

    return (0,0, LIGHT_BLUE[2]-(LIGHT_BLUE[2]-DARK_BLUE[2])*cosVal/(pi/2))
```

Fig. Implementation of color variation on the faces

Part 2 – Final Output

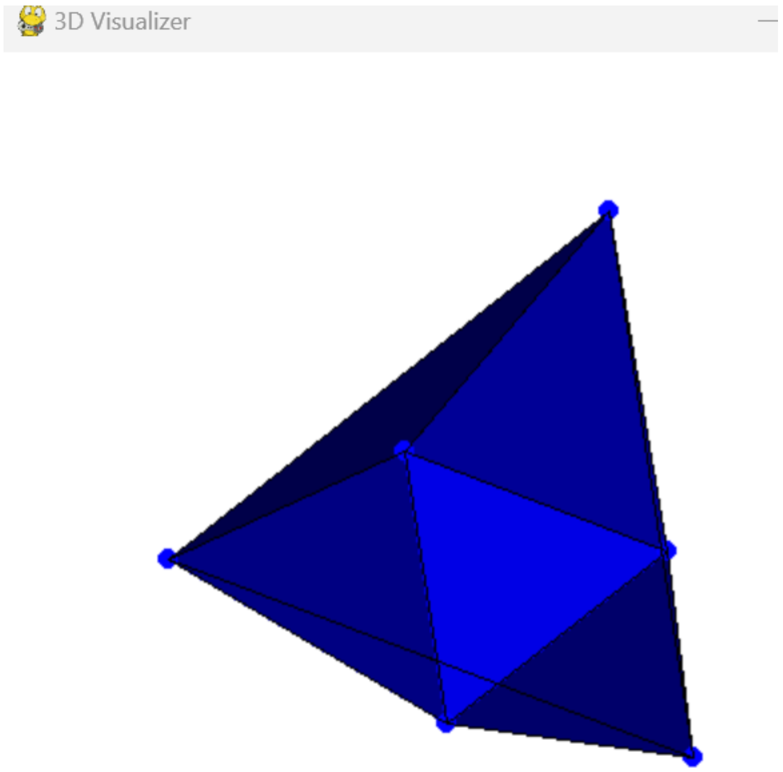


Fig. Output for part 2 of the assignment