CSE 410: Assignment 2

Scene Description

The scene description is provided in the `scene.txt` file. Lines 1-3 of scene.txt state the parameters of the `gluLookAt` function, i.e., eye position (`eyeX`, `eyeY`, and `eyeZ` in Line 1), look position (`lookX`, `lookY`, and `lookZ` in Line 2), and up direction (`upX`, `upY`, and `upZ` in Line 3). Line 4 provides the `gluPerspective` parameters, i.e., field of view along Y axis (`fovY`), aspect ratio indicating the ratio between the field of view along X and the field of view along Y axis (`aspectRatio`), near distance (`near`), and far distance (`far`). The rest of `scene.txt` contains the *display code* to generate/draw the model. The display code contains 7 commands as follows:

1. `triangle` command – this command is followed by three lines specifying the coordinates of the three points of the triangle to be drawn. The points being `p1`, `p2`, and `p3`, 9 double values, i.e., `p1.x`, `p1.y`, `p1.z`, `p2.x`, `p2.y`, `p2.z`, `p3.x`, `p3.y`, and `p3.z` indicate the coordinates. This is equivalent to the following in OpenGL code.
```
glBegin(GL_TRIANGLE);{
      glVertex3f(p1.x, p1.y, p1.z);
      glVertex3f(p2.x, p2.y, p2.z);
      glVertex3f(p3.x, p3.y, p3.z);
}glEnd();
```
2. `translate` command – this command is followed by 3 double values (`tx`, `ty`, and `tz`) in the next line indicating translation amounts along X, Y, and Z axes. This is equivalent to `glTranslatef(tx, ty, tz)` in OpenGL.
3. `scale` command – this command is followed by 3 double values (`sx`, `sy`, and `sz`) in the next line indicating scaling factors along X, Y, and Z axes. This is equivalent to `glScalef(sx, sy, sz)` in OpenGL.
4. `rotate` command – this command is followed by 4 double values in the next line indicating the rotation angle in degree (`angle`) and the components of the vector defining the axis of rotation (`ax`, `ay`, and `az`). This is equivalent to `glRotatef(angle, ax, ay, az)` in OpenGL.
5. `push` command – This is equivalent to `glPushMatrix` in OpenGL.
6. `pop` command – This is equivalent to `glPopMatrix` in OpenGL.
7. `end` command – This indicates the end of the display code.

In this assignment, you will generate the output of the first three stages of the raster based graphics pipeline according to the scene description provided in `scene.txt` file. The output of the stages should be put in `stage1.txt`, `stage2.txt`, and `stage3.txt` file.

Stage 1: Modeling Transformation

In the Modeling transformation phase, the display code in `scene.txt` is parsed, the transformed positions of the points that follow each `triangle` command are determined, and the transformed coordinates of the points are written in `stage1.txt` file. We maintain a stack *S* of transformation matrices which is manipulated according to the commands given in the display code. The pseudo-code for the modeling transformation phase is as follows:

```
initialize empty stack S
S.push(identity matrix)
while true
        input command
        if command = "triangle"
                input three points
                for each three point P
                        P' <- transformPoint(S.top,P)
                        output P'
        else if command = "translate"
                input translation amounts
                generate the corresponding translation matrix T
                S.push(product(S.top,T))
        else if command = "scale"
                input scaling factors
                generate the corresponding scaling matrix T
                S.push(product(S.top,T))
        else if command = "rotate"
                input rotation angle and axis
                generate the corresponding rotation matrix T
                S.push(product(S.top,T))
        else if command = "push"
                //do it yourself
        else if command = "pop"
                //do it yourself
        else if command = "end"
                break
```

Transformation matrix for Translation

```
translate
tx ty tz
```

The transformation matrix for the above translation is as follows:

$$\begin{vmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Transformation matrix for Scaling

```
scale
sx sy sz
```

The transformation matrix for the above scaling is as follows:

$$\begin{vmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Transformation matrix for Rotation

Remember that, the columns of the rotation matrix indicate where the unit vectors along the principal axes (namely, `i`, `j`, and `k`) are transformed. We will use the vector form of Rodrigues formula to determine where `i`, `j`, and `k` are transformed and use those to generate the rotation matrix. The vector form of Rodrigues formula is as follows:

$$R(\vec{x},\vec{a},\theta) = \cos\theta\,\vec{x} + (1-\cos\theta)(\vec{a}.\vec{x})\,\vec{a} + \sin\theta(\vec{a}X\vec{x})$$

In the above formula, $\vec{a}$ is a unit vector defining the axis of rotation, θ is the angle of rotation, and $\vec{x}$ is the vector to be rotated.

Now we outline the process of generating transformation matrix for the following rotation:

```
rotate
angle ax ay az
```

We denote the vector `(ax, ay, az)` by `a`. The steps to generate the rotation matrix are as follows:

```
a.normalize()
c1=R(i,a,angle)
c2=R(j,a,angle)
c3=R(k,a,angle)
```

The corresponding rotation matrix is given below:

$$\begin{vmatrix} c1.x & c2.x & c3.x & 0 \\ c1.y & c2.y & c3.y & 0 \\ c1.z & c2.z & c3.z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Managing Push and Pop

The following table demonstrates how `push` and `pop` works. The state of the transformation matrix stack after execution of each line of the code in the left is shown in the right. Design a data structure that manages `push` and `pop` operations on the transformation matrix stack accordingly.

	Stack State after Lines											
Code	0	1	2	3	4	5	6	7	8	9	10	11
1.Push												
2.Translate$_1$												
3.Push												
4.Rotate$_1$												
5.Pop												
6.Scale$_1$												
7.Push												
8.Rotate$_2$									$T_1S_1R_2$		$T_1S_1S_2$	
9.Pop					T_1R_1		T_1S_1	T_1S_1	T_1S_1	T_1S_1	T_1S_1	
10.Scale$_2$			T_1	T_1	T_1	T_1	T_1	T_1	T_1	T_1	T_1	
11.Pop	I	I	I	I	I	I	I	I	I	I	I	I

Stage 2: View Transformation

In the view transformation phase, the `gluLookAt` parameters in `scene.txt` is used to generate the view transformation matrix *V*, and the points in `stage1.txt` are transformed by *V* and written in `stage2.txt`. The process of generating *V* is given below.

First determine mutually perpendicular unit vectors `l`, `r`, and `u` from the `gluLookAt` parameters.

```
l = look - eye
l.normalize()
r = l X up
r.normalize()
u = r X l
```

Apply the following translation *T* to move the eye/camera to origin.

$$\begin{vmatrix} 1 & 0 & 0 & -eyeX \\ 0 & 1 & 0 & -eyeY \\ 0 & 0 & 1 & -eyeZ \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Apply the following rotation *R* such that the `l` aligns with the -Z axis, `r` with X axis, and `u` with Y axis. Remember that, the rows of the rotation matrix contain the unit vectors that align with the unit vectors along the principal axes after transformation.

$$\begin{vmatrix} r.x & r.y & r.z & 0 \\ u.x & u.y & u.z & 0 \\ -l.x & -l.y & -l.z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Thus the view transformation matrix *V*=*RT*.

Stage 3: Projection Transformation

In the projection transformation phase, the `gluPerspective` parameters in scene.txt are used to generate the projection transformation matrix *P*, and the points in `stage2.txt` are transformed by *P* and written in `stage3.txt`. The process of generating *P* is as follows:

First compute the field of view along X (`fovX`) axis and determine `r` and `t`.

```
fovX = fovY * aspectRatio
t = near * tan(fovY/2)
r = near * tan(fovX/2)
```

The projection matrix *P* is given below:

$$
\begin{vmatrix}
\text{near}/r & 0 & 0 & 0 \\
0 & \text{near}/t & 0 & 0 \\
0 & 0 & -(\text{far}+\text{near})/(\text{far}-\text{near}) & -(2*\text{far}*\text{near})/(\text{far}-\text{near}) \\
0 & 0 & -1 & 0
\end{vmatrix}
$$

Refer to [1] for understanding the 3rd row of the above projection matrix. Image source: [2].

Do's and Dont's

1. Use homogeneous coordinates. The points should be represented by 4*1 matrices and transformations by 4*4 matrices.
2. While transforming a homogeneous point by multiplying it with a transformation matrix, don't forget to scale the resultant point such that the w coordinate of the point becomes 1.
3. Do not use the matrix form of Rodrigues formula directly to generate the rotation matrix. Use the procedure shown above that uses the vector form of Rodrigues formula.
4. Do not specify `gluLookAt` parameters in `scene.txt` such that the looking direction, i.e., `look-eye`, becomes parallel to the up direction.
5. Make sure that the model is situated entirely in front of the near plane.

Reference

[1] http://www.songho.ca/opengl/gl_projectionmatrix.html

[2] http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/