

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
```

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 \mathbf{l} , \mathbf{r} , and \mathbf{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.

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

Apply the following rotation R such that the \mathbf{l} aligns with the $-Z$ axis, \mathbf{r} with X axis, and \mathbf{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.

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

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 axis (fovX) 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. Use **the vector form** of Rodrigues formula to generate the rotation matrix in phase 1. Remember that, the columns of the rotation matrix indicate where the unit vectors along the principal axes are transformed. Use the vector form of Rodrigues formula to determine where i , j , and k are transformed and use them to generate the rotation matrix.
4. Notice that the rotation axis in the `rotate` command might not be normalized. Normalize it before using.
5. Do not specify `gluLookAt` parameters in `scene.txt` such that the looking direction, i.e., `look-eye`, becomes parallel to the up direction.
6. 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/>