CSE 333/533
Assignment 02:Transformations, Viewing and Projection

Abhishek Pratap Singh

2018211 - abhishek18211@iiitd.ac.in

1 TASK

Implement the following 4 tasks:
1. Construct a 4x4 matrix for rotating the rendered cube 30 degrees anti-clockwise about axis v(1,2,2). This task must be done by deriving the matrix using change of basis.

2. Implement the GLM LookAt() function from scratch.

3. Render the rotated cube with custom camera parameters. Compare the results with the GLM LookAt() function.

4. Perform viewing transformations to render different perspectives of the original unrotated cube.

2 IMPLEMENTATION

TASK 1: Transformation: Given is a template code to render a cube in perspective projection. The task 1 requires us to rotate the cube about an arbitrary axis.

To implement rotation of cube about an arbitrary axis, the strategy I used is as follows.

1. Let the given axis vector *(1, 2, 2)* be denoted by **a**. Now an orthonormal **uvw** basis must be formed with **w=a**. To create uvw basis, steps are described below:

a. Make **w** a unit vector.

$$\begin{aligned} w &= \frac{a}{||a||} \\ => w &= (\frac{1}{3}, \frac{2}{3}, \frac{2}{3}) \end{aligned} \tag{1}$$

b. Choose any vector **t** not collinear with **w** and build **u**.

$$\begin{aligned} u &= \frac{t \times w}{||t \times w||} \\ Let\ t &= (\frac{2}{3}, \frac{1}{3}, \frac{2}{3}) \\ => w &= (-0.485071, -0.485071, 0.727607) \end{aligned} \tag{2}$$

c. Complete the basis by computing **v** as:

$$\begin{aligned} v &= w \times v \\ => v &= (0.808452,\ -0.565917,\ 0.16169) \end{aligned} \tag{3}$$

2. Now rotate **uvw** basis to canonical basis **xyz**, then rotate about the z-axis by given angle

$$\phi = 30°.$$

Finally rotate the canonical basis back to **uvw basis**. This process is shown below:

$$R_a(\phi) = \begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}$$

Figure 1: Arbitrary 3D Rotation

5. The 4x4 matrix of the above process looks like:

$$\begin{bmatrix} -0.485071 & 0.808452 & 0.333333 & 0.0 \\ -0.485071 & -0.565917 & 0.666667 & 0.0 \\ 0.727607 & 0.16169 & 0.666667 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \begin{bmatrix} cos30 & -sin30 & 0.0 & 0.0 \\ sin30 & cos30 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \begin{bmatrix} -0.485071 & -0.485071 & 0.727607 & 0.0 \\ 0.808452 & -0.565917 & 0.16169 & 0.0 \\ 0.333333 & 0.666667 & 0.666667 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

4. The resulting cube renders out as:

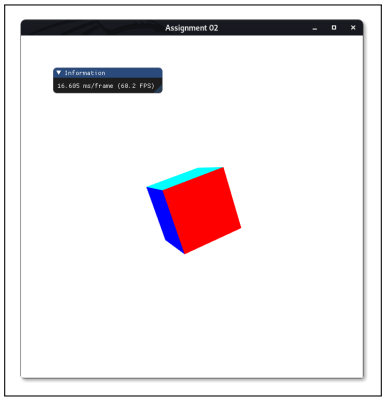

Figure 2: Output render of rotation.

5. This render is compared with the **glm::rotate** output and it is the same. Hence, the steps above are verified and correct.

Task 2: Viewing: An input of the following camera parameters : **eye position e(50, 100, 20)**, **gaze g(center of cube)** and **up position t(0,0,1)** is given and the task is to implement the GLM lookAt() function from scratch. The implementation is as follows:

1. The input parameters **(e,g,t)** are first used to set up a coordinate system with origin e, and a uvw basis. This is done by:

$$\begin{aligned} w &= -\frac{g}{||g||} \\ u &= \frac{t \times w}{||t \times w||} \\ v &= w \times v \end{aligned} \tag{4}$$

2. The cube coordinates are stored in terms of the world origin with basis **xyz**. The coordinates are then converted from **world(xyz) space** to the **camera(uvw) space**. This is done by:

$$\begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3: Camera Transformation

3. Initially, the output was blank. I then notice that to compute **u**, the **eye vector e** must be subtracted from the **gaze vector g**. This is because we need a vector that defines the displacement from the position where the camera is looking at to the position of the camera. The following output is obtained.

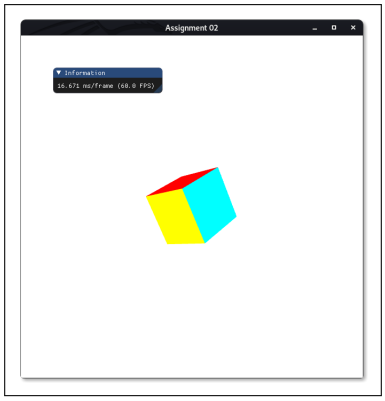

Figure 4: Render after custom lookAt() transformation

This output is exact same as the glm::lookAt() function.

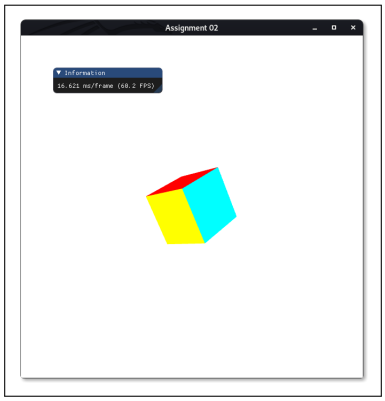

Figure 5: Render after glm::lookAt() transformation

Task 3: Projection: We are required to perform viewing transformations to render the original cube with the following perspectives: One-point, two-point, three-point(bird's eye), three-point(rat's eye). The output renders are below.

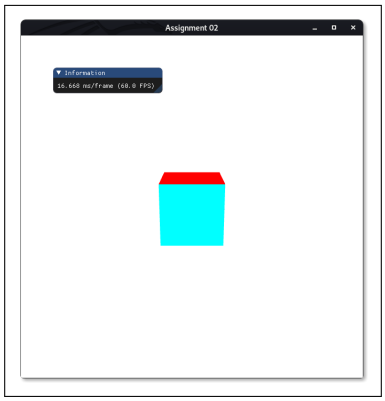

Figure 6: One Point Perspective

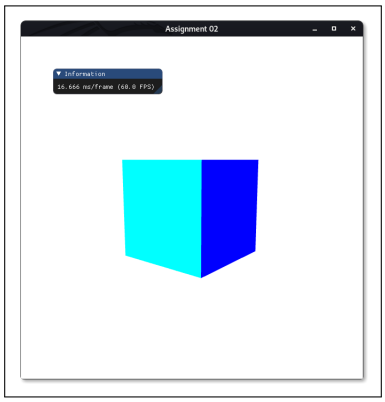

Figure 7: Two Point Perspective

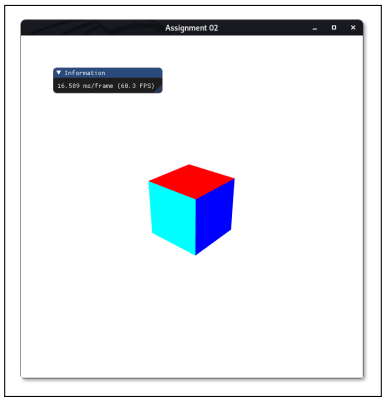

Figure 8: Bird's Eye - Three Point Perspective

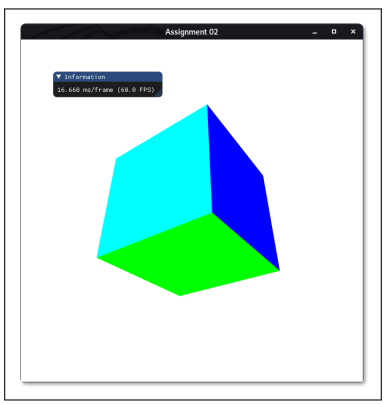

Figure 9: Rat's Eye - Three Point Perspective