CG Assignment 2

Anshul Mendiratta (2018219)

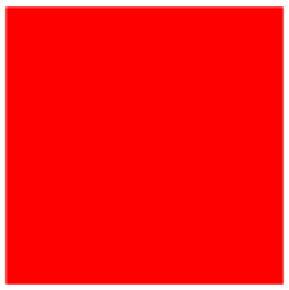
September 22, 2020

Abstract

Study the program and understand how viewing and projection matrices are applied in the OpenGL programmable pipeline.

1 Introduction

We have initially been given a cube, with each face having a different color. We want to apply and study transformation of the cube about an arbitrary axis and construct a method for applying viewing transformations.
The initial cube looks like this -



(1)

2 Transformation About Arbitrary Axis

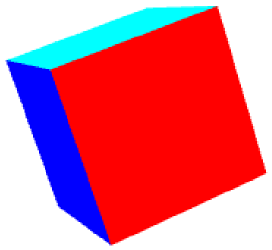
We need to rotate our cube by 30 degrees anti-clockwise about the axis v(1, 2, 2) by using change of basis.

We do the following -

1. We first normalize vector v and assign this newly obtained vector to w.
2. Then we consider some vector t not colinear to w.
3. Assign the cross product of t and w to u and normalize it.
4. Assign the cross product of w and u to v and normalize it.
5. Hence, we have constructed a new basis of u, v, w.
6. We now form a 4X4 matrix M_1 with rows u, v, w.
7. Next we form rotation matrix 4X4 M_2, along the z axis using angle as 30 degrees.
8. Finally we form transformation matrix $M_3 = M_1$.T * M_2 * M_1.

9. Any point vector multiplied with this matrix will be rotated by 30 degrees in anti clockwise direction about the axis v(1, 2, 2).

Applying this to our cube, we have -



(2)

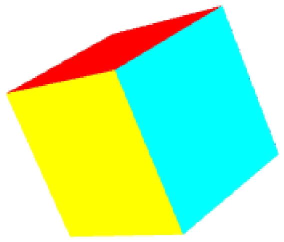
3 Viewing Transformation

We want to implement our own version of openGL's lookAt() function which renders an object with given parameters.

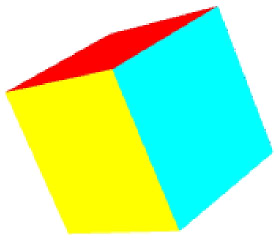
We implement a function called myLookAt() which takes in 3 vectors as parameters - position e, gaze g and up t - and does the following -

1. We first construct a direction vector between the centre of the cube and the position of the camera e, normalize it and assign it to vector w.
2. We then cross t and w and assign it to u after normalization.
3. We then cross w and u and assign it to v after normalization.
4. Now we form a 4X4 matrix M_1, with rows as u, v and w.
5. We also form a 4X4 identity matrix M_2 with last column as (-e[0], -e[1], -e[2], 1)
6. We finally form matrix $M_3 = M_1 * M_2$
7. Matrix M_3 is our required matrix

We can see that both renders are same which is confirmed by the coordinates obtained. Our cube after applying the viewing transformation looks like -



(a) myLookAt



(b) lookAt

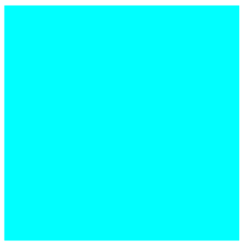
Figure 1: Render of cube using our implementation and OpenGL's implementation

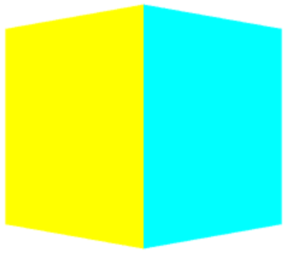
4 Projection Transformation

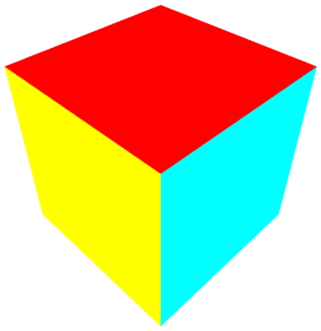
We reset the cube transformation to the default values. We keep gaze at (0,0,0) and up as (0,0,1) and only change position

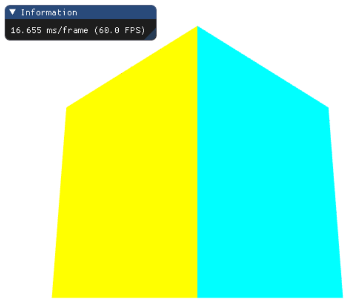
We use the myLookAt function that we implemented earlier. We set the position vector, e for -

1. One Point, e = 0, 40, 0
2. Two Point, e = 50, 50, 0
3. Bird's Eye, e = 30, 30, -10
4. Rat's eye, e = 25, 25, 30


(a) One Point


(b) Two Point


(c) Bird's Eye



```
16.655 ms/frame (60.0 FPS)
```

(d) Rat's Eye

Figure 2: Render of cube from different view points