FPV NAVIGATION

Comp 410: Computer Graphics

Term Project

Eren Limon - 0054129

TABLE OF CONTENTS

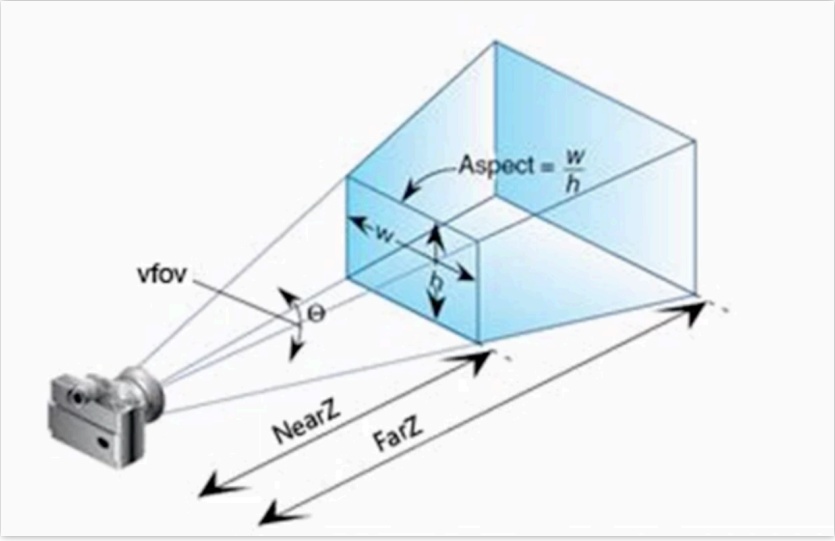
INTRODUCTION

AIMS

The main goal of this project is to give user the ability to navigate the scene with W, A, S, D keys and mouse movements. It is a very core feature of real world OpenGL applications such as games or CAD programs. Because we did not focus on this much on the course, I wanted to choose this topic. The scene in the project can be extended to any world once the core navigation is implemented. Several mesh objects can be added and even a game with a scenario can be created.

BACKGROUND REVIEW RESEARCH

To get started, we first need to think about what happens when we want to navigate a scene. We should obviously choose a perspective projection because we want to see the changes in the objects when we approach to or move away from them.



In the perspective projection, the viewing volume, the frustum, include all the things that we want to see in our view. Parameters like aspect ratio, near and far Z distances and the field of view(fov) angle define the frustum.

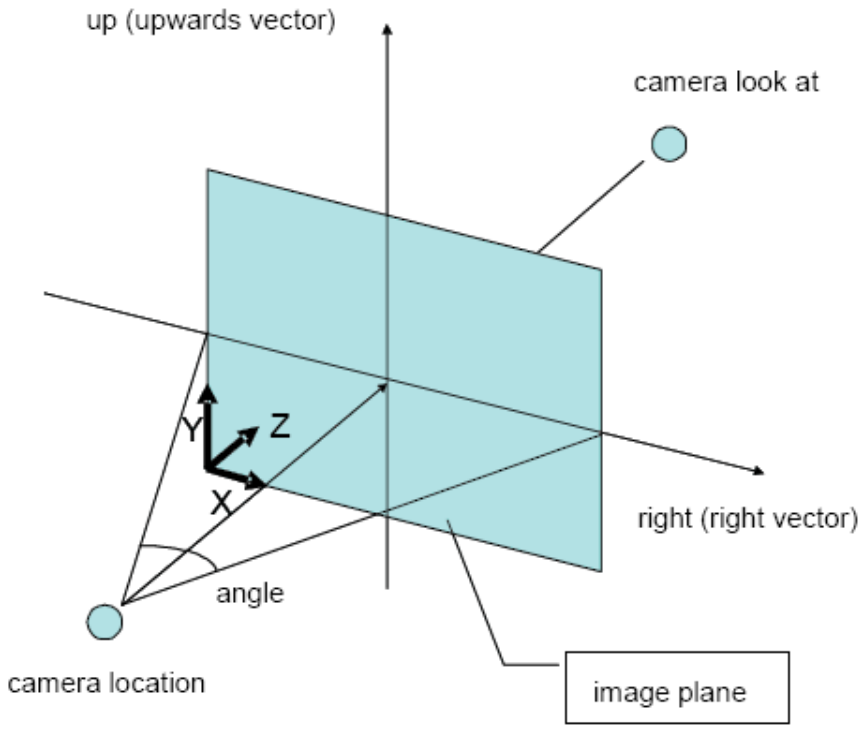
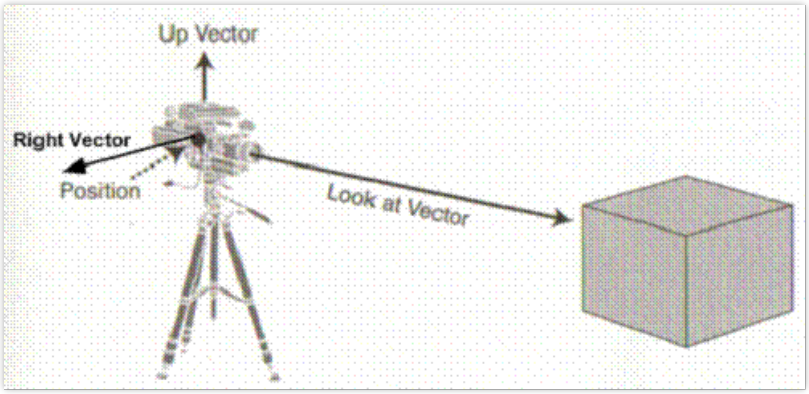


Image plane, where the projection of our objects fall, is essentially what camera sees. In navigation, the main aim of our object, our camera may be looking at up or down. When this happens (when we change the pitch or yaw angle), the image plane becomes unparalleled to Y or X axis of our world. That's why we need to define the upwards and right vector of our cameras.



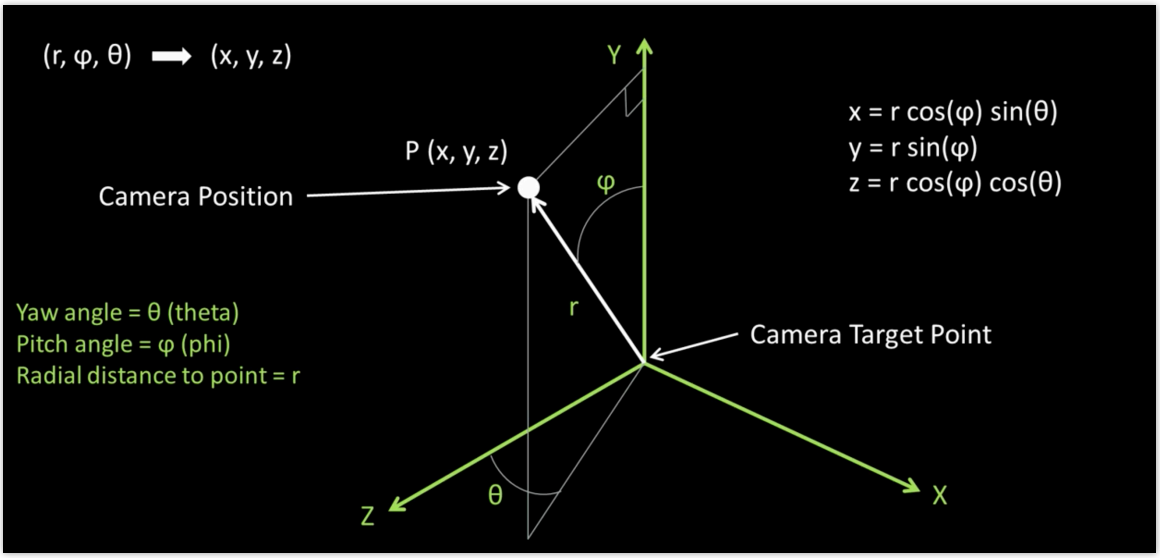
Beside from the up and right vector, our camera should also have a look at vector. This vector defines the direction where our camera looks at. In the navigation process, changing the angle between the look at vector and up or right vectors corresponds to changing the yaw and pitch angle. We change the pitch angle when we look at up or down and we change the yaw angle when we look at right or left.

As we change the yaw and pitch angles, our image plane changes as well. To reflect this change to the scene, we should switch from spherical coordinates that utilizes yaw and pitch angles to cartesian coordinates that our world objects are built upon. This coordinate change is given in Wikipedia as follows:

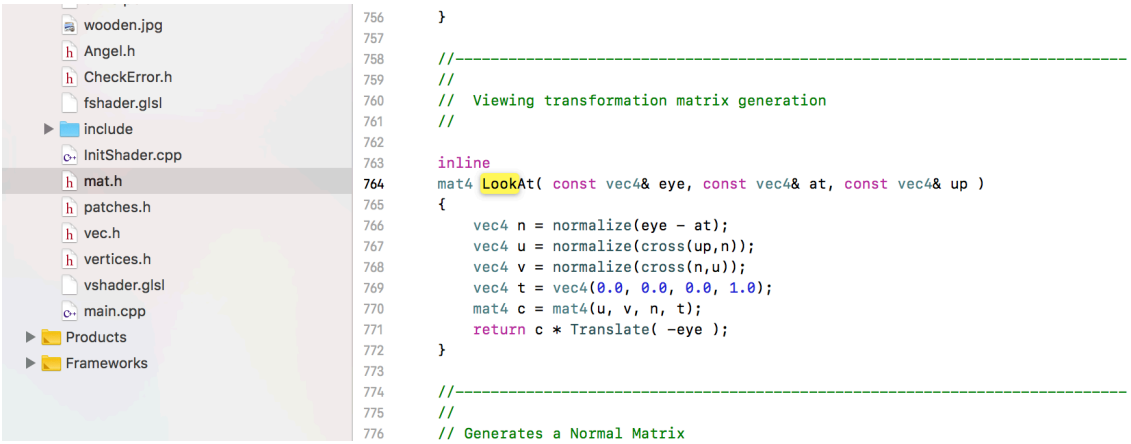
Conversely, the Cartesian coordinates may be retrieved from the spherical coordinates (*radius* r, *inclination* θ, *azimuth* φ), where $r \in [0, \infty)$, $\theta \in [0, \pi]$, $\varphi \in [0, 2\pi)$, by:

$$x = r \sin\theta \cos\varphi$$
$$y = r \sin\theta \sin\varphi$$
$$z = r \cos\theta$$

In our world, Y axis is parallel with the up vector of the world. So, we change the above formulas a little and get: (Z and Y changed)



So, this is how we changed from spherical coordinates to cartesian coordinates. We now have the necessary camera look vector, camera position, target position and direction unit vector of the camera.



Thanks to mat.h library, we have the LookAt function that takes the calculated vectors and gives us the required View Matrix that can directly be applied to objects. (Actually all the scene around our camera moves/rotates when move/rotate the camera)

PROJECT SPECIFICATION / SOLUTION

The project includes a camera, boxes and floor with textures/

CAMERA

Camera is represented by the following attributes:

```
// camera attributes
vec3 camera_position(0.0, 1.0, 5.0);
vec3 target_position(0.0, 0.0, 0.0);
vec3 camera_look;
vec3 camera_up(0.0, 1.0, 0.0);
vec3 camera_right(0.0, 0.0, 0.0);
vec3 world_up(0.0, 1.0, 0.0);
float yaw_angle = M_PI;
float pitch_angle = 0.0;
float fov_angle = 45.0;
```

When we navigate the scene with either keys or mouse, we actually modify these attributes and calculate the view matrix using these

attributes. For the modification of these attributes, 2 methods defined as move_camera and rotate_camera.

```
''
void move_camera(vec3 offset){
    camera_position += offset;

    vec3 new_look;
    new_look.x = cosf(pitch_angle) * sinf(yaw_angle);
    new_look.y = sinf(pitch_angle);
    new_look.z = cosf(pitch_angle) * cosf(yaw_angle);

    camera_look = normalize(new_look);

    camera_right = normalize(cross(camera_look, world_up));
    camera_up = normalize(cross(camera_right, camera_look));

    target_position = camera_position + camera_look;
}
```

Move_camera is for moving around the scene with W, A, S and D keys.

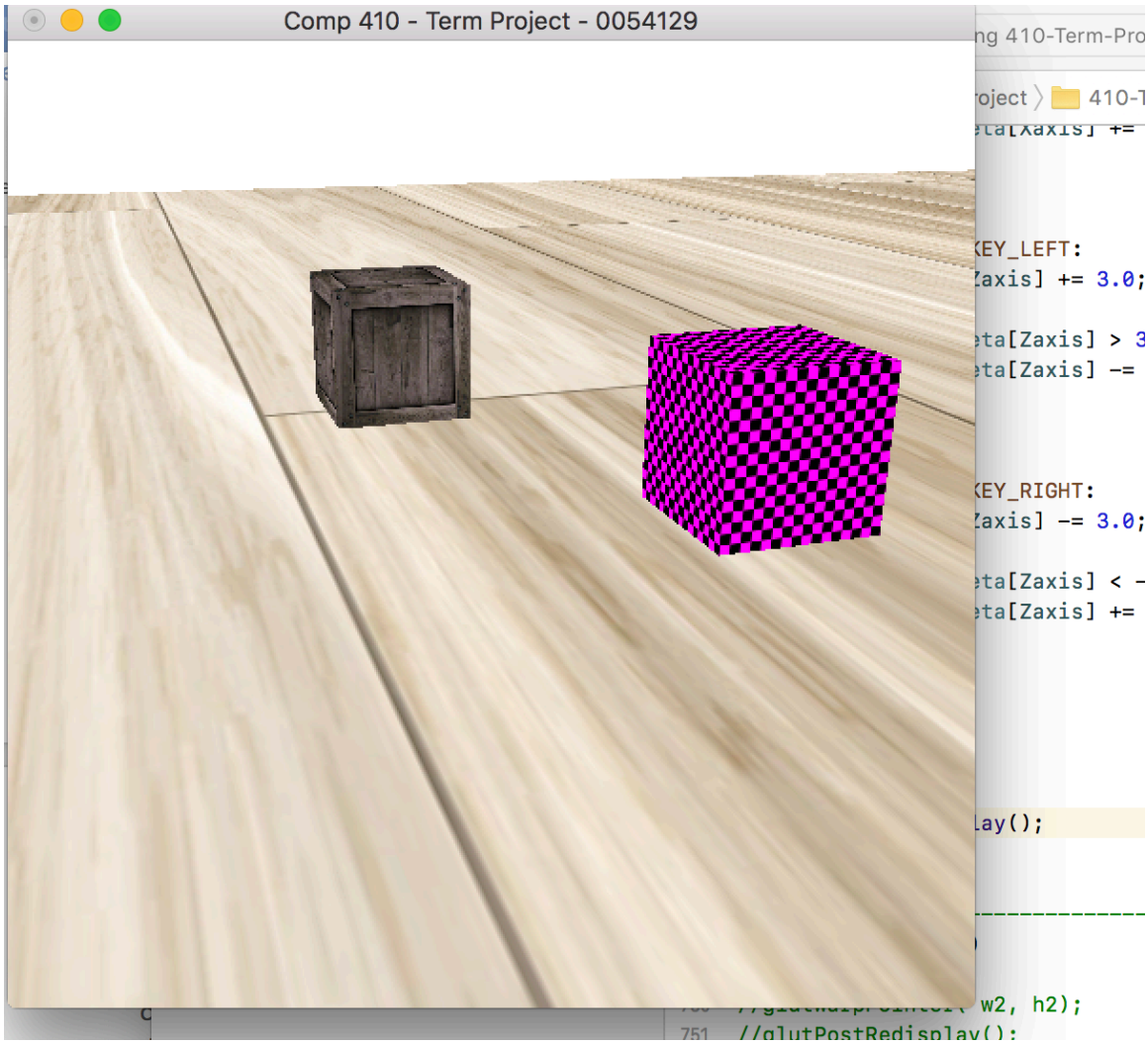
```
void rotate_camera(float yaw, float pitch){
yaw_angle += DegreesToRadians * yaw;
pitch_angle += DegreesToRadians * pitch;
float min = -M_PI / 2.0f + 0.1f;
float max = M_PI / 2.0f - 0.1f;
if( pitch_angle < min){
    if(min > max){
        pitch_angle = max;
    } else {
        pitch_angle = min;
    }
} else {
    if(pitch_angle > max){
        pitch_angle = max;
    } else {
        //pitch_angle = pitch_angle;
    }
}

vec3 new_look;
new_look.x = cosf(pitch_angle) * sinf(yaw_angle);
new_look.y = sinf(pitch_angle);
new_look.z = cosf(pitch_angle) * cosf(yaw_angle);

camera_look = normalize(new_look);

camera_right = normalize(cross(camera_look, world_up));
camera_up = normalize(cross(camera_right, camera_look));

target_position = camera_position + camera_look;
}
```

Rotate_camera is for the orbital movement of the camera. When we move the mouse up and down or right and left, this function helps us. Notice that pitch angle is restricted, this is because our user cannot flip. It can look at to the sky or to the ground within a certain range.

OBJECTS

There are currently 3 objects in the scene. 2 boxes and a floor. Floor can be considered as a box as well. It has a very narrow thickness (Y dimension) and very large extension to X and Z dimensions.

All objects have some kind of textures. One of the boxes has a craft texture. The floor has parquet texture. The other box has a checkerboard texture. Craft and parquet textures are first found as a jpg file and then converted to P3 PPM file. My readPPM function from projects help me to apply these textures to these objects.



SUMMARY
CONCLUSION

The user can now successfully navigate the scene just like she/he is in a game. Any feature that we have focused on our projects (shading, lights, object picking) can be added to this scene as well. I could not add because I tried to solve challenges that I came across in most of my project time by myself.