BRAC UNIVERSITY

Inspiring Excellence

Department of Computer Science and Engineering

Course Code: CSE 423	**Credits: 0.75**
Course Name: Computer Graphics	

Lab 03

OpenGL 3D Introduction and Transformation

I. Topic Overview:

In today's class, we will introduce 3D graphics using PyOpenGL. OpenGL is a powerful graphics library that allows us to render 3D objects efficiently. We will focus on understanding how 3D transformations work and how they are applied to objects in a scene.

In 3D graphics, transformations are essential for positioning, rotating, and scaling objects. These transformations are applied using matrices that modify the coordinates of objects. The three fundamental transformations are:

1. Translation – Moves an object from one place to another in 3D space.
   glTranslatef (tx, ty, tz) for translation
2. Rotation – Rotates an object around an axis (X, Y, or Z).
   glRotatef (angle, x, y, z) for rotation
3. Scaling – Changes the size of an object along the X, Y, and Z axes.
   glScalef (sx, sy, sz) for scaling

To visualize 3D objects properly, we will use one new projection technique:

- **Perspective Projection** (Objects appear smaller as they move further away)

  gluPerspective (fov, aspect, near, far)

gluLookAt() positions the camera and defines its viewing direction.
gluLookAt(eyeX, eyeY, eyeZ, // Camera position

centerX, centerY, centerZ, // Target point

upX, upY, upZ); // Up direction

By the end of the session, students should understand how to apply 3D transformations and projection techniques to create and visualize objects using PyOpenGL. The key takeaways from this assignment are:

- We use transformation matrices to manipulate objects in 3D space.
- **The order of transformations** affects the final position and orientation of objects.
- Projection techniques help us visualize 3D scenes correctly.

II. Anticipated Challenges:

a. The students need to carefully understand how the perspective projection works and next, set the camera and look at target positions otherwise they will not be able to understand the 3D world coordinate system which will not allow them to manipulate the gun, enemy, and bullet positions. It needs to be understood that the window coordinate system and the 3D world coordinate system do not align.

b. The grid in the game has a pattern in creating it which the students need to comprehend as this grid needs to be created dynamically and NOT through hardcode!

c. A very important feature in the game is the cheat mode which will automatically fire bullets by rotating 360° if an enemy is in line of sight. To do this, the students will only need to combine the existing functions that they would have made to fire bullets, turn the gun, and use a simple flag variable to perpetuate this feature.

III. Activity Discussion

******You can not use any other OpenGL function to implement this assignment other than what is given in this template code(3D Lab Assignment Template). Doing so will induce a 50% penalty on the overall assignment mark. [This includes glutTimerFunc too] You have to run the above code file and understand its contents to proceed doing the assignment.******

PyOpenGL transformation example & definitions:

```
glPushMatrix()

# First Transformation: Translation
glTranslatef(2.0, 0.0, 0.0)  # Translation
glColor3f(1.0, 0.0, 0.0)  # Red
draw_square()

# Second Transformation: Rotation
glRotatef(45, 0.0, 0.0, 1.0)  # Rotation
glColor3f(0.0, 1.0, 0.0)  # Green
draw_square()

# Third Transformation: Translation
glTranslatef(2.0, 0.0, 0.0)  # Translation
glColor3f(0.0, 0.0, 1.0)  # Blue
draw_square()

# Fourth Transformation: Rotation
glRotatef(45, 0.0, 0.0, 1.0)  # Rotation
draw_square()

# Fifth Transformation: Translation
glTranslatef(2.0, 0.0, 0.0)  # Translation
glColor3f(1.0, 0.0, 1.0)  # Purple
draw_square()

glPopMatrix()
```

Let's break down this sequence of code line by line to understand what each part does:

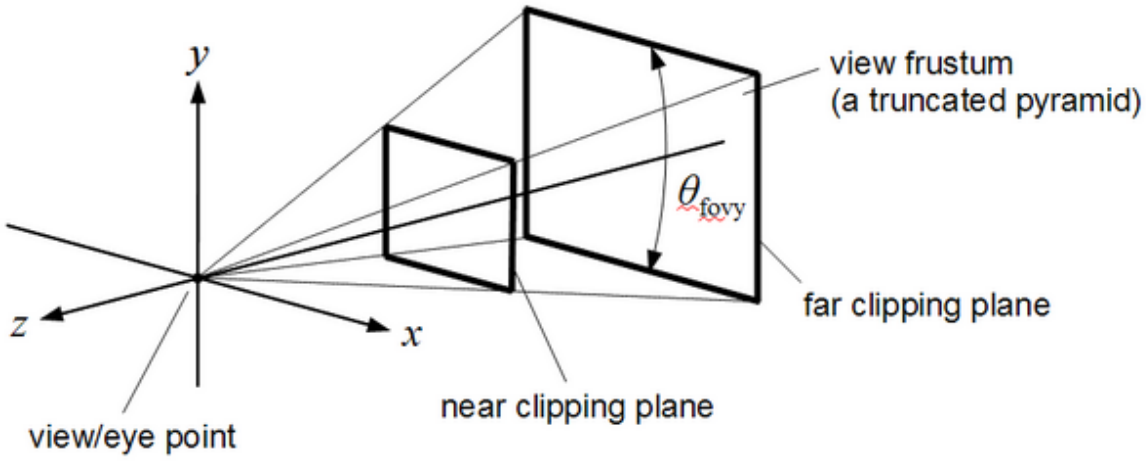
1. glPushMatrix(): Saves the current transformation matrix state to the stack. This allows you to apply transformations and later revert to this saved state using glPopMatrix().

2. glTranslatef(2.0, 0.0, 0.0): Translates (moves) the coordinate system 2 units to the right (along the x-axis). This transformation will affect all subsequent drawing commands.
3. glColor3f(1.0, 0.0, 0.0): Sets the current color to red.
4. draw_square(): Draws a square at the new position (translated 2 units to the right).
5. glRotatef(45, 0.0, 0.0, 1.0): Rotates the coordinate system 45 degrees around the z-axis. This transformation will affect all subsequent drawing commands.
6. glColor3f(0.0, 1.0, 0.0): Sets the current color to green.
7. draw_square(): Draws a square at the current position, which is translated 2 units to the right and rotated 45 degrees.
8. glTranslatef(2.0, 0.0, 0.0): Translates the coordinate system another 2 units to the right. This transformation will affect all subsequent drawing commands.
9. glColor3f(0.0, 0.0, 1.0): Sets the current color to blue.
10. draw_square(): Draws a square at the new position, which is now translated 4 units to the right and rotated 45 degrees.
11. glRotatef(45, 0.0, 0.0, 1.0): Rotates the coordinate system another 45 degrees around the z-axis. This transformation will affect all subsequent drawing commands.
12. glColor3f(1.0, 1.0, 0.0): Sets the current color to yellow.
13. draw_square(): Draws a square at the current position, which is translated 4 units to the right and rotated 90 degrees in total.
14. glTranslatef(2.0, 0.0, 0.0): Translates the coordinate system another 2 units to the right. This transformation will affect all subsequent drawing commands.
15. glColor3f(1.0, 0.0, 1.0): Sets the current color to purple.
16. draw_square(): Draws a square at the new position, which is translated 6 units to the right and rotated 90 degrees in total.
17. glPopMatrix(): Restores the matrix state to what it was before the glPushMatrix(). This undoes all the transformations applied since the glPushMatrix().

By applying these transformations in sequence, each draw_square() command places a square at a different position and orientation, creating a series of squares with combined translations and rotations. The transformations accumulate, so each square is drawn relative to the previously applied transformations.

PyOpenGL - gluPerspective() in details:

The purpose of the 4 parameters is to define a **view frustum (**gluPerspective(fovY, aspect, zNear, zFar)**)**, like this:



where nothing outside of the frustum should be visible on screen(To accomplish this, the parameters are used to calculate a 4x4 matrix, which is then used to transform each vertex into the so-called clip space. There, testing if a vertex is inside the frustum or not is trivial)

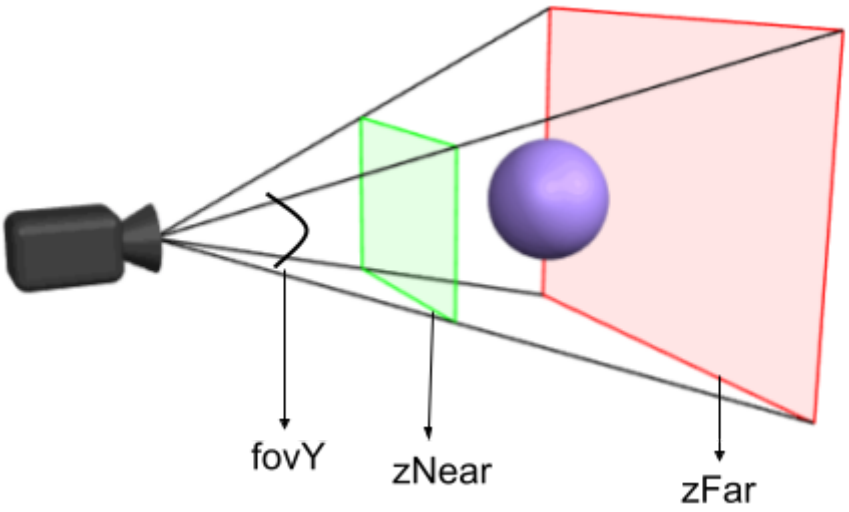
The field of view parameter is the angle in between a plane passing through the camera position as well as the top of your screen & another plane passing through the camera position and the bottom of your screen. The bigger this angle is, the more you can see of the world - but at the same time, the objects you can see will become smaller.

To observe its influence, you can code a simple program where you can gradually increase/decrease the fovY via keypress - then render some spheres or other basic objects & see what happens as you change it.

The aspect ratio is your viewport's aspect ratio. (In the graphic above, the viewport is located at the near clipping plane) Being able to define it at will makes sense, since your viewport's aspect ratio may vary.

The zNear & zFar values define the distance between the camera position & the near and far clipping planes, respectively. Nothing closer to the camera than zNear or farther away than zFar will be visible. Both values must be > 0, and obviously, zFar > zNear. zFar should ideally be chosen so that everything you want to render is visible, but making it larger than necessary wastes depth buffer precision & may lead to flickering effects, called z-fighting. Likewise, setting zNear too close to the camera may cause the same effect - in fact, having a reasonable zNear value is more important than zFar. If you want

to know exactly why this happens, you should read some in-depth explanations, like this one or this one



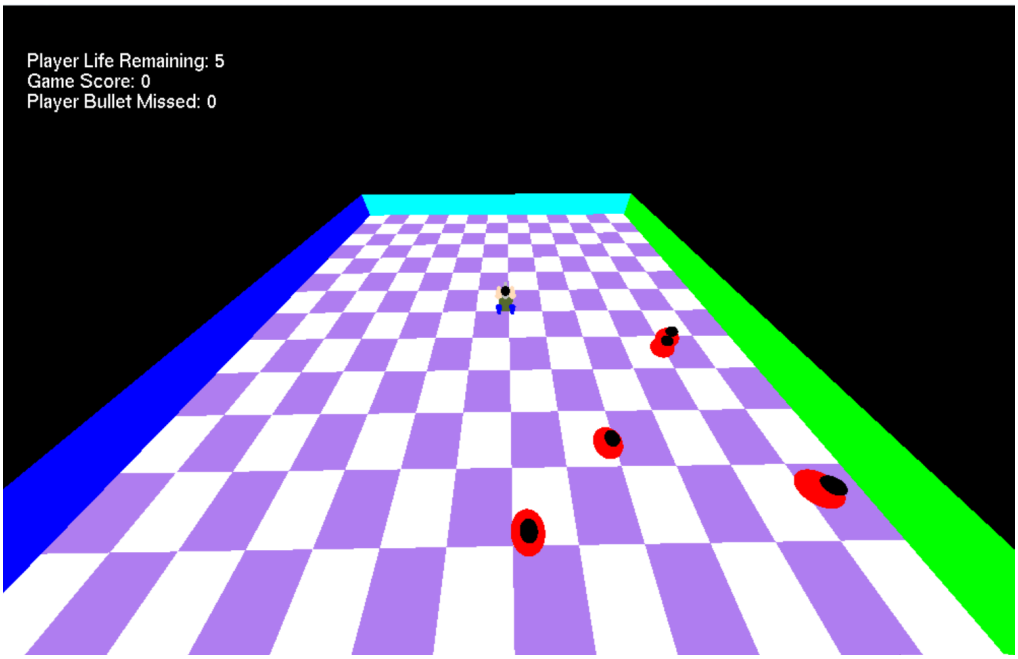
IV. Activity Task

Bullet Frenzy - A 3D Game with Player Movement, Shooting, & Cheat Modes

Overview:

This assignment involves creating a 3D game in which the player controls a character with a gun that can move, rotate, and shoot bullets at enemies. The game environment includes a grid, moving enemies (represented as **spheres**), and features such as cheat modes and camera controls. The game also contains visual feedback on the player's actions and the status of the game.

Gaming Setup:
The start of the game will appear exactly like the image below. The player spawns at the center of the grid.

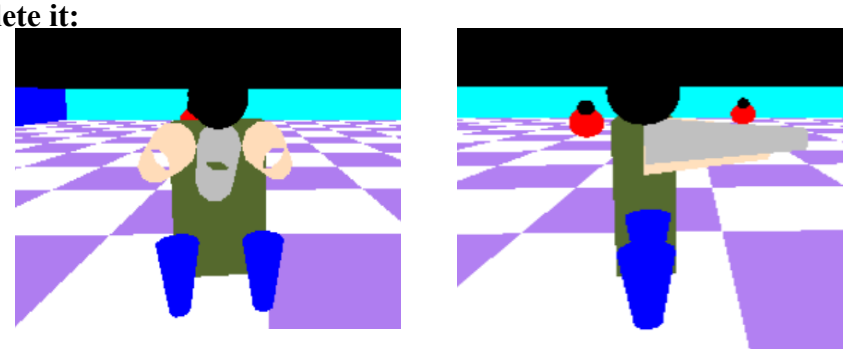


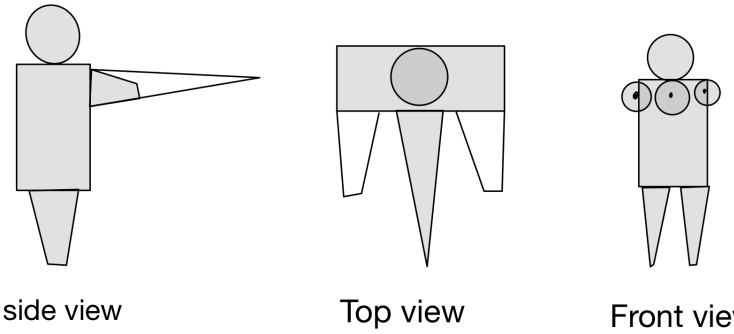
Drawing Components:

The game has three crucial drawing elements.
i) The player **(1 player)**

The player should be exactly the one in the picture below. It should take the following shapes to complete it:

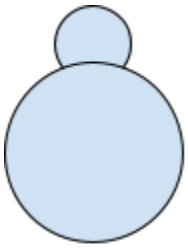
1. **Sphere**
2. **Cylinders**
3. **Cuboids**



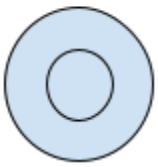


ii) Enemies **(5 enemies all the time)**

The enemy should be exactly like the picture. It should take two spheres to complete.

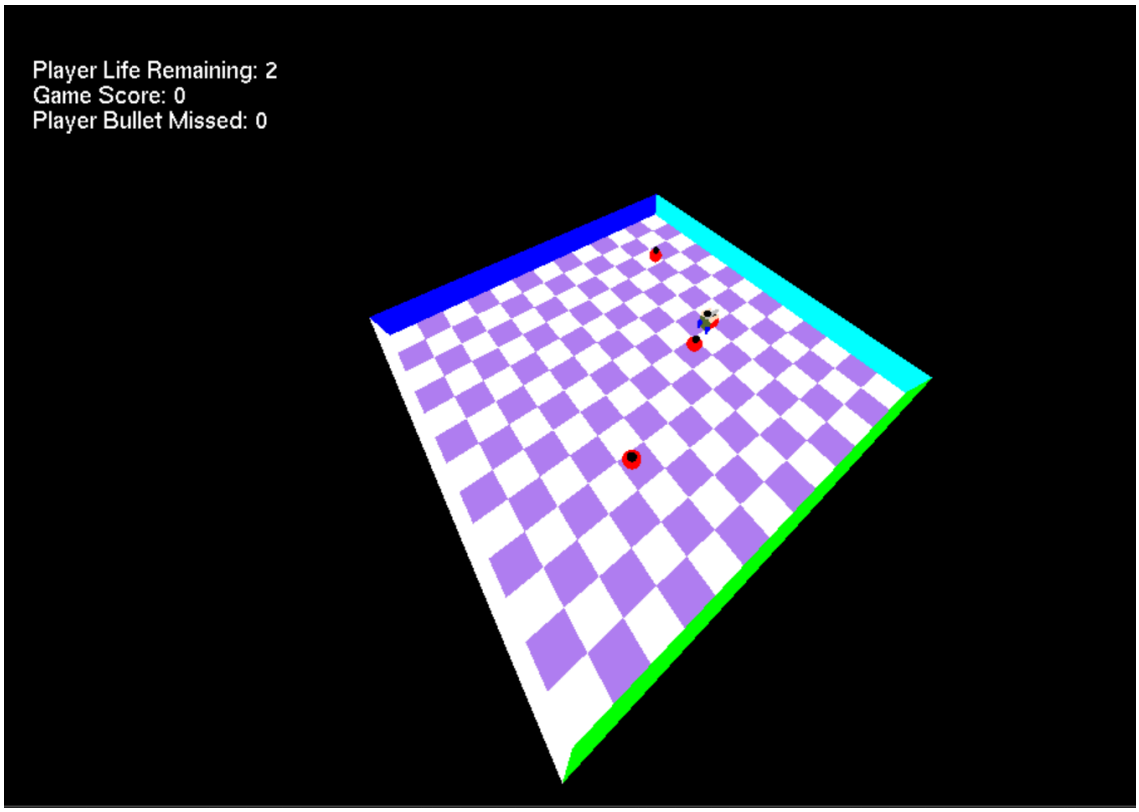


Side/Front View



Top View

iii) The grid floor is surrounded by four vertical boundaries. **(should look exactly like in the image)**



Game Features:

1. **Player Movement and Gun Control**:
    - The player controls a gun that can move, rotate, and fire at enemies.
    - The player can move the gun forward and backward using the **W (forward)** and **S (backward)** keys respectively.
    - The player can rotate the gun left or right using the **A (left)** and **D (right)** keys.

- The player can fire bullets with the **left mouse button**.

  Player Movement

2. **Camera Control**:
   - The camera can be moved up and down using the **UP** and **DOWN** arrow keys.
   - The camera can rotate(around the whole game floor) left and right using the **LEFT** and **RIGHT** arrow keys.

     Camera Control

   - The camera can follow the gun **(first-person mode)** or be fixed at a default position **(default third-person mode)** based on the **RIGHT** mouse button click.

     Camera Perspective

3. **Cheat Modes**:
   - The **C** key t**oggles** cheat mode, which continuously rotates the gun and fires bullets automatically **at enemies when in line of sight**. The player can still move forward and backward in cheat mode.
   - The **V** key toggles automatic gun following which adjusts the camera's perspective **when cheat mode is active only. This feature can only be seen in the first-person mode.**

     Cheat Mode

4. **Game Over and Restart**:
   - The game ends when the player's life reaches **zero** or the player misses **10** bullets. **Life reduces by 1** if an enemy touches the player.
   - The player **will lie down** when the game is over.
   - The **R** key is used to restart the game after it ends.
     Reset to Life remaining 5, Game Score 0 and Bullets Missed 0

     Game Over **Game Feedback**

5. **Bullets and Enemies**:
   - Bullets are made of **cubes** and move in the direction of the gun's angle.
   - Enemies will **continuously** move towards the player. If the player moves, the enemies will follow the player accordingly.
   - If an enemy is hit, **it will respawn at a random point**, i.e. there will always be 5 enemies.
   - The shape of enemies **(spheres) will shrink and expand continuously.**
   - The game checks for interactions between **the bullets and the enemies** and the **player and the enemies**.

Functions and Descriptions: from the template code → **3D Assignment Template**
