Lab Report

Name: **Arnob Das Shacha**

ID: **221-15-5277**

Section: 61_D1

Course Code: CSE 422

Experiment Name: 2D implementation (Translation)

Submitted To:

Sayeda Tasnim Alvi

Senior Lecturer

Dept of CSE

Daffodil International University

Title

2D Implementation: Translation

Introduction

In this project, I implemented **2D Translation** using OpenGL. A simple moving vehicle (car-like shape) was designed using basic 2D shapes. The object continuously moves along the **x-axis** by updating its position variable p inside the display() function. When the object reaches the boundary, it resets and moves again, creating a smooth translation animation. This demonstrates how translation transforms an object's coordinates over time in 2D graphics.

Contents

Functions Used

- **init()**
  Sets the background color and initializes the orthographic projection using `glOrtho()`.
- **display()**
  Contains the drawing logic and updates the value of `p` to achieve translation. It redraws the object to create the animation effect.
- **drawCircle()**
  Draws a circle using the parametric equation of a circle with cosine and sine.

Shapes Used

- **Rectangle (GL_QUADS)**
  Used for drawing the car body and windows.
- **Circle (GL_POLYGON)**
  Used for drawing the car wheels.

Code

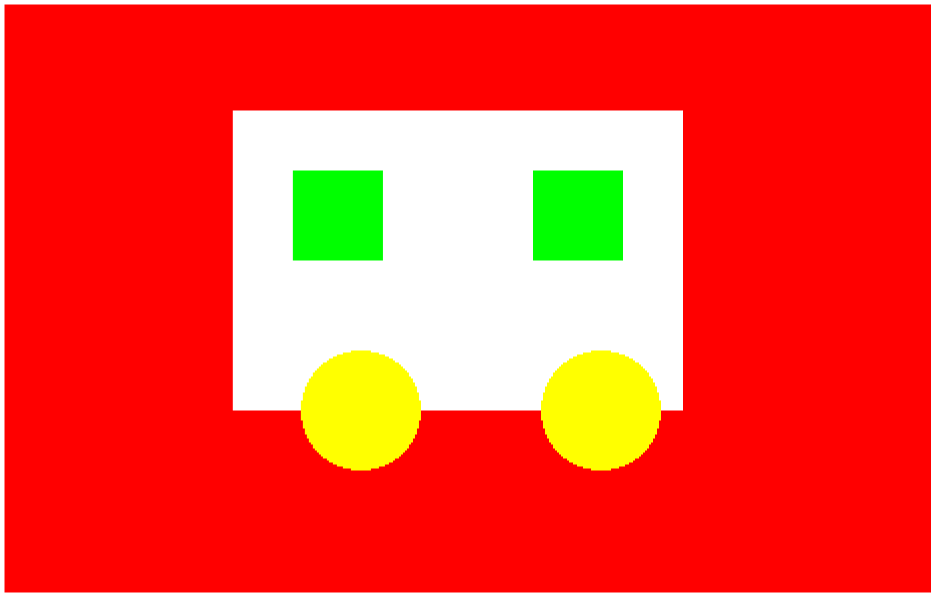
```
#include <GL/gl.h>

#include<windows.h>

#ifdef APPLE

#include <GLUT/glut.h>

#else
```

```
#include <GL/glut.h>
#endif
#include <stdlib.h>
#include <math.h>
void drawCircle(int h, int k, int rx, int ry)
{
    glColor3f(1.0, 0.0, 0.0);  // Yellow color
    glBegin(GL_POLYGON);
    for (int i = 0; i <= 360; i++) {
        glVertex2f(h + rx * cos(3.14159 * i / 180), k + ry * sin(3.14159 * i / 180));
    }
    glEnd();
}
float p= 20.0;
int h, k, rx, ry;
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    if(p<=20) //moving limit with the display measurement
        p=p-.0005; // changing the object position for redisplaying
    else
        p=5; // For backing the object continuously
     glutPostRedisplay(); // To redraw the object in the display
    glBegin(GL_QUADS);
    glColor3f(1.0, 1.0, 1.0);

    glVertex2f(p-15,15);
    glVertex2f(p,15);   // Right to left
```

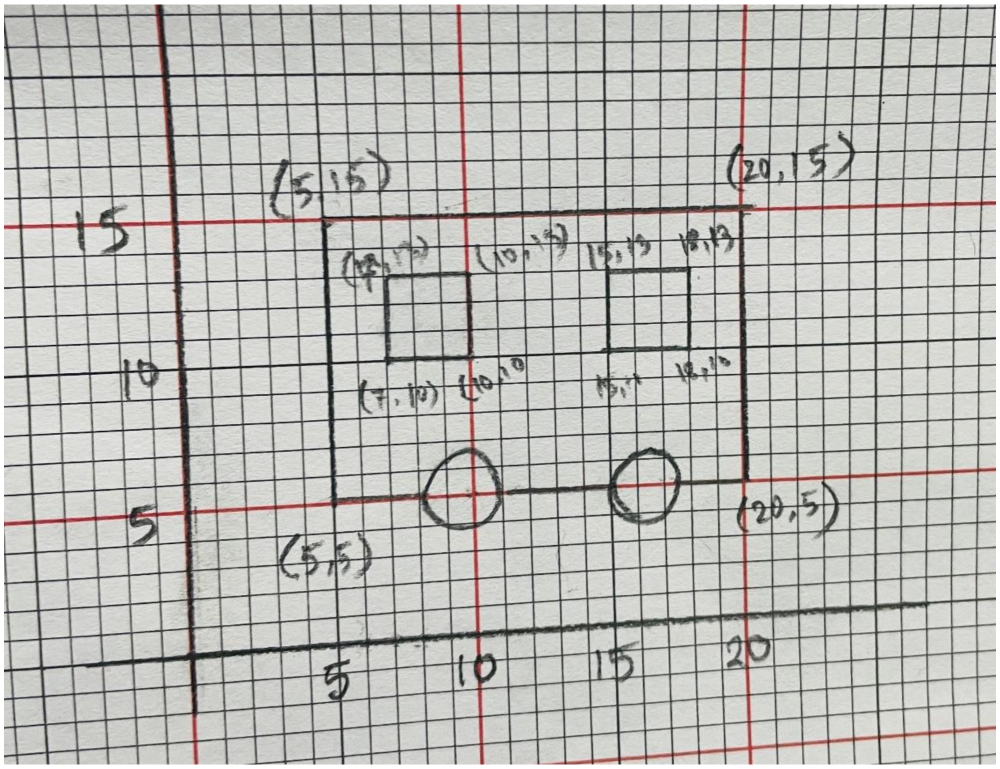
```
glVertex2f(p,5);
glVertex2f(p-15,5);

glColor3f(0.0, 1.0, 0.0); // Yellow color
glVertex2f(p-13,13);
glVertex2f(p-10,13);   // left win
glVertex2f(p-10,10);
glVertex2f(p-13,10);

glColor3f(0.0, 1.0, 0.0); // Yellow color
glVertex2f(p-5,13);
glVertex2f(p-2,13);   // r8 win
glVertex2f(p-2,10);
glVertex2f(p-5,10);

glEnd();

h = p - 11.5, k = 5, rx = 2, ry = 2;
glColor3f(1.0, 1.0, 0.0);
glBegin(GL_POLYGON);
for (int i = 0; i <= 360; i++) {
    glVertex2f(h + rx * cos(3.14159 * i / 180), k + ry * sin(3.14159 * i / 180));
}

glEnd();

h = p - 2.5, k = 5, rx = 2, ry = 2;
glColor3f(1.0, 1.0, 0.0);
```

```
    glBegin(GL_POLYGON);
    for (int i = 0; i <= 360; i++) {
        glVertex2f(h + rx * cos(3.14159 * i / 180), k + ry * sin(3.14159 * i / 180));
    }
    glEnd();
    glFlush();
}
void init(void)
{
    glClearColor (1.0, 0.0, 0.0, 0.0); // Background Color
    glOrtho(-20,20,-20,20,-20,20);
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); //Single Frame
    glutInitWindowSize (600, 600);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("moving_object");
    init();                              // Set up constants with default values
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Output



Graph



Discussion

The main goal of this project was to apply **2D Translation** on a graphical object. I created a vehicle using rectangles and circles. A variable `p` was used to represent the x-position of the object. In each frame, I updated:

```
p = p - 0.0005;
```

This continuously shifts the object to the left. When it reaches the limit, the value resets, allowing indefinite animation. The combination of `glutPostRedisplay()` and updated coordinates produces smooth translation. This project helped me understand how object movement is implemented through simple coordinate transformation in OpenGL.