Computer Graphics (UCS505)

Project On

Multiplayer Ping Pong Game

Submitted By

Abhimanyu Mehta	102103045
Yash Pratap Rajoria	102103526
Varada Gupta	102103542

Group: 3CO2

Submitted To:

Dr. Jyoti Rani

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

1. INTRODUCTION TO THE PROJECT

Ping pong, also known as table tennis, is a 2-player game that is played on a table with a small ball, paddles, and a net. The objective of the game is to hit the ball back and forth over the net with the paddle, with the aim of making the ball land on the opponent's side of the table without them being able to return it. The screen, in this case, represents the table, the net is represented by a dotted line running through the center of the screen vertically. The 2 players can control their respective paddles using keyboard inputs and the ball replicates the physical motion of a ball being reflected after hitting the paddle/ boundary of the table.

Gameplay Instructions

In order to play the game, press the “Left Mouse Button” to launch the ball in a random direction with a random speed. The players can control the paddle movement in two different directions i.e. Up and Down.Player1 uses the keys ‘q’ and ‘a’ to move the left paddle up and down while Player2 uses the keys ‘o’ and ‘l’ to move the right paddle up and down. During the game or after a round, if at any time the player wishes to start a new game, he or she could press the “escape” key. The goal of the game is to return the ball without hitting the vertical wall on the players’ side.

Score Mechanism

If a player misses the ball and the ball hits the vertical part of the window on the player’s side, the other player is granted a point. The scores for both the players are visible clearly on the screen.

2. COMPUTER GRAPHICS CONCEPTS USED

We've used various foundational concepts of Computer Graphics and the functions of OpenGL working on this project. Some of them are mentioned as follows:

Concepts used-

- **Translation**: The concept of translation is used in our game while moving the ball and paddles.
- **Drawing Circle**: For drawing the ball we used the concept of trigonometry.
- **Drawing Quadrilaterals**: Rectangles are drawn for the paddles and the centre dashed net line.
- **Game Concept**: Ping pong is a famous virtual as well as physical game.
- **Keyboard Inputs**: Now for playing the game the users need some kind of input to give. This can be done very easily by using a glut function 'keyboard' and then by specifying the role of the keys in the switch case. In each switch case, you need to define what that key will do.
- **Hitting the paddles and boundaries**: This is the concept that comes into play when the ball hits the paddles or the window boundary. This function defines the trajectory of the ball after hitting. We used the concept by applying the if-else conditions and accordingly changing the direction of the ball.
- **Motion of the Ball**: The concept used for this particular feature of our game which handles the motion of the ball.

Some of the OpenGL functions used

- **glColor3f():** It takes 3 arguments: the red, green, and blue components of the color you want. After you use glColor3f, everything you draw will be in that color.
- **glBegin() and glEnd():** The glBegin subroutine delimits the vertices that define a primitive or group of like primitives.
- **glVertex2f():** It takes 2 parameters, the x and y coordinates of the vertex to be selected.
- **glClear(GL_COLOR_BUFFER_BIT):** glClear function takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared. The value GL_COLOR_BUFFER_BIT indicates the buffers currently enabled for color writing.
- **glPushMatrix():** glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. After a glPushMatrix call, the matrix on top of the stack is identical to the one below it.
- **glTranslatef():** The glTranslatef function produces the translation specified by (x, y, z).
- **glPopMatrix():** glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack.
- **glutSolidSphere():** Renders a solid sphere. Takes 3 parameters – radius, slices and stacks.
- **glutSwapBuffers():** Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the back buffer of the layer in use of the current window to become the contents of the front buffer. The contents of the back buffer then become undefined.
- **glutIdleFunc(NULL):** glutIdleFunc sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received. Passing NULL to glutIdleFunc disables the generation of the idle callback.
- **glutPostRedisplay():** Mark the normal plane of the current *window* as needing to be redisplayed. In the next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback.
- **glClearColor():** glClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers.

- **glMatrixMode(GL_PROJECTION):** It sets the current matrix mode as GL_PROJECTION which applies subsequent matrix operations to the projection matrix stack.
- **glLoadIdentity():** glLoadIdentity replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix with the identity matrix.
- **gluOrtho2D():** gluOrtho2D() sets up a two-dimensional orthographic viewing region.
- **glutMouseFunc():** sets the mouse callback for the *current window*.
- **glutLeyboardFunc():** sets the keyboard callback for the *current window*
- **GL_LINES** – the OpenGL mode for plotting lines
- **GL_QUADS** – the OpenGL mode for plotting quadrilaterals
- **glutStrokeCharacter()** – Without using any display lists, glutStrokeCharacter renders the character in the named stroke font

3. USER DEFINED FUNCTIONS

- **void drawStrokeText(char*string, int x, int y, int z):** This function is used to render the score of the players in GLUT_STROKE_ROMAN font.
- **void drawCenterLines():** This function is used to draw the dashed net (set of thin rectangles with gaps in between) in the center of the screen.
- **void drawPaddle(int x, int y):** This function is used to draw the paddle for each player. The x,y coordinates are the top left coordinates of the paddle on the screen.
- **void drawBall(int x, int y):** This function is used to draw the white circular ball with a radius of 20 on the screen at x,y coordinates.
- v**oid startGame():** This function has the logic for ball movement when it is inside the screen area or when it reflects from the paddle or window boundaries. This function also checks for points scored by either of the players in case they miss the ball.
- **void reshape():** To reshape the window without affecting the ball movement.
- **void mouse(int button, int state, int x, int y):** The callback function for mouse input. Defines the behavior when left mouse button or middle mouse buttons are clicked using simple if-else conditions.
- **void keyboard(unsigned char key, int x, int y) :** Callback function for keyboard input. Helps players control the paddles in vertical directions. Stop the game on clicking ‘Escape’.

4. CODE

```
#define GL_SILENCE_DEPRECATION
#include <GLUT/glut.h>
#include <OpenGL/gl.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

static GLint windowSizeX = 800, windowSizeY= 1200;
static GLint orthoSizeX = 600, orthoSizeY = 400;
static char score_1[20], score_2[20];
static GLint player1_score = 0, player2_score = 0;
static GLint paddle_boundary = 350, paddle_height = 125, paddle_velocity = 16.0;
static GLint player1_paddle_y = 0, player2_paddle_y = 0, paddle_x = 595;
static GLfloat ball_velocity_x = 5, ball_velocity_y = 5, speed_increment = 2;
static GLint ball_pos_x = 0, ball_pos_y = 0, ball_radius = 20;
void init(void) {

   glClearColor (0.92,0.65, 0.24, 0.0);
   glShadeModel (GL_FLAT);
   srand(time(NULL));
}
void drawStrokeText(char* string, int x, int y, int z) {
   int i;
   glPushMatrix();
   glTranslatef(x, y+8, z);
   for (i = 0; i < 20; i++) {
      glutStrokeCharacter(GLUT_STROKE_ROMAN, string[i]);
   }
   glPopMatrix();
}
void drawCenterLines() {
   glColor3f(0.0, 0.0, 0.0);  // Black color for center lines
   glBegin(GL_LINES);
   for (int i = -410; i <= 410; i += 20) {
      glVertex2f(-2, i);
      glVertex2f(2, i);
   }
   glEnd();
}
```

```
void drawPaddle(int x, int y) {
    glPushMatrix();
    glTranslatef(x, y, 0);
    glColor3f(0.2, 0.2, 0.2);
   glBegin(GL_QUADS);
    int height = paddle_height / 2;
    glVertex2f(-5, height);
    glVertex2f(5, height);
    glVertex2f(5, -height);
    glVertex2f(-5, -height);
    glEnd();
    glPopMatrix();
}

void drawBall(int x, int y) {
    glPushMatrix();
    glTranslatef(x, y, 0);
    glColor3f(0.0, 0.0, 0.0);
    glutSolidSphere(ball_radius, 20, 16);
    glPopMatrix();
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    drawCenterLines();
    drawPaddle(-paddle_x, player1_paddle_y);
    drawPaddle(paddle_x, player2_paddle_y);
    drawBall(ball_pos_x, ball_pos_y);

    snprintf(score_1, sizeof(score_1), "%d", player1_score);
    drawStrokeText(score_1, -300, 200, 0);

    snprintf(score_2, sizeof(score_2), "%d", player2_score);
    drawStrokeText(score_2, 200, 200, 0);
    glutSwapBuffers();
    glFlush();
}
```

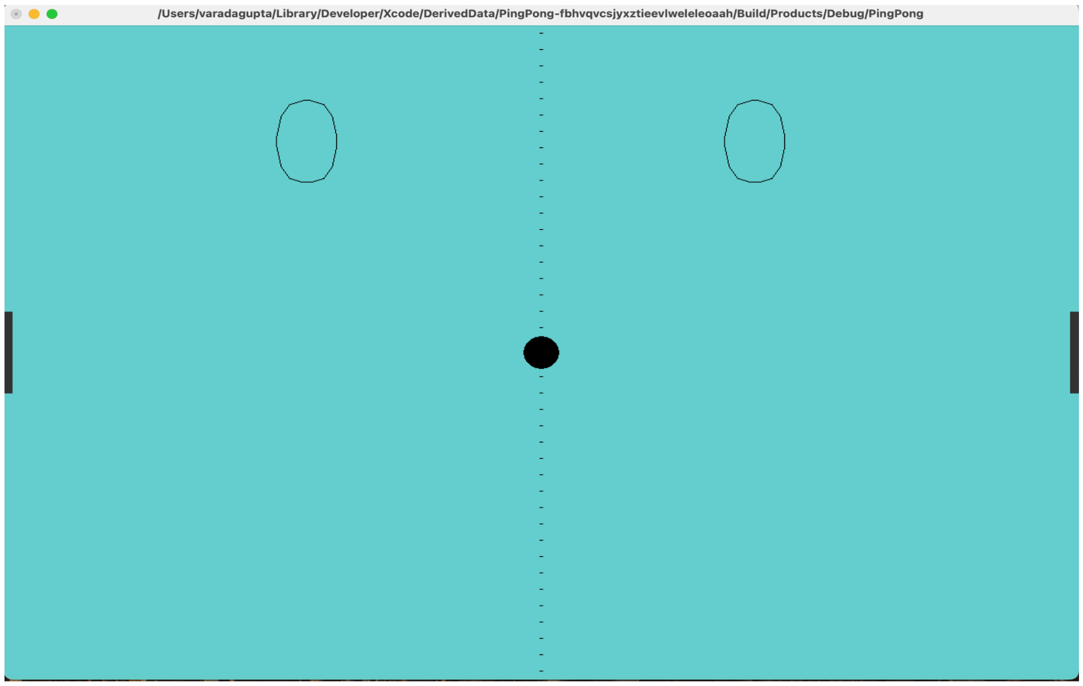
```
void startGame(void) {

    ball_pos_x += ball_velocity_x;
    ball_pos_y += ball_velocity_y;

    if (ball_pos_y + ball_radius > orthoSizeY || ball_pos_y - ball_radius < -orthoSizeY)
        ball_velocity_y = -ball_velocity_y;

    if (ball_pos_x - ball_radius - 5 < -paddle_x && ball_pos_x - ball_radius < -paddle_x)
            if (ball_pos_y < player1_paddle_y + paddle_height && ball_pos_y > player1_paddle_y -
paddle_height) {
            ball_velocity_x = -ball_velocity_x;
            ball_velocity_x += speed_increment;
            paddle_velocity += speed_increment;
        }

    if (ball_pos_x + ball_radius + 5 > paddle_x && ball_pos_x + ball_radius - 5 < paddle_x)
            if (ball_pos_y < player2_paddle_y + paddle_height && ball_pos_y > player2_paddle_y -
paddle_height){
            ball_velocity_x = -ball_velocity_x;
            ball_velocity_x += speed_increment;
            paddle_velocity += speed_increment;
        }

    if (ball_pos_x + ball_radius > orthoSizeX) {
        player1_score++;
        printf("Player 1 = %d\n", player1_score);
        ball_velocity_x = -ball_velocity_x;
    }

    if (ball_pos_x - ball_radius < -orthoSizeX) {
        player2_score++;
        printf("Player 2 = %d\n", player2_score);
        ball_velocity_x = -ball_velocity_x;
    }
    if (player1_score >= 7 || player2_score >= 7) {
        printf("Game Over!\n");
        glutIdleFunc(NULL);
    }
    glutPostRedisplay();
}
```

```
void reshape(int w, int h) {
   glViewport(0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-orthoSizeX, orthoSizeX, -orthoSizeY, orthoSizeY, -100, 100);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
void mouse(int button, int state, int x, int y) {
   switch (button) {
      case GLUT_LEFT_BUTTON:
         if (state == GLUT_DOWN)
            glutIdleFunc(startGame);
         break;
      case GLUT_MIDDLE_BUTTON:
         if (state == GLUT_DOWN) {
            ball_pos_x = ball_pos_y = 0;
            player1_paddle_y = player2_paddle_y = 0;
            player1_score = player2_score = 0;
            glutIdleFunc(NULL);
         }
         break;

         default:
            break;
         }
         }
         void keyboard(unsigned char key, int x, int y) {
            switch (key) {

               case 'q':
                  if (player1_paddle_y < paddle_boundary)
                     player1_paddle_y += paddle_velocity;
                  glutPostRedisplay();
                  break;

               case 'a':
                  if (player1_paddle_y > -paddle_boundary)
                     player1_paddle_y -= paddle_velocity;
                  glutPostRedisplay();
                  break;
```

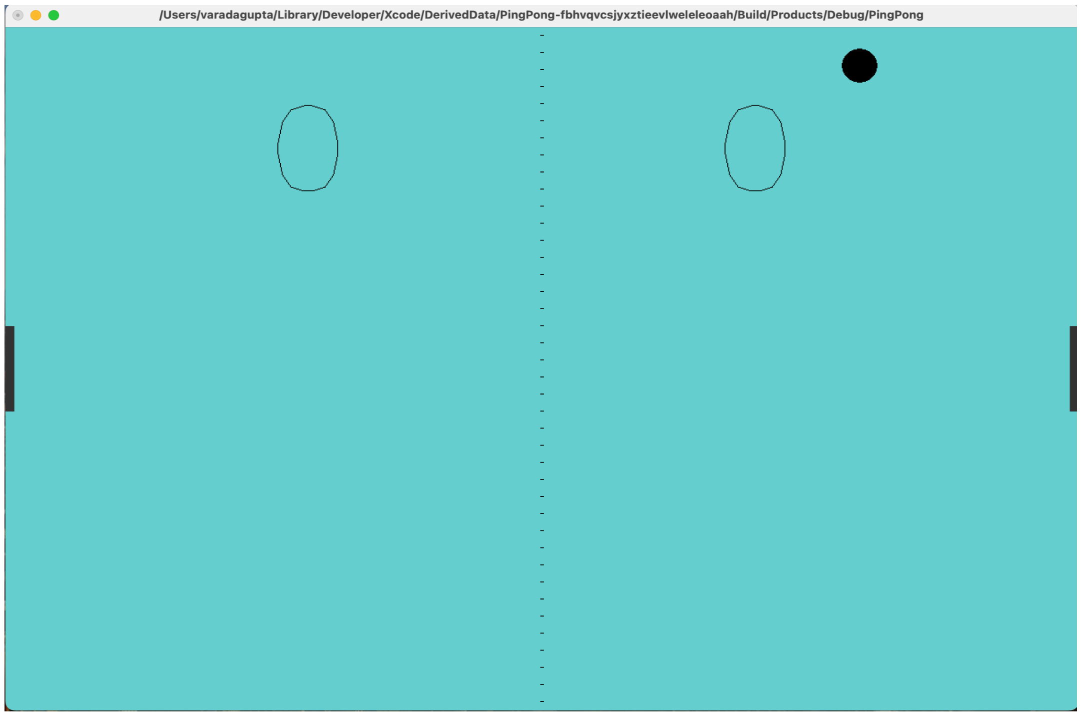
```
        case 'o':
            if (player2_paddle_y < paddle_boundary)
                player2_paddle_y += paddle_velocity;
            glutPostRedisplay();
            break;

        case 'l':
            if (player2_paddle_y > -paddle_boundary)
                player2_paddle_y -= paddle_velocity;
            glutPostRedisplay();
            break;

        case 27:
            exit(0);
            break;
        default:
            break;
    }
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(1200, 800);
    glutInitWindowPosition(10, 10);
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
    glutKeyboardFunc(keyboard);
glutMainLoop();
    return 0;
}
```

OUTPUT/SCREENSHOTS

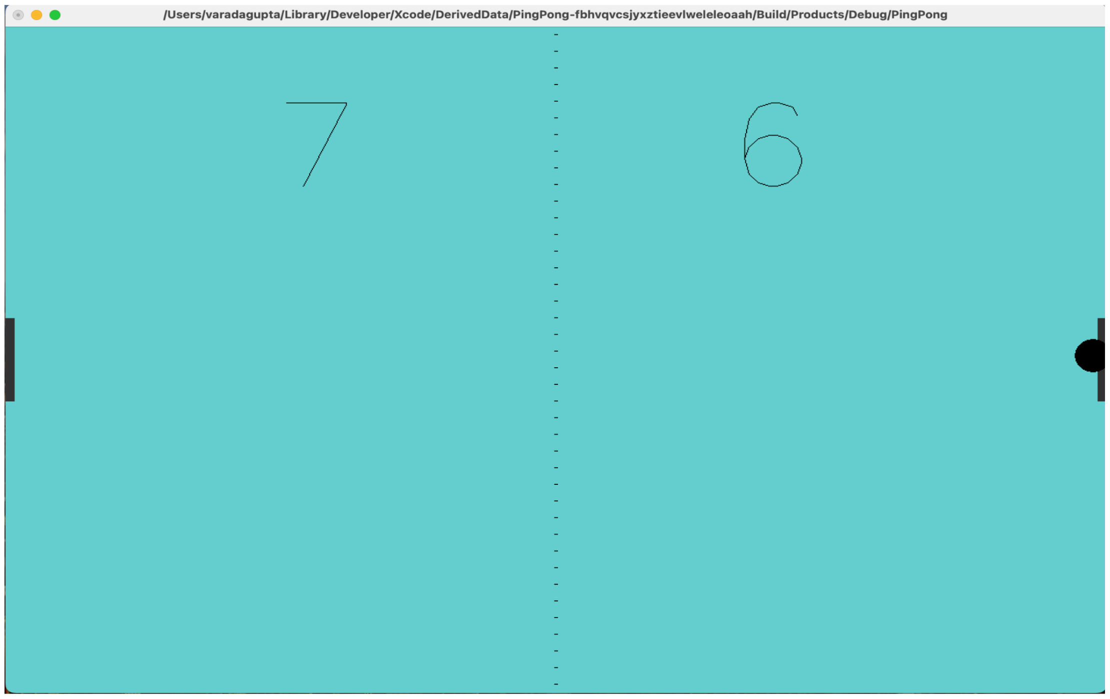
1. Start of game


/Users/varadagupta/Library/Developer/Xcode/DerivedData/PingPong-fbhvqvcsjyxztieevlweleleoaah/Build/Products/Debug/PingPong

2. Ball starts moving


/Users/varadagupta/Library/Developer/Xcode/DerivedData/PingPong-fbhvqvcsjyxztieevlweleleoaah/Build/Products/Debug/PingPong

3. Game Ends with Player 1 as Winner


/Users/varadagupta/Library/Developer/Xcode/DerivedData/PingPong-fbhvqvcsjyxztieevlweleleoaah/Build/Products/Debug/PingPong
7
6