`Computer Graphics (UCS505)

Project on

3D Cricket Pitch Map Modelling

Submitted By

Jasmine Das	102117016
Nandini Naithani	102297003

Group No. 12
B.E. Third Year – 3CS1

Submitted To:

Dr. Harpreet Singh

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 Project

This cricket pitch modelling project utilises OpenGL to render a 3D representation of a cricket field. Users can simulate ball trajectories for different deliveries, control viewpoints, and interact via a menu. It offers an immersive experience for understanding cricket gameplay, showcasing graphics rendering techniques and dynamic object simulation. The project encapsulates cricket dynamics within a visually engaging virtual environment.

Project Features:

Pitch Display: The program renders a 3D representation of a cricket pitch, including the playing area, creases, boundary lines, and other field markings.
Ball Trajectory Simulation: Users can simulate the trajectory of a cricket ball based on different types of deliveries such as yorker length, full length, good length, and short length.
Viewpoint Control: Users can switch between different viewpoints using the keyboard keys 'B', 'F', and 'S', which represent views from behind the stumps, front view (umpire's view), and side view, respectively.
Interactive Menu: The program features an interactive menu that allows users to select various options related to the length of the delivery and clear the simulation.

Key Components:

OpenGL Setup: The project initialises and configures OpenGL for rendering 3D graphics.
Display Functions: Functions are defined to render different components of the cricket pitch, including the pitch itself, sight-screen, ground, stumps, lengths, and the cricket ball.
Viewpoint Control: Functions handle user input for switching between different viewpoints.
Menu Creation: An interactive menu is created using GLUT (OpenGL Utility Toolkit) to provide users with options for selecting delivery lengths and clearing the simulation.
Ball Animation: The trajectory of the cricket ball is simulated using OpenGL transformations to animate its movement from the release point to different locations on the pitch based on the selected delivery type.

2. Computer Graphics concepts used

1. **Vertex Processing:**

   Transformation and Translation: Objects like the cricket pitch, players, and ball are dynamically translated based on user input, simulating their motion.

2. **Clipping and Primitive Assembly:**

   Visibility Determination: Ensures only visible objects are processed and rendered, conserving resources.
   Primitive Assembly: Constructs basic shapes (e.g., pitch, players) from vertices for rendering.

3. **Rasterization:**

   Drawing Game Elements: Converts object vertices into pixels on the screen.
   Pixel Filling: Applies colors and textures to shapes, enhancing visual quality.
   Scan Conversion: Renders smooth movement of objects like players and the ball.

4. **Matrix Stack Operations (OpenGL):**

   Projection Matrix: Defines camera properties, focusing on depth handling.

   Model-view Matrix: Positions objects and orients the camera within the scene.

   Pushing and Popping Matrices: Saves and restores matrix states for efficient rendering.

3. User Defined Functions:

Main Function:

- **int main(int argc, char** argv):** Initializes the GLUT library, sets up the main window, and event handlers. It configures the initial OpenGL context state and enters the event processing loop.

Display and Rendering Functions:

- **void myDisplay(void):** Renders the scene, clears the screen, sets up viewing transformations, and draws game objects.

Game Mechanics Functions:

- **void moveBall(float x, float y, float r):** Calculates new basketball position based on velocity, launch angle, and gravity.
- **void drawBall(float x, float y, float r):** Draws the basketball using OpenGL commands to create a filled circle.

Initialization and Settings:

- **void myInit(void):** Sets initial OpenGL settings such as background color and viewport size.

Input Handling:

- **void keyListener(int key, int, int):** Responds to special keyboard events like arrow keys to adjust shooting power and angle.
- **void restartGame(unsigned char key, int x, int y):** Handles ASCII keyboard inputs, allowing players to restart the game.

Score and Game State:

- **void drawScore():** Renders current score on screen.
- **void checkGameOver():** Monitors game state to determine if game has ended.

4. Code

```
#include <iostream>
#include <GL/gl.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include<iostream>
using namespace std;

void display();
void init();
void reshape(int, int);
void pitchDisplay();
void sightscreen();
void ground();
void stumps();
void lengths();
void ball();
void view(unsigned char key, int x, int y);
void createMenu();
void menu(int);


static int window ,returnmenu, returnsubmenuline1, returnsubmenuline2,
returnsubmenuline3, returnsubmenuline4, value = 0, viewstate=0;
double bowlxi=-0.99, bowlyi=1.0, bowlzi=-5.83, bowlxp=bowlxi, bowlyp=bowlyi,
bowlzp=bowlzi, bowlxf=bowlxp, bowlyf=bowlyp, bowlzf=bowlzp;



int main(int argc, char **argv) // arguments are used to initialize the glut.
{
   cout<<"YOU CAN SWITCH TO DIFFERENT VIEW USING KEYS 'B','F'
AND 'S':"<<endl;
   cout<<" B: VIEW FROM BEHIND THE STUMPS(WICKET
KEPPER'S VIEW)."<<endl;
   cout<<" F: FRONT VIEW(UMPIRES VIEW)."<<endl;
   cout<<" S: SIDE VIEW."<<endl;

   glutInit(&argc, argv);//used to initialize the glut library.
   glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);//
initializing display mode
```

```
    glutInitWindowPosition(0, 0);// sets window position
    glutInitWindowSize(1300,650);//sets window height and width


    window=glutCreateWindow("***PITCHMAP MODELLING***");//creating a
window and naming it.

    createMenu();

    init();
    glutDisplayFunc(display);// display callback.
    glutReshapeFunc(reshape);
    glutKeyboardFunc(view);

    glutMainLoop();//infinite loop to keep the window running unless closed.

    return 0;
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);// to clear the
frame buffer. Drawing is done only after clearing previous frame bufer.
    glLoadIdentity();// resets any transformations of all the current matrices you
are currently in.
    glPointSize(10.0);
    glLineWidth(3.0);

    /*SIDE VIEW.
    glTranslatef(15.28, 0.0, -15);
    glRotatef(90, 0, 1, 0);
    */

    /*keepers view from behinnd the stumps
    glTranslatef(0.0, 0.0, -32);
    glRotatef(180, -1, 0, 0);
    glRotatef(180, 0, 0, 1);
    */
    switch(viewstate)
    {
      case 0:
      glLoadIdentity();
      break;

      case 1:
      glTranslatef(15.28, 0.0, -15.28);
      glRotatef(90, 0, 1, 0);
      break;

      case -1:
      glTranslatef(0.0, 0.0, -32);
  //  glRotatef(180, -1, 0, 0); flips the pitch
  //  glRotatef(180, 0, 0, 1);turns it over.
```

```
    glRotatef(180, 0, 1, 0);
      break;
  }

      pitchDisplay();
      ground();
      sightscreen();
      lengths();
      stumps();
      ball();

      glLineWidth(7.0);
      glBegin(GL_LINES);
         glVertex3f(bowlxi, bowlyi, bowlzi);
         glVertex3f(bowlxp, bowlyp, bowlzp);

         glVertex3f(bowlxp, bowlyp, bowlzp);
         glVertex3f(bowlxf, bowlyf, bowlzf);
      glEnd();

   glutSwapBuffers();// to display frame buffer in double buffer mode.
}

void init()
{
   glClearColor(0.0, 0.5, 1.0, 1.0); //to change color of frame buffer in RGB.
   glEnable(GL_DEPTH_TEST);//to enable depth test ... to tell is something is infront or
behind in a 3d scene.
   glDepthFunc(GL_LEQUAL);//GL_LEQUAL (LESS OR EQUAL) compare ithe depth
of odjects .
}

void reshape(int w, int h)//width and height of new window.
{
   glViewport(0, 0, w, h);// viewport defines in which part of window opengl is gonna draw
in
   //to set 3d projection or 3d view.
   glMatrixMode( GL_PROJECTION);// to specify the vertices or change the projection.
   glLoadIdentity();
   gluPerspective(60, w/h, 2.0, 100.0);
   glMatrixMode(GL_MODELVIEW);
}

void pitchDisplay()
{
   glBegin(GL_POLYGON);//pitch coordinates
   glColor3f(1.0, 0.75, 0.25);

     glVertex3f(1.525, -2.0, -26.56);
     glVertex3f(-1.525, -2.0, -26.56);
     glVertex3f(-1.525, -2.0, -4.0);
     glVertex3f(1.525, -2.0, -4.0);

   glEnd();
```

```
glBegin(GL_LINE_STRIP);
glColor3f(1.0, 1.0, 1.0);

  glVertex3f(1.525, -2.0, -26.56);
  glVertex3f(-1.525, -2.0, -26.56);
  glVertex3f(-1.525, -2.0, -4.0);
  glVertex3f(1.525, -2.0, -4.0);
  glVertex3f(1.525, -2.0, -26.56);

glEnd();

glBegin(GL_LINES);
glColor3f(1.0, 1.0, 1.0);
    //POPPING CREASE.
    glVertex3f(3.0, -2.0, -6.44);
    glVertex3f(-3.0, -2.0, -6.44);

    glVertex3f(3.0, -2.0, -24.12);
    glVertex3f(-3.0, -2.0, -24.12);

   //return crease
    glVertex3f(-1.32, -2.0, -6.44);
    glVertex3f(-1.32, -2.0, -2.0);

    glVertex3f(1.32, -2.0, -6.44);
    glVertex3f(1.32, -2.0, -2.0);

    glVertex3f(-1.32, -2.0, -28.56);
    glVertex3f(-1.32, -2.0, -24.12);

    glVertex3f(1.32, -2.0, -28.56);
    glVertex3f(1.32, -2.0, -24.12);

   //bowling crease.
    glVertex3f(-1.32, -2.0, -5.22);
    glVertex3f(1.32, -2.0, -5.22);

    glVertex3f(-1.32, -2.0, -25.34);
    glVertex3f(1.32, -2.0, -25.34);

   //wide lines.
   glColor3f(0.0, 0.0, 1.0);
    glVertex3f(-0.99, -2.0, -5.22);
    glVertex3f(-0.99, -2.0, -6.44);

    glVertex3f(0.99, -2.0, -5.22);
    glVertex3f(0.99, -2.0, -6.44);

    glVertex3f(-0.99, -2.0, -25.34);
    glVertex3f(-0.99, -2.0, -24.12);

    glVertex3f(0.99, -2.0, -25.34);
    glVertex3f(0.99, -2.0, -24.12);
    glEnd();
}

void sightscreen()
{
    glBegin(GL_POLYGON);//sightscreen of batting end
    glColor3f(0.0, 0.0, 0.0);

      glVertex3f(10.0, 12.0, -75.0);
      glVertex3f(-10.0, 12.0, -75.0);
      glVertex3f(-10.0, -2.0, -75.0);
      glVertex3f(10.0, -2.0, -75.0);

    glEnd();
}

void ground()
{
    glBegin(GL_POLYGON);
    glColor3f(0.0, 0.8, 0.0);

        glVertex3f(35.0, -2.0, -90.0);
        glVertex3f(-35.0, -2.0, -90.0);
        glVertex3f(-70.0, -2.0, 0.0);
        glVertex3f(-35.0, -2.0, 80.0);
        glVertex3f(35.0, -2.0, 80.0);
        glVertex3f(70.0, -2.0, 0.0);

    glEnd();

    //boundary line.
    glBegin(GL_LINES);
    glColor3f(0.0, 0.0, 0.0);

        glVertex3f(-30.0, -2.0, -70.0);
        glVertex3f(-60.0, -2.0, 0.0);
        glVertex3f(-30.0, -2.0, 70.0);
        glVertex3f(30.0, -2.0, 70.0);
        glVertex3f(60.0, -2.0, 0.0);
        glVertex3f(30.0, -2.0, -70.0);
        glVertex3f(-30.0, -2.0, -70.0);

    glEnd();
}

void stumps()
{
    glLineWidth(4.0);
    glColor3f(0.15,0.15,0.15);
    glBegin(GL_LINES);
    //NON STRIKER
        //MIDDLE
        glVertex3f(0.0, -2, -5.22);
        glVertex3f(0.0, -1.299, -5.22);
        //left
        glVertex3f(-0.1143, -2, -5.22);
        glVertex3f(-0.1143, -1.299, -5.22);
        //right
        glVertex3f(0.1143, -2, -5.22);
        glVertex3f(0.1143, -1.299, -5.22);

      //STRIKER
        //MIDDLE
        glVertex3f(0.0, -2, -25.34);
        glVertex3f(0.0, -1.299, -25.34);
        //left
        glVertex3f(-0.1143, -2, -25.34);
        glVertex3f(-0.1143, -1.299, -25.34);
        //right
        glVertex3f(0.1143, -2, -25.34);
        glVertex3f(0.1143, -1.299, -25.34);

    glEnd();

}

void lengths()
{

    glBegin(GL_QUADS);

         //yorker length
    glColor4f(0.0, 0.0, 0.5, 1.0);
        glVertex3f(-1.525, -1.99, -25.34);
        glVertex3f(-1.525, -1.99, -23.34);
    glColor4f(0.5, 0.5, 0.5, 1.0);
        glVertex3f(1.525, -1.99, -23.34);
        glVertex3f(1.525, -1.99, -25.34);

        //full length
    glColor4f(1.0, 0.0, 1.0, 1.0);
        glVertex3f(1.525, -1.99, -23.34);
        glVertex3f(-1.525, -1.99, -23.34);
     glColor4f(0.5, 0.5, 0.5, 1.0);
        glVertex3f(-1.525, -1.99, -19.34);
        glVertex3f(1.525, -1.99, -19.34);

        //good length.
    glColor4f(0.0, 1.0, 0.0, 1.0);
        glVertex3f(1.525, -1.99, -19.34);
        glVertex3f(-1.525, -1.99, -19.34);
    glColor4f(0.5, 0.5, 0.5, 1.0);
        glVertex3f(-1.525, -1.99, -17.34);
        glVertex3f(1.525, -1.99, -17.34);

        //   short length.
    glColor4f(1.0, 0.0, 0.0, 1.0);
```

```
       glVertex3f(1.525, -1.99, -17.34);
       glVertex3f(-1.525, -1.99, -17.34);
    glColor4f(1.0, 0.75, 0.25, 1.0);
       glVertex3f(-1.525, -2.0, -4.0);
       glVertex3f(1.525, -2.0, -4.0);

     glEnd();

        //stump to stump line
    glBegin(GL_POLYGON);

    glColor4f(0.85, 0.0,0.5, 1.0);
       glVertex3f(0.15, -1.98, -24.12);
       glVertex3f(-0.15, -1.98, -24.12);
    glColor4f(1.0, 0.75, 0.25, 1.0);
       glVertex3f(-0.15, -1.98, -5.22);
       glVertex3f(0.15, -1.98, -5.22);

    glEnd();
}

void view(unsigned char key, int x, int y)
{
    if(key=='s')
    {
       viewstate=1;
       glutPostRedisplay();
    }

    if (key=='b')
    {
      viewstate=-1;
      glutPostRedisplay();
    }

    if(key=='f')
    {
       viewstate=0;
       glutPostRedisplay();
    }
}

void createMenu()
{
   //YORKER LENGTH
    returnsubmenuline1 = glutCreateMenu(menu);
    glutAddMenuEntry("MIDDLE STUMP LINE", 11);
    glutAddMenuEntry("4TH STUMP LINE", 12);
    glutAddMenuEntry("6TH STUMP LINE", 13);
    glutAddMenuEntry("WIDE LINE", 14);
    //FULL LENGTH
    returnsubmenuline2 = glutCreateMenu(menu);
    glutAddMenuEntry("MIDDLE STUMP LINE", 21);
    glutAddMenuEntry("4TH STUMP LINE", 22);
    glutAddMenuEntry("6TH STUMP LINE", 23);
    glutAddMenuEntry("WIDE LINE", 24);

    //GOODLENGTH
    returnsubmenuline3 = glutCreateMenu(menu);
    glutAddMenuEntry("MIDDLE STUMP LINE", 31);
    glutAddMenuEntry("4TH STUMP LINE", 32);
    glutAddMenuEntry("6TH STUMP LINE", 33);
    glutAddMenuEntry("WIDE LINE", 34);

    // SHORT LENGTH
    returnsubmenuline4 = glutCreateMenu(menu);
    glutAddMenuEntry("MIDDLE STUMP LINE", 41);
    glutAddMenuEntry("4TH STUMP LINE", 42);
    glutAddMenuEntry("6TH STUMP LINE", 43);
    glutAddMenuEntry("WIDE LINE", 44);

    returnmenu = glutCreateMenu(menu); //function to call menu function and return
    value glutAddMenuEntry("Clear", 1);
    glutAddSubMenu("YORKER LENGTH", returnsubmenuline1);
    glutAddSubMenu("FULL LENGTH", returnsubmenuline2);
    glutAddSubMenu("GOOD LENGTH", returnsubmenuline3);
    glutAddSubMenu("SHORT LENGTH", returnsubmenuline4);
    glutAddMenuEntry("Quit", 0);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

void menu(int num)
{
    if(num == 0)
        {
        glutDestroyWindow(window);
    exit(0);
        }
    else
    {
    value = num;
    }
    glutPostRedisplay();

    bowlxp=bowlxi;
    bowlyp=bowlyi;
    bowlzp=bowlzi;
    bowlxf=bowlxp;
    bowlyf=bowlyp;
    bowlzf=bowlzp;
}

void ball()
{
    //stationary ball at release point
            glTranslatef(-0.99, 1.0, -5.83);
            glColor3f(0.9, 0.0, 0.0);
            glutSolidSphere(0.072, 32, 16);
        glTranslatef(0.99, -1.0, 5.83);

if (value == 1)
    {
        return;
    }

if (value==11)
{
    glPushMatrix();
    if(bowlxf<0.0 && bowlyf>-1.97 && bowlzf>-24.34)
    {
        //          moving ball animation
        glTranslatef(bowlxf, bowlyf, bowlzf);
        glColor3f(1.0, 0.75, 1.0);
        glutSolidSphere(0.072, 32, 16);
        glLoadIdentity();

        bowlxf=bowlxf+0.001;
        bowlyf=bowlyf-0.003;
        bowlzf=bowlzf-0.01869;
     }

     else
     {
         bowlxp=0.0;
         bowlyp=-1.97;
         bowlzp=-24.34;

        glTranslatef(bowlxf, bowlyf, bowlzf);
        glColor3f(1.0, 0.75, 1.0);
        glutSolidSphere(0.072, 32, 16);
        glLoadIdentity();

       if(bowlzf>-26.00)
       {
           bowlxf=0.0;
           bowlyf=bowlyf+0.003;
           bowlzf=bowlzf-0.028;
       }
     }
     glPopMatrix();
}

 if (value==12)
{
    glPushMatrix();
    if(bowlxf<-0.2286 && bowlyf>-1.97 && bowlzf>-24.34)
    {
        //          moving ball animation
        glTranslatef(bowlxf, bowlyf, bowlzf);
        glColor3f(1.0, 0.75, 1.0);
        glutSolidSphere(0.072, 32, 16);
        glLoadIdentity();
```

```
      bowlxf=bowlxf+0.001;
      bowlyf=bowlyf-0.00390;
      bowlzf=bowlzf-0.02431;
    }

    else
    {
       bowlxp=-0.2286;
       bowlyp=-1.97;
       bowlzp=-24.34;

      glTranslatef(bowlxf, bowlyf, bowlzf);
      glColor3f(1.0, 0.75, 1.0);
      glutSolidSphere(0.072, 32, 16);
      glLoadIdentity();

     if(bowlzf>-26.00)
     {
         bowlxf=-0.2286;
         bowlyf=bowlyf+0.003;
         bowlzf=bowlzf-0.028;
     }
    }
    glPopMatrix();
}

 if (value==13)
{
   glPushMatrix();
   if(bowlxf<-0.4572 && bowlyf>-1.97 && bowlzf>-24.34)
   {
     //          moving ball animation
      glTranslatef(bowlxf, bowlyf, bowlzf);
      glColor3f(1.0, 0.75, 1.0);
      glutSolidSphere(0.072, 32, 16);
      glLoadIdentity();

      bowlxf=bowlxf+0.001;
      bowlyf=bowlyf-0.00557;
      bowlzf=bowlzf-0.03474;
    }

    else
    {
       bowlxp=-0.4572;
       bowlyp=-1.97;
       bowlzp=-24.34;

      glTranslatef(bowlxf, bowlyf, bowlzf);
      glColor3f(1.0, 0.75, 1.0);
      glutSolidSphere(0.072, 32, 16);
      glLoadIdentity();
```

```
        if(bowlzf>-26.00)
        {
            bowlxf=-0.4572;
            bowlyf=bowlyf+0.003;
            bowlzf=bowlzf-0.028;
        }
      }
      glPopMatrix();
  }

  if (value==14)
  {
     glPushMatrix();
     if(bowlxf<-0.97 && bowlyf>-1.97 && bowlzf>-24.34)
     {
        //          moving ball animation
        glTranslatef(bowlxf, bowlyf, bowlzf);
        glColor3f(1.0, 0.75, 1.0);
        glutSolidSphere(0.072, 32, 16);
        glLoadIdentity();

        bowlxf=bowlxf+0.000015;
        bowlyf=bowlyf-0.0022275;
        bowlzf=bowlzf-0.0138825;
      }

      else
      {
         bowlxp=-0.97;
         bowlyp=-1.97;
         bowlzp=-24.34;

        glTranslatef(bowlxf, bowlyf, bowlzf);
        glColor3f(1.0, 0.75, 1.0);
        glutSolidSphere(0.072, 32, 16);
        glLoadIdentity();

        if(bowlzf>-26.00)
        {
            bowlxf=-0.97;
            bowlyf=bowlyf+0.003;
            bowlzf=bowlzf-0.028;
        }
      }
      glPopMatrix();

  }

   if (value==21)
  {
     glPushMatrix();
        if(bowlxf<0.0 && bowlyf>-1.97 && bowlzf>-21.34)
        {
            // moving ball animation
            glTranslatef(bowlxf, bowlyf, bowlzf);
            glColor3f(1.0, 0.75, 1.0);
            glutSolidSphere(0.072, 32, 16);
            glLoadIdentity();

            bowlxf=bowlxf+0.001;
            bowlyf=bowlyf-0.003;
            bowlzf=bowlzf-0.01567;
          }
          else
          {
            bowlxp=0.0;
            bowlyp=-1.97;
            bowlzp=-21.34;

            glTranslatef(bowlxf, bowlyf, bowlzf);
            glColor3f(1.0, 0.75, 1.0);
            glutSolidSphere(0.072, 32, 16);
            glLoadIdentity();

            if(bowlzf>-26.00)
            {
              bowlxf=0.0;
              bowlyf=bowlyf+0.003;
              bowlzf=bowlzf-0.03938;
            }
          }
          glPopMatrix();
        }

         if (value==22)
      {
        glPushMatrix();
        if(bowlxf<-0.2286 && bowlyf>-1.97 && bowlzf>-21.34)
        {
          //          moving ball animation
          glTranslatef(bowlxf, bowlyf, bowlzf);
          glColor3f(1.0, 0.75, 1.0);
          glutSolidSphere(0.072, 32, 16);
          glLoadIdentity();

          bowlxf=bowlxf+0.001;
          bowlyf=bowlyf-0.0039;
          bowlzf=bowlzf-0.02037;
        }

        else
        {
          bowlxp=-0.2286;
          bowlyp=-1.97;
          bowlzp=-21.34;

          glTranslatef(bowlxf, bowlyf, bowlzf);
          glColor3f(1.0, 0.75, 1.0);
        glutSolidSphere(0.072, 32, 16);
        glLoadIdentity();

       if(bowlzf>-26.00)
       {
           bowlxf=-0.2286;
           bowlyf=bowlyf+0.003;
           bowlzf=bowlzf-0.03938;
       }
     }
     glPopMatrix();
}

 if (value==23)
{
    glPushMatrix();
    if(bowlxf<-0.4572 && bowlyf>-1.97 && bowlzf>-21.34)
    {
       //          moving ball animation
       glTranslatef(bowlxf, bowlyf, bowlzf);
       glColor3f(1.0, 0.75, 1.0);
       glutSolidSphere(0.072, 32, 16);
       glLoadIdentity();

       bowlxf=bowlxf+0.001;
       bowlyf=bowlyf-0.00557;
       bowlzf=bowlzf-0.02911;
     }

     else
     {
        bowlxp=-0.4572;
        bowlyp=-1.97;
        bowlzp=-21.34;

       glTranslatef(bowlxf, bowlyf, bowlzf);
       glColor3f(1.0, 0.75, 1.0);
       glutSolidSphere(0.072, 32, 16);
       glLoadIdentity();

      if(bowlzf>-26.00)
      {
          bowlxf=-0.4572;
          bowlyf=bowlyf+0.003;
          bowlzf=bowlzf-0.03938;
      }
     }
     glPopMatrix();
}

if (value==24)
{
    glPushMatrix();
    if(bowlxf<-0.97 && bowlyf>-1.97 && bowlzf>-21.34)
    {
      //          moving ball animation
      glTranslatef(bowlxf, bowlyf, bowlzf);
      glColor3f(1.0, 0.75, 1.0);
      glutSolidSphere(0.072, 32, 16);
      glLoadIdentity();

      bowlxf=bowlxf+0.000015;
      bowlyf=bowlyf-0.002227;
      bowlzf=bowlzf-0.011632;
     }

     else
     {
       bowlxp=-0.97;
       bowlyp=-1.97;
       bowlzp=-21.34;

      glTranslatef(bowlxf, bowlyf, bowlzf);
      glColor3f(1.0, 0.75, 1.0);
      glutSolidSphere(0.072, 32, 16);
      glLoadIdentity();

     if(bowlzf>-26.00)
     {
         bowlxf=-0.97;
         bowlyf=bowlyf+0.003;
         bowlzf=bowlzf-0.03938;
     }
     }
     glPopMatrix();
}

 if (value==31)
{
   glPushMatrix();
   if(bowlxf<0.0 && bowlyf>-1.97 && bowlzf>-18.34)
   {
      //          moving ball animation
      glTranslatef(bowlxf, bowlyf, bowlzf);
      glColor3f(1.0, 0.75, 1.0);
      glutSolidSphere(0.072, 32, 16);
      glLoadIdentity();

      bowlxf=bowlxf+0.001;
      bowlyf=bowlyf-0.003;
      bowlzf=bowlzf-0.01263;
   }

   else
   {
      bowlxp=0.0;
      bowlyp=-1.97;
      bowlzp=-18.34;
      glTranslatef(bowlxf, bowlyf, bowlzf);
      glColor3f(1.0, 0.75, 1.0);
      glutSolidSphere(0.072, 32, 16);
      glLoadIdentity();

      if(bowlzf>-26.00)
      {
         bowlxf=0.0;
         bowlyf=bowlyf+0.003;
         bowlzf=bowlzf-0.03232;
      }
    }
         glPopMatrix();
    }

    if (value==32)
    {
      glPushMatrix();
      if(bowlxf<-0.2286 && bowlyf>-1.97 && bowlzf>-18.34)
      {
         //          moving ball animation
         glTranslatef(bowlxf, bowlyf, bowlzf);
         glColor3f(1.0, 0.75, 1.0);
         glutSolidSphere(0.072, 32, 16);
         glLoadIdentity();

         bowlxf=bowlxf+0.001;
         bowlyf=bowlyf-0.0039;
         bowlzf=bowlzf-0.01643;
      }

      else
      {
         bowlxp=-0.2286;
         bowlyp=-1.97;
         bowlzp=-18.34;

         glTranslatef(bowlxf, bowlyf, bowlzf);
         glColor3f(1.0, 0.75, 1.0);
         glutSolidSphere(0.072, 32, 16);
         glLoadIdentity();


         if(bowlzf>-26.00)
         {
            bowlxf=-0.2286;
            bowlyf=bowlyf+0.003;
            bowlzf=bowlzf-0.03232;
         }
      }
    glPopMatrix();
}
```

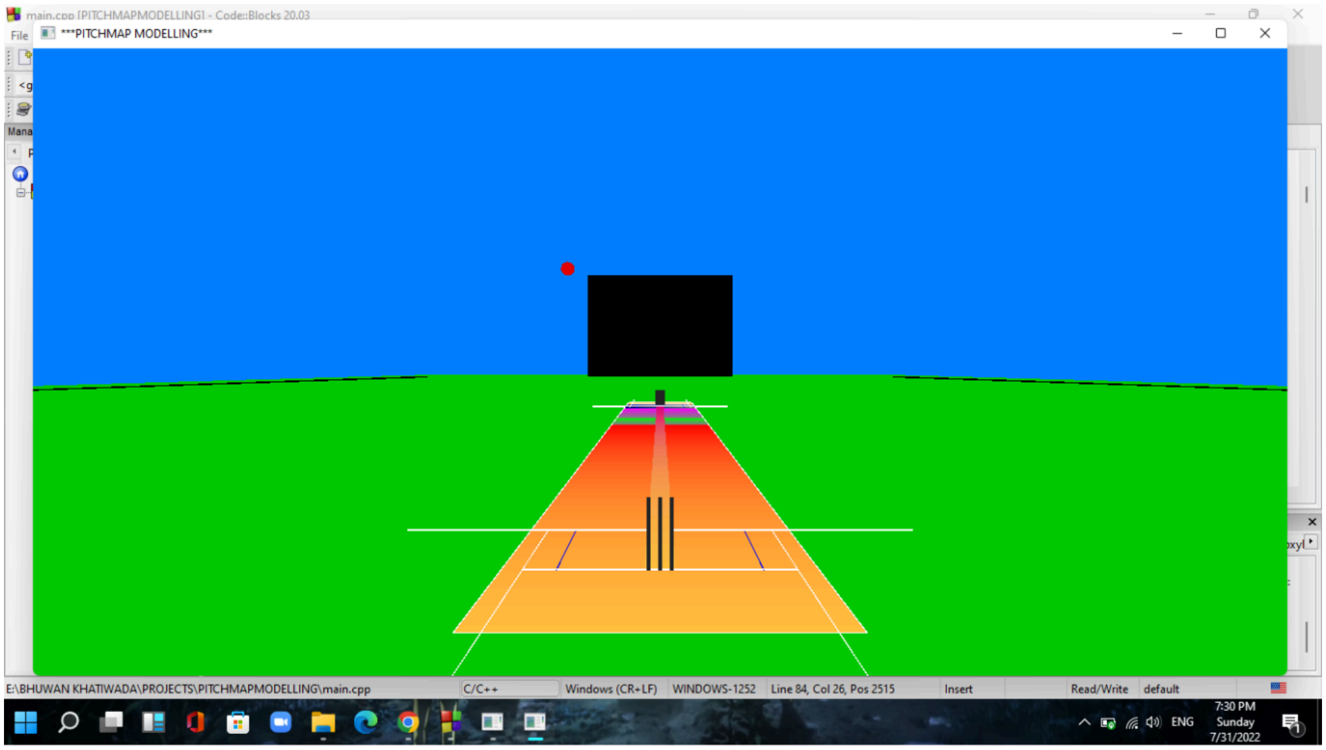
```
 if (value==33)
{
   glPushMatrix();
   if(bowlxf<-0.4572 && bowlyf>-1.97 && bowlzf>-18.34)
   {
     //          moving ball animation
     glTranslatef(bowlxf, bowlyf, bowlzf);
     glColor3f(1.0, 0.75, 1.0);
     glutSolidSphere(0.072, 32, 16);
     glLoadIdentity();

     bowlxf=bowlxf+0.001;
     bowlyf=bowlyf-0.00557;
     bowlzf=bowlzf-0.02347;
    }

    else
    {
      bowlxp=-0.4572;
      bowlyp=-1.97;
      bowlzp=-18.34;

     glTranslatef(bowlxf, bowlyf, bowlzf);
     glColor3f(1.0, 0.75, 1.0);
     glutSolidSphere(0.072, 32, 16);
     glLoadIdentity();

     if(bowlzf>-26.00)
     {
        bowlxf=-0.4572;
        bowlyf=bowlyf+0.003;
        bowlzf=bowlzf-0.03232;
     }
    }
    glPopMatrix();
}

if (value==34)
{
   glPushMatrix();
   if(bowlxf<-0.97 && bowlyf>-1.97 && bowlzf>-18.34)
   {
     //          moving ball animation
     glTranslatef(bowlxf, bowlyf, bowlzf);
     glColor3f(1.0, 0.75, 1.0);
     glutSolidSphere(0.072, 32, 16);
     glLoadIdentity();

     bowlxf=bowlxf+0.000015;
     bowlyf=bowlyf-0.002227;
     bowlzf=bowlzf-0.0093825;
    }

    else
```

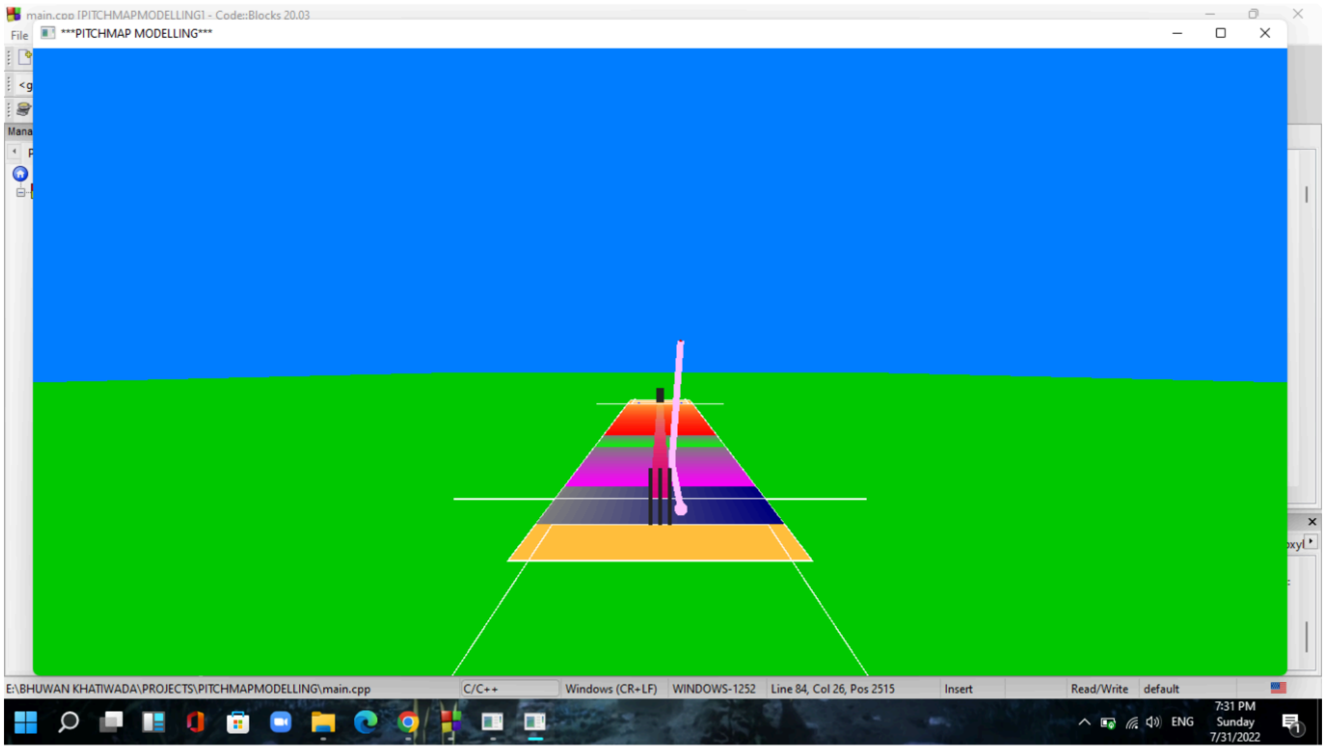
```
      {
        bowlxp=-0.97;
        bowlyp=-1.97;
        bowlzp=-18.34;

        glTranslatef(bowlxf, bowlyf, bowlzf);
        glColor3f(1.0, 0.75, 1.0);
        glutSolidSphere(0.072, 32, 16);
        glLoadIdentity();

       if(bowlzf>-26.00)
       {
          bowlxf=-0.97;
          bowlyf=bowlyf+0.003;
          bowlzf=bowlzf-0.03232;
       }
      }
     glPopMatrix();
   }

   if (value==41)
   {
     glPushMatrix();
     if(bowlxf<0.0 && bowlyf>-1.97 && bowlzf>-15.34)
     {
       //           moving ball animation
       glTranslatef(bowlxf, bowlyf, bowlzf);
       glColor3f(1.0, 0.75, 1.0);
       glutSolidSphere(0.072, 32, 16);
       glLoadIdentity();

       bowlxf=bowlxf+0.001;
       bowlyf=bowlyf-0.003;
       bowlzf=bowlzf-0.009606;
     }

     else
     {
       bowlxp=0.0;
       bowlyp=-1.97;
       bowlzp=-15.34;

       glTranslatef(bowlxf, bowlyf, bowlzf);
       glColor3f(1.0, 0.75, 1.0);
       glutSolidSphere(0.072, 32, 16);
       glLoadIdentity();

       if(bowlzf>-26.00)
       {
          bowlxf=0.0;
          bowlyf=bowlyf+0.003;
          bowlzf=bowlzf-0.016233;
       }
     }
```

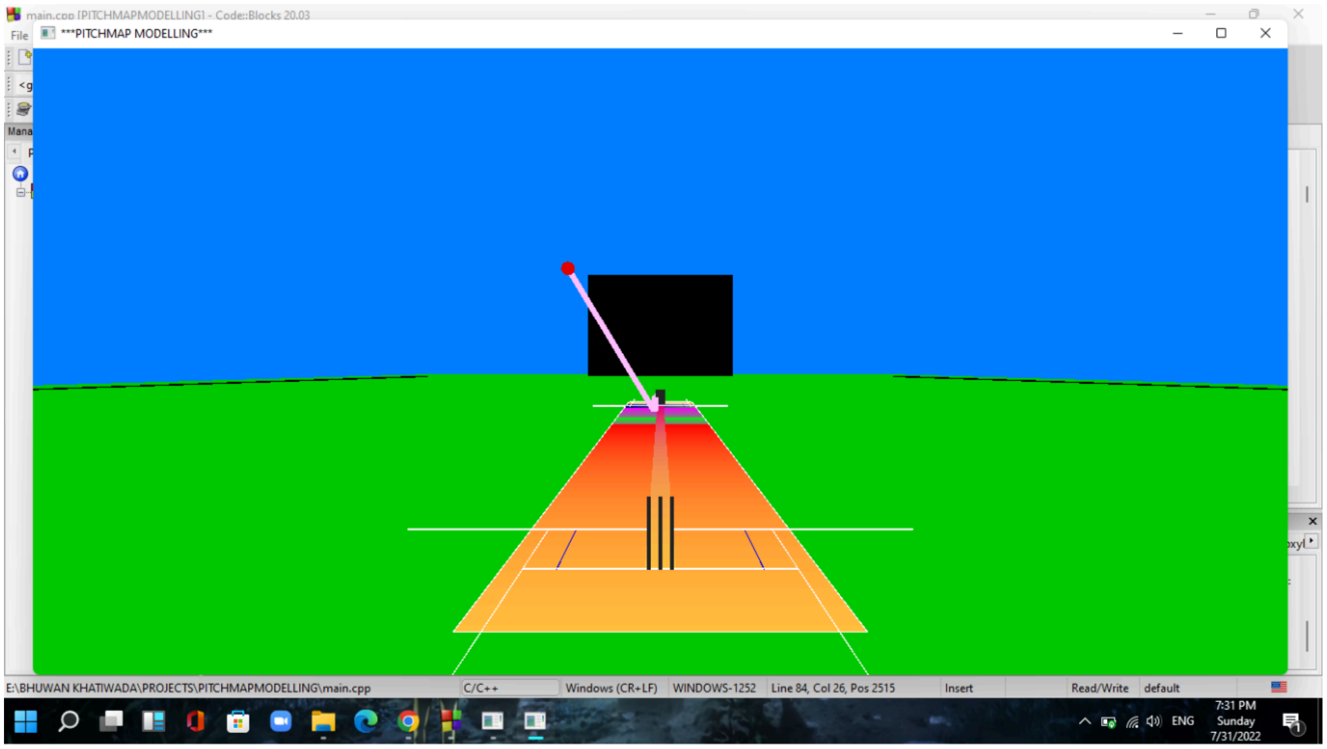
```
            glPopMatrix();
        }

    if (value==42)
    {
        glPushMatrix();
        if(bowlxf<-0.2286 && bowlyf>-1.97 && bowlzf>-15.34)
        {
          //          moving ball animation
          glTranslatef(bowlxf, bowlyf, bowlzf);
          glColor3f(1.0, 0.75, 1.0);
          glutSolidSphere(0.072, 32, 16);
          glLoadIdentity();

          bowlxf=bowlxf+0.001;
          bowlyf=bowlyf-0.0039;
          bowlzf=bowlzf-0.01249;
        }

        else
        {
           bowlxp=-0.2286;
           bowlyp=-1.97;
           bowlzp=-15.34;

          glTranslatef(bowlxf, bowlyf, bowlzf);
          glColor3f(1.0, 0.75, 1.0);
          glutSolidSphere(0.072, 32, 16);
          glLoadIdentity();

         if(bowlzf>-26.00)
         {
             bowlxf=-0.2286;
             bowlyf=bowlyf+0.003;
             bowlzf=bowlzf-0.016233;
         }
        }
        glPopMatrix();
    }

     if (value==43)
    {
        glPushMatrix();
        if(bowlxf<-0.4572 && bowlyf>-1.97 && bowlzf>-15.34)
        {
          //          moving ball animation
          glTranslatef(bowlxf, bowlyf, bowlzf);
          glColor3f(1.0, 0.75, 1.0);
          glutSolidSphere(0.072, 32, 16);
          glLoadIdentity();

          bowlxf=bowlxf+0.001;
          bowlyf=bowlyf-0.00557;
          bowlzf=bowlzf-0.01785;
    }

    else
    {
       bowlxp=-0.4572;
       bowlyp=-1.97;
       bowlzp=-15.34;

      glTranslatef(bowlxf, bowlyf, bowlzf);
      glColor3f(1.0, 0.75, 1.0);
      glutSolidSphere(0.072, 32, 16);
      glLoadIdentity();

     if(bowlzf>-26.00)
     {
         bowlxf=-0.4572;
         bowlyf=bowlyf+0.003;
         bowlzf=bowlzf-0.016233;
     }
    }
    glPopMatrix();
}

if (value==44)
{
   glPushMatrix();
   if(bowlxf<-0.97 && bowlyf>-1.97 && bowlzf>-15.34)
   {
      //          moving ball animation
      glTranslatef(bowlxf, bowlyf, bowlzf);
      glColor3f(1.0, 0.75, 1.0);
      glutSolidSphere(0.072, 32, 16);
      glLoadIdentity();

      bowlxf=bowlxf+0.000015;
      bowlyf=bowlyf-0.002227;
      bowlzf=bowlzf-0.007132;
   }

    else
    {
       bowlxp=-0.97;
       bowlyp=-1.97;
       bowlzp=-15.34;

      glTranslatef(bowlxf, bowlyf, bowlzf);
      glColor3f(1.0, 0.75, 1.0);
      glutSolidSphere(0.072, 32, 16);
      glLoadIdentity();

     if(bowlzf>-26.00)
     {
         bowlxf=-0.97;
         bowlyf=bowlyf+0.00303;
```

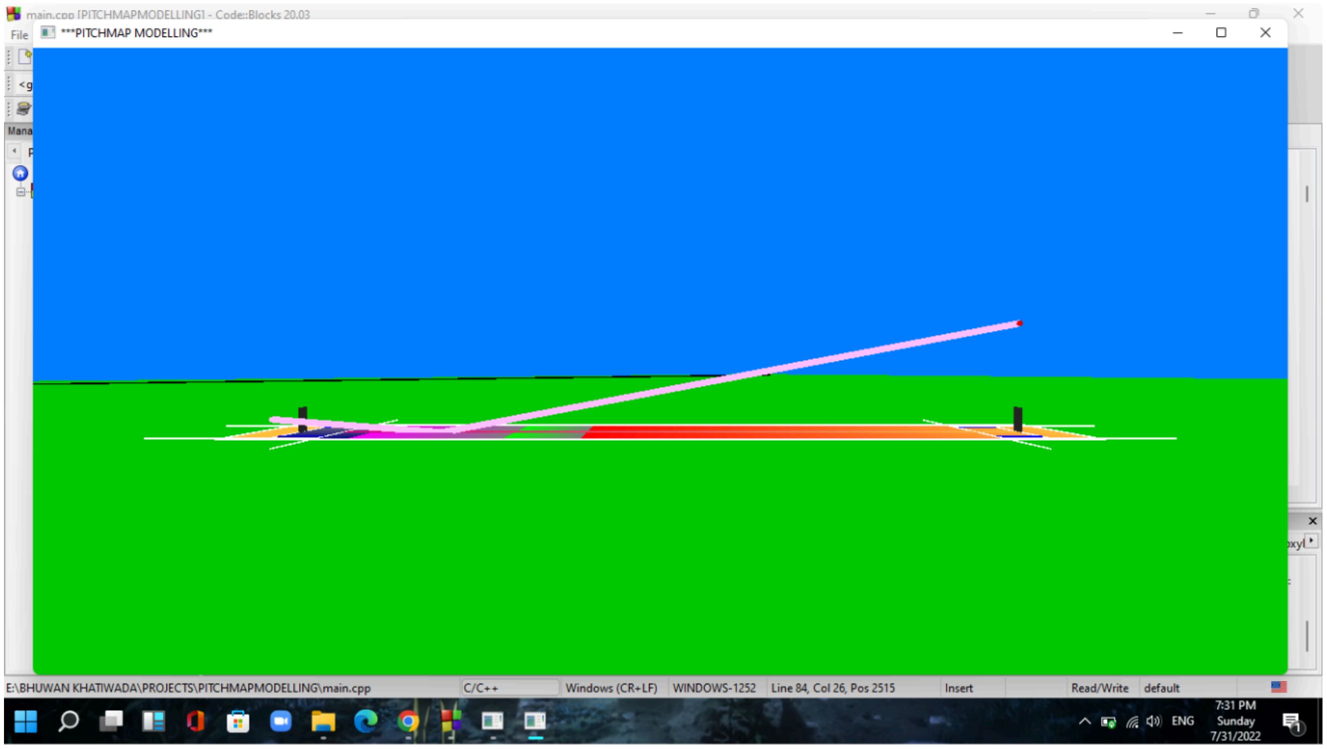
```
              bowlzf=bowlzf-0.01639;
           }
         }
         glPopMatrix();
    }
    glutPostRedisplay();
}
```

5. Output/ Screen shots


main.cpp [PITCHMAPMODELLING] - Code::Blocks 20.03
PITCHMAP MODELLING
E:\BHUWAN KHATIWADA\PROJECTS\PITCHMAPMODELLING\main.cpp
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 84, Col 26, Pos 2515
Insert
Read/Write
default
7:30 PM
Sunday
7/31/2022


main.cpp [PITCHMAPMODELLING] - Code::Blocks 20.03
PITCHMAP MODELLING
E:\BHUWAN KHATIWADA\PROJECTS\PITCHMAPMODELLING\main.cpp
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 84, Col 26, Pos 2515
Insert
Read/Write
default
7:31 PM
Sunday
7/31/2022


main.cpp [PITCHMAPMODELLING] - Code::Blocks 20.03
PITCHMAP MODELLING
E:\BHUWAN KHATIWADA\PROJECTS\PITCHMAPMODELLING\main.cpp
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 84, Col 26, Pos 2515
Insert
Read/Write
default
7:31 PM
Sunday
7/31/2022


main.cpp [PITCHMAPMODELLING] - Code::Blocks 20.03
PITCHMAP MODELLING
E:\BHUWAN KHATIWADA\PROJECTS\PITCHMAPMODELLING\main.cpp
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 84, Col 26, Pos 2515
Insert
Read/Write
default
7:31 PM
Sunday
7/31/2022