SECV2213 FUNDAMENTAL OF COMPUTER GRAPHICS (SECTION 01)

FINAL PROJECT

SUBMISSION DATE: 24TH JUNE 2025

LECTURER NAME : TS. DR. GOH EG SU

GOOGLE DRIVE LINK:
https://drive.google.com/drive/folders/1yepH7VbWIefWqRiVL2Sm1ZM2yJO6KKFb?usp=drive_link

GROUP NAME: TrioGraphiX	
GROUP MEMBERS	**MATRIC NUMBER**
NUR AINA BALQIS BINTI MOHAMAD ZAPARIN	A23CS0151
KRISTINE ELLE BENJAMIN	A23CS0095
NUR AMIERA ZULAIKHA BINTI HARDI	A23CS0153

TABLE OF CONTENTS

1.0 INTRODUCTION.......... 3

2.0 OVERALL CONCEPT & DESIGN.......... 3

2.1 Model Structure.......... 3

2.2 Modularity.......... 5

2.3 Interaction Design.......... 8

2.4 Scene Setup.......... 10

3.0 IMPLEMENTATION & ACHIEVEMENTS.......... 11

4.0 DISCUSSION ON EACH TOPIC.......... 15

4.1 Modeling Technique.......... 15

4.2 Hierarchical Structure.......... 16

4.3 Lighting and Shading.......... 17

4.4 Animation and State Control.......... 17

4.5 Texture Mapping.......... 17

4.6 Audio Integration.......... 18

4.7 Extra Features.......... 19

5.0 SAMPLE OUTPUT.......... 20

6.0 CONCLUSION.......... 23

7.0 REFERENCE.......... 24

1.0 INTRODUCTION

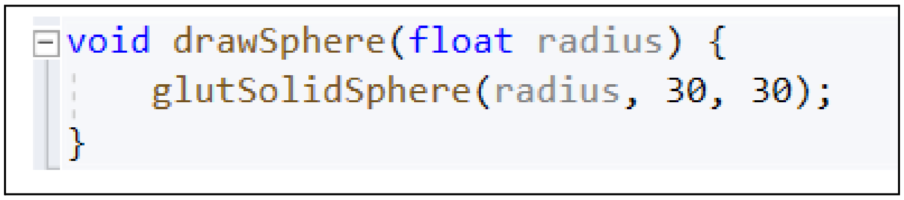
Our final project is about creating a 3D animated model, which is JigglyPuff from Pokemon using OpenGL. The model incorporates a range of interactive features, including singing, blinking, flying, and others. This report presents a comprehensive explanation of the implementation process, highlighting the use of tools such as OpenGL, a BMP image loader, and the Windows Sound API.

2.0 OVERALL CONCEPT & DESIGN

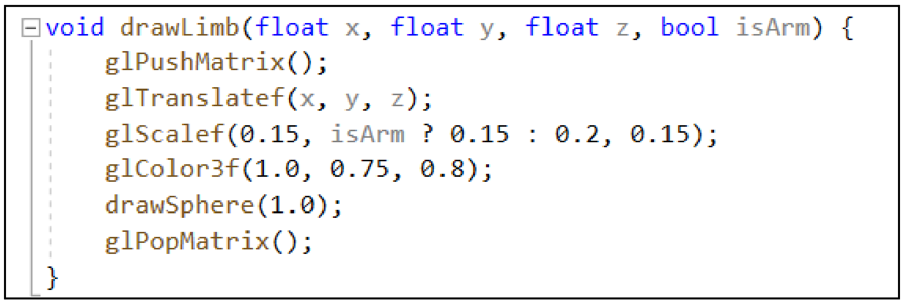
2.1 Model Structure

The base of our model is primarily constructed using spheres that are scaled, translated, and rotated to form different body parts. It utilizes *glutSolidSphere()* as the base geometry for most components. Each part, such as the body, ears, hair curls and limbs, is formed by applying appropriate transformations to spheres.

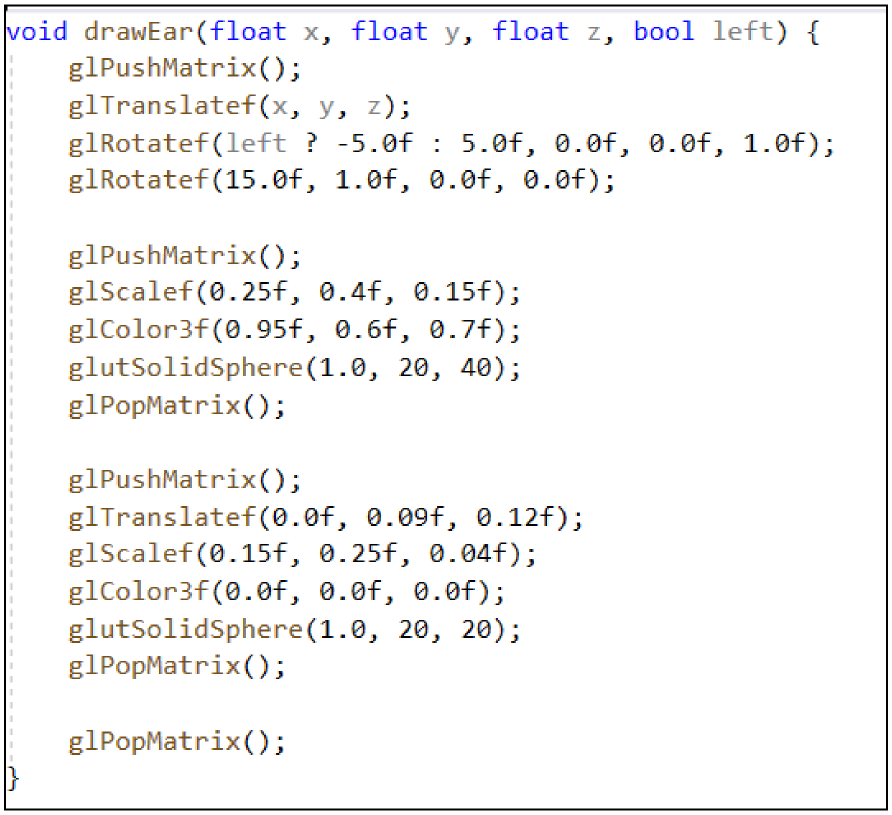
Code reference:



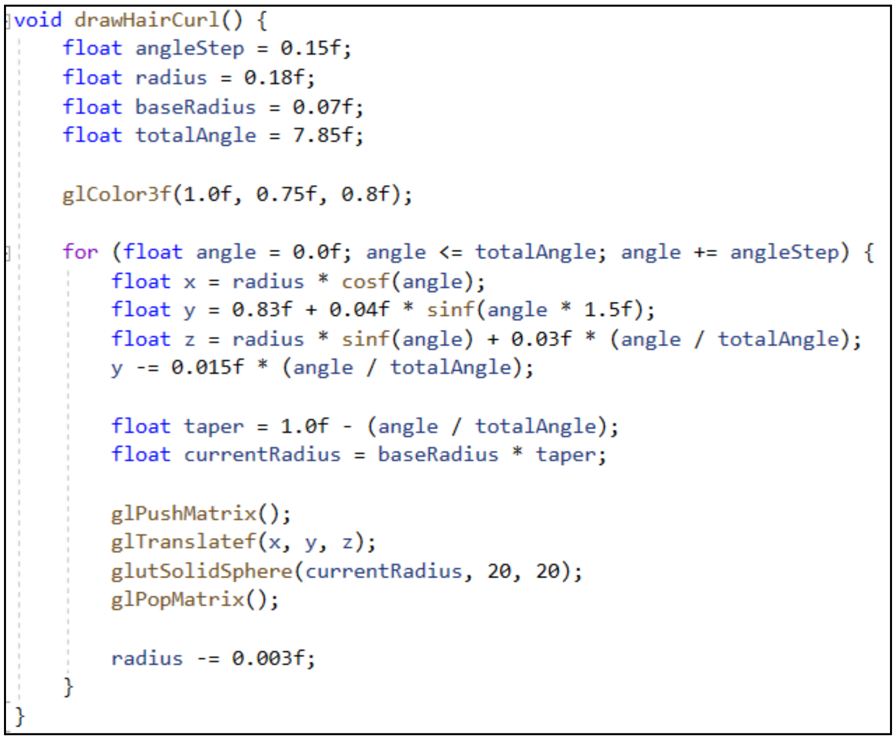
```
void drawSphere(float radius) {
    glutSolidSphere(radius, 30, 30);
}
```



```
void drawLimb(float x, float y, float z, bool isArm) {
    glPushMatrix();
    glTranslatef(x, y, z);
    glScalef(0.15, isArm ? 0.15 : 0.2, 0.15);
    glColor3f(1.0, 0.75, 0.8);
    drawSphere(1.0);
    glPopMatrix();
}
```



```
void drawEar(float x, float y, float z, bool left) {
    glPushMatrix();
    glTranslatef(x, y, z);
    glRotatef(left ? -5.0f : 5.0f, 0.0f, 0.0f, 1.0f);
    glRotatef(15.0f, 1.0f, 0.0f, 0.0f);

    glPushMatrix();
    glScalef(0.25f, 0.4f, 0.15f);
    glColor3f(0.95f, 0.6f, 0.7f);
    glutSolidSphere(1.0, 20, 40);
    glPopMatrix();

    glPushMatrix();
    glTranslatef(0.0f, 0.09f, 0.12f);
    glScalef(0.15f, 0.25f, 0.04f);
    glColor3f(0.0f, 0.0f, 0.0f);
    glutSolidSphere(1.0, 20, 20);
    glPopMatrix();

    glPopMatrix();
}
```

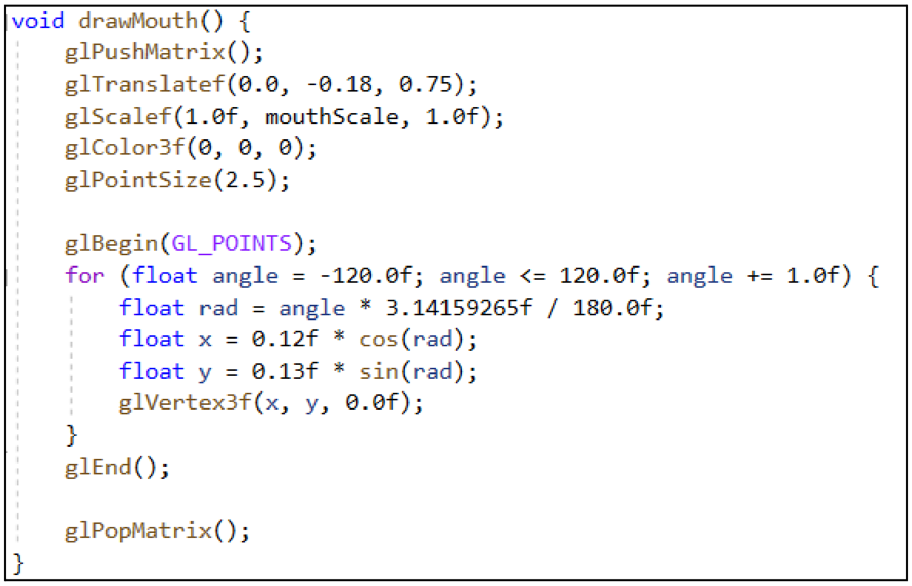


```
void drawHairCurl() {
    float angleStep = 0.15f;
    float radius = 0.18f;
    float baseRadius = 0.07f;
    float totalAngle = 7.85f;

    glColor3f(1.0f, 0.75f, 0.8f);

    for (float angle = 0.0f; angle <= totalAngle; angle += angleStep) {
        float x = radius * cosf(angle);
        float y = 0.83f + 0.04f * sinf(angle * 1.5f);
        float z = radius * sinf(angle) + 0.03f * (angle / totalAngle);
        y -= 0.015f * (angle / totalAngle);

        float taper = 1.0f - (angle / totalAngle);
        float currentRadius = baseRadius * taper;

        glPushMatrix();
        glTranslatef(x, y, z);
        glutSolidSphere(currentRadius, 20, 20);
        glPopMatrix();

        radius -= 0.003f;
    }
}
```

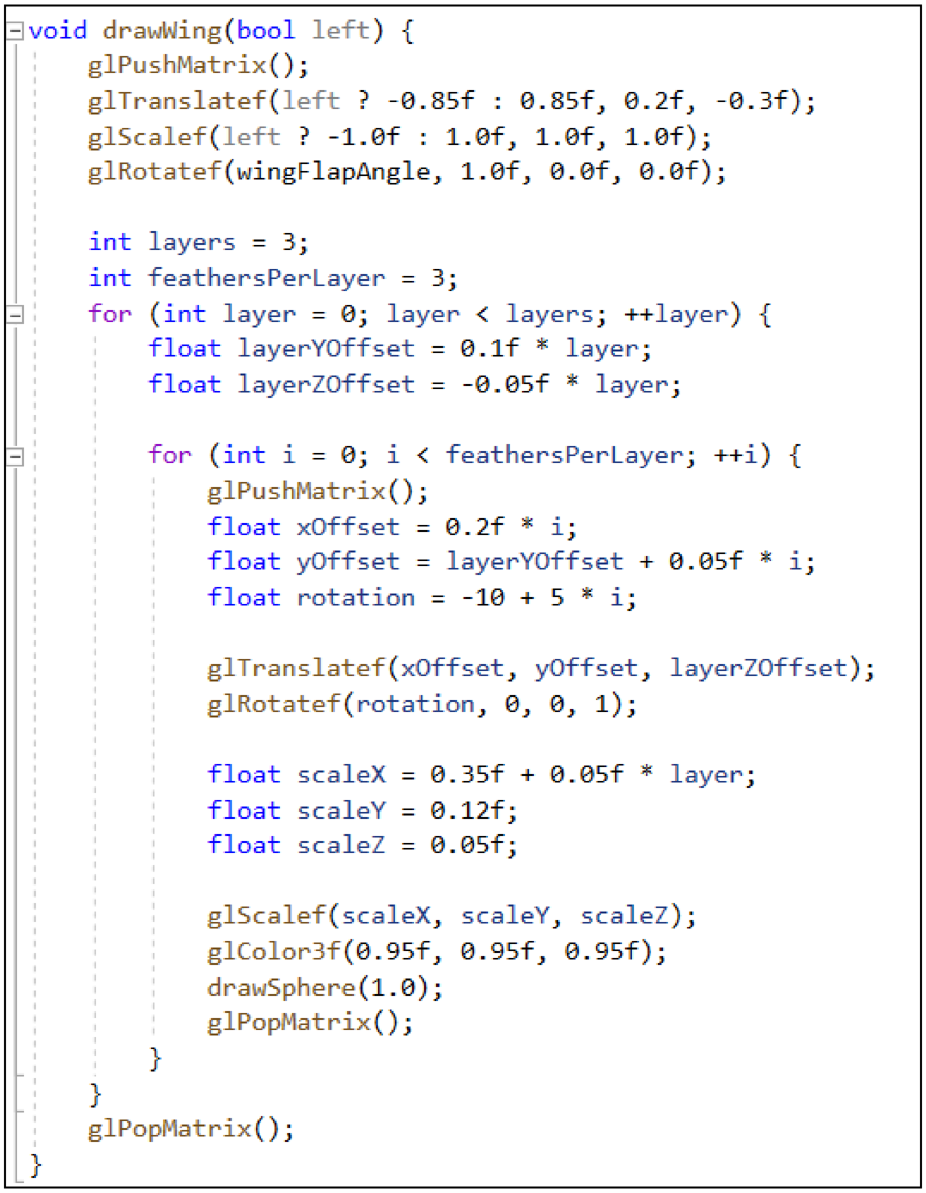
2.2 Modularity

To maintain modularity and facilitate animation of our model, each visual element of the character is implemented in a separate function such as *drawEye(), drawWing(), and drawMouth().*

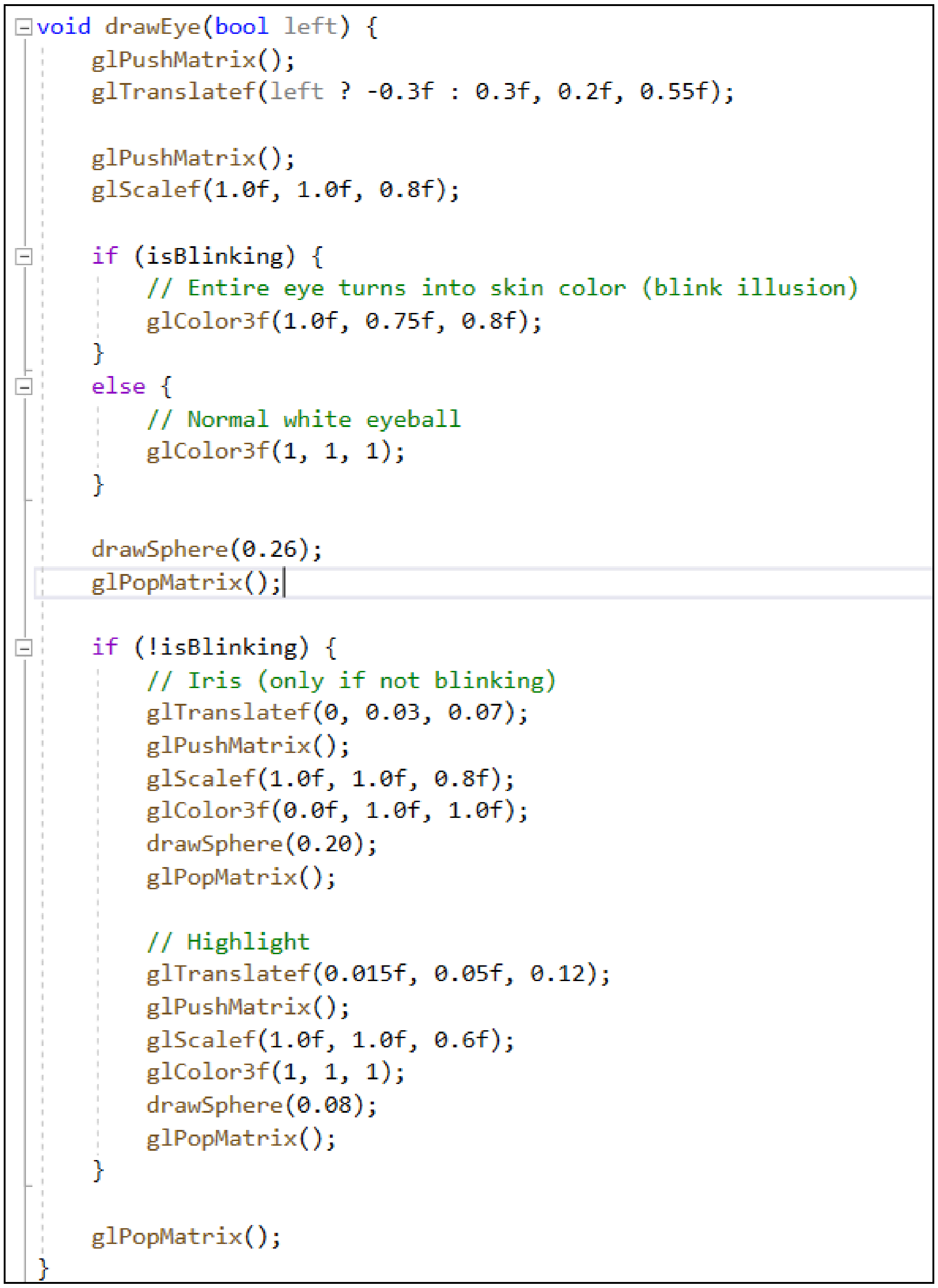
Code reference:



```
void drawMouth() {
    glPushMatrix();
    glTranslatef(0.0, -0.18, 0.75);
    glScalef(1.0f, mouthScale, 1.0f);
    glColor3f(0, 0, 0);
    glPointSize(2.5);

    glBegin(GL_POINTS);
    for (float angle = -120.0f; angle <= 120.0f; angle += 1.0f) {
        float rad = angle * 3.14159265f / 180.0f;
        float x = 0.12f * cos(rad);
        float y = 0.13f * sin(rad);
        glVertex3f(x, y, 0.0f);
    }
    glEnd();

    glPopMatrix();
}
```



```cpp
void drawWing(bool left) {
    glPushMatrix();
    glTranslatef(left ? -0.85f : 0.85f, 0.2f, -0.3f);
    glScalef(left ? -1.0f : 1.0f, 1.0f, 1.0f);
    glRotatef(wingFlapAngle, 1.0f, 0.0f, 0.0f);

    int layers = 3;
    int feathersPerLayer = 3;
    for (int layer = 0; layer < layers; ++layer) {
        float layerYOffset = 0.1f * layer;
        float layerZOffset = -0.05f * layer;

        for (int i = 0; i < feathersPerLayer; ++i) {
            glPushMatrix();
            float xOffset = 0.2f * i;
            float yOffset = layerYOffset + 0.05f * i;
            float rotation = -10 + 5 * i;

            glTranslatef(xOffset, yOffset, layerZOffset);
            glRotatef(rotation, 0, 0, 1);

            float scaleX = 0.35f + 0.05f * layer;
            float scaleY = 0.12f;
            float scaleZ = 0.05f;

            glScalef(scaleX, scaleY, scaleZ);
            glColor3f(0.95f, 0.95f, 0.95f);
            drawSphere(1.0);
            glPopMatrix();
        }
    }
    glPopMatrix();
}
```



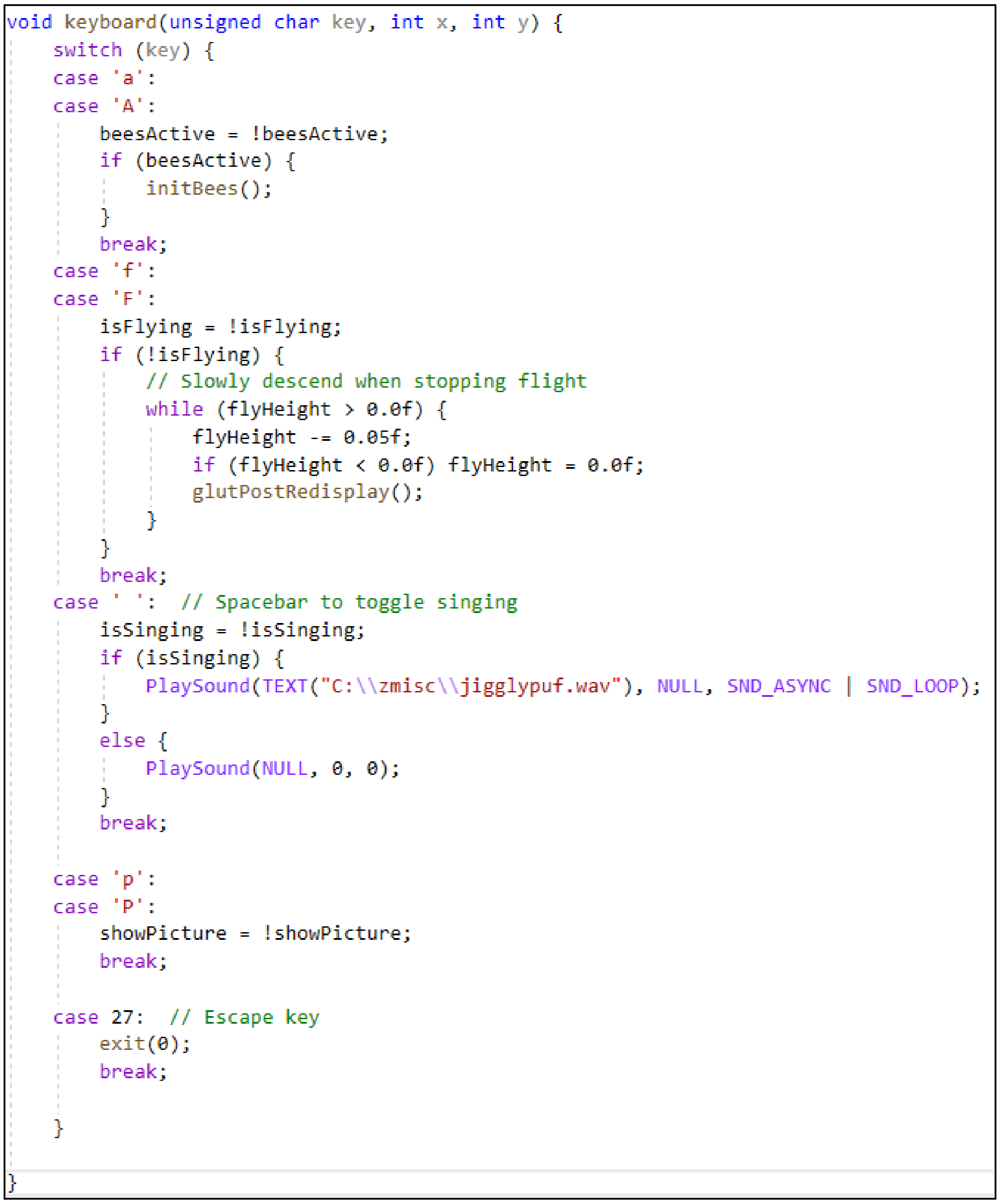
```
void drawEye(bool left) {
    glPushMatrix();
    glTranslatef(left ? -0.3f : 0.3f, 0.2f, 0.55f);

    glPushMatrix();
    glScalef(1.0f, 1.0f, 0.8f);

    if (isBlinking) {
        // Entire eye turns into skin color (blink illusion)
        glColor3f(1.0f, 0.75f, 0.8f);
    }
    else {
        // Normal white eyeball
        glColor3f(1, 1, 1);
    }

    drawSphere(0.26);
    glPopMatrix();

    if (!isBlinking) {
        // Iris (only if not blinking)
        glTranslatef(0, 0.03, 0.07);
        glPushMatrix();
        glScalef(1.0f, 1.0f, 0.8f);
        glColor3f(0.0f, 1.0f, 1.0f);
        drawSphere(0.20);
        glPopMatrix();

        // Highlight
        glTranslatef(0.015f, 0.05f, 0.12);
        glPushMatrix();
        glScalef(1.0f, 1.0f, 0.6f);
        glColor3f(1, 1, 1);
        drawSphere(0.08);
        glPopMatrix();
    }

    glPopMatrix();
}
```

2.3 Interaction Design

User interaction is supported via both keyboard and mouse inputs. The keyboard controls toggling of animations and sounds, while mouse inputs allow for scene rotation.

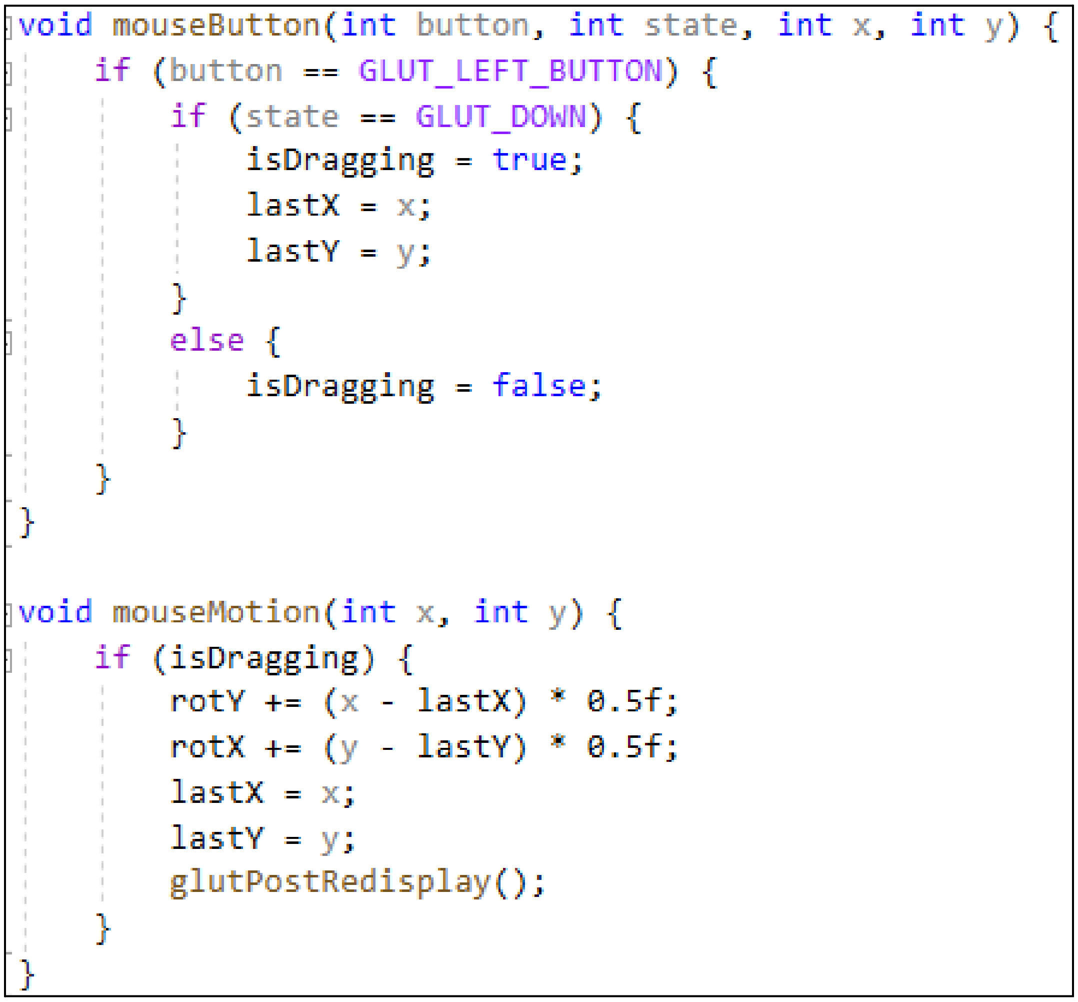
Code reference:

a) Keyboard



```
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
    case 'a':
    case 'A':
        beesActive = !beesActive;
        if (beesActive) {
            initBees();
        }
        break;
    case 'f':
    case 'F':
        isFlying = !isFlying;
        if (!isFlying) {
            // Slowly descend when stopping flight
            while (flyHeight > 0.0f) {
                flyHeight -= 0.05f;
                if (flyHeight < 0.0f) flyHeight = 0.0f;
                glutPostRedisplay();
            }
        }
        break;
    case ' ':  // Spacebar to toggle singing
        isSinging = !isSinging;
        if (isSinging) {
            PlaySound(TEXT("C:\\zmisc\\jigglypuf.wav"), NULL, SND_ASYNC | SND_LOOP);
        }
        else {
            PlaySound(NULL, 0, 0);
        }
        break;

    case 'p':
    case 'P':
        showPicture = !showPicture;
        break;

    case 27:  // Escape key
        exit(0);
        break;

    }

}
```

b) Mouse Input

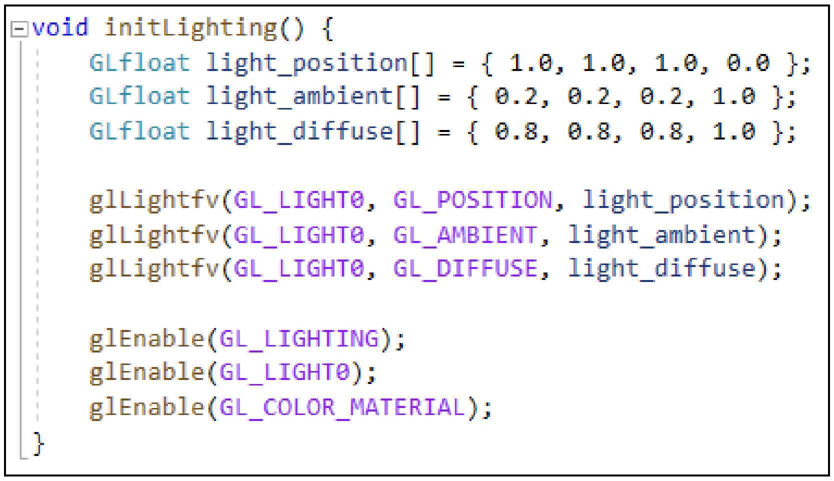


```
void mouseButton(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON) {
        if (state == GLUT_DOWN) {
            isDragging = true;
            lastX = x;
            lastY = y;
        }
        else {
            isDragging = false;
        }
    }
}

void mouseMotion(int x, int y) {
    if (isDragging) {
        rotY += (x - lastX) * 0.5f;
        rotX += (y - lastY) * 0.5f;
        lastX = x;
        lastY = y;
        glutPostRedisplay();
    }
}
```

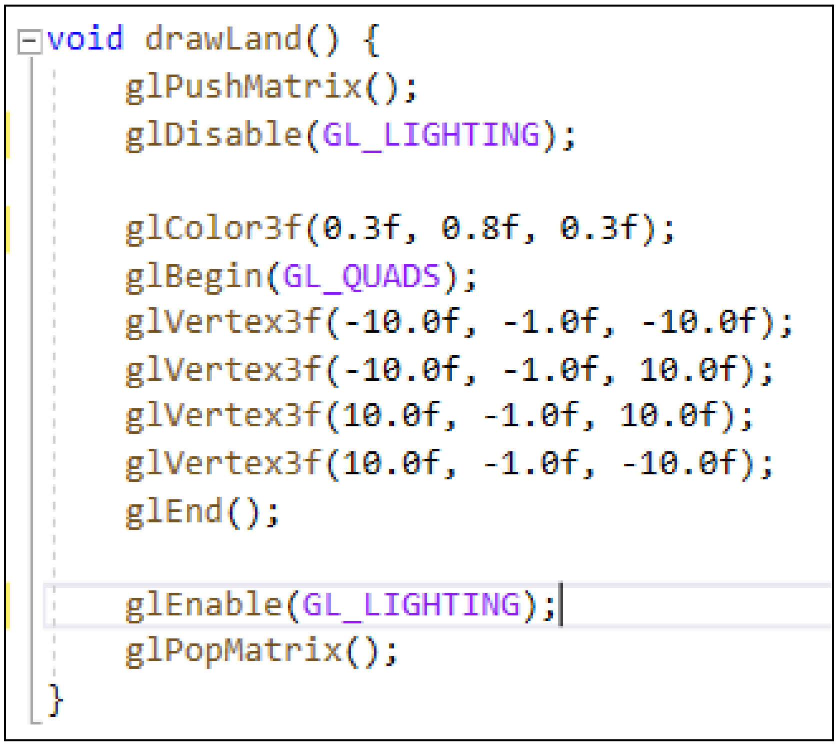
2.4 Scene Setup

A 3D perspective projection is used to render the scene, with lighting enabled for realism. The ground is drawn using a large green quad representing grass.

Code reference:



```
void initLighting() {
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
    GLfloat light_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
    GLfloat light_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };

    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_COLOR_MATERIAL);
}
```



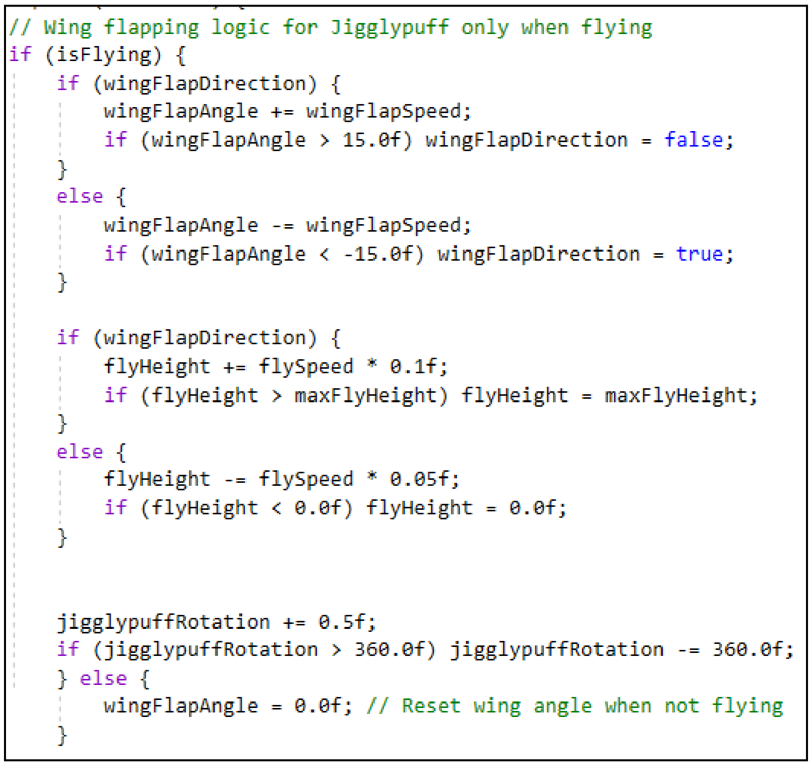
```
void drawLand() {
    glPushMatrix();
    glDisable(GL_LIGHTING);

    glColor3f(0.3f, 0.8f, 0.3f);
    glBegin(GL_QUADS);
    glVertex3f(-10.0f, -1.0f, -10.0f);
    glVertex3f(-10.0f, -1.0f, 10.0f);
    glVertex3f(10.0f, -1.0f, 10.0f);
    glVertex3f(10.0f, -1.0f, -10.0f);
    glEnd();

    glEnable(GL_LIGHTING);
    glPopMatrix();
}
```

3.0 IMPLEMENTATION & ACHIEVEMENTS

Key features that we have implemented in our projects are as follows:

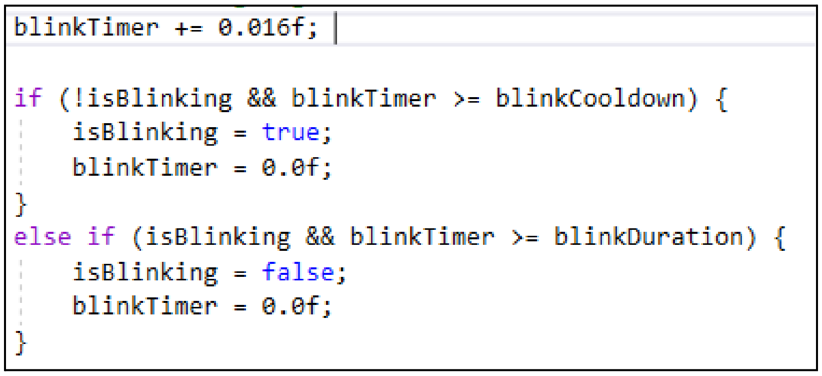
a) Character animation:

- Wing flapping



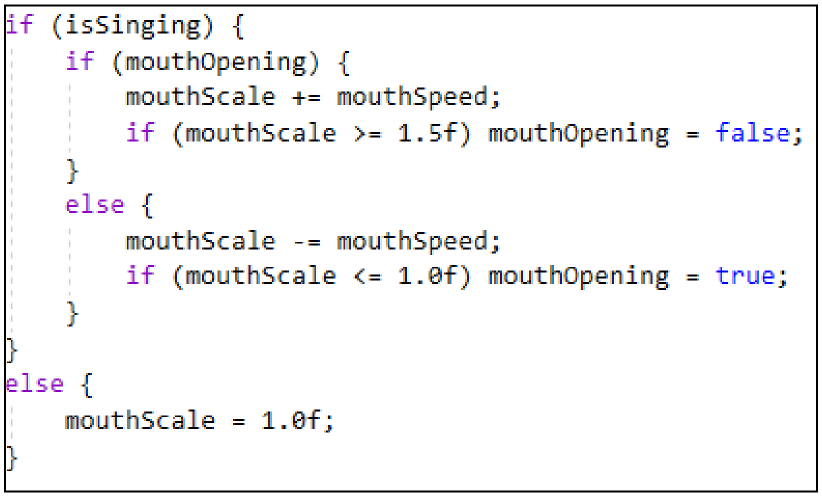
```
// Wing flapping logic for Jigglypuff only when flying
if (isFlying) {
    if (wingFlapDirection) {
        wingFlapAngle += wingFlapSpeed;
        if (wingFlapAngle > 15.0f) wingFlapDirection = false;
    }
    else {
        wingFlapAngle -= wingFlapSpeed;
        if (wingFlapAngle < -15.0f) wingFlapDirection = true;
    }

    if (wingFlapDirection) {
        flyHeight += flySpeed * 0.1f;
        if (flyHeight > maxFlyHeight) flyHeight = maxFlyHeight;
    }
    else {
        flyHeight -= flySpeed * 0.05f;
        if (flyHeight < 0.0f) flyHeight = 0.0f;
    }


    jigglypuffRotation += 0.5f;
    if (jigglypuffRotation > 360.0f) jigglypuffRotation -= 360.0f;
    } else {
        wingFlapAngle = 0.0f; // Reset wing angle when not flying
    }
```

- Blinking eyes with timer



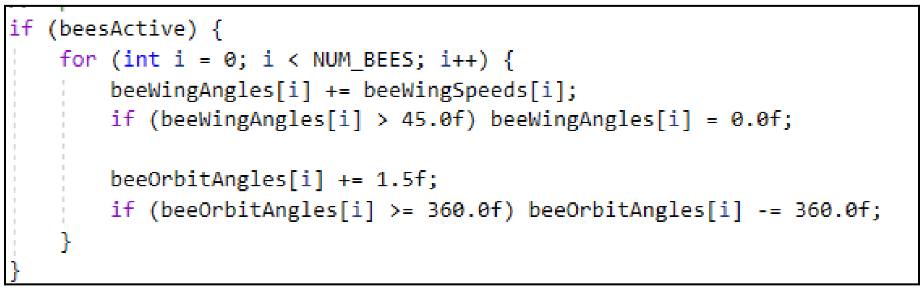
```
blinkTimer += 0.016f;

if (!isBlinking && blinkTimer >= blinkCooldown) {
    isBlinking = true;
    blinkTimer = 0.0f;
}
else if (isBlinking && blinkTimer >= blinkDuration) {
    isBlinking = false;
    blinkTimer = 0.0f;
}
```

- Singing with dynamic mouth scaling and looping sound



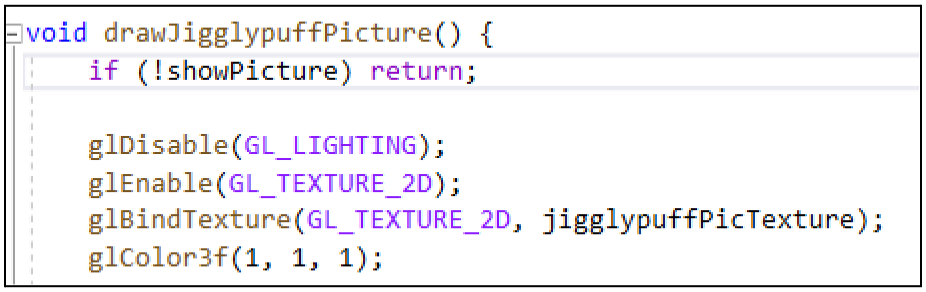
```
if (isSinging) {
    if (mouthOpening) {
        mouthScale += mouthSpeed;
        if (mouthScale >= 1.5f) mouthOpening = false;
    }
    else {
        mouthScale -= mouthSpeed;
        if (mouthScale <= 1.0f) mouthOpening = true;
    }
}
else {
    mouthScale = 1.0f;
}
```

b) Bee swarm animation



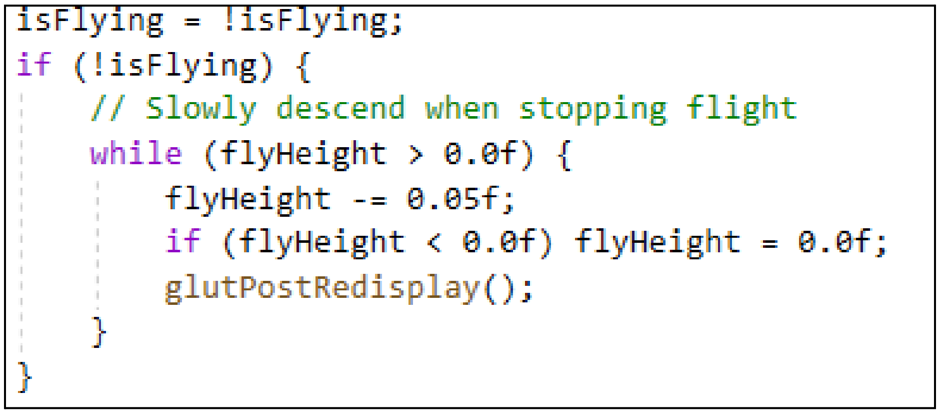
```
if (beesActive) {
    for (int i = 0; i < NUM_BEES; i++) {
        beeWingAngles[i] += beeWingSpeeds[i];
        if (beeWingAngles[i] > 45.0f) beeWingAngles[i] = 0.0f;

        beeOrbitAngles[i] += 1.5f;
        if (beeOrbitAngles[i] >= 360.0f) beeOrbitAngles[i] -= 360.0f;
    }
}
```

c) Picture display



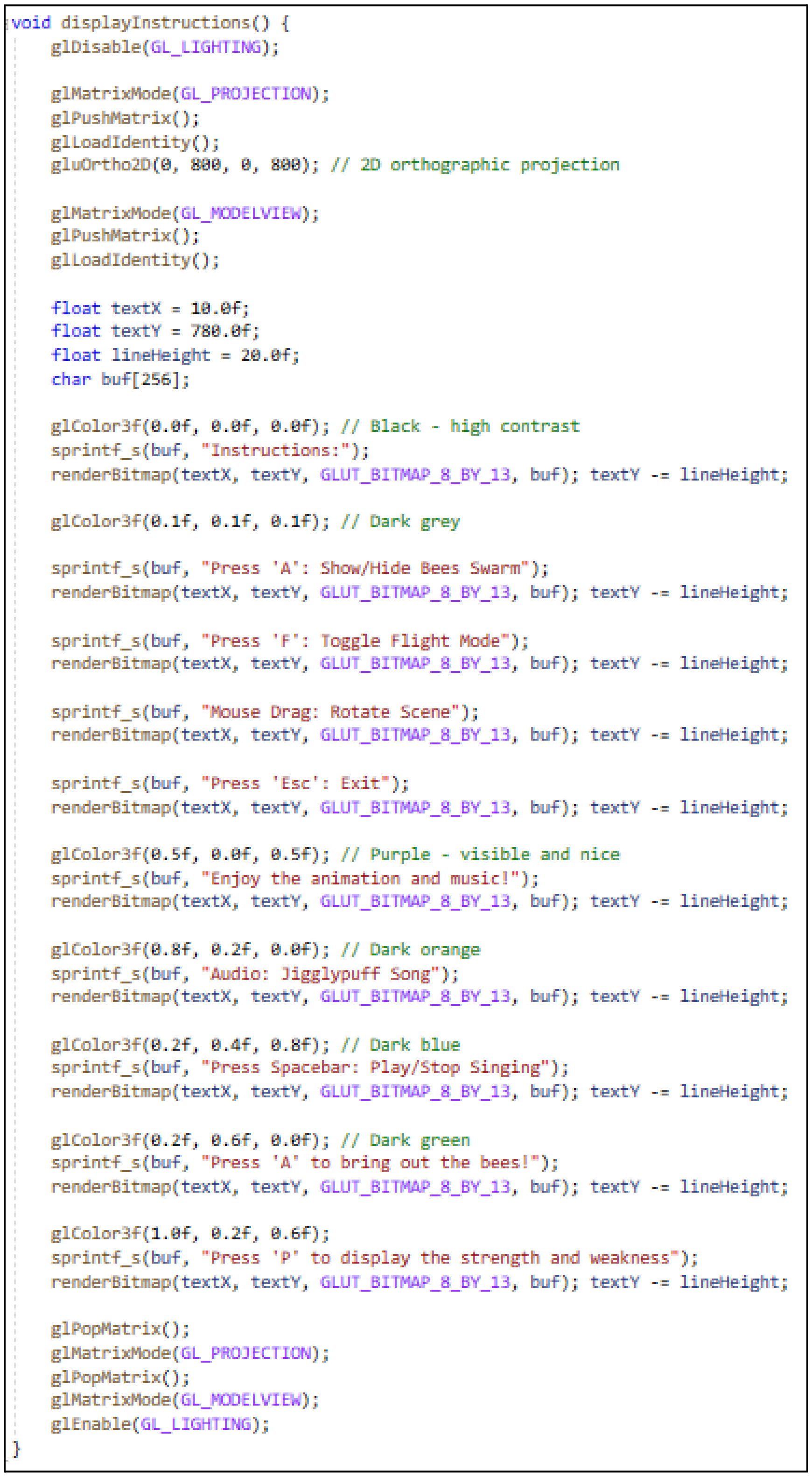
```
void drawJigglypuffPicture() {
    if (!showPicture) return;

    glDisable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, jigglypuffPicTexture);
    glColor3f(1, 1, 1);
```

d) Flying simulation



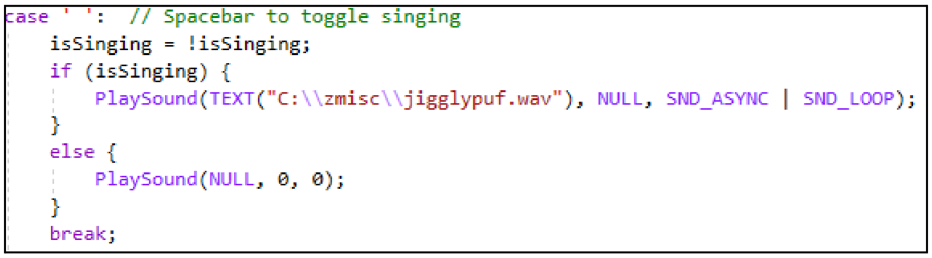
```
isFlying = !isFlying;
if (!isFlying) {
    // Slowly descend when stopping flight
    while (flyHeight > 0.0f) {
        flyHeight -= 0.05f;
        if (flyHeight < 0.0f) flyHeight = 0.0f;
        glutPostRedisplay();
    }
}
```

e) Text Instructions Overlay



```
void displayInstructions() {
    glDisable(GL_LIGHTING);

    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    gluOrtho2D(0, 800, 0, 800); // 2D orthographic projection

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();

    float textX = 10.0f;
    float textY = 780.0f;
    float lineHeight = 20.0f;
    char buf[256];

    glColor3f(0.0f, 0.0f, 0.0f); // Black - high contrast
    sprintf_s(buf, "Instructions:");
    renderBitmap(textX, textY, GLUT_BITMAP_8_BY_13, buf); textY -= lineHeight;

    glColor3f(0.1f, 0.1f, 0.1f); // Dark grey

    sprintf_s(buf, "Press 'A': Show/Hide Bees Swarm");
    renderBitmap(textX, textY, GLUT_BITMAP_8_BY_13, buf); textY -= lineHeight;

    sprintf_s(buf, "Press 'F': Toggle Flight Mode");
    renderBitmap(textX, textY, GLUT_BITMAP_8_BY_13, buf); textY -= lineHeight;

    sprintf_s(buf, "Mouse Drag: Rotate Scene");
    renderBitmap(textX, textY, GLUT_BITMAP_8_BY_13, buf); textY -= lineHeight;

    sprintf_s(buf, "Press 'Esc': Exit");
    renderBitmap(textX, textY, GLUT_BITMAP_8_BY_13, buf); textY -= lineHeight;

    glColor3f(0.5f, 0.0f, 0.5f); // Purple - visible and nice
    sprintf_s(buf, "Enjoy the animation and music!");
    renderBitmap(textX, textY, GLUT_BITMAP_8_BY_13, buf); textY -= lineHeight;

    glColor3f(0.8f, 0.2f, 0.0f); // Dark orange
    sprintf_s(buf, "Audio: Jigglypuff Song");
    renderBitmap(textX, textY, GLUT_BITMAP_8_BY_13, buf); textY -= lineHeight;

    glColor3f(0.2f, 0.4f, 0.8f); // Dark blue
    sprintf_s(buf, "Press Spacebar: Play/Stop Singing");
    renderBitmap(textX, textY, GLUT_BITMAP_8_BY_13, buf); textY -= lineHeight;

    glColor3f(0.2f, 0.6f, 0.0f); // Dark green
    sprintf_s(buf, "Press 'A' to bring out the bees!");
    renderBitmap(textX, textY, GLUT_BITMAP_8_BY_13, buf); textY -= lineHeight;

    glColor3f(1.0f, 0.2f, 0.6f);
    sprintf_s(buf, "Press 'P' to display the strength and weakness");
    renderBitmap(textX, textY, GLUT_BITMAP_8_BY_13, buf); textY -= lineHeight;

    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
    glEnable(GL_LIGHTING);
}
```

f) Audio Display



```
case ' ':  // Spacebar to toggle singing
    isSinging = !isSinging;
    if (isSinging) {
        PlaySound(TEXT("C:\\zmisc\\jigglypuf.wav"), NULL, SND_ASYNC | SND_LOOP);
    }
    else {
        PlaySound(NULL, 0, 0);
    }
    break;
```

4.0 DISCUSSION ON EACH TOPIC

4.1 Modeling Technique

In this project, the modeling technique focuses on primitive-based modeling using OpenGL's GLUT built-in shapes. The main primitive used is glutSolidSphere(), which provides a smooth, rounded surface suitable for constructing the character Jigglypuff, whose design is mostly spherical and soft.

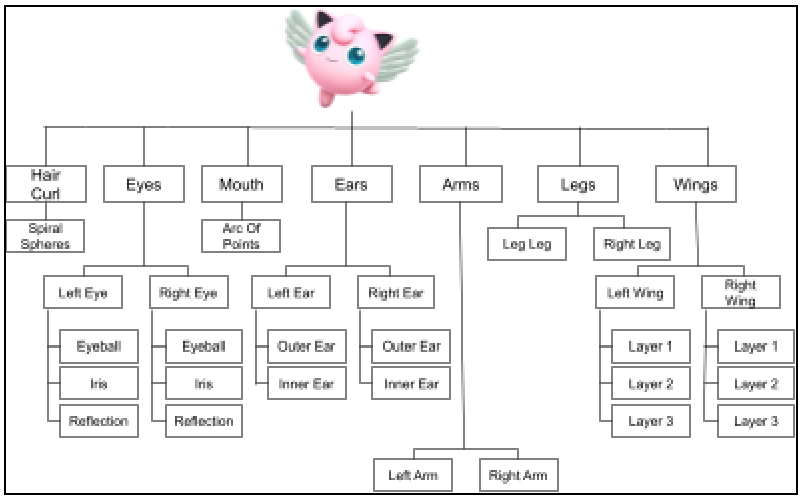
Each part of the character is built by applying geometric transformations to the primitives:

- Translation: Moves the parts to their correct positions relative to the main body.
- Scaling: Adjusts the dimensions of each primitive to achieve the proper proportions. For example, the limbs and ears are made by scaling spheres into elongated or flattened forms.
- Rotation: Rotates components such as hair curls, ears, and wings to the desired angles.

The combination of these transformations allows us to "shape" the primitive into more complex forms without needing custom mesh models. This technique also follows a modular and hierarchical approach. Each component (eyes, ears, mouth, limbs, hair curls, wings) is built in a separate function. This makes the code easier to manage and allows for individual animation of each part.

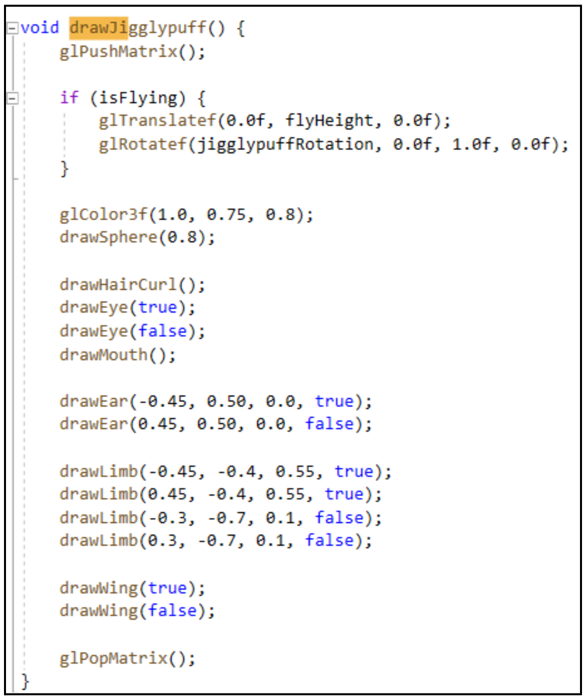
Since Jigglypuff’s character has a rounded cartoon-like style, using spheres as the base primitive results in a model that looks natural and matches the original design of the character. This modeling approach also supports interactive features and animations. For example, during animations like flying, blinking, or singing, only the related parts (such as wings, eyes, or mouth) are updated independently, thanks to the organized structure.

4.2 Hierarchical Structure



This model follows a hierarchical model structure, where transformations are applied in a parent-child relationship using glPushMatrix() and glPopMatrix(). In the drawJigglyPuff() function, the body acts as the root node. Child elements such as eyes, ears, wings, limbs and mouth are positioned to the body. This allows for localized animation for individual parts without affecting the overall character.

Code reference:



```
void drawJigglypuff() {
    glPushMatrix();

    if (isFlying) {
        glTranslatef(0.0f, flyHeight, 0.0f);
        glRotatef(jigglypuffRotation, 0.0f, 1.0f, 0.0f);
    }

    glColor3f(1.0, 0.75, 0.8);
    drawSphere(0.8);

    drawHairCurl();
    drawEye(true);
    drawEye(false);
    drawMouth();

    drawEar(-0.45, 0.50, 0.0, true);
    drawEar(0.45, 0.50, 0.0, false);

    drawLimb(-0.45, -0.4, 0.55, true);
    drawLimb(0.45, -0.4, 0.55, true);
    drawLimb(-0.3, -0.7, 0.1, false);
    drawLimb(0.3, -0.7, 0.1, false);

    drawWing(true);
    drawWing(false);

    glPopMatrix();
}
```

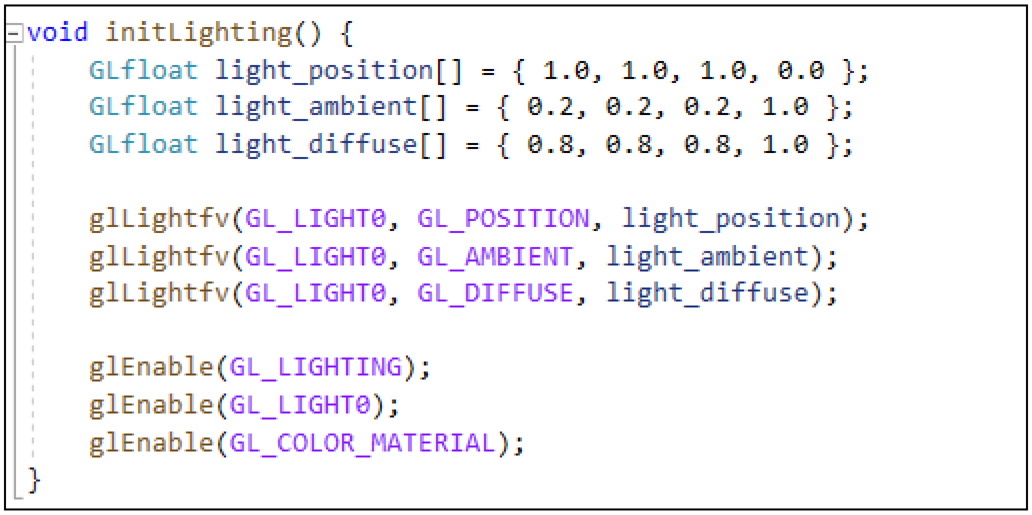
4.3 Lighting and Shading

Lighting in the scene was implemented using a combination of lightings to create a sense of depth and realism.

Type of lighting applied:

1. Ambient Lighting
    - This is to provide a base illumination across all surfaces.
2. Diffuse Lighting
    - This is to simulate directional light interaction based on surface normals.

Code reference:



```
void initLighting() {
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
    GLfloat light_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
    GLfloat light_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };

    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_COLOR_MATERIAL);
}
```

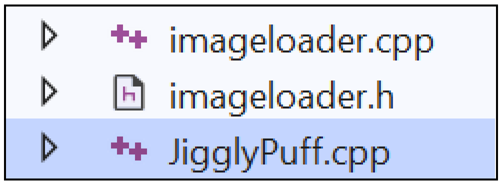
4.4 Animation and State Control

All animation states are controlled using global flags, such as isFlying, isSinging and isBlinking. It is updated in the update() timer function every ~16ms.

4.5 Texture Mapping

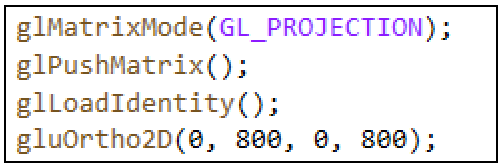
When the 'P' key is pressed, a BMP image of Jigglypuff is rendered as a 2D overlay using a texture-mapped quad in orthographic projection. This was achieved by:

1. Loading the texture using a custom ImageLoader.



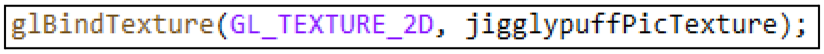
```
imageloader.cpp
imageloader.h
JigglyPuff.cpp
```

2. Switching to glOrtho2D for 2D drawing inside drawJigglypuffPicture() function.

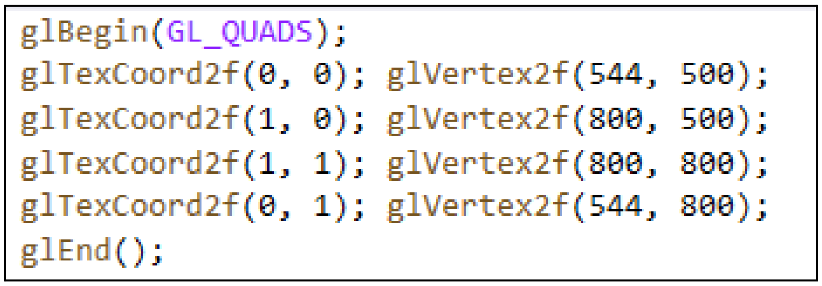


```
glMatrixMode(GL_PROJECTION);
glPushMatrix();
glLoadIdentity();
gluOrtho2D(0, 800, 0, 800);
```

3. Mapping the image onto a rectangular region



```
glBindTexture(GL_TEXTURE_2D, jigglypuffPicTexture);
```

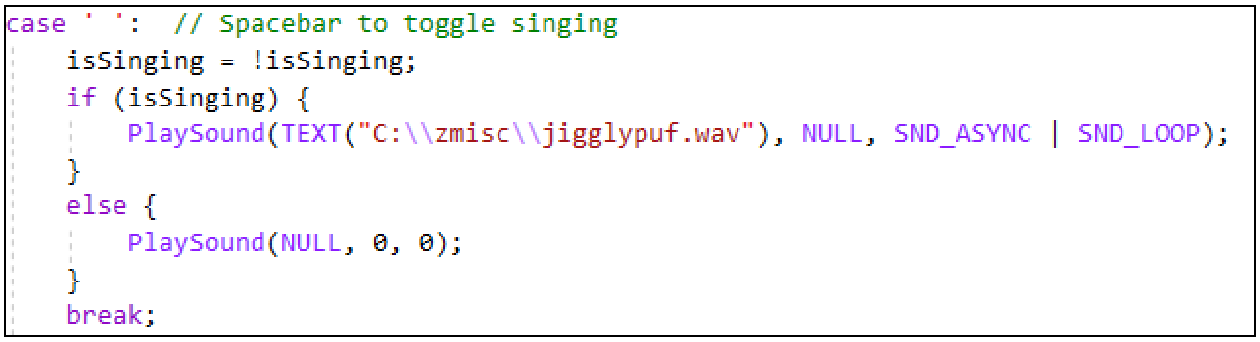


```
glBegin(GL_QUADS);
glTexCoord2f(0, 0); glVertex2f(544, 500);
glTexCoord2f(1, 0); glVertex2f(800, 500);
glTexCoord2f(1, 1); glVertex2f(800, 800);
glTexCoord2f(0, 1); glVertex2f(544, 800);
glEnd();
```

4.6 Audio Integration

Audio is integrated using the Windows Sound API. By pressing the spacebar, it initiates the playback of the audio in a loop. Pressing it again stops the music. This adds an interactive experience to the model.

Code reference:



```
case ' ':  // Spacebar to toggle singing
    isSinging = !isSinging;
    if (isSinging) {
        PlaySound(TEXT("C:\\zmisc\\jigglypuf.wav"), NULL, SND_ASYNC | SND_LOOP);
    }
    else {
        PlaySound(NULL, 0, 0);
    }
    break;
```

4.7 Extra Features

The extra features that we have added to increase the interactivity of our model are:

1. **Bee Swarm**

   Description:

   When the user presses ‘A’ key, a swarm of animated 12 bees appears and orbits around JigglyPuff along with flapping its wing independently using randomized speed.

   Concept used:

   - Trigonometry: Polar coordinates to calculate the orbit of the bee positions.
   - Timing: The wing angles and movements are independently moving under a randomized timing.
   - Transformation hierarchy: Each bee model is reusable and is drawn in its own glPushMatrix().

2. **Text Instruction Overlay**

   Description:

   Instructions are displayed on the top left corner of the screen using 2D text rendering.

   Concept used:

   - Orthographic projection: Switching from 3D to 2D projection using gluOrtho2D.
   - Bitmap text rendering: Disables lighting to render flat 2D text.

3. **Mouse Rotation**

   Description:

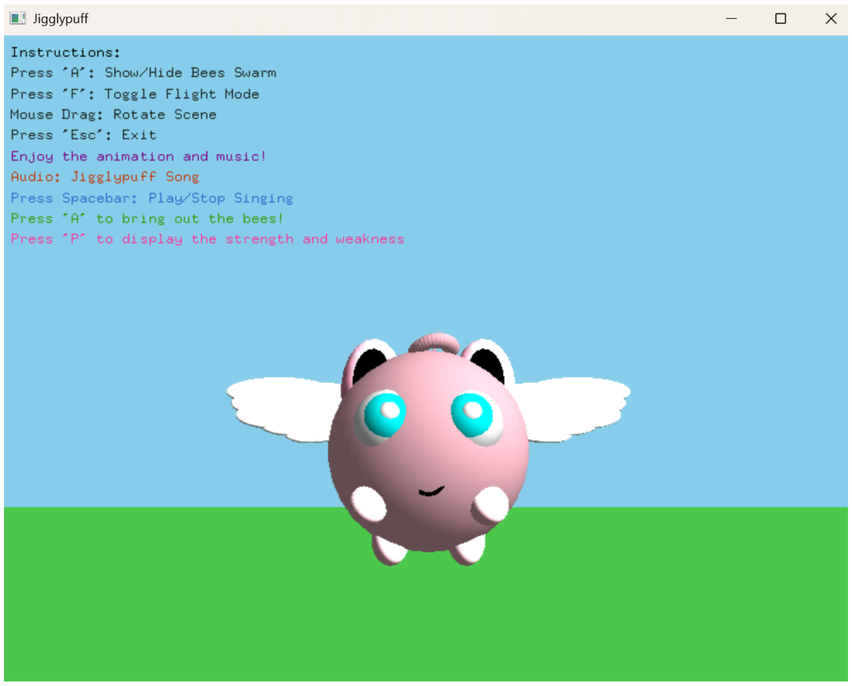
   The user can click and drag with the mouse to rotate the entire 3D scene horizontally and vertically.

   Concept used:

   - View transformation
   - Input mapping.

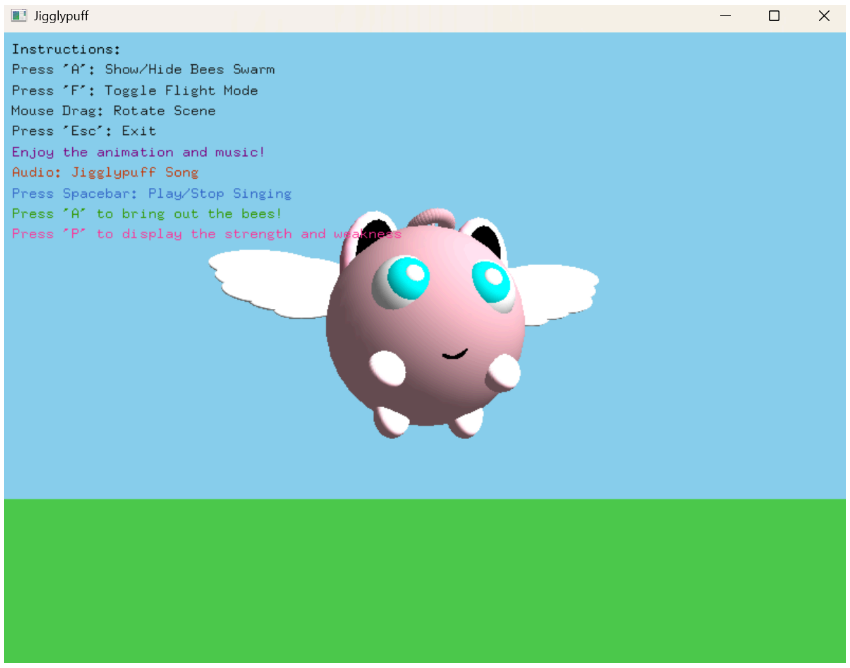
5.0 SAMPLE OUTPUT

JigglyPuff remain constant



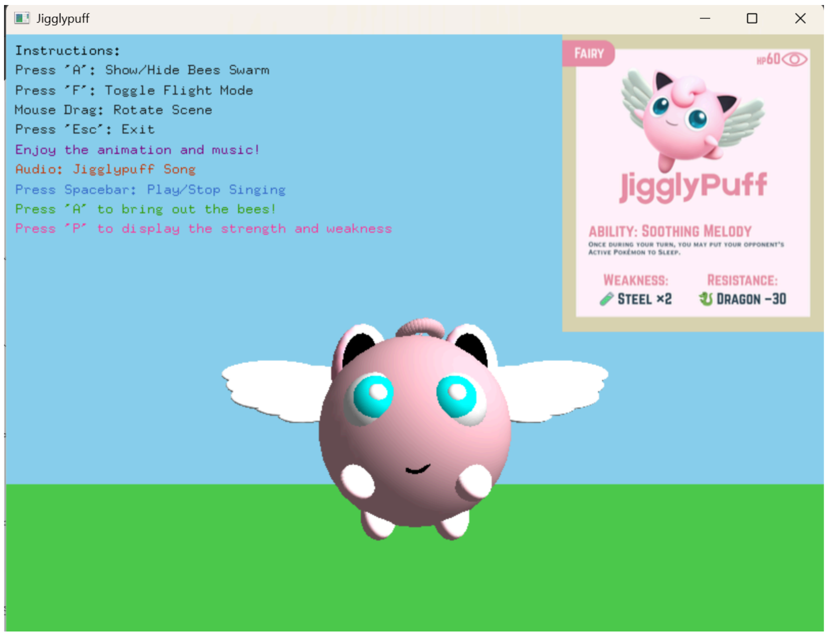
```
Instructions:
Press 'A': Show/Hide Bees Swarm
Press 'F': Toggle Flight Mode
Mouse Drag: Rotate Scene
Press 'Esc': Exit
Enjoy the animation and music!
Audio: Jigglypuff Song
Press Spacebar: Play/Stop Singing
Press 'A' to bring out the bees!
Press 'P' to display the strength and weakness
```

JigglyPuff is flying



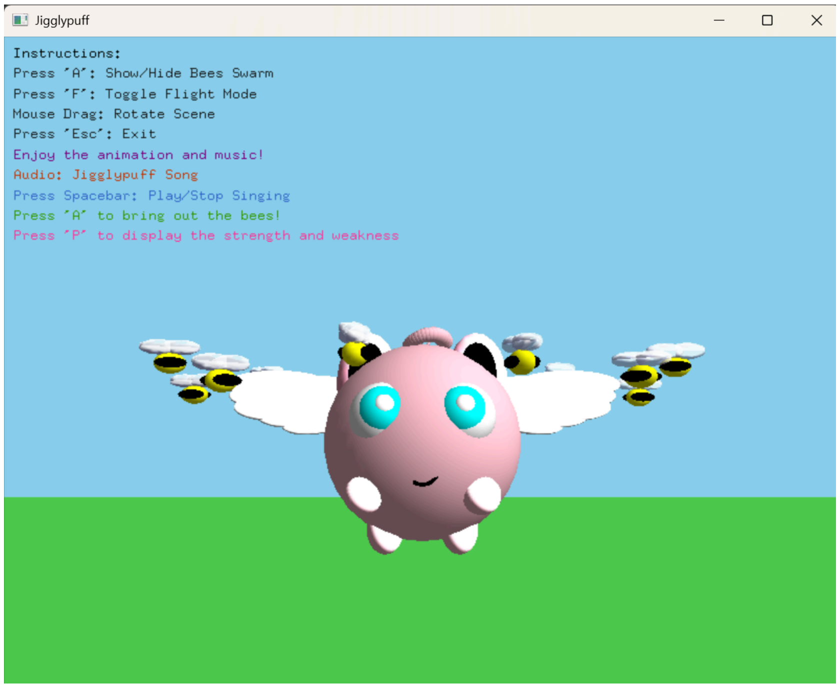
```
Instructions:
Press 'A': Show/Hide Bees Swarm
Press 'F': Toggle Flight Mode
Mouse Drag: Rotate Scene
Press 'Esc': Exit
Enjoy the animation and music!
Audio: Jigglypuff Song
Press Spacebar: Play/Stop Singing
Press 'A' to bring out the bees!
Press 'P' to display the strength and weakness
```

JigglyPuff show the power



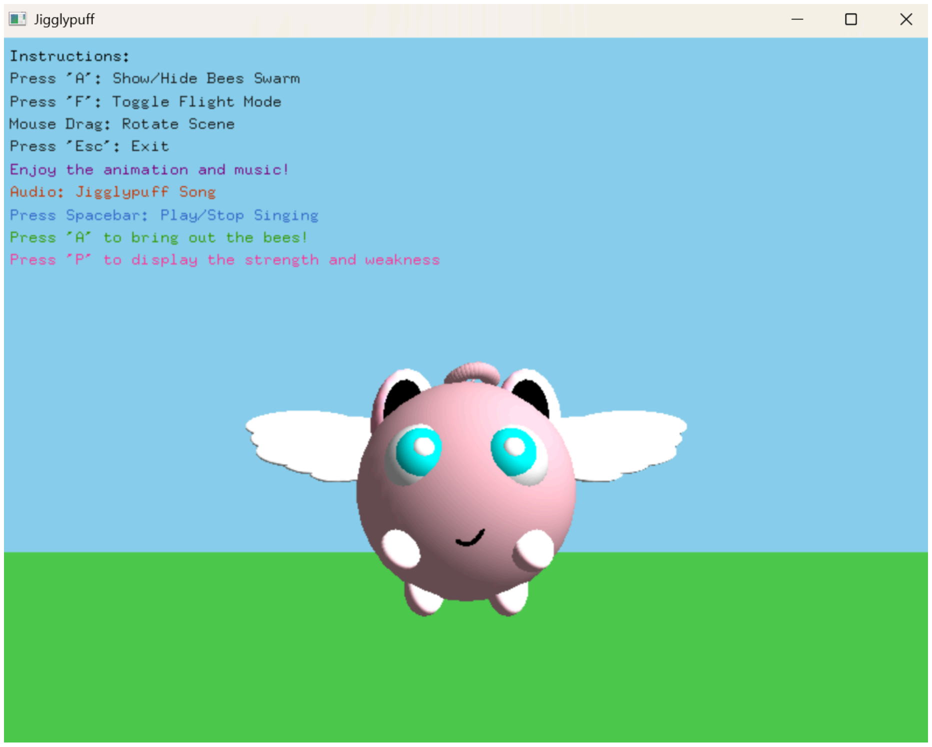
```
Instructions:
Press 'A': Show/Hide Bees Swarm
Press 'F': Toggle Flight Mode
Mouse Drag: Rotate Scene
Press 'Esc': Exit
Enjoy the animation and music!
Audio: Jigglypuff Song
Press Spacebar: Play/Stop Singing
Press 'A' to bring out the bees!
Press 'P' to display the strength and weakness
```

JigglyPuff play with the bees



```
Instructions:
Press 'A': Show/Hide Bees Swarm
Press 'F': Toggle Flight Mode
Mouse Drag: Rotate Scene
Press 'Esc': Exit
Enjoy the animation and music!
Audio: Jigglypuff Song
Press Spacebar: Play/Stop Singing
Press 'A' to bring out the bees!
Press 'P' to display the strength and weakness
```

Jigglypuff singing with mouth movement



```
Instructions:
Press 'A': Show/Hide Bees Swarm
Press 'F': Toggle Flight Mode
Mouse Drag: Rotate Scene
Press 'Esc': Exit
Enjoy the animation and music!
Audio: Jigglypuff Song
Press Spacebar: Play/Stop Singing
Press 'A' to bring out the bees!
Press 'P' to display the strength and weakness
```

6.0 CONCLUSION

In conclusion, this project successfully showcases the use of fundamental computer graphics techniques through the creation of an animated 3D Jigglypuff model using OpenGL. The character is modeled hierarchically using GLUT primitives, with each body part structured through transformation stacks. Proper lighting and shading are implemented using OpenGL's fixed-function pipeline to enhance realism. The use of both perspective and orthographic projections, along with an appropriate camera model, allows for dynamic 3D rendering and 2D overlays. Overall, the project demonstrates a strong understanding of fundamental computer graphics principles.

7.0 REFERENCE

BMP file format - Wikipedia. (n.d.). Wikipedia, the free encyclopedia.Retrieved June 24, 2025, from https://en.wikipedia.org/wiki/BMP_file_format

Dey, S. (2023, September 19). *Creating animations using Transformations in OpenGL*. GeeksforGeeks. Retrieved June 24, 2025, from https://www.geeksforgeeks.org/dsa/creating-animations-using-transformations-opengl/

Jigglypuff | Pokémon Wiki | Fandom. (n.d.). Pokémon Wiki. Retrieved June 24, 2025, from https://pokemon.fandom.com/wiki/Jigglypuff

LearnOpenGL - Coordinate Systems. (n.d.). Learn OpenGL. Retrieved June 24, 2025, from https://learnopengl.com/

The Pixel Swarm. (2012, October 3). thndl. Retrieved June 24, 2025, from https://thndl.com/the-pixel-swarm.html