NOTRE DAME UNIVERSITY BANGLADESH

Computer Graphics Lab Project Report

Course Code: CSE-4204
Course Title: Computer Graphics Lab
Project Title: Egg Drop Saga (Game)

Submitted by:

Name: Istiak Alam
ID: 0692230005101005
Name: Tanvir Hossain Ratul
ID: 0692230005101004
Batch: CSE-20

Submission Date: June 13, 2026

Submitted to:

Humayara Binte Rashid
Lecturer, Dept. of CSE
Notre Dame University Bangladesh

Table of Contents

Abstract

Egg Drop Saga is a 2D arcade-style game developed using OpenGL and GLUT in C++ as part of the Computer Graphics Lab course. The project demonstrates the practical application of fundamental computer graphics concepts including coordinate systems, geometric modeling, transformations, animation, rendering pipelines, and user interaction. In the game, a player controls a bucket to catch eggs dropped by moving chickens while avoiding missed catches that reduce the player's lives. The project incorporates real-time animation, keyboard-based controls, collision detection, score tracking, audio integration, and multiple game states such as menu, gameplay, pause, and game-over screens. The visual elements are created using OpenGL primitive drawing techniques and coordinate-based object construction. The primary objective of the project is to apply theoretical graphics concepts in an interactive environment while gaining experience in event-driven programming and real-time rendering. The final implementation successfully demonstrates the integration of computer graphics principles with game development techniques.

Chapter 1 Introduction

1.1 Project Background

Computer Graphics is a branch of computer science that focuses on the creation, manipulation, and rendering of visual content using computational techniques. OpenGL is one of the most widely used graphics libraries for developing interactive graphical applications and games. As part of the Computer Graphics Laboratory course, students are encouraged to apply theoretical concepts such as coordinate systems, geometric modeling, transformations, animation, rendering, and event handling in a practical project.

The project **Egg Drop Saga** is a 2D arcade-style game developed using C++ and OpenGL (GLUT). The game simulates a farm environment where chickens continuously move and drop eggs from the top of the screen. The player controls a bucket positioned at the bottom of the screen and must catch the falling eggs to earn points while avoiding missed catches that reduce the player's lives. The project integrates graphical object modeling, real-time animation, collision detection, user interaction, and audio feedback into a complete gaming experience.

1.2 Motivation

The inspiration for this project came from classic arcade catching games that emphasize quick reaction and hand-eye coordination. These games are simple in design yet provide an engaging user experience through continuous movement, scoring systems, and increasing difficulty.

From an academic perspective, the project offered an opportunity to implement various computer graphics concepts learned throughout the course in a practical environment. Developing a complete game using OpenGL allowed the team to gain hands-on experience with rendering pipelines, object-oriented design, animation techniques, coordinate-based drawing, and event-driven programming while creating an enjoyable interactive application.

1.3 Objectives

The primary objectives of the Egg Drop Saga project are:

- To develop a fully functional 2D game using OpenGL and GLUT.
- To understand and implement the OpenGL rendering pipeline.
- To apply coordinate-based geometric modeling for creating game objects such as chickens, eggs, buckets, and backgrounds.
- To implement real-time animation for moving objects within the game environment.
- To develop keyboard-based user interaction for controlling game elements.
- To implement collision detection between falling eggs and the player's bucket.
- To create a score and life management system.
- To integrate sound effects and background music for enhanced gameplay.
- To gain practical experience in object-oriented game development using C++.

1.4 Scope

The scope of the project includes the design and implementation of a complete 2D arcade game using OpenGL. The project covers graphical object creation, rendering, animation, collision detection, score management, sound integration, and user interaction through keyboard controls.

The game contains multiple functional components including a main menu, gameplay screen, pause system, score tracking, life management, and game-over interface. All visual elements are generated using OpenGL primitives and coordinate-based drawing techniques rather than external game engines.

However, the project does not include advanced features such as 3D graphics, online multiplayer support, artificial intelligence opponents, physics engines, networking systems, or database integration. The focus remains on demonstrating core computer graphics principles within a 2D interactive environment.

Chapter 2 System Design

2.1 System Overview

Egg Drop Saga is implemented using an event-driven architecture based on OpenGL and GLUT. The system is organized around a central `Game` class that manages gameplay logic, rendering operations, collision detection, animation updates, score management, and state transitions.

The application begins execution from `main.cpp`, where the OpenGL environment, GLUT window, audio system, and callback functions are initialized. Once initialization is complete, GLUT takes control through its event loop and continuously invokes registered callback functions.

The overall execution flow is shown below:

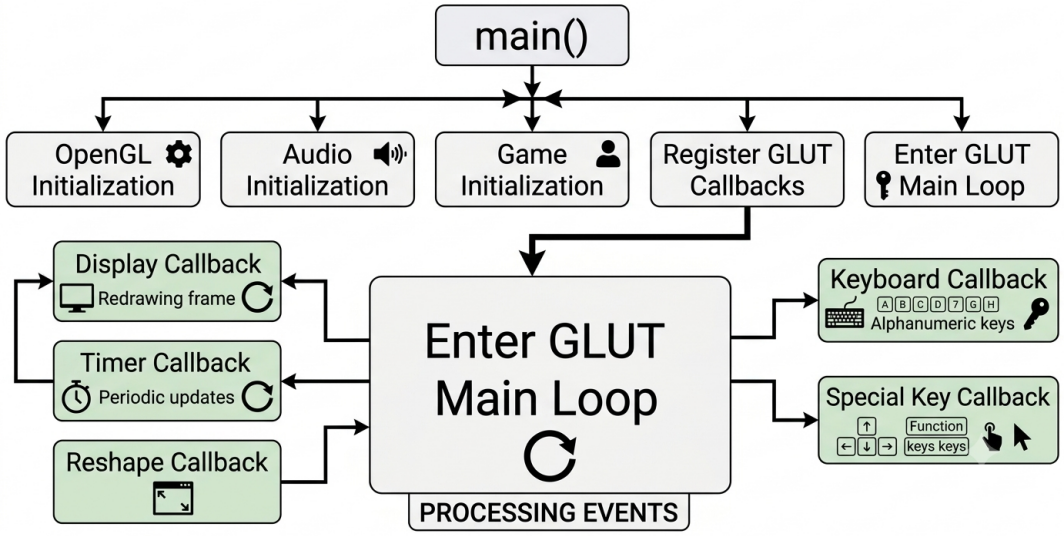

Figure 2.1: Application Initialization

The system continuously updates and renders the game at approximately 60 frames per second using a timer-based animation loop.

2.2 Game State Management

The game operates using a finite state machine (FSM) implemented through the `GameState` enumeration.

```
enum GameState
{
HOME,
PLAYING,
GAME_OVER
};
```

The state transitions occur as follows:

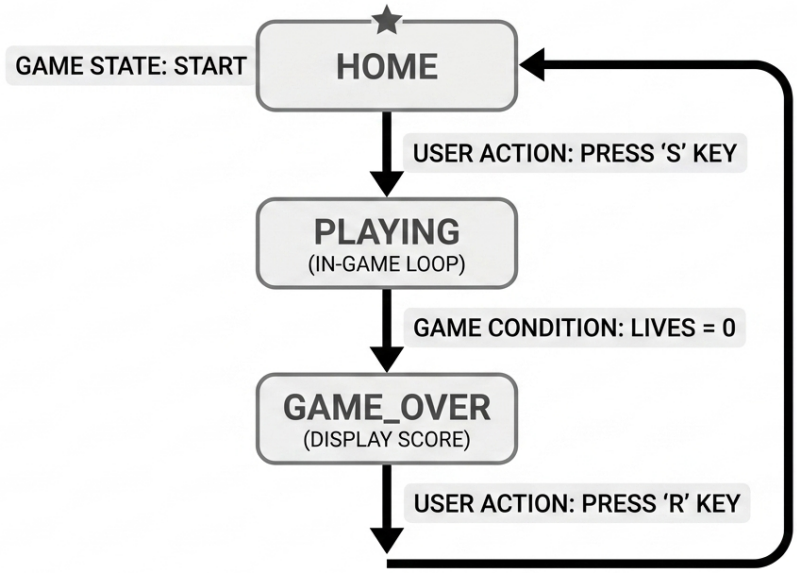

Figure 2.2: Game State Transition

This approach simplifies gameplay management by ensuring that only the logic associated with the active state is executed.

2.3 OpenGL Rendering Pipeline

The rendering process of Egg Drop Saga follows a structured pipeline that is executed repeatedly during gameplay.

Stage 1: Initialization

During program startup, the following operations are performed:

- GLUT initialization using `glutInit()`.
- Creation of the game window.

- Activation of RGB color mode.
- Activation of double buffering.
- Configuration of alpha blending.
- Setup of the orthographic projection.
- Initialization of game resources.
- Loading of audio files.

Stage 2: Input Processing

Player input is captured through GLUT callback functions.

- `keyboard()` handles standard keyboard input.
- `specialKeys()` handles arrow key input.

The input is forwarded to:

```
game.handleInput()
game.handleSpecialInput()
```

These functions update the bucket position, game state, pause state, and audio settings.

Stage 3: Update Cycle

The timer callback executes every 16 milliseconds.

```
glutTimerFunc(16, timer, 0);
```

This produces approximately:

$FPS = \frac{1000}{16} \approx 60$

During each update cycle:

- Background animation is updated.
- Home screen animation is updated.
- Eggs are spawned if necessary.
- Egg positions are updated.
- Collision detection is performed.
- Score and life counters are updated.

Stage 4: Rendering

The display callback invokes:

```
game.render();
```

The rendering order is:

1. Clear frame buffer.
2. Draw background.
3. Draw chickens.
4. Draw eggs.
5. Draw bucket.
6. Draw score text.
7. Draw life indicators.
8. Draw state-specific screens.

Stage 5: Buffer Swapping

After rendering completes:

```
glutSwapBuffers();
```

The back buffer becomes visible while the next frame is rendered in the hidden buffer.
This double-buffering technique eliminates screen flickering and ensures smooth animation.

2.4 Coordinate System Design

Egg Drop Saga uses a two-dimensional orthographic coordinate system configured through:

```
gluOrtho2D(0, GAME_WIDTH, 0, GAME_HEIGHT);
```

The coordinate system is defined as follows:

- Origin (0,0) is located at the bottom-left corner.
- Positive X values extend toward the right.
- Positive Y values extend upward.
- All game objects are positioned using world coordinates.

Examples of object positioning:

- Bucket is initialized near the bottom of the screen.
- Chickens are positioned along the upper wire.
- Eggs spawn below randomly selected chickens.
- UI elements are positioned using fixed coordinates.

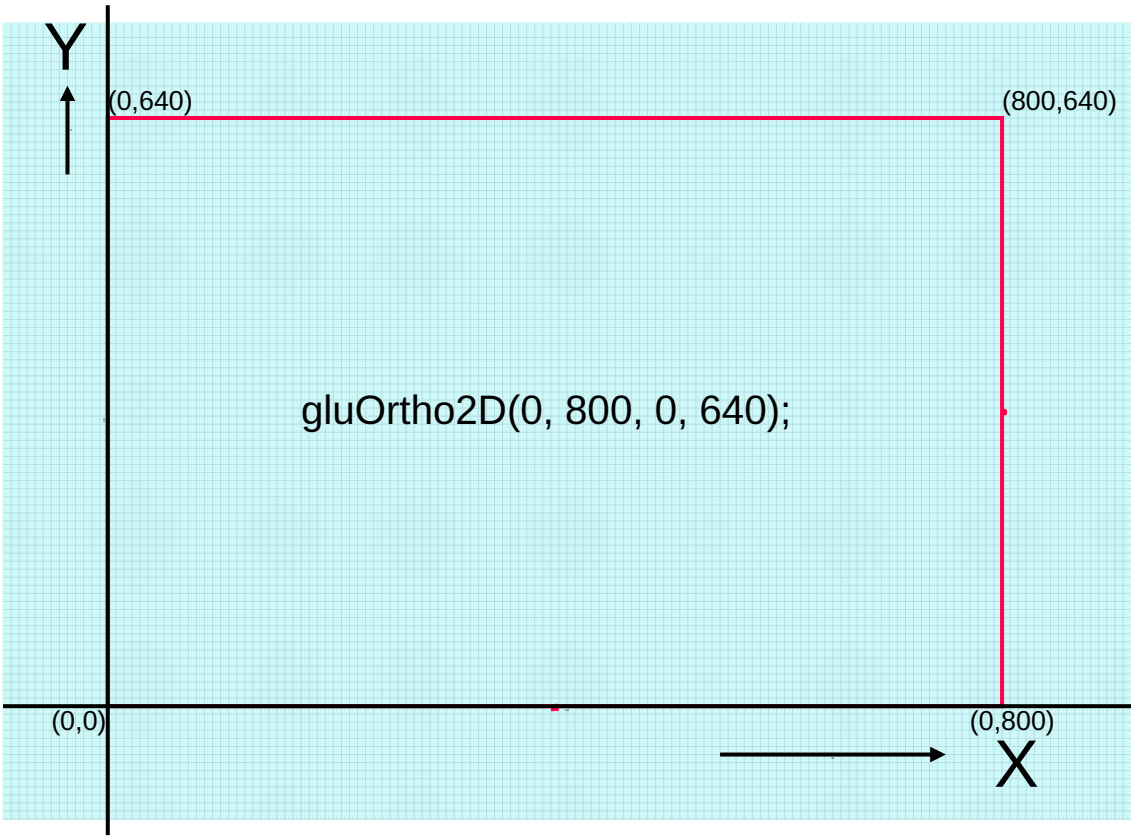

Figure 2.3: orthographics Graph of the Game Screen

2.5 Component Description

The Egg Drop Saga game is composed of several independent modules. Each module is responsible for a specific functionality and collectively contributes to the complete gameplay experience.

Game Module

The `Game` class acts as the central controller of the application.

Responsibilities include:

- Managing game states (Home, Playing, Game Over)
- Updating game objects
- Rendering all scene components
- Handling collision detection
- Maintaining score and life counters
- Coordinating audio playback

The Game class serves as the communication bridge between all other modules.

Chicken Module

The Chicken module represents the chickens positioned near the top portion of the game screen.

Main responsibilities:

- Maintaining chicken positions
- Rendering chicken graphics
- Providing egg spawning locations

Five chickens are created during game initialization and distributed evenly along the upper wire.

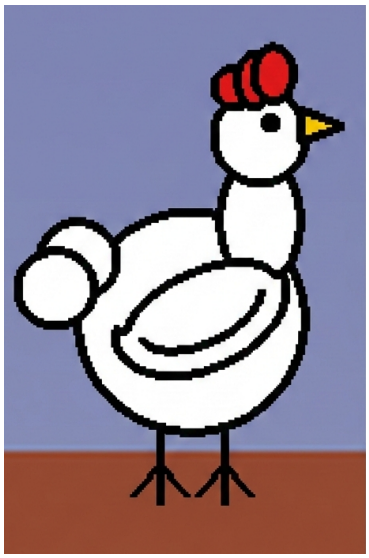

Figure 2.4: In Game Chicken

The chicken model is constructed using primitive OpenGL shapes including:

- Circular body
- Circular head
- Elliptical neck
- Polygon wing
- Tail feathers
- Comb
- Beak
- Legs

Transformation operations such as translation and rotation are applied to position body parts correctly.

Egg Module

The Egg module controls the falling eggs and their associated animations.
Responsibilities include:

- Egg generation
- Vertical movement
- Ground collision handling
- Squash animation
- Broken egg rendering
- Shadow generation

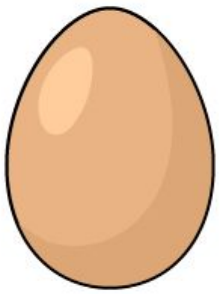

Figure 2.5: In Game Egg

The egg transitions through three states:

1. Falling State
2. Squashing State
3. Broken State

This creates a more realistic visual effect compared to immediate disappearance after ground impact.

Bucket Module

The bucket serves as the player-controlled object.
Responsibilities include:

- Horizontal movement
- Collision detection
- Catching eggs
- Boundary restriction

Figure 2.6: In Game Bucket

The bucket is drawn using:

- Semi-elliptical basket body
- Gradient shading
- Curved outlines
- Decorative weave patterns
- Ground shadow

Movement is restricted within the game boundaries to prevent the bucket from leaving the visible play area.

AudioManager Module

The AudioManager module handles all sound effects and music.
Responsibilities include:

- Loading audio files
- Playing background music
- Playing sound effects
- Managing audio volume

Separate audio tracks are used for:

- Home Screen Music
- Gameplay Music
- Game Over Music
- Egg Catch Effect
- Egg Break Effect
- Chicken Sound Effect

Background Module

The Background module draws the complete game environment.
Responsibilities include:

- Sky rendering
- Ground rendering
- Environmental decorations
- Static scenery

Figure 2.7: Game Background

This layer is rendered before all gameplay objects to create proper depth ordering.

2.6 Transformation and Animation Logic

The project uses several geometric transformations to create movement and animation effects.

Translation

Translation is the most frequently used transformation in the project.

The general translation equation is:
$P'(x, y) = (x + t_x, ; y + t_y)$
where:

- $P(x, y)$ is the original point.
- $P'(x, y)$ is the translated point.
- t_x and t_y represent displacement values.

Chicken Translation Each chicken is positioned using:

```
glTranslatef(x, y, 0);
```

This moves the chicken model from the origin to its assigned position on the wire.

Egg Translation The egg falls downward by continuously decreasing its y-coordinate.
$y_{new} = y_{old} - speed$
As the game score increases, egg speed also increases, increasing game difficulty.

Bucket Translation The bucket moves horizontally according to keyboard input.
$x_{new} = x_{old} \pm speed$
where:
$speed = 50$
Boundary checks ensure that the bucket remains within the visible game area.

Scaling

Scaling is used during the egg squash animation.

The transformation is implemented as:

```
glScalef(scaleX, scaleY, 1.0f);
```

When the egg hits the ground:

- Vertical scale decreases.
- Horizontal scale increases.

This creates a realistic squash-and-stretch effect commonly used in animation principles.

2.7 Collision Detection Mathematics

Collision detection is implemented between the bucket and the falling egg.

The collision condition is:

$Egg_X > Bucket_X$

and

$Egg_X < Bucket_X + Bucket_{Width}$

and

$Egg_Y - Radius < Bucket_Y + Bucket_{Height}$

If all conditions are satisfied simultaneously, the egg is considered successfully caught.

The actual implementation is:

```
return (egg.x > x &&
egg.x < x + width &&
egg.y - egg.radius < y + height);
```

When a collision occurs:

- Catch sound effect is played.
- Score increases by one.
- Current egg is removed.
- A new egg is generated.

2.8 Object Lifecycle

The lifecycle of an egg is illustrated below:

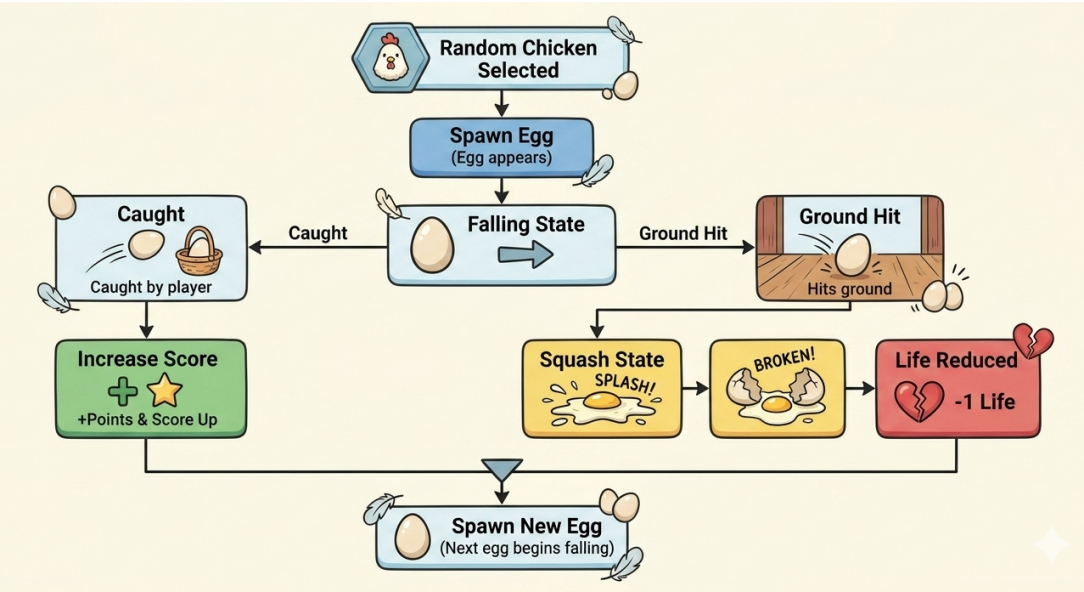

Figure 2.8: Egg Life Cycle

This lifecycle governs the complete gameplay mechanics and ensures continuous interaction between the player and falling objects.

2.9 OpenGL Drawing Primitives Used

The graphical objects in Egg Drop Saga are constructed entirely using OpenGL's immediate mode rendering primitives. Complex game objects such as chickens, eggs, buckets, shadows, and user interface elements are created by combining multiple geometric primitives.

The project demonstrates practical applications of fundamental computer graphics concepts including polygon modeling, curve approximation, transformations, shading, and animation.

GL_TRIANGLE_FAN

The `GL_TRIANGLE_FAN` primitive is the most frequently used primitive in the project.

It constructs a series of connected triangles sharing a common center vertex, making it ideal for creating circular and curved shapes.

Applications in the Project:

- Chicken body
- Chicken head
- Chicken tail feathers
- Chicken comb
- Egg body
- Egg highlights
- Egg shadow
- Broken egg splash
- Bucket body
- Heart icons (life indicators)

Example implementation:

```
glBegin(GL_TRIANGLE_FAN);
glVertex2f(centerX, centerY);

for(int i=0; i<=60; i++)
{
float angle = 2*PI*i/60;
glVertex2f(
centerX + radius*cos(angle),
centerY + radius*sin(angle)
);
}
glEnd();
```

This technique approximates circles and ellipses by connecting multiple vertices around a central point.

GL_POLYGON

The GL_POLYGON primitive is used to create filled irregular shapes composed of multiple connected vertices.

Applications in the Project:

- Chicken wing geometry

Example:

```
glBegin(GL_POLYGON);
...
glEnd();
```

This primitive allows the creation of more organic shapes that cannot be represented using simple rectangles or circles.

GL_TRIANGLES

The GL_TRIANGLES primitive is used to create individual triangular shapes.

Applications in the Project:

- Chicken beak

Example:

```
glBegin(GL_TRIANGLES);
glVertex2f(...);
glVertex2f(...);
glVertex2f(...);
glEnd();
```

The triangular beak provides a simple yet effective representation of the chicken's facial features.

GL_QUADS

The GL_QUADS primitive is used to create rectangular surfaces using four vertices.

Applications in the Project:

- Ground platform
- Grass layer
- Basket rim
- User interface panels

Example:

```
glBegin(GL_QUADS);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glVertex2f(x3,y3);
glVertex2f(x4,y4);
glEnd();
```

Quadrilaterals are efficient for constructing flat surfaces and environmental elements.

GL_LINES

The GL_LINES primitive is used for rendering independent line segments.

Applications in the Project:

- Chicken legs
- Chicken toes
- Basket weaving pattern

Example:

```
glBegin(GL_LINES);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();
```

These line segments provide additional visual details without significantly increasing rendering complexity.

GL_LINE_LOOP

The GL_LINE_LOOP primitive creates a closed outline by automatically connecting the last vertex back to the first vertex.

Applications in the Project:

- Egg outline
- Chicken body outline
- Chicken head outline
- Chicken comb outline
- Chicken beak outline
- Tail feather outlines
- Basket rim outline
- Broken egg components

Example:

```
glBegin(GL_LINE_LOOP);
...
glEnd();
```

The use of outlines improves object visibility and visual separation between adjacent shapes.

GL_LINE_STRIP

The `GL_LINE_STRIP` primitive draws a continuous connected sequence of lines.

Applications in the Project:

- Basket curved outline
- Wing feather details

Example:

```
glBegin(GL_LINE_STRIP);
...
glEnd();
```

This primitive is useful for representing curved boundaries and decorative details.

2.10 Graphics Techniques Implemented

In addition to primitive-based modeling, several graphics techniques were incorporated to improve the visual quality of the game.

Gradient Shading

Gradient shading is used extensively throughout the project.

Examples include:

- Egg surface shading
- Bucket body shading
- Heart icons
- Egg yolk rendering

Different vertex colors are assigned to neighboring vertices, allowing OpenGL to interpolate colors smoothly across the surface.

Alpha Blending

Transparency effects are implemented using OpenGL blending.

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA);
```

Applications include:

- Egg shadows
- Bucket shadows
- Egg highlights

This technique creates smoother and more realistic visual effects.

Shadow Rendering

Dynamic shadows are generated beneath falling eggs and the bucket.

The shadow size changes according to the object's height above the ground.

As the egg falls:

- Shadow becomes smaller.
- Shadow becomes darker.

This creates an illusion of depth in the two-dimensional environment.

Squash and Stretch Animation

The project implements the classical animation principle of squash and stretch.

When an egg reaches the ground:

- Vertical scale decreases.
- Horizontal scale increases.

This transformation produces a more natural and visually appealing impact animation before the egg breaks.

Double Buffering

To eliminate rendering flicker, the project uses OpenGL double buffering.

```
glutInitDisplayMode(
GLUT_DOUBLE | GLUT_RGB
);
```

After each frame is rendered:

```
glutSwapBuffers();
```

The back buffer becomes visible while rendering continues in the hidden buffer.

This ensures smooth animation and stable frame presentation.

This chapter presented the complete system design of Egg Drop Saga. The overall architecture, component interactions, rendering pipeline, coordinate system, object lifecycle, animation techniques, collision detection algorithms, and OpenGL primitives were discussed in detail. The implementation demonstrates practical applications of fundamental computer graphics concepts including geometric modeling, transformations, shading, animation, event-driven programming, and real-time rendering using OpenGL and GLUT.

Chapter 3 Implementation

3.1 Implementation

This chapter describes the practical implementation of the Egg Drop Saga game using C++, OpenGL, GLUT, and SDL2_mixer. It explains the development environment, implementation process, integration of modules, and important code segments used to create the final application.

3.2 Tools and Development Environment

The development of Egg Drop Saga required several software tools and libraries, which are listed below:

Tool/Library	**Purpose**
C++	Primary programming language used for game development.
OpenGL	Graphics rendering library used for drawing and displaying visual objects.
GLUT (OpenGL Utility Toolkit)	Provides window management, event handling, keyboard input, and rendering callbacks.
SDL2_mixer	Used for audio playback including background music and sound effects.
Code::Blocks	Integrated Development Environment (IDE) used for coding and debugging.
MinGW GCC Compiler	Compiles and links the C++ source code.
Windows + Linux Operating System	Development and testing platform.
GitHub	Version control and project collaboration.

Table 3.1: Tools and Libraries

The graphics subsystem was implemented using OpenGL's immediate mode rendering functions. GLUT was used to manage windows, user input, and event callbacks, while SDL2_mixer provided support for background music and sound effects.

3.3 Project Structure

The Egg Drop Saga project follows a modular object-oriented architecture where each major game component is implemented as an independent class. This design improves code maintainability, readability, and scalability by separating rendering, game logic, audio management, and user interface functionalities into distinct modules.

The overall project directory structure is shown below:

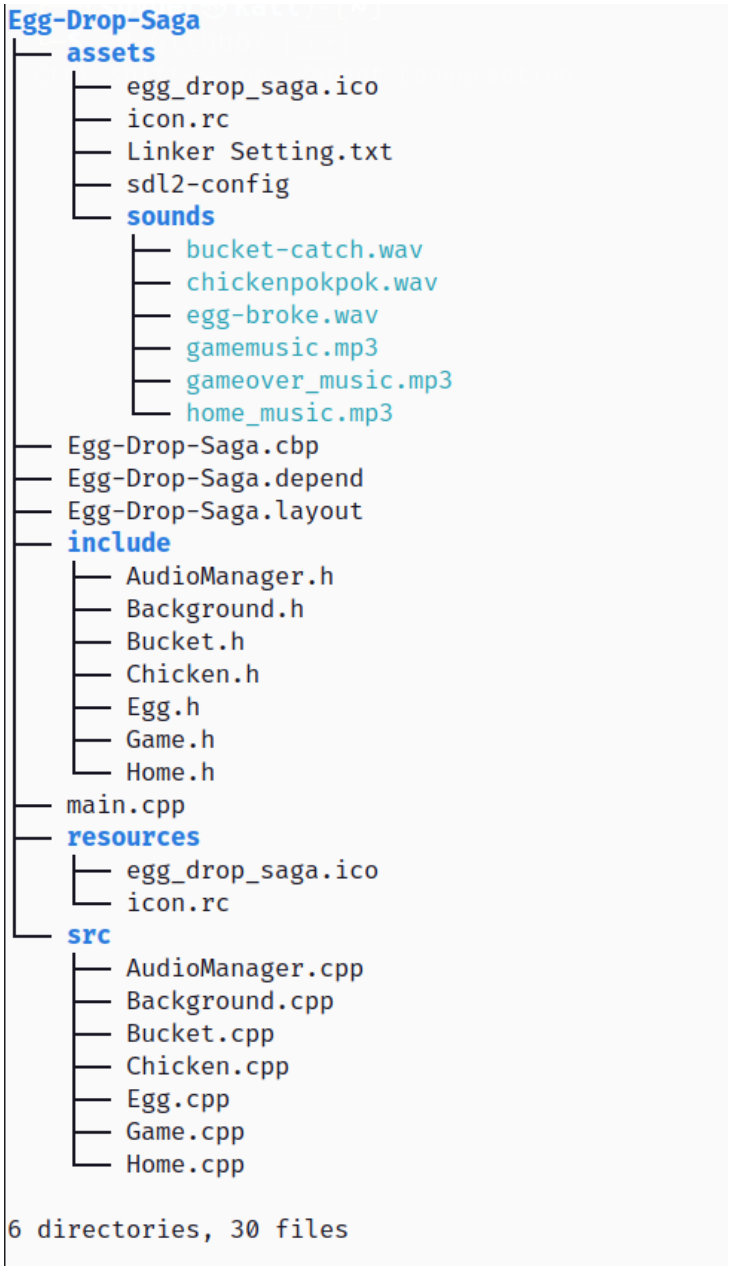

Figure 3.1: Project File Structure

The purpose of each major component is described below:

Module	Responsibility
main.cpp	Entry point of the application. Initializes OpenGL, creates the game window, registers callback functions, and starts the GLUT event loop.
Game Class	Controls the overall game state, score system, life management, collision detection, rendering coordination, and gameplay flow.
Chicken Class	Creates and animates the chickens that move horizontally and drop eggs.
Egg Class	Manages egg creation, falling animation, collision checking, and respawning logic.
Bucket Class	Represents the player-controlled bucket used to catch falling eggs.
Background Class	Draws the complete game environment including sky, ground, scenery, decorative objects, and visual layers.
Home Class	Handles the home screen, menu interface, instructions, and navigation before gameplay begins.
AudioManager Class	Controls background music, sound effects, audio loading, playback, and stopping mechanisms using SDL2_mixer.
Assets Directory	Stores sound effects, music files, application icons, and additional project resources.

Table 3.2: Project Modules and Responsibilities

The modular architecture ensures that graphical rendering, user interaction, animation, collision detection, and audio processing remain logically separated, making the project easier to understand, debug, and extend in future versions.

3.4 Implementation Process

The game was developed incrementally following several implementation stages.

Stage 1: Window Creation

The first stage involved creating the OpenGL window and configuring the rendering environment.

Main tasks:

- Initializing GLUT.
- Creating the application window.
- Configuring double buffering.
- Setting the orthographic projection.
- Enabling alpha blending.

Stage 2: Environment Construction

The background scene was developed first to establish the visual environment.

Main tasks:

- Sky rendering.
- Ground rendering.
- Decorative scenery.
- Environmental animation.

Stage 3: Object Modeling

The core game objects were created using OpenGL primitives.

Implemented objects:

- Chicken
- Egg
- Bucket
- Heart Indicator

Each object was constructed using combinations of circles, polygons, quadrilaterals, and line segments.

Stage 4: Gameplay Mechanics

Gameplay functionality was then integrated.

Features implemented:

- Egg spawning system
- Bucket movement
- Collision detection
- Score calculation
- Life management
- Game over conditions

Stage 5: Audio Integration

Audio support was added using SDL2_mixer.

Implemented audio features:

- Home screen music
- Gameplay music
- Game over music
- Egg catch sound effect
- Egg break sound effect
- Chicken sound effect

3.4.1 Audio Environment Configuration

Before implementing the game, several development environment configurations were required to properly integrate OpenGL, GLUT, SDL2, and SDL2_mixer within the Code::Blocks IDE on Windows.

SDL2 Include Directory Configuration

The SDL2 header directory was added to the compiler search paths.

- Search Directory:

```
C:\Libraries\SDL2\include\SDL2
```

This allows the compiler to locate SDL2 and SDL2_mixer header files during compilation.

Linker Settings

The following linker options were configured in Code::Blocks:

```
-static
-static-libgcc
-static-libstdc++
-lmingw32
-lSDL2main
-lSDL2
-lSDL2_mixer
```

If static linking is not required, the `-static` related options can be removed.

Execute Directory Configuration

The project execution directory was configured as:

`Project Directory`

This ensures that game assets such as sound files and resources are loaded correctly during runtime.

Linked Libraries

The following libraries were linked with the project:

- mingw32
- SDL2_main
- SDL2
- SDL2_mixer

These libraries provide multimedia functionality including audio playback and SDL initialization.

Cross-Platform SDL Header Configuration

To prevent conflicts between SDL and the Windows entry point, the following preprocessor directive was added in the audio manager implementation:

```
#ifdef _WIN32
#define SDL_MAIN_HANDLED
#endif
```

This allows the project to use the standard C++ `main()` function while maintaining compatibility with SDL2.

Application Icon Integration

A custom application icon was integrated into the Windows version of the game using the Win32 API.

The following code was used in `main.cpp`:

```
HWND hwnd = GetActiveWindow();
HICON hIcon = (HICON)LoadImage(
    GetModuleHandle(NULL),
    MAKEINTRESOURCE(IDI_ICON1),
    IMAGE_ICON,
    32, 32,
    LR_DEFAULTCOLOR
);

SendMessage(hwnd, WM_SETICON, ICON_BIG, (LPARAM)hIcon);
SendMessage(hwnd, WM_SETICON, ICON_SMALL, (LPARAM)hIcon);
```

This configuration assigns the custom `egg_drop_saga.ico` icon to both the application window and taskbar representation.

Resource File Configuration

The project uses a Windows resource script file (`icon.rc`) to register the application icon.

The resource file references:

```
egg_drop_saga.ico
```

This ensures that the compiled executable contains the custom application icon.

3.5 Game Object Implementation

Chicken Object

The Chicken class represents stationary chickens positioned near the top of the screen.

Implementation features:

- Circular body generation
- Head construction
- Wing modeling
- Tail generation
- Beak creation
- Leg rendering

The chicken is constructed using multiple OpenGL primitives and positioned using translation transformations.

Egg Object

The Egg class manages all falling eggs.

Key functionalities:

- Dynamic spawning
- Vertical movement
- Shadow rendering
- Squash animation
- Breaking animation
- State management

The falling motion is achieved by continuously updating the egg's vertical coordinate.

```
y -= speed;
```

Bucket Object

The Bucket class represents the player-controlled catching mechanism.
Implementation features:

- Horizontal movement
- Boundary restriction
- Collision handling
- Decorative basket rendering
- Dynamic shadow generation

The bucket position is updated through keyboard input.

```
x += speed;
x -= speed;
```

3.6 Game Loop Implementation

The game operates using GLUT's event-driven architecture.
The timer callback executes every 16 milliseconds.

```
glutTimerFunc(16, timer, 0);
```

This produces approximately 60 frames per second.
The update cycle performs the following operations:

1. Update background animation.
2. Spawn new eggs.
3. Update egg positions.
4. Perform collision detection.
5. Update score and lives.
6. Request screen redraw.

3.7 User Input Implementation

Keyboard controls are implemented through GLUT callback functions.
User input is processed through:

```
glutKeyboardFunc(keyboard);
glutSpecialFunc(specialKeys);
```

Key	Function
S	Start Game
A	Move Left
D	Move Right
Left Arrow	Move Left
Right Arrow	Move Right
Space	Pause / Resume
M	Toggle Music
F	Fullscreen Mode
ESC	Restore Window
Q	Quit Game
R	Return to Home Screen

Table 3.3: Game Controls

3.8 Collision Detection Implementation

Collision detection determines whether the bucket successfully catches a falling egg.

The implementation checks:

- Horizontal overlap.
- Vertical overlap.
- Egg radius.

Implementation:

```
return (egg.x > x &&
egg.x < x + width &&
egg.y - egg.radius < y + height);
```

When a collision occurs:

- Score increases.
- Catch sound effect is played.
- Egg is removed.
- A new egg is generated.

3.9 Audio System Implementation

Audio functionality is managed through the AudioManager module.

The audio system performs:

- Initialization of SDL2_mixer.
- Loading audio resources.

- Music playback control.
- Sound effect playback.
- Volume management.

Audio playback changes dynamically according to the current game state.

3.10 Important Code Snippets

Game Initialization

The following code initializes the game state.

```
score = 0;
lives = 3;
state = HOME;
bucket = Bucket(
GAME_WIDTH / 2 - 60,
GAME_HEIGHT * 0.03f
);
```

This establishes the initial game conditions.

Egg Spawning

Eggs are generated from randomly selected chickens.

```
int index = rand() % chickens.size();
float x = chickens[index].getX();
float y = chickens[index].getY() - 40;
```

This creates variability in gameplay.

Double Buffering

Smooth rendering is achieved through:

```
glutSwapBuffers();
```

which exchanges the front and back buffers after each frame.

This chapter described the implementation details of Egg Drop Saga. The development environment, project structure, object creation process, game loop, collision detection system, audio integration, and user interaction mechanisms were discussed. The modular implementation approach allowed individual components to be developed independently and integrated into a complete interactive OpenGL game.

Chapter 4 Results and Output

This chapter presents the final output of the Egg Drop Saga game and evaluates the successful implementation of the designed features. The project was tested under various gameplay conditions to verify the functionality of object rendering, animation, collision detection, user interaction, and audio integration. The results demonstrate that the game operates smoothly while maintaining consistent visual quality and responsive controls.

4.1 Home Screen

The home screen serves as the entry point of the game. It provides the title of the game, gameplay instructions, and available controls. Background music is played to create an engaging atmosphere before gameplay begins.

The player can start the game by pressing the `S` key or exit the application using the `Q` key.

Figure 4.1: Home Screen of Egg Drop Saga

The home screen successfully provides a user-friendly interface and introduces the gameplay mechanics before entering the main game scene.

4.2 Main Gameplay Scene

Figure 4.2 illustrates the primary gameplay environment. Five chickens are positioned near the top of the screen while eggs are generated randomly beneath them. The player controls a basket located near the ground and attempts to catch the falling eggs.

The gameplay scene includes:

- Animated background environment
- Five chicken models
- Dynamic egg spawning system
- Player-controlled basket
- Score display
- Life indicator using heart icons

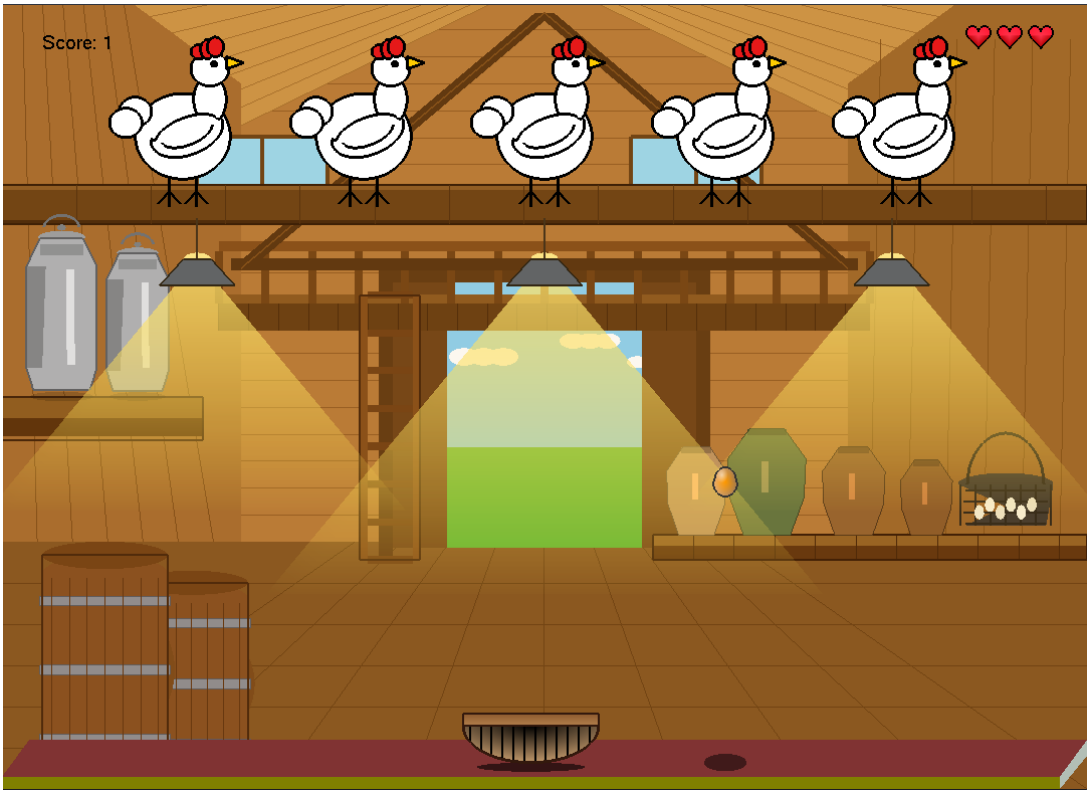

Figure 4.2: Main Gameplay Scene

The scene demonstrates successful integration of all graphical components and gameplay elements.

4.3 Egg Falling Animation

One of the primary gameplay mechanics involves eggs falling from randomly selected chickens. Each egg is rendered using curved geometry, gradient shading, and a dynamic shadow to improve visual realism.

The shadow changes size and transparency according to the egg's height above the ground, creating a depth perception effect within the two-dimensional environment.

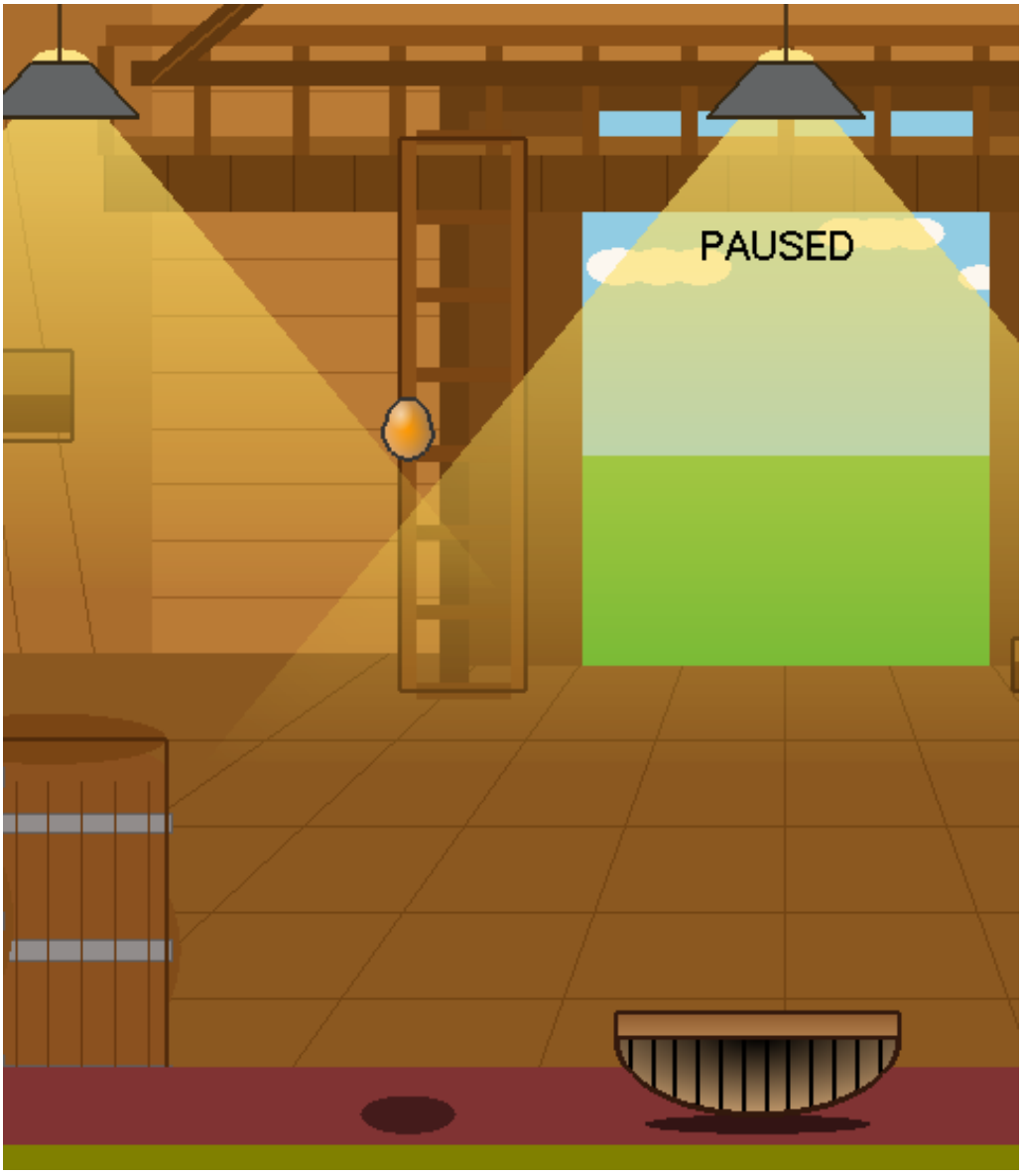

Figure 4.3: Falling Egg Animation

The animation operates smoothly at approximately 60 frames per second, providing a responsive gameplay experience.

4.4 Egg Catching Mechanism

When the basket successfully intercepts a falling egg, the collision detection system is triggered. The egg is removed from the scene, the player's score increases, and a catch sound effect is played.

This functionality demonstrates the successful implementation of real-time collision detection and score management.

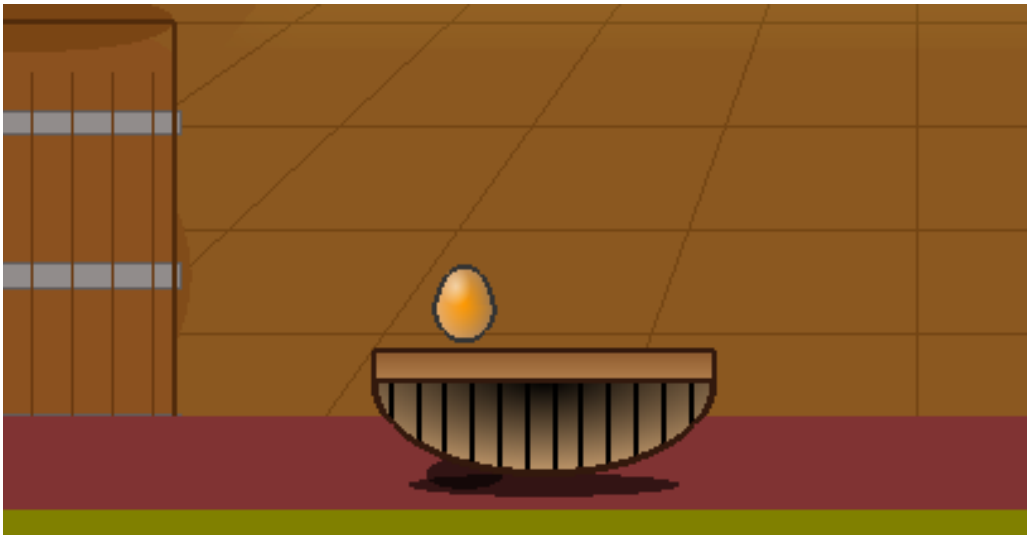

Figure 4.4: Successful Egg Catching

The collision system accurately detects interactions between the egg and basket without noticeable delay.

4.5 Egg Breaking Animation

If an egg reaches the ground without being caught, a squash-and-stretch animation is executed before displaying the broken egg state.

The animation sequence consists of:

1. Egg reaches the ground.
2. Vertical compression occurs.
3. Horizontal expansion occurs.
4. Egg breaking effect is displayed.
5. Life count decreases.

This feature demonstrates the application of animation principles commonly used in computer graphics and game development.

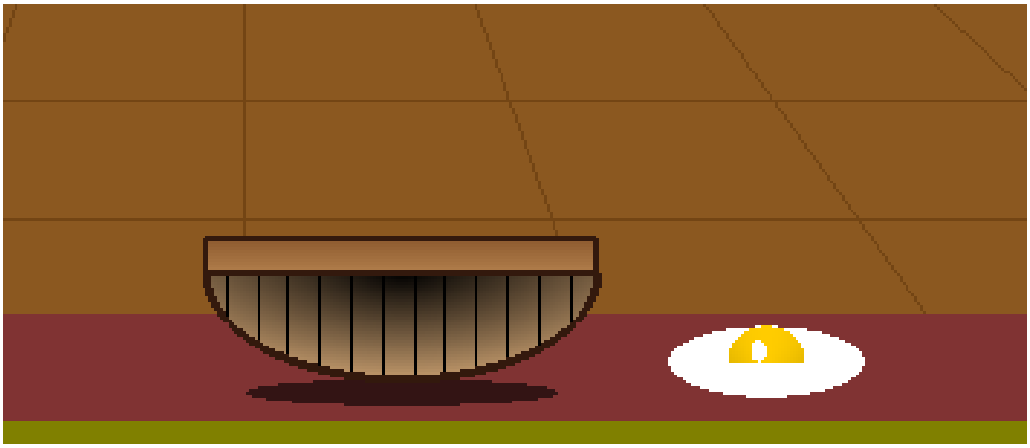

Figure 4.5: Egg Breaking Animation

4.6 Pause and Resume System

The game includes a pause functionality activated through the `Space` key.
During the paused state:

- Object updates are suspended.
- Egg movement stops.
- Score and life values remain unchanged.
- Rendering continues normally.

Upon resuming, a temporary `RESUMED` message is displayed to inform the player that gameplay has continued.

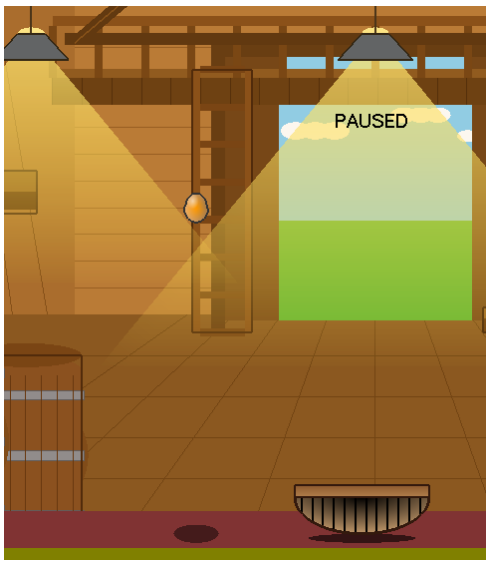

Figure 4.6: Pause State

The pause system improves usability and allows players to temporarily suspend gameplay when necessary.

4.7 Game Over Screen

When all available lives are exhausted, the game enters the game over state.

The game over screen displays:

- Game Over message
- Final score
- Restart instructions
- Exit option

Additionally, the background music is replaced with a dedicated game over soundtrack.

Figure 4.7: Game Over Screen

This screen successfully communicates the end of the game session and provides options for replaying or exiting the application.

4.8 Audio System Output

The SDL2_mixer audio subsystem was successfully integrated with the OpenGL application.

Different sound effects and music tracks are triggered based on gameplay events.

Audio Type	**Purpose**
Home Music	Played on home screen
Gameplay Music	Played during gameplay
Game Over Music	Played after losing all lives
Chicken Sound	Played when an egg is spawned
Catch Sound	Played when an egg is caught
Break Sound	Played when an egg breaks

Table 4.1: Audio Features Implemented

The audio feedback significantly improves user engagement and gameplay immersion.

4.9 Performance Evaluation

The game was tested on a Windows-based system using OpenGL and GLUT.

Performance observations include:

- Stable rendering performance.
- Smooth object animation.
- Responsive keyboard controls.
- Accurate collision detection.
- Proper audio synchronization.
- No visible rendering flicker.

The timer callback was configured as:

```
glutTimerFunc(16, timer, 0);
```

which corresponds to approximately 60 frames per second.

Double buffering was employed using:

```
GLUT_DOUBLE
```

to eliminate screen tearing and provide smooth visual updates.

4.10 Implemented Features Summary

This table summarizes the major features successfully implemented in the project.

Feature	Status
Home Screen Interface	Completed
Background Rendering	Completed
Chicken Modeling	Completed
Egg Rendering	Completed
Bucket Modeling	Completed
Egg Falling Animation	Completed
Squash-and-Stretch Animation	Completed
Collision Detection	Completed
Score Management	Completed
Life Management	Completed
Pause and Resume System	Completed
Game Over System	Completed
Audio Integration	Completed
Fullscreen Support	Completed
Responsive Controls	Completed

Table 4.2: Implemented Features

The results demonstrate that Egg Drop Saga successfully achieves its design objectives. All major gameplay components, including rendering, animation, collision detection, user interaction, audio playback, and game state management, function as intended. The final application provides a complete interactive gaming experience while showcasing fundamental computer graphics concepts such as geometric modeling, transformations, animation, shading, and real-time rendering using OpenGL and GLUT.

Chapter 5 Discussion

This chapter discusses the challenges encountered during the development of Egg Drop Saga, the solutions adopted to overcome those challenges, and the knowledge gained throughout the implementation process. The project provided practical experience in OpenGL programming, object modeling, animation, user interaction, and real-time game development.

5.1 Challenges Faced During Development

Although the final implementation appears straightforward from the user's perspective, several technical challenges were encountered during development.

Object Modeling Complexity

One of the primary challenges was designing visually appealing game objects using only OpenGL primitives. Since the project did not use external image assets for the main game objects, every object had to be constructed manually using geometric shapes.

The chicken model required multiple components including:

- Body
- Head
- Wing
- Tail
- Comb
- Beak
- Legs

Similarly, the basket and egg models required careful adjustment of dimensions and proportions to achieve a natural appearance.

Constructing these objects using only vertices and OpenGL primitives required extensive experimentation and refinement.

Maintaining Proper Aspect Ratio

Another challenge was preserving the game's visual appearance when the window size changed.

Different screen resolutions can stretch or distort graphical objects if the viewport and projection settings are not managed correctly.

Without proper aspect ratio handling:

- Eggs could appear wider or narrower.
- Chickens could become distorted.
- Gameplay coordinates could become inconsistent.

A custom reshape function was implemented to maintain the original game aspect ratio while adapting to different window dimensions.

Collision Detection Accuracy

Collision detection is one of the most important gameplay mechanics.

Initially, the basket occasionally failed to detect eggs correctly, especially near the basket edges.

Several adjustments were required to determine an appropriate collision boundary that balanced gameplay fairness and responsiveness.

The final implementation checks both horizontal overlap and vertical intersection between the egg and basket, resulting in consistent collision behavior.

Animation Timing

Creating smooth animations using GLUT's timer system required careful timing management.

The project includes several animations:

- Falling eggs
- Egg squash effect
- Egg breaking animation
- Dynamic shadows
- Background animations

Maintaining smooth transitions while preserving gameplay responsiveness required synchronization between the update and rendering systems.

Audio Integration

Integrating SDL2_mixer with an OpenGL GLUT application introduced additional complexity.

Challenges included:

- Library configuration
- Linker setup
- Sound loading
- Audio state management
- Music switching between game states

Careful initialization and cleanup procedures were implemented to ensure reliable audio playback.

Game State Management

The project contains three major states:

- Home Screen
- Playing State
- Game Over State

Managing transitions between these states while maintaining score, lives, music, and object states required additional planning and testing.

Proper state management was essential for creating a smooth user experience.

5.2 Solutions Adopted

Several solutions were implemented to address the challenges encountered during development.

Modular Programming Approach

The project was divided into separate modules:

- Game Module
- Chicken Module
- Egg Module
- Bucket Module
- Background Module

- Home Module
- Audio Module

This modular architecture simplified debugging, testing, and future development.

Each module was responsible for a specific functionality, reducing code complexity and improving maintainability.

Use of OpenGL Transformations

Transformations such as translation, rotation, and scaling were used extensively.

Examples include:

- Positioning chickens across the screen.
- Rotating wing components.
- Scaling the egg during squash animation.
- Translating objects into world coordinates.

Using transformation matrices reduced the need for recalculating individual vertices.

Viewport and Projection Adjustment

To solve aspect ratio issues, the viewport dimensions were calculated dynamically inside the reshape callback function.

This ensured that:

- Objects retained their proportions.
- Gameplay coordinates remained consistent.
- Visual quality remained stable across different resolutions.

Event-Driven Architecture

GLUT's callback-based architecture simplified user input handling and rendering updates.

The project uses dedicated callback functions for:

- Display rendering
- Keyboard input
- Special key input
- Window resizing
- Timed updates

This structure improved responsiveness and separated application logic into manageable components.

5.3 Computer Graphics Concepts Applied

The project successfully applied several important computer graphics concepts studied during the course.

Geometric Modeling

Every game object was modeled using geometric primitives such as:

- Triangles
- Quadrilaterals
- Polygons
- Circles
- Ellipses

Complex objects were formed by combining multiple primitive shapes.

Coordinate Systems

The project uses a two-dimensional orthographic coordinate system.

The OpenGL projection setup maps screen coordinates into the virtual game world. This concept was essential for:

- Object positioning
- Movement calculations
- Collision detection
- User interface rendering

Transformations

The project demonstrates practical use of geometric transformations.

Translation was used to position objects, rotation was used to orient components such as wings, and scaling was applied to generate squash-and-stretch animations.

These transformations allowed efficient manipulation of graphical objects without re-defining their geometry.

Animation Principles

Animation plays a central role in the project.

Examples include:

- Falling motion of eggs
- Dynamic shadow updates

- Squash-and-stretch effects
- User interface transitions

These animations demonstrate the importance of time-based updates in real-time graphics applications.

Color Interpolation and Shading

Gradient shading techniques were implemented using vertex color interpolation.

Examples include:

- Egg surface shading
- Basket body shading
- Heart indicators
- Egg yolk rendering

This produced smoother and more visually appealing objects compared to flat coloring.

Alpha Blending

Transparency effects were achieved using OpenGL blending functions.

Alpha blending was used for:

- Shadows
- Highlights
- Visual effects

These effects improved realism and visual depth within the game scene.

5.4 Lessons Learned

The development of Egg Drop Saga provided valuable practical experience beyond theoretical classroom concepts.

Through this project, the following skills were strengthened:

- OpenGL rendering techniques
- Object-oriented programming in C++
- Event-driven programming
- Game loop design
- Animation implementation
- Collision detection algorithms

- Audio integration using SDL2_mixer
- Modular software development
- Debugging and performance optimization

The project also demonstrated how multiple computer graphics concepts can be integrated into a complete interactive application.

5.5 Educational Value of the Project

Egg Drop Saga serves as a practical example of applying computer graphics principles to a real-world software project.

The project combines:

- Geometric modeling
- Transformations
- Rendering
- Animation
- User interaction
- Multimedia integration

As a result, it provides a comprehensive learning experience that reinforces both theoretical and practical aspects of Computer Graphics.

This chapter discussed the challenges encountered during development, the solutions implemented to address them, and the computer graphics concepts applied throughout the project. The development process provided valuable experience in OpenGL programming, animation, object modeling, collision detection, and multimedia integration. The successful completion of the project demonstrates the practical application of fundamental computer graphics principles in the creation of an interactive game.

Chapter 6 Conclusion

The development of **Egg Drop Saga** successfully demonstrated the practical application of fundamental Computer Graphics concepts using OpenGL and GLUT. The project achieved its primary objective of creating an interactive two-dimensional game that combines object modeling, animation, user interaction, collision detection, and audio integration within a real-time rendering environment.

Throughout the development process, various graphical objects such as chickens, eggs, baskets, hearts, and environmental elements were designed using OpenGL geometric primitives. These individual components were combined to create a complete and visually engaging game scene. The project also incorporated animation techniques including falling motion, dynamic shadows, and squash-and-stretch effects, which enhanced realism and improved the overall user experience.

The implementation of collision detection enabled accurate interaction between the falling eggs and the player's basket, forming the core gameplay mechanic. In addition, the integration of SDL2_mixer provided background music and sound effects, significantly increasing player immersion. Features such as score tracking, life management, pause and resume functionality, game state transitions, fullscreen support, and custom application icons further contributed to the completeness of the final application.

From an educational perspective, this project provided valuable hands-on experience with OpenGL programming and reinforced several important topics studied in Computer Graphics, including coordinate systems, geometric modeling, transformations, viewport management, rendering pipelines, animation principles, color interpolation, alpha blending, and event-driven programming. The project also strengthened programming skills in C++, object-oriented design, debugging, modular software development, and multimedia integration.

Overall, Egg Drop Saga successfully fulfills the goals established during the project proposal stage and demonstrates how theoretical Computer Graphics concepts can be applied to create a functional and interactive software application. The completed project serves as a practical example of real-time graphics programming and provides a strong foundation for future exploration of advanced game development and computer graphics techniques.

6.1 Achievement of Objectives

The objectives defined at the beginning of the project were successfully achieved, as summarized below:

Objective	Status
Develop a complete 2D game using OpenGL and GLUT	Achieved
Implement geometric object modeling	Achieved
Apply transformations for positioning and animation	Achieved
Create real-time gameplay interaction	Achieved
Implement collision detection mechanisms	Achieved
Integrate sound effects and background music	Achieved
Maintain smooth rendering performance	Achieved
Demonstrate Computer Graphics concepts in practice	Achieved

Table 6.1: Project Objectives and Achievement Status

Chapter 7 Future Work

Although Egg Drop Saga successfully implements the core gameplay mechanics and demonstrates several important Computer Graphics concepts, there are numerous opportunities for future enhancement. Additional features and technical improvements could further improve gameplay experience, visual quality, and software architecture.

7.1 Enhanced Graphics and Visual Effects

Future versions of the game can incorporate more advanced graphical features to create a richer visual experience.

Possible improvements include:

- Particle effects for egg breaking animations.
- Dynamic lighting and shading techniques.
- Animated backgrounds with moving clouds and environmental elements.
- Improved shadow rendering using advanced blending techniques.
- Smooth sprite-based animations for chickens and other game objects.

These enhancements would increase realism and provide a more engaging visual presentation.

7.2 Multiple Difficulty Levels

The current implementation increases difficulty by gradually increasing egg falling speed. Future versions could introduce dedicated difficulty modes such as:

- Easy
- Medium
- Hard
- Expert

Each mode could modify:

- Egg falling speed

- Spawn frequency
- Number of lives
- Score multipliers

This would make the game more accessible to players with different skill levels.

7.3 Additional Gameplay Mechanics

Several new gameplay features can be added to increase variety and challenge.
Examples include:

- Golden eggs that provide bonus points.
- Rotten eggs that reduce score or lives.
- Power-ups that temporarily enlarge the basket.
- Slow-motion effects for easier egg catching.
- Combo scoring systems for consecutive catches.

These mechanics would make gameplay more dynamic and rewarding.

7.4 High Score System

The current version stores scores only during a single game session. A future enhancement could implement a persistent high-score system using file handling.
Features may include:

- Local leaderboard storage.
- Player name registration.
- Top score tracking.
- Score history management.

This would encourage replayability and competition among players.

7.5 Improved User Interface

The user interface can be expanded to provide a more professional gaming experience.
Potential additions include:

- Dedicated settings menu.
- Volume control options.
- Difficulty selection screen.

- Pause menu with navigation options.
- Animated game-over screen.

These features would improve usability and overall presentation.

7.6 Advanced Audio Features

While the current version includes background music and sound effects, future versions may incorporate:

- Multiple music tracks.
- Dynamic music transitions.
- Adjustable volume controls.
- Context-sensitive sound effects.
- Environmental audio effects.

Such enhancements would provide a more immersive gameplay experience.

7.7 Cross-Platform Deployment

The project is currently configured primarily for Windows environments.

Future development may include:

- Native Linux support.
- macOS compatibility.
- Cross-platform build automation.
- Platform-independent resource management.

This would allow the game to reach a wider range of users.

7.8 Migration to Modern OpenGL

The current implementation uses the OpenGL fixed-function pipeline, which is suitable for educational purposes and introductory graphics programming.

A future version could migrate to modern OpenGL techniques including:

- Vertex Buffer Objects (VBOs)
- Vertex Array Objects (VAOs)
- GLSL shaders
- GPU-based rendering
- Advanced texture mapping

This would improve rendering efficiency and provide experience with contemporary graphics development practices.

7.9 Mobile and Touchscreen Support

Future versions could be adapted for mobile platforms by introducing:

- Touch-based controls.
- Responsive user interfaces.
- Mobile-friendly graphics optimization.
- Android and iOS deployment support.

This would significantly increase accessibility and user reach.

Future Vision

The long-term vision for Egg Drop Saga is to transform it from a simple educational OpenGL project into a fully featured casual arcade game. By integrating advanced graphics, expanded gameplay mechanics, persistent player progression, and modern rendering techniques, the project could evolve into a more sophisticated and entertaining application while continuing to serve as an effective demonstration of Computer Graphics principles.

References

1. D. Hearn and M. P. Baker, *Computer Graphics with OpenGL*, 4th ed. Upper Saddle River, NJ, USA: Pearson Education, 2011.

2. M. Kilgard, "The OpenGL Utility Toolkit (GLUT) Programming Interface," Silicon Graphics Inc. **[Online]**.
Available: https://www.opengl.org/resources/libraries/glut/. [Accessed: Jun. 2026].

3. *OpenGL Official Documentation*. Khronos Group. **[Online]**.
Available: https://www.opengl.org/documentation/. [Accessed: Jun. 2026].

4. *Khronos OpenGL Reference Pages*. Khronos Group. **[Online]**.
Available: https://registry.khronos.org/OpenGL/. [Accessed: Jun. 2026].

5. *SDL2 Documentation*. Simple DirectMedia Layer (SDL). **[Online]**.
Available: https://wiki.libsdl.org/. [Accessed: Jun. 2026].

6. *SDL2_mixer Documentation*. Simple DirectMedia Layer (SDL). **[Online]**.
Available: https://wiki.libsdl.org/SDL2_mixer. [Accessed: Jun. 2026].

7. FreeGLUT Project Documentation. **[Online]**.
Available: https://freeglut.sourceforge.net/. [Accessed: Jun. 2026].

8. Microsoft Corporation, "Windows API Documentation," Microsoft Learn. **[Online]**.
Available: https://learn.microsoft.com/windows/win32/. [Accessed: Jun. 2026].

9. B. Stroustrup, *The C++ Programming Language*, 4th ed. Boston, MA, USA: Addison-Wesley Professional, 2013.

10. Code::Blocks Team, "Code::Blocks IDE Documentation." **[Online]**.
Available: https://www.codeblocks.org/. [Accessed: Jun. 2026].

11. Istiak Alam and Tanveer Ratul, Department of Computer Science and Engineering, Notre Dame University Bangladesh, Computer Graphics Lab Project Documentation, 2026.

    Github: Egg-Drop-Saga.
    Website: Download here.