VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JNANA SANGAMA", BELAGAVI– 590018

MINI PROJECT REPORT ON

"CAMERA SYSTEM USING OPENGL"

Submitted in partial fulfillment of Sixth Semester Computer Graphics and Image processing Laboratory (21CSL66) in Computer Science and Engineering

Submitted By,

SINCHANA A B : 4GH21CS044

Under the Guidance of

Dr.Vasantha Kumara M B.E, M.Tech, Ph.D.

Assistant Professor

Dept. of CSE

GEC, Hassan

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING,

GOVERNMENT ENGINEERING COLLEGE,

HASSAN-573201

GOVERNMENT ENGINEERING COLLEGE HASSAN – 573 201

Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

2023-24

CERTIFICATE

Certified that the Mini Project entitled **"CAMERA SYSTEM USING OPENGL"** carried out by **Ms. Sinchana A B (4GH21CS044)** a bonafide student of **Government Engineering College, Hassan** in partial fulfillment of **Sixth semester** in **Computer Graphics and Image processing Laboratory (21CSL66) Bachelor of Engineering** in **Computer Science and Engineering** of the **Visvesvaraya Technological University**, **Belagavi** during the year 2023-2024. It is certified that all correction / suggestions indicated during internal evaluation has been incorporated in the report. The Mini Project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the said Degree.

Project Guide	**Head of the Department**
Dr.Vasantha Kumara M BE, M.Tech, Ph.D. Assistant Professor Dept. of CSE GEC, Hassan	**Dr.Vani V G** B.E, M.Tech, Ph.D. Head of the Department Dept. of CSE GEC, Hassan

External Viva

Name and Signature of Examiner with Date:

1.

2.

DECLARATION

I, SINCHANA A B student of Sixth Semester B.E in GOVERNMENT ENGINEERING COLLEGE, Hassan bearing USN 4GH21CS044 ,hereby declare that the project entitled "CAMERA SYSTEM USING OPENGL " has been carried out by me under the supervision of our Guide, Dr. Vasantha Kumara M, B.E, M.Tech, Ph.D., Assistant Professor, Dept of CS&E, GEC Hassan, have submitted in partial fulfillment of the requirements for the award of the Sixth Semester of B.E in CS&E by the Visveswaraya Technological University, Belagavi during the academic year 2023-2024. This report has not been submitted to any other Organization/University for the award of degree or certificate.

- SINCHANA A B (4GH21CS044)

ACKNOWLEDGEMENT

I consider it a privilege to whole-heartedly express my gratitude and respect to each and every one who guided and helped us in the successful completion of this Project Report.

I, very thankful to the Principal **Dr. Girish D P**,B.E, MTech, PhD, for being kind enough to provide me an opportunity to work on a project in this institution.

I, also thankful to **Dr. Vani V G** B.E, MTech, HOD, Department of Computer Science, for her co-operation and encouragement at all moments of my approach.

I would greatly mention the enthusiastic influence provided by **Dr. Vasantha Kumara M,** B.E, M. Tech, Assistant Professor, Project Guide for his ideas and co-operation showed on me during my venture and making Project as a great success.

I would also like to thank our parents and well-wishers as well as our dear classmates for their guidance and their kind co-operation.

Finally, it is my pleasure and happiness to the friendly co-operation showed by all the staff members of Computer Science Department, GECH.

-SINCHANA A B (4GH21CS044)

ABSTRACT

a camera system for a 3D graphics application using OpenGL. The system provides a Camera class that abstracts the management of the camera's position, orientation, and projection settings. It supports dynamic view adjustments through user input, allowing for intuitive navigation within the 3D environment. The camera system calculates the view and projection matrices essential for rendering scenes from the camera's perspective. By integrating this system into the rendering pipeline, the project enables flexible and accurate scene visualization. And user can easily handle. The design emphasizes modularity and maintainability, facilitating enhanced user interaction and visual experiences.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OpenGL

OpenGL (Open Graphics Library) is a powerful, cross-platform application programming interface (API) designed for rendering 2D and 3D graphics. Developed and maintained by the Khronos Group, OpenGL provides a comprehensive set of functions for creating and manipulating graphical content in a wide range of applications, from simple graphical interfaces to complex 3D simulations.

1.2 Key Features of OpenCV

- **Cross-Platform Compatibility**: OpenGL is supported on various operating systems, including Windows, macOS, and Linux, making it a versatile choice for graphics programming across different environments.
- **Hardware Acceleration**: OpenGL leverages the capabilities of modern graphics processing units (GPUs) to deliver high-performance rendering. This allows for efficient handling of complex graphical operations and large datasets.
- **State-Based Architecture**: OpenGL uses a state machine architecture where various settings (such as color, texture, and lighting) are managed through a series of state variables. This design simplifies the process of configuring rendering parameters and applying graphical effects.
- **Shader Support:** OpenGL incorporates programmable shaders written in GLSL (OpenGL Shading Language), enabling developers to write custom vertex and fragment shaders. This flexibility allows for advanced visual effects and optimizations tailored to specific needs.
- **Extensive Functionality:** OpenGL provides a wide array of functions for rendering primitives (points, lines, triangles), handling textures, managing transformations, and performing other essential graphics operations. It supports both immediate mode and modern approaches like vertex buffer objects (VBOs) and frame buffer objects (FBOs) for efficient data handling.

1.3 Applications of OpenCV

1. 3D and 2D games development
2. Scientific and Engineering simulations
3. Scientific Visualization
4. CAD and 3D Modeling
5. Virtual Reality (VR) and Augmented Reality (AR)
6. Interactive Media and Art
7. Graphics Research and Development

1.4 Project Overview

The objective of this project is to develop a robust and flexible camera system for a 3D application using OpenGL. The camera system will manage the camera's position, orientation, and projection settings to enable dynamic and interactive viewing experiences within a 3D environment. This system will be designed to integrate seamlessly with OpenGL's rendering pipeline, allowing for accurate and efficient visualization of scenes from various perspectives.

1.5 Scope of the Project

The scope of this project involves designing and implementing a comprehensive camera system for 3D graphics applications using OpenGL. This includes creating a Camera class to manage essential attributes such as position, orientation, and projection settings. The system will compute and update the view and projection matrices, integrating seamlessly with OpenGL's rendering pipeline to ensure accurate scene rendering. User interaction will be supported through intuitive controls for adjusting camera movement and orientation based on keyboard and mouse input. The project also encompasses performance optimization to handle real-time updates efficiently. focusing instead on fundamental camera management and integration.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

camera systems play a critical role in determining how scenes are viewed and interacted with. A well-designed camera system is essential for providing dynamic and immersive user experiences, especially in applications such as video games, simulations, and virtual reality. The literature survey explores various approaches and techniques related to camera systems in computer graphics, with a focus on their integration with rendering pipelines like OpenGL.

2.2 Camera Systems in Computer Graphics

Camera systems are fundamental in computer graphics, responsible for capturing and rendering scenes from specific viewpoints. Traditional camera models in graphics often involve a perspective projection, where objects appear smaller as they get further from the viewer, mimicking human vision. In contrast, orthographic projection maintains object size regardless of depth, useful for architectural and engineering drawings.

Key resources include:

- **"Computer Graphics: Principles and Practice"** by John F. Hughes et al.: This seminal textbook provides foundational knowledge on camera models and projection techniques, explaining the mathematical underpinnings of view and projection matrices.

- **"Real-Time Rendering"** by Tomas Akenine-Möller et al.: Offers insights into advanced camera systems and real-time rendering techniques, emphasizing the importance of efficient matrix computations and their impact on performance.

2.3 Camera Basics

OpenGL by itself is not familiar with the concept of a *camera*, but we can try to simulate one by moving all objects in the scene in the reverse direction, giving the illusion that we are moving.
Orthographic cameras maintain the relative size of objects regardless of distance, while perspective cameras simulate depth, making objects appear smaller as they move away from the camera

2.4 View and Projection Matrices

The computation of view and projection matrices is central to camera systems. The view matrix transforms world coordinates into camera coordinates, positioning the camera and orienting it to view the scene. The projection matrix then projects these coordinates onto the 2D screen. Notable techniques and tools include:

- **"OpenGL Programming Guide:** The Official Guide learning opengl.com/ Provides practical guidance on implementing view and projection matrices using OpenGL functions such as glm::lookAt and glm::perspective.
- **"Fundamentals of Computer Graphics" :** Covers the theory and application of view and projection matrices, detailing how transformations are applied in 3D graphics.

2.5 Real-Time Performance and Optimization

Optimizing camera systems for real-time performance is crucial in applications such as video games and simulations, where frame rates and responsiveness are critical. in this area focuses on minimizing computational overhead and ensuring smooth performance:

- **"GPU Zen: Advanced Rendering Techniques"** by Wenzel Jakob and Eric Heitz: Addresses optimization techniques for graphics pipelines, including strategies for improving the performance of camera systems.
- **"Introduction to Computer Graphics and the Vulkan API"** by Kenwright et al.: Although focused on Vulkan, it provides valuable insights into performance considerations and optimization strategies applicable to OpenGl

CHAPTER 3

SYSTEM DESIGN

3.1 Functional Requirements

1. **Rendering**: Render 2D or 3D scenes using OpenGL. Display the scene from the perspective of a virtual camera. Ensure real-time rendering updates based on camera movements and scene changes.
2. **Camera Control**: Allow users to control the position and orientation of the virtual camera. Implement zoom functionality to adjust the field of view or scale of the rendered scene.
3. **User Interface (UI)**: Provide intuitive UI elements (e.g., sliders, buttons) to manipulate camera parameters Support switching between different camera modes (e.g., perspective projection, orthographic projection).
4. **Scene Management**: Enable adding, removing, or manipulating objects within the scene. Implement lighting models and shaders to enhance scene realism.Support texture mapping for realistic object appearances.
5. **Display Pencil Sketch**: The pencil sketch image must be displayed in a full screen window after processing.
6. **Error Handling**: The application must handle errors such as unsupported file formats, missing files, and invalid parameter values gracefully.
7. **Exit Functionality**: The application must close all windows and exit when the user decides to quit.

3.2 Non-Functional Requirements

1. **Performance**: The application should maintain a minimum frame rate (e.g., 30 frames per second) for smooth rendering and interaction. OpenGL rendering should be optimized to minimize CPU and GPU usage while maximizing frame rates.
2. **Usability**: The UI should be intuitive and user-friendly, with clear controls for manipulating the camera, The application should support common accessibility standards.
3. **Compatibility**: The application should be compatible with multiple operating systems .

4. **Scalability**: The application should scale gracefully with increasing complexity of scenes and number of objects without significant degradation in performance.
5. **Reliability**: The application should operate reliably without frequent crashes or unexpected behavior. Robust error handling mechanisms should be in place to gracefully manage and recover from errors or exceptions.
6. **Maintainability**: Design the application with clear separation of concerns and modular components to facilitate maintenance and updates. maintenance and future enhancements.

3.3 Software Requirements

1. **OpenGL Library**:

   - **GLFW (Graphics Library Framework):** For window and input management.
   - **GLM (OpenGL Mathematics):** For vector and matrix operations aligned with OpenGL conventions

2. **Python**:

   - **Version:** Python 3.x
   - **Purpose:** To run the script and interact with the OpenGL and Glfw libraries.

3. **User Interface (UI) Library**:

   - **Version:** Comes bundled with standard Python installations
   - **Purpose:** For creating the graphical user interface, including file dialogs.

4. **NumPy Library**:

   - **Version:** NumPy 1.x or later
   - **Purpose:** For handling array operations and mathematical functions required for image processing.

3.4 Hardware Requirements

1. **Processor**:
    - A modern multi-core processor (e.g., Intel i5/i7 or AMD Ryzen) to ensure smooth performance during image processing.
2. **Memory (RAM)**:
    - Minimum of 4 GB RAM; 8 GB or more recommended for handling larger images and more complex processing tasks.
3. **Storage**:
    - Sufficient disk space to store the application and processed images. A minimum of 100 MB of free space is recommended.
4. **Display**:
    - A monitor with a resolution of at least 1024x768 pixels to effectively display images and the application interface.
5. **Graphics Hardware**:
    - While not strictly necessary, having a dedicated graphics card can enhance performance for more intensive image processing tasks.

These requirements ensure that the application runs efficiently and provides a good user experience while performing image processing tasks.

3.5 System Architecture

1. **Input Handling**: **Keyboard and Mouse Input**: To control camera movements and settings.
2. **Camera Management**:
   - **Camera Class**: Manages camera properties like position, orientation, and zoom.
   - **View Matrix:** Transforms world coordinates to camera coordinates.
   - **Projection Matrix**: Defines how the scene is projected onto the screen (perspective or orthographic).
3. **Rendering Pipeline:**
   - **Vertex Shader**: Processes vertex positions and attributes.
   - **Frame Buffer**: Stores the final rendered image.
4. **OpenGL Context**:
   - **Window Creation:** Using libraries like GLFW or SDL.
   - **Context Management:** Ensures OpenGL commands are executed in the correct context.

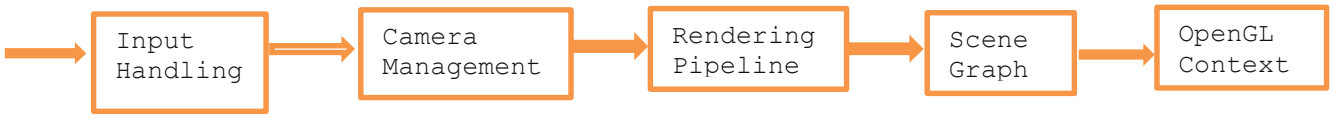

Fig 3.1 System Architecture

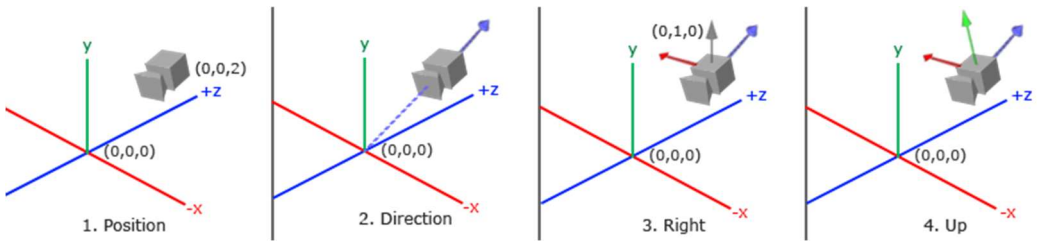

Fig 3.2 Camera positions

CHAPTER 4

IMPLEMENTATION

The implementation of this building camera system, which uses OpenGL and GLFW to apply a camera position effect to an windows and display it in full screen mode.

The project consists of:

1. Camera / view space using OpenGL.
2. Look at view effect to the object.
3. Look around and movement speed.

1. Camera / view space

- **Camera position:** The camera position is a vector in world space that points to the camera's position.

  glm::vec3 cameraPos = glm::vec3(0.0f, 0.0f, 3.0f);

- **Camera Direction:** the camera point to the origin of our scene: (0,0,0) Subtracting the camera position vector from the scene's origin vector thus results in the direction vector .

  glm::vec3 cameraTarget = glm::vec3(0.0f, 0.0f, 0.0f);

  glm::vec3 cameraDirection = glm::normalize(cameraPos - cameraTarget);

  Key Steps:

  - **Right – axis**: glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f);
    glm::vec3 cameraRight = glm::normalize(glm::cross(up, cameraDirection)).
  - **Up-axis:** glm::vec3 cameraUp = glm::cross(cameraDirection, cameraRight);

2. Look at view:

- **Function:** a coordinate space using 3 perpendicular (or non-linear) axes you can create a matrix with those 3 axes plus a translation vector and you can transform any vector to that coordinate space by multiplying it with this matrix.

Key Steps:

- GLM then creates the LookAt matrix that we can use as our view matrix:

  Glm::mat4 view;

  View = glm::lookat(glm::vec3(0.0f, 0.0f, 3.0f).

  glm::vec3(0.0f, 0.0f, 0.0f),

  glm::vec3(0.0f, 1.0f, 0.0f));

- glfwGetTime function:

  const float radius = 10.0f;

  float camX = sin(glfwGetTime()) * radius;

  float camZ = cos(glfwGetTime()) * radius;

  glm::mat4 view;

  view = glm::lookAt(glm::vec3(camX, 0.0, camZ), glm::vec3(0.0, 0.0, 0.0), glm::vec3(0.0, 1.0, 0.0));

3. mouse input and perspective projection matrix

- **Function:** get input from user()
- **Description:** This function opens when user click and scroll th mouse,

Code for input function:

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
   if (firstMouse)
   {
      lastX = xpos;
      lastY = ypos;
      firstMouse = false;
   }

   float xoffset = xpos - lastX;
   float yoffset = lastY - ypos;
   lastX = xpos;
   lastY = ypos;

   float sensitivity = 0.1f;
   xoffset *= sensitivity;
   yoffset *= sensitivity;

   yaw   += xoffset;
   pitch += yoffset;
```

```
if(pitch > 89.0f)
   pitch = 89.0f;
if(pitch < -89.0f)
   pitch = -89.0f;

glm::vec3 direction;
direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
direction.y = sin(glm::radians(pitch));
direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));
cameraFront = glm::normalize(direction);
 }
```

Zoom function:

```
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
   fov -= (float)yoffset;
   if (fov < 1.0f)
      fov = 1.0f;
   if (fov > 45.0f)
      fov = 45.0f;
}
```

perspective projection matrix

```
projection = glm::perspective(glm::radians(fov), 800.0f / 600.0f, 0.1f, 100.0f);
```

Scroll callback function:

```
glfwSetScrollCallback(window, scroll_callback);
```

This implementation covers viewing the object, processing window and view the object effect, and displaying the window and zoom and scroll out function we implemented ,

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

This project the development of a camera system using OpenGL has demonstrated the powerful capabilities and flexibility of the OpenGL framework for 3D graphics rendering. By leveraging OpenGL's robust features, we have successfully implemented a camera system that allows for dynamic and interactive control of the viewing perspective within a 3D environment. The system supports key functionalities such as camera translation, rotation, and zoom, providing users with an immersive and adaptable viewing experience. The integration of matrix transformations and the use of view and projection matrices have been critical in achieving smooth and accurate camera movements. The successful application of OpenGL in this project reaffirms its value as a fundamental tool for 3D graphics development and opens up exciting possibilities for future exploration and innovation in the field.

5.2 Future work

For future development, the camera system could benefit from integrating advanced modes like orbit and fly-through perspectives, as well as path-following functionality for dynamic scene interactions. Enhancing the user interface to allow customizable settings and adding depth of field effects would improve realism. Exploring VR and AR integration could offer immersive experiences, while optimizing performance for large-scale scenes and adding multi-camera support would address scalability. Implementing automated transitions for cinematic sequences, user-defined presets, and improved collision detection would enhance usability. Additionally, incorporating machine learning for intelligent adjustments and developing a plugin system for third-party extensions would further extend the system's capabilities. Gathering user feedback would ensure the system meets practical needs and preferences. allowing users to easily experiment with different settings and see immediate results.

CHAPTER 6

APPENDICES

6.1 Appendices A: Snapshots

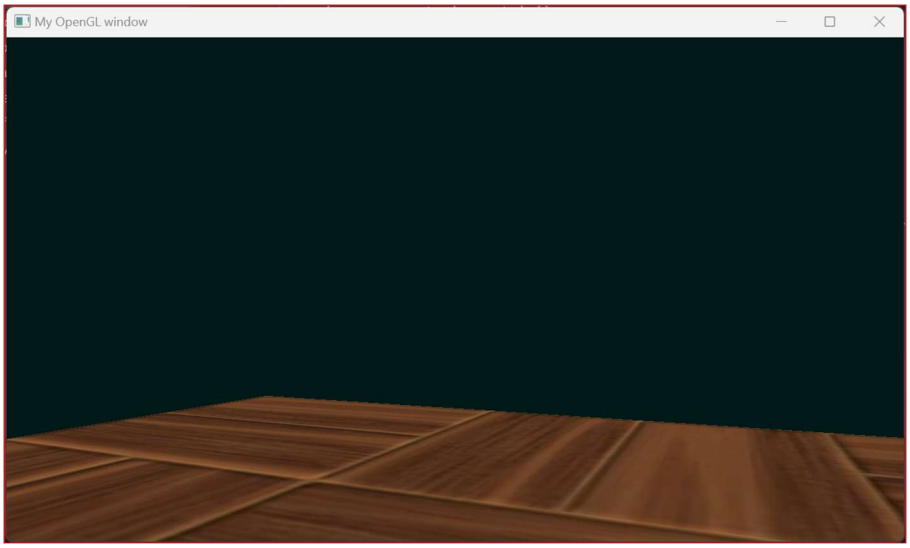

Fig 6.1 Camera view of window

Fig 6.2 Camera view of an object

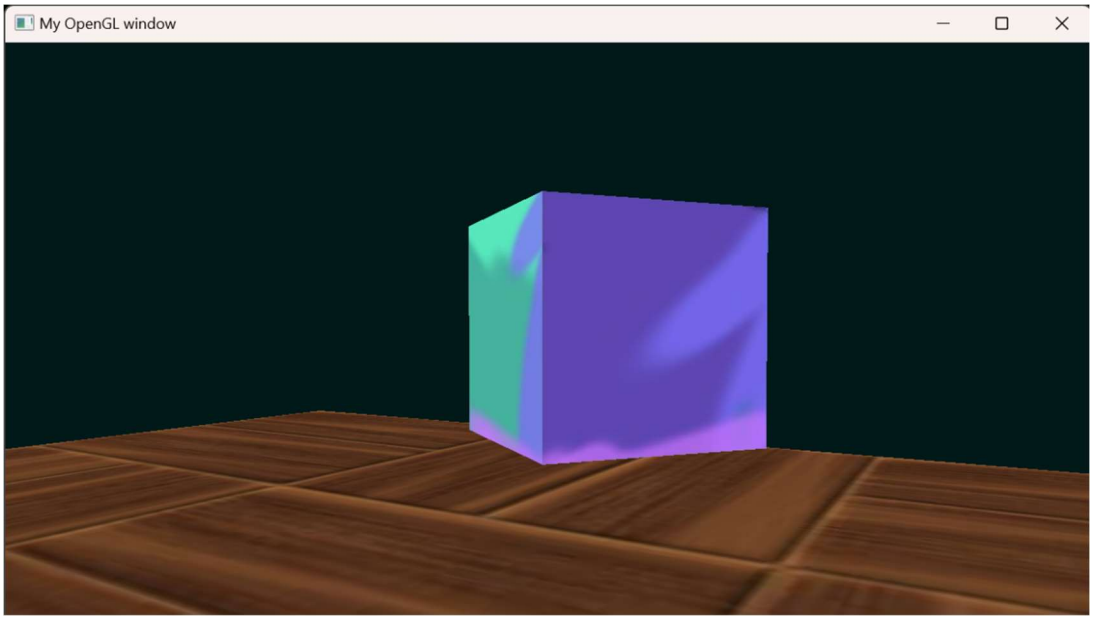

Fig 6.3 Camera view of rotation object

REFERENCES

[1] Learning OpenGL Programming Guide: The Official Guide to Learning OpenGL by Dave Shreiner, Graham Sellers, John Kessenich, and Bill Licea-Kane

[2] Programming Computer Vision with Python: Tools and algorithms for analyzing images by Jan Erik Solem

[3] https://learnopengl.com/Getting-started/Camera

[4] https://docs.opengl.org/4.x/d4/d13/tutorial_py_filtering.html#:~:text=Gaussian%20Blurring&text=It%20is%20done%20with%20the,directions%2C%20sigmaX%20and%20sigmaY%20respectively.