MTE391 SOFTWARE DESIGN PROJECT

FINAL PROJECT

3D SIMULATION

Submitted by:

Zeynep Sude Koç (EEE)

Project Mentors:

Dr. Doğan Çörüş

Faculty of Engineering and Natural Sciences

Kadir Has University

Fall 2024

Libraries and Dependencies: The code uses crucial libraries such as OpenGL, GLFW (for window and input operations), and GLEW (for OpenGL extensions). Additionally, GLM (OpenGL Mathematics Library) is utilized for mathematical operations.

Sphere Creation: The function `calculateSphereVertices` calculates the vertex positions of a sphere and stores them in a vector. This provides the basic data for visualizing spheres.

Collision Detection: The functions `checkCollisions` and `checkCubeCollisions` detect collisions between spheres and with cube boundaries. A simple collision response is implemented: the velocities of the colliding spheres are exchanged.

Physics Update: The positions of the spheres are updated over time, and a gravity effect is added.

Visualization: Spheres and a cube are visualized using OpenGL. The color of the spheres is randomly chosen and they rotate at a specific velocity.

Camera Control: The movement of the camera is controlled by the user's keyboard input. This allows you to change the view of the 3D scene.

Time Management: Time measurement is done using `glfwGetTime`, and the elapsed time for each loop is calculated. This is used for physics updates and animations.

Shaders: The code includes vertex and fragment shaders, and these shaders are used to render the spheres and cube.

Render Loop: The main loop creates the window and performs OpenGL operations. In each loop, the positions of the spheres are updated, collisions are checked, and visualization is performed.

Cleanup and Shutdown: When the program ends, the created OpenGL objects are cleaned up, and the GLFW window is closed.

The code demonstrates the fundamentals of 3D visualization using OpenGL and includes topics such as collision detection. It also addresses camera movement and time-based animations.

Challenges Encountered During the Development of the Code

Understanding of OpenGL: Grasping the concepts and workings of OpenGL.

Library Dependencies: OpenGL projects often require dependencies like GLFW, GLEW, GLM. Installing these libraries correctly and integrating them into the project can be a challenge.

Shader Programming: Shaders play a crucial role in OpenGL projects. Writing and compiling vertex and fragment shaders initially created complexity.

Time Management: Providing accurate time management for animations and physical simulations, updating objects that change over time.

Debugging: Debugging OpenGL projects can be challenging as errors might not lead to visible results.

Additional Features and Interesting Aspects of Your Simulation

- Spheres start at random positions and velocities and move over time.
- Interaction is provided with keyboard controls (w,a,s,d).
- When spheres collide, their velocity vectors are exchanged. This collision control makes the interactions between spheres realistic.

PROJECT PLAN

Installation of Required Libraries:

Installation of GLEW (OpenGL Extension Wrangler Library).

Installation of GLFW (for window management and inputs).

Installation of GLM (OpenGL Mathematics, for mathematical operations).

Including these libraries into your project after installation.

Project Configuration:

Configuring the project using CMake or another configuration tool.

Linking GLEW, GLFW, and GLM in the project.

Window Creation and Initialization of OpenGL Context:

Creating a window using GLFW.

Accessing OpenGL functions by initializing GLEW.

Preparation of Shader Program:

Writing Vertex and Fragment shaders.

Compiling shaders and linking them to a shader program.

Geometry Definition:

Writing functions that will create vertex and index data for the sphere.

Using Vertex Buffer Objects (VBO) and Element Buffer Objects (EBO) to transfer data to the GPU.

Render Loop:

Creating a render loop that continuously draws until the window is closed.

Clearing the screen and drawing geometries with each draw.

Input Control:

Using GLFW callback functions to capture user inputs (keyboard, mouse, etc.).

Matrix Transformations:

Defining and calculating model, view, and projection matrices using GLM.

Sending these matrices to the shader program as uniforms.

Camera System:

Establishing a camera system that can change the viewer position and direction of view.

Physics Simulation:

Writing physical functions that will simulate the movement and collisions of balls.

Drawing Settings:

Enabling depth testing using the Z-buffer.

Cleanup Operations:

Properly releasing resources when the program is closed.

Debugging and Testing:

Compiling, running, and debugging the code.

Documentation and Comments:

Adding explanatory comments to the code.

Additional Features and Improvements:

Researching and implementing additional graphics features such as lighting, shading, textures.