RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE LÉNSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

École Nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département Génie Industriel
Data Science et Intelligence Artificielle

VIRTUAL REALITY PROJECT REPORT

Solar system and 3D Interactive room

Binôme :
REGOUI Meriem
RAHIM Amel
Date : 02/06/2025

Contents

1 Overview

This report presents two OpenGL-based projects that showcase fundamental concepts of 3D graphics programming and interactive visualization. The first project simulates a simplified solar system featuring the Sun, Earth, and Moon, demonstrating the use of transformations, texture mapping, and camera controls to animate celestial motion realistically. The second project models a 3D interactive room, complete with walls, furniture, doors, and windows, allowing user navigation and interaction through keyboard controls. Both projects leverage key OpenGL features such as shaders, lighting, and object rendering to create immersive and visually appealing scenes. Together, these projects highlight practical applications of graphics programming, scene management, and user interaction, forming a foundation for more advanced developments in computer graphics and virtual environments.

2 SOLAIR SYSTEM

2.1 Introduction

This project simulates a simplified solar system consisting of the Sun, Earth, and Moon. The planets revolve and rotate using basic trigonometric functions, and textures are applied for realism. The background is a starfield that adds visual depth. The simulation demonstrates core OpenGL concepts such as shaders, transformations, camera handling, and object rendering.

2.2 Tools and Libraries Used

- **OpenGL**: API used for rendering 2D and 3D vector graphics.
- **GLFW**: Library for window creation and input handling.
- **GLAD**: Loads OpenGL function pointers.
- **GLM**: Provides classes and functions for vector and matrix mathematics.
- **STB Image**: Loads image files to be used as textures.

2.3 Main Code Components and Logic

2.3.1 Main Loop Logic

The main rendering loop consists of the following operations:

1. Calculate delta time for smooth movement.
2. Process user inputs for camera movement.
3. Clear the color and depth buffers.
4. Render background (starfield).
5. Render planets (Sun, Earth, Moon) with appropriate transformations.
6. Render orbital paths.
7. Swap buffers and poll for events.

2.3.2 important Functions and Their Roles

- `glfwInit()`: Initializes GLFW.
- `glfwCreateWindow()`: Creates a window with an OpenGL context.
- `gladLoadGLLoader()`: Loads OpenGL functions via GLAD.
- `glEnable(GL_DEPTH_TEST)`: Enables depth testing to correctly render overlapping 3D objects.
- `Shader class (Shader.h)`: Loads, compiles, and links vertex and fragment shaders.
- `Sphere class (Sphere.h)`: Generates a UV sphere mesh with vertices and indices.
- `loadTexture(path)`: Loads image files (e.g., JPG, PNG) into OpenGL textures.
- `Camera class`: Implements view transformation and handles keyboard/mouse input.
- `glm::perspective()`: Creates a perspective projection matrix.
- `glm::lookAt()`: Generates a view matrix simulating a camera.
- `glm::translate(), rotate(), scale()`: Apply transformations to position and animate objects.

2.3.3 Planet Rendering Logic

Each celestial body is represented by a textured sphere. The logic for each body is as follows:

Sun

- Rendered at the origin (0, 0, 0).
- Uses a static model matrix with no translation.

Earth

- Orbits the Sun using:

$$x = R \cdot \cos(\theta), \quad z = R \cdot \sin(\theta)$$

  where R is the orbital radius and θ is a function of time.

- Rotates on its axis using `glm::rotate()`.

Moon

- Orbits the Earth using a similar circular function.
- Position is calculated relative to the Earth's current position.

2.3.4 Camera and Controls

- Movement: Handled using keys :

  A: Move the camera to the left.

  D: Move the camera to the right.

  Z: Move the camera forward.

  G: Move the camera upward.

  echap: to get out.

- Zoom: Controlled with the scroll wheel.
- The camera position is updated using delta time to ensure consistent movement speed.

2.4 Rendering and Shaders

2.4.1 Vertex Shader

Handles transformation of vertex positions using:

$$gl\_Position = projection \times view \times model \times vec4(position, 1.0)$$

2.5 Fragment Shader

Computes final pixel color using texture sampling:

$$FragColor = texture(texture1, TexCoords)$$

2.5.1 Background Rendering

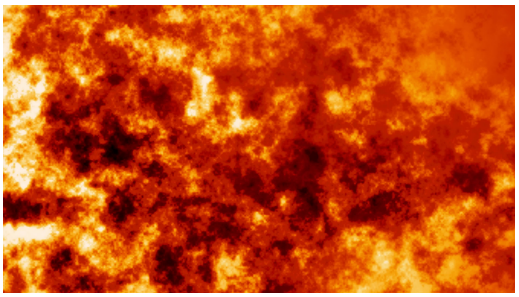
The background is drawn first using a full-screen quad. Depth testing is disabled for the background so that it doesn't interfere with the planets.

Figure 1: The galaxy

2.6 Textures

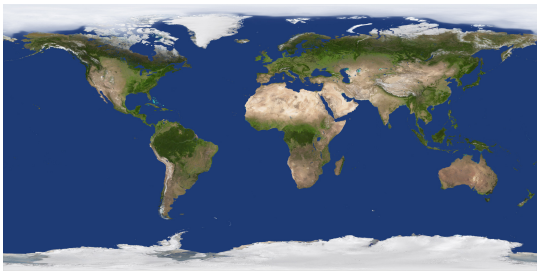
Textures are mapped onto spheres using UV coordinates. The `stb_image` library loads texture data into OpenGL. Each sphere has a dedicated texture:

(a) Galaxy



(b) sun

(a) Moon



(b) Earth

Figure 2: Textures used in our planets

2.6.1 Orbits and Visual Enhancements

- Orbits are visualized using `GL_LINE_LOOP` with a precomputed circular set of points.
- The scene is rendered in layers: background first, then planets, then orbit lines.