Project_report

Project Title: MarketMaven - 3D Shopping Market Visualization with OpenGL.

<u>Objectives:</u>

Main objectives of this project are to:

a) Develop a 3D model of a shopping center with detailed textures and lighting effects.
b) Utilize OpenGL and GLSL shaders to enhance visual realism.
c) Implement interactivity for camera movement and lighting toggles.
d) Demonstrate the use of custom geometric shapes and transformations to create a complete scene.

<u>Introduction:</u>

OpenGL (Open Graphics Library) is a powerful API for rendering 2D and 3D graphics. This project focuses on utilizing OpenGL to create a virtual shopping center environment, incorporating 3D models and applying textures to various surfaces to enhance visual realism.

In today's technologically advanced world, immersive and interactive 3D visualizations are crucial across various sectors, from gaming and virtual reality to architectural design. This project focuses on developing "MarketMaven," a 3D shopping market visualization built using OpenGL as its core rendering engine. By integrating advanced rendering techniques, custom geometric shapes, and intuitive interactive controls, the project aims to deliver a realistic and captivating representation of a shopping center environment.

<u>Theory:</u>

3D Graphics Basics:

1. **Vertices and Primitives**: The building blocks of 3D models are vertices, which are connected to form geometric primitives such as triangles.
2. **Coordinate Systems**: OpenGL uses a series of coordinate systems, including world space, view space, and clip space, which are transformed through matrices.
3. **Lighting and Shading**: Proper lighting models, such as Phong , enhance the realism of scenes by simulating how light interacts with surfaces.

OpenGL Rendering Pipeline:

1. The rendering pipeline includes stages like vertex processing, primitive assembly, rasterization, fragment shading, and output merging.
2. GLSL shaders enable customization of the vertex and fragment stages for advanced effects like dynamic lighting and texture mapping.

Basics of 3D Object Creation:

1. **Cube**: A cube is created by defining 8 vertices and connecting them with triangles to form 6 faces. Texture coordinates can be applied for each face to map textures.
2. **Cylinder**: A cylinder can be constructed by creating a circle for the base, duplicating it for the top, and connecting corresponding vertices between the two circles with quads or triangles.
3. **Sphere**: A sphere is typically generated using parametric equations or subdivision algorithms. Latitude and longitude lines are used to divide the surface into smaller triangles.
4. **Bezier Object**: Bezier objects are created using control points and Bezier curves. The curves are calculated using the Bernstein polynomial, which allows for smooth and continuous shapes.

Lighting and Texturing:

1. **Lighting**: OpenGL supports various lighting models, including ambient, diffuse, and specular lighting. Combining these components with light sources (point, directional, or spotlights) adds realism.
2. **Texturing**: Textures are mapped onto objects using UV coordinates. OpenGL provides functions for loading and applying 2D textures, ensuring correct alignment with the object's geometry.

Camera Theory:

1. A camera in OpenGL is simulated using the view matrix, which transforms world coordinates into camera space. The camera's position, orientation, and projection type (orthographic or perspective) determine how the scene is viewed.
2. Movement and rotation of the camera are implemented using transformations, allowing for first-person or third-person navigation.

Interactivity:

1. User interactivity in OpenGL applications is achieved through event handling for input devices such as the keyboard and mouse.

2. Camera systems often use transformations to allow movement and rotation within the 3D scene.

Transformations:

Transformations like translation, rotation, and scaling are applied using matrices. These transformations are fundamental to positioning and animating objects in a 3D scene.

Methodology:

Design Phase:

1. Created a blueprint of the shopping center layout, including key elements like stores, pathways, and lighting fixtures.
2. Determined the textures and lighting effects required for realism.

Development Phase:

1. **Environment Setup**: I have set up an OpenGL project using a development environment such as Visual Studio.
2. **Geometry Modeling**: Created custom geometric shapes for elements like walls, windows, and objects within the shopping center.
3. **Shader Programming**: Wrote GLSL shaders to handle dynamic lighting, textures, and shading effects.
4. **Interactivity**: Implemented controls for moving the camera and toggling lighting modes using keyboard and mouse input.

Implementation Details:

1. **Textures**: Used image files for textures and map them onto surfaces using UV mapping.
2. **Lighting**: Implemented multiple light sources (e.g., ambient, diffuse, and specular) to achieve a realistic effect.
3. **Camera System**: Used transformations to simulate first-person or third-person navigation.

Testing and Optimization:

1. Tested the application for performance issues and ensure smooth interaction.
2. Optimized geometry and shaders to reduce rendering overhead.

Pseudo Code of Used Functions:

Function drawFan():

Inputs:

VAO: Vertex Array Object (for drawing cubes).

lightingShader: Shader object used for rendering.

matrix: Transformation matrix.

Initialization

Define identity matrix identityMatrix.

Initialize transformation matrices (translateMatrix, rotateXMatrix, rotateYMatrix, rotateZMatrix, scaleMatrix, model, etc.).

Set a default color for the fan (color).

Fan is ON (fanOn == true):

Draw Fan Rod:

Translate to rod's position.

Scale for rod dimensions.

Apply transformations and set color.

Draw Fan Center:

Rotate fan based on angle r.

Translate and scale for fan center.

Render using sphere object.

Draw Fan Propellers:

For each propeller (left, right, up, down):

Translate to propeller's position.

Apply rotational and inverse transformations.

Scale for propeller dimensions.

Apply transformations and set color.

Render as a cube.

Update Rotation:

Increment r for continuous rotation.

Function drawCurveObj ()

1. Use the lighting shader and set material properties.
2. Bind the VAO and render the curve object using glDrawElements.
3. Unbind the VAO.

Function nCr ()

Calculate C(n,r)C(n, r)C(n,r) using iterative multiplication and division for efficiency.

Function BezierCurve ()

Initialize x and y coordinates to 0.

```
    For each control point:
        Compute the Bernstein polynomial coefficient.
        Update x and y with the weighted control point values.
    Store computed coordinates in xy.

Function hollowBezier ()
    Initialize Variables:
        Define variables for coordinates, normals, and parameters for curve tessellation.
    Compute Bezier Curve:
        Step through t to compute points on the curve using BezierCurve.
        Compute cylindrical coordinates (x, y, z) for each curve point.
        Normalize and store vertex normals.
    Generate Indices:
        Create triangles connecting adjacent stacks and slices.
    Combine Data:
        Merge coordinates and normals into a vertex array.
    Create VAO, VBO, and EBO:
        Copy vertex and index data to GPU.
        Set vertex attributes for position and normal.
    Return:
        Return the VAO identifier for rendering.
```

Results and Observations:

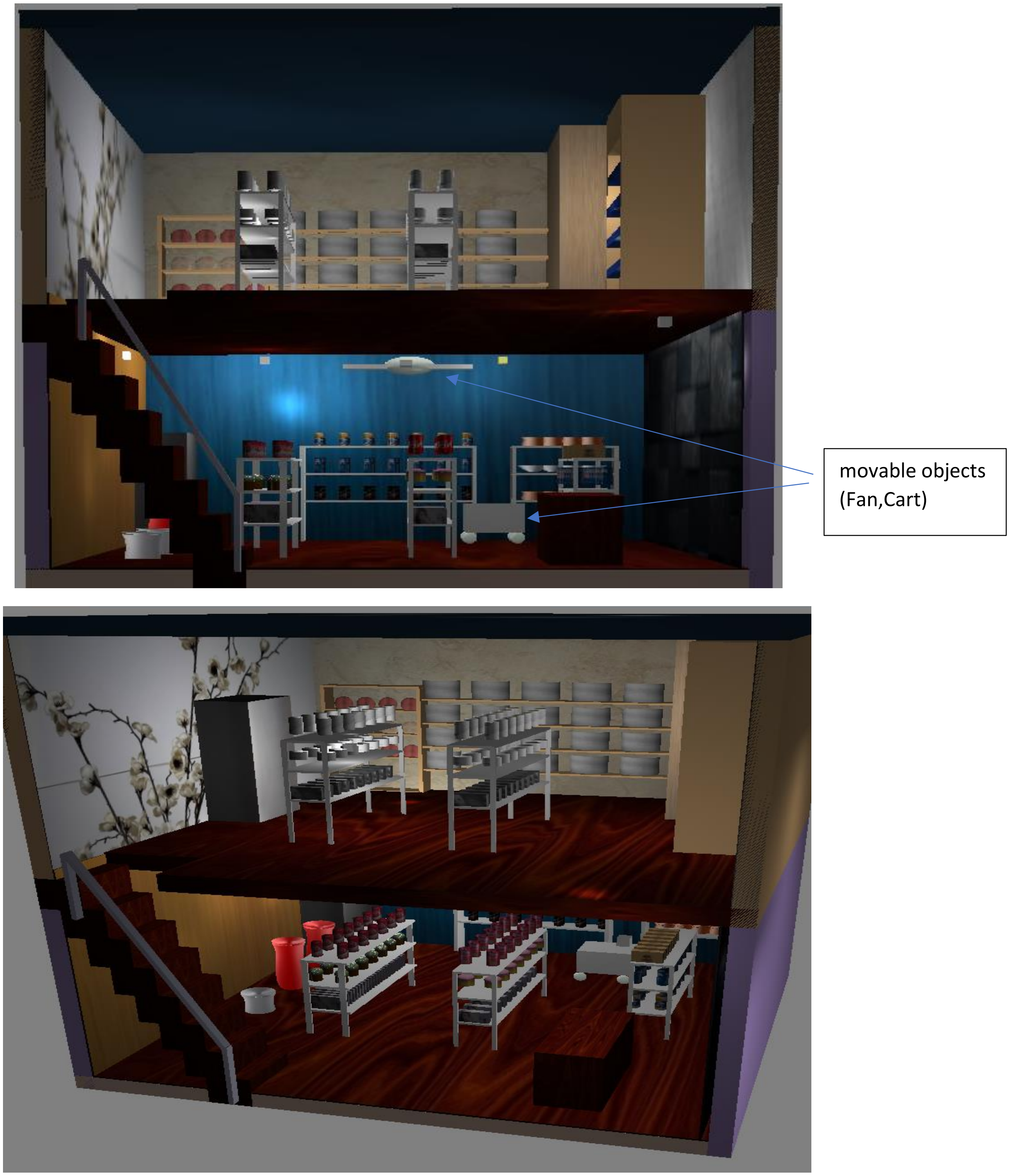

Fig A : Final output of this project

Shapes used:

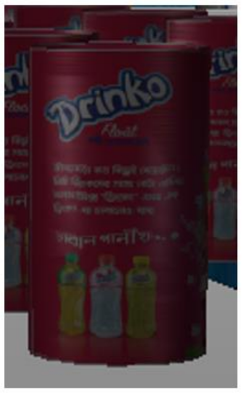

Fig 1: cylinder

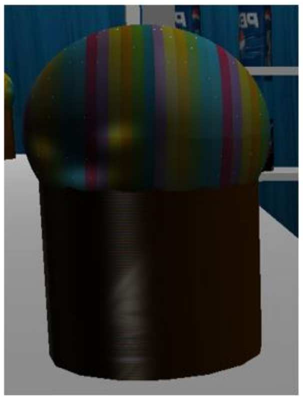

Fig 2:Sphere

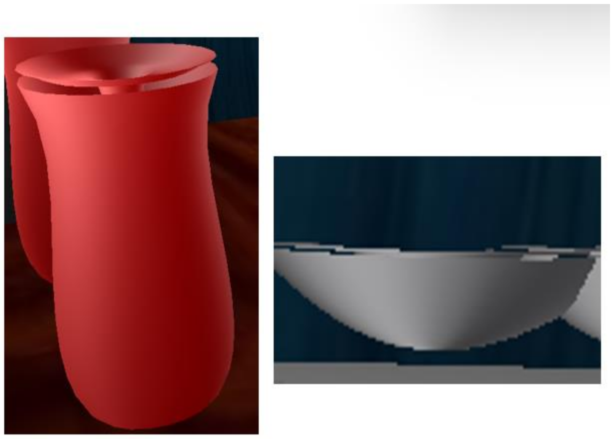

Fig 3: Objects with Bezier Curve

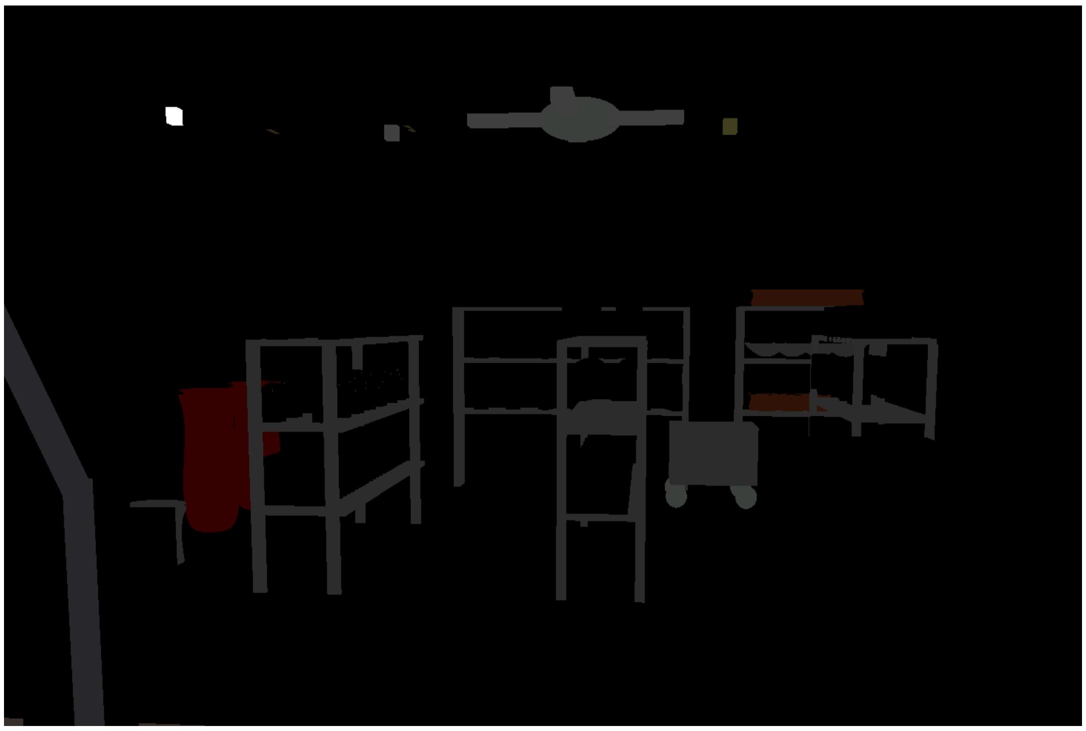

Fig 2: Directional Light On (only)

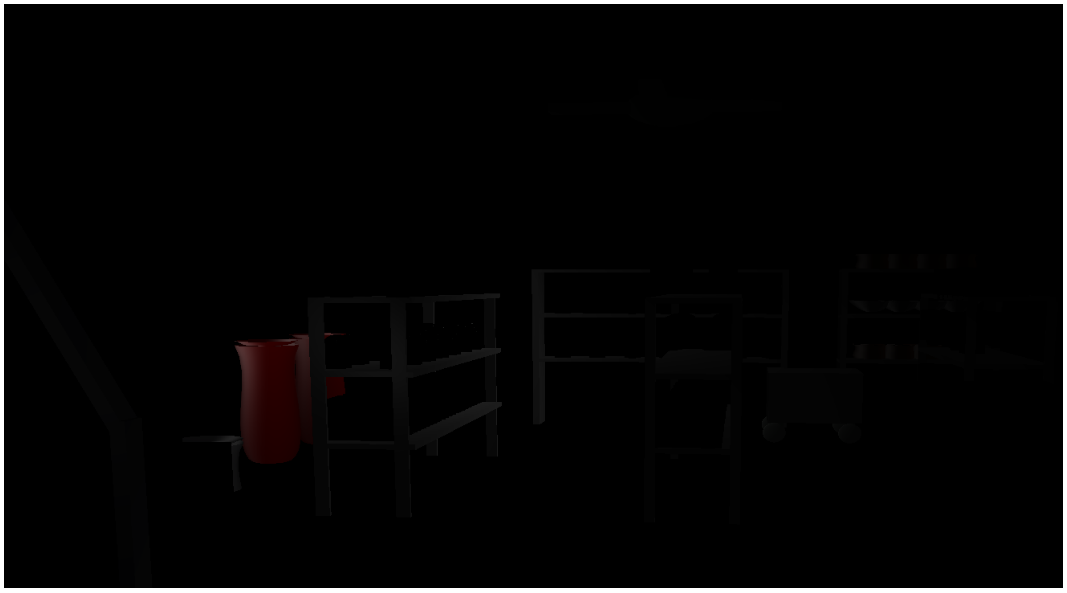

Fig 3: Only Spot light On

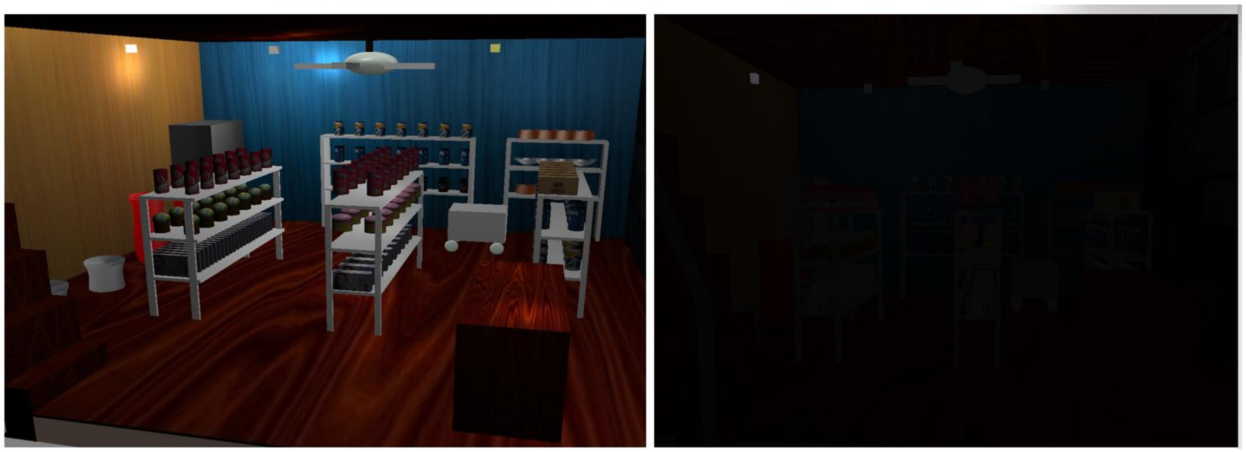

Fig 4 : Output Ambient Light Off/On

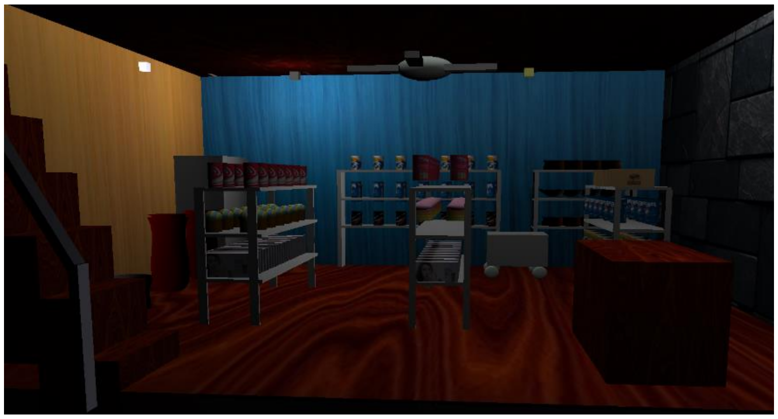

Fig 5: Only Diffuse Light On

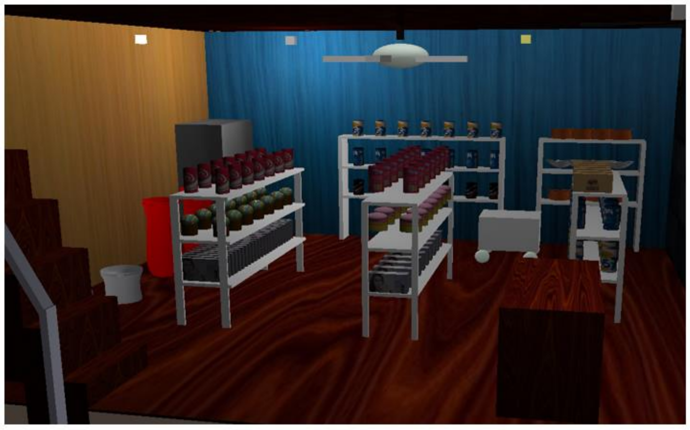
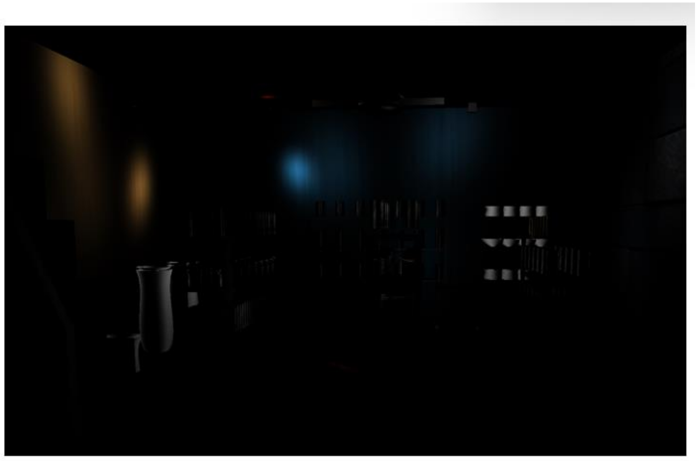

Fig 6: Specular Light Off/On

Functional Description of Keys:

General Controls

ESC → Close the window

Camera Movement

1.G → Move forward

2.T → Move backward

3.F→ Move left

4.H→ Move right

5.E → Move down

6.Q → Move up

Lighting Functionality

1.Directional Light On and Off

2.Point Light 1 and 3 On and Off

3.Spot Light On and Off

4.Ambient Light On and Off

5.Diffuse Reflection On and Off

6.Specular Reflection On and Off

Conclusion:

The "MarketMaven" project successfully demonstrates the capabilities of OpenGL for creating an immersive 3D shopping market visualization. By leveraging advanced techniques such as GLSL shaders, texture mapping, and dynamic lighting, the project achieves a balance between performance and realism. The inclusion of interactivity further enhances user engagement, making the visualization both functional and intuitive. This project highlights the potential of 3D graphics in applications like architectural visualization, virtual tours, and educational tools, paving the way for further exploration in real-time 3D rendering technologies.

References:

1. https://learnopengl.com/Getting-started/Textures
2. https://www.opengl-tutorial.org/beginners-tutorials/tutorial-5-a-textured-cube/
3. https://www.khronos.org/opengl/wiki/how_lighting_works
4. https://stackoverflow.com/questions/7996053/opengl-illumination-and-shining
5. https://learnopengl.com/Lighting/Basic-Lighting
6. https://www.khronos.org/opengl/wiki/how_lighting_work