**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

Jnana Sangama, Belagavi – 590018, Karnataka State, India

Mini Project Entitled

# "Terrain Rendering"

Submitted in partial fulfillment of the requirements for the award of degree of

**BACHELOR OF ENGINEERING**
**IN**
**COMPUTER SCIENCE AND ENGINEERING**

**For the academic year 2016-2017**

**Submitted by:**

**NAMITA (1MV14CS058)**

Carried out at

**Sir M. Visvesvaraya Institute of Technology**
**Bangalore - 562157**

Under Guidance of

**Mrs. Monika Rani HG**
Asst. Professor
**SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY**
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**BENGALURU – 562157**

**SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY**

(Affiliated to Visvesvaraya Technological University, Belagavi)

Bangalore – 562157

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

## CERTIFICATE

Certified that the mini project work entitled **“TERRAIN RENDERING’’** is a bonafide work carried out by **Namita (1MV14CS058)** in partial fulfillment for the award of Degree of **Bachelor of Engineering in Computer Science Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2016-2017 in **Computer Graphics and Visualization Laboratory.** The mini project report has been approved as it satisfies the academic requirements in respect of the mini project work prescribed for the course of Bachelor of Engineering Degree.

| | |
|---|---|
| Signature of the guide | Signature of the HoD |
| **Mrs. Monika Rani HG** | **Prof. Dilip K Sen** |
| Asst. Professor, Department of CSE | HoD, Department of CSE |
| Sir. MVIT | Sir. MVIT |

External Examiner

Internal Examiner

## ACKNOWLEDGEMENT

The successful completion of any work depends upon the cooperation and help of many people and not just those who directly execute the work. It is difficult to express in words our profound sense of gratitude to those who helped us, but we make a humble attempt to do so.

We express our sincere gratitude to our principal, Sir MVIT for providing facilities.

We wish to place on record our sincere thanks to **Prof Dilip K. Sen, Head of the Department**, Computer Science and Engineering for encouragement and support.

We express our sincere gratitude to our internal guide **Mrs. Monika Rani H G, Asst. Professor in Computer Science and Engineering Department** for giving us the valuable suggestions regarding the project.

We also thank the members of the faculty of CSE department, whose suggestions helped us in the course of this project.

Our heartfelt thanks to all the above mentioned people who have contributed in the accomplishment of this project.

**NAMITA (1MV14CS058)**

## ABSTRACT

This project demonstrates rendering of a simple terrain. It displays a basic landscape having trees where the user can move across the terrain using the keyboard directional keys. The user view is a first person view.

The keyboard keys can be used as follows:

Left arrow key to move left

Right arrow key to move right

Top arrow key to move forward

Bottom arrow key to move backward

Key U to move up

Key D to move down

# CONTENTS

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 Overview

The ancient Chinese proverb, “a picture is worth ten thousand words” became a cliché in our society. Graphics provides one of the most natural ways of communication with the computer, since our highly developed 2D and 3D pattern recognition ability allows us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics thus permits extensive, high bandwidth user computer interaction. This significantly enhances our ability to understand data, to perceive trends and to visualize real and imaginary objects, indeed to create a “virtual world” that we can explore from arbitrary points and views. It makes communication more efficient, graphics makes possible higher quality and more precise results or products, greater productivity, and lower analysis and design cost.

OpenGL is a premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry’s most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment

## 1.2 Applications of Computer Graphics

- Computational biology
- Computational physics
- Computer-aided design
- Computer simulation
- Graphic design
- Information visualization
- Scientific visualization

# CHAPTER 2

# LITERATURE SURVEY

## 2.1 Early Graphic Systems

Computer graphics started with pen plotter model. We had Cathode Ray Tube Display showing the graphics. Each line drawn was a result of intense calculation which was a huge overhead a few years back.

## 2.2 OpenGL

OpenGL is a standard specification defining cross platform API for writing applications that produce 2D and 3D graphics. It contains multiple different function calls that help develop complex graphics with help of simple primitives. Developed by Silicon Graphics Inc. in 1992. Now it's managed by the nonprofit technology consortium, the Khronos Group. OpenGL has a set of library that help in various functions. They are GL, GLU and GLUT. OpenGL Library or GL provides a powerful yet primitive set of commands. OpenGL Utility Library or GLU contains several routines that help setting up matrices for specific viewing orientation, projections and surface rendering. OpenGL Utility Toolkit Library or GLUT contains routines that help in windowing functions and is system independent.

## 2.3 Existing System

The existing graphics systems were the graphics header in C/C++. These graphics system are not system independent. Moreover, the underlying hardware knowledge is important for proper working of the code. Moreover, only 2D graphics were supported. Complex graphics concepts like camera position, shading, 3D graphics, material properties were absent.

## 2.4 Proposed System

To achieve 3D graphics effects, OpenGL software was made. Moreover, system hardware independence and cross platform support OpenGL became famous. OpenGL is more streamlined than other graphics system APIs. The concept of building from primitives made it widely accepted by developers. It even supports animations, function driven events, callback functions. The transformation functions provide a more powerful ability to graphic coders to design their dreams digitally.

# CHAPTER 3

# SYSTEM REQUIREMENTS

## 3.1 User Requirements

- Easy to understand and should be simple.
- The built-in functions should be utilized to the maximum extent.
- OpenGL library facilities should be used.

## 3.2 Hardware Requirements

- Intel Pentium CPU 2.66 GHZ
- Minimum of 1 GB RAM
- Standard 108 Keyboard
- Recommended monitor resolution 800x600

## 3.3 Software Requirements

- Windows 64-bit Operating System / Linux Operating System
- OpenGL Tools
- OpenGL Extension Wrangler Library (GLEW)
- Lightweight Java Game Library (LWJGL) 2
- Slick-Util
- Java programming IDE and JDK

**IDE:** Eclipse IDE or NetBeans IDE

# CHAPTER 4

# DESIGN AND IMPLEMENTATION

To design the ‘Terrain Rendering Scene’ using the glut library, we need to understand various concepts, components and utility functions that are essential to implement/integrate the required visual effects. Hence by using the following pipeline shown in Fig 4.1 we designed our project.

## 4.1 Design

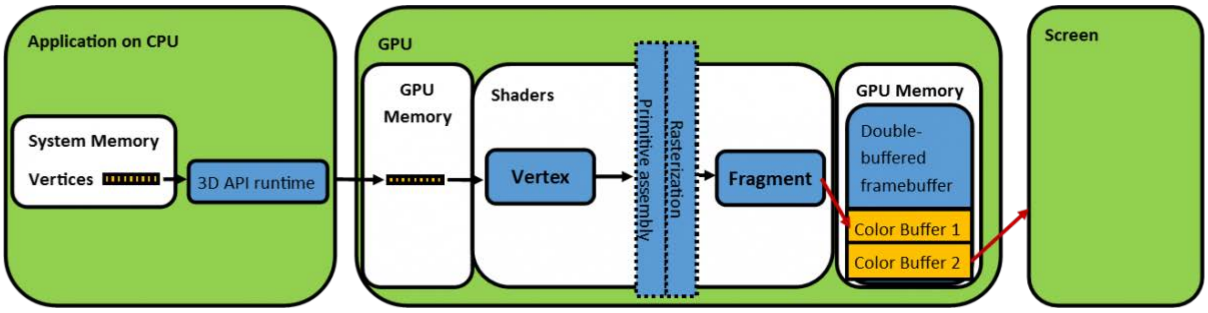


**FIG 4.1 OpenGL pipeline**

The basic functionality of the graphics pipeline is to transform the 3D scene, given a certain camera position and camera orientation, into a 2D image that represents the 3D scene from this camera's viewpoint. The application running on the CPU is the starting point for the graphics pipeline. The application will be responsible for the creation of the vertices and it will be using a 3D API to instruct the CPU/GPU to draw these vertices to the screen. We'll typically want to transfer our vertices to the memory of the GPU. As soon as the vertices have arrived on the GPU, they can be used as input to the shader stages of the GPU. The first shader stage is the vertex shader, followed by the fragment shader. The input of the fragment shader will be provided by the rasterizer and the output of the fragment shader will be captured in a color buffer which resides in the backbuffer of our double-buffered framebuffer. The contents of the front buffer from the double-buffered framebuffer is displayed on the screen. In order to create animation, the front and back buffer will need to swap roles as soon as a new image has been rendered to the backbuffer.

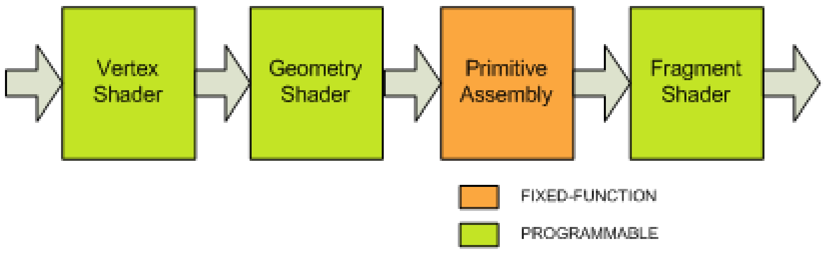


**FIG 4.2 Imaging process**

Each object comprises a set of graphical primitives. Each primitive comprises a set of vertices. We can think of the collection of primitive types and vertices as defining the geometry of the scene. We must process all these vertices in a similar manner to form an image in the frame buffer. The four major steps in the imaging process:

1. Vertex processing
2. Clipping and primitive assembly
3. Rasterization
4. Fragment processing

In the first block of our pipeline, each vertex is processed independently. The two major functions of this block are to carry out coordinate transformations and to compute a color for each vertex.

The second fundamental block in the implementation of the standard graphics pipeline is for clipping and primitive assembly. We must do clipping because of the limitation that no imaging system can see the whole world at once. Cameras have film of limited size, and we can adjust their fields of view by selecting different lenses.

The primitives that emerge from the clipper are still represented in terms of their vertices and must be converted to pixels in the frame buffer. The output of the rasterizer is a set of fragments for each primitive. A fragment can be thought of as a potential pixel that carries with it information, including its color and location, that is used to update the corresponding pixel in the frame buffer.

The final block in our pipeline takes in the fragments generated by the rasterizer and updates the pixels in the frame buffer

We have used LWJGL (Lightweight Java Game Library) through this process extensively. LWJGL is a Java library that enables cross-platform access to popular native APIs useful in the development of graphics (OpenGL), audio (OpenAL) and parallel computing (OpenCL) applications. This access is direct and high-performance, yet also wrapped in a type-safe and user-friendly layer, appropriate for the Java ecosystem. LWJGL is an enabling technology and provides low-level access. It is not a framework and does not provide higher-level utilities than what the native libraries expose.

## 4.2 Header Files

**#include <stdio.h>**

**stdio.h** which stands for "standard input/output header" is the header in the C standard library that contains macro definitions, constants and declarations of functions and types used for various standard input and output operations.

**#include <math.h>**

**math.h** is a header file in the standard library of the C programming language designed for basic mathematical operation.

**#include <windows.h>**

**windows.h** is a Windows-specific header file for the C/C++ programming language which contains declarations for all of the functions in the Windows API. It defines a very large number of Windows specific functions that can be used in C. Here data structure used is float and integer type.

## 4.3 Simple Geometry

**void glBegin(glEnum mode):** This function initiates a new primitive of type mode and starts the collection of vertices. Values of this mode include GL_POINTS, GL_LINE_STRIP and GL_QUADS.

**void glEnd():** Terminates a list of vertices.

**glVertex2f(coordinates):** This function defines the vertices of 2D figure with float as data type.

**glutInit(int argc, char *argv):** Initialize GLUT. The arguments from main are passed in and can be used by the application.

**glutCreateWindow(char*title):** Create a Window on the display, the string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

**glutInitDisplayMode(unsigned int mode):** Request a display with the properties in mode the value of mode is determined by the logical or of options including the color model (GLUT_RGB,GLUT_INDEX) and buffering (GLUT_SINGLE,GLUT_DOUBLE).

**glutInitWindowSize(int width, int height):** Specifies the initial height and width of the window in pixels.

**glutInitWindowPosition(int x, int y):** Specifies the initial position of top-left corner of the window in pixels.

**glutMainLoop():** Causes the program to enter an event processing loop. It should be the last statement in main.

**glutDisplayFunc(void (*func)(void)):** Registers the display function that is executed when the window needs to be redrawn.

**glutPostRedisplay():** Requests that the display callback be executed after the current callback returns.

## 4.4 Interaction

**glutKeyboardFunc(void(*func)(unsigned char key,int x,int y)):** Sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback.

**glutIdleFunc(void(*f)(void)P):** Returns an identifier for a top-level menu and registers the callback function f that returns an integer value corresponding to the menu entry selected.

## 4.5 Transformations

**glMatrixMode(GL_PROJECTION):** Here the mode will be projection mode, specifies subsequent transformation matrix to an identity matrix.

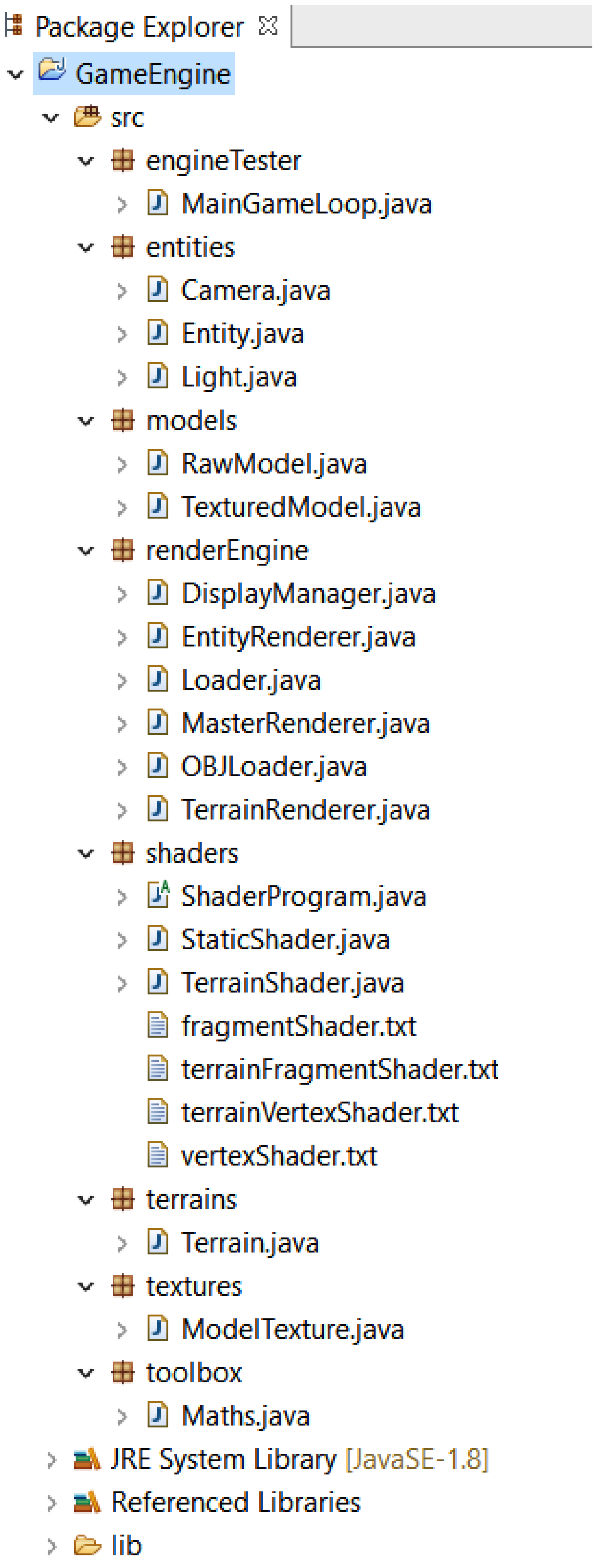
**glLoadIdentity():** Set the current transformation matrix to an identity stack corresponding to the current matrix mode.

## 4.6 Viewing.

**gluOrtho2D(left,right,bottom,top):** It defines a two dimensional viewing rectangle in the plane z=0.

**glOrtho(left, right, bottom, top, nearVal, farVal):** The glOrtho function describes a perspective matrix that produces a parallel projection. The (left, bottom, near) and (right, top, near) parameters specify the points on the near clipping plane that are mapped to the lower-left and upper-right corners of the window, respectively, assuming that the eye is located at (0, 0, 0). The far parameter specifies the location of the far clipping plane.

## 4.7 Project Structure



```
Package Explorer
GameEngine
  src
    engineTester
      MainGameLoop.java
    entities
      Camera.java
      Entity.java
      Light.java
    models
      RawModel.java
      TexturedModel.java
    renderEngine
      DisplayManager.java
      EntityRenderer.java
      Loader.java
      MasterRenderer.java
      OBJLoader.java
      TerrainRenderer.java
    shaders
      ShaderProgram.java
      StaticShader.java
      TerrainShader.java
      fragmentShader.txt
      terrainFragmentShader.txt
      terrainVertexShader.txt
      vertexShader.txt
    terrains
      Terrain.java
    textures
      ModelTexture.java
    toolbox
      Maths.java
  JRE System Library [JavaSE-1.8]
  Referenced Libraries
  lib
```

**FIG 4.3 Project structure**

## 4.8 Main Function

The main function must be called before any other GLUT/OpenGL calls. Execution of any program starts at main function. The main function gets argument argc and argv, ie. count and the pointer to the string array of arguments. It is from this place that the execution begins and calls the various user defined functions on the user defined objects.

```
public static void main(String[] args) {
        DisplayManager.createDisplay();
        Loader loader = new Loader();
        Light light = new Light(new Vector3f(20000, 20000, 2000), new Vector3f(1, 1,
1));
        Terrain terrain1 = new Terrain(-1, -1, loader, new
ModelTexture(loader.loadTexture("grass")));
        Terrain terrain2 = new Terrain(0, -1, loader, new
ModelTexture(loader.loadTexture("grass")));
        Camera camera = new Camera();
        MasterRenderer renderer = new MasterRenderer();

        while(!Display.isCloseRequested()) {
                camera.move();
                renderer.processTerrain(terrain1);
                renderer.processTerrain(terrain2);
                renderer.render(light, camera);
                DisplayManager.updateDisplay();
        }
        renderer.cleanUp();
        loader.cleanUp();
        DisplayManager.closeDisplay();
}
```

# CHAPTER 5

## SOURCE CODE

**MainGameLoop.java**

```
public class MainGameLoop {
	public static void main(String[] args) {
		DisplayManager.createDisplay();
		Loader loader = new Loader();
		RawModel model = OBJLoader.loadObjModel("tree", loader);
		TexturedModel texturedModel = new TexturedModel(model, new
ModelTexture(loader.loadTexture("tree")));
		List<Entity> entities = new ArrayList<Entity>();
		Random random = new Random();
		for(int i = 0; i < 500; i++){
			entities.add(new Entity(texturedModel, new
Vector3f(random.nextFloat()*800 - 400, 0, random.nextFloat() * -600), 0, 0, 0, 3));
		}

		Light light = new Light(new Vector3f(20000, 20000, 2000), new
Vector3f(1, 1, 1));
		Terrain terrain1 = new Terrain(-1, -1, loader, new
ModelTexture(loader.loadTexture("grass")));
		Terrain terrain2 = new Terrain(0, -1, loader, new
ModelTexture(loader.loadTexture("grass")));
		Camera camera = new Camera();
		MasterRenderer renderer = new MasterRenderer();

		while(!Display.isCloseRequested()) {
			camera.move();
			renderer.processTerrain(terrain1);
			renderer.processTerrain(terrain2);
			for(Entity entity:entities) {
				renderer.processEntity(entity);
		}
			renderer.render(light, camera);
			DisplayManager.updateDisplay();
		}
		renderer.cleanUp();
		loader.cleanUp();
		DisplayManager.closeDisplay();
```

```
        }
}
```

**Camera.java**

```
public class Camera {
        private Vector3f position = new Vector3f(0, 5, 0);
        private float pitch = 10;     // high or low
        private float yaw;            // left or right
        private float roll;           // titled

        public Camera() {}
        public void move() {
                if (Keyboard.isKeyDown(Keyboard.KEY_UP)) {
                        position.z -= 0.2f;
                }
                if (Keyboard.isKeyDown(Keyboard.KEY_DOWN)) {
                        position.z += 0.2f;
                }
                if (Keyboard.isKeyDown(Keyboard.KEY_RIGHT)) {
                        position.x += 0.2f;
                }
                if (Keyboard.isKeyDown(Keyboard.KEY_LEFT)) {
                        position.x -= 0.2f;
                }
                if (Keyboard.isKeyDown(Keyboard.KEY_U)) {
                        position.y += 0.2f;
                }
                if (Keyboard.isKeyDown(Keyboard.KEY_D)) {
                        position.y -= 0.2f;
                }
        }
        public Vector3f getPosition() {
                return position;
        }
        public float getPitch() {
                return pitch;
        }
        public float getYaw() {
                return yaw;
        }
        public float getRoll() {
                return roll;
        }
}
```

## DisplayManager.java

```
public class DisplayManager {
        private static final int WIDTH = 1280;
        private static final int HEIGHT = 720;
        private static final int FPS_CAP = 120;
        private static final String TITLE = "Simple Terrain";
        public static void createDisplay() {
                ContextAttribs attribs = new ContextAttribs(3, 2)
                                .withForwardCompatible(true)
                                .withProfileCore(true);
                try {
                        Display.setDisplayMode(new DisplayMode(WIDTH, HEIGHT));
                        Display.create(new PixelFormat(), attribs);
                        Display.setTitle(TITLE);
                } catch (LWJGLException e) {
                        e.printStackTrace();
                }
                GL11.glViewport(0, 0, WIDTH, HEIGHT);
        }
        public static void updateDisplay() {
                Display.sync(FPS_CAP);
                Display.update();
        }
        public static void closeDisplay() {
                Display.destroy();
        }
}
```

## EntityRenderer.java

```
public class EntityRenderer {
        private StaticShader shader;

        public EntityRenderer(StaticShader shader, Matrix4f projectionMatrix) {
                this.shader = shader;
                shader.start();
                shader.loadProjectionMatrix(projectionMatrix);
                shader.stop();
        }
```

```
	public void render(Map<TexturedModel, List<Entity>> entities) {
		for (TexturedModel model : entities.keySet()) {
			prepareTexturedModel(model);
			List<Entity> batch = entities.get(model);
			for (Entity entity : batch) {
				prepareInstance(entity);
				GL11.glDrawElements(GL11.GL_TRIANGLES,
model.getRawModel().getVertexCount(), GL11.GL_UNSIGNED_INT, 0);
			}
			unbindTexturedModel();
		}
	}
	private void prepareTexturedModel(TexturedModel model) {
		RawModel rawModel = model.getRawModel();
		GL30.glBindVertexArray(rawModel.getVaoID());
		GL20.glEnableVertexAttribArray(0);
		GL20.glEnableVertexAttribArray(1);
		GL20.glEnableVertexAttribArray(2);
		ModelTexture texture = model.getTexture();
		shader.loadShineVariable(texture.getShineDamper(),
texture.getReflectivity());
		GL13.glActiveTexture(GL13.GL_TEXTURE0);
		GL11.glBindTexture(GL11.GL_TEXTURE_2D, model.getTexture().getID());
	}
	private void unbindTexturedModel() {
		GL20.glDisableVertexAttribArray(0);
		GL20.glDisableVertexAttribArray(1);
		GL20.glDisableVertexAttribArray(2);
		GL30.glBindVertexArray(0);
	}
	private void prepareInstance(Entity entity) {
		Matrix4f transformationMatrix =
Maths.createTransformationMatrix(entity.getPosition(), entity.getRotX(),
				entity.getRotY(), entity.getRotZ(),
entity.getScale());
		shader.loadTransformationMatrix(transformationMatrix);
	}
}
```

**MasterRenderer.java**

```
public class MasterRenderer {
	private static final float FOV = 70;
	private static final float NEAR_PLANE = 0.1f;
	private static final float FAR_PLANE = 1000;
	private Matrix4f projectionMatrix;
	private StaticShader shader = new StaticShader();
	private EntityRenderer renderer;
	private TerrainRenderer terrainRenderer;
	private TerrainShader terrainShader = new TerrainShader();
	private Map<TexturedModel, List<Entity>> entities = new
HashMap<TexturedModel, List<Entity>>();
	private List<Terrain> terrains = new ArrayList<Terrain>();
	public MasterRenderer() {
		GL11.glEnable(GL11.GL_CULL_FACE);
		GL11.glCullFace(GL11.GL_BACK);
		createProjectionMatrix();
		renderer = new EntityRenderer(shader, projectionMatrix);
		terrainRenderer = new TerrainRenderer(terrainShader,
projectionMatrix);
	}
	public void render(Light sun, Camera camera) {
		prepare();
		shader.start();
		shader.loadLight(sun);
		shader.loadViewMatrix(camera);
		renderer.render(entities);
		shader.stop();
		terrainShader.start();
		terrainShader.loadLight(sun);
		terrainShader.loadViewMatrix(camera);
		terrainRenderer.render(terrains);
		terrainShader.stop();
		terrains.clear();
		entities.clear();
	}
	public void processTerrain(Terrain terrain) {
		terrains.add(terrain);
	}
	public void processEntity(Entity entity) {
		TexturedModel entityModel = entity.getModel();
		List<Entity> batch = entities.get(entityModel);
		if (batch != null) {
			batch.add(entity);
		} else {
			List<Entity> newBatch = new ArrayList<Entity>();
			newBatch.add(entity);
```

```
				entities.put(entityModel, newBatch);
			}
		}
	public void cleanUp() {
		shader.cleanUp();
		terrainShader.cleanUp();
	}
	public void prepare() {
		GL11.glEnable(GL11.GL_DEPTH_TEST);
		GL11.glClear(GL11.GL_COLOR_BUFFER_BIT | GL11.GL_DEPTH_BUFFER_BIT);
		GL11.glClearColor(0.49f, 89f, 0.98f, 1);
	}
	private void createProjectionMatrix() {
		float aspectRatio = (float) Display.getWidth() / (float) 
Display.getHeight();
		float y_scale = (float) ((1f / Math.tan(Math.toRadians(FOV / 2f))) * 
aspectRatio);
		float x_scale = y_scale / aspectRatio;
		float frustum_length = FAR_PLANE - NEAR_PLANE;
		projectionMatrix = new Matrix4f();
		projectionMatrix.m00 = x_scale;
		projectionMatrix.m11 = y_scale;
		projectionMatrix.m22 = -((FAR_PLANE + NEAR_PLANE) / frustum_length);
		projectionMatrix.m23 = -1;
		projectionMatrix.m32 = -((2 * NEAR_PLANE * FAR_PLANE) / 
frustum_length);
		projectionMatrix.m33 = 0;
	}
}
```

## OBJLoader.java

```
public class OBJLoader {
	public static RawModel loadObjModel(String fileName, Loader loader) {
		FileReader fr = null;
		try {
			fr = new FileReader(new File("res/models/" + fileName + 
".obj"));
		} catch (FileNotFoundException e) {
			System.err.println("Couldn't load file!");
			e.printStackTrace();
		}
		BufferedReader reader = new BufferedReader(fr);
```

```
        String line;
        List<Vector3f> vertices = new ArrayList<Vector3f>();
        List<Vector2f> textures = new ArrayList<Vector2f>();
        List<Vector3f> normals = new ArrayList<Vector3f>();
        List<Integer> indices = new ArrayList<Integer>();
        float[] verticesArray = null;
        float[] normalsArray = null;
        float[] textureArray = null;
        int[] indicesArray = null;
        try {

            while (true) {
                line = reader.readLine();
                String[] currentLine = line.split(" ");
                if (line.startsWith("v ")) {
                    Vector3f vertex = new
Vector3f(Float.parseFloat(currentLine[1]), Float.parseFloat(currentLine[2]),
Float.parseFloat(currentLine[3]));
                    vertices.add(vertex);
                } else if (line.startsWith("vt ")) {
                    Vector2f texture = new
Vector2f(Float.parseFloat(currentLine[1]), Float.parseFloat(currentLine[2]));
                    textures.add(texture);
                } else if (line.startsWith("vn ")) {
                    Vector3f normal = new
Vector3f(Float.parseFloat(currentLine[1]), Float.parseFloat(currentLine[2]),
Float.parseFloat(currentLine[3]));
                    normals.add(normal);
                } else if (line.startsWith("f ")) {
                    textureArray = new float[vertices.size() * 2];
                    normalsArray = new float[vertices.size() * 3];
                    break;
                }
            }
            while (line != null) {
                if (!line.startsWith("f ")) {
                    line = reader.readLine();
                    continue;
                }
                String[] currentLine = line.split(" ");
                String[] vertex1 = currentLine[1].split("/");
                String[] vertex2 = currentLine[2].split("/");
                String[] vertex3 = currentLine[3].split("/");
```

```
                             processVertex(vertex1, indices, textures, normals,
textureArray, normalsArray);
                             processVertex(vertex2, indices, textures, normals,
textureArray, normalsArray);
                             processVertex(vertex3, indices, textures, normals,
textureArray, normalsArray);
                             line = reader.readLine();
                      }
                      reader.close();
               } catch (Exception e) {
                      e.printStackTrace();
               }
               verticesArray = new float[vertices.size() * 3];
               indicesArray = new int[indices.size()];
               int vertexPointer = 0;
               for (Vector3f vertex : vertices) {
                      verticesArray[vertexPointer++] = vertex.x;
                      verticesArray[vertexPointer++] = vertex.y;
                      verticesArray[vertexPointer++] = vertex.z;
               }
               for (int i = 0; i < indices.size(); i++) {
                      indicesArray[i] = indices.get(i);
               }
               return loader.loadToVAO(verticesArray, textureArray, normalsArray,
indicesArray);
       }
       private static void processVertex(String[] vertexData, List<Integer>
indices, List<Vector2f> textures, List<Vector3f> normals, float[] textureArray,
float[] normalsArray) {
               int currentVertexPointer = Integer.parseInt(vertexData[0]) - 1;
               indices.add(currentVertexPointer);
               Vector2f currentTex = textures.get(Integer.parseInt(vertexData[1]) -
1);
               textureArray[currentVertexPointer * 2] = currentTex.x;
               textureArray[currentVertexPointer * 2 + 1] = 1 - currentTex.y;
               Vector3f currentNorm = normals.get(Integer.parseInt(vertexData[2]) -
1);
               normalsArray[currentVertexPointer * 3] = currentNorm.x;
               normalsArray[currentVertexPointer * 3 + 1] = currentNorm.y;
               normalsArray[currentVertexPointer * 3 + 2] = currentNorm.z;
       }
}
```

**TerrainRenderer.java**

```
public class TerrainRenderer {
	private TerrainShader shader;
	public TerrainRenderer(TerrainShader shader, Matrix4f projectionMatrix) {
		this.shader = shader;
		shader.start();
		shader.loadProjectionMatrix(projectionMatrix);
		shader.stop();
	}
	public void render(List<Terrain> terrains) {
		for (Terrain terrain : terrains) {
			prepareTerrain(terrain);
			loadModelMatrix(terrain);
			GL11.glDrawElements(GL11.GL_TRIANGLES,
terrain.getModel().getVertexCount(), GL11.GL_UNSIGNED_INT, 0);
			unbindTexturedModel();
		}
	}
	private void prepareTerrain(Terrain terrain) {
		RawModel rawModel = terrain.getModel();
		GL30.glBindVertexArray(rawModel.getVaoID());
		GL20.glEnableVertexAttribArray(0);
		GL20.glEnableVertexAttribArray(1);
		GL20.glEnableVertexAttribArray(2);
		ModelTexture texture = terrain.getTexture();
		shader.loadshineVariable(texture.getShineDamper(),
texture.getReflectivity());
		GL13.glActiveTexture(GL13.GL_TEXTURE0);
		GL11.glBindTexture(GL11.GL_TEXTURE_2D, texture.getID());
	}
	private void unbindTexturedModel() {
		GL20.glDisableVertexAttribArray(0);
		GL20.glDisableVertexAttribArray(1);
		GL20.glDisableVertexAttribArray(2);
		GL30.glBindVertexArray(0);
	}
	private void loadModelMatrix(Terrain terrain) {
		Matrix4f transformationMatrix = Maths.createTransformationMatrix(new
Vector3f(terrain.getX(), 0, terrain.getZ()), 0, 0, 0, 1);
		shader.loadTransformationMatrix(transformationMatrix);
	}
}
```

**ShaderProgram.java**

```
public abstract class ShaderProgram {
	private int programID;
	private int vertexShaderID;
	private int fragmentShaderID;
	private static FloatBuffer matrixBuffer = BufferUtils.createFloatBuffer(16);
	public ShaderProgram(String vertexFile, String fragmentFile) {
		vertexShaderID = loadShader(vertexFile, GL20.GL_VERTEX_SHADER);
		fragmentShaderID = loadShader(fragmentFile, GL20.GL_FRAGMENT_SHADER);
		programID = GL20.glCreateProgram();
		GL20.glAttachShader(programID, vertexShaderID);
		GL20.glAttachShader(programID, fragmentShaderID);
		bindAttributes();
		GL20.glLinkProgram(programID);
		GL20.glValidateProgram(programID);
		getAllUniformLocations();
	}
	protected abstract void getAllUniformLocations();
	protected int getUniformLocation(String uniformName) {
		return GL20.glGetUniformLocation(programID, uniformName);
	}
	public void start() {
		GL20.glUseProgram(programID);
	}
	public void stop() {
		GL20.glUseProgram(0);
	}
	public void cleanUp() {
		stop();
		GL20.glDetachShader(programID, vertexShaderID);
		GL20.glDetachShader(programID, fragmentShaderID);
		GL20.glDeleteShader(vertexShaderID);
		GL20.glDeleteShader(fragmentShaderID);
		GL20.glDeleteProgram(programID);
	}
	protected abstract void bindAttributes();
	protected void bindAttribute(int attribute, String variableName) {
		GL20.glBindAttribLocation(programID, attribute, variableName);
	}
	protected void loadFloat(int location, float value) {
		GL20.glUniform1f(location, value);
	}
	protected void loadInt(int location, int value) {
		GL20.glUniform1i(location, value);
	}
	protected void loadVector(int location, Vector3f vector) {
		GL20.glUniform3f(location, vector.x, vector.y, vector.z);
	}
	protected void loadVector(int location, Vector4f vector) {
		GL20.glUniform4f(location, vector.x, vector.y, vector.z, vector.w);
	}
	protected void load2DVector(int location, Vector2f vector) {
		GL20.glUniform2f(location, vector.x, vector.y);
	}
	protected void loadBoolean(int location, boolean value) {
		float toLoad = 0;
		if (value) {
			toLoad = 1;
		}
		GL20.glUniform1f(location, toLoad);
	}
	protected void loadMatrix(int location, Matrix4f matrix) {
		matrix.store(matrixBuffer);
		matrixBuffer.flip();
		GL20.glUniformMatrix4(location, false, matrixBuffer);
	}
	private static int loadShader(String file, int type) {
		StringBuilder shaderSource = new StringBuilder();
		try {
			BufferedReader reader = new BufferedReader(new
FileReader(file));
			String line;
			while ((line = reader.readLine()) != null) {
				shaderSource.append(line).append("//\n");
			}
			reader.close();
		} catch (IOException e) {
			e.printStackTrace();
			System.exit(-1);
		}
		int shaderID = GL20.glCreateShader(type);
		GL20.glShaderSource(shaderID, shaderSource);
		GL20.glCompileShader(shaderID);
		if (GL20.glGetShaderi(shaderID, GL20.GL_COMPILE_STATUS) ==
GL11.GL_FALSE) {
                        System.out.println(GL20.glGetShaderInfoLog(shaderID, 500));
                        System.err.println("Could not compile shader!");
                        System.exit(-1);
                }
                return shaderID;
        }
}
```

**StaticShader.java**

```
public class StaticShader extends ShaderProgram {
        private static final String VERTEX_FILE = "src/shaders/vertexShader.txt";
        private static final String FRAGMENT_FILE =
"src/shaders/fragmentShader.txt";
        private int locationTranformationMatrix;
        private int locationProjectionMatrix;
        private int locationViewMatrix;
        private int locationLightPosition;
        private int locationLightColour;
        private int locationShineDamper;
        private int locationReflectivity;
        public StaticShader() {
                super(VERTEX_FILE, FRAGMENT_FILE);
        }
        @Override
        protected void bindAttributes() {
                super.bindAttribute(0, "position");
                super.bindAttribute(1, "textureCoords");
                super.bindAttribute(2, "normal");
        }
        @Override
        protected void getAllUniformLocations() {
                locationTranformationMatrix =
super.getUniformLocation("transformationMatrix");
                locationProjectionMatrix =
super.getUniformLocation("projectionMatrix");
                locationViewMatrix = super.getUniformLocation("viewMatrix");
                locationLightPosition = super.getUniformLocation("lightPosition");
                locationLightColour = super.getUniformLocation("lightColour");
                locationShineDamper = super.getUniformLocation("shineDamper");
                locationReflectivity = super.getUniformLocation("reflectivity");
        }
```

```
	public void loadTransformationMatrix(Matrix4f matrix) {
		super.loadMatrix(locationTranformationMatrix, matrix);
	}
	public void loadProjectionMatrix(Matrix4f projection) {
		super.loadMatrix(locationProjectionMatrix, projection);
	}
	public void loadViewMatrix(Camera camera) {
		Matrix4f viewMatrix = Maths.createViewMatrix(camera);
		super.loadMatrix(locationViewMatrix, viewMatrix);
	}
	public void loadLight(Light light) {
		super.loadVector(locationLightPosition, light.getPosition());
		super.loadVector(locationLightColour, light.getColour());
	}
	public void loadShineVariable(float damper, float reflectivity) {
		super.loadFloat(locationShineDamper, damper);
		super.loadFloat(locationReflectivity, reflectivity);
	}
}
```

## TerrainShader.java

```
public class TerrainShader extends ShaderProgram {
	private static final String VERTEX_FILE =
"src/shaders/terrainVertexShader.txt";
	private static final String FRAGMENT_FILE =
"src/shaders/terrainFragmentShader.txt";
	private int locationTransformationMatrix;
	private int locationProjectionMatrix;
	private int locationViewMatrix;
	private int locationLightPosition;
	private int locationLightColour;
	private int locationShineDamper;
	private int locationReflectivity;
	public TerrainShader() {
		super(VERTEX_FILE, FRAGMENT_FILE);
	}
	@Override
	protected void bindAttributes() {
		super.bindAttribute(0, "position");
		super.bindAttribute(1, "textureCoords");
		super.bindAttribute(2, "normal");
	}
	@Override
	protected void getAllUniformLocations() {
		locationTransformationMatrix = 
super.getUniformLocation("transformationMatrix");
		locationProjectionMatrix = 
super.getUniformLocation("projectionMatrix");
		locationViewMatrix = super.getUniformLocation("viewMatrix");
		locationLightPosition = super.getUniformLocation("lightPosition");
		locationLightColour = super.getUniformLocation("lightColour");
		locationShineDamper = super.getUniformLocation("shineDamper");
		locationReflectivity = super.getUniformLocation("reflectivity");
	}
	public void loadshineVariable(float damper, float reflectivity) {
		super.loadFloat(locationShineDamper, damper);
		super.loadFloat(locationReflectivity, reflectivity);
	}
	public void loadTransformationMatrix(Matrix4f matrix) {
		super.loadMatrix(locationTransformationMatrix, matrix);
	}
	public void loadLight(Light light) {
		super.loadVector(locationLightPosition, light.getPosition());
		super.loadVector(locationLightColour, light.getColour());
	}
	public void loadViewMatrix(Camera camera) {
		Matrix4f viewMatrix = Maths.createViewMatrix(camera);
		super.loadMatrix(locationViewMatrix, viewMatrix);
	}
	public void loadProjectionMatrix(Matrix4f projection) {
		super.loadMatrix(locationProjectionMatrix, projection);
	}
}
```

## Terrain.java

```
public class Terrain {
	private static final float SIZE = 800;
	private static final int VERTEX_COUNT = 128;
	private float x;
	private float z;
	private RawModel model;
	private ModelTexture texture;
	private RawModel generateTerrain(Loader loader) {
		int count = VERTEX_COUNT * VERTEX_COUNT;
		float[] vertices = new float[count * 3];
		float[] normals = new float[count * 3];
		float[] textureCoords = new float[count * 2];
		int[] indices = new int[6 * (VERTEX_COUNT - 1) * (VERTEX_COUNT * 1)];
		int vertexPointer = 0;
		for (int i = 0; i < VERTEX_COUNT; i++) {
			for (int j = 0; j < VERTEX_COUNT; j++) {
				vertices[vertexPointer * 3] = (float) j / ((float)
VERTEX_COUNT - 1) * SIZE;
				vertices[vertexPointer * 3 + 1] = 0;
				vertices[vertexPointer * 3 + 2] = (float) i / ((float)
VERTEX_COUNT - 1) * SIZE;
				normals[vertexPointer * 3] = 0;
				normals[vertexPointer * 3 + 1] = 1;
				normals[vertexPointer * 3 + 2] = 0;
				textureCoords[vertexPointer * 2] = (float) j /
((float) VERTEX_COUNT - 1);
				textureCoords[vertexPointer * 2 + 1] = (float) i /
((float) VERTEX_COUNT - 1);
				vertexPointer++;
			}
		}
		int pointer = 0;
		for (int gz = 0; gz < VERTEX_COUNT - 1; gz++) {
			for (int gx = 0; gx < VERTEX_COUNT - 1; gx++) {
				int topLeft = (gz * VERTEX_COUNT) + gx;
				int topRight = topLeft + 1;
				int bottomLeft = ((gz + 1) * VERTEX_COUNT) + gx;
				int bottomRight = bottomLeft + 1;
				indices[pointer++] = topLeft;
				indices[pointer++] = bottomLeft;
				indices[pointer++] = topRight;
				indices[pointer++] = topRight;
				indices[pointer++] = bottomLeft;
				indices[pointer++] = bottomRight;
			}
		}
		return loader.loadToVAO(vertices, textureCoords, normals, indices);
	}
}
```

# CHAPTER 6

## SNAPSHOTS

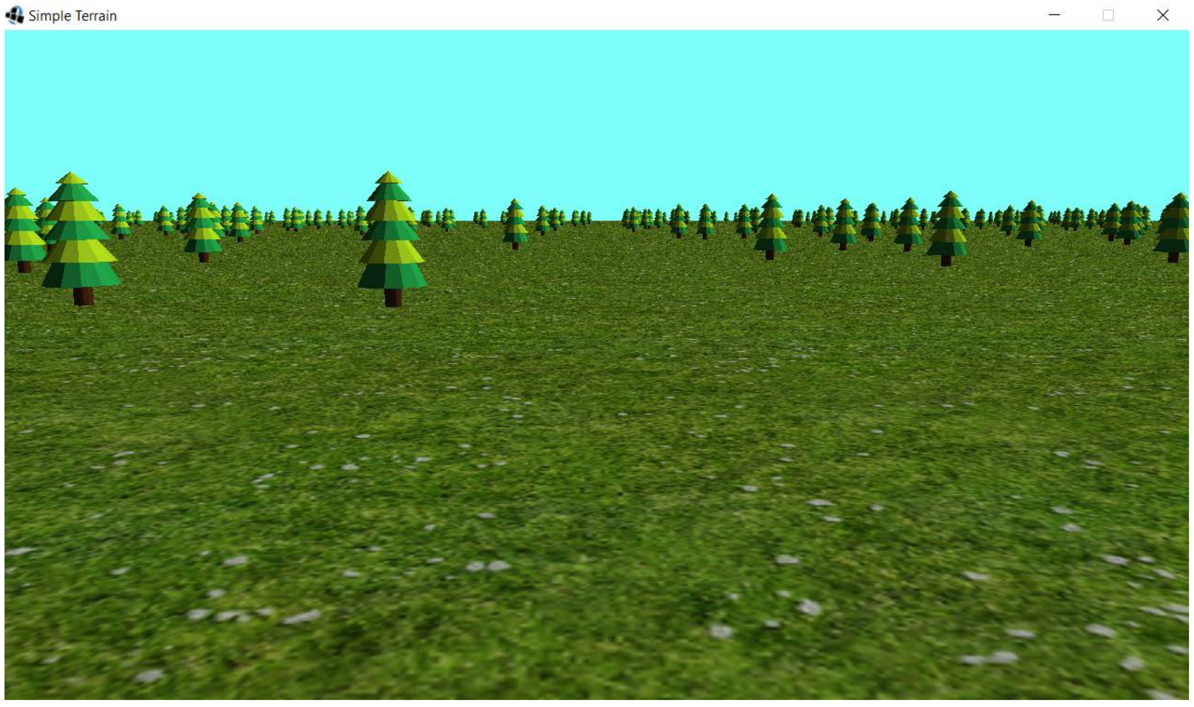


FIG 6.1 INITIAL POSITION OF THE SCENE

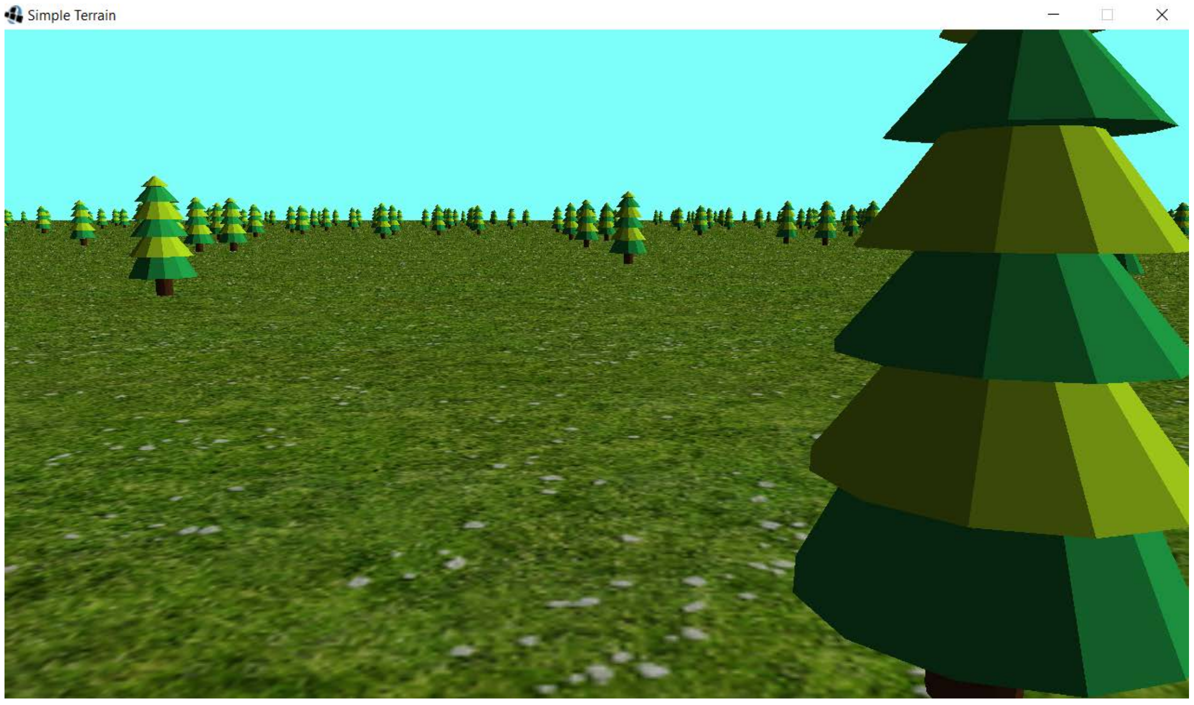


FIG 6.2 MOVING CLOSER TO THE SCENE

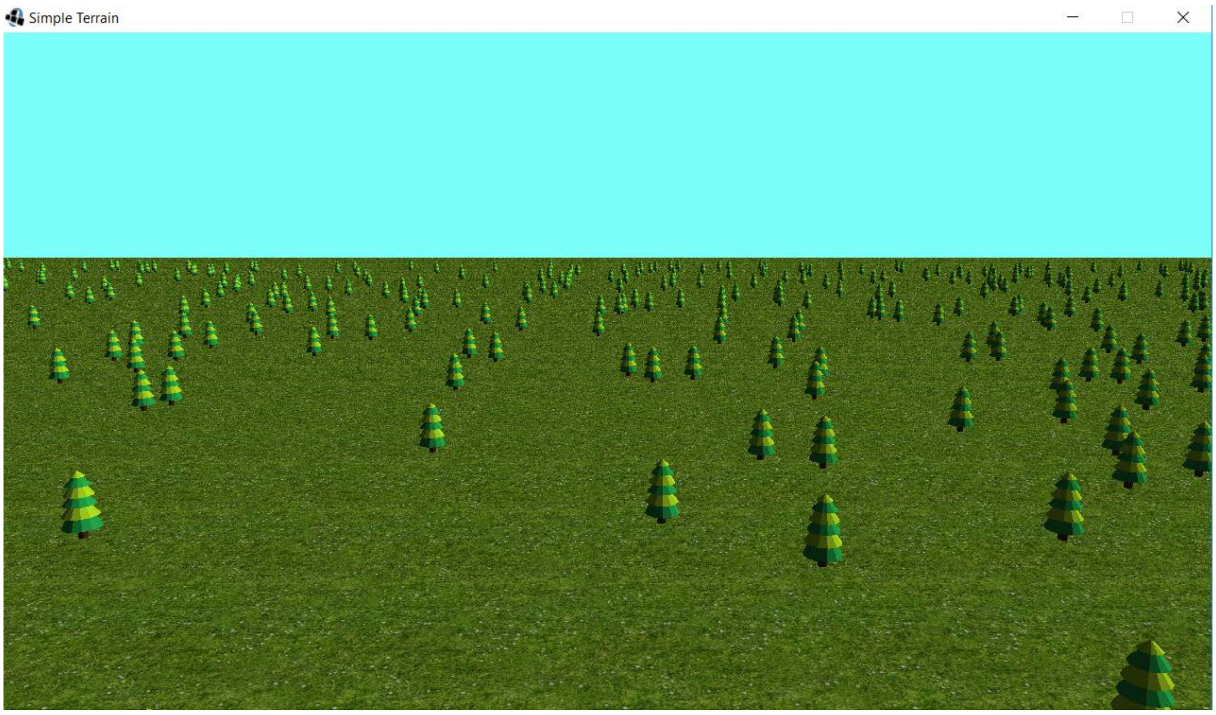


**FIG 6.3 MOVING UP FROM THE SCENE**

# CHAPTER 7

## CONCLUSION

An attempt has been made to develop an OpenGL graphics package, which meets necessary requirements of the users successfully. It enables us to learn about the basic concept in OpenGL graphics and graphics and know standard library graphics function and also to explore some other function. OpenGL graphics is a huge library which consists of numerous functions.

The various shapes at lower level or to simulate any real thing animation etc. at high level. This project has given us an insight into the use of Computer graphics. As we had to use many built-in and user defined functions, we have to managed to get a certain degree of familiarity with these functions and have now understood the power of these functions. We were able to comprehend the true nature of the most powerful tool graphics in OpenGL and have understood the reason why graphics is so powerful for programmers.

We can now converse with the certain degree of confidence about graphics in OpenGL. Finally, we have implemented this mini projection "Terrain Rendering" using OpenGL package. We would like to end by saying that doing this graphics project has been a memorable experience in which we have learned a lot, although there is a scope for further improvement. We got to know a lot of different applications of OpenGL while doing this project.

Expected output will be a terrain where the user can move freely using keyboard arrow keys.

## FURTHER SCOPE

Scope of further improvement:

- The scene can be made with a second person view where the user can be seen.
- Various lightening effects can be implemented to make it more attractive.
- Skybox can be used to make the scene larger than it is.

# BIBLIOGRAPHY

- **Books:**
  - **Computer Graphics** (OpenGL Version)

    Donald Hearn and Pauline Baker, 2nd edition, Pearson Education, 2003
  - **Interactive Computer Graphics**

    Edward Angel, 5th edition, Addison Wesley, 2009

- **Reference Websites:**
  - https://www.opengl.org/
  - https://www.learnopengl.com/
  - https://www.tutorialspoint.com/
  - https://www.lwjgl.org/