OpenGL Abandoned Amusement Park

Graphic Processing Project

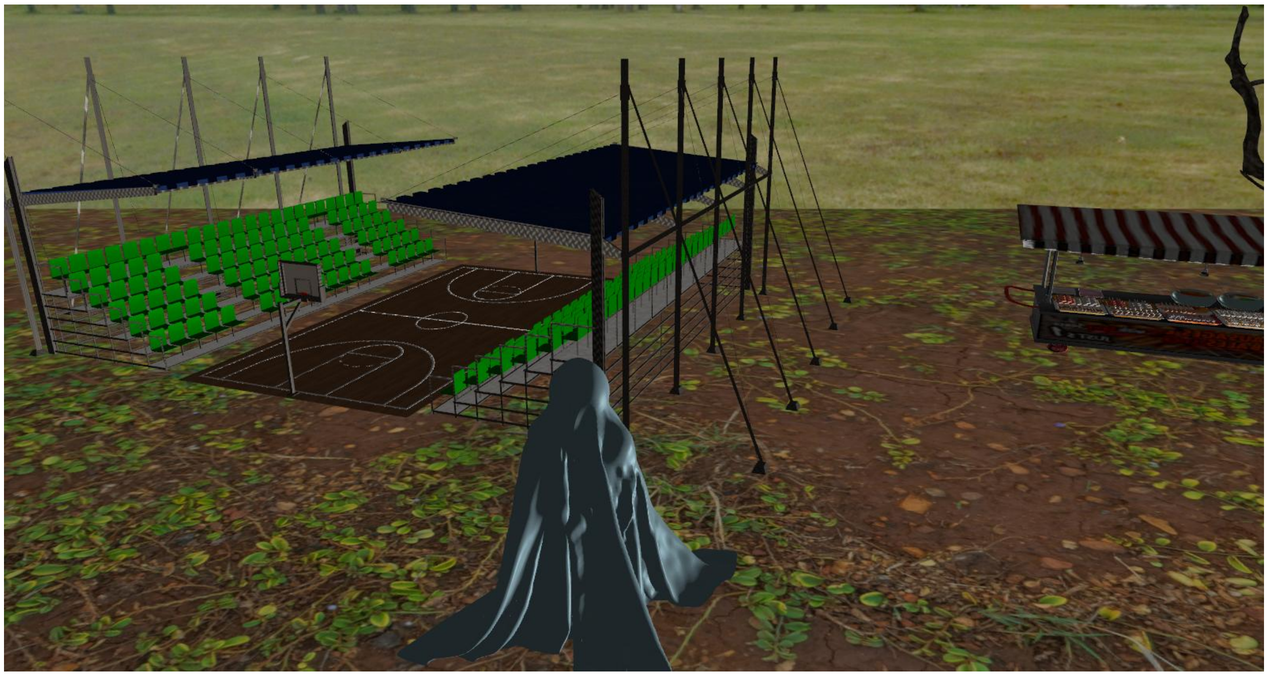
STUDENT: Pop Alexandra GROUP: 30435

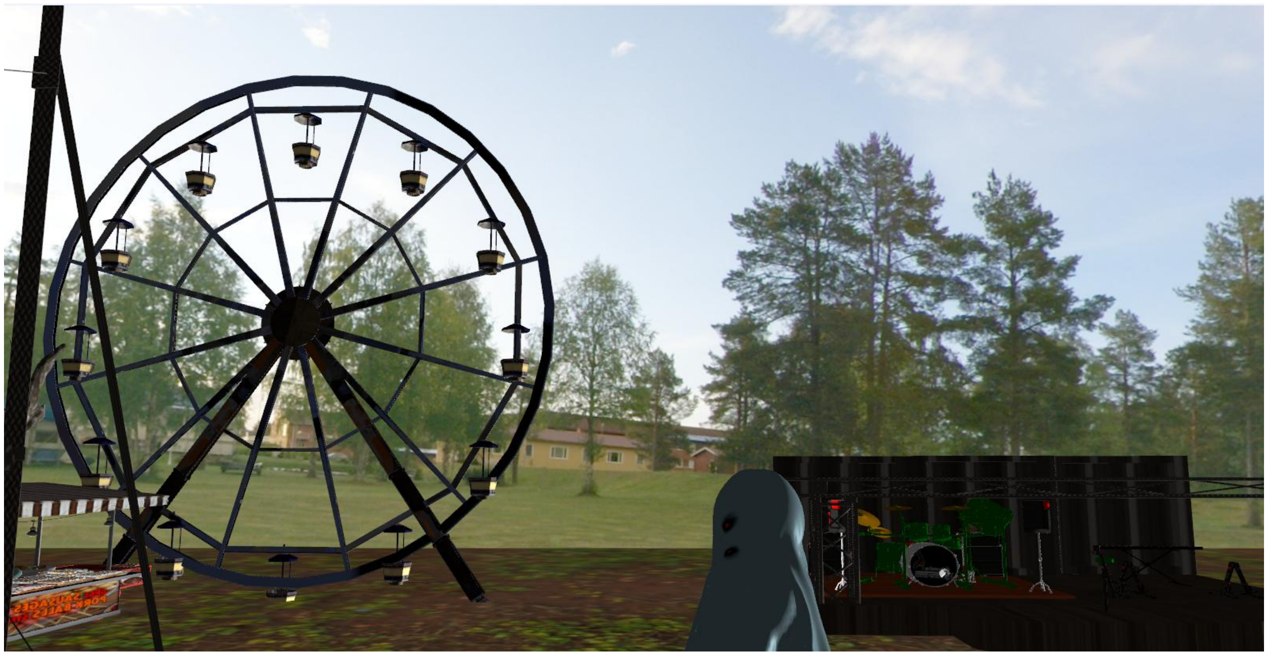
1. Contents

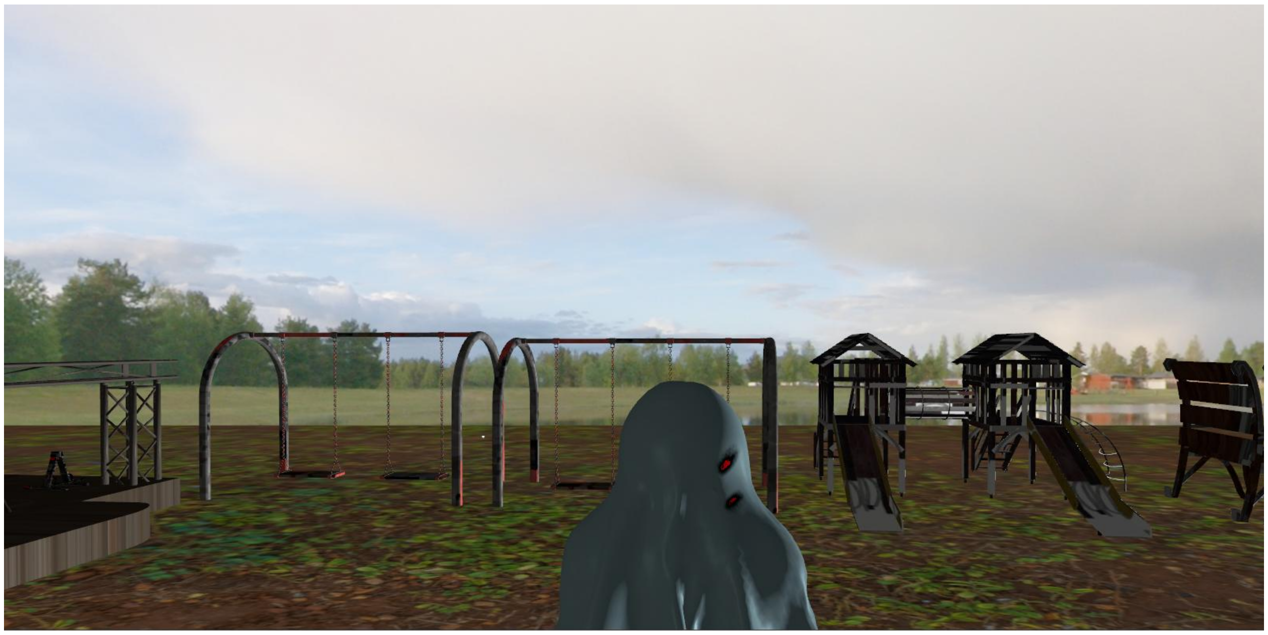
2. Subject Specification

This documentation describes a **3D computer graphics project** depicting an **abandoned amusement park** using OpenGL. The environment showcases a **mysterious, eerie** scene with **decayed rides, overgrown vegetation, broken attractions**. The primary focus is on realistic rendering techniques, lighting effects, weather simulation, and animation.

The scene is designed to evoke an eerie atmosphere, with **dynamic lighting, fog, and weather effects** that enhance the sense of desolation. The amusement park is haunted by a moving ghost. Be aware!







- Fig 1, 2, 3, 4: Scene overview showcasing most of the abandoned park

3. Scenario

3.1. Scene and Objects Description

The abandoned amusement park consists of various static and animated objects that create an immersive atmosphere. The scene includes:

- **A Ferris wheel**
- **A food truck with hanging lights**
- **A floating ghost entity**
- **A wooden stage with instruments on it (drums, piano, microphone, guitar support)**
- **Tables with chess sets on them and benches**
- **Environmental effects such as fog, rain, and dynamic lighting**
- **Old swings and slides**
- **A basketball field with stands**
- **An Old tree**
- **A skybox that supports day and night transitions**

The scene integrates real-time lighting adjustments, shadow mapping, and weather effects to enhance realism and immersion. Players can interact with various functionalities, including adjusting lighting conditions and activating different atmospheric effects.

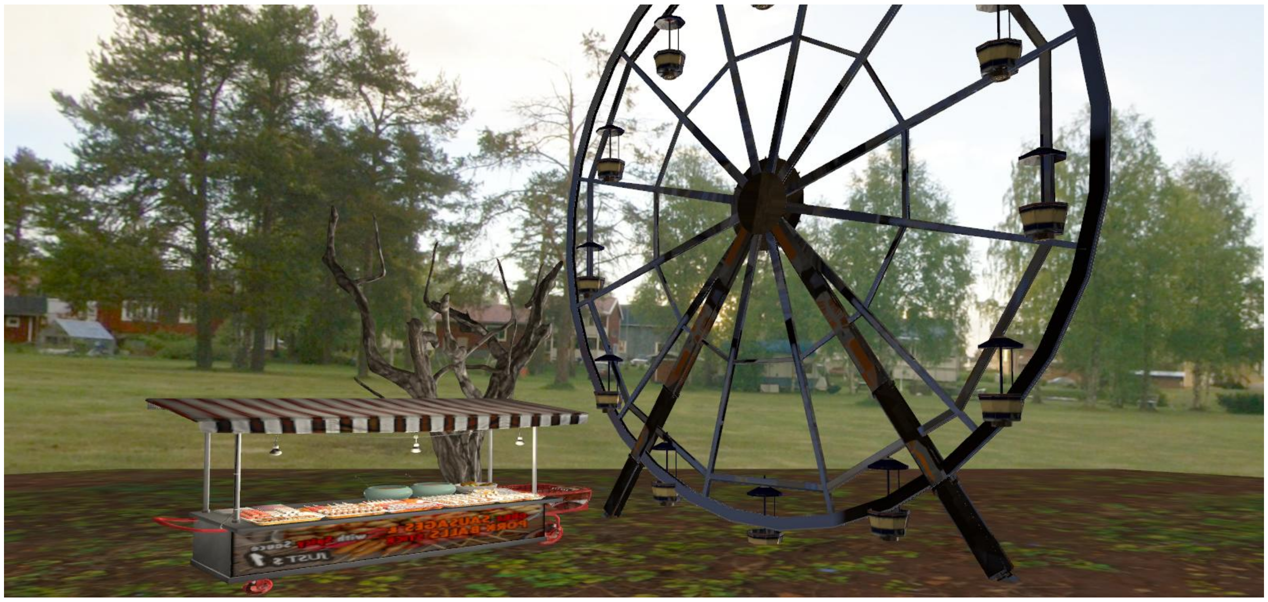
3.2. Functionalities

1. **Camera Control and Animation**

    - The user can move freely around the scene (forward, backward, sideways) and rotate the camera (mouse look).
    - A predefined **camera animation** moves the user's viewpoint along certain key positions in the scene, presenting important landmarks.
    - When pressing space you accelerate the camera movement speed by x4 - in order to move faster around checkpoints

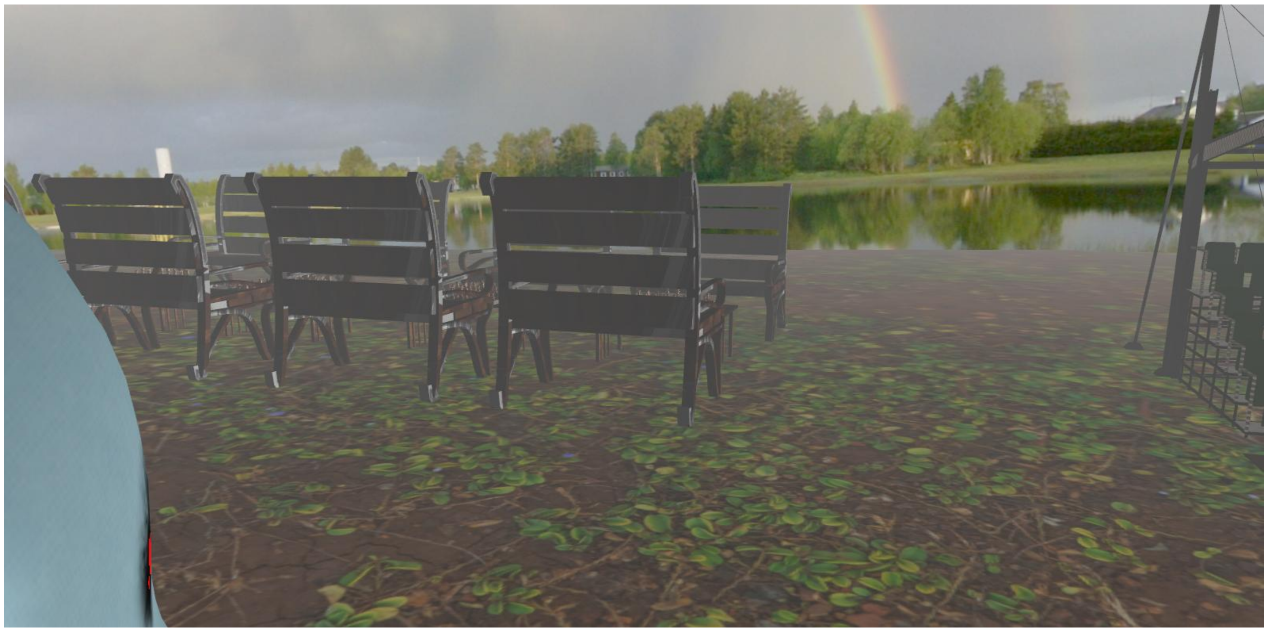
2. **Directional and Point Lights**

    - **Sunlight (directional light)** for the main illumination, its intensity is also controlled through the day/dusk mode, it’s the only light that creates the shadows in the scene
    - **Multiple point lights** on the food truck create additional lighting effects, they are all triggered at once and can not be triggered individually at the moment.



- Fig 5: Multiple point lights from the food truck lights-

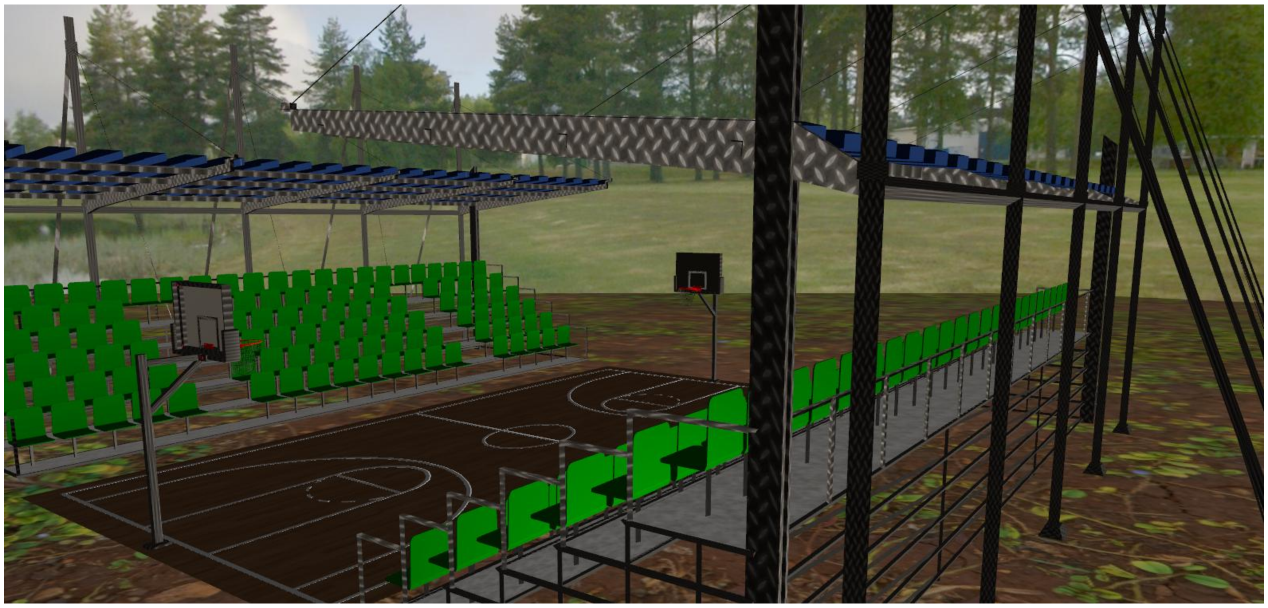
3. **Fog**
   - A **fog system** that can be toggled on/off, adding to the desert's harsh environment.
   - The fog is thick, on purpose, and it spans the whole map

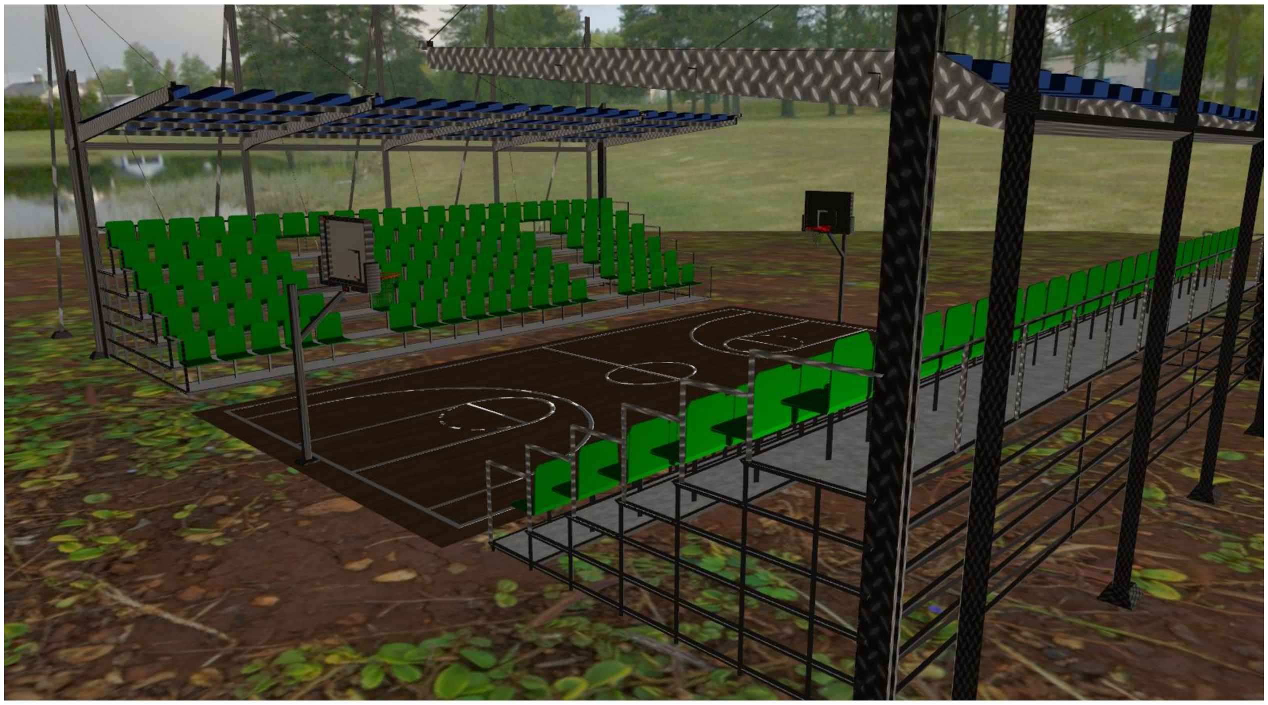


- Fig 6: Fog effect -

4. **Wind/Oasis Animation**

- A subtle **wind effect** distorts object vertices using a sinus/cosine function in the vertex shader, simulating heat haze or desert wind.
- The effect also generates waves in the water which look like it's rising above sea level

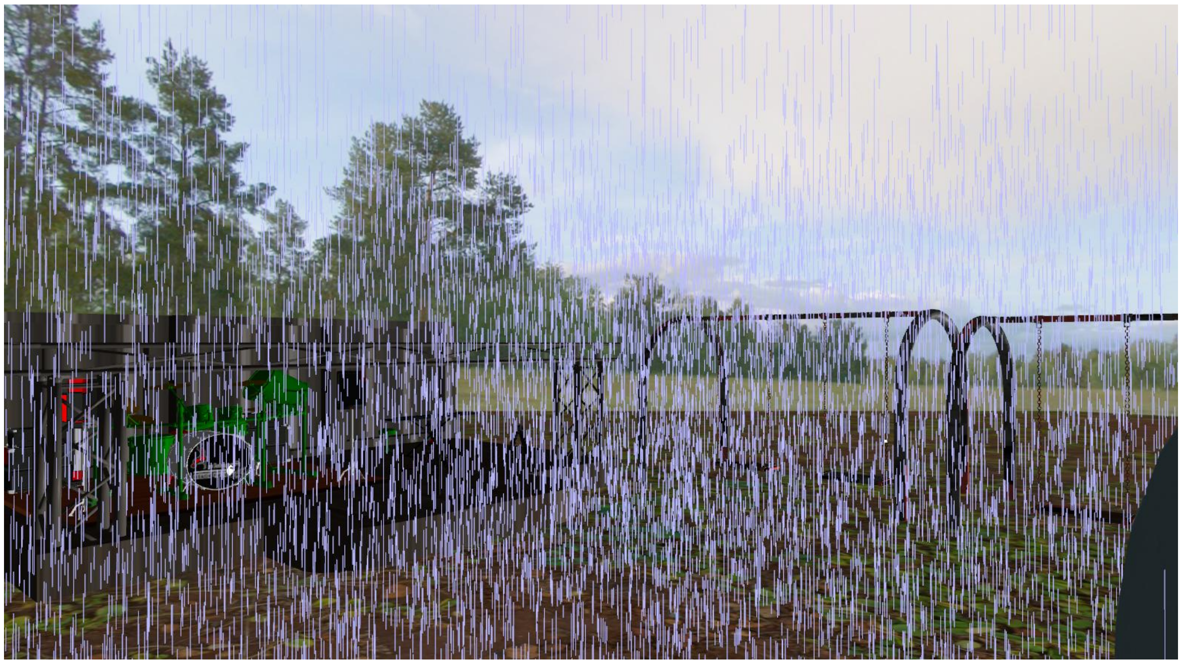




- Fig 7,8: Wind effect causing some changes -

5. **Rain**

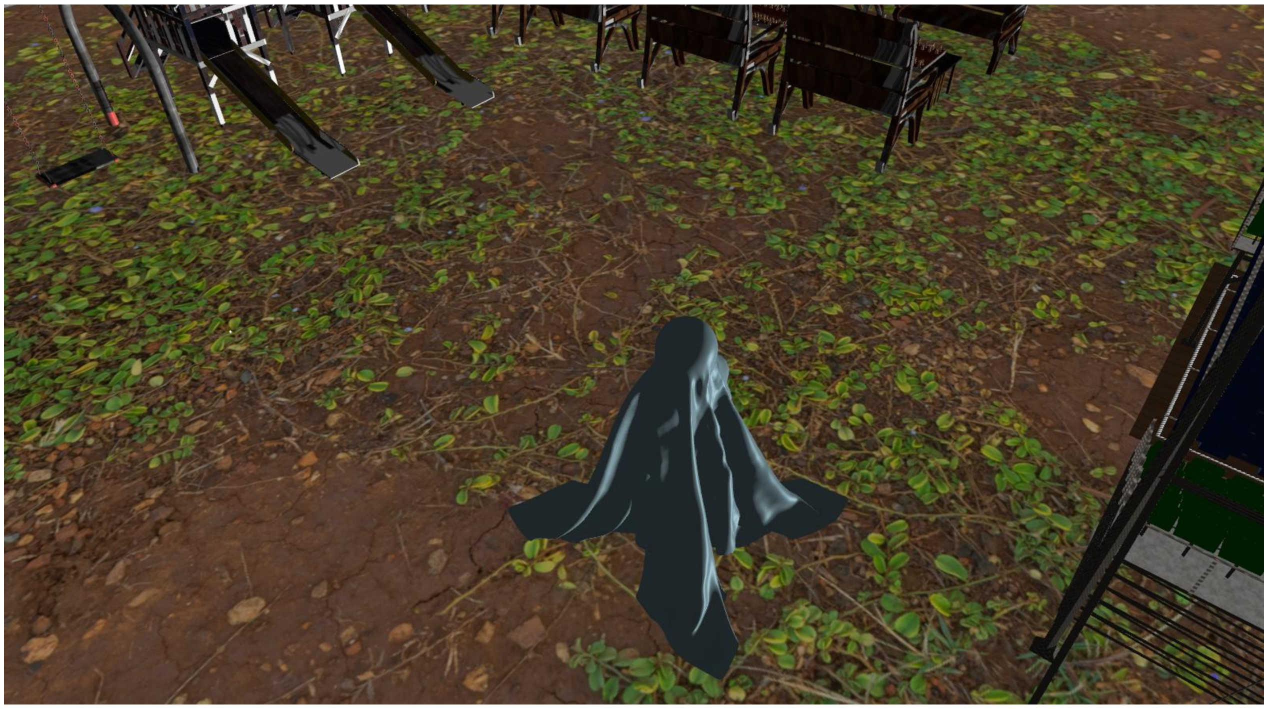
- A **rain system** that spawns numerous raindrops falling from the sky, they have varying speeds and different spawn points



- Fig 9: Rain effect -

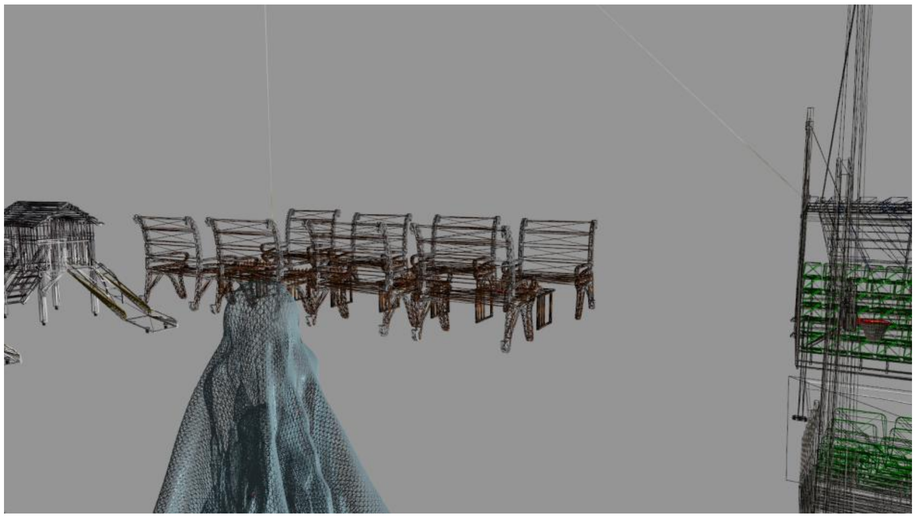
6. **Day/Night Cycle (Skybox Switching)**
   - The scene allows switching between **day mode** (bright field with clouds and a lake) and **dusk/night mode** (creepy forest with stairs).
   - The intensity and color of the sunlight are adjusted accordingly.

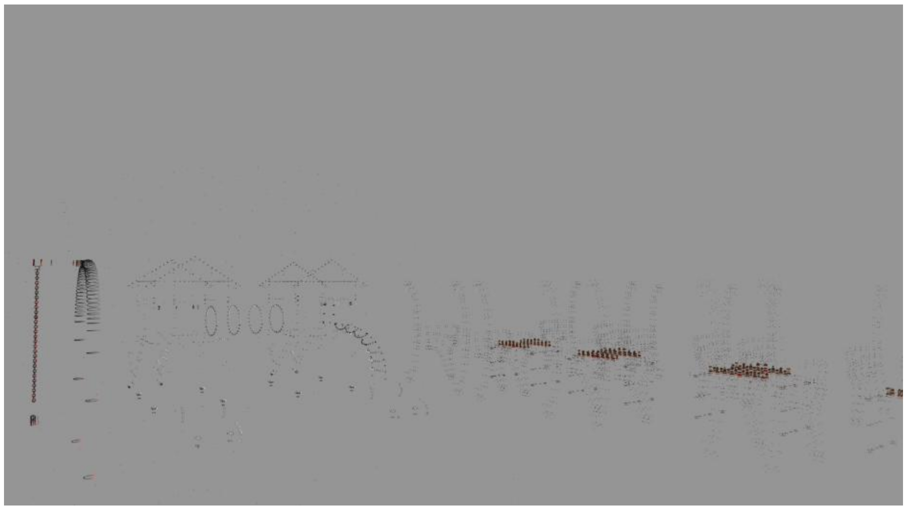
7. **Animated Ghost**
   - The ghost moves using scaling, rotation and transformation creating a pulsating effect

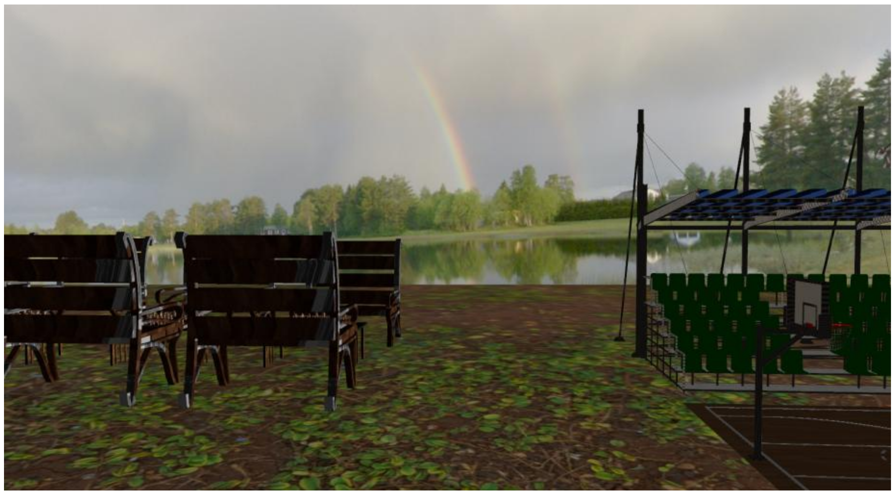


- Fig 10: Animated ghost-

8. **Multiple options for visualizing the surface (solid, wireframe, or polygonal)**







- Fig 11, 12, 13: View from the three vision modes -

4. Implementation Details

4.1. Per Feature Design & Implementation

Camera and Animation

The camera system leverages the **gps::Camera** class to handle movement and rotations. It maintains three primary vectors: **cameraPosition**, **cameraFrontDirection**, and **cameraUpDirection**. Smooth camera animations are achieved by interpolating positions along a predefined path using linear interpolation.

These animations provide a cinematic overview of the scene, transitioning smoothly between key points.

Multiple Light Sources

My implementation uses **directional light** (e.g., sunlight) and **point lights** (localized sources such as lamps) to create a dynamic scene.

1. **Directional Light**:
   - Simulates a distant light source with parallel rays, ideal for effects like sunlight.
   - Provides ambient, diffuse, and specular components.
   - Configured using direction and intensity, it interacts with shadows.
2. **Point Lights**:
   - These localized light sources illuminate nearby areas and fade with distance.
   - Attenuation is calculated using constant, linear, and quadratic factors to ensure realistic light falloff.

The **Phong lighting model** is applied for each point of light, combining:

1. **Ambient** light for consistent base illumination.
2. **Diffuse** light for angle-based intensity changes.
3. **Specular** light for reflective highlights.
4. Attenuation to ensure light fades realistically with distance.

Fog Effect

Fog introduces atmospheric depth and realism by blending fragment colors with the fog color based on distance. The effect is computed using an **exponential formula** controlled by density parameters. This simulates environmental phenomena such as mist in forests or haze in vast, open areas. Most of the implementation was taken from the last lab session we had on-site at the university. (lab 11)

Shadow Mapping

Shadow mapping is pivotal in achieving realistic lighting and depth in 3D scenes. It involves two stages:

1. **Depth Map Generation**: The scene is rendered from the light source's perspective to create a depth map. This map records the distance of the nearest objects to the light source for every direction. This process essentially captures which parts of the scene are visible to the light, forming the basis for shadow determination.

2. **Shadow Computation**: During the rendering pass, the depth of each fragment (pixel) is transformed into light space and compared to the depth map. If the fragment's depth exceeds the

recorded depth value, it is in shadow. This mechanism creates the realistic appearance of shadows by identifying occluded areas.

Key aspects of shadow mapping include:

- **Bias Correction**: A small bias is applied to avoid precision issues that can cause "shadow acne," where shadow artifacts appear on lit surfaces due to rounding errors.
- **Shadow Intensity**: The shadow's darkness is adjustable via a shadow intensity parameter, allowing control over the visual balance between light and shadow.
- **Smoothness Techniques**: To mitigate aliasing and jagged shadow edges, techniques like percentage-closer filtering (PCF) or screen-space soft shadows can be used. These methods smooth the transition between shadowed and lit areas, resulting in more natural shadows.

Advanced implementations can leverage cascaded shadow maps (CSM) for directional lights, dividing the depth map into sections to maintain high resolution over vast environments. This is particularly useful for scenes with long viewing distances.

Day-Night Cycle (Dusk Mode)

The day-night cycle dynamically alters the scene's lighting, transitioning between bright daylight

and moody nighttime. This is achieved using:

1. **Sunlight Color and Brightness**:
   - The **sunlight intensity** is reduced during the night, creating a soft, ambient glow.
   - Daytime uses a bright white light (1.0, 1.0, 1.0), while nighttime adopts a cooler, dimmer tone (0.2, 0.2, 0.4).
   - A lightBrightness parameter scales the directional light's effect during these transitions.
2. **Skybox Switching**:
   - The skybox changes texture sets based on the time of day, seamlessly blending with the lighting. For example:
     - **Daytime**: A vibrant, sunny sky with clouds
     - **Nighttime**: A darker, lost in the forest mysterious space with muted tones.

Ghost Animation

The ghost entity in the **abandoned amusement park** is designed to move in a fluid and eerie manner. This is implemented using **sinusoidal movement** to create a natural floating effect. The animation consists of three key components:

1. **Vertical Floating:**

The ghost’s **Y-position oscillates** using a sine function:

$$yOffset = ghostHeight \times \sin(ghostSpeed \times ghostTime)$$

This creates an **up-and-down bobbing motion**, making it appear as though the ghost is levitating.

2. **Swaying Rotation:**

A subtle rotation effect is applied to simulate **air resistance or spectral energy**:

$$rotationAngle = ghostRotation \times \cos(ghostSpeed \times ghostTime \times 0.5)$$

The ghost **slightly tilts left and right**, adding to its ethereal presence.

3. **Pulsating Scale Variation:**

To enhance the supernatural appearance, the ghost’s **size fluctuates** dynamically:

$$scaleFactor = 1.0 + 0.1 \times \sin(ghostSpeed \times ghostTime \times 2.0)$$

This effect makes the ghost seem to **expand and contract** like a shifting apparition.

Oasis Effect (Vertex Shader Wind Simulation)

The oasis effect simulates the shimmering heat distortion of a desert environment using vertex displacement. The vertex shader introduces sinusoidal wave motion to displace vertices along their normals:

- **Wave Dynamics**: A sine-cosine wave is applied, modulating based on time and vertex positions (vPosition.x and vPosition.z).
- **Wind Strength**: Controlled by enableWind and influenced by time, allowing dynamic toggling and variation of the distortion.

Rain System

The rain system is technically robust, employing a dedicated **rain shader** for efficiently handling the rendering and coloring of raindrops. This shader ensures that the rain effect remains visually distinct and doesn't interfere with other elements of the scene. By isolating the rain rendering logic, the system maintains flexibility and modularity, allowing adjusting the color or transparency, without affecting the rest of the pipeline.

Spawning Algorithm

The raindrop spawning algorithm ensures a natural and dynamic distribution by randomizing the initial position of each drop. A **randomSpawnAbove()** function is used, which assigns each drop a position vector within a defined 3D bounding box above the scene. The x and z coordinates are randomized across a wide range to create a scattered effect, while the y coordinate places the raindrops consistently at a high altitude.

Movement Logic

Each raindrop is assigned a **velocity vector** that determines its downward speed. This velocity is also randomized within a range, simulating the variability of raindrop sizes and wind effects. The raindrops move each frame by updating their position based on their velocity and the elapsed time (delta time). When a raindrop falls below a predefined lower bound (e.g., the ground level), it is reset to a new position above the scene using the same spawning logic.

GPU Optimization

To optimize performance, the system uses **GPU-based rendering**:

1. The raindrop positions are stored in a GPU buffer and updated dynamically using **glBufferSubData**.
2. Each drop is represented by a **line primitive** consisting of two points: the head and tail, calculated in real time.

3. The vertex buffer holds all the line segments, and the shader renders them efficiently, minimizing CPU-GPU data transfer.

4.2. Graphics Model

The project leverages **OpenGL 4.1 Core** features to achieve high-performance rendering, complemented by **GLFW** for cross-platform window management and real-time user input handling. **GLEW** is utilized for the loading of OpenGL extensions where applicable, and **GLM** serves as the mathematical backbone for vector and matrix transformations.

For content creation, **Blender** was used to design and assemble assets such as the terrain, military vehicles, and environment props. Each asset underwent preprocessing, including applying accurate scaling, orientation adjustments, and UV unwrapping for optimal texture mapping before export. Textures, sourced ones, were applied to the models.

On top of that we can mention:

1. **Shaders**: Custom GLSL shaders are used for advanced rendering techniques, including directional and point lighting, shadow mapping, and fog effects.
2. **Skybox**: A procedural or high-resolution image-based skybox frames the environment, contributing to the sense of depth and atmospheric immersion.
3. **Efficient Asset Integration**: Assets were exported in formats optimized for OpenGL (e.g., OBJ), ensuring compatibility while retaining high polygon fidelity for detailed visuals.

This comprehensive approach blends state-of-the-art tools, custom algorithms, and a robust rendering pipeline to deliver a visually compelling and technically sophisticated graphics model.

4.3. Data Structures

Data structures like camera matrices, light properties, and texture samplers are passed to the shader, influencing how fragments are shaded. For instance:

- The light-space transformation matrix aids shadow mapping by converting world-space coordinates into light-space.
- Arrays for point light positions enable dynamic updates for multiple lights.
- Fog parameters dictate blending in the fragment shader, adjusting based on scene distance.

This structured approach combines lighting, shadowing, and environmental effects, creating a rich and immersive visual experience.

4.4. Class Hierarchy

1. Camera

   ├── sets camera parameters, movement, rotation

2. Model3D

   ├── loads, stores, and renders 3D mesh data

3. Shader

   ├── loads and links vertex and fragment shader programs

4. SkyBox

   ├── manages skybox cube maps and rendering

5. RainSystem

   ├── manages the lifecycle, spawning, and rendering of raindrops

6. (main.cpp)

   ├── orchestrates window creation, event handling, scene update, and rendering

5. Graphical User Interface Presentation & User Manual

- **Keyboard Controls**

  - W / A / S / D: Move the camera forward, left, backward, and right.
  - J / L: Rotate the directional sunlight around the scene's center (left/right).
  - K: Toggle point light sources on or off.
  - 1, 2, 3: Switch between **wireframe**, **points**, and **filled** polygon rendering modes.
  - F: Toggle **fog** on or off.
  - G: Toggle **wind** effect.
  - R: Toggle **rain** system.
  - N: Switch between **day/night** skybox modes.
  - X: It activates the ghost animation.

    - Z: Lets the user inside the ghost and he can move under the ghost.
    - C: Start the **camera path** animation for a cinematic overview.
    - M: Debug mode to show or hide the **depth map** (for shadow debugging).
    - P: Toggle point light sources on or off.betwwn directional lights giving the ground more options for brightness.
    - ESC: Exit the application.

- **Mouse Controls**
    - **Move the mouse** horizontally/vertically to rotate the camera (yaw/pitch).

When the application starts, the user can freely explore the environment. Pressing keys such as R and G will dynamically change the atmosphere (adding rain and wind). The scene's realism is further enhanced at night (key N) when street lamps (point lights) become more obvious and the environment dims.

6. Conclusions and Further Developments

This project illustrates multiple advanced graphical techniques:

- A highly qualitative scene with good textures on the complex objects
- Multi-lighting model (directional + point lights)
- Object animations (Ghost animation)
- Simple vertex shader displacement (wind/heat haze effect)
- Particle system for rain
- Shadow mapping
- Day & Night cycle
- Fog effect

Potential Future Improvements:

This project successfully implements an interactive abandoned amusement park with advanced graphics techniques. Possible future improvements include:

- **AI-driven ghost movement**.
- **More interactive objects**.
- **VR compatibility for immersive experience**.

  - **Complex Weather Transitions**: Gradually move between clear skies and storm conditions (dynamic cloud coverage, heavier rain).
  - **Vehicle Animations**: Add path-following logic for tanks or half-tracks to drive through the village.
- **More Advanced Sound Design**: Additional ambient desert noises (wind gusts)

7. References

1. **OpenGL Programming Guide** – Official documentation for modern OpenGL usage.
2. **GLFW Documentation** – For window and input handling.
3. **GLEW** – The OpenGL Extension Wrangler Library documentation.
4. **glm** – OpenGL Mathematics library for transformation matrices, vectors, and quaternions.
5. **Online Blender Tutorials**