OpenGL Afghan War Scene

Graphic Processing Project

STUDENT: Mihoc Daniel-Alexandru
GROUP: 30435

1. Contents

2. Subject Specification

This documentation describes a 3D computer graphics project depicting a scene inspired by the Afghan War of 1979–1989. The environment illustrates a small desert town near a lake in mountainous terrain. The main features are a variety of military vehicles (half-tracks, tanks, helicopters) on a road that ascends a hill, surrounded by houses, old gunnery emplacement, and desolate cannons. The terrain is complex with various heights and the houses are positioned in various ways like on the side of the hill, next to the lake, and close to the road. The houses are diverse and some were personally crafted by putting common elements together like the entrance or the roof. The scene depicts a town takeover as the armored column marches towards a hill in the distance - with a helicopter in the back of the column ensuring the safety of the convoy.

- Fig 1: Scene overview showcasing most of the village and all the vehicles -

3. Scenario

3.1. Scene and Objects Description

The scene takes place in an arid desert location featuring:

- **A small village** near a lake, set within a **mountainous** environment.
- **Military vehicles** (e.g., half-tracks, tanks) advance on a **cobblestone paved road** that climbs a hill through the village.
- **Houses, gunneries, and cannons**, scattered across the terrain, showing signs of conflict.
- **A helicopter** (animated rotor) patrols overhead, adding dynamism to the scenario.
- **Street lamps** were placed along the main road (4 in total), indicating potential night lighting.

Each object in the scene benefits from high-quality textures and also has shadows from the main sun's directional light. Based on the user preference, the scene can be further enhanced with fog, rain & thunder, or an oasis/wind effect that creates a hallucination effect for the user. The sunlight intensity and sky block are also controlled depending on the user preference, while the helicopter rotor is also animated for flying. The best way to visualize the main points of the scene is to use the camera animation present on demand and if need be for further lighting there are street lamps with point light scattered through the road.

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 lamp posts along the road, create additional lighting effects, they are all triggered at once and can not be triggered individually at the moment.

- Fig 2: Multiple point lights from the lamps on the road -

3. **Shadows**

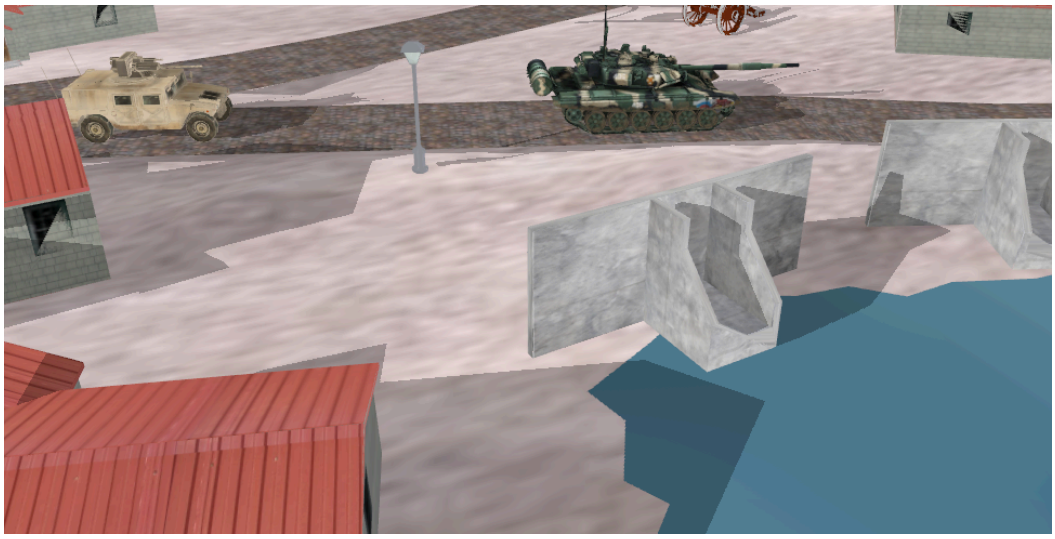
   - **Shadow mapping is used to cast realistic shadows by** directional sunlight.
   - The point lights currently do not cast shadows (can be implemented as an extension).

- Fig 3: Shadow of the objects created by the sunlight -

4. **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 3: Fog as seen from the road next to a point light source -

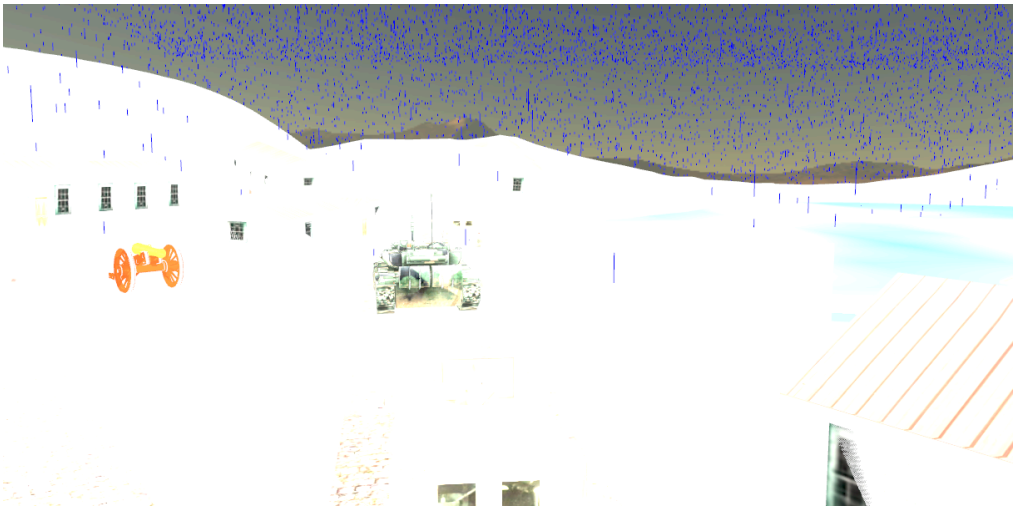
5. **Wind/Oasis Animation**
    - A subtle **wind effect** distorts object vertices using a sinus/cosine function in the vertex shader, simulating heat haze or desert wind.
    - The effect also generates waves in the water which look like it's rising above sea level



- Fig 4: Wind/Oasis effect causing some small flood in the village -

6. **Rain and Thunder**
    - A **rain system** that spawns numerous raindrops falling from the sky, they have varying speeds and different spawn points
    - **Thunder** flashes randomly, brightening the scene momentarily and playing a sound

- Fig 5: Rain effect as seen from the northern hill -



- Fig 6: The lightning effect as seen from the village -

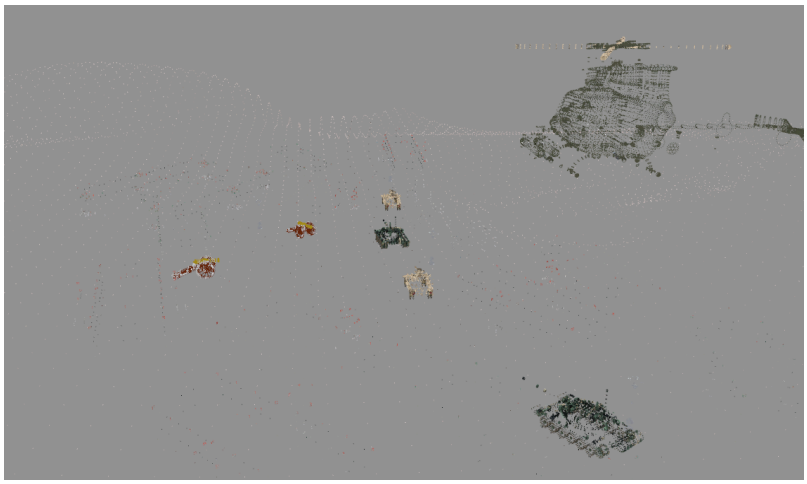
7. **Day/Night Cycle (Skybox Switching)**

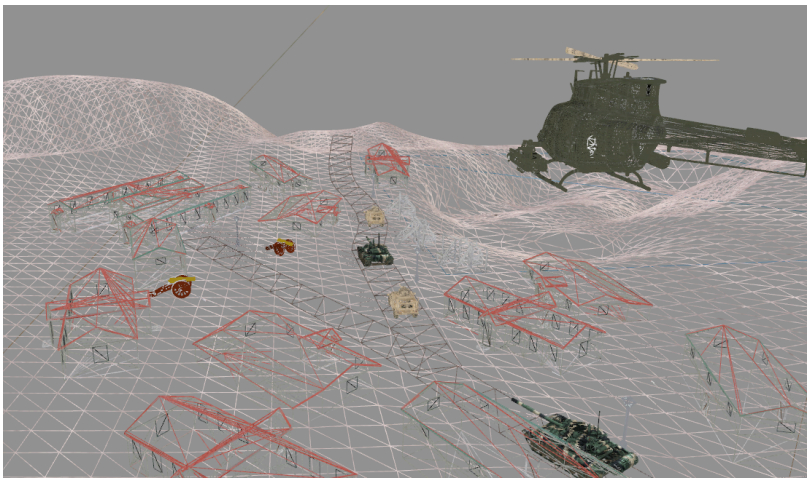
    - The scene allows switching between **day mode** (bright desert sky) and **dusk/night mode** (mars-like or darker sky).
    - The intensity and color of the sunlight are adjusted accordingly.

8. **Animated helicopter rotor**
    - The helicopter has its rotor animated to make it look like it is flying

- Fig 7: Animated helicopter rotor -

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





- Fig 8,9,10: Bird's eye view of the scene 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.
    - Up to 10 point lights can be dynamically added, with their positions and intensities defined in the shader.

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 clear horizons.
        - **Nighttime**: A darker, starry sky with muted tones.

Helicopter Rotor Animation

To implement a rotating helicopter rotor, the rotor blades were modeled separately in Blender. Ensuring precise control over their pivot point, aligning it perfectly with the helicopter's center of rotation. The rotor's coordinates were transposed to OpenGL's coordinate system, rotation is driven by time-dependent updates to the angle, creating a realistic spinning effect.

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.

Thunder Effect

The thunder effect enhances the scene's ambiance during storms by introducing dynamic lighting and sound. A **thunderBrightness** parameter temporarily boosts the scene's brightness to mimic lightning strikes, while asynchronous sound effects simulate thunderclaps. The randomness of lightning intervals adds a natural and unpredictable element, making the environment feel alive and immersive. However, the effect is secondary to the rain feature being active only when rain is toggled by the user.

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).
    - 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.
    - H: Toggle the helicopter rotor's rotation animation.
    - 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.
    - 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 (Helicopter rotor)
- Simple vertex shader displacement (wind/heat haze effect)
- Particle system for rain, plus random thunder events
- Shadow mapping
- Day & Night cycle
- Fog effect

Potential Future Improvements:

1. **Extend Shadow Mapping to Point Lights**: Currently, only the directional light (sun) casts shadows. Point lights with omnidirectional shadow mapping would add realism to nighttime scenes.
2. **Collision Detection**: Interactions between camera and environment objects could prevent clipping through terrain and buildings.
3. **Complex Weather Transitions**: Gradually move between clear skies and storm conditions (dynamic cloud coverage, heavier rain).
4. **Vehicle Animations**: Add path-following logic for tanks or half-tracks to drive through the village.
5. **More Advanced Sound Design**: Additional ambient desert noises (wind gusts, occasional gunfire echoes).

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** https://www.youtube.com/watch?v=P7882NHtv5U&list=PLrgcDEgRZ_kndoWmRkAK4Y7ToJdOf-OSM&index=3https://www.youtube.com/watch?v=P7882NHtv5U&list=PLrgcDEgRZ_kndoWmRkAK4Y7ToJdOf-OSM&index=3