Assignment 1: 2D Truck Scene & Interface

Implementation Tutorial

Course: CSC1336 - Interactive Computer Graphics

0. Project Overview

Goal: Create a simple 2D scene with a truck using modern OpenGL (FreeGLUT).

Key Requirements:

1. **Scene:** A truck on a sloped ground.
2. **Animation:** Truck moves (translation) and wheels rotate.
3. **Interaction:** Ground slope adjustment (-15° to 15°).
4. **Interface:** A hierarchical tree-view UI (Frame Stats, Display, Controls).

1. Development Environment

To build this project, we are using a specific environment setup for Windows.

- **Language:** C++ (Standard C++11 or higher recommended).
- **Compiler: MinGW / GCC** (GNU Compiler Collection for Windows).
- **Graphics Library: FreeGLUT** (Open Source version of the OpenGL Utility Toolkit).
  - *Why FreeGLUT?* It handles window creation, keyboard/mouse input, and the event loop, allowing us to focus on drawing code.
- **IDE:** VS Code (Visual Studio Code) is recommended for editing, using the terminal to compile.

2. Compiling the Project (Script Analysis)

```
g++ -c -o hw1.o hw1.cpp -D FREEGLUT_STATIC -I...
```

1. **`g++`**: The C++ compiler command.
2. **`-c`**: "Compile only." Creates an object file (`.o`) but does not link it into an executable yet.
3. **`-o hw1.o`**: Specifies the **output** filename.
4. **`-D FREEGLUT_STATIC`**: Defines a preprocessor macro. This tells FreeGLUT we are linking to the *static* library (`.lib` or `.a`) rather than a DLL. **Crucial for avoiding missing DLL errors.**
5. **`-I<Path>`**: "Include Path." Tells the compiler where to look for header files (like `GL/freeglut.h`).

2.1 The Linking Stage

After compiling to `.o`, you must **link** the libraries to create the final `.exe`. A complete single-step command often looks like this:

```
g++ hw1.cpp -o hw1.exe -D FREEGLUT_STATIC \
    -I"Path/To/Include" \
    -L"Path/To/Lib" \
    -lfreeglut_static -lopengl32 -lwinmm -lgdi32
```

- **`-L`**: Library Path (Where are the `.a` or `.lib` files?).
- **`-lfreeglut_static`**: Links the specific FreeGLUT library.
- **`-lopengl32`**: Links the standard Windows OpenGL driver.

3. OpenGL Basics: The State Machine

OpenGL acts as a **State Machine**. When you set a property, it remains set until you change it.

Example: Color Persistence

```
glColor3f(1.0f, 0.0f, 0.0f); // Set state to RED
drawRect(...);               // Drawn in RED

drawCircle(...);             // STILL drawn in RED!

glColor3f(0.0f, 1.0f, 0.0f); // Set state to GREEN
drawRect(...);               // Now drawn in GREEN
```

Key Takeaway: Always set the color/style immediately before drawing an object to ensure it looks correct, regardless of what drew previously.

4. OpenGL Basics: Coordinate Systems

We use two matrix modes to handle coordinates:

1. **Projection Matrix (`GL_PROJECTION`):**

- Defines the "Lens" of the camera.
- We used `glOrtho(-1, 1, -1, 1, ...)` to set up a 2D coordinate system where the left edge is -1 and the right edge is +1.

2. **ModelView Matrix (`GL_MODELVIEW`):**

- Defines the position of objects *relative* to the camera.
- `glTranslatef`, `glRotatef` modify this matrix.
- This is where we move the truck and ground.

5. Matrix Stack (`glPushMatrix` / `glPopMatrix`)

This is essential for hierarchical modeling (The Truck).

- `glPushMatrix()`: "Save the current coordinate system."
- *Imagine creating a bookmark.*
- **Transformations**: Apply move/rotate.
- *Write on the page.*
- `glPopMatrix()`: "Restore the saved system."
- *Go back to the bookmark.*

In our code:

We `Push` (save ground location) -> `Translate` (move to wheel location) -> `Draw Wheel` -> `Pop` (return to ground location). This ensures the second wheel doesn't get

6. OpenGL Basics: Primitives

All shapes in our assignment are built from simple primitives inside `glBegin()` and `glEnd()`.

1. **`GL_QUADS`**: Takes 4 vertices. Used for the ground and truck body rectangles.

```
glBegin(GL_QUADS);
glVertex2f(x, y); ...
glEnd();
```

2. **`GL_TRIANGLE_FAN`**: Takes a center point and a list of rim points. Used to approximate **Circles** (Wheels and slider handles).
3. **`GL_LINES`**: Takes pairs of vertices. Used for the UI checkmarks and the tree-view dotted lines.

2. Scene Hierarchy (The "Scene Graph")

The most critical part of this assignment is handling the relationship between the ground and the truck.

The Logic:

- The **Ground** is the parent object.
- The **Truck** is a child of the Ground.
- The **Wheels** are children of the Truck.

Why?
If we parent the truck to the ground, we don't need complex trigonometry (`tan()`) to calculate the truck's height. We just rotate the coordinate system, and the truck "sticks" to the slope automatically.

3. Implementing the Hierarchy

```
// 1. GLOBAL TRANSFORMATION (The Pivot)
// Move origin to pivot point (Top-Right of ground), Rotate, Move back.
glTranslatef(1.0f, -0.5f, 0.0f);
glRotatef(groundAngle, 0, 0, 1);
glTranslatef(-1.0f, 0.5f, 0.0f);

    // 2. DRAW GROUND (Parent)
    drawRect(-1.0f, -0.8f, 2.0f, 0.3f);

    // 3. DRAW TRUCK (Child)
    glPushMatrix();
        // Move along the slope (X axis)
        glTranslatef(truckX, -0.5f + wheelRadius, 0.0f);

        drawTruckBody();
        drawWheels(); // Rotated locally
    glPopMatrix();
```

4. UI Implementation: Immediate Mode

Instead of creating complex objects for buttons, we used an **Immediate Mode** approach. The UI is redrawn from scratch every frame based on the current state.

Structure:

- **Tree View:** A list of rows (`drawTreeRow`).
- **Logic:** Every frame, we call functions like:

```
drawTreeRow(row++, 0, "- Controls", ...);
drawTreeRow(row++, 1, "Ground Slope", ...);
```

This makes the UI very easy to rearrange and debug.

5. UI Interaction Logic

How do we know what the user clicked?

Row-Based Detection:

Since every UI element is a fixed height line, we can calculate the "Row Index" from the mouse Y position.

```
// Formula: (Top_Y - Mouse_Y) / Row_Height
float relY = UI_Y_START - glY;
int row = (int)((relY - 0.02f) / UI_LINE_HEIGHT);

if (row == 4) toggleWireframe();
if (row == 11) startDraggingSlider(SlopeID);
```

Note: We added a `-0.02f` offset to align the visual text position with the mathematical click area.

6. Critical Fixes & Challenges

A. The "Floating Truck"

- **Problem:** Calculating `y = x * tan(theta)` caused the truck to sink into the ground when the pivot wasn't at (0,0).
- **Solution:** Removed the `tan()` math. Used `glPushMatrix()` to rotate the entire coordinate system. The truck now draws at a fixed Y relative to the rotated ground.

B. Invisible Objects (Depth Testing)

- **Problem:** The UI or scene elements disappeared randomly.
- **Solution:** Since this is a 2D assignment using the "Painter's Algorithm" (drawing order), we explicitly disabled depth testing.

```
glDisable(GL_DEPTH_TEST); // For UI and 2D overlays
```

7. Meeting Requirements

Requirement	Implementation Status
800x800 Window	`const int WIDTH = 800;`
Wireframe Toggle	Implemented in UI Row 4.
Pivot Point	Corrected to Top-Right `(1.0, -0.5)`.
Hierarchy UI	Replaced widgets with Tree View (Fig 1 style).
Slope Limits	Clamped between -15.0 and 15.0.

8. Summary & Usage Controls

Keyboard:

- **Left/Right:** Move Truck
- **Up/Down:** Adjust Slope
- **W:** Toggle Wireframe
- **ESC:** Quit

Mouse:

- **Click & Drag:** Adjust sliders (Red, Green, Blue, Slope) or toggle checkboxes in the UI panel.