AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Faculty of Science and Technology

Assignment Title:	Project Report		
Project Name:	**View of Dhaka City**	Date of Submission:	12 Oct, 2025
Course Title:	COMPUTER GRAPHICS	Group	06
		Section:	F
Semester:	SUMMER 2024-25	Course Teacher:	Dipta Justin Gomes

Name	ID	Contributions
Md. Mahabub Al Hashan Sourov	23-50693-1	27%
Fahad Ibne Firoz	23-52505-2	28%
Md Alif Ziad Sarkar	23-50497-1	25%
Md Tauhidul Islam	22-49388-3	20%

Introduction

This combined report analyzes three independent but related OpenGL/GLUT-based visual simulation projects developed as academic assignments :

1. Uttara (Residential Area)
2. Airport Area
3. 300ft View
4. Merged (Combined City Environment)

Each project implements a small 2D animated urban scene that demonstrates standard computer-graphics concepts: geometric modeling of environment elements (roads, buildings, vehicles, water bodies), basic lighting and shading, continuous animation via timers/idle callbacks, input-driven camera/scene control, and simple scene management. The three projects were created using the classical immediate-mode OpenGL / GLUT approach (fixed-function pipeline), which is well-suited for teaching fundamental transformations, view setup and simple lighting while keeping implementation accessible.

The motivation behind these projects is pedagogical: to synthesize graphics fundamentals into compact interactive scenes that visually demonstrate motion, parallax, layering, and environment dynamics. Each scene places a viewer in an urban context (residential district, airport complex, or a long boulevard like the 300ft view) and composes the world from reusable drawing functions (buildings, vehicles, sky, water). Animations are implemented by updating object state variables (positions, angles) inside an idle/timer function and redrawing the scene, demonstrating real-time rendering basics. User controls (keyboard and mouse) are added to manipulate the camera, toggle animations, switch day/night modes, or change weather, allowing direct interaction and experimentation.

Despite differences in scale and implementation detail, the projects share a common architecture and educational goals:

- Modular draw functions (each object or object family is encapsulated in its function).
- Global or scene-scoped state variables govern animation and time of day.
- A central display function composes objects in correct order with transformations to achieve perspective or layering.
- Event handlers (keyboard/mouse) enable simple interactivity and experimentation with animation parameters.

This combined report documents the conceptual design and implementation choices across the three projects, compares and contrasts their approaches, and synthesizes a single feature set and roadmap for future improvements. It is intended to be a single, copy-paste-ready academic-style document suitable for inclusion in a course submission, with object IDs and function mappings preserved so the grader can cross-check functions in the implementation files. The report avoids reproducing full source listings inline (those are placed in the implemented repository) while giving a clear map of what each file contains and how the system is organized.

By analyzing three projects together we can more easily identify reusable patterns (e.g., building blocks for urban scenes), evaluate trade-offs (2D orthographic vs 3D perspective), and recommend incremental improvements (texturing, improved lighting, physics-based interactions). The combined view also showcases how the same core ideas scale from a more static 2D-style residential scene (Uttara) to a complex airport scene (multiple animated planes, rain) to a 3D boulevard with parallax (300ft view)

Background of the problem

Real-time visualization of urban scenes is a broad domain with many applications: simulation, training, planning, entertainment, and education. In a computer graphics course context, compact urban scene projects are excellent vehicles to teach the essentials: coordinate systems, projection, hierarchical modeling, transformations, animation loops, and event-driven interaction.

The core problems these projects aim to address are:

1. **Modeling multiple heterogeneous objects in a single scene.** An urban scene contains many kinds of objects with differing behaviors: static (buildings, ground), periodic (lights, window shimmer), and moving (vehicles, planes, clouds). A key challenge is to structure the code so each object is modular and composable while keeping the global update and rendering loop simple.
2. **Animating objects smoothly in real time.** Smooth motion requires consistent state updates tied to a timer or idle callback, minimal per-frame CPU overhead, and careful state resetting (wrap-around positions) to maintain continuous motion. Students must decide whether to animate the world relative to a static camera (moving objects) or move the camera through a static world; both approaches have trade-offs for complexity and perceived parallax.

3. **Representing depth and perspective.** The projects include both 2D orthographic scenes (easier to implement, like Uttara/Marged variants) and 3D scenes with perspective projection (300ft). Presenting depth convincingly without advanced shading or textures requires layering, scale changes (objects farther away are smaller), shading/light and motion parallax to cue depth.
4. **Day-night/weather transitions and lighting.** Simulating time-of-day or weather demonstrates lighting and color manipulation. The Airport project includes weather states (SUNNY/OVERCAST/RAINY) and raindrop simulations, exposing students to conditional rendering, particle-like systems, and integrating animation with input controls.
5. **Interactivity & user control.** Allowing users to change camera angles, toggle animations, and adjust speeds provides a sandbox to observe consequences of parameter changes. Proper event handling and gracefully handling state changes are important.
6. **Balancing complexity and pedagogy.** These projects must be implementable within a single semester and with immediate-mode OpenGL; thus emphasis is on clarity over advanced features. That choice imposes constraints (no shaders, limited performance benefits), but offers clarity in teaching transformation composition and the rendering pipeline.

Each project addresses these problems differently: Uttara uses a rich set of 2D components (roads, buses, trains, lakes) with explicit IDs and animation functions that represent common urban objects; Airport expands the object set to include many airport-specific elements and a weather/raindrop subsystem; 300ft implements a 3D view with perspective, lighting and parallax (moving buildings and camera controls). Comparing them highlights how the same educational goals are realized under different constraints (2D vs 3D, particle-like rain vs simple moving clouds, orthographic vs perspective projections).

The background for selecting these scenes is practical and mnemonic: students can more quickly form mental models of roads, runways, buildings and vehicles — objects that are intuitively understood — allowing them to focus on graphics techniques rather than domain modeling. These projects are therefore well-suited for teaching, assessment, and incremental extension (textures, models, path-finding traffic simulation) in later coursework or capstone projects.

Methods used

All three projects were developed using the OpenGL fixed-function pipeline with GLUT for windowing, input, and the main loop. This is a lesson-focused stack that emphasizes geometry, transformations, and simple lighting, without the additional complexity of modern programmable pipelines.

Common architectural patterns

1. **Modular draw functions** – Each conceptual object has a dedicated function (e.g., drawBuilding(), Plane_2(), drawCloud()); functions encapsulate geometry and local transformations. This modular design helps reuse code, enables multiple instances (draw many buildings with different transforms), and maps directly to IDs/object lists.
2. **Global scene state** – Animation and time-of-day states are managed via global variables (positions, speeds, current_time enum, sunAngle, etc.). These are updated by timer or idle callbacks.
3. **Animation loop** – GLUT timers (glutTimerFunc) or glutIdleFunc are used to increment object positions and angles, then trigger redisplay. Example patterns:
    - carZ += speed; if (carZ > limit) carZ = reset;
    - sunAngle += delta; if (sunAngle > 360) sunAngle = 0;
4. **Event-driven interactivity** – glutKeyboardFunc and glutMouseFunc handle toggles (day/night, animation on/off), speed changes, and camera adjustments. This allows run-time experimentation without recompilation.
5. **Scene composition** – The display() function sets the view (either gluOrtho2D for 2D scenes or gluPerspective + gluLookAt for 3D), calls object drawing functions in an order that respects depth (background sky, ground, static geometry, then moving objects), and swaps buffers.
6. **Simple lighting & shading** – Where present (300ft), glEnable(GL_LIGHTING) and a directional/positional light (GL_LIGHT0) provide basic diffuse/ambient/specular components. Color material is enabled for simpler shading control.

Project-specific methods

- **Uttara (2D/orthographic)** uses layered polygons for roads, lakes, bridges and multiple vehicles. Window-rendered text functions are used for labels. Animation is driven by timers that update vehicle positions and use modular drawCar, drawBus,

drawTrain functions. Day/night toggle changes sky color and lamp rendering. The project also maps many object IDs to functions which aids grading and modular testing.

- **Airport (2D with many objects + weather)** uses an enumerated WeatherState to switch between sunny/overcast/rainy states. A simple raindrop particle system is implemented using a std::vector<Raindrop> for many small line segments that fall and reset, demonstrating a simple particle approach and reuse of arrays with randomized starting positions and speeds. Planes, taxiways and terminals are implemented as polygonal compositions; wheels and other rounded elements are approximated by many-sided polygons. Event handlers allow changing weather states and adjusting raindrop speeds. Object ID lists and functions are provided extensively in the Airport file.
- **300ft (2D with Day/night cycle)** uses 2D OpenGL cityscape simulation with a dynamic day/night cycle, using gluPerspective and gluLookAt to render a vibrant urban scene featuring labeled buildings (AUSIS, ICCB, Eye H, BASUNDH, RUPAY), animated vehicles, and parallax scrolling; it incorporates lighting for depth, camera controls (a/d/w/s/z/x), transformation stacks (glPushMatrix/glPopMatrix), and modular functions like setupLights(), drawSky(), drawGround(), drawCity(), drawCar(), drawSun(), and drawCloud() to build a richly layered environment with both lit geometry and unlit sky gradients.
-

Feature Set

All three projects — *Uttara (Residential Area)*, *Airport Area*, and *Beautiful Dhaka 300ft View* — demonstrate the practical use of OpenGL and GLUT to create interactive animated environments.
Each scene showcases modular object rendering, animation through timers, and user-driven control via keyboard and mouse inputs.
The projects progressively advance in complexity: Uttara focuses on layered 2D animation, Airport integrates weather and aircraft simulation, and 300ft introduces 3D perspective and lighting.

A. Uttara (Residential Area Scene)

Description:

The Uttara project presents a peaceful residential environment with roads, bridges, lakes, vehicles, and buildings. It features both day and night modes, animated transport systems (buses, cars, trains, boats), and simple text rendering for labels like "Train Station."

Main Functionalities:

- Day/Night toggle using color transition and lamps.
- Continuous animation of cars, trains, and boats.
- Multiple layered roads and railway systems.
- Procedural buildings and environment structures.

Key	Function
D / N	Toggle Day or Night mode
S	Start/Stop global animation
↑ / ↓	Increase or decrease vehicle speed
T	Start/Stop train animation
B	Start/Stop bus animation
P	Pause all vehicles
R	Reset animation to default
ESC	Exit program

Mouse Action	Function
Left Click	Pause or resume boat animation
Right Click	Toggle car animation on/off

B. Airport Area Scene

Description:
The Airport project simulates a busy airfield environment including multiple runways, terminals, parked aircraft, airport buses, and dynamic weather effects. It introduces a raindrop particle system, plane takeoff and landing animations, and environmental changes based on user-selected weather conditions.

Main Functionalities:

- Weather control system (Sunny, Overcast, Rainy).
- Plane and ground vehicle animation with realistic motion.

- Dynamic sky color change and lighting based on weather.
- Raindrop simulation using random particle movement.

Key	Function
1	Set weather to Sunny
2	Set weather to Overcast
3	Set weather to Rainy (start rain animation)
P	Toggle plane takeoff and landing
C	Toggle airport bus movement
↑ / ↓	Adjust raindrop fall speed
← / →	Increase or decrease raindrop count
R	Reset to default weather (Sunny)
ESC	Exit program

Mouse Action	Function
Left Click	Start/Stop plane movement
Right Click	Change weather mode via quick context menu

C. Beautiful Dhaka — 300ft View

Description:

The 300ft project offers a view of 300feet area highway. The scene includes moving cars, rows of tall buildings, clouds, and sun/moon that changes dynamically. It depicts how a trip along this area would look like.

Main Functionalities:

- Dynamic sun and cloud movement.
- Car animation through the 300feet area.

  Dynamic Day and Night Cycle

Key	Function
D / N	Toggle Day or Night mode
S	Start/Stop global animation
↑ / ↓	Increase or decrease vehicle speed
P	Pause all vehicles
R	Reset animation to default
ESC	Exit program

Mouse Action	Function
Left Click	Pause or resume boat animation
Right Click	Toggle car animation on/off

D. Merged Project (Combined City Environment)

Short Description:
The merged project integrates selected elements from all three scenes into a single interactive environment — combining the **urban depth of 300ft**, **transport systems from Uttara**, and **weather and animation from Airport**.
It demonstrates modular scene management and the ability to switch or merge scenarios dynamically through keyboard input.

Merged Functionalities:

- Combined rendering of roads, airports, and city buildings.
- Shared animation manager handling cars, planes, and trains simultaneously.
- Unified day/night and weather system that affects all objects.
- Common input controls for toggling between "City," "Airport," and "Highway" scenes

Key	Function
1	Load Uttara scene
2	Load Airport scene
3	Load 300ft city view
W / S / A / D	Move or rotate camera (active in 3D mode)
L	Toggle lighting or lamps
R	Reset all scenes
ESC	Exit application

Object lists (IDs & mapping)

Uttara

1. ID201 — trainArea — `trainArea_201()`
2. ID202 — Lake — `Lake_202()`
3. ID203 — Sky — `Sky_203()`
4. ID204 — FullRoad — `FullRoad_204()`
5. ID205 — FullSubRoad — `FullSubRoad_205()`
6. ID206 — Bus1 — `Bus1_206()`
7. ID207 — Bus2 — `Bus2_207()`
8. ID208 — Car1 — `Car1_208()`
9. ID209 — Car2 — `Car2_209()`
10. ID210 — Car3 — `Car3_210()`

11. ID211 — Truck1 — `Truck1_211()`
12. ID212 — Train — `Train_212()`
13. ID213 — Plane — `Plane_213()`
14. ID215 — TrainLine — `TrainLine_215()`
15. ID216 — TrainStation — `TrainStation_216()`
16. ID217 — Lamp — `lamp_217()`
17. ID218 — Bridge over lake — `bridge_218()`
18. ID219 — Boat1 — `boat1_219()`
19. ID220 — Boat2 — `boat2_220()`
20. ID221 — Signboard — `signboard_221()`
21. ID222 — Building1 — `building1_222()`
22. ID223 — Building2 — `building2_223()`
23. ID224 — Building3 — `building3_224()`
24. ID225 — Building4 — `building4_225()`
25. ID226 — Building5 — `building5_226()`
26. ID227 — Building6 — `building6_227()`
27. ID232 — Sun & Moon — `sunMoon_232()`

Airport

1. ID101 — Grass — `Grass()`
2. ID102 — Catkin — `Catkin()`
3. ID103 — Lake — `Lake()`
4. ID104 — Runways1 — `Runways1()`
5. ID105 — Runways2 — `Runways2()`
6. ID106 — Taxiways1 — `Taxiways1()`
7. ID107 — Taxiways2 — `Taxiways2()`
8. ID108 — Taxiways3 — `Taxiways3()`
9. ID109 — Aviation1 — `Aviation1()`
10. ID110 — Aviation2 — `Aviation2()`
11. ID111 — Parking — `Parking()`
12. ID112 — Terminal1 — `Terminal1()`
13. ID113 — Terminal2 — `Terminal2()`
14. ID114 — TerminalLamp1 — `Terminal_Lamp1()`
15. ID115 — TerminalLamp2 — `Terminal_Lamp2()`
16. ID116 — Tower — `Tower()`
17. ID117 — Plane in the runway 1 — `Plane_1()`
18. ID118 — Plane 1 in the Parking — `Plane_2()`
19. ID119 — Plane 2 in the Parking — `Plane_3()`
20. ID120 — Plane 1 in the Runway 2 — `Plane_4()`
21. ID121 — Plane2 in the Runway 2 — `Plane_5()`
22. ID122 — PrivateJet1 — `Private_Jet1()`
23. ID123 — PrivateJet2 — `Private_Jet2()`
24. ID124 — PrivateJet3 — `Private_Jet3()`
25. ID125 - ID138 — RawFighterPlane — . . .

300ft

- ID401 — Road 1 — Road1()
- ID402 — Road 2 — Road2()
- ID403 — Footpath — Footpath()
- ID404 — Under Lake — underlake()
- ID405 — Road 3 — Road3()
- ID406 — Tree 1 — Tree1()
- ID407 — Tree 2 — Tree2()
- ID408 — Tree 4 — Tree4()
- ID409 — Tree 3 — Tree3()
- ID410 — Car (Main Vehicle) — CAR_FULL()
- ID411 — Car 2 — Car_2()
- ID412 — Lamp Post 1 — Lamp_post_1()
- ID413 — Lamp Post 2 — Lamp_post_2()
- ID414 — Lamp Post 3 — Lamp_post_1() (called within Lamp_post_2)
- ID415 — Lamp Post 4 — Lamp_post_1() (called within Lamp_post_2)
- ID416 — Lamp Post 5 — Lamp_post_1() (called within Lamp_post_2)
- ID417 — Traffic Light — Traffic_light()
- ID418 — Sky — sky()
- ID419 — Building (AUSIS) — building_ausis()
- ID420 — Building (ICCB) — building_iccb()
- ID421 — Eye Hospital — building_Eye_hospital()
- ID422 — Bank — bank()
- ID423 — Commercial Building — commercial_building()
- ID424 — Office Building — office()
- ID425 — Sun — Sun()
- ID426 — Cloud — cloud()
- ID427 — Plane — Plane()
- ID428 — Bus — Bus()
- ID429 — Car 3 — car_3()
- ID430 — Truck — Truck()
- ID431 — Garden / Grass Area — Garden()
- ID432 — Tree Line — Tree()
- ID433 — Wheel (Car Wheel) — circle1()
- ID434 — Lamp Light — circle2()
- ID435 — Truck Wheel — circle7()
- ID436 — Traffic Light (Red) — circle9()
- ID437 — Traffic Light (Yellow) — circle10()
- ID438 — Traffic Light (Green) — circle11()
- ID439 — Lamp Glow (Night Light Effect) — circle2() (used within Lamp_post_1)

Future Directions

The three projects form an excellent foundation for progressive improvements. Below are prioritized recommendations (technical and pedagogical):

1. **Move to shader-based pipeline** — Replace the fixed-function pipeline with OpenGL core profile and GLSL shaders. This allows per-pixel lighting, normal mapping for richer building details, and more efficient batching. (Pedagogical benefit: introduces modern GPU programming.)
2. **Textures and materials** — Add texture mapping for roads (asphalt), building facades, and terminal surfaces. Use simple texture atlases to keep performance good and demonstrate UV mapping.
3. **Level-of-Detail (LOD) and instancing** — For large city scenes, implement LOD or instance rendering for repeated geometry (buildings, lamp-posts). This reduces CPU/GPU overhead and scales better.
4. **Improved particle system** — Replace current raindrops with a more advanced particle system (GPU-based if possible) supporting splashes and wind, and properly frame-rate independent integration.
5. **Physics and collisions** — Add simple kinematic motion and collision checks if vehicle control is introduced (e.g., a driven car). For more realistic simulation, integrate a light-weight physics engine or implement simple bounding-box checks.
6. **AI/traffic simulation** — Add multiple vehicles with path-following, traffic lights logic and simple intersection behaviors. This demonstrates state machines and emergent behaviors.
7. **Audio & UI** — Add ambient city sounds, engine noise, and UI overlays (FPS, toggles). Audio increases immersion and demonstrates integrating multimedia.
8. **Export & data-driven scene** — Move object definitions to external data files (JSON or XML) so the scene can be altered without recompiling (e.g., add buildings dynamically).
9. **Mobile/web porting** — Consider porting to WebGL (via Emscripten) for easy sharing, or to a cross-platform engine to run on mobile devices.
10. **Accessibility & parameterization** — Make camera, animation speed and weather parameters adjustable via an in-scene GUI (e.g., ImGui) for quick demos and grading.

Each of these directions provides a clear next step in improving realism, interactivity, and performance and is aligned with learning goals for intermediate/advanced graphics coursework.

Conclusion

The three projects (Uttara, Airport, and Beautiful Dhaka 300ft View) successfully demonstrate foundational computer graphics concepts in distinct but complementary scenarios. Uttara focuses on a richly-featured 2D residential environment with multiple transport modes and object IDs for systematic evaluation. Airport expands into weather-driven animation and a large set of airport-specific assets and functions (including a basic particle approach for rain). The 300ft scene demonstrates how 3D perspective, lighting and parallax can enhance perception of depth for an urban boulevard.

Taken together, they form a coherent teaching package: modular geometry and function mapping, real-time animation, basic lighting, and interactive controls. The consistent use of function-per-object and ID mapping aids grading and reusability. The projects are deliberately implemented using immediate-mode OpenGL for clarity; their code organization makes them suitable baselines for iterative enhancement — adding textures, shaders, physics, and improved particle systems.

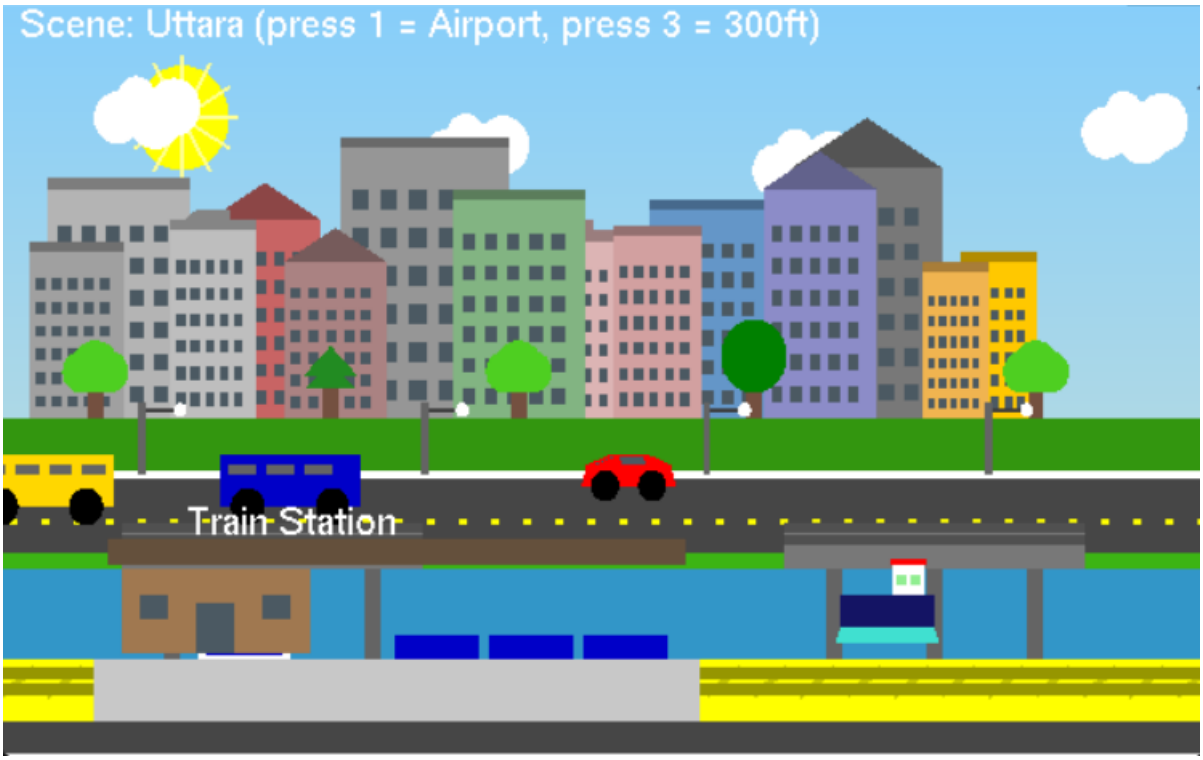
The next steps are straightforward: migrate to modern OpenGL, add richer materials, improve particle and traffic behaviors, and consolidate assets into a data-driven scene description. These enhancements will extend the projects' educational value and allow them to scale to more realistic simulations while preserving the clear learning objectives that made the original implementations effective.

Link of the implementation in Github :

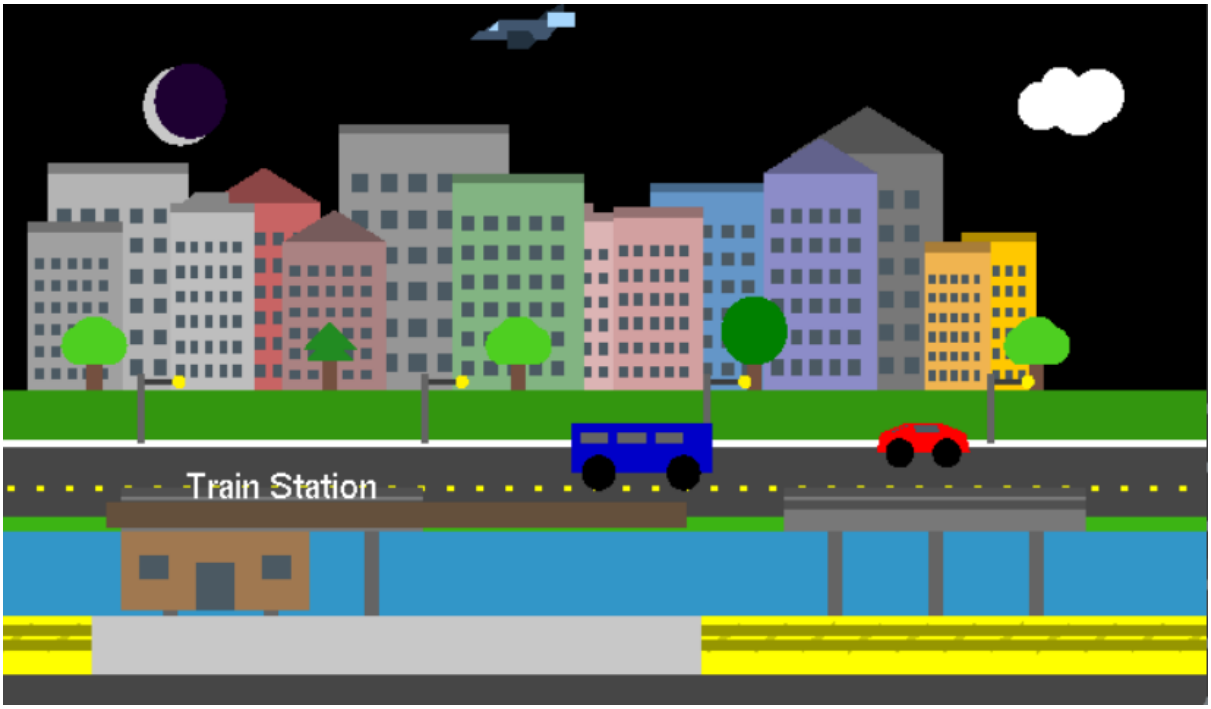
Github Link- https://github.com/Fahad-Firoz/ComputerGraphics_OpenGL_Projects

Screenshots Of The Project

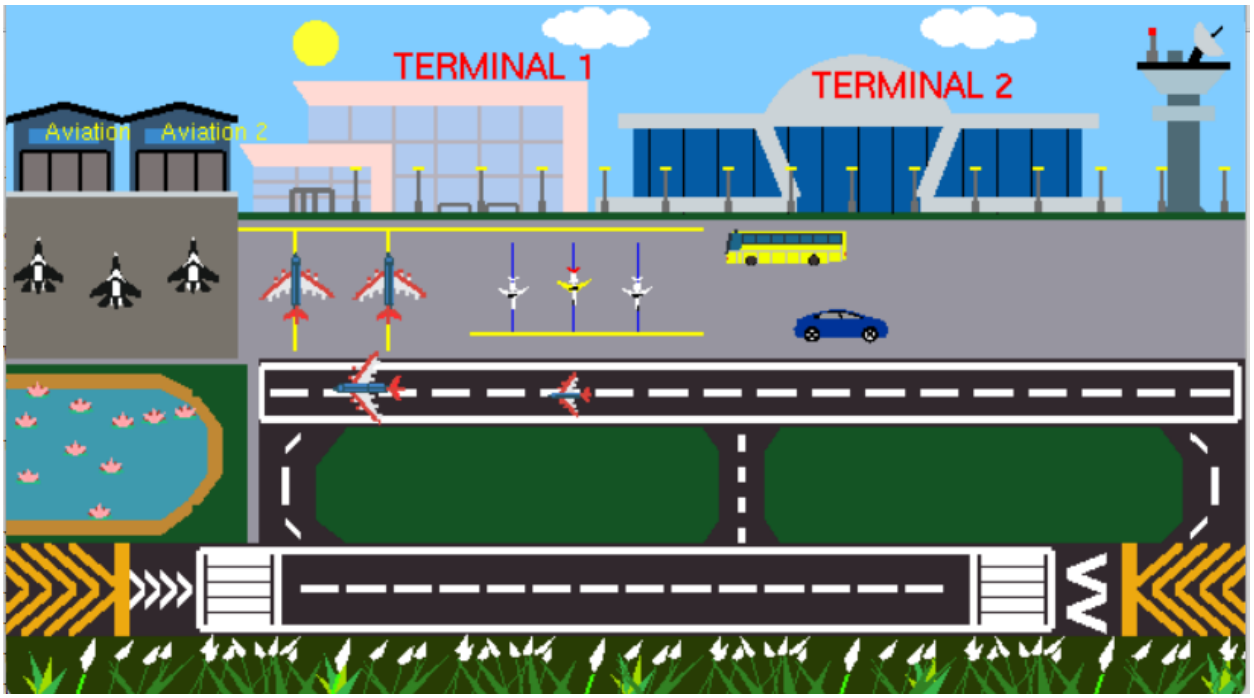
A.Uttara Daytime(Also the introductory scene)



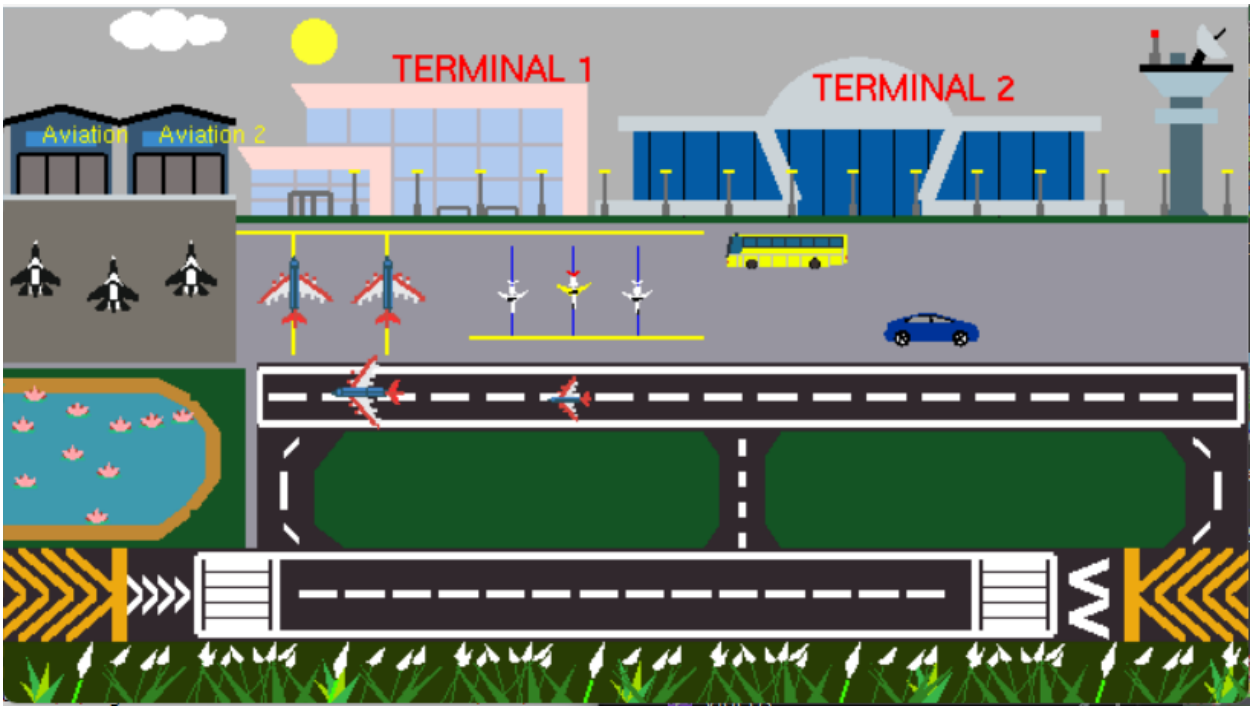
Uttara Nighttime



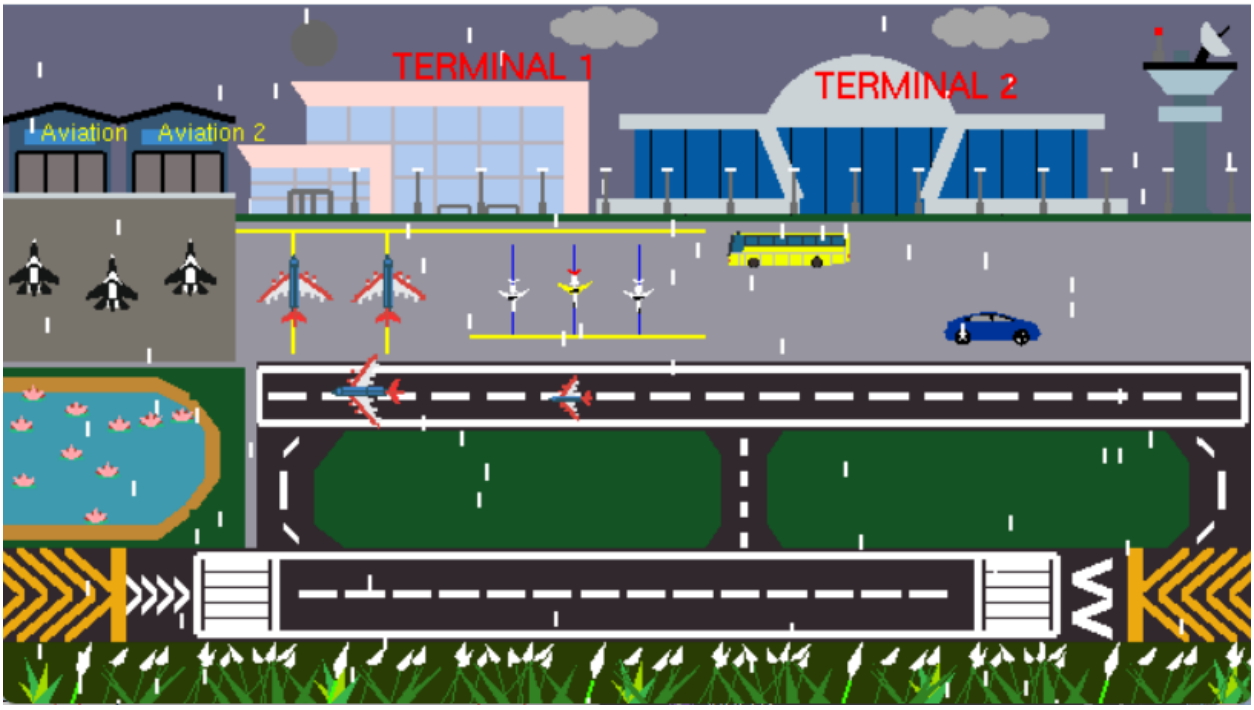
B.Airport(Daytime)



Airport(Nighttime)



Airport(Rainy Weather)



C.300FT(Daytime)

300FT(Nighttime)

List of References

- **OpenGL Programming Guide (The Red Book) – Dave Shreiner, Mason Woo, Jackie Neider, and Tom Davis, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V*, 9th Edition, Addison-Wesley, 2016.**
- **OpenGL Utility Toolkit (GLUT) Documentation – Mark J. Kilgard, *OpenGL Utility Toolkit (GLUT) API Documentation*, Silicon Graphics Inc., 1996.**
- **OpenGL Reference Manual (The Blue Book) – Dave Shreiner, Graham Sellers, John Kessenich, *OpenGL Reference Manual*, 4th Edition, Addison-Wesley, 2013.**
- **AIUB Computer Graphics Course Materials, *Department of Computer Science, Faculty of Science & Technology, American International University-Bangladesh (AIUB)*, 2024.**