

Air Race - Documentation

Anastasios Karampekios

01229412

1 Brief Description of Implementation

Air Race is a project that implements a 3D air race game using C++ and the Vulkan graphics API.

1.1 Core Architecture

The game is built on top of the GCG Lab Framework that uses VulkanLaunchpad and was extended to cover the needs of the game:

- **Physics System:** Bullet Physics library integration for realistic flight dynamics.
- **Rendering Pipeline:** Modern Vulkan-based graphics engine with compute shader support.
- **Resource Management:** Efficient asset loading and memory management.
- **Input System:** Real-time flight controls with keyboard input processing.

1.2 Key Implementation Features

- Real-time Vulkan rendering with multiple pass pipeline.
- GPU-accelerated particle systems using compute shaders.
- Physics-based flight simulation with 6 degrees of freedom.
- Dynamic environmental mapping with cubemap reflections.
- Procedural water animation through vertex shader mathematics.
- Gate collision detection and race progression system.

2 Game Features

2.1 Core Gameplay

- **Objective:** Navigate through sequential gates in a time-limited air race.
- **Flight Mechanics:** Aircraft control with pitch, roll, yaw, and throttle.
- **Race System:** 9 gates positioned over an ocean environment with progress tracking.
- **Timer Challenge:** 60-second countdown creates time pressure.
- **Win/Lose Conditions:** Success by completing all gates; failure by collision, water crash, or timeout.

2.2 Visual Features

- **Aircraft Model:** Loaded 3D airplane as .obj with texturing and materials.
- **Ocean Environment:** Water surface as texture with dynamic wave animation.
- **Skybox Environment:** 360-degree environmental cubemap for immersive atmosphere.
- **Particle Effects:** Real-time exhaust smoke simulation from aircraft engines.
- **Environmental Mapping:** Realistic surface reflections on both aircraft and water.
- **Gate Visualization:** Clearly marked racing gates with collision detection.

2.3 User Interface

- **HUD System:** Real-time display of aircraft speed, timer, current gate, and total gates.
- **Debug Modes:** Wireframe rendering, culling mode cycling, shader hot-reload.
- **Game States:** Clear success/failure feedback with restart functionality.

2.4 Asset Sources

- **Aircraft Model:** Custom .obj model with material definitions.
- **Skybox Textures:** Cubemap environment for atmospheric rendering.
- **Water Textures:** DDS format textures for realistic ocean surface materials.

3 Implemented Effects

3.1 GPU Particle System using Compute Shader

Technical Description:

- **Compute Shader Approach:** Utilizes Vulkan compute shaders for massively parallel particle simulation.
- **Particle Count:** Up to 8000+ particles simulated simultaneously.
- **Workgroup Optimization:** 64 particles per compute workgroup for optimal GPU utilization.
- **Memory Architecture:** GPU-only storage buffers eliminate CPU-GPU transfer overhead.

Physics Simulation:

- **Gravitational Forces:** Realistic downward acceleration affecting particle trajectories.
- **Wind Effects:** Configurable wind forces create natural atmospheric interaction.
- **Turbulence:** Randomized velocity perturbations simulate atmospheric turbulence.
- **Lifecycle Management:** Dynamic particle spawning, aging, and destruction.

Visual Features:

- **Billboard Rendering:** Particles automatically orient toward camera for optimal visibility.
- **Alpha Blending:** Smooth transparency effects based on particle lifetime.
- **Size Animation:** Particles expand over time simulating smoke dispersion.

3.2 Vertex Shader Animation

Technical Implementation:

- **Vertex Displacement:** Real-time height calculation using trigonometric functions.
- **Normal Calculation:** Mathematical derivatives compute surface normals for lighting.
- **Time-based Animation:** Continuous wave progression creates dynamic ocean surface.
- **Performance Optimization:** All calculations performed in vertex shader for efficiency.

Physical Accuracy:

- **Sinusoidal Components:** Both sine and cosine functions create varied wave patterns.
- **Frequency Variation:** Different wave frequencies simulate various conditions.
- **Directional Diversity:** Multi-directional waves prevent repetitive patterns.

3.3 Environmental Mapping System

Cubemap Reflection:

- **360-Degree Environment:** Six-sided skybox cubemap captures complete surrounding environment.
- **Real-time Reflection Vectors:** Fragment shader calculates reflection angles using surface normals.
- **Fresnel Effects:** Physically-based fresnel equations determine reflection intensity.
- **View-dependent Reflections:** Reflection appearance changes based on camera perspective.

3.4 Real-time Physics Integration

Implementation: Bullet Physics library integration in C++.

Flight Dynamics:

- **Rigid Body Simulation:** Aircraft modeled as physically accurate rigid body.
- **Angular Dynamics:** Pitch, roll, and yaw responses.
- **Collision Detection:** Precise collision between aircraft and environment objects.

Gate Collision System:

- **Geometric Intersection:** Mathematical calculation of aircraft-gate penetration.
- **Race Progress Tracking:** Sequential gate completion validation.
- **Collision Feedback:** Immediate game state changes upon collision events.

4 Controls and Features

4.1 Button Functions

- **Arrow Keys:** Control aircraft pitch (up/down) and roll (left/right) for precise navigation.
- **A/D Keys:** Adjust yaw to turn the aircraft's nose left or right.
- **W/S Keys:** Increase (W) or decrease (S) aircraft speed.

- **H Key:** Toggles the heads-up display showing speed, timer, and gate progress.
- **R Key:** Restarts the game after completion or failure.
- **F1 Key:** Enables wireframe rendering for debugging.
- **F2 Key:** Cycles through face culling modes for visualization.
- **F5 Key:** Triggers real-time shader recompilation for immediate visual updates.

4.2 Success Criteria

- **Complete Race:** Pass through all gates in correct order.
- **Time Limit:** Finish before 60-second timer expires.
- **Avoid Crashes:** Prevent collision with gates or water surface.
- **Victory Screen:** Green success screen indicates race completion.

5 External Libraries and References

5.1 Core Dependencies

1. **Vulkan SDK 1.3.216.0+** - Modern graphics API
 - URL: <https://vulkan.lunarg.com/>
 - Purpose: Primary rendering backend for high-performance graphics.
2. **Bullet Physics 3.x** - Physics simulation
 - URL: <https://github.com/bulletphysics/bullet3>
 - Purpose: Realistic flight dynamics, collision detection, and rigid body simulation.
3. **GLFW 3.x** - Window management and input
 - URL: <https://www.glfw.org/>
 - Purpose: Cross-platform window creation, input handling, and context management.
4. **Assimp 5.2.5** - Asset importing
 - URL: <https://github.com/assimp/assimp>
 - Purpose: 3D model loading (.obj, .fbx, .gltf formats).
5. **Dear ImGui** - User interface
 - URL: <https://github.com/ocornut/imgui>
 - Purpose: Immediate mode GUI for debug displays and HUD elements.
6. **GLM (OpenGL Mathematics)** - Mathematical operations
 - URL: <https://github.com/g-truc/glm>
 - Purpose: Vector/matrix mathematics for 3D transformations.
7. **VulkanLaunchpad Framework** - Custom rendering foundation
 - URL: <https://github.com/cg-tuwien/VulkanLaunchpad>
 - Purpose: Vulkan wrapper and utility functions for streamlined development.