3D Flight Simulator: Implementation of Computer Graphics Concepts

Computer Graphics Project Team
Waseda University

January 2025

Abstract

This paper provides a detailed technical analysis of a 3D flight simulator implemented using Three.js, focusing on the mathematical foundations and computer graphics concepts utilized. The implementation demonstrates advanced graphics techniques including quaternion-based rotations, physics-based animations, particle systems, procedural terrain generation, real-time rendering optimizations, minimap integration, HUD systems, and custom shaders.

1 Mathematical Foundations

1.1 3D Transformations

The core transformation pipeline uses homogeneous coordinates and matrix operations:

1.1.1 Rotation Matrices

Euler angle rotations are implemented using:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

1.1.2 Quaternion Rotations

Aircraft orientation uses quaternions to avoid gimbal lock:

$$q = \cos\left(\frac{\theta}{2}\right) + (x\mathbf{i} + y\mathbf{j} + z\mathbf{k})\sin\left(\frac{\theta}{2}\right) \tag{2}$$

Quaternion multiplication for rotation composition:

$$q_1 \otimes q_2 = (w_1 w_2 - \vec{v_1} \cdot \vec{v_2}, w_1\vec{v_2} + w_2\vec{v_1} + \vec{v_1} \times \vec{v_2}) \tag{3}$$

2 Flight Physics System

2.1 Aerodynamics Model

The flight model implements the following forces:

2.1.1 Lift Force

$$F_{lift} = \frac{1}{2}\rho v^2 S C_L \sin(\alpha) \tag{4}$$

where:

- ρ = air density (1.225 kg/m^3)
- v = airspeed
- S = wing area
- C_L = lift coefficient
- α = angle of attack

2.1.2 Drag Force

$$F_{drag} = \frac{1}{2}\rho v^2 S C_D \tag{5}$$

where $C_D = C_{D0} + kC_L^2$ (parasitic and induced drag)

3 Particle Systems

Particle motion follows Newtonian physics:

3.1 Position Update

$$\vec{p}(t) = \vec{p}_0 + \vec{v}_0 t + \frac{1}{2}\vec{a}t^2 \tag{6}$$

3.2 Particle Life Cycle

Opacity calculation:

$$opacity = \frac{life_{remaining}}{life_{total}} \tag{7}$$

4 Terrain Generation

Multi-octave noise function:

$$height(x, z) = \sum_{i=1}^{n} A_i \sin(f_i x) \cos(f_i z) \tag{8}$$

where:

- A_i = amplitude for octave i
- f_i = frequency for octave i

5 HUD System

The HUD displays key metrics such as speed, altitude, pitch, roll, G-force, and fuel percentage. These values are computed using:

- Speed (v):
$$v = \sqrt{v_x^2 + v_y^2 + v_z^2} \tag{9}$$
- Altitude (h):
$$h = p_y \tag{10}$$
- Pitch (θ_{pitch}):
$$\theta_{pitch} = \arcsin\left(\frac{v_y}{v}\right) \tag{11}$$
- Roll (θ_{roll}):
$$\theta_{roll} = \arctan 2(v_x, v_z) \tag{12}$$
- G-force (g):
$$g = 1 + \frac{|\vec{a}|}{g_0} \tag{13}$$
where g_0 is the acceleration due to gravity.

6 Minimap System

The minimap provides a top-down view of the airplane's position and heading. The coordinates are scaled as follows:

$$x_{map} = \frac{p_x}{scale} + center_x, \quad z_{map} = \frac{p_z}{scale} + center_z \tag{14}$$

The direction vector is computed using:

$$\vec{d} = R(\vec{d}_{initial}) \tag{15}$$

where R is the rotation matrix based on the airplane's current orientation.

7 Missile System

Missiles follow a linear trajectory with particle-based trail effects. The missile's position updates are computed using:

$$\vec{p}_{missile}(t) = \vec{p}_{launch} + \vec{v}_{missile}t \tag{16}$$

Trail particles decay over time, with their opacity defined as:

$$opacity_{trail} = \frac{life_{remaining}}{life_{total}} \tag{17}$$

8 Shaders and Rendering Techniques

Shaders play a critical role in enhancing the visual quality of the simulator. The following shaders are implemented:

8.1 Sky Gradient Shader

A custom shader is used to create a gradient sky. The vertex and fragment shaders are defined as:

- Vertex Shader:

$$v_{position} = modelMatrix \cdot vec4(position, 1.0) \tag{18}$$

- Fragment Shader:

$$color = mix(bottomColor, topColor, max(normalize(v_{worldPosition}).y, 0.0)) \tag{19}$$

8.2 Phong Lighting

Implemented in shaders for dynamic lighting effects. The lighting equation is:

$$I = k_a i_a + k_d i_d(\vec{L} \cdot \vec{N}) + k_s i_s(\vec{R} \cdot \vec{V})^n \tag{20}$$

9 Graphics Pipeline Implementation

9.1 Render Loop Algorithm

1: Update physics state

$$\vec{F}_{total} = \vec{F}_{thrust} + \vec{F}_{lift} + \vec{F}_{drag} + \vec{F}_{gravity} \tag{21}$$

2: Update velocities

$$\vec{v}_{new} = \vec{v}_{old} + \frac{\vec{F}_{total}}{m}\Delta t \tag{22}$$

Update positions

$$\vec{p}_{new} = \vec{p}_{old} + \vec{v}_{new}\Delta t \tag{23}$$

Update rotations via quaternion

$$q_{new} = q_{old} \otimes \Delta q \tag{24}$$

Update particle systems
Perform collision detection
Update camera matrix

$$M_{view} = M_{translation} M_{rotation} \tag{25}$$

Render scene

10 Lighting and Materials

10.1 Phong Lighting Model

Implemented using:

$$I = k_a i_a + k_d i_d(\vec{L} \cdot \vec{N}) + k_s i_s(\vec{R} \cdot \vec{V})^n \tag{26}$$

where:

- k_a, k_d, k_s = material coefficients
- i_a, i_d, i_s = light intensities
- $\vec{L}$, $\vec{N}$, $\vec{R}$, $\vec{V}$ = light, normal, reflection, and view vectors

11 Camera System

Camera follows aircraft using smooth interpolation:

$$p_{camera} = lerp(p_{current}, p_{target}, \alpha) \tag{27}$$

where $\alpha = 0.1$ for smooth transition.

12 Collision Detection

Ground collision implemented using height-based detection:

$$collision = p_y < terrain_{height}(p_x, p_z) + offset \tag{28}$$

13 Performance Optimizations

13.1 LOD System

Level of detail calculation:

$$detail_{level} = \left\lfloor \log_2\left(\frac{distance}{base_{distance}}\right) \right\rfloor \tag{29}$$

14 Results

The implementation achieves:

- 60+ FPS rendering performance
- Realistic flight physics
- Particle system with 1000+ active particles
- Dynamic terrain generation
- Smooth camera transitions
- Real-time collision detection
- HUD and minimap integration
- Custom shader effects for sky and lighting

15 Conclusion

This implementation demonstrates the practical application of advanced computer graphics concepts, combining mathematical principles with real-time rendering techniques to create an interactive flight simulation environment.