Task 1.7: Space Vehicle Animation

COMP3811 Computer Graphics – Coursework 2

Overview. This section details the implementation of a procedural animation system for the space vehicle. The animation simulates a rocket launch sequence with a curved trajectory, dynamic acceleration, and automatic orientation alignment. The system is interactive, allowing the user to control the playback state, and integrates seamlessly with the dynamic lighting system developed in Section 1.6.

Interaction Design. The animation state is managed through a state machine within the main application loop, controlled by user input via `GLFW` key callbacks:

- **Start/Pause (F Key):** Pressing `F` toggles the animation state. If the animation is inactive, it initializes the `animationActive` flag and resets the timer. If already active, it toggles the `animationPaused` flag, allowing users to freeze the vehicle mid-flight for inspection.
- **Reset (R Key):** Pressing `R` completely resets the simulation. The vehicle returns to its launch position (`vehicleOriginalPos`), the timer is zeroed, and the animation logic is disabled until the next start command.

Trajectory Generation. The vehicle's flight path is defined procedurally using parametric equations based on the elapsed animation time t. To satisfy the requirement for a "curved path where it slowly accelerates from a standstill," the trajectory combines three distinct motion components:

1. **Vertical Ascent (Y):** The height is modeled to simulate a rapid initial climb that gradually levels off, mimicking a gravity turn. This is achieved using a quadratic damping function: $y(t) = H \cdot t \cdot (1 - k \cdot t)$, where H scales the maximum height.
2. **Forward Acceleration (Z):** Forward movement follows a cubic function $z(t) = -D \cdot t^3$. This ensures the velocity starts at zero and increases non-linearly, providing the visual effect of the vehicle building up thrust and overcoming inertia.
3. **Horizontal Arc (X):** To add complexity to the curve, a sine wave function $x(t) = A \cdot \sin(\frac{\pi}{2}t)$ introduces a lateral arc, resulting in a 3D corkscrew-like trajectory rather than a simple 2D parabola.

The system handles these phases using a normalized time parameter that transitions from an acceleration phase (quadratic interpolation) to a coasting phase (linear interpolation) after a fixed duration (3.0 seconds).

Orientation and Alignment. To ensure the vehicle always faces its direction of travel, the system calculates the instantaneous velocity vector $\mathbf{v}$ by computing the analytical derivatives of the position functions ($dx/dt, dy/dt, dz/dt$) with respect to time.

The orientation is derived by aligning the vehicle's local up vector ($\mathbf{up} = (0, 1, 0)$) with the normalized velocity vector $\hat{\mathbf{v}}$. This alignment is computed using the axis-angle rotation method:

1. **Rotation Axis:** $\mathbf{axis} = \mathbf{up} \times \hat{\mathbf{v}}$
2. **Rotation Angle:** $\theta = \arccos(\mathbf{up} \cdot \hat{\mathbf{v}})$

A rotation matrix is constructed from this axis and angle using Rodrigues' rotation formula and applied to the vehicle's model matrix. This ensures the rocket tilts naturally into the curve as it accelerates.

Dynamic Lighting Integration. The three point lights implemented in Section 1.6 are attached to the vehicle's local coordinate system. In every frame of the animation, the light positions are updated by adding their constant local offsets to the vehicle's current world position. These updated positions are passed to the shader uniforms, ensuring the coloured glow travels with the rocket, illuminating the launchpad during takeoff and the terrain during flight.

Visual Result.

[Figure 1.7.1: The space vehicle mid-flight, demonstrating the curved trajectory and orientation alignment. The coloured point lights can be seen traveling with the vehicle, illuminating the surrounding air.]