Project 2 — 3D Hierarchical Modelling & Projections

Students: **Ilia Taitsel (67258)**, **Oleksandra Kozlova (68739)**

November 11, 2025

Variable-Dependent Parameters

Only transforms based on user input, camera control, or runtime updates are shown below. Constant values (e.g., base or cabin scale) defined in `scene.json` remain fixed, and they are omitted.

Tank Translation

The root node `tank.translation` controls the global position of the tank. Movement occurs along the world x-axis when the user presses:

$$\texttt{q} : \text{forward}, \quad \texttt{e} : \text{backward}.$$

Translation is updated by a fixed speed constant:

$$\text{tank.translation}[0] \leftarrow \text{tank.translation}[0] \pm \texttt{TANK\_SPEED}, \quad \texttt{TANK\_SPEED} = 0.1.$$

This translation affects all child nodes (base, cabin, wheels (discussed below), etc.) through the hierarchical model.

Cannon Rotation

The cannon is mounted on `the cannon_base`, rotating around its z-axis. Its pitch is restricted to realistic limits:

$$\theta_{\min} = -17^\circ, \qquad \theta_{\max} = 80^\circ.$$

In key press:

$$\texttt{w} : +\texttt{CANNON\_STEP} \text{ (raise)}, \qquad \texttt{s} : -\texttt{CANNON\_STEP} \text{ (lower)},$$

and we apply:

$$\text{cannon\_base.rotation}[2] \leftarrow \min\big(\text{cannon\_base.rotation}[2] + 5^\circ,\, 80^\circ\big).$$

$$\text{cannon\_base.rotation}[2] \leftarrow \max\big(\text{cannon\_base.rotation}[2] - 5^\circ,\, -17^\circ\big).$$

Wheel Rotation

All twelve wheel nodes (`wheel1--wheel12`) rotate around their y-axis when the tank moves. Their angular step is:

$$\text{rotationSpeed} = \begin{cases} +20^\circ, & \texttt{q} \text{ (forward)}, \\ -20^\circ, & \texttt{e} \text{ (backward)}. \end{cases}$$

Each wheel is updated as:

$$\text{wheel.rotation}[1] \leftarrow (\text{wheel.rotation}[1] + \text{rotationSpeed}) \bmod 360.$$

Ground Tiles

Ground tiles are generated in two nested loops:

$$i, j \in \{-N/2, \ldots, N/2 - 1\}, \quad N = \texttt{TILES\_PER\_SIDE} = 24.$$

Each tile block_{ij} uses constants:

$$s = \texttt{TILE\_SIZE} = 0.5, \qquad h = \texttt{TILE\_HEIGHT} = 0.05.$$

Its transforms are as follows:

$$T_{ij} = (i\,s, -h/2, j\,s), \qquad S = (s, h, s),$$

and the color alternates according to:

$$(i + j) \bmod 2 = \begin{cases} 0 \Rightarrow \text{GRAY}, \\ 1 \Rightarrow \text{WHITE}. \end{cases}$$

Camera Zoom (Mouse Wheel)

The zoom factor `zoom` starts at 1 and is modified by the mouse wheel:

$$\text{if scroll up: } \texttt{zoom} \leftarrow \texttt{zoom}/\texttt{ZOOM\_STEP}, \qquad \text{if scroll down: } \texttt{zoom} \leftarrow \texttt{zoom} \cdot \texttt{ZOOM\_STEP},$$

with $\texttt{ZOOM\_STEP} = 1.1$. This affects the orthographic and perspective projections:

$$\text{ortho}(-az,\, az,\, -z,\, z,\, n,\, f), \quad \text{or} \quad \text{perspective}(\text{fovy} = 60^\circ z/2,\, a,\, n,\, f),$$

where a is the viewport aspect ratio, z is the zoom factor and (n, f) are near and far clipping planes.

View 4 Parameters (Axonometric / Oblique)

For axonometric mode:

$$\theta = 35^\circ,\ \gamma = 45^\circ, \quad \text{updated by } \texttt{←/→}(\pm\theta),\ \texttt{↑/↓}(\pm\gamma).$$

For oblique mode:

$$\alpha = 45^\circ,\ l = 0.5, \quad \text{updated by } \texttt{←/→}(\pm\alpha),\ \texttt{↑/↓}(\pm 0.1 \text{ on } l).$$

They are toggled with key `8`, while key `9` switches between parallel and perspective projections.

Hole Parameters (Our Game Feature)

The position and radius of the hole vary dynamically:

$$(x_h, z_h) \in [-5, 5]^2, \quad r \in [r_{\min}, r_{\max}] = [0.08, 0.8].$$

Its behavior is governed by:

$$\Delta r = 0.06, \quad \texttt{hole.shrinking} \in \{false, true\}.$$

On each hit:

$$r \leftarrow \begin{cases} \max(r_{\min}, r - \Delta r), & \text{if shrinking}, \\ \min(r_{\max}, r + \Delta r), & \text{if growing}. \end{cases}$$

A new random position (x_h, z_h) is generated after every successful hit.

Extra Feature: "Tomato Strike" Game

We implemented an additional interactive gameplay element: a **dynamic ground hole** that changes position and size after each hit. This feature introduces challenge and progression, as well as a scoring system stored across browser sessions.

Hole Behaviour

The hole is rendered as a thin black cylinder on the ground plane ($y \approx 0$). At initialization, its position and radius are:

$$(x_h, z_h) \in [-5, 5]^2, \qquad r = r_{\max} = 0.8.$$

Each time a tomato projectile hits the hole, a new random position is generated:

$$(x_h, z_h) \leftarrow (\text{rand}(-5, 5),\ \text{rand}(-5, 5)).$$

The radius r changes between the shrinking and growing phases:

$$r_{\min} = 0.08, \quad r_{\max} = 0.8, \quad \Delta r = 0.06.$$

When `hole.shrinking` is true:

$$r \leftarrow \max(r_{\min},\ r - \Delta r),$$

otherwise:

$$r \leftarrow \min(r_{\max},\ r + \Delta r).$$

The direction is changed whenever a limit is reached.

Hit Detection

A hit is detected when a tomato's center (x, z) is close to the ground and inside the hole's radius:

$$|y| < 0.05, \qquad (x - x_h)^2 + (z - z_h)^2 \le r^2.$$

Upon a hit, the tomato is removed, the score is increased, and the hole is relocated.

Scoring System

The score depends on the current hole radius (difficulty) and the active hit streak:

$$\text{points} = \text{round}\big(10 \cdot (0.5 + 0.5\,\tfrac{r_{\max}}{r}) \cdot (1 + 0.25\,\text{streak})\big).$$

Variables used:

$$\texttt{BASE\_POINTS} = 10, \quad \texttt{STREAK\_BONUS} = 0.25.$$

Each successful hit adds to both `score` and `streak`; missed tomatoes reset the streak counter.

Score Persistence Feature

An additional improvement was implemented to **preserve the best score** between page reloads using the browser's `localStorage` API. When the user achieves a new best score, it is stored permanently with:

`localStorage.setItem("bestScore", String(bestScore))`.

At page load, the saved value is restored using:

`bestScore = Number(localStorage.getItem("bestScore") || 0)`.

This allows the best score to remain available even after refreshing or reopening the page. The on-screen HUD automatically displays the values of:

`Score`, `Best`, `Streak`,

and updates them in real time.

Access

The feature is always active during normal play:

- Press `z` to fire tomatoes.
- Each hit updates score, streak, and hole size/location.
- The `Best Score` is automatically saved across sessions.
- Keys `x` and `b` reset the current and best scores respectively.

Scene Graph

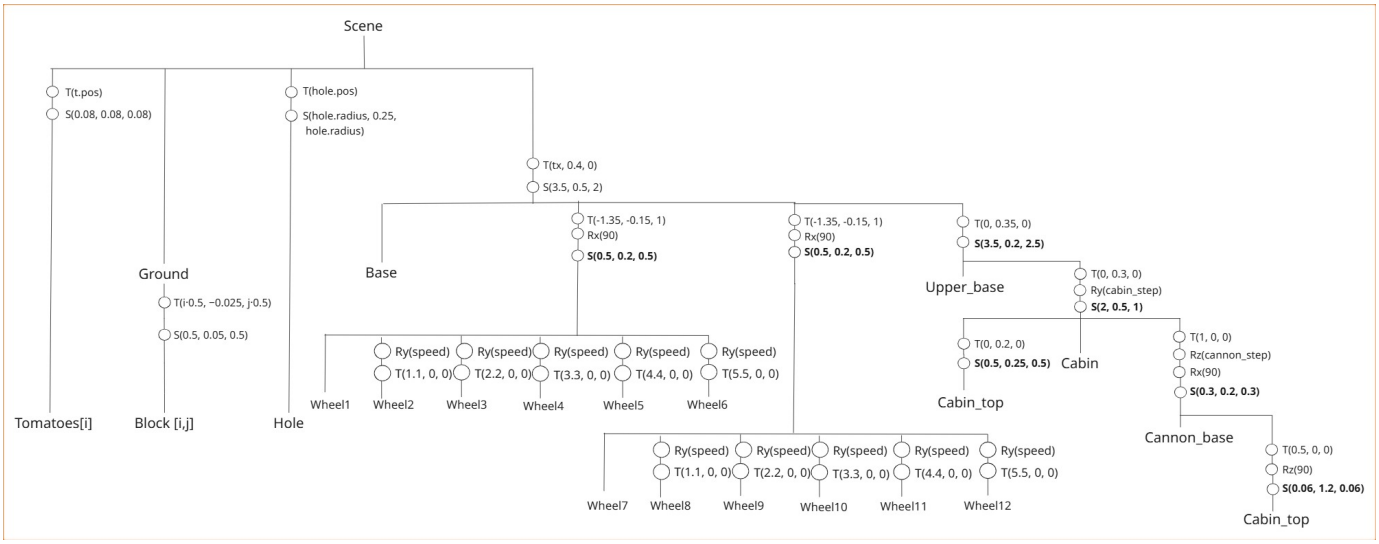

Figure 1: Scene graph of the tank, ground, hole, and projectile hierarchy.

Figure 1. The updated scene graph illustrates the complete hierarchy of our model. Each node contains its local transformations: translation (`T`), rotation (`R`), and scale (`S`). Dynamic transformations such as `T(tx, 0.4, 0)`, `Ry(cabin_step)`, `Rz(cannon_step)`, and `Ry(speed)` represent variables controlled at runtime through user input. Static values define the tank's structure, ground tiles, hole, and tomato geometry. The tank subtree contains all its parts — base, upper base, cabin, cannon base, and cannon — with wheels rotating when the tank moves. Scale matrices shown in **bold** denote nodes where `inheritScale = false`; in these cases, the parent's scale is compensated by multiplying with its inverse, but this correction is omitted here for clarity.