Camera Path & Export Design Note

Camera / Path Representation

The camera path is stored as an ordered array of immutable keyframes rather than a continuous recording. Each keyframe captures a complete camera state at a specific moment—position (Vector3), orientation (Quaternion), and field of view (FOV). Array order defines time: index 0 is the start, index N−1 is the end.

This discrete representation was chosen for three reasons: (1) **editability**—keyframes can be inserted, removed, or reordered without recomputing the path; (2) **compactness**—only meaningful poses are stored, avoiding redundant samples; and (3) **determinism**—the animation is defined by a finite set of exact values rather than time-based sampling.

Rotations are stored as normalized quaternions instead of Euler angles to avoid gimbal lock and to enable smooth spherical linear interpolation (slerp). Camera motion is constrained to yaw and pitch to maintain a stable world-up orientation; roll is explicitly eliminated, so arbitrary tilt (e.g., Dutch angles) is not supported. Position is interpolated using Catmull–Rom splines so the path passes smoothly through all keyframes, while rotation is interpolated by extracting and interpolating yaw and pitch before reconstructing a roll-free quaternion. FOV is linearly interpolated.

Both preview playback and timeline scrubbing evaluate the same stateless (doesn't store past data) interpolation function using normalized time $t \in [0,1]$. Because keyframes are immutable and interpolation has no side effects, the same inputs always produce the same camera state.

Export Pipeline

Export converts the abstract camera path into a concrete MP4 video. When the user starts an export, they specify a duration; total frames are computed as duration × FPS (default 30). For each frame i, normalized time is sampled as $t = i / (totalFrames - 1)$, ensuring the first and last frames exactly match the first and last keyframes.

For every frame, the camera state is evaluated using the same interpolation logic as preview playback. The scene is rendered to an offscreen WebGLRenderTarget at a fixed resolution (e.g., 1280×720), and pixels are read back, encoded as JPEG, and sent to a Node.js server. The server streams frames into FFmpeg, which encodes the final MP4. Progress is reported as framesSent / totalFrames, and the user can safely cancel at any time, triggering cleanup of partial output and FFmpeg processes.

Using an offscreen render target ensures resolution independence, avoids UI disruption during export, and keeps output deterministic regardless of the user's display size. Editor-only features (such as frustum gizmos) are hidden during export to ensure a clean final render.

Performance Consideration: GPU Readback Cost

Export performance is dominated by GPU-to-CPU readback, as each rendered frame must be copied from GPU memory into system memory before encoding and upload. This readback introduces a synchronization point that limits throughput regardless of rendering speed. To manage this cost, frames are rendered sequentially to an offscreen target at a fixed FPS and streamed incrementally to the server, avoiding large client-side buffers while ensuring deterministic and stable export behaviour.