Programmable World Model

Zheng-Hui Huang[△,⋆], Guixu Lin[△,⋆], Jiacheng Lin, Yi-Chuan Huang[△], Ruihan Yu[△], Muyao Niu[△], Siqi Yang[△], Yu-Lun Liu, Yung-Yu Chuang, Kaipeng Zhang[△,†], Zhixiang Wang[△,†]

[△]Alaya Lab

Recent video world models generate increasingly realistic and interactive visual experiences, yet lack reliable mechanisms for maintaining persistent world state and enforcing programmable rules over extended interactions. We introduce **Programmable World Model**, a framework that decouples world-state evolution from visual observation generation. An agent translates natural-language instructions into executable programs that specify entity states and state-transition rules, enabling direct control over individual entities and their interactions. A lightweight engine executes these programs to update and maintain an explicit, persistent global world state, including off-screen entities and non-visual attributes. To connect world state with visual generation, we introduce state-augmented 3D oriented bounding boxes (OBBs) as an intermediate representation. This representation, together with the target camera trajectory, is deterministically compiled into pixel-aligned spatiotemporal conditioning signals for a pretrained video model serving as the generative renderer. This design allows users to create playable games with predefined mechanics, direct control over individual entities, and persistent world state throughout gameplay. We further introduce CombatStateBench, a benchmark for evaluating programmable world models. On CombatStateBench, our method achieves 94% Count Accuracy and 98% State Accuracy, substantially outperforming existing interactive video world models while supporting coherent long-horizon generation. These results demonstrate the effectiveness of separating explicit state evolution from generative rendering for building persistent, programmable worlds.

Project page: https://alaya-lab.github.io/pwm
Code: https://github.com/AlayaLab/pwm
Correspondence: Zhixiang Wang (Project Lead), Kaipeng Zhang
Note: ⋆ denotes equal contributions
Date: September 9, 2026

ALAYA LAB

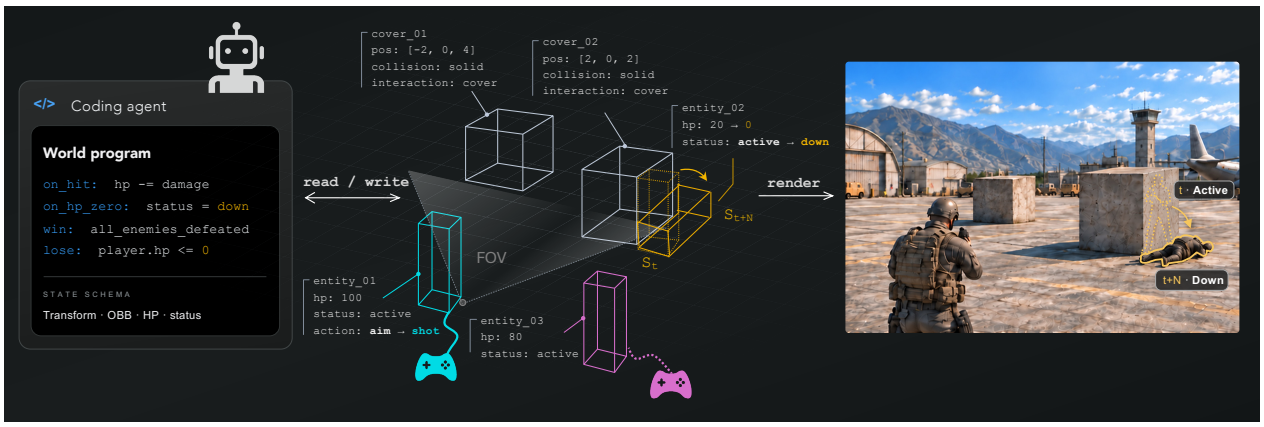

Figure 1 Overview of the programmable world model. Given a single reference image and a user description, a coding agent generates executable programs that specify entity states and interaction rules. A lightweight engine executes these programs to maintain and evolve an explicit world state, represented by state-augmented 3D OBBs. This representation is compiled into conditioning signals for a generative renderer, which produces realistic visual observations consistent with the world state. This design allows users to create playable games by defining game mechanics in advance, directly controlling individual entities, and maintaining persistent world state throughout gameplay.

1 Introduction

Recent video world models [2, 16, 17, 18, 20] seek to build interactive world engines on top of generative video models [19, 7]. By predicting subsequent observations from visual histories and user actions, they synthesize increasingly realistic and responsive environments, opening new possibilities for immersive world creation.

However, turning video world models into interactive world engines requires capabilities beyond generating plausible observations. First, existing control interfaces primarily specify cameras, actions, or high-level prompts, offering limited support for directly addressing and manipulating individual entities. Second, these models generally lack an explicit, persistent global state that can be accessed and updated independently of the current view. Such state must account for off-screen entities and nonvisual information, including inventory, task progress, and interaction history, which is also important for a multiplayer world engine. Third, users cannot readily program the rules governing world evolution, such as specifying the conditions under which a door opens or how an interaction affects other entities. Prompting a desired outcome does not establish an executable rule that consistently governs subsequent interactions.

We introduce **Programmable World Model**, a framework that decouples world-state evolution from visual observation generation (Figure 1). A coding agent translates user instructions into executable programs that define entity states and world rules, while a generative renderer produces observations conditioned on the evolving state. Connecting these components requires an interface that is compact and directly editable by programs, yet sufficiently expressive to guide visual generation. Our central question is therefore: *what representation supports programmable world-state evolution while preserving the spatial and semantic structure needed for visual generation?*

Representation choice shapes explicit controllability, the cost of constructing and evolving world state, and the potential mismatch between training and inference conditions. As illustrated in Figure 2, text descriptions are easy to author but do not by themselves impose geometric or other dense constraints, while 2D boxes and masks provide image-space control tied to particular viewpoints [21]. Detailed 3D scenes and structural proxies [6], together with dense rendering conditions such as G-buffers [11, 12], enable finer-grained control but demand richer supervision and more complex inference-time construction. Crucially, training representations are extracted from observed dynamics, whereas inference requires constructing structural trajectories from desired state transitions. Specifying that a character falls, for example, is simpler than generating its detailed body poses and limb trajectories. More detailed representations therefore require the explicit system to resolve additional motion and geometry, and the resulting controls may differ from those extracted during training.

Following these requirements, we represent each entity using a *state-augmented 3D oriented bounding box (OBB)*. Each OBB specifies an entity's position, extent, and orientation in a shared world coordinate system using a compact, fixed set of geometric parameters, without requiring a mesh, material, skeleton, or predefined animation. We augment this geometry with persistent identity, semantic category, dynamic state, and appearance information. Inspired by advances in generative rendering [11, 12], we communicate this structured spatial information through rasterized controls. While approaches such as RenderFormer [23] directly map structured scene tokens to images, we explicitly resolve camera projection and entity-to-pixel correspondence through a deterministic *state compiler*. The compiler projects the evolving OBBs along the target camera trajectory and rasterizes them into pixel-aligned spatiotemporal controls. These controls guide entity identity, semantics, orientation, motion, and state changes, while the video model synthesizes the fine geometry, appearance, articulation, and secondary dynamics left unspecified by the representation.

Given a single reference image and a natural-language description, a VLM-based coding agent constructs the initial entity layout, assigns states and attributes, and writes programs defining interactions, event triggers, and objectives. During interaction, a state executor applies player actions and events according to these rules. The state compiler converts the updated world state into rendering controls, and the generative renderer produces the next observations. Users can revise world behavior through further instructions to the coding agent. To train the renderer, we develop a video data curation pipeline that recovers the spatial and semantic supervision required by this interface.

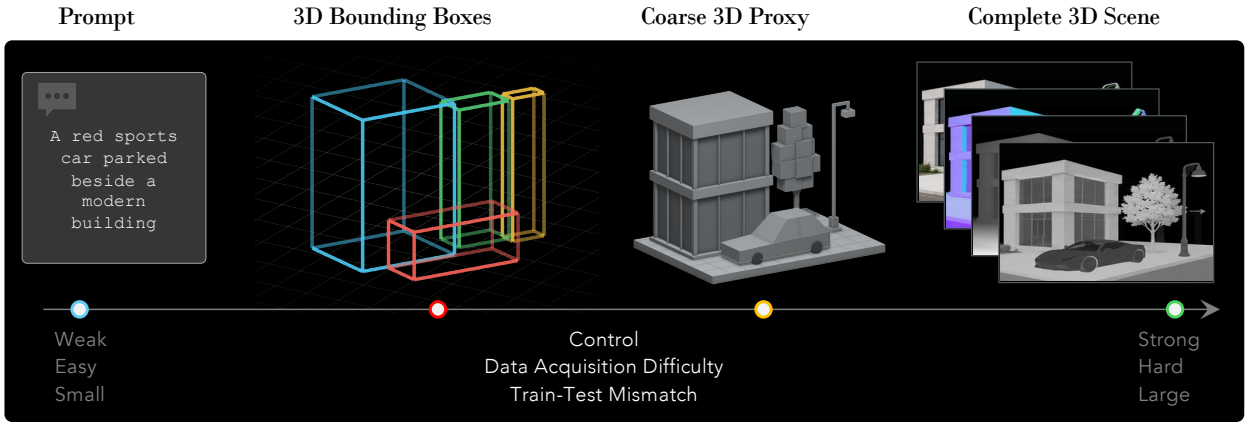

Figure 2 State representation trade-offs. Candidate representations range from lightweight semantic descriptions to detailed geometric structures [11]. Greater structural detail enables finer-grained control but increases the cost of training-data annotation and inference-time state construction and evolution. Moreover, representations extracted from observations or recorded from engines during training may differ from those constructed programmatically at inference time, creating a potential training–inference mismatch. We adopt state-augmented 3D OBBs as an intermediate abstraction that supports explicit world-space control while leaving fine geometry, articulation, and secondary visual dynamics to the generative renderer.

We introduce CombatStateBench, a controlled benchmark for evaluating consistency between generated videos and engine-maintained world states. Through combat scenarios with diverse camera and entity motion, it measures whether generated videos preserve visible alive-character counts and visually realize death states.

Our contributions are threefold:

- We introduce **Programmable World Model**, which decouples executable world-state evolution from generative rendering to enable entity-level control, persistent state management, and user-programmable world rules. State-augmented 3D OBBs and a deterministic state compiler connect this editable world state to video generation through view-consistent, pixel-aligned spatiotemporal controls.
- We develop a data curation pipeline that extracts spatial and semantic supervision from videos, providing a practical path toward scaling training data for programmable generative worlds.
- We introduce CombatStateBench, a controlled benchmark for evaluating consistency between generated videos and engine-maintained world states, focusing on visible alive-character counts and the visual realization of death states under diverse camera and entity motion.

2 Representation Trade-offs

The choice of representation shapes what a system can explicitly control, the cost of achieving that control, and the potential mismatch between representations extracted from training observations and those constructed programmatically at inference time. As illustrated in Figure 2, candidate representations span a spectrum from lightweight text descriptions to detailed 3D structures. Increasing structural detail enables finer-grained control over entity geometry, spatial relationships, and their evolution over time. However, richer structural supervision is more costly to obtain, and more structural degrees of freedom must be explicitly maintained and evolved during inference. More importantly, representations extracted from observed dynamics during training may differ from those constructed programmatically from high-level state transitions at inference time, creating a potential training–inference mismatch. We therefore seek an intermediate level of abstraction that supports explicit controllability and training–inference alignment while keeping training-data acquisition and inference-time evolution tractable.

Lightweight representations offer limited explicit control over world-space geometry. Text descriptions can convey entity categories, attributes, and high-level states, but do not by themselves impose geometric

constraints on an entity's position, extent, or orientation in a shared world coordinate system. 2D bounding boxes and masks provide more direct spatial control, but operate in image space: their positions, scales, and visible regions change with the camera viewpoint. These representations therefore constrain individual observations without defining an underlying world-space structure that can be deterministically reprojected across views. For a programmable world, it is more natural to specify entity geometry in a shared world coordinate system and derive its projection in each observation from the target camera.

At the other end of the spectrum, greater structural detail does not necessarily yield a more suitable representation. Complete 3D scenes, articulated entity models, and dynamic 3D geometry encode finer-grained structure than entity-level bounding boxes, while G-buffers provide dense, view-dependent surface information. Such detail supports more precise control but also introduces additional costs. First, training requires richer and more accurate structural supervision, increasing data acquisition and processing costs and making it harder to scale to open-domain settings. Second, the system must explicitly specify and evolve more structural degrees of freedom during inference, taking responsibility for geometric and dynamic details that a coarser representation would leave to the generative model.

Consider a character transitioning from standing to falling. A coarse representation may only need to describe changes in global position, orientation, and spatial occupancy, whereas a complete articulated or dynamic 3D representation must additionally characterize high-dimensional structural changes such as body pose, limb motion, and local deformation. As the representation becomes more detailed, the system gradually moves from specifying *what changes in the world* to specifying *how that change unfolds geometrically and dynamically*. In conventional 3D environments equipped with complete animation systems or physics simulators, such details can be produced by existing simulation mechanisms. Our focus, however, is on open-domain programmable generative worlds, where we aim to rely on generative models to realize visual dynamics rather than reconstruct a complete 3D animation and simulation stack.

Under this setting, the same representation is also obtained through fundamentally different paths during training and inference. In the training data, the dynamic process has already been realized, and the corresponding structural representation therefore describes an already realized evolution:

$$\text{Training: realized dynamics} \longrightarrow \text{structural representation.} \tag{1}$$

Inference proceeds in the opposite direction. The system first receives a high-level state transition and must actively produce the corresponding time-varying structural representation before the visual outcome is generated. The generative model then realizes these structural constraints as visual dynamics:

$$\text{Inference: state transition} \longrightarrow \text{structural representation} \longrightarrow \text{generated dynamics.} \tag{2}$$

This training-inference asymmetry means that a representation that can be obtained during training is not necessarily equally easy to produce and evolve at inference time. Finer-grained representations provide more precise explicit control, but also require the system to determine more structural degrees of freedom and their temporal evolution from high-level state transitions. As the representation approaches complete articulated motion or dynamic 3D geometry, this process increasingly resembles the high-dimensional dynamic realization problem addressed by conventional animation, motion generation, or physical simulation.

From this perspective, representation choice determines the boundary between *explicit structural control* and *generative dynamic completion*. A representation that is too weak leaves entity positions, world-space structure, and state-dependent changes that should be explicitly constrained to the generative model. A representation that is too strong not only requires more demanding training supervision, but also forces the system to explicitly determine large amounts of low-level geometric and dynamic detail at inference time. Pretrained video models already provide strong visual and dynamic priors for completing precise shape, texture, material, articulation, local deformation, illumination, and secondary motion. We therefore explicitly represent only the structure necessary for entity persistence, world-space consistency, and state-dependent changes, while leaving finer-grained dynamic realization to the generative model.

Following this principle, we adopt *state-augmented 3D oriented bounding boxes (OBBs)* as an intermediate representation. A 3D OBB specifies an entity's position, extent, and orientation in a shared world coordinate

system, and is associated with persistent identity as well as relevant semantic and dynamic states. Compared with text, 2D bounding boxes, and masks, OBBs provide a view-independent and reprojectable world-space scaffold. Compared with more explicit representations such as G-buffers, complete 3D scenes, articulated models, or dynamic 3D geometry, they restrict the structure that must be explicitly represented and evolved to a compact entity-level space. Time-varying OBBs can express coarse structural changes in entity position, orientation, and spatial occupancy, while internal articulation, local deformation, and other fine-grained dynamics are completed by the generative renderer under these constraints.

3 Related Work

3.1 Interactive Video World Models

Recent video world models have made rapid progress toward interactive visual environments, supporting action-conditioned generation, controllable camera motion, real-time interaction, and increasingly long-horizon rollouts [8, 16, 18]. These models are typically trained primarily with pixel-level generation objectives, which supervise whether the predicted observations are visually plausible but do not explicitly require the model to maintain a persistent, structured world state over time. As a result, entities, attributes, relations, and interaction outcomes may be represented only implicitly in the generative context, without being organized into an authoritative state that must remain consistent across occlusion, camera motion, or long interaction horizons. Recent systems introduce temporal or spatial memory to improve visual consistency over extended rollouts [17, 20, 22], but such memory is still optimized primarily for observation generation rather than for preserving executable world facts. Our work instead separates these two responsibilities: a lightweight engine explicitly maintains and advances the canonical world state, while the video model serves as a generative renderer conditioned on the resulting state.

3.2 Explicit-state World Modeling

Recent work has begun to move beyond purely observation-centric world modeling by explicitly representing the internal state of interactive environments. StatePlay [14] jointly predicts visual observations and game-state variables, allowing the predicted state to guide visual generation and improve mechanics consistency. However, because the state itself is predicted by the model, state errors can directly lead to incorrect world updates and may accumulate over long interaction horizons. MASS [3] further introduces an authoritative typed state for multiplayer world modeling and separates state dynamics from view rendering. Its shared state, however, is advanced by a learned Logic Engine, so the authoritative state evolution still depends on learned transition dynamics and remains susceptible to transition prediction errors. In contrast, our framework maintains a canonical world state in a lightweight engine and executes state transitions according to explicit rules, making the world state directly programmable, editable, and verifiable.

3.3 Generative Rendering

Generative rendering leverages learned generative models to synthesize photorealistic visual observations from structured scene conditions, reducing reliance on conventional graphics pipelines. Different approaches adopt renderer-facing representations with varying levels of geometric explicitness. DiffusionRenderer [12] performs diffusion-based forward and inverse rendering using G-buffers that encode geometric and appearance-related scene attributes. The AlayaRenderer series [11, 13] extends this paradigm to dynamic video and world rendering. AlayaRenderer [11] focuses on high-fidelity generative rendering of dynamic worlds, while AlayaRenderer-Flash [13] further improves generation efficiency for real-time interactive rendering. These works demonstrate that generative priors can synthesize rich appearance, material, lighting, and dynamic details from explicit structural rendering signals.

Moving toward lighter structural conditioning, Coarse-to-Real [6] synthesizes dynamic scenes from coarse 3D proxies, using simplified geometry to constrain scene layout, camera motion, and object trajectories while leaving fine-grained geometry, appearance, articulation, and secondary dynamics to the generative model. Together, these works illustrate a spectrum of generative rendering interfaces, where more explicit

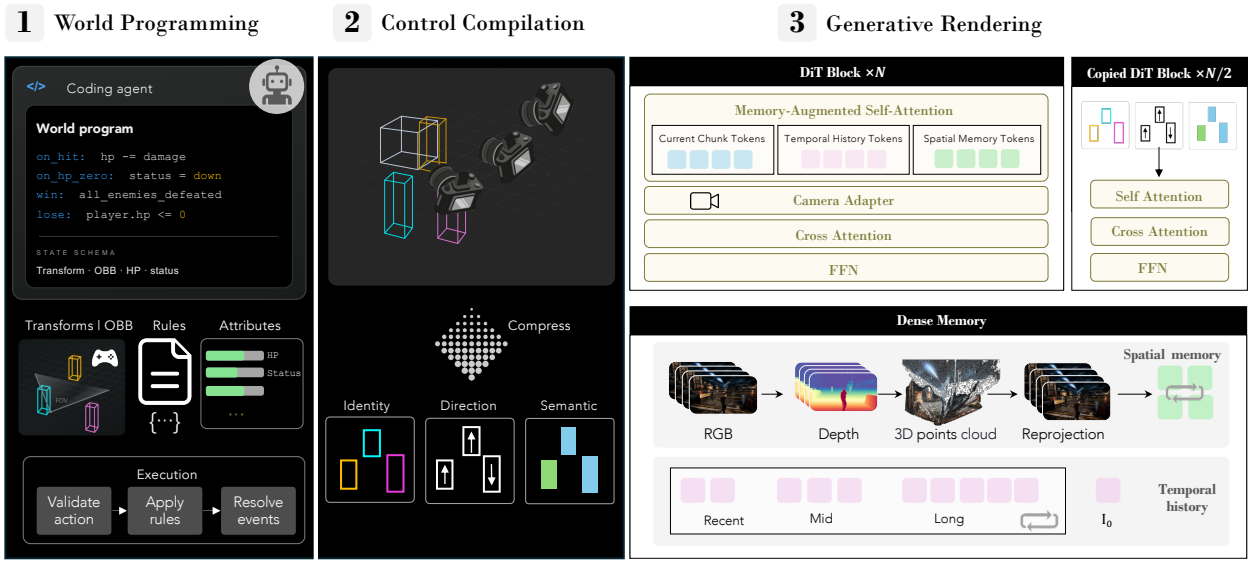

Figure 3 Architecture of the programmable world model. (1) Agent-orchestrated world programming starts from natural-language specifications to instantiate an executable canonical world state, which is maintained and advanced by a lightweight engine according to player actions and world rules. (2) Control compilation projects state-augmented 3D OBBs under the target camera into pixel-aligned identity, semantic, and motion-direction controls. (3) Generative rendering conditions a pretrained camera-controlled video generation model on these structured controls and visual history to synthesize the next video chunk. Completed chunks are incorporated into temporal and geometry-aligned spatial memories to support long-horizon generation.

representations provide stronger structural constraints, while lighter representations delegate a larger fraction of visual and dynamic realization to the generative prior.

4 Method

4.1 Problem Formulation

We consider an interactive generative world initialized from a visual observation I_0. At interaction step t, a player takes an action a_t, and the system maintains a canonical world state s_t that records the persistent information required to execute world interactions. The overall interaction loop is

$$s_t \xrightarrow[a_t]{F} s_{t+1} \xrightarrow[C_{t+1}]{P} M^{\text{ctrl}}_{t+1} \xrightarrow{G} I_{t+1}, \tag{3}$$

where F denotes the state transition executed by the lightweight engine, C_{t+1} is the target camera at the next interaction step, P is the deterministic state compiler that projects the updated world state under C_{t+1} into camera-aligned spatial controls, and G denotes the generative renderer that synthesizes the corresponding visual observation I_{t+1}. We detail the representation, programming, and state transition F of the programmable world in Section 4.2, the state compiler P in Section 4.3, and the generative renderer G in Section 4.4.

4.2 Agent-Orchestrated Box World

World representation and initialization. We represent the programmable world using a set of persistent entities embedded in 3D space. Each entity is grounded by a 3D oriented bounding box (OBB), which specifies its position, extent, and orientation, and is associated with state variables such as its identity, semantic category, and functional attributes. For example, a character may have a persistent identity, a 3D pose, a health value, and a faction label. The world additionally stores relations between entities, such as hostility or ownership,

together with executable rules that determine how player actions and events modify these states. Formally, we denote the canonical world state at interaction step t as

$$s_t = (\mathcal{E}_t, \mathcal{A}_t, \mathcal{Q}_t; \mathcal{R}_t), \tag{4}$$

where $\mathcal{E}_t$ contains the persistent entities and their 3D poses, $\mathcal{A}_t$ contains their semantic and functional attributes, $\mathcal{Q}_t$ contains inter-entity relations, and $\mathcal{R}_t$ denotes the executable world rules. Together, these components define both the current world configuration and how it can evolve under interaction.

Given the initial observation I_0, an off-the-shelf 3D detector first recovers the visible entities and their geometric layout, providing the initial 3D OBBs and persistent entity identifiers. These detections initialize the geometric component $\mathcal{E}_0$. The remaining attributes, relations, and rules are then specified by the agent orchestrator, as described below.

Agent-orchestrated world programming. An agent orchestrator combines the recovered 3D layout with the player's natural-language world description q to produce an engine-readable world program. The program specifies the initial states and attributes of the detected entities, their relations, the actions they support, and the rules governing how actions and events update the world. It may further include global constraints, event triggers, and task objectives. The program is grounded to the detected entities through their persistent identifiers and is executed by the engine to instantiate the initial canonical state s_0.

For example, for a combat scenario, the orchestrator can assign detected characters to opposing factions, initialize their health and combat states, specify valid attack relations, and define the effects of actions such as shooting, including damage, death, and objective updates. Rather than constructing a conventional game with detailed 3D assets and low-level simulation logic, the orchestrator programs a lightweight box world in which 3D OBBs provide the geometric scaffold and structured states, relations, and rules define the interaction semantics.

Engine execution. Once instantiated, the world program defines how the canonical state evolves under interaction. The engine transition F introduced in Eq. (3) operates on the current state s_t and player action a_t:

$$s_{t+1} = F(s_t, a_t). \tag{5}$$

At each interaction step, the engine evaluates the action against the current attributes, relations, and executable rules $\mathcal{R}_t$, checks whether the action is valid, applies its effects, and resolves any triggered events. The resulting changes to entities, attributes, relations, and potentially the rule set are recorded in the updated canonical state s_{t+1}.

The canonical world state may also contain variables that are not directly expressed in visual observations, such as health, inventory, or faction membership, yet still govern future state transitions. These latent world facts are maintained explicitly by the engine and can influence subsequent state updates and rendered outcomes. After each interaction step, the engine therefore produces an updated canonical state s_{t+1} in world coordinates. This state specifies the result of the interaction, but it is not yet a renderer-facing representation. The next stage converts its visually relevant content into spatial controls aligned with the target camera.

4.3 Control Compilation

From world states to spatial controls. Given the updated canonical state s_{t+1} produced by the engine, the state compiler P constructs a renderer-facing spatial representation without altering the underlying world state. Specifically, it extracts the renderer-relevant state variables and projects them under the target camera C_{t+1}:

$$M_{t+1}^{\text{ctrl}} = P(s_{t+1}, C_{t+1}). \tag{6}$$

The canonical state is defined with respect to the 3D world and may contain variables that have no direct spatial realization in the rendered view. The compiler therefore focuses on properties that must be spatially grounded for visual generation. In particular, it projects the entity OBBs under the target camera and

augments their projected regions with identity, semantic, and object-motion attributes. The projected OBBs specify where each entity should appear, while the associated attributes indicate which persistent instance occupies the region, which semantic category it belongs to, and in which direction the entity itself is moving.

Across frames, the displacement of a projected OBB may result from object motion, camera motion, or both. We therefore additionally provide an explicit camera-relative object-motion state derived from the entity's world-space velocity. This state describes the entity's own motion independently of the apparent displacement induced by the target camera trajectory, thereby reducing camera–object motion ambiguity.

Structured projected-box controls. A projected OBB specifies the spatial support of an entity in the target view, but its geometry alone does not encode persistent identity, semantic category, or object-motion direction. We therefore construct three spatially aligned control maps for these complementary properties.

The identity map M_{t+1}^{id} establishes persistent instance correspondence across time. During interaction, an entity may become occluded or leave and later re-enter the camera view, making it difficult to preserve its appearance using visual history alone. We assign each persistent entity identifier k_i to an identity slot that remains fixed throughout the generated sequence. We maintain a bank of K learnable identity embeddings:

$$\mathcal{E}^{\text{id}} = \left\{ q_j^{\text{id}} \right\}_{j=1}^{K}. \tag{7}$$

The embedding associated with the assigned identity slot is rasterized over the visible projection of the entity's OBB, providing a spatial key associated with its appearance reference. During training, entities within each sample are uniformly assigned distinct slots from this bank. The assignment is randomized across samples but kept fixed across all frames of the same sequence, preventing individual identity embeddings from becoming spuriously associated with a particular object category, appearance, or viewpoint. At inference time, the assigned slot is preserved for each persistent entity across interaction steps.

The semantic map M_{t+1}^{sem} provides category-level information complementary to the instance-specific identity map. Given the semantic label y_i maintained in the canonical state, a pretrained text encoder produces the corresponding semantic embedding:

$$q_i^{\text{sem}} = E_{\text{text}}(y_i). \tag{8}$$

The embedding is rasterized over the visible projection of the entity's OBB, while background locations are assigned zero features. When multiple projected boxes overlap, depth-aware rasterization retains the representation of the visible entity. Entities sharing the same semantic label therefore share the same semantic representation while remaining distinguishable through their identity slots. This semantic guidance is particularly useful for entities that are absent from the initial observation or lack a reliable appearance reference.

The direction map M_{t+1}^{dir} provides an explicit camera-relative representation of object motion. For each entity, the compiler takes its world-space velocity maintained by the engine, rotates it into the coordinate system of the target camera, and quantizes the resulting direction into one of seven states: FORWARD, BACKWARD, LEFT, RIGHT, STATIC, UP, or DOWN. We maintain a bank of learnable embeddings for these direction states:

$$\mathcal{E}^{\text{dir}} = \left\{ q_\ell^{\text{dir}} \right\}_{\ell=1}^{7}. \tag{9}$$

The embedding associated with an entity's direction state is rasterized over the visible projection of its OBB, while background and invalid regions are assigned zero features. Importantly, the direction state is computed from the entity's world-space velocity rather than from its image-space displacement. A static entity therefore remains assigned the STATIC state even when camera movement causes its projected location to change. Together with the target camera trajectory, this representation allows the renderer to separate object motion from apparent motion induced by the camera.

Finally, the identity, semantic, and direction maps are concatenated along the channel dimension to form the structured projected-box control:

$$M_{t+1}^{\text{ctrl}} = \text{Concat}\left(M_{t+1}^{\text{id}}, M_{t+1}^{\text{sem}}, M_{t+1}^{\text{dir}} \right). \tag{10}$$

The resulting representation separates *which instance* occupies a controlled region, *what semantic category* it belongs to, and *in which direction* the entity itself moves. Together with the projected OBB geometry and target camera trajectory, these maps provide explicit spatial, identity, semantic, and object-motion guidance to the generative renderer.

4.4 Generative Renderer

4.4.1 Conditional Video Generation

We build the renderer on top of LingBot-World-v1 [18], which provides a pretrained camera-controlled video generation backbone together with an existing camera-conditioning module. Let T denote the fixed number of frames generated within one rendering window, and let $\mathcal{C} = \{C_t\}_{t=1}^{T}$ denote the corresponding target camera trajectory. We retain the original camera-conditioning pathway and provide it with $\mathcal{C}$. Because the projected-box controls are constructed under the same camera trajectory, the inherited camera module specifies the global viewpoint evolution, while the compiled controls specify the spatial support, persistent identity, semantic category, and camera-relative motion direction of the engine-maintained entities under those views.

To incorporate these controls, we attach a trainable Structured Spatial ControlNet to the pretrained backbone. For each frame, the state compiler constructs the structured projected-box control

$$M_t^{\text{ctrl}} = \text{Concat}\left(M_t^{\text{id}}, M_t^{\text{sem}}, M_t^{\text{dir}}\right), \tag{11}$$

as defined in Eq. (10). Applying this construction over the rendering window yields the control sequence $M_{1:T}^{\text{ctrl}} = \left\{M_t^{\text{ctrl}}\right\}_{t=1}^{T}$. The identity map provides persistent instance correspondence, the semantic map specifies the category of each projected entity, and the direction map explicitly represents the entity's own motion in the target camera coordinate system. In particular, because the direction state is derived from world-space object velocity rather than image-space displacement, it complements the camera-conditioning pathway by distinguishing object motion from apparent motion induced by the camera trajectory.

The control sequence is encoded at the spatial resolution of the video latent and processed by the ControlNet together with the noisy video latent. The ControlNet produces layer-wise control features $r^{(\ell)}$, which are projected to the backbone feature dimension and injected into the corresponding main blocks:

$$h_{\text{main}}^{(\ell)} \leftarrow h_{\text{main}}^{(\ell)} + r^{(\ell)}. \tag{12}$$

During training, the pretrained main branch, including its inherited camera-conditioning pathway, is frozen, while the newly introduced spatial-control branch is optimized. The two conditioning pathways play complementary roles: the camera module controls the global viewpoint trajectory, whereas the spatial-control branch constrains where each entity appears, which persistent instance and semantic category it represents, and how the entity itself moves under that trajectory.

The complete fixed-window rendering process is written as

$$I_{1:T} = G_{\theta,\phi}\left(I_0, \mathcal{C}, M_{1:T}^{\text{ctrl}}\right), \tag{13}$$

where $I_{1:T} = \{I_t\}_{t=1}^{T}$ denotes the generated T-frame video clip, θ denotes the frozen parameters of the pretrained backbone, and ϕ denotes the trainable parameters of the spatial-control branch. The initial observation I_0 provides the visual context for scene appearance, $\mathcal{C}$ specifies the target viewpoint trajectory, and $M_{1:T}^{\text{ctrl}}$ provides state-dependent entity guidance for the generated clip. The pretrained generative prior synthesizes visual details not explicitly represented by the projected-box controls, including object shape, texture, material, articulation, illumination, and secondary motion.

4.4.2 Chunk-Autoregressive Long-Horizon Rendering

To support long-horizon generation, we extend the renderer in a chunk-autoregressive manner. Denoising remains bidirectional within each chunk, while information is propagated causally across chunk boundaries.

We denote the generated frames, target camera trajectory, and compiled spatial controls of the n-th chunk by $I^{(n)}$, $\mathcal{C}^{(n)}$, and $M^{\text{ctrl},(n)}$, respectively, where each sequence spans L frames. The first chunk is initialized from the observed frame I_0 through the original image-to-video conditioning pathway, with zero-filled temporal history inputs and no spatial memory conditioning. For each subsequent chunk, the renderer additionally receives a temporal history $\mathcal{H}^{(n)}_{\text{temp}}$ and a geometry-aligned spatial memory $\mathcal{H}^{(n)}_{\text{spa}}$:

$$I^{(n)} = G_{\theta,\phi,\psi}\left(\mathcal{C}^{(n)}, M^{\text{ctrl},(n)}, \mathcal{H}^{(n)}_{\text{temp}}, \mathcal{H}^{(n)}_{\text{spa}}\right), \qquad n \geq 2, \tag{14}$$

where ψ denotes the trainable parameters introduced for cross-chunk visual conditioning. Both memory inputs are constructed exclusively from observations available before generating the current chunk.

Temporal history. For each chunk after the first, we maintain a bounded latent history from previously completed chunks. The history is organized into recent, mid-range, and long-range temporal context, with recent latents projected into tokens at finer spatiotemporal resolution and more distant latents at progressively coarser resolutions. This multi-scale representation preserves detailed recent information while extending the effective temporal context beyond a single chunk. In addition to the autoregressive history, we retain the latent representation of the initial observation I_0 as a persistent anchor.

During training, temporal-history latents are constructed from preceding source-video frames and stochastically degraded, including noise perturbation, feature corruption, and partial history dropping, so that the renderer learns to tolerate imperfect autoregressive context. During inference, each completed chunk contributes its generated latents to the temporal history, yielding a causal long-horizon rollout.

Geometry-aligned spatial memory. Temporal history is limited by a finite history window and therefore retains only local recent context during long autoregressive rollouts. As earlier observations fall outside this window, substantial camera motion, occlusion, or viewpoint revisitation can lead to missing or drifting visual content. To provide a more persistent global visual memory, we adopt the geometry-aligned spatial memory mechanism of AlayaWorld [17]. Completed RGB frames are lifted into a world-space memory using estimated depth and the corresponding camera parameters. Before generating a new chunk, relevant past observations are retrieved and reprojected into the target camera views, then encoded as spatial latent tokens for conditioning. The memory is updated only from completed past chunks, preserving causal generation across chunk boundaries.

4.5 Automatic Training Data Pipeline

Training our model requires structured annotations of camera geometry, persistent instance identities, semantic information, and object motion. Since such annotations are rarely available for in-the-wild videos, we develop an automatic data engine that recovers camera parameters, semantic labels, instance tracks, and object trajectories from unlabeled videos. These annotations are subsequently converted into camera-aligned conditioning maps, while the estimated camera trajectory is provided separately to the renderer.

Camera-parameter estimation. Given an input video $\mathcal{V}$, we use ViPE [9] to estimate the camera intrinsics, camera poses, and metric depth map for each frame. The depth and intrinsics allow image-space observations to be lifted into 3D, while the camera poses establish a shared world coordinate system across frames. Together, they support per-frame 3D object annotation and compensate for camera motion when recovering object trajectories.

Semantic-category discovery. An agent powered by Qwen3-VL [1] analyzes each video to generate a global caption describing its overall content and identify the discrete and countable object categories relevant to the scene. The resulting category set defines the semantic vocabulary used for subsequent instance segmentation. It can also be replaced with a predefined vocabulary when constructing a domain-specific dataset.

Instance segmentation and tracking. Given the discovered category set, SAM3 [4] performs video instance segmentation and tracking, producing temporally associated instance masks, 2D bounding boxes, semantic labels, and persistent track identities. Each instance inherits the semantic category used to prompt SAM3,

allowing objects of the same category to share a semantic label while retaining distinct track identities. Masks with insufficient visible area are discarded to suppress unreliable annotations caused by extremely small or heavily occluded objects.

Object-trajectory recovery. For each visible instance, we provide its tracked 2D bounding box to WildDet3D [10], together with the corresponding RGB frame, metric depth map, and camera intrinsics, to estimate a per-frame 3D oriented bounding box:

$$b_i^t = \left(\mathbf{p}_i^t, \mathbf{d}_i^t, \mathbf{R}_i^t, \ell_i, k_i\right), \tag{15}$$

where $\mathbf{p}_i^t$, $\mathbf{d}_i^t$, and $\mathbf{R}_i^t$ denote the box center, dimensions, and orientation, respectively. The semantic label ℓ_i specifies the object category, while k_i associates the box with its persistent instance track. Using the estimated camera poses, we transform the per-frame boxes into a shared world coordinate system. Boxes associated with the same track identity then form a temporally consistent object trajectory.

We estimate object motion from the displacement between consecutive world-space box centers along each trajectory. Computing the displacement in the shared world coordinate system removes apparent motion caused by camera movement. We then rotate the displacement into the current camera coordinate system and quantize it into seven states: static, left, right, up, down, forward, and backward. A small displacement threshold is used to identify static objects.

Conditioning-map generation. Finally, we project the estimated 3D OBBs into each frame using the corresponding camera parameters. The rasterization produces identity, semantic, and direction maps, with a z-buffer resolving visibility when multiple projected boxes overlap. Based on the persistent track identities, semantic labels, and quantized motion states, we further construct the identity, semantic, and object-motion maps used to condition our model.

5 Experiments

5.1 Experimental Setup

Training Data. We collect HUD-free gameplay videos from *Cyberpunk 2077*, *Forza Horizon 6*, and *Grand Theft Auto V*. The *Cyberpunk 2077* footage is captured from a first-person perspective, whereas *Forza Horizon 6* and *Grand Theft Auto V* are recorded from third-person perspectives using diverse camera viewpoints. The collected videos are subsequently processed using the Data Engine described in Sec. 4.5 to construct paired video-control training data.

Benchmark. We construct CombatStateBench, a controlled benchmark comprising 50 clips for evaluating whether generated videos faithfully reflect engine-maintained world states. For each scene, the Data Engine described in Sec. 4.5 reconstructs the initial 3D layout from the first frame, including the entities, their 3D boxes, semantic attributes, and camera parameters. Conditioned on this initial layout and the extracted game rules, an AI agent autonomously evolves a short combat scenario, producing a complete sequence of box-based world states. The scenarios cover diverse combinations of camera and entity motion, including interactions involving entities that initially lie outside the camera view. When a death event occurs, the target transitions to the corresponding state and remains in the scene thereafter.

Each benchmark sequence contains synchronized 3D boxes, entity states, camera parameters, projected entity controls, and instance masks. An automatic verifier checks initial-frame reprojection, metric depth consistency, box geometry, ground contact, temporal continuity, prescribed camera and entity motion, and the persistence of state transitions. Only sequences that satisfy all consistency checks are retained for evaluation.

5.2 Quantitative Results

Baselines. We compare our method with LingBot-World-V2 [5] and YUME [15], two representative interactive video world models. We evaluate both methods using the same initial observations and benchmark

Table 1 Video quality and world-state evaluation on 50 CombatStateBench clips. All values are reported as percentages. Count Accuracy is evaluated over 400 sampled frames (eight per clip), and State Accuracy over 50 death events, with three post-transition frames sampled per event.

Method	Imaging	Subject Cons.	Background Cons.	Temporal Stability	Count Acc.	State Acc.
LingBot-World-V2 [5]	67.46	81.87	91.89	96.85	40.75	8.00
YUME [15]	64.10	92.35	93.63	98.76	32.00	58.00
Ours	**67.62**	**94.74**	**96.98**	**99.00**	**94.00**	**98.00**

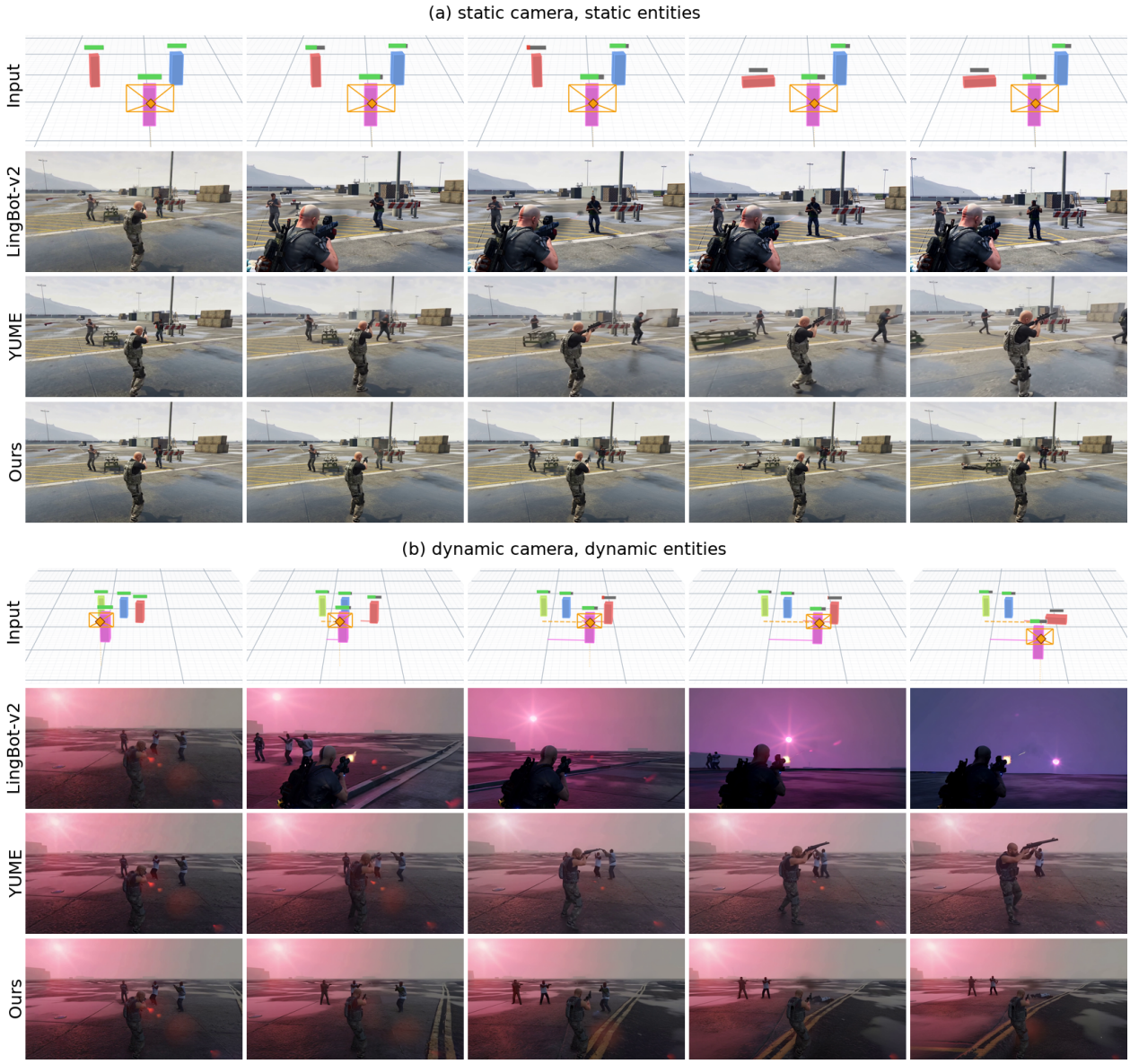

Figure 4 Qualitative comparison of entity-death interactions over five generated frames. We compare LingBot-World-v2 and YUME with our method under (a) a static camera and (b) a dynamic camera with moving entities. The input rows visualize the structured control signals provided to our renderer. Our method follows the specified death events while preserving the remaining entities and the scene evolution.

transitions while retaining their native conditioning pathways. Since neither method exposes an external instance-level state interface for explicitly maintaining entity identities, counts, and states, we communicate

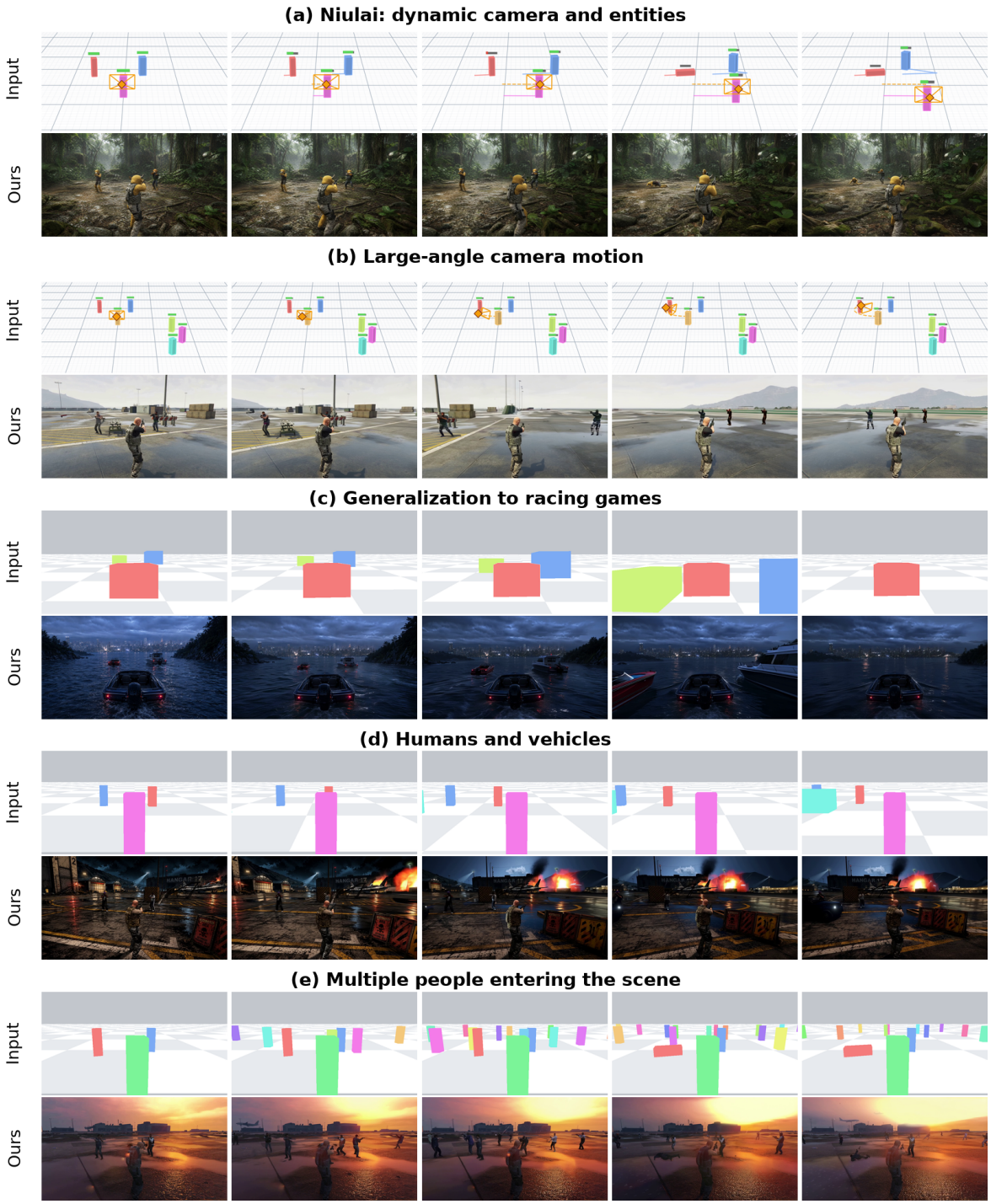

Figure 5 Additional qualitative results shown at five selected time steps. (a) Generalization to a novel minotaur scene with dynamic camera and entity motions. (b) Under a large-angle camera rotation, the model preserves scene consistency and correctly reveals characters located behind the initial camera view, following the persistent world state maintained by the engine. (c) Generalization to a racing game with moving cameras and vehicles. (d) Joint rendering of heterogeneous object categories, including humans and vehicles. (e) Selected frames from an 897-frame autoregressive sequence in which many NPCs progressively enter the scene. The input rows visualize the global scene state in (a)–(b) and camera-view 3D OBB maps with ground-grid references in (c)–(e). The generated results consistently follow the specified camera motion, spatial layout, and entity states.

state transitions through prompt switching. At each transition, the text condition is updated as

```
{environment description}.  {action description}.
```
(16)

For example, when a specified NPC is killed, the action description is updated to instruct the model to render the corresponding event. These baselines allow us to evaluate whether engine-maintained state and structured spatial controls provide more reliable interaction than the implicit state representations of existing video world models.

VLM-based Evaluation. We use `Qwen3.6-27B` as the VLM judge. The judge observes only RGB frames generated by each method. It is not provided with ground-truth bounding boxes, character identities, expected object counts, death locations, or other privileged spatial annotations.

Existing video world models generally lack explicit instance-level control. Consequently, evaluating all methods through object-level correspondence or bounding-box alignment would not provide a meaningful comparison. We therefore introduce two permissive, global metrics that measure whether the rendered video agrees with the state maintained by the engine: *Count Accuracy* and *State Accuracy*.

Count Accuracy. For each generated video, we randomly sample eight frames. For every sampled frame, the VLM counts the number of visibly alive characters in the rendered image. We compare this prediction with the number of visible alive characters maintained by the engine:

$$\text{CountAcc} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\left[\hat{n}_i = n_i^{\text{eng}}\right], \tag{17}$$

where N is the total number of sampled frames, and $\hat{n}_i$ and n_i^{eng} denote the VLM-predicted and engine-recorded entity counts, respectively.

State Accuracy. We additionally evaluate whether character death events are visually realized. For each engine-recorded death event, we uniformly sample three frames after the state transition and ask the VLM whether at least one frame depicts a dead character. We compute

$$\text{StateAcc} = \frac{\text{number of visually realized death events}}{\text{total number of death events}}. \tag{18}$$

This metric does not require the VLM to identify which character died or where the event occurred; it only checks whether the observable visual state is consistent with the engine state.

As shown in Table 1, our method substantially outperforms both LingBot-World-V2 and YUME in world-state consistency. For Count Accuracy, Ours achieves 94.00, exceeding LingBot-World-V2 by 53.25 percentage points and YUME by 62.00 percentage points. This result indicates that our engine-maintained representation more reliably preserves the intended number of visibly alive characters throughout the generated rollout.

For State Accuracy, Ours reaches 98.00, outperforming LingBot-World-V2 by 90.00 percentage points and YUME by 40.00 percentage points. Notably, both metrics evaluate only coarse, globally observable properties and do not directly reward the instance-level correspondence enabled by our structured control representation. Nevertheless, the consistent improvements over both baselines demonstrate that maintaining world state in an external engine and transmitting it through structured spatial controls provides substantially more reliable control than relying on the implicit state representation of a generative video model.

Video Quality. We evaluate perceptual and temporal quality using four VBench [24] metrics. Imaging Quality measures frame-level clarity, exposure, noise, and overall visual quality, while Subject Consistency evaluates whether the identity, appearance, and structure of foreground entities remain stable throughout the video. Background Consistency measures the cross-frame stability of the surrounding scene, and Temporal Stability is based on the VBench temporal-flickering score, where higher values indicate less flicker.

As shown in Table 1, Ours achieves the best performance across all four metrics. For Imaging Quality, Ours achieves 67.62, compared with 67.46 for LingBot-World-V2 and 64.10 for YUME. For Subject Consistency, Ours obtains 94.74, exceeding LingBot-World-V2 by 12.87 percentage points and YUME by 2.39 percentage

points, indicating stronger preservation of entity identity, appearance, and structure. For Background Consistency, Ours achieves 96.98, surpassing LingBot-World-V2 by 5.09 percentage points and YUME by 3.35 percentage points. Finally, for Temporal Stability, Ours reaches 99.00, compared with 96.85 for LingBot-World-V2 and 98.76 for YUME. Overall, these results demonstrate that our structured controls improve entity and background consistency while preserving strong frame-level quality and temporal stability.

5.3 Qualitative Results

Figure 4 compares our method with the prompt-switching baseline under the same initial observations and interaction sequences. Although the baseline can often generate a visually plausible response to an action prompt, it does not reliably preserve the resulting world state. For example, characters that should remain alive may disappear, killed characters may continue moving, and the number of visible characters may change without a corresponding engine event. In contrast, our method more faithfully reflects the engine-maintained state while retaining the visual realism of the underlying video generator.

Figure 5 presents qualitative results across diverse visual styles, camera motions, object categories, and interaction dynamics. For (b) and (e), we sample first frames from GTA V data that are unseen during training, whereas the first frames in (a), (c), and (d) are generated using GPT Image 2. In (a) and (b), the control signals are visualized in the global scene coordinate system, explicitly showing the engine-maintained entity states, spatial layout, and camera motion. In (c)–(e), we instead visualize the camera-view 3D OBB maps together with a ground-grid reference. Across both representations, the generated videos closely follow the specified camera motion and object configurations. Specifically, (a) depicts a minotaur scene with dynamic characters, demonstrating generalization to novel character appearances and environments. In (b), the camera undergoes a large-angle rotation, revealing three characters initially placed behind the first-frame view. The model preserves the scene structure throughout the camera motion and correctly renders these previously unobserved entities according to the persistent world state maintained by the engine. Example (c) extends the system to a racing scenario, showing that the proposed framework can support a different game genre rather than overfitting to a particular game or type of interaction. Example (d) demonstrates the joint rendering of heterogeneous object categories, including humans and vehicles, within the same scene. Finally, (e) shows an autoregressive sequence in which many NPCs progressively enter the scene. Despite the increasing number of simultaneously visible entities, the model maintains reasonable visual and temporal stability, suggesting that the system can support more complex and evolving worlds.

6 Conclusion

We introduced Programmable World Model to explore a simple premise: a generative world should not rely on visual generation alone to remember what is true about the world. By maintaining world state explicitly and using the video model primarily as a renderer, our framework makes persistent interactions more reliable without sacrificing generative flexibility. Our results show that this separation leads to substantially stronger state consistency and remains effective across long-horizon generation, unseen entities, visual styles, and interaction domains. We believe this points toward a broader direction for world models in which state is executable and verifiable, while appearance remains generative.

References

[1] S. Bai, Y. Cai, R. Chen, K. Chen, X. Chen, Z. Cheng, L. Deng, W. Ding, C. Gao, C. Ge, W. Ge, Z. Guo, Q. Huang, J. Huang, F. Huang, B. Hui, S. Jiang, Z. Li, M. Li, M. Li, K. Li, Z. Lin, J. Lin, X. Liu, J. Liu, C. Liu, Y. Liu, D. Liu, S. Liu, D. Lu, R. Luo, C. Lv, R. Men, L. Meng, X. Ren, X. Ren, S. Song, Y. Sun, J. Tang, J. Tu, J. Wan, P. Wang, P. Wang, Q. Wang, Y. Wang, T. Xie, Y. Xu, H. Xu, J. Xu, Z. Yang, M. Yang, J. Yang, A. Yang, B. Yu, F. Zhang, H. Zhang, X. Zhang, B. Zheng, H. Zhong, J. Zhou, F. Zhou, J. Zhou, Y. Zhu, and K. Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[2] P. J. Ball, J. Bauer, F. Belletti, B. Brownfield, A. Ephrat, S. Fruchter, A. Gupta, K. Holsheimer, A. Holynski, J. Hron, C. Kaplanis, M. Limont, M. McGill, Y. Oliveira, J. Parker-Holder, F. Perbet, G. Scully, J. Shar,

S. Spencer, O. Tov, R. Villegas, E. Wang, J. Yung, C. Baetu, J. Berbel, D. Bridson, J. Bruce, G. Buttimore, S. Chakera, B. Chandra, P. Collins, A. Cullum, B. Damoc, V. Dasagi, M. Gazeau, C. Gbadamosi, W. Han, E. Hirst, A. Kachra, L. Kerley, K. Kjems, E. Knoepfel, V. Koriakin, J. Lo, C. Lu, Z. Mehring, A. Moufarek, H. Nandwani, V. Oliveira, F. Pardo, J. Park, A. Pierson, B. Poole, H. Ran, T. Salimans, M. Sanchez, I. Saprykin, A. Shen, S. Sidhwani, D. Smith, J. Stanton, H. Tomlinson, D. Vijaykumar, L. Wang, P. Wingfield, N. Wong, K. Xu, C. Yew, N. Young, V. Zubov, D. Eck, D. Erhan, K. Kavukcuoglu, D. Hassabis, Z. Gharamani, R. Hadsell, A. van den Oord, I. Mosseri, A. Bolton, S. Singh, and T. Rocktäschel. Genie 3: A new frontier for world models. 2025.

[3] Z. Cai, S. Yang, Y. Wang, Z. Gao, Y. Liu, S. Weng, E. Wu, K. Zhang, and B. Shi. Mass: Multiplayer world models with authoritative shared state, 2026.

[4] N. Carion, L. Gustafson, Y.-T. Hu, S. Debnath, R. Hu, D. Suris, C. Ryali, K. V. Alwala, H. Khedr, A. Huang, J. Lei, T. Ma, B. Guo, A. Kalla, M. Marks, J. Greer, M. Wang, P. Sun, R. Rädle, T. Afouras, E. Mavroudi, K. Xu, T.-H. Wu, Y. Zhou, L. Momeni, R. Hazra, S. Ding, S. Vaze, F. Porcher, F. Li, S. Li, A. Kamath, H. K. Cheng, P. Dollár, N. Ravi, K. Saenko, P. Zhang, and C. Feichtenhofer. Sam 3: Segment anything with concepts, 2025.

[5] Z. Gao, Q. Wang, J. Zhu, J. Chen, Z. Liu, Q. Bai, J. Wang, Y. Yuan, H. Wang, Y. Lu, et al. Infinite worlds with versatile interactions. *arXiv preprint arXiv:2607.07534*, 2026.

[6] G. Gomez-Nogales, Y. Hong, C. Ge, P. Zhuang, M. Comino-Trinidad, D. Casas, and Y. Zhou. Coarse-to-real: Generative rendering for populated dynamic scenes. *arXiv preprint arXiv:2601.22301*, 2026.

[7] Y. HaCohen, N. Chiprut, B. Brazowski, D. Shalem, D. Moshe, E. Richardson, E. Levin, G. Shiran, N. Zabari, O. Gordon, P. Panet, S. Weissbuch, V. Kulikov, Y. Bitterman, Z. Melumian, and O. Bibi. Ltx-video: Realtime video latent diffusion. *arXiv preprint arXiv: 2501.00103*, 2024.

[8] Y. Hong, Y. Mei, C. Ge, Y. Xu, Y. Zhou, S. Bi, Y. Hold-Geoffroy, M. Roberts, M. Fisher, E. Shechtman, K. Sunkavalli, F. Liu, Z. Li, and H. Tan. Relic: Interactive video world model with long-horizon memory, 2025.

[9] J. Huang, Q. Zhou, H. Rabeti, A. Korovko, H. Ling, X. Ren, T. Shen, J. Gao, D. Slepichev, C.-H. Lin, et al. Vipe: Video pose engine for 3d geometric perception. *arXiv preprint arXiv:2508.10934*, 2025.

[10] W. Huang, J. Zhang, S. Li, J. Duan, Y. Cheng, J. Cho, M. Wallingford, R. Soraki, C. D. Kim, S. Liu, et al. Wilddet3d: Scaling promptable 3d detection in the wild. 2026.

[11] Z.-H. Huang, Z. Wang, J. Tan, R. Yu, Y. Zhang, B. Zheng, Y.-L. Liu, Y.-Y. Chuang, and K. Zhang. Generative world renderer. *arXiv preprint arXiv:2604.02329*, 2026.

[12] R. Liang, Z. Gojcic, H. Ling, J. Munkberg, J. Hasselgren, C.-H. Lin, J. Gao, A. Keller, N. Vijaykumar, S. Fidler, et al. Diffusionrenderer: Neural inverse and forward rendering with video diffusion models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26069–26080. IEEE, 2025.

[13] G. Lin, Z.-H. Huang, S. Yang, M.-H. Yang, K. Zhang, and Z. Wang. Generative world renderer at the speed of play, 2026.

[14] Z. Lin, Z. Wang, C. Tan, B. Wen, and Y. Jin. Stateplay: State-aware game world models for mechanics-consistent generation, 2026.

[15] X. Mao, S. Lin, Z. Li, C. Li, W. Peng, T. He, J. Pang, M. Chi, Y. Qiao, and K. Zhang. Yume: An interactive world generation model. *arXiv preprint arXiv:2507.17744*, 2025.

[16] W. Sun, H. Zhang, H. Wang, J. Wu, Z. Wang, Z. Wang, Y. Wang, J. Zhang, T. Wang, and C. Guo. Worldplay: Towards long-term geometric consistency for real-time interactive world modeling. *arXiv preprint arXiv:2512.14614*, 2025.

[17] A. Team, K. Zhang, C. Li, Y. Zhan, Y. Ge, Y. Yin, J. Tan, K. He, L. Fan, M. Zhai, et al. Alayaworld: Interactive long-horizon world modeling–full technical report. *arXiv preprint arXiv:2607.18367*, 2026.

[18] R. Team, Z. Gao, Q. Wang, Y. Zeng, J. Zhu, K. L. Cheng, Y. Li, H. Wang, Y. Xu, S. Ma, et al. Advancing open-source world models. *arXiv preprint arXiv:2601.20540*, 2026.

[19] T. Wan, A. Wang, B. Ai, B. Wen, C. Mao, C.-W. Xie, D. Chen, F. Yu, H. Zhao, J. Yang, J. Zeng, J. Wang, J. Zhang, J. Zhou, J. Wang, J. Chen, K. Zhu, K. Zhao, K. Yan, L. Huang, M. Feng, N. Zhang, P. Li, P. Wu, R. Chu, R. Feng, S. Zhang, S. Sun, T. Fang, T. Wang, T. Gui, T. Weng, T. Shen, W. Lin, W. Wang, W. Wang, W. Zhou, W. Wang, W. Shen, W. Yu, X. Shi, X. Huang, X. Xu, Y. Kou, Y. Lv, Y. Li, Y. Liu, Y. Wang, Y. Zhang, Y. Huang, Y. Li, Y. Wu, Y. Liu, Y. Pan, Y. Zheng, Y. Hong, Y. Shi, Y. Feng, Z. Jiang, Z. Han, Z.-F. Wu, and Z. Liu. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv: 2503.20314*, 2025.

[20] Z. Wang, Z. Liu, J. Li, K. Huang, B. Xu, F. Kang, M. An, P. Wang, B. Jiang, Y. Wei, Y. Xietian, J. Pei, L. Hu, B. Jiang, H. Xue, Z. Wang, H. Sun, W. Li, W. Ouyang, X. He, Y. Liu, Y. Li, and Y. Zhou. Matrix-game 3.0: Real-time and streaming interactive world model with long-horizon memory, 2026.

[21] X. Yang, B. Li, Y. Zhang, Z. Yin, L. Bai, L. Ma, Z. Wang, J. Cai, T.-T. Wong, H. Lu, et al. Vlipp: Towards physically plausible video generation with vision and language informed physical prior. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12360–12370. IEEE, 2025.

[22] J. Yi, M. Kim, P. H. Cho, W. Jang, S. Yun, and S. Kim. Worldkv: Efficient world memory with world retrieval and compression, 2026.

[23] C. Zeng, Y. Dong, P. Peers, H. Wu, and X. Tong. Renderformer: Transformer-based neural rendering of triangle meshes with global illumination. In *ACM SIGGRAPH 2025 Conference Papers*, 2025.

[24] D. Zheng, Z. Huang, H. Liu, K. Zou, Y. He, F. Zhang, Y. Zhang, J. He, W.-S. Zheng, Y. Qiao, and Z. Liu. VBench-2.0: Advancing video generation benchmark suite for intrinsic faithfulness. *arXiv preprint arXiv:2503.21755*, 2025.