A Appendix A: Task Decomposition and Physical Property Inference

We present detailed prompt templates for our two-stage vision-language analysis process. These prompts are designed to enable structured scene understanding and precise physical parameter estimation.

A.1 Task Decomposition

The first stage employs a comprehensive prompt template that guides the vision-language model to decompose user instructions into atomic operations while analyzing potential physical interactions.

```
Analyze the image and the instruction "{user instruction}" to provide a structured
    scene understanding that focuses on potential physical interactions. Return the
    analysis in the exact JSON format specified below:
1. "decomposed_tasks": An array of task objects, where each task represents an atomic
     operation:
   - "task_type": Must be one of ["insert", "remove", "control"]
   - "task_prompt": A clear, specific instruction for this sub-task
   - "task_object": The primary object involved in this task
2. "main_objects": Array of strings naming the key objects that will be directly
    manipulated in the animation
3. "other_objects": Array of strings naming objects that could be physically affected
     by the movement or manipulation of the main objects (consider collisions,
    interactions, or displacement)
4. "ground_names": Array of strings identifying all potential surfaces where
    main_objects and other_objects might move to or interact with during the
    animation
5. "edge_names": Array of strings identifying boundaries that main_objects and
    other_objects might reach during movement. These can include:
   - Physical scene boundaries
   - Edges of other objects in the scene
   - Surfaces of stationary objects that could act as boundaries

Required JSON format:
{
    "decomposed_tasks": [
        {"task_type": null, "task_prompt": "", "task_object": ""},
        {"task_type": null, "task_prompt": "", "task_object": ""}
    ],
    "main_objects": ["", ""],
    "other_objects": ["", ""],
    "ground_names": [],
    "edge_names": []
}

When analyzing potential interactions and boundaries, consider:
- The complete trajectory of all possible movements
- Secondary effects and chain reactions
- All possible surfaces objects might move across
- Any objects that could become boundaries or obstacles
```

A.2 Physical Property Inference

Building upon the task decomposition results, the second stage utilizes a specialized prompt template to infer physical properties and motion parameters.

```
You will be given an image and the following scene elements from previous analysis:

decomposed_tasks: {decomposed_tasks}
```

```
main_objects: {main_objects}
other_objects: {other_objects}
ground_names: {ground_names}

Consider the user instruction: "{user_instruction}" and analyze the physical
    properties of each element, infer initial velocity for control tasks, and
    determine appropriate gravity value.

Format Requirement:
You must provide your answer in the following JSON format:

{
    "object_properties": {
        // For each object in main_objects and other_objects
        "<object_name>": {
            "mass": number, // in grams
            "friction": number, // 0.0 (frictionless) to 1.0 (high friction)
            "elasticity": number // 0.0 (no bounce) to 1.0 (perfect bounce)
        }
    },
    "ground_properties": {
        // For each surface in ground_names
        "<ground_name>": {
            "friction": number,
            "elasticity": number
        }
    },
    "init_velocity": {
        // For each object involved in control tasks
        "<object_name>": {
            "value": [number, number],
        }
    },
    "gravity": {
        "value": number,
    }
}

Property Specifications:
- mass: Measured in grams, considering object size and material
- friction: Coulomb friction model (0.0 = frictionless, 1.0 = maximum friction)
- elasticity: Bounce coefficient (0.0 = no bounce, 1.0 = perfect elastic bounce)
- force: Based on action words in control tasks (e.g., "push", "pull", "lift")
- gravity: Adjust from default 980 cm/s^2 based on depth perspective and motion
```

B Appendix B: Vision Inference Implementation

Our vision inference pipeline integrates four key components designed to extract comprehensive scene information from single images: instance segmentation, depth estimation, illumination analysis, and scene reconstruction. Each component is specifically optimized for physics-aware animation tasks.

Instance Segmentation: We leverage Grounding-SAM [**?**] for precise instance-level segmentation, utilizing object, ground, and edge names from previous task decomposition as grounding inputs. This approach ensures accurate mask generation aligned with physical interaction requirements. The implementation utilizes the official codebase[1] with default parameters for optimal performance.

[1] https://github.com/IDEA-Research/Grounded-Segment-Anything

Depth and Normal Estimation: Depth and normal maps are estimated using Geowizard [**?**], providing crucial geometric information for our depth-aware physical simulation. The model generates high-fidelity depth maps that enable accurate spatial relationship modeling. Implementation details and parameters follow the original repository[2].

Illumination Decomposition: For lighting analysis, we employ an intrinsic decomposition model [**?**] to extract detailed shading and albedo maps. This decomposition enables physically accurate lighting adaptation during animation. The implementation follows the official repository[3] specifications.

Scene Reconstruction: Our scene reconstruction framework employs task-specific approaches: Clipaway [**?**] for object removal tasks[4], and Diffree [**?**][5] for object insertion operations. This dual approach ensures high-quality scene composition while maintaining physical plausibility.

C Appendix C: Collision Handling

Our collision handling framework maintains physical plausibility through three key mechanisms operating in concert across multiple depth layers.

Depth-aware Resolution: Our system discretizes the depth map into layers of 10-pixel intervals, allowing objects to simultaneously exist across multiple depth layers based on their spatial extent in the scene. During object motion, depth layer transitions trigger scale adjustments with a factor $\lambda = 0.99$ per layer, ensuring perspective-consistent size changes while maintaining continuous depth representation.

Collision Detection: The system performs collision detection by examining boundary overlaps between objects, grounds, and edges, validating collisions only when the involved entities share overlapping depth layers. This depth-aware approach ensures that interactions occur exclusively between elements existing within the same depth-based spatial context.

Response Calculation: Upon collision detection, the system computes response trajectories following conservation laws within shared depth layers, treating motion like reflection along the depth layer plane. This approach applies uniformly to object-object, object-ground, and object-edge collisions, with response directions determined by the collision surface normal within the respective depth layer.

The complete collision handling process can be formally described as:

Algorithm 1 Depth-aware Collision Handling

Input: Objects $\mathcal{O}$, Depth Map D, Time step Δt
Output: Updated object states
for each time step t **do**
  for each object $o_i \in \mathcal{O}$ **do**
    Assign depth layers $L_i = \{l | d_{min} \leq D(o_i) \leq d_{max}\}$
    Scale $s_i = \lambda^{\Delta l}$ where Δl is layer transitions
  end for
  for each object pair (o_i, o_j) and scene elements **do**
    if boundaries overlap **and** $L_i \cap L_j \neq \emptyset$ **then**
      Compute collision normal $\vec{n}$ in shared layer
      Update velocities: $\vec{v}_{new} = \vec{v}_{old} - 2(\vec{v}_{old} \cdot \vec{n})\vec{n}$
    end if
  end for
  Update object positions and states
end for

The proposed collision handling algorithm processes interactions in a depth-aware manner. For each object, depth layers are assigned based on depth map values, with a scaling factor $\lambda = 0.99$ applied during layer transitions. Collision detection validates both boundary overlaps and depth layer intersections ($L_i \cap L_j \neq \emptyset$). The velocity update follows a reflection-based model: $\vec{v}_{new} = \vec{v}_{old} - 2(\vec{v}_{old} \cdot \vec{n})\vec{n}$ where $\vec{n}$ is the collision normal in the shared depth layer. This formulation ensures momentum conservation while maintaining physical plausibility within each depth layer. The

[2]https://github.com/fuxiao0719/GeoWizard
[3]https://github.com/compphoto/Intrinsic
[4]https://github.com/YigitEkin/CLIPAway
[5]https://github.com/OpenGVLab/Diffree

same principle applies to object-object, object-ground, and object-edge collisions, with collision normals determined by the respective interaction surfaces. The complete algorithm integrates depth-layer discretization, collision validation, and response calculation into a unified framework, enabling efficient handling of complex physical interactions while preserving depth relationships.

D Appendix D: Physics-Constrained Refinement

Following PhysGen [**?**], we implement a physics-constrained refinement process. Given a relit video $\tilde{\mathbf{V}}$, we first encode it to obtain the latent representation $\tilde{\mathbf{z}}_0$. Our objective is to derive the denoised latent code $\mathbf{z}_0$ utilizing the pretrained latent diffusion model p_θ.

E Appendix E: Ablation Study

We conduct ablation studies to evaluate key components through quantitative metrics and parameter sensitivity analysis. The following table summarizes component-wise contributions:

Table 1: Ablation study results evaluating key components of our framework.

Method	CLIP-Sim (↑)	FID (↓)	Motion-FID (↓)
Full Model	16.64	102.32	389.64
w/o Depth-Aware Physics Simulation	16.21	112.15	398.87
w/o Depth-Aware Relighting	16.43	108.43	395.91
w/o Physics-Constrained Refinement	16.55	106.76	394.21

We conduct ablation experiments to analyze the contribution of each component in our framework. As shown in the table, removing depth-aware physics simulation leads to a 9.6% degradation in FID (102.32 $\rightarrow$ 112.15) and 2.6% drop in CLIP-Sim score (16.64 $\rightarrow$ 16.21), indicating its importance for visual quality. The absence of depth-aware relighting results in a significant impact on motion coherence (Motion-FID increases by 1.6%), while physics-constrained refinement primarily affects visual quality (FID increases by 4.3%). Notably, all variants maintain relatively stable CLIP-Sim scores, suggesting robust semantic consistency across different configurations. These results validate the effectiveness of our full model's component integration.

F Appendix F: Runtime Analysis

We analyze the computational efficiency of each component in our framework. All experiments were conducted on a single NVIDIA A6000 GPU. Tab 2 shows the average runtime for each stage when processing a single 512×512 image to run 160 simulation steps and sampling 16 frames.

Table 2: Runtime analysis of PhysLayer components

Component	Time (seconds)
Scene Understanding	10
Vision Inference (Control task)	21
Vision Inference (Insertion task)	25
Vision Inference (Removal task)	26
Physics Simulation	5
Relighting	4
Video Synthesis	60
Total	$100 \sim 105$

G Appendix G: Layer Count Sensitivity Analysis

We evaluate the effect of depth layer count L on 30 videos. With L=2, coarse discretization degrades FID (111.87) and M-FID (400.53). Performance improves at L=4 (FID 103.45, M-FID 391.12) and peaks near L=6 (FID 102.78, M-FID 390.21), with marginal regression at L=8 (FID 103.14, M-FID 391.48) due to over-fragmentation of depth

clusters. Our adaptive formula $L = \min(N_{obj}/2, 8)$ achieves the best overall results by matching layer count to scene complexity.