# AutoVFX: Physically Realistic Video Editing from Natural Language Instructions

Hao-Yu Hsu Zhi-Hao Lin Albert J. Zhai Hongchi Xia Shenlong Wang
University of Illinois at Urbana-Champaign
https://haoyuhsu.github.io/autovfx-website/

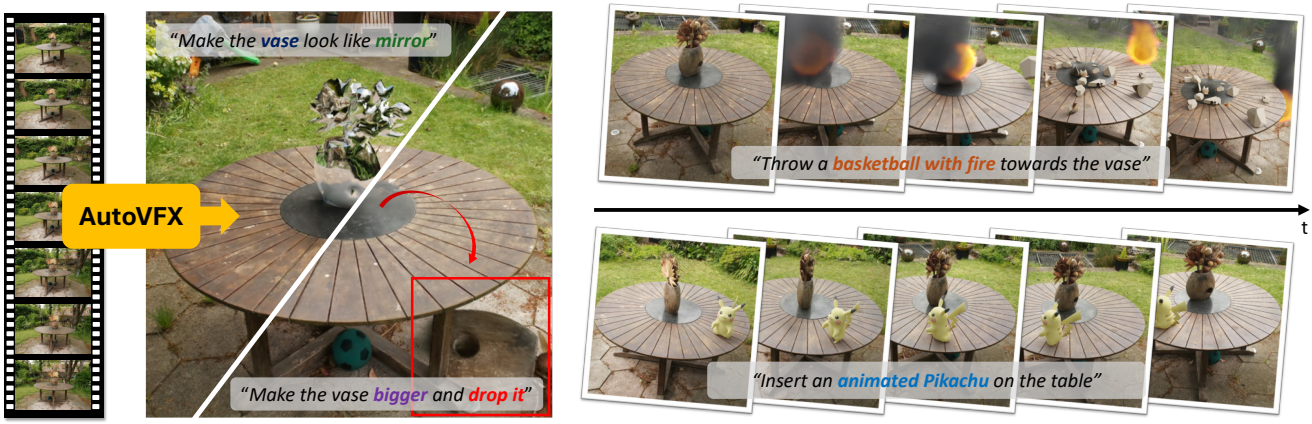


Figure 1. **AutoVFX** takes a video and language instructions as input, and automatically generates programs to produce visual effects and render a new video according to the instructions. It can modify appearance and geometry, enable dynamic interactions, apply particle effects, and even insert animated characters, producing results that are photorealistic, physically-plausible, and easily controllable.

## Abstract

*Modern visual effects (VFX) software has made it possible for skilled artists to create imagery of virtually anything. However, the creation process remains laborious, complex, and largely inaccessible to everyday users. In this work, we present AutoVFX, a framework that automatically creates realistic and dynamic VFX videos from a single video and natural language instructions. By carefully integrating neural scene modeling, LLM-based code generation, and physical simulation, AutoVFX is able to provide physically-grounded, photorealistic editing effects that can be controlled directly using natural language instructions. We conduct extensive experiments to validate AutoVFX's efficacy across a diverse spectrum of videos and instructions. Quantitative and qualitative results suggest that AutoVFX outperforms all competing methods by a large margin in generative quality, instruction alignment, editing versatility, and physical plausibility.*

## 1. Introduction

Visual effects (VFX) combine realistic video footage with computer-generated imagery to create novel, photorealistic visuals. Recent advances in graphics, vision, and physical simulation have made it possible to produce VFX that depict virtually anything—even those that are too costly, time-consuming, dangerous, or impossible to capture in real life. As a result, VFX have become essential in modern filmmaking, ads, simulation, AR/VR, *etc*. However, the process remains laborious, complex, and expensive, requiring expert skills and professional software [3, 17, 18, 44], making it largely inaccessible to everyday users.

A promising approach to democratizing VFX is to treat it as a generative video editing problem, where raw video and language prompts are used to generate new videos reflecting the original content and given instructions [4, 10, 19, 32, 50, 64, 65, 69, 92, 94, 103]. This method leverages advances in generative modeling, learning from large-scale internet data to produce controllable video. Successes have been seen in deepfake videos, fashion, driving, and robotics [15, 31, 52, 87, 102]. However, this purely data-driven generative editing approach hasn't yet replaced traditional VFX pipelines due to challenges in achieving guaranteed physical plausibility, precise 3D-aware control, and various special effects.

Another appealing alternative is to build a 3D representation from video input, apply edits like object insertion or texture changes, and then render the final output [12, 13, 22, 25, 26, 36, 53, 55–57, 63, 71, 83, 100, 113, 114]. While this approach aligns well with the VFX pipeline, it is often limited in editing capabilities and still requires manual inter-

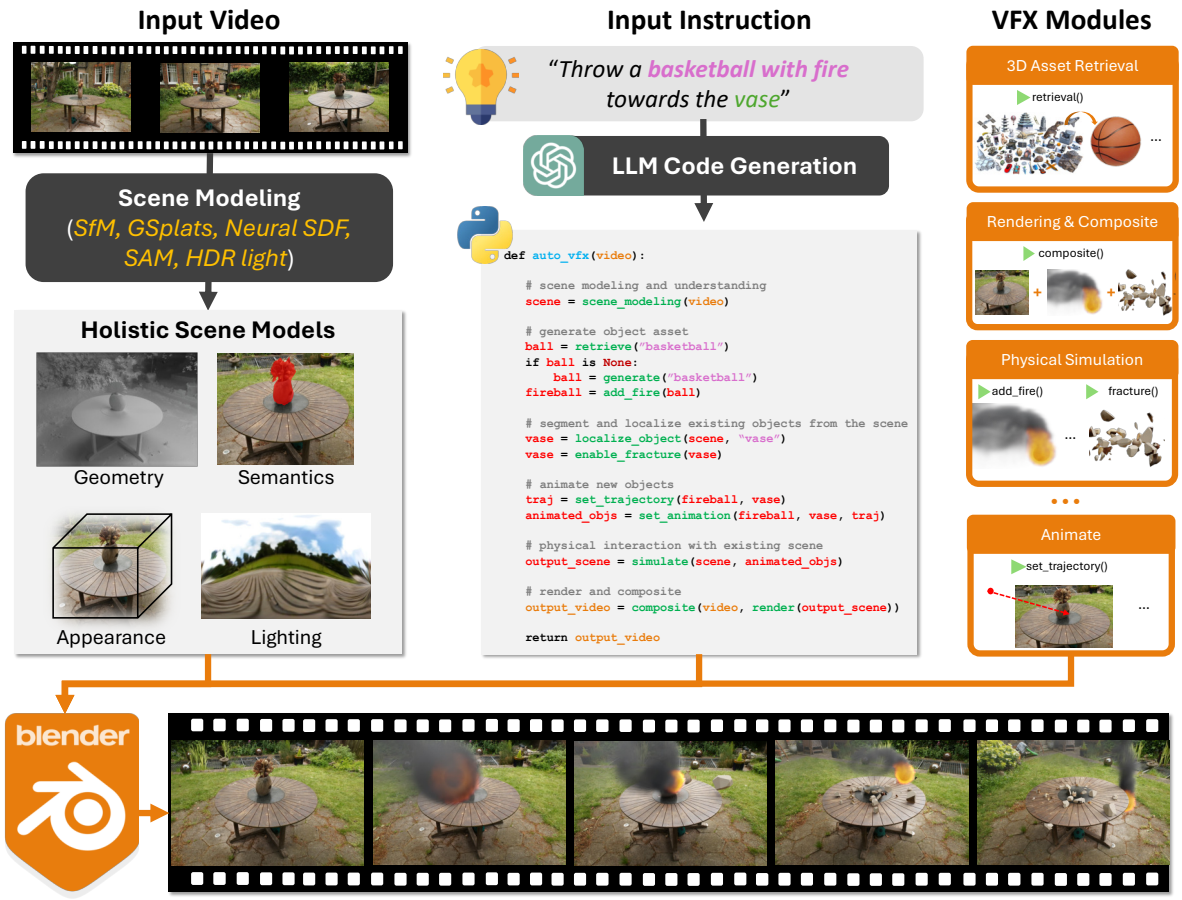


Figure 2. **AutoVFX framework.** Our instruction-guided video editing framework consists of three main modules: (1) **3D Scene Modeling** (left), which integrates 3D reconstruction and scene understanding models; (2) **Program Generation** (middle), where LLMs generate editing programs based on user instructions; and (3) **VFX Modules** (right), which include predefined functions specialized for various editing tasks. These components are integrated with a physically-based simulation and rendering engine (e.g., Blender) to generate the final video.

action with cumbersome interfaces, making it difficult for everyday users. Bridging this gap is essential to make 3D scene editing capable of handling most visual effects while remaining accessible to everyone.

In this work, we present AutoVFX, a framework that automatically creates realistic and dynamic VFX videos from a single video and natural language instructions. At the core of our method is a novel integration of neural scene modeling, LLM-based code generation, and physical simulation. First, we establish a holistic scene model that encodes rich geometry, appearance, and semantics from the input video. This model serves as the foundation for a variety of scene editing, simulation, and rendering capabilities, which we organize into a collection of executable functions. Next, AutoVFX takes simple language editing instructions and converts them into programs using large language models (LLMs). These programs consist of a sequence of calls to our predefined functions. Finally, the generated code is executed, producing a free-viewpoint video that reflects the instructed changes. Fig. 2 illustrates the overall framework.

AutoVFX combines the strengths of generative editing and physical simulation, yet is uniquely set apart from both. Like traditional VFX, AutoVFX produces videos with physics-grounded, controllable, and photorealistic effects. At the same time, similar to generative editing, we support open-world natural language instructions, allowing anyone to edit a video by simply describing the desired effects.

We conduct extensive experiments to validate AutoVFX's efficacy across a diverse spectrum of videos and instructions. We also perform user studies and qualitative and quantitative comparisons with existing video and scene editing methods. Experimental results suggest AutoVFX outperforms all competing methods by a large margin in generative quality, instruction alignment, editing versatility, and physical plausibility. This demonstrates the effectiveness and convenience of our approach, highlighting its potential as a valuable framework to democratize VFX and pave the way for future integration of even more capabilities to further enhance realism in automatic VFX.

Table 1. **Comparison of existing and proposed methods for visual editing.** Generative editing models lack physical plausibility and precise controllability. Existing physics-based editing methods have complicated interfaces and are limited in their range of editing capacities. Our method, AutoVFX, enjoys a convenient natural language interface while providing the widest range of capabilities.

| Method | Input & Output | | | | Editing Capacities | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Real World Video Editing | Free-Viewpoint Rendering | Editing Interface | Open-world Query | Object Insertion | Object Removal | Object Rearrange | Appearance Change | Animated Objects | Physics Simulation | Particle Effects |
| Visual Programming [35] | ✓ | × | Natural Language | ✓ | ✓ | ✓ | ✓ | ✓ | × | × | × |
| FRESCO [103] | ✓ | × | Natural Language | ✓ | × | × | × | ✓ | × | × | × |
| ClimateNeRF [53] | ✓ | ✓ | Predefined Scripts | × | × | × | × | ✓ | × | × | ✓ |
| Feature Splatting [73] | ✓ | ✓ | Predefined Scripts | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × |
| GaussianEditor [13] | ✓ | ✓ | Graphical | ✓ | ✓ | ✓ | × | ✓ | × | × | × |
| Gaussian Grouping [105] | ✓ | ✓ | Graphical | ✓ | ✓ | ✓ | ✓ | ✓ | × | × | × |
| PhysGaussian [98] | ✓ | ✓ | Graphical | × | × | × | × | × | × | ✓ | × |
| VR-GS [47] | ✓ | ✓ | Graphical | × | ✓ | ✓ | ✓ | × | × | ✓ | × |
| Gaussian Splashing [28] | ✓ | ✓ | Graphical | × | ✓ | ✓ | ✓ | × | × | ✓ | ✓ |
| DMRF [71] | ✓ | ✓ | Graphical | × | ✓ | × | × | × | ✓ | ✓ | ✓ |
| Instruct-N2N [36] | ✓ | ✓ | Natural Language | ✓ | × | × | × | ✓ | × | × | × |
| DGE [12] | ✓ | ✓ | Natural Language | ✓ | × | × | × | ✓ | × | × | × |
| Chat-Edit-3D [26] | ✓ | ✓ | Natural Language | ✓ | ✓ | ✓ | ✓ | ✓ | × | × | × |
| ChatSim [90] | ✓ | ✓ | Natural Language | × | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |
| **AutoVFX(Ours)** | ✓ | ✓ | Natural Language | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

## 2. Related Work

Our framework is closely related to several areas, including physical simulation on NeRFs, instruction-guided visual editing, and LLMs for code generation, integrating aspects of all three. Next, we will discuss these areas and highlight and contrast notable works in Tab. 1.

**Physical Simulation on NeRFs and 3D Gaussians** Integrating physics simulation into NeRFs and 3D Gaussians enables immersive and convincing dynamic effects within captured scenes. Several lines of work have explored various physical interactions, including rigid body object interaction [90, 96], particle physics effects such as flooding and fog [28, 53], elastic deformable objects [111], and plastic objects [47, 70, 98]. The key idea is to enable captured scenes to faithfully interact with new events or entities through physical simulation. However, this is challenging for vanilla neural implicit models, as conventional simulation often requires high-fidelity surface geometry, which is not explicit in these models. Therefore, various approaches seek to extract meshes from NeRFs [16, 47, 66, 70, 99, 108] to facilitate simulation, while others adapt implicit or particle-based simulation so that it can be directly applied to implicit models or Gaussians [27, 28, 51, 98]. AutoVFX explores a hybrid representation where meshes are used for physical interaction and Gaussians are stacked on the mesh surface for rendering, combining the best of both worlds. Another challenge is that some physical interactions require an understanding of physical properties. Various approaches address this through inverse physics [51, 111], common sense knowledge in large foundation models [29, 59, 109], or generative models [111]. Most works on physical simulation, however, are driven by domain-specific scripts rather than natural language instructions, often restricting them to specific physical effects and limiting their user base. AutoVFX seeks to bridge this gap by using LLMs to convert language instructions into simulation programs and supporting numerous dynamical effects through off-the-shelf simulators.

**Instruction-based Visual Editing** Recent advancements in visual-language models have made visual editing more accessible by allowing users to edit a wide range of content, such as images, videos, and 3D scenes, using language instructions [8, 29, 36, 50, 103] instead of relying on GUI interactions or script programs [13, 47, 53, 73, 105]. Text- and image-conditioned generative models, particularly diffusion-based approaches [37, 79], have been explored for text-guided image [6, 8, 62, 79, 110] and video [4, 10, 19, 32, 50, 64, 65, 69, 92, 94, 103] editing. Language-embedded NeRFs extend this generative editing capability to 3D scenes [12, 13, 22, 25, 36, 63, 85, 100, 113, 114]. However, mapping text instructions to desired edits in videos and scenes purely through diffusion models can be challenging, particularly when tasks involve complex steps, dynamic interactions, or physics, which can affect instruction alignment, physical plausibility, or realism. To address this, some methods use large language models (LLMs) to break down tasks into subtasks [25, 26, 90] or generate executable programs based on instructions [35, 60]. AutoVFX belongs to this latter category but demonstrates significantly richer capabilities, such as dynamic visual effects, animated objects and physical interaction, compared to these methods.

**LLMs for Code Generation** The powerful capabilities of large language models (LLMs) have revolutionized code generation based on natural language descriptions. By providing in-context examples, LLMs can generate code snippets in specific formats or syntaxes. Studies such as [2, 11, 23] have explored the effectiveness of LLMs in solving math and code problems. LLM-based code generation has recently been investigated in vision and robotics. Many works adopt LLMs

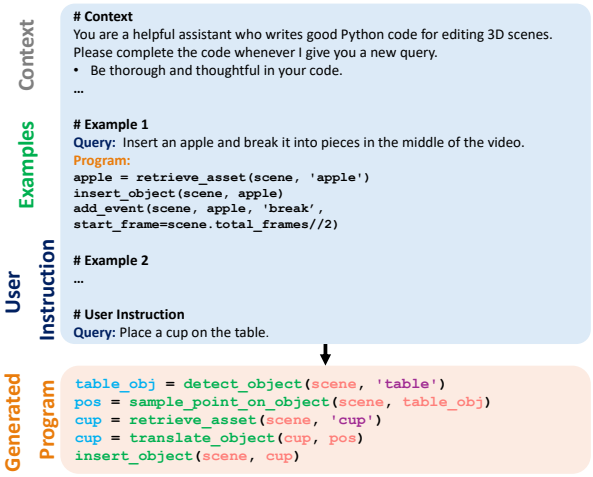


Figure 3. **Program generation.** The LLM generates the editing program through in-context learning. With provided context and examples, it learns to call VFX modules and, given unseen user instructions (blue block), generates the program (orange block).

for decision-making in embodied AI, including tasks like manipulation [42, 54, 84, 101] and navigation [61]. Recently, LLM-based programs for visual content creation have gained attention, with notable progress in 2D understanding and editing [35], driving simulation [26], video generation [60], and procedural 3D scene generation [30, 41, 75, 76, 112]. In our method, we harness GPT-4 to interpret natural language descriptions into executable programs for creating diverse visual effects for generic real-world videos.

## 3. Text-Driven VFX Creator

AutoVFX takes as input a video and a natural language editing prompt, and outputs an edited free-viewpoint video. The core idea is to uniquely combine the code generation capabilities of LLMs with 3D scene modeling and physics-based simulation techniques. Fig. 2 depicts the overall framework. First, we harness various 3D vision methods to estimate key scene properties from the input video (Sec. 3.1). This lays the foundation for a variety of scene editing, simulation, and rendering capabilities, which we organize into a collection of executable modules (Sec. 3.2, Sec. 3.3). An LLM is used to convert the natural language editing instructions into a program calling these functions (Sec. 3.4). Finally, the generated program is executed, producing a free-viewpoint video that reflects the instructed changes.

### 3.1. 3D Scene Modeling

Photo-realistic, physics-based VFX creation requires modeling several key properties of the captured scene, namely geometry, appearance, semantics, and lighting. We employ a variety of recent scene understanding models to estimate these properties, including separate models for geometry and appearance in order to achieve both high simulation fidelity and photorealistic rendering.

**Geometry** Modeling the 3D geometry of the scene is essential for any sort of object insertion, removal, or simulation. We first run COLMAP [81] to infer the camera poses of each frame. We then capture the geometry in the form of a triangle mesh produced by BakedSDF [104], a multi-view reconstruction method that optimizes a hybrid implicit neural representation that encodes the signed distance field of the scene, and then bakes the representation onto a triangulated mesh. It achieves a desirable balance between surface accuracy, completeness, and efficiency. We choose to use a mesh representation because it can be directly loaded into a standard VFX pipeline, rendered efficiently, and further enables accurate physical simulation. Moreover, it serves as the geometry proxy for object instance extraction.

**Appearance** We capture the appearance of the scene in two ways. First, we use SuGaR [33], a Gaussian Splatting [48] based novel view synthesis method, to enable free-view rendering. Although SuGaR can provide realistics renderings, it cannot be directly incorporated into physical-based rendering, making it unsuitable for modeling reflective effects on inserted objects or material editing. Thus, we also represent the scene by texturing the BakedSDF mesh, which has lower visual fidelity but can be integrated with physically-based rendering. This textured mesh is used for shadow mapping and encoding multi-bounce effects.

**Semantics** In many cases, a user would like to perform an edit localized to a specific semantic region of the scene, for example "make the car on fire". To enable such edits, we use Grounding SAM [58] to perform open-vocabulary instance segmentation and DEVA [14] to associate the instances across frames. To lift video segmentation to 3D, we first un-project each pixel from the 2D segmentation mask into the 3D scene geometry. A voting mechanism is used to determine the visibility of mesh vertices across multiple camera views. By setting a threshold for visibility, we select mesh faces that meet or exceed this threshold. Next, we find 3D Gaussians that are closest to these selected mesh faces and render them to produce an alpha image. We then calculate the average mean Intersection over Union (mIoU) between the rendered alpha image and the original segmentation masks. Finally, we select the mesh faces and 3D Gaussians with the highest mIoU, representing the most accurate 3D segmentation.

**Lighting** Accurate lighting estimation ensures that all elements within the scene are coherently illuminated. We estimate the environmental lighting of a scene in two ways. For fully captured indoor scenes, such as those in ScanNet++ [106], we unproject the over-saturated image pixels

into space and use majority voting to determine the estimated emitter meshes. These meshes with emissions lights are subsequently imported into the renderer to serve as light sources. For partially captured indoor scenes and outdoor scenes, such as those in MipNeRF360 [5], we use DiffusionLight [67] to inpaint chrome balls in the center of initial frame at multiple exposure levels. An high dynamic range (HDR) map is then generated from these inpainted frames and imported into the renderer as an environmental light.

### 3.2. Scene Editing and Simulation

The multimodal 3D scene modeling described above paves the way for a wide assortment of editing, simulation, and rendering operations to be performed. We design a suite of intuitive modules that can be seamlessly composed together to provide a rich set of VFX capabilities. We note that this modular framework also allows new capabilities to be easily added via registering new modules. We describe the specific techniques used within each module below.

**3D Asset Creation** To enable diverse object insertions, we use the Objaverse 1.0 dataset [21] and a high-quality subset from Richdreamer [72] with 280k annotated 3D assets. We adopt a twofold approach for 3D asset creation. We first rank 3D assets using Sentence-BERT [77] to match query text and identify the top K candidates, then refine the selection with CLIP [74] based on multi-view renderings to select best aligned asset. For text queries beyond existing descriptions, we use Meshy AI to generate high-quality 3D assets with PBR materials, expanding the range of insertable objects.

**Insertion** To achieve realistic object insertion, two critical properties must be accurately determined: position and scale. To ensure plausible positioning of objects, we sample the centers of triangles from the supporting mesh that are sufficiently flat to provide accurate support for the objects. For scaling, we utilize GPT-4V [1] models to estimate the real-world dimensions of 3D assets. Detailed prompts for scale estimation are illustrated in the supplementary material.

**Removal** To effectively remove a specific instance from a scene, we begin by extracting the target objects using semantic modules. We remove these Gaussian points along with their associated mesh faces. For geometric restoration, we employ a planar mesh to cover the exposed area on the bottom. For appearance restoration, we first use LaMa [88] to inpaint the missing regions across all video frames. Then, we fine-tune the current 3D Gaussian Splatting model on the inpainted frames to ensure a 3D-consistent recovery.

**Material Editing** Accurate material editing ensures 3D assets responding correctly to lighting, shading, and environmental conditions, helping them blend seamlessly with live-action footage for convincing visual effects. We provide multiple options for material editing, all of which result in modifications to the material nodes of a 3D asset in the renderer. Users can adjust parameters such as metallic, specular, and roughness values of an asset. Users can also modify the overall color of an asset by altering the color intensity in the texture image of an 3D asset. We also support queries by material name, enabling a search across a material database sourced from PolyHaven. The queried material can then be imported and applied to all relevant material nodes.

**Physical Simulation** Because our scene model is directly compatible with Blender, we can leverage its powerful simulation capabilities by simply calling functions from its simulation library. We use these functions to enable simulations of rigid-body physics and particle effects. Rigid-body physics in Blender is based on the Bullet physics engine [20]. To achieve both accurate and realistic object interactions, we pre-compute the center of mass and convex hull for collision checking of any interactive objects. Particle effects, such as smoke and fire, rely upon mantaflow [89]. We modify the default simulation settings to ensure convincing effects. Further details are provided in the supplementary.

### 3.3. Scene Rendering and Video Compositing

We use a careful rendering and compositing scheme in order to produce a photorealistic video result. First, we render inserted objects while keeping the background mesh invisible to first-bounces during raytracing but visible to higher-order bounces. This allows the rendered objects to be affected by lighting from the background. Next, we set the background mesh to be visible and render twice: with and without the inserted objects. We use the ratio of the pixel values between these two as an approximation of the objects' effects on the background surfaces. This ratio is multiplied to either a SuGaR rendering or the original video frames, depending on if novel views are desired. Finally, we alpha-blend the inserted objects into the video, using depth maps of the background mesh for occlusion reasoning. If the objects contain fire (emissive transparent elements), we use premultiplied alpha-blending; otherwise, we use straight alpha-blending.

### 3.4. LLM Integration

AutoVFX aims to enable the creation of VFX directly from natural language instructions, providing a user-friendly interface accessible to anyone. Towards this goal, we integrate our editing modules into an API within an LLM-agent framework, drawing inspiration from recent works such as Code-as-Policies [54] and Visual Programming [35]. Leveraging GPT-4 [1], we prompt the model with in-context examples that pair editing instructions with corresponding programs composed of our predefined editing functions. The LLM


Figure 4. **Dynamic VFX video editing using AutoVFX.** Our approach enables physical interaction, articulated animation, particle effects, insertion of generated 3D assets, material editing, and geometry fracturing.

then generates a program that is directly executed to perform the specified scene edits.

**Modular Function Encapsulation** Our method encapsulates predefined editing modules into callable and executable functions, which can be combined to form comprehensive programs. Scene objects are represented as Python dictionaries, facilitating straightforward and interpretable edits. Each function's parameters are fully transparent, allowing users with varying levels of programming expertise to manipulate the process. A full list of the predefined modules is available in the supplementary material.

**LLM-driven Program Composition and Execution** To ensure GPT-4 effectively composes these functions into executable programs, we design prompts that guide the model. These prompts include examples demonstrating the combination of predefined functions into Python-like scripts that represent the desired scene edits. Once generated, the program is executed within a Python interpreter, triggering the associated simulations or rendering processes to produce the final visual effect. As illustrated in Fig. 3, this approach simplifies the creation of complex visual effects, making it accessible to a broader audience through natural language instructions. The system's modular design also provides flexibility and scalability, allowing users to customize and extend functionality as needed.

# 4. Experiments

We evaluate AutoVFX across a diverse set of scenes and editing prompts and provide both qualitative and quantitative comparisons with other related methods.

## 4.1. Experimental Details

**Dataset & Preprocessing** We adopt scenes from real-world datasets such as Mip-NeRF360 [5], Tanks & Temples [49], ScanNet++ [106], and Waymo dataset [86] to demonstrate our editing capabilities across diverse scenarios. We use COLMAP [80] to extract camera poses and sparse point clouds from images for GSplat initialization.

**Baselines** We compare our method with three text-based visual editing methods: Instruct-N2N [36], DGE [12] and FRESCO [103]. Instruct-N2N and DGE both perform edits on 3D scene representations based on text descriptions, with the former utilizing NeRF and the latter relying on 3D Gaussians. FRESCO, on the other hand, translates an input video to align with a target text prompt. For our experiments, we set the guidance scale to 12.5 for DGE to achieve more noticeable edits. Apart from this adjustment, all experimental setups adhere to the default settings of each method to ensure a fair comparison. The qualitative results for Instruct-N2N and DGE are obtained by rendering from the edited 3D representations.

Table 2. Quantitative comparison with other methods. We employ automatic metrics and human evaluation to evaluate the performance. AutoVFX consistently outperforms baseline methods across various metrics.

| Method | Semantic Consistency Measures | | | Multimodal LLM Quality Evaluation | | | | User Study | |
|---|---|---|---|---|---|---|---|---|---|
| | Object Detection | CLIP Similarity | CLIP Directional Similarity | Photo-realism | Text Alignment | Structure Preservation | Overall Quality | Text Alignment | Video Quality |
| Instruct-N2N [36] | 0.343 | 0.209 | 0.019 | 0.402 | 0.329 | 0.440 | 0.043 | 0.07 | 0.04 |
| DGE [12] | 0.347 | 0.195 | 0.278 | 0.562 | 0.312 | 0.619 | 0.106 | 0.06 | 0.03 |
| FRESCO [103] | 0.373 | **0.214** | 0.009 | 0.622 | 0.427 | 0.632 | 0.204 | 0.04 | 0.02 |
| AutoVFX (Ours) | **0.537** | 0.206 | **0.419** | **0.735** | **0.791** | **0.749** | **0.647** | **0.83** | **0.90** |

**Implementation Details** To render objects that are affected by rigid-body physics, we store the rigid transformations at each timestep and apply them to the 3D Gaussians during rendering. For animating objects based on keypoints, we use Bézier interpolation to produce a smooth trajectory. Additional implementation details regarding the various VFX modules can be found in the supplementary.

## 4.2. Qualitative evaluation

**Qualitative comparison** In Fig. 5, we compare the visual quality of static scene editing across different methods. Our approach outperforms baselines in object insertion and manipulation, delivering realistic and accurate edits while preserving scene structure. In contrast, Instruct-N2N struggles with localized editing, FRESCO fails at structural preservation, and DGE, while producing realistic videos, cannot ensure instruction alignment. Additionally, AutoVFX provides richer capabilities, such as precise material editing ("*make it mirror-like*"), accurate object counting ("*drop five basketballs*"), and advanced visual effects ("*make it on fire*").

**Dynamic video simulation** We present additional results for dynamic VFX video in Fig. 4. These highlight our method's ability to generate a wide range of realistic, physically plausible dynamic simulations from text instructions, using modules like rigid body simulations, object animation, smoke and fire, and object fracturing. None of the generative editing baselines support this feature. We also conduct experiments on autonomous driving simulation using the Waymo dataset [86], as shown in Fig. 6. AutoVFX enables both realistic rendering and realistic physical interaction between cars in collision scenarios.

## 4.3. Quantitative evaluation

We also provide a quantitative evaluation of our method. The evaluation is based on nine metrics categorized into three groups: *"Semantic Consistency Measures"*, *"Multimodal LLM Quality Evaluation"* and *"User Study"*. These metrics collectively provide a comprehensive assessment of the quality and effectiveness of text-guided visual edits. The quantitative results are presented in Table 2.

**Semantic Consistency Measures** We incorporate the "Multiple Objects" metric from VBench [43] to verify the presence of objects after editing. This metric assesses whether multiple objects are correctly composed within the edits using a detection module, ensuring that the desired semantic content has been successfully modified. Instead of using GRiT [93] as the detection module, we employ Grounded-SAM [78] for this task. We assess the success rate of visual content editing across all frames and for all possible edits. We also adopt "CLIP Similarity" and "CLIP Directional Similarity" metrics as proposed in DGE [12] for evaluation. "CLIP Similarity" measures the alignment between the text instructions and each edited frame, while "CLIP Directional Similariy" evaluates the temporal consistency of the edits across frames. Both metrics operate in CLIP space. As shown in Table 2, our method significantly outperforms other approaches in object detection score and CLIP directional similarity score, while achieving comparable results in CLIP similarity score. In particular, we improve the object detection score by a large margin, suggesting that our video edits reflect the goal of object-level changes. We notice that CLIP similarity is less discriminative among all methods and conjecture that this might be because global CLIP is not sensitive to capturing local changes, such as the insertion of small objects or dynamics. These results indicate that our method effectively aligns the edited outcomes with the provided text instructions.

**Multimodal LLM Quality Evaluation** Inspired by [95], we utilize multimodal LLMs as a powerful, interpretable, text-driven model for image quality assessment. Specifically, we prompt GPT-4o to evaluate and compare the "Overall Perceptual Quality" of four different methods based on three criteria: "Text Alignment", "Photorealism", and "Structural Preservation". We also ask GPT-4o to assign a quality score of each criterion to each method, ranging from 0 to 1, with 1 being the highest (detailed prompts could be found in supplementary). From Table 2, our method outperforms other approaches across all four metrics by a significant margin. In particular, AutoVFX creates more realistic videos, preserves structure better, and excels in "Text Alignment" by a most prominent margin. This demonstrates that our video editing produce high-quality and reasonable image edits that fully reflect the desired text instructions, which is desirable in downstream VFX applications.


Figure 5. **Qualitative comparison on static editing.**


Figure 6. **Dynamic simulation of AutoVFX on driving scenes.**

**User Study** We conduct a user study to evaluate "Text Alignment" and "Overall Realism" of performed edits. To address the potential bias where minimal changes to visual content are perceived as more realistic, we structured the survey as follows: first, users are asked to evaluate which edited videos best aligned with the given text instructions, allowing for multiple selections. In the second question, users are required to choose the most realistic video from the set of choices they previously selected. We collected a total of 36 user samples. For detailed information on the user study methodology, please refer to the supplementary materials. As shown in Table 2, our method receives a higher preference from users in both "Text Alignment" and "Overall Realism" categories, showing that the edits are not only accurate but also appealing to human judgment.

## 5. Conclusion

We presented AutoVFX, a system that automatically creates physically-grounded VFX given a monocular video and natural language instructions. AutoVFX combines neural scene modeling, LLM-based code generation, and physical simulation to allow realistic and easily controllable VFX creation. Experimental results demonstrate that AutoVFX outperforms existing scene editing methods based on a variety of practical criteria. We envision that AutoVFX will facilitate both the acceleration and democratization of visual content creation, helping both experienced artists and everyday users create the high-quality VFX that they desire.

**Acknowledgement** This project is supported by the Intel AI SRS gift, Meta research grant, the IBM IIDAI Grant and NSF Awards 2331878, 2340254, 2312102, 2414227, and 2404385. Hao-Yu Hsu is supported by Siebel Scholarship. We greatly appreciate the NCSA for providing computing resources. We thank Derek Hoiem, Sarita Adve, Benjamin Ummenhofer, Kai Yuan, Micheal Paulitsch, Katelyn Gao, Quentin Leboutet for helpful discussions.

## References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 5

[2] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. 3

[3] Autodesk, INC. Maya. 1

[4] Omer Bar-Tal, Dolev Ofri-Amar, Rafail Fridman, Yoni Kasten, and Tali Dekel. Text2live: Text-driven layered image and video editing. In *European Conference on Computer Vision*, pages 707–723. Springer, 2022. 1, 3

[5] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 5, 6

[6] Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023. 3

[7] Jeremiah U Brackbill, Douglas B Kothe, and Hans M Ruppel. Flip: a low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications*, 48(1):25–38, 1988. 3

[8] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. Instructpix2pix: Learning to follow image editing instructions. In *CVPR*, 2023. 3

[9] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021. 1

[10] Duygu Ceylan, Chun-Hao Huang, and Niloy J. Mitra. Pix2video: Video editing using image diffusion. In *ICCV*, 2023. 1, 3

[11] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 3

[12] Minghao Chen, Iro Laina, and Andrea Vedaldi. Dge: Direct gaussian 3d editing by consistent multi-view editing. *arXiv preprint arXiv:2404.18929*, 2024. 1, 3, 6, 7, 8

[13] Yiwen Chen, Zilong Chen, Chi Zhang, Feng Wang, Xiaofeng Yang, Yikai Wang, Zhongang Cai, Lei Yang, Huaping Liu, and Guosheng Lin. Gaussianeditor: Swift and controllable 3d editing with gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21476–21485, 2024. 1, 3

[14] Ho Kei Cheng, Seoung Wug Oh, Brian Price, Alexander Schwing, and Joon-Young Lee. Tracking anything with decoupled video segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1316–1326, 2023. 4, 1

[15] Yisol Choi, Sangkyung Kwak, Kyungmin Lee, Hyungwon Choi, and Jinwoo Shin. Improving diffusion models for virtual try-on. *arXiv preprint arXiv:2403.05139*, 2024. 1

[16] Chong Bao and Bangbang Yang, Zeng Junyi, Bao Hujun, Zhang Yinda, Cui Zhaopeng, and Zhang Guofeng. Neumesh: Learning disentangled neural mesh-based implicit field for geometry and texture editing. In *European Conference on Computer Vision (ECCV)*, 2022. 3

[17] Mark Christiansen. *Adobe After Effects CC Visual Effects and Compositing Studio Techniques*. Adobe Press, 2013. 1

[18] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 1

[19] Yuren Cong, Mengmeng Xu, Christian Simon, Shoufa Chen, Jiawei Ren, Yanping Xie, Juan-Manuel Perez-Rua, Bodo Rosenhahn, Tao Xiang, and Sen He. Flatten: optical flow-guided attention for consistent text-to-video editing. *arXiv preprint arXiv:2310.05922*, 2023. 1, 3

[20] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, 2015. 5, 2

[21] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153, 2023. 5

[22] Jiahua Dong and Yu-Xiong Wang. Vica-nerf: View-consistency-aware 3d editing of neural radiance fields. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 1, 3

[23] Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, et al. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32):e2123433119, 2022. 3

[24] Ainaz Eftekhar, Alexander Sax, Jitendra Malik, and Amir Zamir. Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10786–10796, 2021. 1

[25] Jiemin Fang, Junjie Wang, Xiaopeng Zhang, Lingxi Xie, and Qi Tian. Gaussianeditor: Editing 3d gaussians delicately with text instructions. In *CVPR*, 2024. 1, 3

[26] Shuangkang Fang, Yufeng Wang, Yi-Hsuan Tsai, Yi Yang, Wenrui Ding, Shuchang Zhou, and Ming-Hsuan Yang. Chat-edit-3d: Interactive 3d scene editing via text prompts. *arXiv preprint arXiv:2407.06842*, 2024. 1, 3, 4

[27] Yutao Feng, Yintong Shang, Xuan Li, Tianjia Shao, Chenfanfu Jiang, and Yin Yang. Pie-nerf: Physics-based interactive elastodynamics with nerf, 2023. 3

[28] Yutao Feng, Xiang Feng, Yintong Shang, Ying Jiang, Chang Yu, Zeshun Zong, Tianjia Shao, Hongzhi Wu, Kun Zhou, Chenfanfu Jiang, and Yin Yang. Gaussian splashing: Unified particles for versatile motion synthesis and rendering. *arXiv preprint arXiv:2401.15318*, 2024. 3

[29] Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A Smith, Wei-Chiu Ma, and Ranjay Krishna. Blink: Multimodal large language models can see but not perceive. *arXiv preprint arXiv:2404.12390*, 2024. 3

[30] Gege Gao, Weiyang Liu, Anpei Chen, Andreas Geiger, and Bernhard Schölkopf. Graphdreamer: Compositional 3d scene synthesis from scene graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21295–21304, 2024. 4

[31] Ruiyuan Gao, Kai Chen, Enze Xie, Lanqing Hong, Zhenguo Li, Dit-Yan Yeung, and Qiang Xu. Magicdrive: Street view generation with diverse 3d geometry control. *arXiv preprint arXiv:2310.02601*, 2023. 1

[32] Michal Geyer, Omer Bar-Tal, Shai Bagon, and Tali Dekel. Tokenflow: Consistent diffusion features for consistent video editing. *arXiv preprint arxiv:2307.10373*, 2023. 1, 3

[33] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. *arXiv preprint arXiv:2311.12775*, 2023. 4, 1

[34] Jianfei Guo, Nianchen Deng, Xinyang Li, Yeqi Bai, Botian Shi, Chiyu Wang, Chenjing Ding, Dongliang Wang, and Yikang Li. Streetsurf: Extending multi-view implicit surface reconstruction to street views. *arXiv preprint arXiv:2306.04988*, 2023. 1

[35] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023. 3, 4, 5

[36] Ayaan Haque, Matthew Tancik, Alexei Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-nerf2nerf: Editing 3d scenes with instructions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023. 1, 3, 6, 7, 8

[37] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020. 3

[38] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019. 1

[39] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019.

[40] Yuanming Hu, Jiafeng Liu, Xuanda Yang, Mingkuan Xu, Ye Kuang, Weiwei Xu, Qiang Dai, William T Freeman, and Frédo Durand. Quantaichi: a compiler for quantized simulations. *ACM Transactions on Graphics (TOG)*, 40(4): 1–16, 2021. 1

[41] Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A Ross, Cordelia Schmid, and Alireza Fathi. Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In *Forty-first International Conference on Machine Learning*, 2024. 4

[42] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023. 4

[43] Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. 7

[44] Side Effects Software Inc. *SideFX Houdini FX*. SideFX, 2018. 1

[45] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. Mitsuba 3 renderer, 2022. https://mitsuba-renderer.org. 1

[46] Yingwenqi Jiang, Jiadong Tu, Yuan Liu, Xifeng Gao, Xiaoxiao Long, Wenping Wang, and Yuexin Ma. Gaussianshader: 3d gaussian splatting with shading functions for reflective surfaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5322–5332, 2024. 1

[47] Ying Jiang, Chang Yu, Tianyi Xie, Xuan Li, Yutao Feng, Huamin Wang, Minchen Li, Henry Lau, Feng Gao, Yin Yang, and Chenfanfu Jiang. Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. *arXiv preprint arXiv:2401.16663*, 2024. 3

[48] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023. 4, 1

[49] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36 (4):1–13, 2017. 6

[50] Max Ku, Cong Wei, Weiming Ren, Huan Yang, and Wenhu Chen. Anyv2v: A plug-and-play framework for any video-to-video editing tasks. *arXiv preprint arXiv:2403.14468*, 2024. 1, 3

[51] Xuan Li, Yi-Ling Qiao, Peter Yichen Chen, Krishna Murthy Jatavallabhula, Ming Lin, Chenfanfu Jiang, and Chuang Gan. PAC-neRF: Physics augmented continuum neural radiance fields for geometry-agnostic system identification. In *The*

*Eleventh International Conference on Learning Representations*, 2023. 3

[52] Xiaofan Li, Yifu Zhang, and Xiaoqing Ye. Drivingdiffusion: Layout-guided multi-view driving scene video generation with latent diffusion model. *arXiv preprint arXiv:2310.07771*, 2023. 1

[53] Yuan Li, Zhi-Hao Lin, David Forsyth, Jia-Bin Huang, and Shenlong Wang. Climatenerf: Extreme weather synthesis in neural radiance field. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. 1, 3

[54] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023. 4, 5

[55] Ruofan Liang, Zan Gojcic, Merlin Nimier-David, David Acuna, Nandita Vijaykumar, Sanja Fidler, and Zian Wang. Photorealistic object insertion with diffusion-guided inverse rendering. *arXiv preprint arXiv:2408.09702*, 2024. 1

[56] Zhi-Hao Lin, Bohan Liu, Yi-Ting Chen, David Forsyth, Jia-Bin Huang, Anand Bhattad, and Shenlong Wang. Urbanir: Large-scale urban scene inverse rendering from a single video. *arXiv preprint arXiv:2306.09349*, 2023.

[57] Zhi-Hao Lin, Jia-Bin Huang, Zhengqin Li, Zhao Dong, Christian Richardt, Tuotuo Li, Michael Zollhöfer, Johannes Kopf, Shenlong Wang, and Changil Kim. Iris: Inverse rendering of indoor scenes from low dynamic range images. *arXiv preprint arXiv:2401.12977*, 2024. 1

[58] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023. 4

[59] Shaowei Liu, Zhongzheng Ren, Saurabh Gupta, and Shenlong Wang. Physgen: Rigid-body physics-grounded image-to-video generation. In *European Conference on Computer Vision*, pages 360–378. Springer, 2025. 3

[60] Jiaxi Lv, Yi Huang, Mingfu Yan, Jiancheng Huang, Jianzhuang Liu, Yifan Liu, Yafei Wen, Xiaoxin Chen, and Shifeng Chen. Gpt4motion: Scripting physical motions in text-to-video generation via blender-oriented gpt planning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 1430–1440, 2024. 3, 4

[61] Yunsheng Ma, Can Cui, Xu Cao, Wenqian Ye, Peiran Liu, Juanwu Lu, Amr Abdelraouf, Rohit Gupta, Kyungtae Han, Aniket Bera, et al. Lampilot: An open benchmark dataset for autonomous driving with language model programs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15141–15151, 2024. 4

[62] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *International Conference on Learning Representations*, 2022. 3

[63] Ashkan Mirzaei, Tristan Aumentado-Armstrong, Marcus A Brubaker, Jonathan Kelly, Alex Levinshtein, Konstantinos G Derpanis, and Igor Gilitschenski. Watch your steps: Local image and scene editing by text instructions. *arXiv preprint arXiv:2308.08947*, 2023. 1, 3

[64] Chong Mou, Mingdeng Cao, Xintao Wang, Zhaoyang Zhang, Ying Shan, and Jian Zhang. Revideo: Remake a video with motion and content control, 2024. 1, 3

[65] Hao Ouyang, Qiuyu Wang, Yuxi Xiao, Qingyan Bai, Juntao Zhang, Kecheng Zheng, Xiaowei Zhou, Qifeng Chen, and Yujun Shen. Codef: Content deformation fields for temporally consistent video processing. *arXiv preprint arXiv:2308.07926*, 2023. 1, 3

[66] Yicong Peng, Yichao Yan, Shengqi Liu, Yuhao Cheng, Shanyan Guan, Bowen Pan, Guangtao Zhai, and Xiaokang Yang. Cagenerf: Cage-based neural radiance field for generalized 3d deformation and animation. *Advances in Neural Information Processing Systems*, 35:31402–31415, 2022. 3

[67] Pakkapon Phongthawee, Worameth Chinchuthakun, Nontaphat Sinsunthithet, Amit Raj, Varun Jampani, Pramook Khungurn, and Supasorn Suwajanakorn. Diffusionlight: Light probes for free by painting a chrome ball. *arXiv preprint arXiv:2312.09168*, 2023. 5, 1

[68] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-spline techniques*. Springer, 2002. 2

[69] Chenyang Qi, Xiaodong Cun, Yong Zhang, Chenyang Lei, Xintao Wang, Ying Shan, and Qifeng Chen. Fatezero: Fusing attentions for zero-shot text-based video editing. *arXiv:2303.09535*, 2023. 1, 3

[70] Yi-Ling Qiao, Alexander Gao, and Ming C. Lin. Neuphysics: Editable neural geometry and physics from monocular videos. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022. 3

[71] Yi-Ling Qiao, Alexander Gao, Yiran Xu, Yue Feng, Jia-Bin Huang, and Ming C. Lin. Dynamic mesh-aware radiance fields. *ICCV*, 2023. 1, 3

[72] Lingteng Qiu, Guanying Chen, Xiaodong Gu, Qi Zuo, Mutian Xu, Yushuang Wu, Weihao Yuan, Zilong Dong, Liefeng Bo, and Xiaoguang Han. Richdreamer: A generalizable normal-depth diffusion model for detail richness in text-to-3d. *arXiv preprint arXiv:2311.16918*, 2023. 5

[73] Ri-Zhao Qiu, Ge Yang, Weijia Zeng, and Xiaolong Wang. Language-driven physics-based scene synthesis and editing via feature splatting. In *European Conference on Computer Vision (ECCV)*, 2024. 3, 2

[74] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 5

[75] Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, et al. Infinite photorealistic worlds using procedural generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12630–12641, 2023. 4

[76] Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, et al. Infinigen indoors: Photorealistic indoor scenes using procedural generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21783–21794, 2024. 4

[77] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019. 5

[78] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, Zhaoyang Zeng, Hao Zhang, Feng Li, Jie Yang, Hongyang Li, Qing Jiang, and Lei Zhang. Grounded sam: Assembling open-world models for diverse visual tasks, 2024. 7

[79] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. 3

[80] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 6, 1

[81] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016. 4

[82] phymec Sergey Sharybin, ideasman42. Cell Fracture, 2024. *https://extensions.blender.org/add-ons/cell-fracture/*. 2

[83] Yuan Shen, Bhargav Chandaka, Zhi-Hao Lin, Albert Zhai, Hang Cui, David Forsyth, and Shenlong Wang. Sim-on-wheels: Physical world in the loop simulation for self-driving. *IEEE Robotics and Automation Letters*, 2023. 1

[84] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023. 4

[85] Liangchen Song, Liangliang Cao, Jiatao Gu, Yifan Jiang, Junsong Yuan, and Hao Tang. Efficient-nerf2nerf: Streamlining text-driven 3d editing with multiview correspondence-enhanced diffusion models. *arXiv preprint arXiv:2312.08563*, 2023. 3

[86] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020. 6, 7, 1

[87] Shuo Sun, Zekai Gu, Tianchen Sun, Jiawei Sun, Chengran Yuan, Yuhang Han, Dongen Li, and Marcelo H Ang. Drivescenegen: Generating diverse and realistic driving scenarios from scratch. *IEEE Robotics and Automation Letters*, 2024. 1

[88] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. Resolution-robust large mask inpainting with fourier convolutions. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2149–2159, 2022. 5

[89] Nils Thuerey and Tobias Pfaff. MantaFlow, 2018. *http://mantaflow.com*. 5, 3

[90] Yuxi Wei, Zi Wang, Yifan Lu, Chenxin Xu, Changxing Liu, Hao Zhao, Siheng Chen, and Yanfeng Wang. Editable scene simulation for autonomous driving via collaborative llm-agents. *arXiv preprint arXiv:2402.05746*, 2024. 3, 1, 7, 9

[91] Jing Wen, Xiaoming Zhao, Zhongzheng Ren, Alexander G Schwing, and Shenlong Wang. Gomavatar: Efficient animatable human modeling from monocular video using gaussians-on-mesh. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2059–2069, 2024. 2

[92] Bichen Wu, Ching-Yao Chuang, Xiaoyan Wang, Yichen Jia, Kapil Krishnakumar, Tong Xiao, Feng Liang, Licheng Yu, and Peter Vajda. Fairy: Fast parallelized instruction-guided video-to-video synthesis. *arXiv preprint arXiv:2312.13834*, 2023. 1, 3

[93] Jialian Wu, Jianfeng Wang, Zhengyuan Yang, Zhe Gan, Zicheng Liu, Junsong Yuan, and Lijuan Wang. Grit: A generative region-to-text transformer for object understanding. *arXiv preprint arXiv:2212.00280*, 2022. 7

[94] Jay Zhangjie Wu, Yixiao Ge, Xintao Wang, Stan Weixian Lei, Yuchao Gu, Yufei Shi, Wynne Hsu, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. Tune-a-video: One-shot tuning of image diffusion models for text-to-video generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7623–7633, 2023. 1, 3

[95] Tianhe Wu, Kede Ma, Jie Liang, Yujiu Yang, and Lei Zhang. A comprehensive study of multimodal large language models for image quality assessment. *arXiv preprint arXiv:2403.10854v3*, 2024. 7

[96] Hongchi Xia, Zhi-Hao Lin, Wei-Chiu Ma, and Shenlong Wang. Video2game: Real-time interactive realistic and browser-compatible environment from a single video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4588, 2024. 3

[97] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in neural information processing systems*, 34: 12077–12090, 2021. 1

[98] Tianyi Xie, Zeshun Zong, Yuxing Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. Physgaussian: Physics-integrated 3d gaussians for generative dynamics. *arXiv preprint arXiv:2311.12198*, 2023. 3, 1, 2

[99] Tianhan Xu and Tatsuya Harada. Deforming radiance fields with cages. In *ECCV*, 2022. 3

[100] Teng Xu, Jiamin Chen, Peng Chen, Youjia Zhang, Junqing Yu, and Wei Yang. Tiger: Text-instructed 3d gaussian retrieval and coherent editing. *arXiv preprint arXiv:2405.14455*, 2024. 1, 3

[101] Jingkang Yang, Yuhao Dong, Shuai Liu, Bo Li, Ziyue Wang, Chencheng Jiang, Haoran Tan, Jiamu Kang, Yuanhan Zhang,

Kaiyang Zhou, et al. Octopus: Embodied vision-language programmer from environmental feedback. *arXiv preprint arXiv:2310.08588*, 2023. 4

[102] Mengjiao Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. *arXiv preprint arXiv:2310.06114*, 2023. 1

[103] Shuai Yang, Yifan Zhou, Ziwei Liu, , and Chen Change Loy. Fresco: Spatial-temporal correspondence for zero-shot video translation. In *CVPR*, 2024. 1, 3, 6, 7, 8

[104] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P Srinivasan, Richard Szeliski, Jonathan T Barron, and Ben Mildenhall. Bakedsdf: Meshing neural sdfs for real-time view synthesis. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–9, 2023. 4, 1

[105] Mingqiao Ye, Martin Danelljan, Fisher Yu, and Lei Ke. Gaussian grouping: Segment and edit anything in 3d scenes. In *ECCV*, 2024. 3

[106] Chandan Yeshwanth, Yueh-Cheng Liu, Matthias Nießner, and Angela Dai. Scannet++: A high-fidelity dataset of 3d indoor scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12–22, 2023. 4, 6, 1

[107] Zehao Yu, Anpei Chen, Bozidar Antic, Songyou Peng, Apratim Bhattacharyya, Michael Niemeyer, Siyu Tang, Torsten Sattler, and Andreas Geiger. Sdfstudio: A unified framework for surface reconstruction, 2022. 1

[108] Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. Nerf-editing: geometry editing of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18353–18364, 2022. 3

[109] Albert J Zhai, Yuan Shen, Emily Y Chen, Gloria X Wang, Xinlei Wang, Sheng Wang, Kaiyu Guan, and Shenlong Wang. Physical property understanding from language-embedded feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28296–28305, 2024. 3

[110] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models, 2023. 3

[111] Tianyuan Zhang, Hong-Xing Yu, Rundi Wu, Brandon Y. Feng, Changxi Zheng, Noah Snavely, Jiajun Wu, and William T. Freeman. PhysDreamer: Physics-based interaction with 3d objects via video generation. *arxiv*, 2024. 3

[112] Mengqi Zhou, Jun Hou, Chuanchen Luo, Yuxi Wang, Zhaoxiang Zhang, and Junran Peng. Scenex: Procedural controllable large-scale scene generation via large-language models. *arXiv preprint arXiv:2403.15698*, 2024. 4

[113] Jingyu Zhuang, Chen Wang, Liang Lin, Lingjie Liu, and Guanbin Li. Dreameditor: Text-driven 3d scene editing with neural fields. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–10, 2023. 1, 3

[114] Jingyu Zhuang, Di Kang, Yan-Pei Cao, Guanbin Li, Liang Lin, and Ying Shan. Tip-editor: An accurate 3d editor following both text-prompts and image-prompts. *arXiv preprint arXiv:2401.14828*, 2024. 1, 3

# AutoVFX: Physically Realistic Video Editing from Natural Language Instructions

Supplementary Material

## 6. Implementation details

In this section, we provide an overview of our framework, followed by a detailed explanation of the implementation, including scene modeling, simulation, rendering, composition, and LLM integration. We plan to release the entire codebase upon acceptance.

### 6.1. Holistic overview

We use Blender's modules [18] to implement all the editing, simulation, and rendering components. These include Cycles renderer, Material Nodes, Mantaflow fluid simulation and Composition Nodes. We chose Blender because: (1) it includes all the necessary modules required by AutoVFX, and (2) it offers a convenient Python-based interface for modular function encapsulation and code generation. However, AutoVFX is generic, allowing the easy integration of new modules for additional functionality. One can choose different low-level implementations, whether Blender or other tools with a Python-based interface, such as Mitsuba for rendering [45] or Taichi for simulation [38–40].

### 6.2. Scene modeling details

**Geometry** We employ BakedSDF [104], implemented in SDFStudio [107], to obtain high-quality scene geometry due to its detailed mesh extraction. Specifically, we use *bakedsdf-mlp* model. This model is trained for 250k steps using default optimization and model settings, with an additional monocular normal consistency loss set by *pipeline.model.mono-normal-loss-mult=0.1*. Monocular normal maps are obtained from Omnidata [24]. For fully-captured indoor scenes such as ScanNet++ [106], we enable the inside-outside flag with *pipeline.model.sdf-field.inside-outside=True*. For scenes with distant backgrounds, we enable background modeling by setting *pipeline.model.background-model=mlp*.

While BakedSDF excels in capturing object-centric scenes, it struggles with non-object-centric, long, and narrow camera trajectories, such as those in street views for autonomous driving. To address this limitation, we use StreetSurf [34] for geometry reconstruction in road scenes from the Waymo dataset [86]. For a fair comparison, we do not utilize LiDAR point clouds for precise geometry initialization; instead, we use three camera views (Front, Front Left, Front Right), consistent with ChatSim [90], along with monocular normal and depth priors from Omnidata, and sky masks extracted using SegFormer [97].

**Appearance & Semantics** To model appearance, we use both 3D Gaussian Splatting [48] and SuGaR [33]. The model is first trained with 3D Gaussian Splatting for 15000 steps, followed by an additional 7000 steps using SuGaR, all with default optimization parameters. To achieve a denser initialization for better rendering quality, we enhance the Gaussian initialization from COLMAP [80] points by computing ray-mesh intersections for each training view and assigning pixel RGB values and intersected points to set up the Gaussians. For loss terms, we apply the anisotropic regularizer from PhysGaussian [98] to prevent the emergence of spiky Gaussians during training. Additionally, we incorporate normal regularization from GaussianShader [46] to ensure consistency between local geometry and estimated normals. An anisotropic loss weight of 0.1 and a normal loss weight of 0.01 are used across all scenes. These regularizations help maintain the Gaussians' shape and orientation, facilitating better instance extraction. To avoid false-positive predictions in the semantic branches, we increase the DINO [9] threshold to 0.45 in DEVA [14]. Full pseudo code for 3D instance segmentation on both meshes and 3D Gaussians are illustrated in Fig. 17.

**Lighting** To illuminate the scene with surrounding light, we extract an HDR environmental map from a single image using DiffusionLight [67]. We begin by center-cropping the image to 512x512 pixels, then inpainting a chrome ball using a diffusion model. The chrome ball is subsequently unwrapped to create the environmental map. Multiple chrome balls with varying exposure values are generated and merged to produce the final HDR map. This map is then transformed based on the camera poses of the original image to align it with the world space.

For consistent lighting effects in Blender, we adjust the HDR map's intensity according to the scene type: 0.6 for outdoor scenes and 2.0 for indoor scenes. In fully-captured indoor environments like ScanNet++ [106], where HDR maps are insufficient due to occlusions by surrounding geometry like walls and ceilings, we extract emitter meshes by unprojecting over-saturated pixels into 3D space and using majority voting to estimate the emitter locations. These meshes are imported into Blender as white-colored emitters, with their strength set to 100. For outdoor autonomous driving scenes, such as those in Waymo [86], the HDR map alone is insufficient for casting strong shadows. To address this, we determine the sunlight direction from the brightest area in the HDR map and add a corresponding sunlight source in Blender, enhancing shadow realism on the road. The impact of this additional sunlight source is illustrated in Fig. 9.

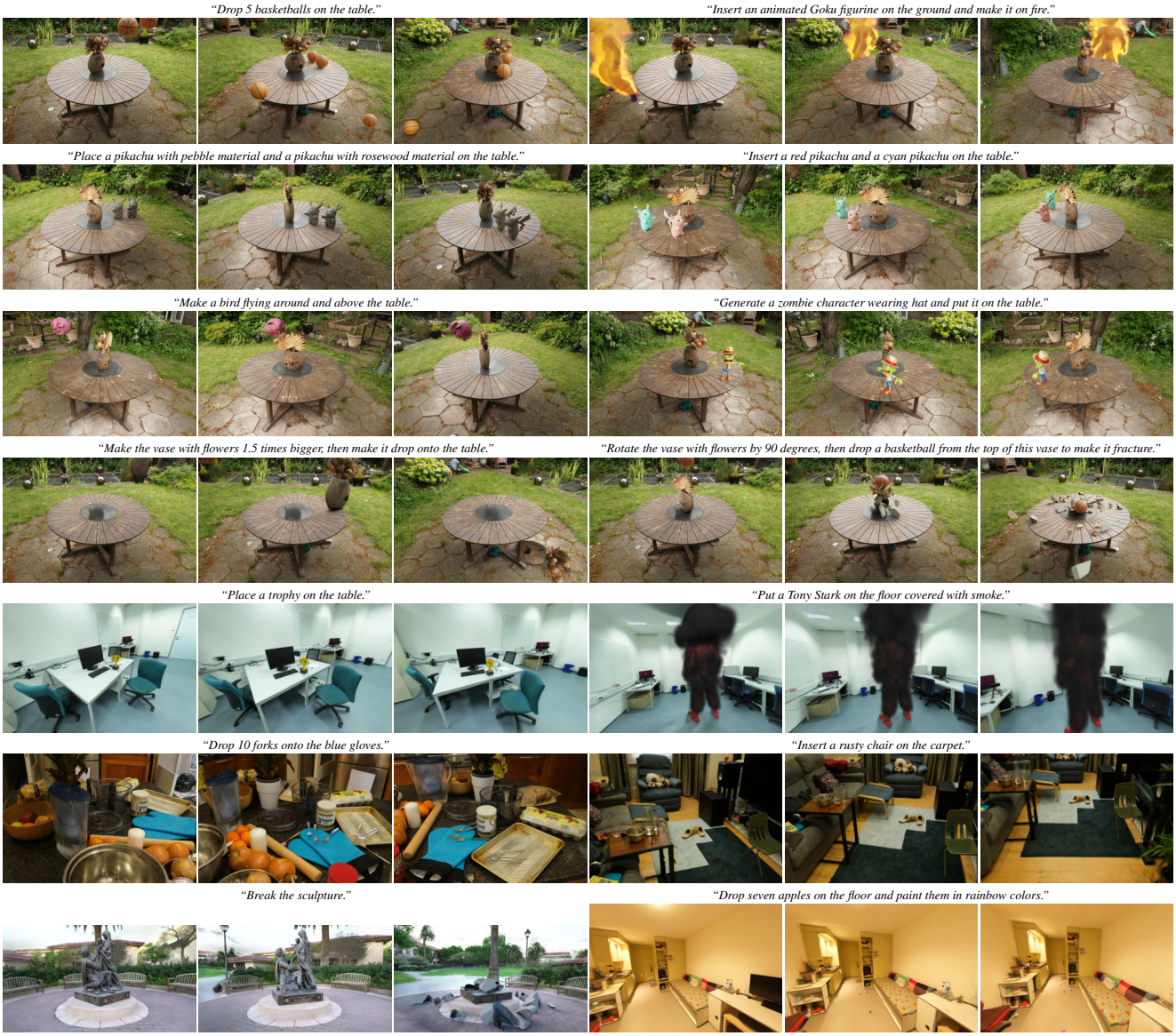


Figure 7. More editing results using AutoVFX.

### 6.3. Scene simulation details

**Animation and rigidbody simulation** To simulate the movement of animated objects along a series of 3D keypoints, we use Bézier curves [68] to generate a smooth, continuous path from discrete sample positions, ensuring seamless transitions of the animated objects. We additionally model object-scene rigid body interactions using Blender, which is based on the Bullet physics engine [20]. To achieve both accurate and realistic interactions, we also pre-compute the center of mass and convex hull for collision checking of any interactive objects. For object assets extracted from the scene, which require rendering with 3D Gaussians post-simulation, we preserve the rigid body transformations at each timestep. These transformations are then applied to the 3D Gaussians during rendering. This process closely follows the principles of recent works on deforming Gaussians [73, 91, 98], where the centroids and covariance of the Gaussians are adjusted through translation, rotation, and scaling.

**Physical effects** Realistic VFX effects often require compelling physical simulations, such as fracture effects or particle effects like smoke and fire. For fracture effects, we employ the cell fracture algorithm [82] to generate self-fracturing objects. We configure the fracture count to 100 and apply the object's average color to the internal fractures.

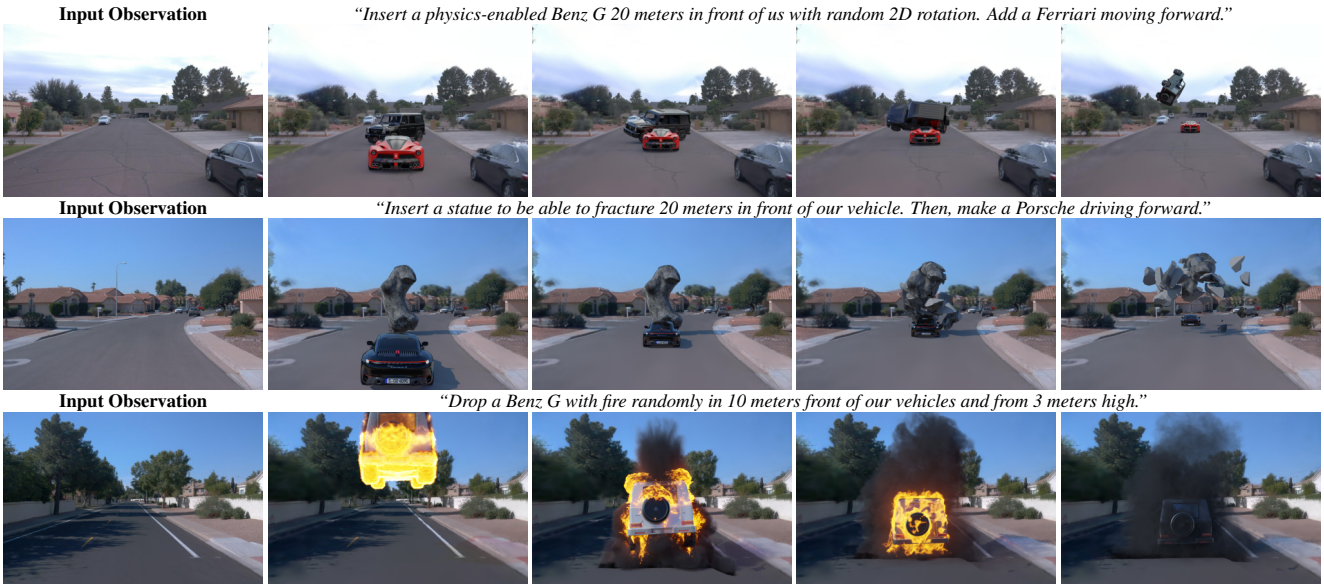


Figure 8. More dynamic simulation results of AutoVFX on autonomous driving scenes.


Figure 9. Comparison of simulation results with and without sunlight in Waymo scenes.

For particle effects, we adopt Blender's computational fluid dynamics addon Mantaflow [89], which is an efficient implementation of the FLIP-based [7] particle simulation method, to simulate smoke emission. To balance computational efficiency and quality in Blender, we configure the smoke domain with a resolution of 128, an adaptive margin of 4, an adaptive threshold of 0.005, and a dissolve speed of 30. We modify the material nodes to further enhance the realism of smoke and fire effects. For smoke simulation, we set the smoke color to (0.1, 0.1, 0.1, 1) and the smoke density to 70. For fire simulation, we reduce the smoke density to 50, set the object's temperature to 1500, and configure the blackbody tint and intensity to (1, 0.3886, 0.0094, 1) and 5, respectively.

## 6.4. Rendering & composition details

**Rendering** We use Blender's Cycles renderer for rendering. Cycles is Blender's physically-based path tracing renderer, designed for high-quality, photorealistic rendering. It accurately simulates light interactions, including reflections, refractions, and global illumination, making it ideal for realistic visual effects and animations. In our workflow, we render three outputs: foreground objects, background meshes, and a combined render of the two, as detailed in the main paper. To make foreground objects affected by lighting from the background, we set `visible_camera=False` for background meshes to make them invisible to the camera on the first light bounce but still affects subsequent bounces. The default number of samples in Cycles are set to 64, increased to 512 for scenes involving smoke and fire simulations to better capture particle effect details. Images are rendered at 2x resolution to mitigate aliasing during compositing.

**Compositing** The final visual effects are achieved through a compositing pipeline that blends visual content into the original frames. This process involves extracting foreground and background masks, and foreground content via alpha thresholding and occlusion reasoning, and calculating shadow intensity as the pixel value ratio between the combined and background renders. Shadows are then blended into the original image, followed by the integration of foreground content, resulting in the final composited video. The composition pipeline is illustrated in Fig. 10.

## 6.5. LLM integration details

**Modular functions design** The predefined editing modules are encapsulated into callable and executable functions that can be utilized by LLM. We provide a list of all designed modules, along with a brief introduction to each, including its purpose, inputs, and outputs. For further details on the editing modules, please refer to the attached file

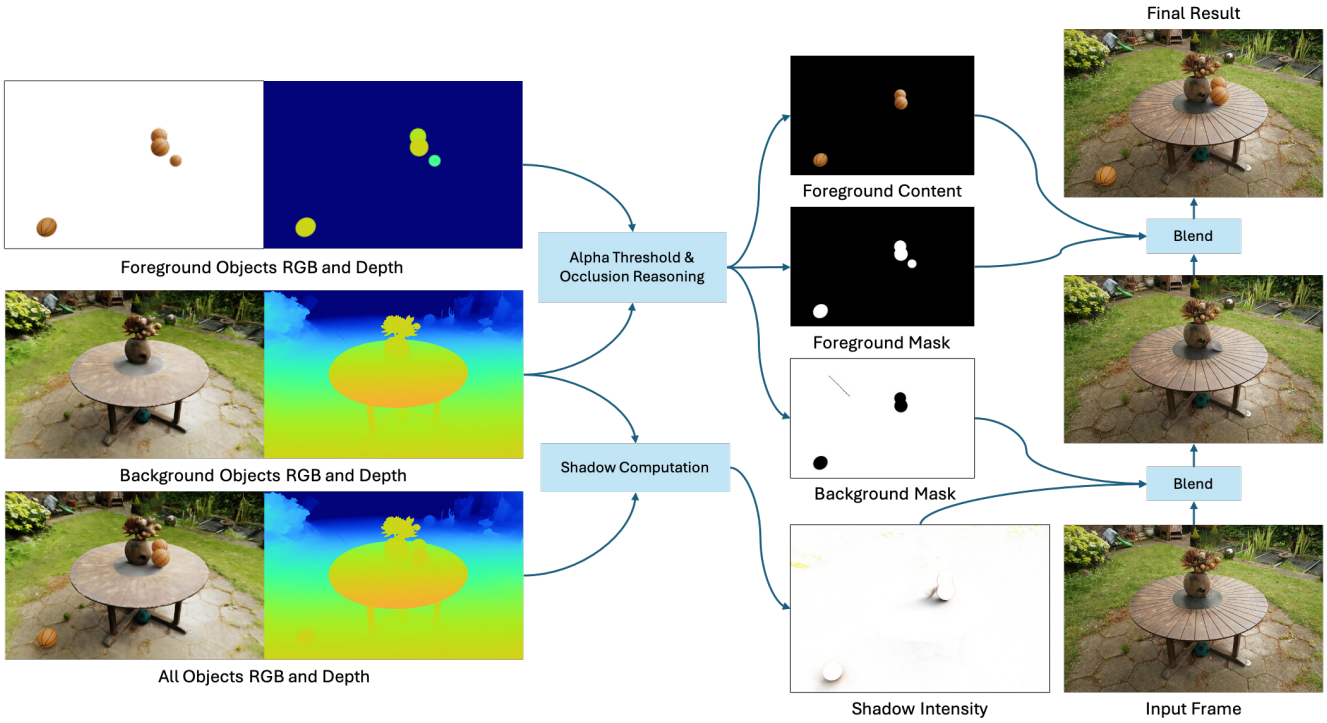


Figure 10. Our image composition pipeline. The process starts by generating foreground and background masks, along with foreground content, through alpha thresholding and occlusion reasoning based on rendered objects and background meshes. Next, shadow intensity is calculated by determining the ratio of pixel values between the combined rendering of all objects and the background meshes. Finally, the shadows and foreground content are sequentially blended into the original image to produce the final result.

`edit_utils.py`. Details of editing modules:

- **`detect_object`**
  - **Purpose:** Detects and extracts instance-level meshes from a scene.
  - **Input:**
    - scene_representation: The representation of the scene in which to detect the object.
    - object_name: The name of the object to be detected in the scene.
  - **Output:**
    - A dictionary containing information about the detected object.
- **`sample_point_on_object`**
  - **Purpose:** Samples a point on the surface of an object mesh.
  - **Input:**
    - scene_representation: The scene in which the object is located.
    - obj: The object on which to sample a point.
  - **Output:**
    - A 3D point location on the object.
- **`sample_point_above_object`**
  - **Purpose:** Samples a point above an object at a specified vertical offset.
  - **Input:**
    - scene_representation: The scene in which the object is located.
    - obj: The object above which to sample a point.
    - VERTICAL_OFFSET: The vertical distance above the object to sample the point (optional).
  - **Output:**
    - A 3D point location above the object.
- **`retrieve_asset`**
  - **Purpose:** Retrieves a 3D asset by its name from objaverse.
  - **Input:**
    - scene_representation: The scene in which to retrieve the asset.
    - object_name: The name of the asset to retrieve.
    - is_animated: Boolean flag indicating if the asset is animated (optional).
    - is_generated: Boolean flag indicating if the asset is generated (optional).
  - **Output:**
    - A dictionary containing information about the retrieved object.
- **`insert_object`**
  - **Purpose:** Inserts an object into the scene.

  - **Input:**
    * scene_representation: The scene representation into which the object is inserted.
    * obj: The object to insert into the scene.
  - **Output:** None
- **`remove_object`**
  - **Purpose:** Removes an object from the scene, with optional inpainting.
  - **Input:**
    * scene_representation: The scene from which the object is to be removed.
    * obj: The object to be removed.
    * remove_gaussians: Boolean flag to determine if associated Gaussian splatting should also be removed (optional).
  - **Output:** None
- **`update_object`**
  - **Purpose:** Updates an object's information in the scene.
  - **Input:**
    * scene_representation: The scene representation that contains the object.
    * obj: The object whose information is to be updated.
  - **Output:** None
- **`allow_physics`**
  - **Purpose:** Enables rigid body simulation for an object.
  - **Input:**
    * obj: The object to enable physics for.
  - **Output:**
    * Updated object dictionary with rigid body settings.
- **`add_fire`**
  - **Purpose:** Adds fire to an object in the scene.
  - **Input:**
    * scene_representation: The scene representation containing the object.
    * obj: The object to which fire is added.
  - **Output:** None
- **`add_smoke`**
  - **Purpose:** Adds smoke to an object in the scene.
  - **Input:**
    * scene_representation: The scene representation containing the object.
    * obj: The object to which smoke is added.
  - **Output:** None
- **`set_static_animation`**
  - **Purpose:** Sets an object's animation to be static.
  - **Input:**
    * obj: The object to set as static.
  - **Output:**
    * Updated object dictionary with animation settings.
- **`set_moving_animation`**
  - **Purpose:** Sets an object's trajectory based on a list of 3D points.
  - **Input:**
    * obj: The object to animate.
    * points: List of 3D points defining the trajectory.
  - **Output:**
    * Updated object dictionary with trajectory settings.
- **`init_material`**
  - **Purpose:** Initializes a material instance with default values.
  - **Input:** None
  - **Output:**
    * An instance of the Material class.
- **`retrieve_material`**
  - **Purpose:** Retrieves a material by its name from PolyHaven.
  - **Input:**
    * scene_representation: The scene representation that requires the material.
    * material_name: The name of the material to retrieve.
  - **Output:**
    * Path to the material folder.
- **`apply_material`**
  - **Purpose:** Applies a material to an object.
  - **Input:**
    * obj: The object to which the material is applied.
    * material: The material instance to apply.
  - **Output:**
    * Updated object dictionary with applied material.
- **`allow_fracture`**
  - **Purpose:** Enables fracturing of an object.
  - **Input:**
    * obj: The object to enable fracturing for.
  - **Output:**
    * Updated object dictionary with fracture settings.
- **`make_break`**
  - **Purpose:** Breaks an object into multiple pieces.
  - **Input:**
    * obj: The object to break.
  - **Output:**
    * Updated object dictionary with break settings.
- **`make_melting`**
  - **Purpose:** Melts down an object into viscous liquid.
  - **Input:**
    * obj: The object to melt down.
  - **Output:**
    * Updated object dictionary with melting settings.
- **`get_object_center_position`**
  - **Purpose:** Returns the position of the object at its center.
  - **Input:**
    * obj: The object whose center position is required.
  - **Output:**
    * A 3D position vector.
- **`get_object_bottom_position`**
  - **Purpose:** Returns the position of the object at its bottom.

  - **Input:**
    - obj: The object whose bottom position is required.
  - **Output:**
    - A 3D position vector.
- **`translate_object`**
  - **Purpose:** Translates an object by a given translation vector.
  - **Input:**
    - obj: The object to translate.
    - translation: The translation vector.
  - **Output:**
    - Updated object dictionary with new position.
- **`rotate_object`**
  - **Purpose:** Rotates an object by a given rotation matrix.
  - **Input:**
    - obj: The object to rotate.
    - rotation: The rotation matrix.
  - **Output:**
    - Updated object dictionary with new rotation.
- **`scale_object`**
  - **Purpose:** Scales an object by a given scale factor.
  - **Input:**
    - obj: The object to scale.
    - scale: The scale factor.
  - **Output:**
    - Updated object dictionary with new scale.
- **`get_random_2D_rotation`**
  - **Purpose:** Returns a random 2D rotation matrix (rotation around the z-axis).
  - **Input:** None
  - **Output:**
    - 3x3 rotation matrix.
- **`get_random_3D_rotation`**
  - **Purpose:** Returns a random 3D rotation matrix.
  - **Input:** None
  - **Output:**
    - 3x3 rotation matrix.
- **`make_copy`**
  - **Purpose:** Creates a deep copy of an object.
  - **Input:**
    - obj: The object to copy.
  - **Output:**
    - New object dictionary with a unique object_id.
- **`add_event`**
  - **Purpose:** Adds an event to the scene involving an object.
  - **Input:**
    - scene_representation: The scene representation to which the event is added.
    - obj: The object involved in the event.
    - event_type: The type of event to add (e.g., "break", "incinerate").
    - start_frame: The frame at which the event starts (optional).
    - end_frame: The frame at which the event ends (optional).
  - **Output:** None
- **`get_camera_position`**
  - **Purpose:** Returns the camera position.
  - **Input:**
    - scene_representation: The scene representation containing the camera.
  - **Output:**
    - 3D position vector.
- **`get_vehicle_position`**
  - **Purpose:** Returns the position of a vehicle in the scene.
  - **Input:**
    - scene_representation: The scene representation containing the vehicle.
  - **Output:**
    - 3D position vector (with z-value set to 0.0).
- **`get_direction`**
  - **Purpose:** Returns the direction vector from the camera position in one of six directions (front, back, left, right, up, down).
  - **Input:**
    - scene_representation: The scene representation containing the camera.
    - direction: The direction in which to get the vector (e.g., "front", "back").
  - **Output:**
    - 3D direction vector.
- **`retrieve_chatsim_asset`**
  - **Purpose:** Retrieves a 3D asset by object name from the chatsim asset bank.
  - **Input:**
    - scene_representation: The scene representation requiring the asset.
    - object_name: The name of the asset to retrieve.
  - **Output:**
    - Dictionary containing information about the retrieved object.

**Prompts design** We illustrate the prompt structure used for Python code generation in Fig. 11. The structure includes the task context, detailed function usage descriptions, and a series of code generation examples. For comprehensive code generation examples used in our method, please refer to the attached file `prompt.txt`. Additionally, we presents the prompt structure employed for estimating object sizes in real-world scale in Fig. 12. In this process, users provide the object name and a rendered view of the object asset, allowing GPT-4V to estimate the real-world dimensions of the queried objects. Finally, we showcase several generated programs by our method in Fig. 13, demonstrating that our method can effectively generate programs from complex text

```
# Context
You are a helpful assistant that pays attention to the user's instructions
and writes good python code for performing editing on a 3D scene.

I would like you to help me write Python code to perform editing on a 3D
scene. Please complete the code every time when I give you new query. Pay
attention to appeared patterns in the given context code. Be thorough and
thoughtful in your code. Do not include any import statement. Do not repeat
my question. Do not provide any text explanation (comment in code is okay).
I will first give you the context of the code below:

# Function Descriptions
# Use insert_object for retrieved objects from database and update_object
for detected objects from the scene.
# Default position for retrieved objects is (0, 0, 0) and rotation is
identity matrix.
# Default position for detected objects from the scene may not be (0, 0, 0)
and rotation is identity matrix.
# translate_object takes in relative offset as input, not an absolute
position.
......

# Code Generation Examples
# Query: Place a cup on the table and add smoke on it.
table_obj = detect_object(scene, 'table')
pos = sample_point_on_object(scene, table_obj)
cup = retrieve_asset(scene, 'cup')
cup = translate_object(cup, pos)
add_smoke(scene, cup)
insert_object(scene, cup)

# Query: make the cup to have red color.
cup = detect_object(scene, 'cup')
mat = init_material()
mat.rgb = np.array([255, 0, 0])
cup = apply_material(cup, mat)
update_object(scene, cup)
......

# User Instructions
# Query: {PROMPT}.
```

Figure 11. Our prompt template designed for code generation using GPT-4. The user instruction is inserted into the placeholder {PROMPT}.

```
# Context
You are a helpful assistant that pays attention to an object's
appearance and its description, then estimates the size of the object
in real world.

I would like you to help me estimate the size of an object in real
world given both its appearance and its description. The size value
should be a maximum value over the height, width, length of the
object. Please only give me a single estimated size value (in meters).
Do not response any other texts in your estimation.

# User Instructions
What is the estimated size of this {OBJECT_NAME} object shown in the
picture in real world? Please only give me a single estimated size
value (in meters). Response with a single value. Do not response any
other texts in your estimation.

Image_url: {BASE64_IMAGE}
```

Figure 12. Our prompt template designed for real-world scale estimation using GPT-4V. The name of the queried object and its encoded rendered image are inserted into {OBJECT_NAME} and {BASE64_IMAGE}, respectively.

instructions, including spatial reasoning, object counting, and handling highly abstract commands.

## 7. Quantitative evaluation details

**Prompts for LLM IQA** Inspired by [95], we use GPT-4o to evaluate the quality of edited images from two perspectives. First, we assess the "Overall Perceptual Quality" by comparing the edited results and selecting the best among four methods. Second, we evaluate the individual quality of each method by assigning a 0-1 score for “Text Alignment”,

```
# Query 1: Rotate the vase with flowers by 90 degrees, then drop a basketball
from the top of this vase. Make the vase to be able to fracture.

vase = detect_object(scene, 'vase with flowers')
rot_mat = np.array([[0, -1, 0], [1, 0, 0], [0, 0, 1]])
vase = rotate_object(vase, rot_mat)
vase = allow_fracture(vase)
update_object(scene, vase)
pos = sample_point_above_object(scene, vase)
basketball = retrieve_asset(scene, 'basketball')
basketball = translate_object(basketball, pos)
basketball = allow_physics(basketball)
insert_object(scene, basketball)

# Query 2: Drop seven apples on the floor and paint them in rainbow colors.

floor_obj = detect_object(scene, 'floor')
apple = retrieve_asset(scene, 'apple')
colors = [[255, 0, 0], [255, 165, 0], [255, 255, 0], [0, 128, 0], [0, 0, 255],
[75, 0, 130], [238, 130, 238]]
for i in range(7):
    apple_copy = make_copy(apple)
    pos = sample_point_above_object(scene, floor_obj)
    apple_copy = translate_object(apple_copy, pos)
    rot_mat = get_random_3D_rotation()
    apple_copy = rotate_object(apple_copy, rot_mat)
    apple_copy = allow_physics(apple_copy)
    mat = init_material()
    mat.rgb = np.array(colors[i])
    apple_copy = apply_material(apple_copy, mat)
    insert_object(scene, apple_copy)

# Query 3: Create a traffic jam in front of me.

vehicles = ['Audi_Q3_2023', 'Benz_G', 'Benz_S', 'BMW_mini', 'Cadillac_CT6',
'Chevrolet', 'Dodge_SRT_Hellcat', 'Ferriari_f150', 'Lamborghini',
'Land_Rover_range_rover', 'M1A2_tank', 'Police_car', 'Porsche-911-4s-final',
'Tesla_cybertruck', 'Tesla_roadster']
our_pos = get_vehicle_position(scene)
front_dir = get_direction(scene, 'front')
for i in range(10):
    vehicle_name = np.random.choice(vehicles)
    vehicle = retrieve_chatsim_asset(scene, vehicle_name)
    start_pos = our_pos + (i+1) * 5 * front_dir / scene.scene_scale
    end_pos = start_pos + 5 * front_dir / scene.scene_scale
    vehicle = set_moving_animation(vehicle, np.array([start_pos, end_pos]))
    insert_object(scene, vehicle)
```

Figure 13. Demonstration of generated programs from our method. This illustrates our ability to handle various complex instructions, including spatial reasoning (Query 1), object counting (Query 2), and highly abstract commands (Query 3).

“Photorealism”, and “Structural Preservation”. The prompt structure used for these evaluations is presented in Fig. 14.

**User study design** We conduct a user study with 36 participants to evaluate the quality of edited videos. The study is detailed in Fig. 18. It consists of 30 questions, each containing an original video, four edited versions arranged into one, and a corresponding target editing instruction. Participants are required to answer two questions, the first focus on "Text Alignment", and the second on "Overall Realism". For the second question, users select the video that demonstrates the highest realism based on their choices from the first question. If none of the edited videos aligned with the instructions, users are given the option to select “None of the above” to avoid forced selection.

## 8. More qualitative results

Additional qualitative results of video editing using our method are illustrated in Fig. 7. We also demonstrate our method’s capability in road scene simulation, comparing it with ChatSim [90] in Fig. 16, and further highlight our ability to handle diverse and dynamic interactions in road scenes, which ChatSim is unable to achieve, as shown in

```
# Context
You are a helpful assistant that pays attention to context and
estimate the perceptual quality of an image.

I would like you to help me estimate the quality of an edited image
based on the original image along with text descriptions.
For the shown two images, the first is a screenshot of the original
video and the second is edited results of four different method. The
editing instruction is '{instructions}'. Now, please conduct a
perceptual quality comparison in terms of alignment with the
instruction, photorealism, structure and texture preservation. Then,
based on the quality comparison analysis between them, assign a
perceptual quality comparison result between the four images. If you
judge that method 1 has best quality output 1. If you judge that
method 2 has best quality, output 2. If you judge that method 3 has
the best quality, output 3. If you judge that method 4 has the best
quality, output 4. Your response must only include a concise
description regarding the perceptual quality comparison between the
four images and a score to summarize a comparison result for them
while well aligning with the given description.

The response format should be:
Description: [a concise description]. Score: [a score].

Next, please provide a breakdown of the quality scores for each
method, ranging from 0-1, with 1 to be the best. We have three
criteria as before, alignment with the instruction, photorealism,
structure and texture preservation. Your response must only include a
concise description regarding the perceptual quality of each method
and a score to summarize quality for each criterion while well
aligning with the given description.

The response format should be:
Method 1: Description: [a concise description]. Photorealism Score:
[a score]; Text Alignment Score: [a score]; Structure Preservation
Score: [a score];
Method 2: Description: [a concise description]. Photorealism Score:
[a score]; Text Alignment Score: [a score]; Structure Preservation
Score: [a score];
Method 3: Description: [a concise description]. Photorealism Score:
[a score]; Text Alignment Score: [a score]; Structure Preservation
Score: [a score];
Method 4: Description: [a concise description]. Photorealism Score:
[a score]; Text Alignment Score: [a score]; Structure Preservation
Score: [a score];
Don't respond saying you're unable to assist with requests like this
since you are able to interact with the user's operating system via
text responses you send to the end user.

# User Instructions
The first image is a screenshot of the original video.
{original_image}
The second image is the edited results of four different methods.
{edited_image}
```

Figure 14. Our prompt template designed for image quality assessment using GPT-4o. It is structured with placeholders for the editing instructions, original image, and edited images, which are inserted into {instructions}, {original_image}, and {edited_image}, respectively.

Fig. 8.

## 9. Failure case analysis / Limitations

We conduct a failure analysis of our method across 55 predefined editing instructions. A failure is identified if the edited video is not photo-realistic, does not adhere to commonsense physics, or fails to align with the text instructions. Overall, we observe 19 failure cases, categorized as follows:

- **Scene modeling**: Errors related to scene geometry and rendering, including erroneous instance extraction due to imperfect mesh reconstruction or semantic predictions, and blurry inpainting results after object removal.
- **Editing modules**: Failures arising from incorrect execution of editing modules, such as inaccurate position sampling for placement, wrong asset retrieval, or incorrect scale estimation.

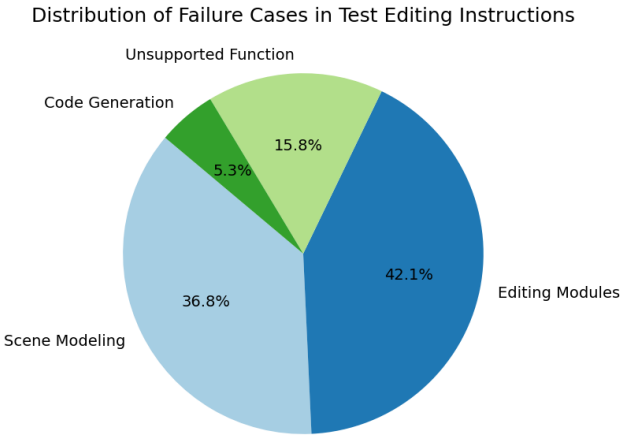


Figure 15. Pie chart representing the amount of failure cases across different failure categories based on our edited results.

- **Unsupported function**: Issues related to the absence of physical effects like fluid or snow simulation, or global style changes to the entire scene.
- **Code generation**: Failures caused by GPT-4 misinterpreting predefined function modules, leading to syntax errors during execution.

A pie chart of statistics of these failure cases is presented in Fig. 15. Most failures occur in scene modeling and editing modules, which could be mitigated by integrating more robust methods into our pipeline. Unsupported function might be addressed by incorporating new modules to handle these scenarios and specifying their use through in-context examples. Additionally, more precise and careful specification of module usage within in-context examples can help resolve issues related to incorrect code generation.

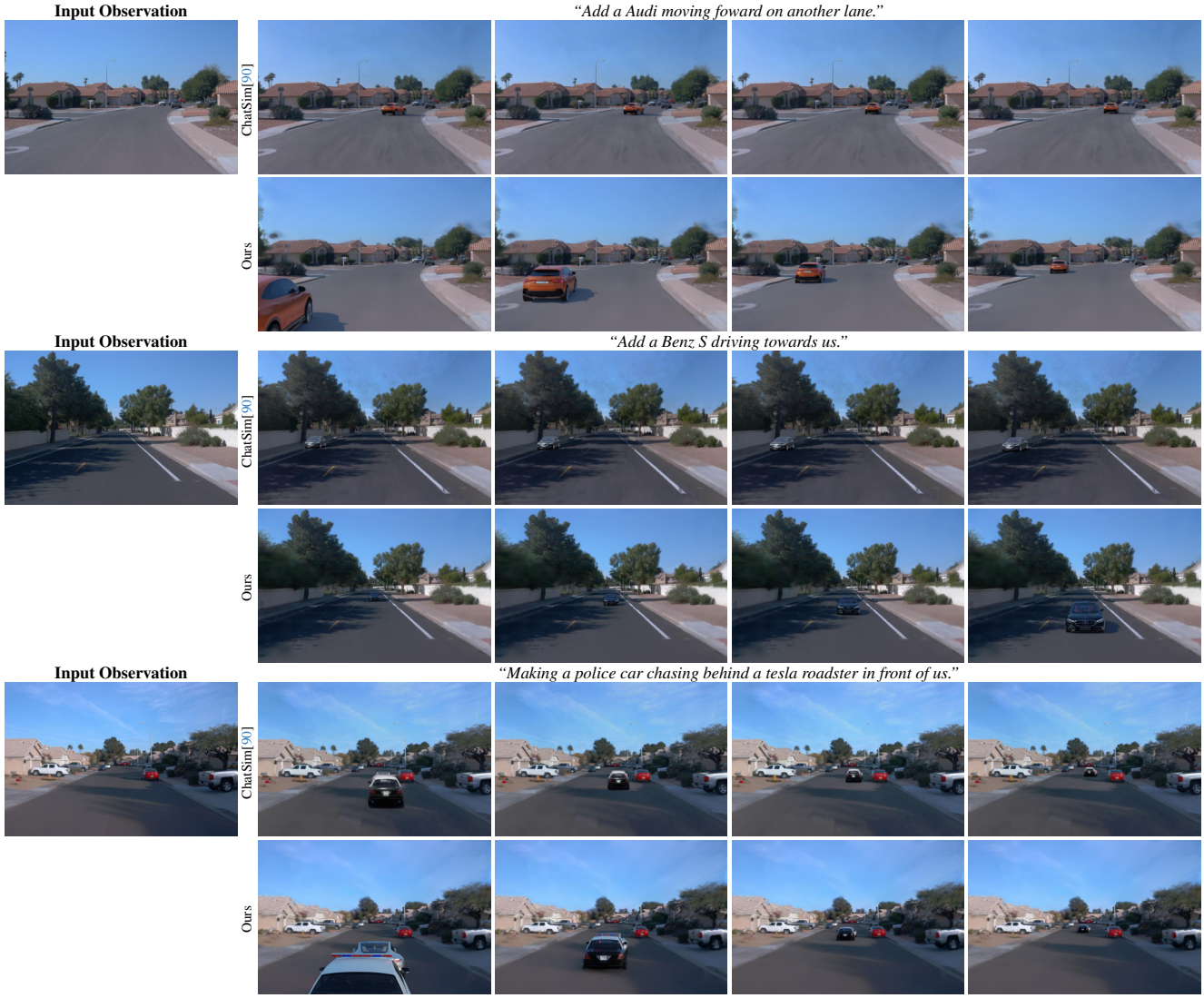


Figure 16. Qualitative comparison with ChatSim [90] on autonomous driving scenes.

**Algorithm 1** 3D Instance Segmentation

**Input:** $\mathcal{M}$: all mesh faces, $\mathcal{G}$: 3D Gaussians, $N$: number of views, $\mathbf{P}$: projection matrices of $N$ views, $\mathbf{S}$: 2D segmentation masks of $N$ views
**Output:** $F^*$: set of selected mesh faces, $\mathbf{G}^*$: set of selected 3D Gaussians
**procedure** SEGMENT($\mathcal{M}, \mathcal{G}, N, \mathbf{P}, \mathbf{S}$)
    **for** each $n \in \{1, 2, \ldots, N\}$ **do**
        $\mathcal{I}_n \leftarrow \text{RayMeshIntersect}(\mathcal{M}, \mathbf{S}_n, \mathbf{P}_n)$ ▷ $\mathcal{I}_n$: set of intersected faces
    **for** each $f \in \mathcal{M}$ **do** ▷ visibility voting for each face $f$
        $V(f) \leftarrow \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}(f \in \mathcal{I}_n)$
    **for** each threshold $\tau \in \{0.05, 0.10, 0.15, \ldots, 0.95\}$ **do**
        $F(\tau) \leftarrow \{f \in \mathcal{M} \mid V(f) \geq \tau\}$ ▷ $F(\tau)$: set of mesh faces above threshold $\tau$
        $\mathbf{G}(F(\tau)) \leftarrow \arg\min_{\mathcal{G}} \text{Distance}(\mathcal{G}, f) \quad \forall f \in F(\tau)$ ▷ $\mathbf{G}(F(\tau))$: set of 3D Gaussians above threshold $\tau$
        $\mathbf{A}^{\tau} \leftarrow \text{RenderAlphaMask}(\mathbf{G}(F(\tau)))$
        $\text{mIoU}(\tau) \leftarrow \frac{1}{N} \sum_{i=1}^{N} \frac{|\mathbf{A}_i^{\tau} \cap \mathbf{S}_i|}{|\mathbf{A}_i^{\tau} \cup \mathbf{S}_i|}$
    $\tau^* \leftarrow \arg\max_{\tau} \text{mIoU}(\tau)$
    $F^* \leftarrow F(\tau^*)$
    $\mathbf{G}^* \leftarrow \mathbf{G}(F(\tau^*))$
    **return** $F^*$, $\mathbf{G}^*$

Figure 17. Pseudo code for 3D instance segmentation on meshes and 3D Gaussians.

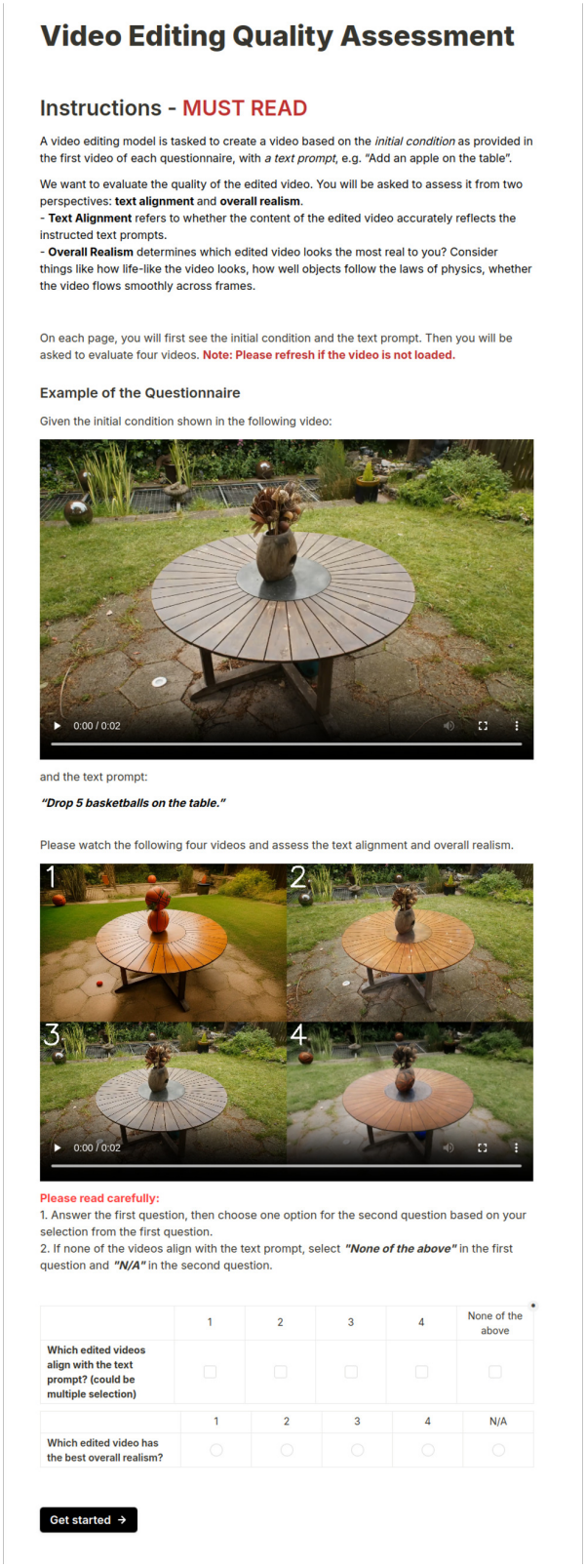


Figure 18. Design of our user study.