

AutoVFX: Physically Realistic Video Editing from Natural Language Instructions

Supplementary Material

1. 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.

1.1. Holistic overview

We use Blender’s modules [4] 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 [12] or Taichi for simulation [9–11].

1.2. Scene modeling details

Geometry We employ BakedSDF [27], implemented in SDFStudio [29], to obtain high-quality scene geometry due to its detailed mesh extraction. Specifically, we use *baked sdf-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.monocular-normal-loss-mult=0.1*. Monocular normal maps are obtained from Omnidata [6]. For fully-captured indoor scenes such as ScanNet++ [28], 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 [8] for geometry reconstruction in road scenes from the Waymo dataset [20]. 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 [22], along with monocular normal and depth priors from Omnidata, and sky masks extracted using SegFormer [25].

Appearance & Semantics To model appearance, we use both 3D Gaussian Splatting [14] and SuGaR [7]. 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 [18] 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 [26] to prevent the emergence of spiky Gaussians during training. Additionally, we incorporate normal regularization from GaussianShader [13] 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 [2] threshold to 0.45 in DEVA [3]. Full pseudo code for 3D instance segmentation on both meshes and 3D Gaussians are illustrated in Fig. 11.

Lighting To illuminate the scene with surrounding light, we extract an HDR environmental map from a single image using DiffusionLight [15]. 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++ [28], 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 [20], 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. 3.

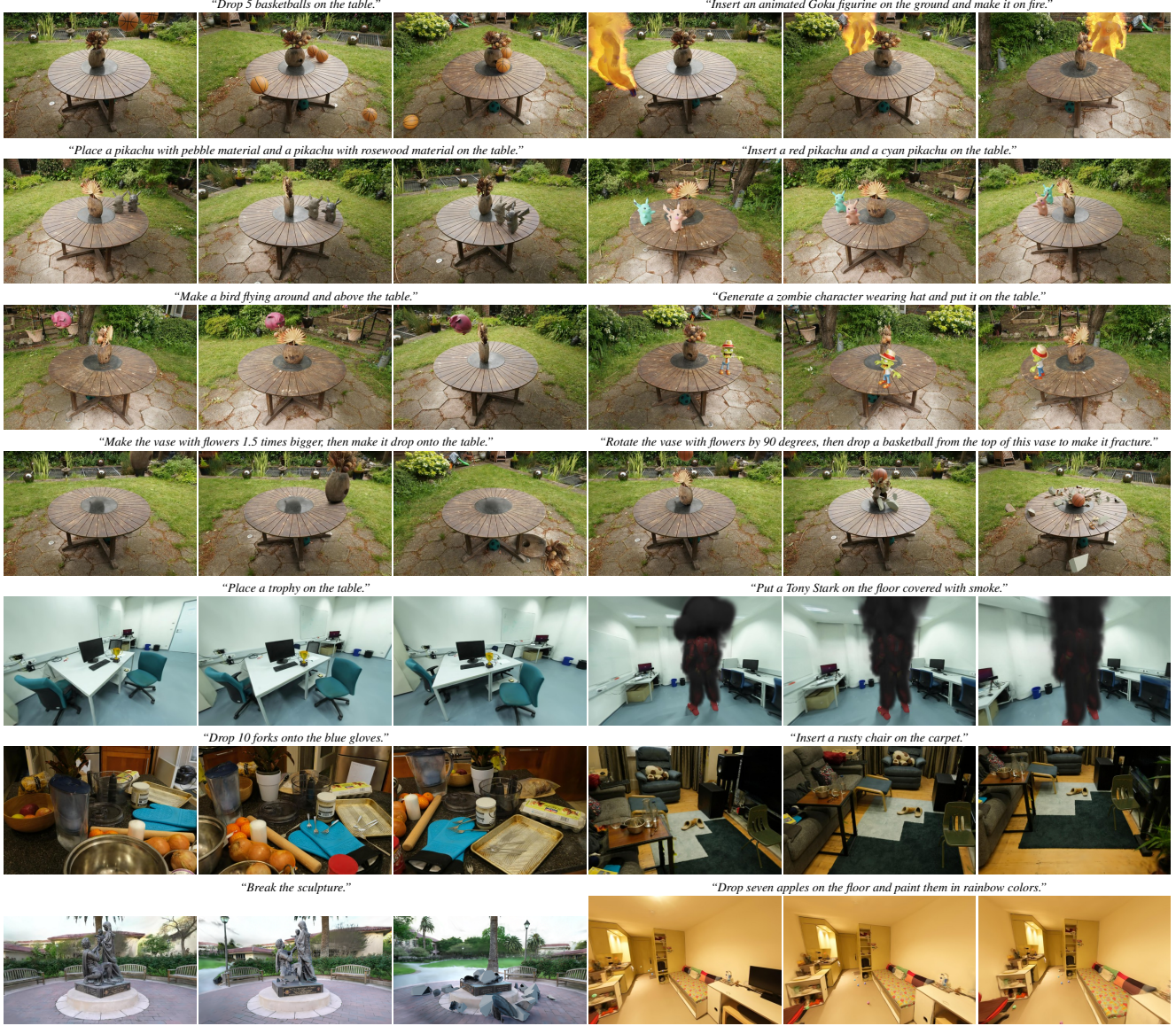


Figure 1. More editing results using AutoVFX.

1.3. Scene simulation details

Animation and rigidbody simulation To simulate the movement of animated objects along a series of 3D key-points, we use Bézier curves [16] 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 [5]. 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 [17, 23, 26], 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 [19] to generate self-fracturing objects. We configure the fracture count to 100 and apply the object’s average color to the internal fractures.

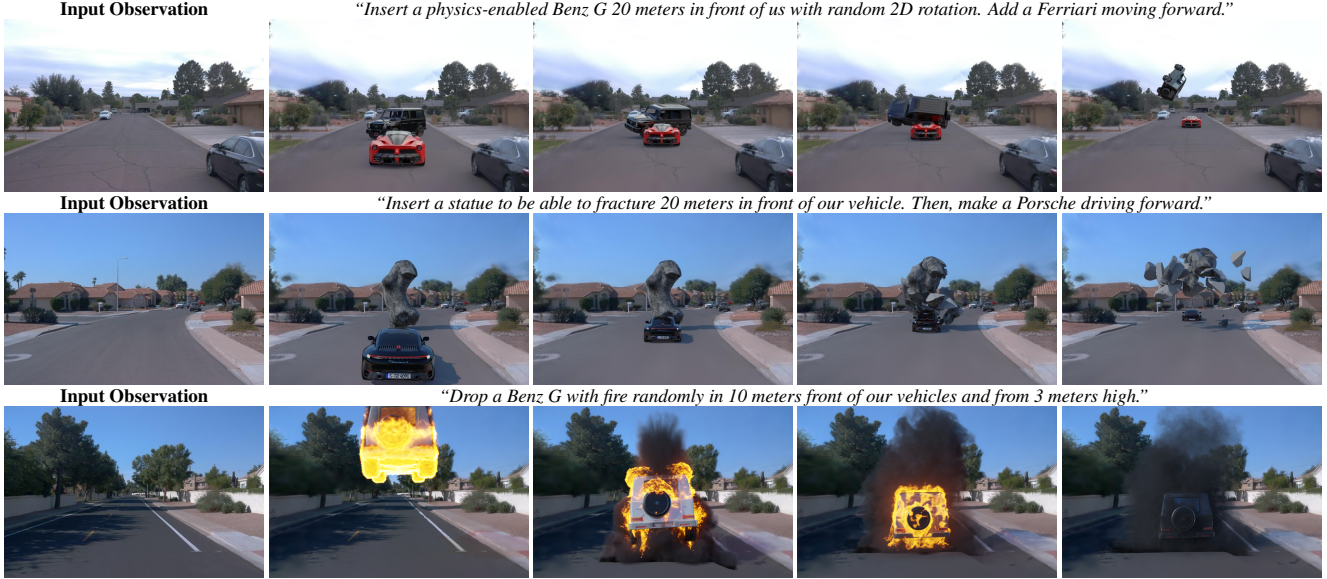


Figure 2. More dynamic simulation results of AutoVFX on autonomous driving scenes.

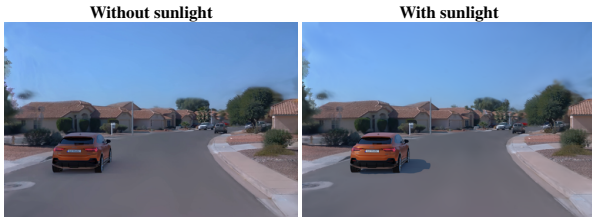


Figure 3. Comparison of simulation results with and without sunlight in Waymo scenes.

For particle effects, we adopt Blender’s computational fluid dynamics addon Mantaflow [21], which is an efficient implementation of the FLIP-based [1] 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.

1.4. 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. 4.

1.5. 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

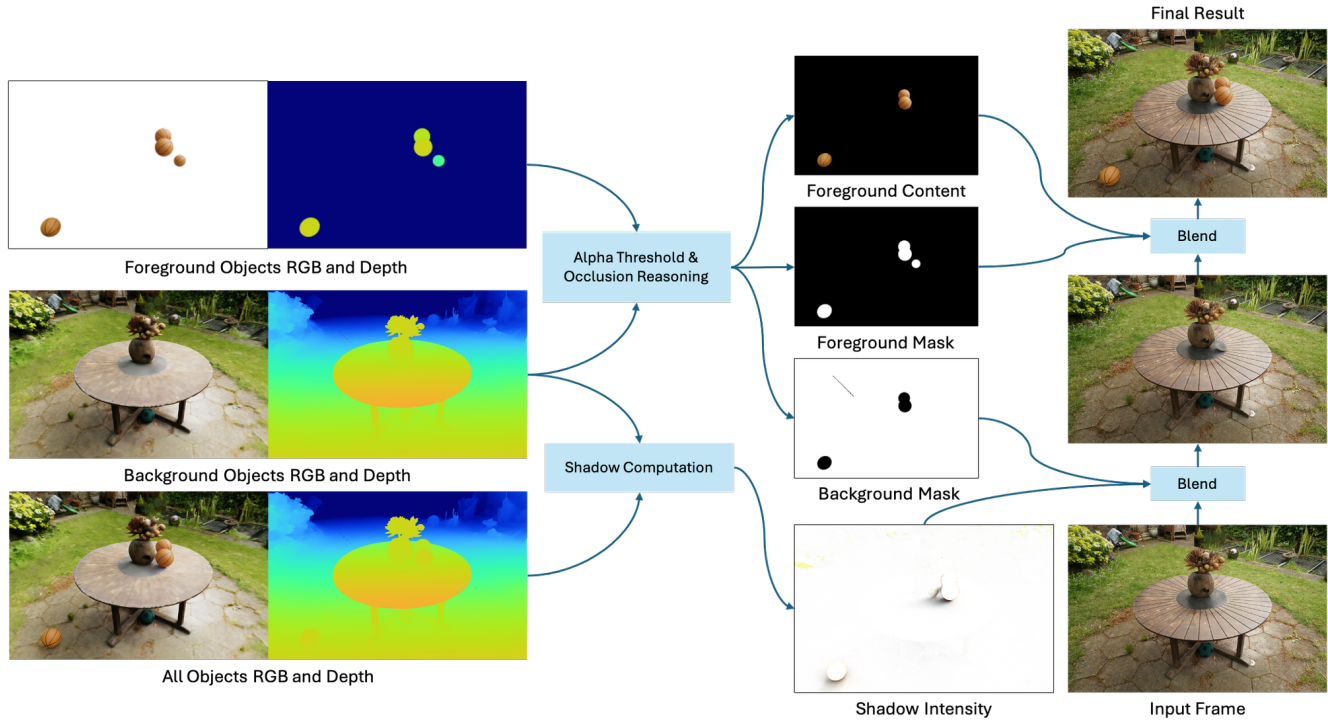


Figure 4. Our image composition pipeline. The process starts by generating foreground and background masks, along with foreground content, through alpha thresholding and occlusion reasoning based on rendered objects and background meshes. Next, shadow intensity is calculated by determining the ratio 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.


```

# 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 5. 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: data:image/png;base64,{BASE64_IMAGE}

```

Figure 6. Our prompt template designed for real-world scale estimation using GPT-4V. The name of the queried object and its encoded rendering image are inserted into {OBJECT_NAME} and {BASE64_IMAGE}, respectively.

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

2. Quantitative evaluation details

Prompts for LLM IQA Inspired by [24], 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', 'Ferrari_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 7. 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. 8.

User study design We conduct a user study with 36 participants to evaluate the quality of edited videos. The study is detailed in Fig. 12. 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.

3. More qualitative results

Additional qualitative results of video editing using our method are illustrated in Fig. 1. We also demonstrate our method’s capability in road scene simulation, comparing it with ChatSim [22] in Fig. 10, and further highlight our ability to handle diverse and dynamic interactions in road scenes, which ChatSim is unable to achieve, as shown in Fig. 2.

```

# 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 8. 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.

4. 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.
- **Unsupported function:** Issues related to the absence of physical effects like fluid or snow simulation, or global

Distribution of Failure Cases in Test Editing Instructions

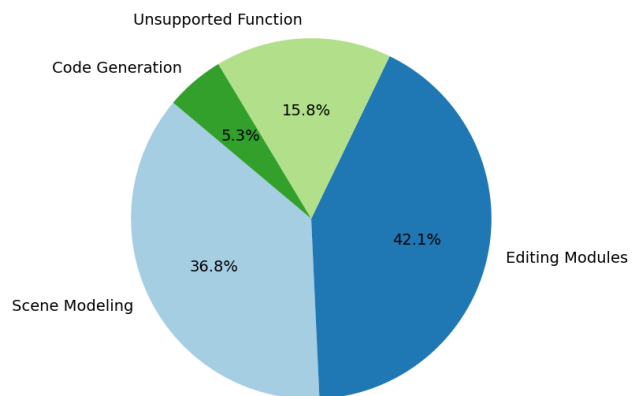


Figure 9. Pie chart representing the amount of failure cases across different failure categories based on our edited results.

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. 9. 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.

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.

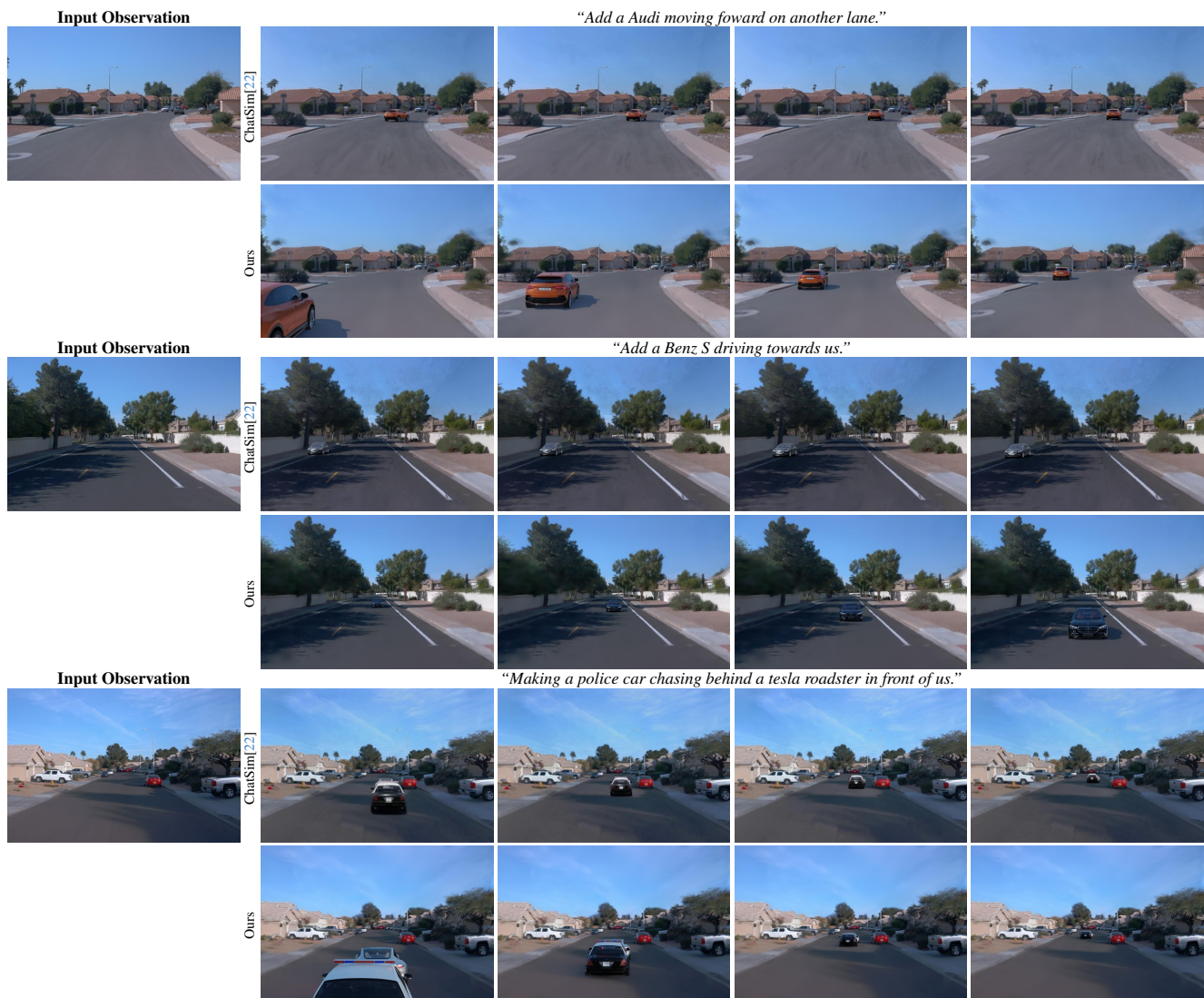


Figure 10. Qualitative comparison with ChatSim [22] on autonomous driving scenes.

Algorithm 1 3D Instance Segmentation

```
1: 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
2: Output:  $F^*$ : set of selected mesh faces,  $\mathbf{G}^*$ : set of selected 3D Gaussians
3: procedure SEGMENT( $\mathcal{M}, \mathcal{G}, N, \mathbf{P}, \mathbf{S}$ )
4:   for each  $n \in \{1, 2, \dots, N\}$  do
5:      $\mathcal{I}_n \leftarrow \text{RayMeshIntersect}(\mathcal{M}, \mathbf{S}_n, \mathbf{P}_n)$   $\triangleright \mathcal{I}_n$ : set of intersected faces
6:   for each  $f \in \mathcal{M}$  do  $\triangleright$  visibility voting for each face  $f$ 
7:      $V(f) \leftarrow \frac{1}{N} \sum_{n=1}^N \mathbb{I}(f \in \mathcal{I}_n)$ 
8:   for each threshold  $\tau \in \{0.05, 0.10, 0.15, \dots, 0.95\}$  do
9:      $F(\tau) \leftarrow \{f \in \mathcal{M} \mid V(f) \geq \tau\}$   $\triangleright F(\tau)$ : set of mesh faces above threshold  $\tau$ 
10:     $\mathbf{G}(F(\tau)) \leftarrow \arg \min_{\mathcal{G}} \text{Distance}(\mathcal{G}, f) \quad \forall f \in F(\tau)$   $\triangleright \mathbf{G}(F(\tau))$ : set of 3D Gaussians above threshold  $\tau$ 
11:     $\mathbf{A}^\tau \leftarrow \text{RenderAlphaMask}(\mathbf{G}(F(\tau)))$ 
12:     $\text{mIoU}(\tau) \leftarrow \frac{1}{N} \sum_{i=1}^N \frac{|\mathbf{A}_i^\tau \cap \mathbf{S}_i|}{|\mathbf{A}_i^\tau \cup \mathbf{S}_i|}$ 
13:     $\tau^* \leftarrow \arg \max_{\tau} \text{mIoU}(\tau)$ 
14:     $F^* \leftarrow F(\tau^*)$ 
15:     $\mathbf{G}^* \leftarrow \mathbf{G}(F(\tau^*))$ 
16: return  $F^*, \mathbf{G}^*$ 
```

Figure 11. Pseudo code for 3D instance segmentation on meshes and 3D Gaussians.

Video Editing Quality Assessment

Instructions - MUST READ

A video editing model is tasked to create a video based on the *initial condition* as provided in the first video of each questionnaire, with a *text prompt*, e.g. "Add an apple on the table".

We want to evaluate the quality of the edited video. You will be asked to assess it from two perspectives: **text alignment** and **overall realism**.

- **Text Alignment** refers to whether the content of the edited video accurately reflects the instructed text prompts.
- **Overall Realism** determines which edited video looks the most real to you? Consider things like how life-like the video looks, how well objects follow the laws of physics, whether the video flows smoothly across frames.

On each page, you will first see the initial condition and the text prompt. Then you will be asked to evaluate four videos. **Note: Please refresh if the video is not loaded.**

Example of the Questionnaire

Given the initial condition shown in the following video:



and the text prompt:

"Drop 5 basketballs on the table."

Please watch the following four videos and assess the text alignment and overall realism.



Please read carefully:

1. Answer the first question, then choose one option for the second question based on your selection from the first question.
2. If none of the videos align with the text prompt, select **"None of the above"** in the first question and **"N/A"** in the second question.

	1	2	3	4	None of the above
Which edited videos align with the text prompt? (could be multiple selection)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	1	2	3	4	N/A
Which edited video has the best overall realism?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Get started →

Figure 12. Design of our user study.

References

- [1] 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**
- [2] 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**
- [3] 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. **1**
- [4] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. **1**
- [5] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, 2015. **2**
- [6] Ainaz Eftekhari, 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**
- [7] 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. **1**
- [8] 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**
- [9] 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**
- [10] 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.
- [11] 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**
- [12] 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**
- [13] 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**
- [14] 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. **1**
- [15] 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. **1**
- [16] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-spline techniques*. Springer, 2002. **2**
- [17] 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. **2**
- [18] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. **1**
- [19] phymec Sergey Sharybin, ideasman42. Cell Fracture, 2024. <https://extensions.blender.org/add-ons/cell-fracture/>. **2**
- [20] Pei Sun, Henrik Kretschmar, 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. **1**
- [21] Nils Thuerey and Tobias Pfaff. MantaFlow, 2018. <http://mantaflow.com>. **3**
- [22] 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. **1, 7, 9**
- [23] 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**
- [24] 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**
- [25] 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**
- [26] 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. **1, 2**
- [27] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P Srinivasan, Richard Szeliski, Jonathan T Barron, and Ben Mildenhall. Bakedsd: Meshing neural sdfs for real-time view synthesis. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–9, 2023. **1**
- [28] 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. [1](#)

- [29] 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](#)