CVPR
#

Video2Game: Real-time, Interactive, Realistic and Browser-Compatible Environment from a Single Video

Anonymous CVPR submission

Paper ID

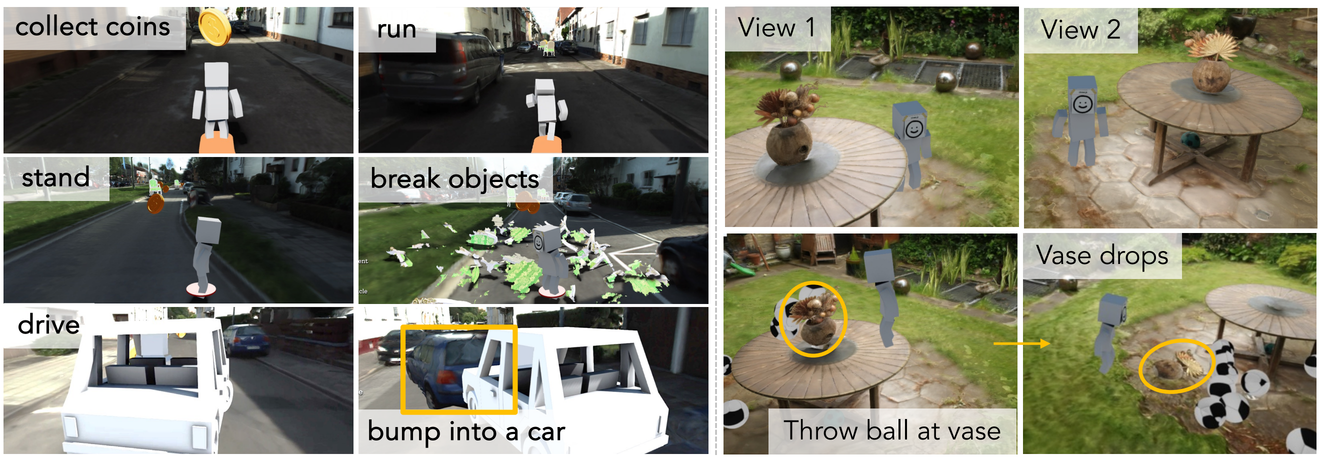

Figure 1. **Video2Game** takes an input video of an arbitrary scene and automatically transforms it into a real-time, interactive, realistic and browser-compatible environment. The users can freely explore the environment and interact with the objects in the scene.

Abstract

Creating high-quality and interactive virtual environments, such as games and simulators, often involves complex and costly manual modeling processes. In this paper, we present Video2Game, a novel approach that automatically converts videos of real-world scenes into realistic and interactive game environments. At the heart of our system are three core components: (i) a neural radiance fields (NeRF) module that effectively captures the geometry and visual appearance of the scene; (ii) a mesh module that distills the knowledge from NeRF for faster rendering; and (iii) a physics module that models the interactions and physical dynamics among the objects. By following the carefully designed pipeline, one can construct an interactable and actionable digital replica of the real world. We benchmark our system on both indoor and large-scale outdoor scenes. We show that we can not only produce highly-realistic renderings in real-time, but also build interactive games on top.

1. Introduction

Crafting a visually compelling and interactive environment is crucial for immersive experiences in various domains, such as video games, virtual reality applications, and self-driving simulators. This process, however, is complex and expensive. It demands the skills of experts in the field and the use of professional software development tools [21, 24]. For instance, Grand Theft Auto V [23], known for its intricately detailed environment, was one of the most expensive video games ever developed, with a budget over $265 million primarily for asset creation. Similarly, the development of the CARLA autonomous driving simulator [19] involves a multidisciplinary team of 3D artists, programmers, and engineers to meticulously craft and texture the virtual cityscapes, creating its lifelike environments.

An appealing alternative to extensive manual modelling is creating environments directly from the real world. For instance, photogrammetry, a technique for constructing digital replicas of objects or scenes from overlapping real-world photographs, has been utilized for environment creation [52, 53]. Success stories also span various games and simulators. However, most use cases are limited to creating object assets and necessitate significant post-processing, such as material creation, texturing, and geometry fixes [66]. People thus turns to neural radiance fields (NeRFs) [46], as it offers a more promising approach to

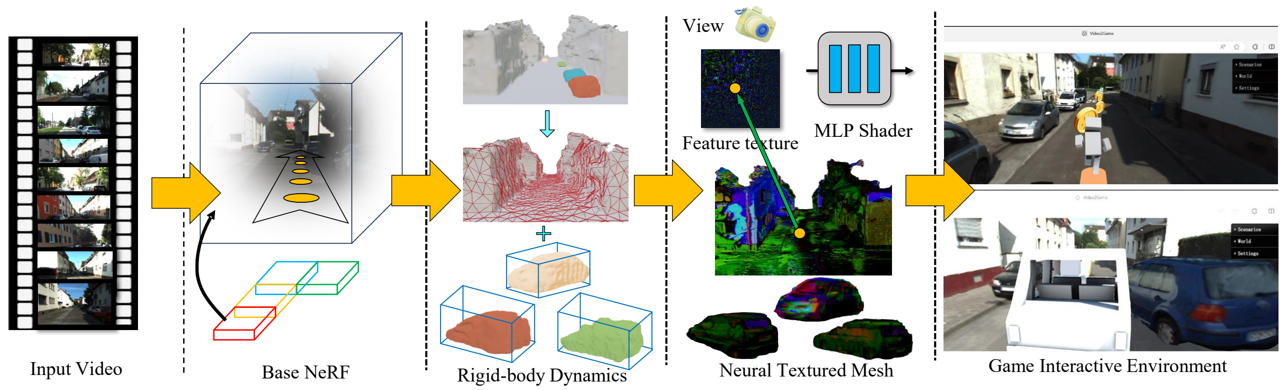

Figure 2. **Overview of Video2Game:** Given multiple posed images from a single video as input, we first construct a large-scale NeRF model that is realistic and possesses high-quality surface geometry. We then transform this NeRF model into a mesh representation with corresponding rigid-body dynamics to enable interactions. We utilize UV-mapped neural texture, which is both expressive and compatible with game engines. Finally, we obtain an interactive virtual environment that virtual actors can interact with, can respond to user control, and deliver high-resolution rendering from novel camera perspectives – all in real-time.

modeling large scenes. With careful design [13, 22, 39, 48, 61], NeRF is able to render free-viewpoint, photo-realistic images efficiently. However, crafting an interactive environment entails more than just creating a visually high-fidelity digital twin; it also involves building a **physically plausible**, **immersive**, **real-time** and importantly, **interactive** world tailored to user experiences. Furthermore, we expect such a virtual world to be **compatible with real-time interaction interfaces** such as common game engines. Despite its promise, the use of NeRF to create interactive environments from real-world videos remains largely unexplored.

In this paper, we introduce Video2Game, a novel approach to automatically converting a video of a scene into a realistic and interactive virtual environment. Given a video as input, we first construct a NeRF that can effectively capture the geometric and visual information of a large-scale, unbounded) scene. Then we distill the NeRF into a game engine-compatible, neural textured mesh. This significantly improves the rendering efficiency while maintains the overall quality. To model the interactions among the objects, we further decompose the scene into individual actionable entities and equip them with respective physics model. Finally, we import our automatically generated assets into a WebGL-based game engine and create a playable game. The resulting virtual environment is photo-realistic, interactive, and runs in real-time. See Fig. 1 for demonstration. In summary, our key contributions are:

- A novel 3D modeling algorithm for real-time, free-viewpoint rendering and physical simulation, surpassing state-of-the-art NeRF baking methods with added rigid-body physics for enhanced simulation.
- An automated game-creation framework to transform a scene video into an interactive, realistic environment, compatible with current game engines.

2. Related Works

Given a single video, we aim to create a real-time, interactive game where the agents (*e.g.*, the character, the car) can navigate and explore the reconstructed digital world, interact with objects in the scene (*e.g.*, collision and manipulate objects), and achieve their respective tasks (*e.g.*, collecting coins, shooting targets). We draw inspirations from multiple areas and combine the best of all. In this section, we will briefly review those closely related areas which forms the foundation of our work.

Novel view synthesis (NVS): Our work builds upon the success of novel view synthesis [14, 25, 35, 62], which is crucial for our game since it enables the agents to move freely and view the reconstructed world seamlessly from various perspectives. Among all these approaches [26, 60, 68, 85, 86], we exploit neural radiance field (NeRF) [46] as our underlying representation. NeRF has emerged as one of the most promising tools in NVS since its introduction [49–51], and has great performance across a wide range of scenarios [36, 56, 75, 81]. For instance, it can be easily extended to handle various challenging real-world scenarios such as learning from noisy camera poses [38, 70], reflectance modeling for photo-realistic relighting [69, 83], and real-time rendering [16, 39, 55, 65, 76]. In this work, we combine recent advances in NeRF with *physics modeling* to build an immersive digital world from one single video, moving from *passive* NVS to our complete solution for *embodied* world modeling where agents can *actively* explore and interact with the scene.

Controllable video generation: Using different control signals to manipulate the output of a visual model has garnered great interest in the community. This has had a profound impact on content creation [57, 58], digital editing [11, 34], and simulation [30, 31, 40]. One could also leverage large foundation models to control video content using text [57, 58]. However, they lack fine-grained and real-time control over the generated content. Alternatively, training (conditional) generative models for each scene enables better disentanglement of dynamics (*e.g.*, foreground *vs.* background) and supports better control. For instance, one can represent a self-driving scene [31] or a Pacman game [30] as latent codes and generate video frames based on control inputs with a neural network. One can also learn to control the players within tennis games [44, 45, 79, 80]. Our work falls under the second line of research, where the model takes user control signals (*e.g.*, a keystroke from the keyboard) as input and responds by rendering a new scene. However, instead of focusing on a specific scene (*e.g.*, tennis games), we have developed a pipeline that allows the creation of a playable environment from a single video of a generic scene. Additionally, we model everything in 3D, which enables us to effectively capture not only view-dependent appearance but also physical interactions among rigid-body equipped objects. Importantly, we adopt a neural representation compatible with graphics engines, enabling users to play the entire game in their browser at an interactive rate.

Data-driven simulation: Building a realistic simulation environment has been a longstanding challenge. [19, 29, 67, 71]. While it's promising, we come close to mirror the real world only in recent years [10, 15, 42, 43, 59, 74, 75]. The key insight of these work is to build models by leveraging real-world data. Our work closely aligns with this line of research on building high-fidelity simulators from real-world data, with a few key differences. First, existing works mainly focus on *offline* training and evaluation [10, 43, 74, 75], whereas our system runs at an interactive rate and allows for *online*, *real-time* control. Second, some existing works[41, 43, 72, 87] need additional data modality like LiDAR point clouds for geometry reconstruction, but RGB video is all we need. Third, most photo-realistic simulators don't model physical interactions. However, we supports various physics modeling and allows agents to interact with the environment. Last, existing simulators are typically resource-intensive , while our system is lightweight and can be easily accessible in common engines.

3. Video2Game

Given a sequence of images or a video of a scene, our goal is to construct an *interactable* and *actionable* digital twin, upon which we can build real-time, interactive games or realistic (sensor) simulators. Based on the observations that prevalent approaches to constructing digital replica mainly focus on visual appearance and ignore the underlying physical interactions, we carefully design our system such that it can not only produce high-quality rendering across viewpoints, but also support the modeling of physical actions (*e.g.*, navigation, collision, manipulation, etc). At the heart of our systems is a compositional implicit-explicit 3D representation that is effective and efficient for both sensor and physics simulation. By decomposing the world into individual entities, we can better model and manipulate their physical properties (*e.g.*, specularity, mass, friction), and simulate the outcomes of interactions more effectively.

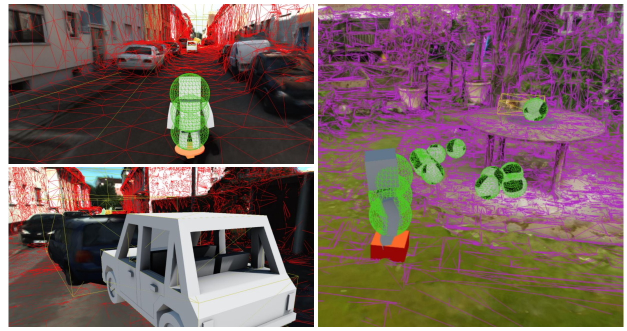

Figure 3. **Visualization of automatically computed collision geometry:** Sphere collider (green), box collider (yellow), convex polygon collider (purple) and trimesh collider (red).

We start by introducing a NeRF model that can effectively capture the geometric and visual information of a large-scale, unbounded scene (Sec. 3.1). Next, we present an approach to convert the NeRF into a game-engine compatible mesh with neural texture maps, significantly improving the rendering efficiency while maintaining the quality (Sec. 3.2). To enable physical interactions, we further decompose the scene into individual actionable entities and equip them with respective physics models (Sec. 3.3). Finally, we describe how we integrate our interactive environment into a WebGL-based game engine, allowing users to play and interact with the virtual world in real time on their personal browser. Fig. 2 provides an overview of our proposed framework.

3.1. Large-scale NeRF

Preliminaries: Instant-NGP [48] is a notable variant of NeRF, which represents the radiance field with a combination of spatial hash-based voxels and neural networks: $\mathbf{c}, \sigma = F_\theta(\mathbf{x}, \mathbf{d}; \Phi) = \texttt{MLP}_\theta(\texttt{It}(\mathbf{x}, \Phi), \mathbf{d})$. Given a 3D point $\mathbf{x} \in \mathbb{R}^3$ and a camera direction $\mathbf{d} \in \mathbb{R}^2$ as input, Instant-NGP first interpolate the point feature $\texttt{It}(\mathbf{x}, \Phi)$ from the adjacent voxel features Φ. Then the point feature and the camera direction are fed into a light-weight multilayer perception (MLP) to predict the color $\mathbf{c} \in \mathbb{R}^3$ and density $\sigma \in \mathbb{R}^+$. To render the scene appearance, we first cast a ray $\mathbf{r}(t) = \mathbf{o}+t\mathbf{d}$ from the camera center $\mathbf{o}$ through the pixel center in direction $\mathbf{d}$, and sample a set of 3D points $\{\mathbf{x}_i\}$ along the ray. We then query their respective color $\{\mathbf{c}_i\}$ and density $\{\sigma_i\}$ and obtain the color of the pixel through alpha-composition: $\mathbf{C}_{\text{NeRF}}(\mathbf{r}) = \sum_i w_i \mathbf{c}_i$. Similarly, the

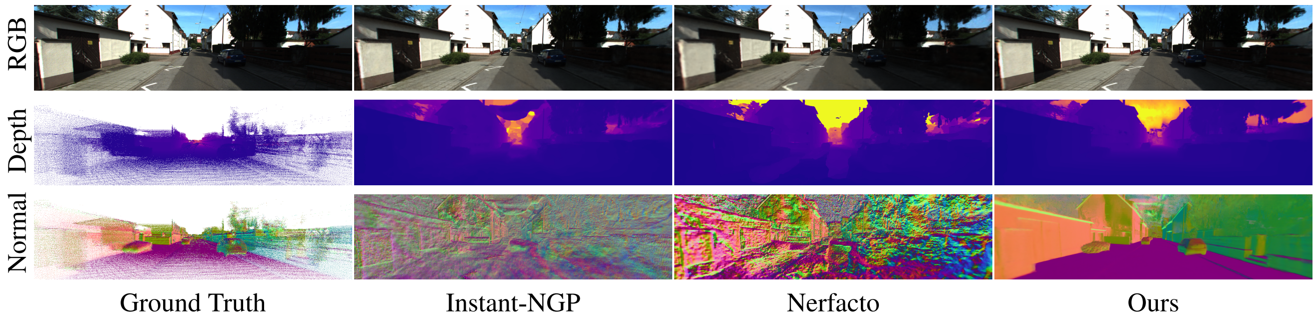

Figure 4. **Qualitative comparisons among NeRF models.** The rendering quality of our base NeRF is superior to baselines, and with leveraging monocular cues, we substantially improve rendered geometry compared to other baselines. This significantly facilitates NeRF baking in subsequent stages. Here we consider depths measured by LiDAR point cloud in KITTI-360 and compute normals based on it.

expected depth can be computed by: $\mathbf{D}_{\text{NeRF}}(\mathbf{r}) = \sum_i w_i t_i$. Here, w_i indicates the blending weight that is derived from the densities $\{\sigma_i\}$. We refer the readers to [46] for more details. To learn the voxel features Φ and the MLP weights θ, we compute the difference between the ground truth color and the rendered color: $\mathcal{L}_{\text{rgb}} = \sum_{\mathbf{r}} \|\mathbf{C}_{\text{GT}}(\mathbf{r}) - \mathbf{C}_{\text{NeRF}}(\mathbf{r})\|_2^2$.

Large-scale NeRF: While Instant-NGP [48] has shown promising results on densely observed and bounded scenes, its performance starts to degrade when extending to sparsely-captured, large-scale, unbounded environments. To mitigate these issues, we propose several enhancements:

$$\mathbf{c}, \sigma, s, \mathbf{n} = F_\theta(\mathbf{x}, \mathbf{d}; \Phi) = \texttt{MLP}_\theta(\texttt{It}(\texttt{Ct}(\mathbf{x}), \Phi), \mathbf{d}). \quad (1)$$

First of all, we exploit the contraction function $\texttt{Ct}(\mathbf{x})$ [12] to map the unbounded coordinates into a bounded region. In addition to radiance and density, we predict the semantics s and the surface normal $\mathbf{n}$ of the 3D points, guided with 2D priors to better regularize the scene geometry. Furthermore, we divide large-scale scenes into several blocks [63] to capture the fine-grained details. We now describe these enhancements in more details.

Depth: High-quality geometry is critical for modeling physical interactions. Inspired by MonoSDF [78], we leverage off-the-shelf monocular depth estimators [20, 27] to guide and improve the underlying geometry. We first predict the depth of the scene from rendered RGB images. Then we minimize the discrepancy between the rendered depth and the predicted depth via $\mathcal{L}_{\text{depth}} = \sum_{\mathbf{r}} \|(a\mathbf{D}_{\text{NeRF}}(\mathbf{r}) + b) - \mathbf{D}_{\text{mono}}(\mathbf{r})\|_2^2$, where a and b are the scale and shift that aligns the two distribution [54].

Surface normals: Similar to depth, we encourage the normal estimated from NeRF to be consistent with the normal predicted by the off-the-shelf estimator [20, 27]. The normal of a 3D point $\mathbf{x}_i$ can be either analytically derived from the estimated density $\mathbf{n}_i = (1 - \frac{\nabla_{\mathbf{x}} \sigma_i}{\|\nabla \sigma_i\|})$, or predicted by the MLP header as in Eq. 1. We could aggregate them via volume render: $\mathbf{N}(\mathbf{r}) = \sum_i \mathbf{w}_i \mathbf{n}_i$. Empirically we find that adopting both normals and promoting their mutual consistency works the best, since the MLP header offers more flexibility. We thus employ $\mathcal{L}_{\text{normal}} = \|\mathbf{N}_{\text{mlp}}(\mathbf{r}) - \mathbf{N}_{\text{mono}}(\mathbf{r})\|_2^2 + \|\mathbf{N}_{\text{mlp}}(\mathbf{r}) - \mathbf{N}_{\text{density}}(\mathbf{r})\|_2^2$.

Semantics: We also predict semantic logits for each sampled 3D points with our MLP. This helps us capture the correlation between semantics and geometry [36, 84]. We render the semantic map with volume rendering $\mathbf{S}_{\text{NeRF}}(\mathbf{r}) = \sum_i \mathbf{w}_i \mathbf{s}_i$ and compute the cross-entropy with that of a 2D segmentation model $\mathcal{L}_{\text{semantics}} = \texttt{CE}\left(\mathbf{S}_{\text{mono}}, \mathbf{S}_{\text{NeRF}}\right)$.

Regularization: We additionally adopt two regularization terms. To reduce floaters in the scene, for each randomly sampled 3D point $\mathbf{x}$, we penalize its density by $\mathcal{L}_{\text{sp}} = \sum 1 - \exp(-\alpha\sigma(\mathbf{x}))$, where $\alpha > 0$ [77]. For each sky pixel (which we derived from the semantic MLP), we encourage its depth $\mathbf{D}_{\text{NeRF}}(\mathbf{r}^{\text{sky}})$ to be as far as possible. The loss is defined as: $\mathcal{L}_{\text{sky}} = \sum_{\mathbf{r}^{\text{sky}}} \exp(-\mathbf{D}_{\text{NeRF}}(\mathbf{r}^{\text{sky}}))$.

Blocking: Capitalizing on a single Instant-NGP to cover an extraordinarily large scene such as KITTI-360 [37] would often lead to inferior results. We thus adopt a strategy akin to BlockNeRF [63] where we divided the whole scene into numerous blocks and model each region with a separate Instant-NGP. Adjacent regions maintain substantial overlaps to ensure smooth transition.

Learning: We jointly optimize the voxel feature Φ and the MLP weights θ by minimizing the following loss:

$$\mathcal{L}_{\text{total}}^{\text{NeRF}} = \mathcal{L}_{\text{rgb}} + \mathcal{L}_{\text{normal}} + \mathcal{L}_{\text{semantics}} + \mathcal{L}_{\text{depth}} + \mathcal{L}_{\text{sky}} + \mathcal{L}_{\text{sp}} \quad (2)$$

3.2. NeRF Baking

We aim to create a digital replica that users (or agents) can freely explore and act upon in real time. Although our large-scale NeRF effectively renders high-quality images and geometry, its efficiency is limited by the computational costs associated with sampling 3D points. The underlying volume density representation further complicates

Method	Representation	KITTI-360			Gardenvase			Interactive Compatibility		
		PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	Real time	Rigid-body physics	Scene decomposition
InstantNGP [48]	Volume	27.46	0.853	0.165	25.90	0.757	0.191	✗	✗	✗
Nerfacto [64]	Volume	23.20	0.763	0.238	22.16	0.517	0.283	✗	✗	✗
Video2Game	Volume	**27.62**	**0.871**	**0.131**	**26.57**	**0.815**	**0.143**	✗	✗	✗
Gauss. Spl. [28]	Points	17.85	0.615	0.428	27.50	0.858	0.099	✓	✗	✗
MobileNeRF [16]	Mesh	19.67	0.627	0.452	22.80	0.505	0.365	✓	✗	✗
BakedSDF* [76]	Mesh	22.37	0.757	0.302	22.68	**0.514**	0.369	✓	✓	✗
Video2Game	Mesh	**23.35**	**0.765**	**0.246**	**22.81**	0.508	**0.363**	✓	✓	✓

Table 1. **Quantitative results on novel view synthesis and interactive compatibility analysis.** Video2Game produces better or comparable results across scenes, suggesting the effectiveness of our NeRF and mesh model. The performance improves the most when tackling unbounded, large-scale scenes in KITTI-360. We note that existing NeRFs cannot reach the interactive rate required for real-time games. While point-based rendering significantly improves the speed, it does not support rigid body physics. BakedSDF [76] represents the whole scene with one single mesh, thus does not support object-level interactions.

the problem. For instance, it's unclear how to define physical interaction with such a representation (*e.g.*, defining collision). The representation is also not compatible with common graphics engines. While recent software, such as the NeRFStudio Blender plugin and LumaAI Unreal add-on, has made some strides, their interaction capabilities and scene geometry quality are still not optimal for real-time user engagement, especially when the scene is large and the observations are relatively sparse. To overcome these challenges, we draw inspiration from recent NeRF meshing advancements and present a novel NeRF baking framework that efficiently transforms our NeRF representation into a game-engine compatible mesh. As we will show in Sec. 4, this conversion greatly enhances rendering efficiency while preserving quality and facilitates physical interactions.

Mesh representation: Our mesh $\mathcal{M} = (\mathbf{V}, \mathbf{F}, \mathbf{T})$ comprises vertices $\mathbf{V} \in \mathbb{R}^{|V|\times 3}$, faces $\mathbf{F} \in \mathbb{N}^{|F|\times 3}$ and a UV neural texture map $\mathbf{T} \in \mathbb{R}^{H\times W\times 6}$. Following [65], we store the base color in the first three dimension of $\mathbf{T}$, and encode the specular feature in the rest. The initial mesh topology are obtained by marching cubes in the NeRF density field. We further prune the invisible faces. conduct mesh decimation and edge length regularization. The UV coordinate of each vertex is calculated via xatlas [7].

Rendering: We leverage differentiable renderers [33] to render our mesh into RGB images $\mathbf{C}_\mathrm{R}$ and depth maps $\mathbf{D}_\mathrm{R}$. Specifically, we first rasterize the mesh into screen space and obtain the UV coordinate for each pixel i. Then we sample the corresponding texture feature $\mathbf{T}_i = [\mathbf{B}_i; \mathbf{S}_i]$ and feed it into our customized shader. Finally, the shader computes the sum of the view-independent base color $\mathbf{B}_i$ and the view-dependent MLP $\mathtt{MLP}_\theta^{\text{shader}}(\mathbf{S}_i, \mathbf{d}_i)$:

$$\mathbf{C}_\mathrm{R} = \mathbf{B}_i + \mathtt{MLP}_\theta^{\text{shader}}(\mathbf{S}_i, \mathbf{d}_i). \tag{3}$$

The MLP is lightweight and can be baked in GLSL.

Learning: We train the shader MLP $\mathtt{MLP}_\theta^{\text{shader}}$ and the neural texture map $\mathbf{T}$ by minimizing the color difference between the mesh and the ground truth, and the geometry difference between the mesh and the NeRF model:

$$\mathcal{L}_{\mathbf{T},\theta}^{\text{mesh}} = \sum_{\mathbf{r}} \|\mathbf{C}_\mathrm{R}(\mathbf{r}) - \mathbf{C}_\mathrm{GT}(\mathbf{r})\| + \|\mathbf{D}_\mathrm{R}(\mathbf{r}) - \mathbf{D}_\mathrm{NeRF}(\mathbf{r})\|. \tag{4}$$

Anti-aliasing: Common differentiable rasterizers only take the center of each pixel into account. This may lead to aliasing in the learned texture map. To resolve this issue, we randomly perturb the optical center of the camera by 0.5 pixels along each axis at every training step. This ensure all the regions within a pixel get rasterized.

Relationship to existing work: Our approach is closely related to recent work on NeRF meshing [16, 55, 65, 76], but there exist key differences. While MobileNeRF [16] also adopts an explicit mesh with neural textures, they mainly capitalize on planar primitives. The quality of the reconstructed mesh is thus inferior. BakedSDF [76] offers excellent runtime and rendering quality, but their vertex coloring approach has limited resolution for large scenes. NeRF2Mesh [65] lacks depth distillation and doesn't adopt contraction space for unbounded scenes. They also have a sophisticated multi-stage training and multi-resolution refinement process. Finally, MeRF [55], though efficient, still relies on volume-rendering.

3.3. Representation for Physical Interaction

Our mesh model facilitates efficient novel-view rendering in real time and allows for basic *rigid-body* physical interactions. For example, the explicit mesh structure permits an agent to "stand" on the ground. Nevertheless, beyond navigation, an agent should be capable of performing various actions including collision and manipulation. Furthermore, a scene comprises not only the background but also interactable foreground objects, each possessing unique physical properties. For instance, a street-bound car is much heavier than a flower vase. When struck by another object, a car may barely move but the vase may fall and shatter.

To enhance physical interaction realism, we decompose the scene into discrete, actionable entities, each endowed with specific physical characteristics (*e.g*., mass, friction). This approach, in conjunction with *rigid-body* physics, allows for the effective simulation that adheres to physical laws.

Scene decomposition: Directly editing and decomposing a mesh is extremely difficult due to topology change. Fortunately, neural fields are inherently compositional in 3D. By identifying the objects each spatial region belongs to, we can use neural fields to guide the decomposition of the mesh. Specifically, we sample a 3D point $\mathbf{x}_i$ within each voxel i and determine its semantic category either through the predicted semantic logits s_i or by verifying whether the point is within a specified bounding box. The process is repeated for all voxels to segment the entire scene. Then, for each object, we perform NeRF meshing individually, setting the density of the remaining areas to zero. The intersections between objects are automatically resolved by marching cube. Finally, we initialize the neural texture of these new, individual meshes from the original mesh model. For newly created faces, we employ nearest neighbor inpainting on the neural texture map, which empirically yields satisfactory results. Fig. 1 shows an example where a vase is separated from a table. The middle of the table is original occluded yet we are able to maintain high-quality rendering.

Physical parameters reasoning: The next step is to equip decomposed individual meshes with various physics-related attributes so that we can effectively model and simulate their physical dynamics. In this work, we focus on rigid body physics, where each entity i is represented by a collision geometry col_i, mass m_i, and friction parameters f_i. We support fours types of collision geometry with different levels of complexity and efficiency: box, sphere, convex polygon, and triangle mesh. Depending on the object and the task of interest, one can select the most suitable collision check for them. For other physical parameters (*e.g*. mass, friction), one can either set them manually or query large language models (LLMs) for an estimation.

Physical interactions: Rigid body dynamics, while simple, can support a variety of interactions. With the collision check, an user/agent can easily navigate through the environment while respecting the geometry of the scene. The agents will no longer be stuck in a road or cut through a wall. It also allows the agent to interact with the objects in the scene. For instance, one can push the objects towards the location of interest. The object movement will be determined by its mass and other physical properties such as the friction. We can also manipulate the objects by adopting a magnet grasper, following AI2-Thor [32]. This opens the avenue towards automatic creation of realistic, interactive virtual environment for robot learning.

3.4. Interactive Environment

We deploy our interactive environment within a real-time, browser-based game engine. We manage the underlying logic and assets using Sketchbook [3], a Game Engine based on Three.js that leverages WebGL [4] for rendering. This combination ensures high efficiency while offering the flexibility and sophistication required for intricate rendering tasks. It also allows us to easily integrate content from different scenes together. We have further extended Sketchbook's capabilities by implementing a GLSL-based shader [2]. This enables real-time computation of our MLP-based specular shader during deployment. For physics simulation, we use Cannon.js [1], which assures realism and efficiency in the motion within our interactive environment. It supports not only rigid body dynamics but also more sophisticated modeling techniques. For example, we can pre-compute the fracturing effect for dynamic objects. Upon experiencing a significant force, these objects are realistically simulated by the real-time physics engine, which handles the interactions between the fractured pieces and the rest of the scene, such as their falling and settling on the ground. Besides browser-based engine, the virtual environments from Video2Game pipeline could be also integrated into both **Blender** [17] and **Unreal** engines [21] (see Fig. 6).

4. Experiments

We begin by presenting our experimental setup, followed by a comparison of our model with state-of-the-art approaches. Next, we conduct an extensive analysis of our model's distinctive features and design choices. Then we demonstrate how we constructed a web browser-compatible game capable of delivering a smooth interactive experience exceeding 100 frames per second (FPS), all derived from a single video source. Finally, we showcase the capabilities of our model in robot simulation through two demonstrations.

4.1. Setup

Dataset: We evaluate the effectiveness of Video2Game across three distinct scenes in various scenarios, including "Gardenvase" [12], an outdoor object-centric scene; the KITTI-360 dataset [37], a large-scale self-driving scene with a sequence that forms a closed loop, suitable for car-racing and Temple Run-like games; and finally, an indoor scene from the VR-NeRF [73] dataset, showcasing the potential for robot simulations.

Metrics: To evaluate the quality of the rendered images, we adopt PSNR, SSIM, and LPIPS [82]. For geometry reconstruction, we evaluate with LiDAR point cloud in KITTI-360 dataset. Root mean square deviation (RMSE), mean absolute error (MAE), and outlier rate are applied to measure the disparity existing in estimated geometry.

Our model: For NeRF, we adopt hashgrid encoding [47] and two-layer MLP for each header. For textured mesh,

G.T. | MobileNeRF [16] | BakedSDF [76] | Ours

Figure 5. **Qualitative comparisons among mesh models.** We compare our mesh rendering method with others in Garden scene [12].

Blender | Unreal

Figure 6. **Video2Game in Blender and Unreal Engine.**

Method	Outlier-%↓	RMSE↓	MAE↓
Instant-NGP [48]	22.89	4.300	1.577
Nerfacto [64]	50.95	8.007	2.863
Gauss. Spl. [28]	91.08	11.768	8.797
BakedSDF* (offline) [76]	43.78	5.936	2.509
Video2Game (Our NeRF)	**13.23**	**3.028**	**1.041**

Table 2. **Quantitative evaluation on NeRF geometry.** Our NeRF renders significantly more accurate depth compared with the baselines. The unit is meter and the outlier threshold is 1.5 meters.

Method	Volume Rendering			Mesh Rastization		
	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
Vanilla NGP	27.46	0.853	0.165	22.54	0.716	0.350
+ Regularization terms	27.52	0.861	0.157	22.97	0.732	0.303
+ Monocular cues	**27.62**	**0.871**	**0.131**	**23.35**	**0.765**	**0.246**

Table 3. **Ablation studies** on KITTI-360.

we conduct marching cubes on the NeRF and post-process it to a fixed precision. We set the texture image size to 4096x4096. For GLSL shader, we design a light-weight two-layer MLP, which enables efficient real-time rendering. For KITTI-360 (see Sec. 3.1), we divide the whole scene into 16 blocks and create a skydome mesh for the sky.

Baselines: To evaluate the visual and geometry quality of our model, we compare against SOTA approaches in neural rendering and neural reconstruction. **Instant-NGP** [48] is a NeRF-based method that exploits multi-resolution hashing encoding. **Nerfacto** [64] extends the classic NeRF with learnable volumetric sampling and appearance embedding. **3D Gaussian Splatting** [28] leverages 3D Gaussians and achieves fast training and rendering. **MobileNeRF** [16] adopts a hybrid NeRF-mesh representation. It can be baked into a texture map and enable real-time rendering. **BakedSDF** [76] adopts a volume-surface scene representation. It models view-dependent appearance efficiently by baking spherical Gaussians into the mesh.

4.2. Experimental results

Novel view synthesis: Tab. 1 shows the rendering performance and interactive compatibility of our model against the baselines on KITTI-360 [37] and Gardenvase [12]. Our NeRF achieves superior performance when compared to state-of-the-art neural volume render approaches across different scenes. Though [28] performs best in Gardenvase, it fails to handle the sparse camera settings in KITTI-360, where it learns bad 3D orientations of Gaussians. Our baked mesh outperforms other mesh rendering baselines significantly in KITTI-360 and performs similarly in Gardenvase as shown in Fig. 5. Additionally, our pipeline has the highest interactive compatibility among all baselines.

Geometry reconstruction: Our model performs significantly better than the baseline regarding geometry accuracy (see Tab. 2). We provide some qualitative results in Fig. 4, demonstrating that our model can generate high-quality depth maps and surface normals, whereas those produced by the baselines contain more noise.

Ablation study: To understand the contribution of each component in our model, we begin with the basic Instant-NGP [48] and sequentially reintroduce other components. The results in Tab. 3 show that our regularization and monocular cues improve the quality of both volume rendering in NeRF and mesh rasterization. Additionally, we do observe a decline in rendering performance when converting NeRF into game engine-compatible meshes.

4.3. Video2Game

We have shown our approach's effectiveness in rendering quality and reconstruction accuracy across various setups. Next, we demonstrate the construction of a web browser-compatible game enabling player control and interaction with the environment at over 100 FPS.

Data preparation: We build our environments based on videos in Gardenvase [12], KITTI-360 [37] and VR-NeRF [73] mentioned in Sec. 4.1, using our proposed approach. The outcomes include executable environments with mesh geometry, materials, and rigid-body physics, all encoded in GLB and texture files.

Game engine: We build our game based on several key components in our game engine mentioned in Sec. 3.4. By leveraging them, our game generates a highly realistic visual rendering as well as physical interactions (see Fig. 1) and runs smoothly at an interactive rate across various platforms, browsers, and hardware setups (see Tab. 4). As for other game engines (see Fig. 6), in Blender [17] we showcase the compatibility of our exported assets with other game engines. For Unreal [21], we further demonstrate a real-time game demo where a humanoid robot can freely interact within the Gardenvase scene, such as standing on the table and kicking off the central vase. These prove the compatibility of our proposed pipeline.

Interactive game features: *Movement in games:* Agents can navigate the area freely within the virtual environment where their actions follow real-world physics and are constrained by collision models. *Shooting game:* For realistic shooting physics, we calculated the rigid-body collision dynamics for both the central vase and the surrounding scene (see Fig. 3), separated using mesh semantic filtering. We used a box collider for the vase and convex polygon colliders for the background. The player shoots footballs with a sphere collider at the vase on the table, causing it to fly off and fall to the ground (see Fig. 1). *Temple-Run like game:* The agent collects coins while running in the KITTI Loop composed of four streets in KITTI-360. Obstructive chairs on the road can be smashed thanks to pre-computed fracture animations. The agent can also drive and push roadside vehicles existing in the scene forward by crashing into them. This interactivity is achieved through rigid-body dynamics simulation and collision modeling.

Robot simulation: We demonstrate the potential of leveraging Video2Game for robot simulation using the VRNeRF dataset. We reconstruct the scene and segment simulatable rigid-body objects (*e.g.*, the fruit bowl on the table). We show two demos in Fig. 7: a Stretch Robot pushing the bowl off the table and a Fetch Robot performing pick-and-place actions. We employ PyBullet [18] to simulate the underlying physics with the help of corresponding collision models. Since real-time grasping simulation is challenging, following existing robot simulation frameworks [8, 9, 32], objects near the Fetch gripper are automatically picked up. This demonstrates our model's ability to convert a real-time video stream into a virtual environment, allowing robots to rehearse before acting in the real environment.

	Platform	FPS (hz)	CPU-Usage (%)	GPU-Usage (%)
Mac M1 Pro	Mac OS, Chrome	102	34	70
Intel Core i9 + NV 4060	Windows, Edge	240	6	74
AMD 5950 + NV 3090	Linux, Chrome	144	20	40

Table 4. **Runtime Analysis.** Our interactive environment can run in real-time across various hardware setup and various platforms. User actions may slightly vary, which could lead to minor variations in runtime.

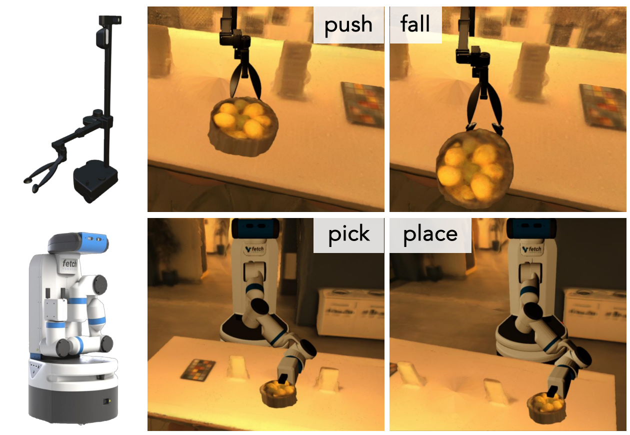

Figure 7. **Robot simulation in VRNeRF [73] dataset.** We demonstrate the possibility of conducting robot learning in our virtual environments using Stretch Robot [6] and Fetch Robot [5].

5. Conclusion

We present a novel approach to converting real-world video footage into playable, real-time, and interactive game environments. Specifically, we combine the potential of NeRF modeling with physics modeling and integrate them into modern game engines. Our approach enables any individual to transform their surroundings into an interactive digital environment, unlocking exciting possibilities for 3D content creation, with promising implications for future advancements in digital game design and robot simulation.

References

[1] Cannon.js. https://schteppe.github.io/cannon.js/. 6
[2] GLSL. https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language. 6
[3] Sketchbook. https://github.com/swift502/Sketchbook. 6
[4] WebGL. https://www.khronos.org/webgl/. 6

[5] Fetch Mobile Manipulator. https://fetchrobotics.borealtech.com/robotics-platforms/fetch-mobile-manipulator/?lang=en. 8

[6] Stretch® Research Edition. https://hello-robot.com/product. 8

[7] Xatlas. https://github.com/mworchel/xatlas-python. 5

[8] RoboTHOR: An Open Simulation-to-Real Embodied AI Platform. *CVPR*, 2020. 8

[9] ManipulaTHOR: A Framework for Visual Object Manipulation. In *CVPR*, 2021. 8

[10] Alexander Amini, Tsun-Hsuan Wang, Igor Gilitschenski, Wilko Schwarting, Zhijian Liu, Song Han, Sertac Karaman, and Daniela Rus. Vista 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles. In *ICRA*, 2022. 3

[11] Omer Bar-Tal, Dolev Ofri-Amar, Rafail Fridman, Yoni Kasten, and Tali Dekel. Text2live: Text-driven layered image and video editing. In *ECCV*, 2022. 2

[12] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022. 4, 6, 7, 8

[13] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. *ECCV*, 2022. 2

[14] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, 1993. 2

[15] Yun Chen, Frieda Rong, Shivam Duggal, Shenlong Wang, Xinchen Yan, Sivabalan Manivasagam, Shangjie Xue, Ersin Yumer, and Raquel Urtasun. Geosim: Realistic video simulation via geometry-aware composition for self-driving. In *CVPR*, 2021. 3

[16] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *CVPR*, 2023. 2, 5, 7

[17] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 6, 8

[18] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021. 8

[19] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator, 2017. 1, 3

[20] 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. 4

[21] Epic Games. Unreal engine. 1, 6, 8

[22] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 2

[23] Rockstar Games. Grand theft auto v. 2014. 1

[24] John K Haas. A history of the unity game engine. 2014. 1

[25] Benno Heigl, Reinhard Koch, Marc Pollefeys, Joachim Denzler, and Luc Van Gool. Plenoptic modeling and rendering from image sequences taken by a hand-held camera. In *DAGM-Symposium*, 1999. 2

[26] Ronghang Hu, Nikhila Ravi, Alexander C Berg, and Deepak Pathak. Worldsheet: Wrapping the world in a 3d sheet for view synthesis from a single image. In *ICCV*, 2021. 2

[27] Oğuzhan Fatih Kar, Teresa Yeo, Andrei Atanov, and Amir Zamir. 3d common corruptions and data augmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18963–18974, 2022. 4

[28] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (ToG)*, 42(4):1–14, 2023. 5, 7

[29] Pradeep K Khosla and Takeo Kanade. Parameter identification of robot dynamics. In *IEEE conference on decision and control*, 1985. 3

[30] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with gamegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1231–1240, 2020. 2, 3

[31] Seung Wook Kim, Jonah Philion, Antonio Torralba, and Sanja Fidler. Drivegan: Towards a controllable high-quality neural simulation. In *CVPR*, 2021. 2, 3

[32] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai, 2022. 6, 8

[33] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics*, 39(6), 2020. 5

[34] Yao-Chih Lee, Ji-Ze Genevieve Jang, Yi-Ting Chen, Elizabeth Qiu, and Jia-Bin Huang. Shape-aware text-driven layered video editing. *arXiv*, 2023. 2

[35] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996. 2

[36] Yuan Li, Zhi-Hao Lin, David Forsyth, Jia-Bin Huang, and Shenlong Wang. Climatenerf: Physically-based neural rendering for extreme climate synthesis. *arXiv*, 2022. 2, 4

[37] Yiyi Liao, Jun Xie, and Andreas Geiger. Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *IEEE TPAMI*, 2022. 4, 6, 7, 8

[38] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *ICCV*, 2021. 2

[39] Zhi-Hao Lin, Wei-Chiu Ma, Hao-Yu Hsu, Yu-Chiang Frank Wang, and Shenlong Wang. Neurmips: Neural mixture of planar experts for view synthesis. In *CVPR*, 2022. 2

[40] 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, 2023. 2

[41] Jeffrey Yunfan Liu, Yun Chen, Ze Yang, Jingkang Wang, Sivabalan Manivasagam, and Raquel Urtasun. Real-time neural rasterization for large scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8416–8427, 2023. 3

[42] Fan Lu, Yan Xu, Guang Chen, Hongsheng Li, Kwan-Yee Lin, and Changjun Jiang. Urban radiance field representation with deformable neural mesh primitives, 2023. 3

[43] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. Lidarsim: Realistic lidar simulation by leveraging the real world. In *CVPR*, 2020. 3

[44] Willi Menapace, Stephane Lathuiliere, Sergey Tulyakov, Aliaksandr Siarohin, and Elisa Ricci. Playable video generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10061–10070, 2021. 3

[45] Willi Menapace, Stéphane Lathuiliere, Aliaksandr Siarohin, Christian Theobalt, Sergey Tulyakov, Vladislav Golyanik, and Elisa Ricci. Playable environments: Video manipulation in space and time. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3584–3593, 2022. 3

[46] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 2, 4

[47] Thomas Müller. tiny-cuda-nn, 2021. 6

[48] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM TOG*, 2022. 2, 3, 4, 5, 7

[49] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *CVPR*, 2021. 2

[50] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *ICCV*, 2021.

[51] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv*, 2021. 2

[52] Cristina Portalés, José Luis Lerma, and Santiago Navarro. Augmented reality and photogrammetry: A synergy to visualize physical and virtual city environments. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65(1):134–142, 2010. 1

[53] Charalambos Poullis, Andrew Gardner, and Paul Debevec. Photogrammetric modeling and image-based rendering for rapid virtual environment creation. In *Proceedings of ASC*, 2004. 1

[54] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE TPAMI*, 2020. 4

[55] Christian Reiser, Richard Szeliski, Dor Verbin, Pratul P Srinivasan, Ben Mildenhall, Andreas Geiger, Jonathan T Barron, and Peter Hedman. Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *arXiv preprint arXiv:2302.12249*, 2023. 2, 5

[56] Konstantinos Rematas, Andrew Liu, Pratul P Srinivasan, Jonathan T Barron, Andrea Tagliasacchi, Thomas Funkhouser, and Vittorio Ferrari. Urban radiance fields. In *CVPR*, 2022. 2

[57] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-a-video: Text-to-video generation without text-video data. *arXiv*, 2022. 2, 3

[58] Uriel Singer, Shelly Sheynin, Adam Polyak, Oron Ashual, Iurii Makarov, Filippos Kokkinos, Naman Goyal, Andrea Vedaldi, Devi Parikh, Justin Johnson, et al. Text-to-4d dynamic scene generation. *arXiv*, 2023. 2, 3

[59] Sanghyun Son, Yi-Ling Qiao, Jason Sewall, and Ming C Lin. Differentiable hybrid traffic simulation. *TOG*, 2022. 3

[60] Pratul P Srinivasan, Richard Tucker, Jonathan T Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. Pushing the boundaries of view extrapolation with multiplane images. In *CVPR*, 2019. 2

[61] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 2

[62] Richard Szeliski and Polina Golland. Stereo matching with transparency and matting. In *Sixth International Conference on Computer Vision*, 1998. 2

[63] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *CVPR*, 2022. 4

[64] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. *arXiv*, 2023. 5, 7

[65] Jiaxiang Tang, Hang Zhou, Xiaokang Chen, Tianshu Hu, Errui Ding, Jingdong Wang, and Gang Zeng. Delicate textured mesh recovery from nerf via adaptive surface refinement. *arXiv preprint arXiv:2303.02091*, 2023. 2, 5

[66] ADRIAN SAVARI Thomas, MOHD FAHRUL Hassan, MUSTAFFA Ibrahim, M Nasrull, A Rahman, SZ Sapuan, and F Ahmad. A study on close-range photogrammetry in image based modelling and rendering (imbr) approaches and post-processing analysis. *Journal of Engineering Science and Technology*, 14(4):1912–1923, 2019. 1

[67] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012. 3

[68] Shubham Tulsiani, Richard Tucker, and Noah Snavely. Layer-structured 3d scene inference via view synthesis. In *ECCV*, 2018. 2

[69] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. Ref-nerf: Structured view-dependent appearance for neural radiance fields. In *CVPR*, 2022. 2

[70] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. Nerf–: Neural radiance fields without known camera parameters. *arXiv*, 2021. 2

[71] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http://torcs.sourceforge.net*, 2000. 3

[72] Yuwen Xiong, Jingkang Ma, Wei-Chiu Wang, and Raquel Urtasun. Ultralidar: Learning compact representations for lidar completion and generation. *CVPR*, 2023. 3

[73] Linning Xu, Vasu Agrawal, William Laney, Tony Garcia, Aayush Bansal, Changil Kim, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, Aljaž Božič, Dahua Lin, Michael Zollhöfer, and Christian Richardt. VR-NeRF: High-fidelity virtualized walkable spaces. In *SIGGRAPH Asia Conference Proceedings*, 2023. 6, 8

[74] Zhenpei Yang, Yuning Chai, Dragomir Anguelov, Yin Zhou, Pei Sun, Dumitru Erhan, Sean Rafferty, and Henrik Kretzschmar. Surfelgan: Synthesizing realistic sensor data for autonomous driving. In *CVPR*, 2020. 3

[75] Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. *CVPR*, 2023. 2, 3

[76] 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. *Siggraph*, 2023. 2, 5, 7

[77] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields, 2021. 4

[78] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *in arXiv*, 2022. 4

[79] Haotian Zhang, Ye Yuan, Viktor Makoviychuk, Yunrong Guo, Sanja Fidler, Xue Bin Peng, and Kayvon Fatahalian. Learning physically simulated tennis skills from broadcast videos. *ACM Trans. Graph.* 3

[80] Haotian Zhang, Cristobal Sciutto, Maneesh Agrawala, and Kayvon Fatahalian. Vid2player: Controllable video sprites that behave and appear like professional tennis players. *ACM Transactions on Graphics (TOG)*, 40(3):1–16, 2021. 3

[81] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv*, 2020. 2

[82] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 6

[83] Xiuming Zhang, Pratul P Srinivasan, Boyang Deng, Paul Debevec, William T Freeman, and Jonathan T Barron. Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. *ACM TOG*, 2021. 2

[84] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew J. Davison. In-place scene labelling and understanding with implicit scene representation, 2021. 4

[85] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *arXiv*, 2018. 2

[86] Yiming Zuo and Jia Deng. View synthesis with sculpted neural points. *in arXiv*, 2022. 2

[87] Vlas Zyrianov, Xiyue Zhu, and Shenlong Wang. Learning to generate realistic lidar point clouds. In *ECCV*, 2022. 3