ClimateNeRF: Physically-based Neural Rendering for Extreme Climate Synthesis

Yuan Li[1,2*] Zhi-Hao Lin[1*] David Forsyth[1] Jia-Bin Huang[3] Shenlong Wang[1]
[1]University of Illinois at Urbana-Champaign [2]Zhejiang University
[3]University of Maryland, College Park

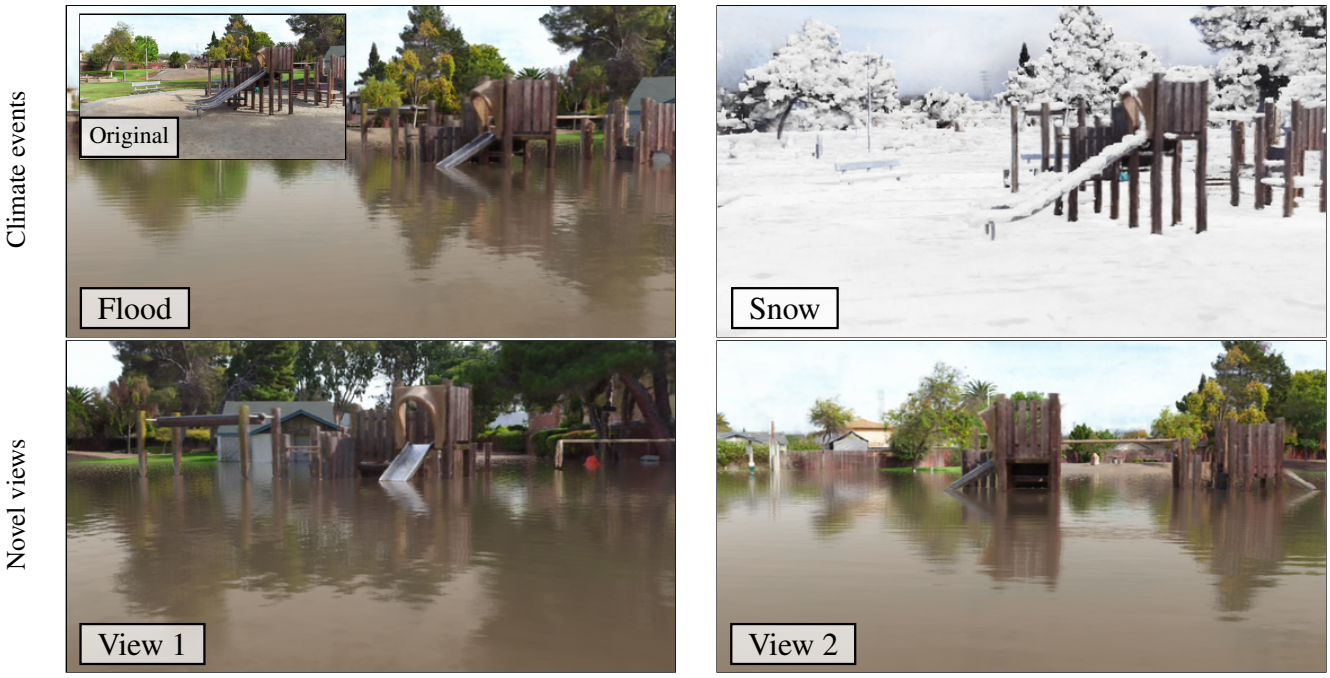


Figure 1. **Extreme climate synthesis.** We fuse NeRF modelling and physical simulation to produce 3D consistent renderings of scenes with simulated physical effects. We apply our method to climate effect simulation: what will it look like if the playground floods? or is covered in snow? Note in particular reflections and ripple effects on water; accumulated snow on horizontal surfaces; trees darkened by wintertime; and consistency of geometry (but not ripples!) across views. Please visit our project page for the interactive illustrations.

Abstract

Physical simulations produce excellent predictions of weather effects. Neural radiance fields produce SOTA scene models. We describe a novel NeRF-editing procedure that can fuse physical simulations with NeRF models of scenes, producing realistic movies of physical phenomena inthose scenes. Our application – Climate NeRF – allows people to visualize what climate change outcomes will do to them.

ClimateNeRF allows us to render realistic weather effects, including smog, snow, and flood. Results can be controlled with physically meaningful variables like water level. Qualitative and quantitative studies show that our simulated results are significantly more realistic than those from SOTA 2D image editing and SOTA 3D NeRF stylization.

1. Introduction

This paper describes a novel procedure that fuses physical simulations with NeRF models [53, 55] of scenes to produce realistic movies of physical phenomena in those scenes. We apply our method to produce compelling simulations of possible climate change outcomes – what would the playground look like after a minor flood? a major flood? a blizzard?

Our application – Climate NeRF – is aimed at an important problem. Cumulative small changes are hard to reason about, and most people find it difficult to visualize what climate change outcomes will do *to them* [9, 25, 29, 41]. Steps to slow CO_2 emissions (say, reducing fossil fuel use) or to moderate outcomes (say, building flood control measures) come with immediate costs and distant benefits. It is hard to support such steps if one can't visualize their effects.

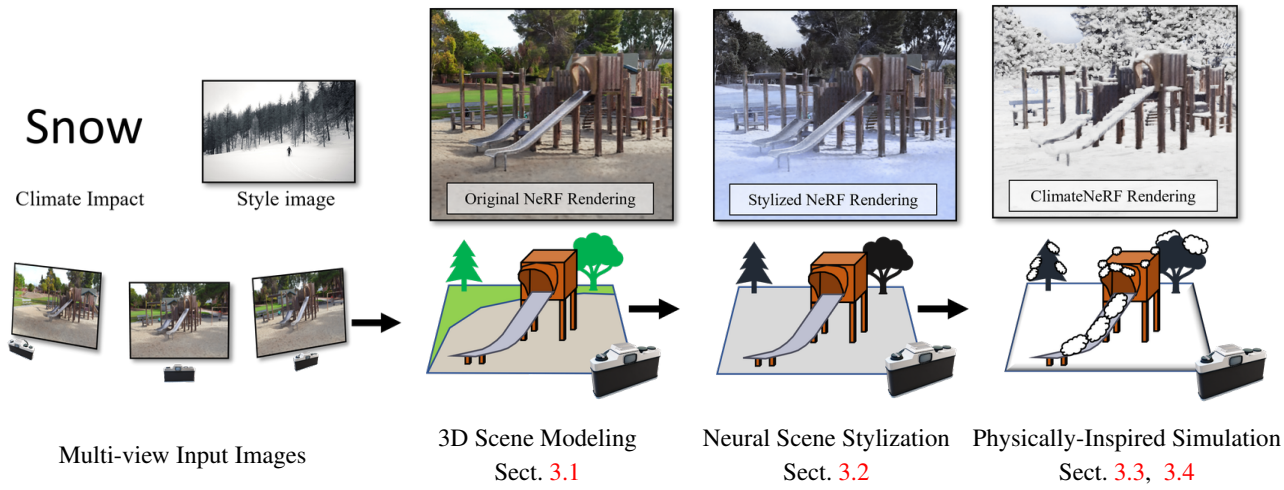


Figure 2. **Method Overview.** Our method takes multiple posed images, the targeted climate event simulation (e.g., snow), and optionally a user-selected style image as inputs. First, we reconstruct the 3D scene using instant NGP [55] (a variant of NeRF) (Sect. 3.1). The reconstructed radiance fields allow us to synthesize high-quality novel view imagery of the scene efficiently. Second, we optionally finetune the learned instant-NGP model so that it captures the styles of the provided style image (Sect. 3.2). Such 3D consistent stylization is particularly useful for modeling weather effects that are hard to capture via physical simulation. Third, we simulate the climate events by integrating the relevant physical entities (snow, water, smog) to the scene and rendering physically plausible images.

We show how to merge *physical simulations* – which produce excellent predictions of weather effects, but only moderate images – with *neural radiance fields* – which produce SOTA scene models but (as far as we know) have never been used together with physical simulations. Traditional physical simulations can produce realistic weather effects for 3D scenes in a conventional graphics pipeline [15, 19, 21, 27, 31, 69, 97]. But these methods operate on conventional polygon models. Building polygon models that produce compelling renderings from a few images of a scene remains challenging. Neural radiance fields (NeRFs) produce photorealistic 3D scene models from few images [3, 10, 45, 53, 55, 72, 82]. Our method draws from a large literature, reviewed below, that explores editing these models.

ClimateNeRF allows us to render realistic weather effects, including smog, snow, and flood. These effects are consistent over frames, so that compelling movies result. At a high level, we: adjust scene images to reflect global effects of the physics; build a NeRF model of a scene from those adjusted images; recover an approximate geometric representation; apply the physical simulation in that geometry; then render using a novel ray tracer. Adjusting the images is important. For example, trees tend to have less saturated images in winter. We use a novel style transfer approach in an NGP framework to obtain these global effects without changing scene geometry (Section 3.2). Our ray tracer merges the physical and NeRF models by carefully accounting for ray effects during rendering (Section 3.3). An eye ray could, for example, first encounter a high NeRF density (and so return the usual result); or it could strike an inserted water surface (and so be reflected to query the model again).

We demonstrate the effectiveness of ClimateNeRF in various 3D scenes from the Tanks and Temple, MipNeRF360, and KITTI-360 datasets [3, 37, 44]. We compared to the state-of-the-art 2D image editing methods, such as stable diffusion inpainting [61], ClimateGAN [66]; state-of-the-art 3D NeRF stylization [90]. Both qualitative and quantitative studies show that our simulated results are significantly more realistic than the other competing methods. Furthermore, we also demonstrate the controllability of our physically-inspired approaches, such as changing the water level, wind strength and direction, and thickness of snow and smog.

Our approach results in view consistency (so we can make movies, which is difficult to do with frame-by-frame synthesis); compelling photorealism (because the scene is a NeRF representation); and is controllable (because we can adjust physically meaningful parameters in the simulation). As Fig. 1 illustrates, results are photo-realistic, physically plausible, and temporally consistent.

2. Related Work

Since 1981, the Earth's temperature has been increasing over 0.32° F (0.18° C) per decade [8], with impacts on health, environment, and economy and disproportionally affecting people in low-income communities and developing countries [30]. Complex scientific figures from statistical analysis and forecasting do not resonate well with everyday people [9, 25, 29, 41], and various 2D image synthesis techniques have been applied to bring climate effects to a captured image.

Climate simulation: The importance of making climate simulations accessible is well known. [65] collects climate

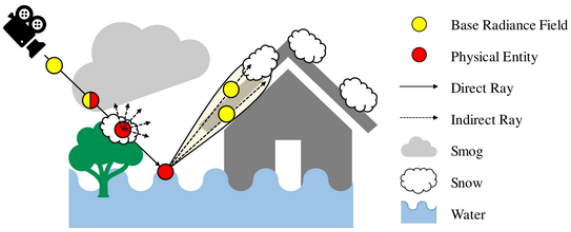
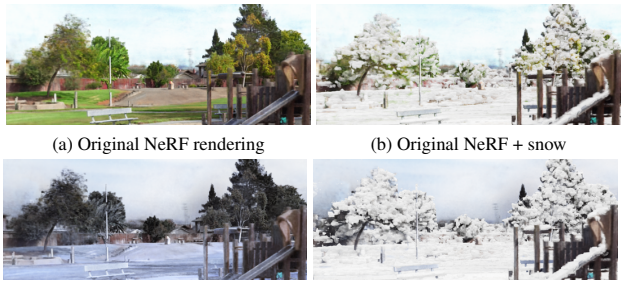


Figure 3. **Rendering Procedure of ClimateNeRF.** We first determine the position of physical entities (smog particle, snow balls, water surface) with physical simulation. We can then render the scene with desired effects by modeling the light transport between the physical entities and the scene. More specifically, we follow the volume rendering process and fuse the estimated color and density from 1) the original radiance field (by querying the trained instant-NGP model) and 2) the physical entities (by physically based rendering). Our rendering procedure thus maintain the realism while achieving complex, yet physically plausible visual effects.

images dataset and performs image editing with CycleGAN [95]. [17, 66] leverage depth information to estimate water mask, and perform GAN-based image editing and inpainting. [28] simulate fog and snow. These methods offer realistic effects for a single image, but cannot provide immersive, view-consistent climate simulation. In contrast, ClimateGAN allows the view to move without artifacts.

Novel view synthesis: Neural Radiance Field (NeRF) [2, 3, 53, 76] leverage differentiable rendering for scene reconstruction producing photorealistic novel view synthesis. Training and rendering can be accelerated using multiple smaller MLPs and customized CUDA kernel functions [45, 60]. Sample efficiency can be improved by modeling the light field or calculating ray intersections with explicit geometry, such as voxel grids [48, 60, 82], octrees [87], planes [45, 81], and point cloud [84, 98]. Explicit geometric representations improve efficiencies [24, 50, 70, 84, 92]. We build on Instant-NGP [55] as it offers fast learning and rendering, is more memory efficient than grid-like structures, and accelerates our simulation pipeline.

Manipulating neural radiance fields: NeRF entangles lighting, geometry and materials, making appearance editing hard. A line of work [6, 7, 43, 62, 67, 91, 93] decomposes neural scene representations into familiar scene variables (like environment lighting, surface normal, diffuse color, or BRDF). Each can then be edited with predictable results on rendering. Image segmentations and inpaintings can be "lifted" to 3D, allowing object removal [5, 39]. An alternative is to edit source images with text [79]. In contrast, our method does not directly edit the NeRF while producing substantial changes to scene appearance. Geometry editing by transforming a NeRF into a mesh and manipulating that (so editing shapes and compositing) has been demonstrated [14, 85, 89]. In contrast, our method renders the original NeRF together with simulation results.



(a) Original NeRF rendering (b) Original NeRF + snow

(c) NeRF with stylization (d) NeRF with stylization + snow

Figure 4. **Stylization.** Performing snow simulation with the original NeRF model leads to incompatible scene appearance (e.g., snow on green vegetations). We address this issue by first finetuning the appearance of the NeRF model to match the style of the provided style image. Our snow simulation on the stylized NeRF model shows visually more appealing results.

Style transfer: Data-driven 2D stylization [32, 35, 57] is a powerful tool to change image color and texture, and can even change the content following masks and labels provided by users [1, 46, 47, 96]. Diffusion models [61, 63, 64] can generate extraordinary image quality based on text input. However, these 2D-based methods cannot currently generate 3D-consistent view synthesis; ours can. Another line of work [12, 13, 90] performs stylization on neural radiance fields. Given an style image, one changes the scene color into the same style. This line of work mainly focuses on artistic effects; in contrast, our changes are driven by physical simulations.

Physically-based simulation of weather: Physical simulation of weather in computer graphics has too long a history to allow comprehensive review here. Fournier and Reeves obtain excellent capillary wave simulations with simple Fourier transform reasoning [22] (and we adopt their method), with modifications by [31, 74]; [20] simulates smoke; and [21, 56, 69] simulate snow in wind using metaballs and fluid dynamics. Our method demonstrates how to benefit from such simulations while retaining the excellent scene modelling properties of NeRF style models.

3. Method

ClimateNeRF fuses physical simulations with NeRF models of scenes to produce realistic videos of climate change effects. A simple example illustrates how components interact in our approach. Imagine we wish to build a model of a flooded scene in Fall. We acquire images, apply a Fall style (Section 3.2), and build a NeRF from the results (Section 3.1). We then use geometric information in that NeRF to compute a water surface. This is represented using a density field, a color field together with normal and BRDF representations (Section 3.3). Finally, to render we query the model with rays. The details are elaborate (Section 3.4), but the general idea is straightforward: we edit the NeRF's

(a) Original NeRF (b) Depth map (c) Water surface (d) Normal map with wave (e) Final ClimateNeRF

Figure 5. **Flooding Simulation.** We first estimate the vanishing point direction based on the original image (a) and depth (b). With the vertical vanishing direction (yellow arrows painted (c)), we can insert a planar water surface $\mathbf{n}_w(x - \mathbf{o}_w)$. We use FFT based water surface simulation to produce a spatiotemporal surface normal map in (d). Our ClimateNeRF renders the scene with the simulated flood through ray tracing NeRF (e).

density and color functions to represent effects like smog; and we intercept rays to represent specular effects. So if a ray encounters high density in the NeRF first, we use the NeRF integral for that ray; but if the first collision is with the water surface, we reflect that ray in the water surface, then query the NeRF with the reflected ray. Fig. 2 provides an overview of our approach.

3.1. 3D Scene Reconstruction

NeRF builds a parametric scene representation that supports realistic rendering from multiple images of a scene obtained at known poses. The scene is represented by a field

$$(\sigma, \mathbf{c}) = F_\theta(\mathbf{x}, \mathbf{d}), \tag{1}$$

which accepts position $\mathbf{x}$ and direction $\mathbf{d}$ and predicts density $\sigma \in \mathbb{R}$ and color $\mathbf{c} \in \mathbb{R}^3$. This function is encoded in a multi-layer perceptron (MLP) with learnable parameters θ. Rendering is by querying radiance along appropriate choices of ray, computed as a volume integral [34]. This integral is estimated by drawing samples along the ray, evaluating the density and color at those samples, then accumulating values. The rendering process is differentiable, so that NeRF can be trained by minimizing the image reconstruction loss over training views through gradient descent: $\min_\theta \sum_{\mathbf{r}} \|C(\mathbf{r}) - C_{gt}(\mathbf{r})\|_2^2$.

There are numerous variants of NeRF. We use instant-NGP [55] to reconstruct the scene. This is an efficient NeRF alternative that explicitly stores multi-resolution features γ for scene representation. During rendering, given an input point, a local feature $\gamma(\mathbf{x})$ is firstly queried through a spatial hashing function [73] and is then sent to a shallow multi-layer perceptron (MLP) to compute the final density σ and color $\mathbf{c}$: $\sigma, \mathbf{c} = F_\theta(\mathbf{x}, \mathbf{d}; \gamma(\mathbf{x}))$. This explicit feature encoding and spatial partition are particularly suitable for ClimateNeRF because we can edit local features relatively easily.

A physical simulation needs access to the surface normal of any point to compute interactions with snow and water, and it needs access to the point's semantics (in the sense of semantic segmentation) to transfer style. We expand the NGP and allow it to output both semantic logic $\mathbf{s}$ and surface normal $\mathbf{n}$. There is no semantic or surface normal ground truth during training. We use an off-the-shelf pretrained monocular semantic segmentation network [83] to produce semantic maps for each image. We use density gradients $\hat{\mathbf{n}} = -\frac{\nabla\sigma}{\|\nabla\sigma\|}$ [6, 67] (cf. Ref-NeRF [77]) to guide the predicted surface normals $\mathbf{n}$ with a weighted MSE loss.

To simulate (say) a blizzard, we must add snow to the scene and turn trees dark, but should not change the shape of the house. To keep spatial features intact at the stylization stage, we disentangle instant-NGP's latent feature (as in [10, 70]). For each voxel in the NGP model, we split the latent feature into geometry features γ_{geo} and appearance features γ_{app}. The geometry features are trained to render density. The appearance features are used for rendering color, semantics and normals. We will freeze the geometry feature vector during the stylization stage and change only the appearance feature vector.

Our 3D scene representation model is an improved instance-NGP model which renders density σ, color $\mathbf{c}$, semantic log posterior $\mathbf{s}$ and surface normal $\mathbf{n}$, given a query point and ray direction, so

$$(\sigma, \mathbf{c}, \mathbf{s}, \mathbf{n}) = F_\theta(\mathbf{x}, \mathbf{d}; \gamma_{\text{geo}}, \gamma_{\text{app}}) \tag{2}$$

(more details in supplementary).

3.2. Stylization

Deciduous trees drop their leaves in winter, and physical simulation is not an efficient way to capture effects like this. We use FastPhotoStyle [42] to transfer style to rendered images from a pre-trained model F_θ. We only transfer only to regions 'terrain', 'vegetation', or 'sky' regions to mimic natural weather change phenomena. The resulting images look realistic but are not necessarily view-consistent. Hence, a student instant-NGP model is fined-tuned to ensure the view consistency of the style-transferred scene. This is trained to minimize the color difference between our student NeRF-rendered results and style-transferred images. We keep the geometry intact and alter only appearance to achieve this goal, so only the appearance feature code γ_{app} is optimized during the style transfer stage:

$$\min_{\gamma_{\text{app}}} \sum_{\mathbf{r} \in \mathcal{R}} \|C(\mathbf{r}) - C_{\text{stylized}}(\mathbf{r})\|_2^2 \tag{3}$$

(a) Original NeRF (b) Surface normal (c) Metaball centers (red) (d) Snow with diffuse model (e) Snow with scattering

Figure 6. **Snow simulation.** We first locate metaballs on object surfaces facing upward based on surface normal values (b). With metaballs (centers painted in red), we can estimate the density $\sigma_{snow}(\mathbf{x})$ and color $\mathbf{c}(\mathbf{x})$ with a parzen window density estimator. (d) and (e) show the differences between fully diffuse model and scattering approximations, shadowed parts in (d) are lit in (e).

where $C(\mathbf{r})$ is renderd color and C_{stylized} is the style-transferred in the same view. This gives us a new NGP model $(\sigma, \mathbf{c}') = F'_\theta(\mathbf{x}, \mathbf{d}, \ell_i^{(a)})$ which encodes the style. Fig. 4 demonstrates the visual effects of such NeRF stylization. This optional style transfer step simulates composite effects, such as a flood in Fall, where the original images were captured in Spring.

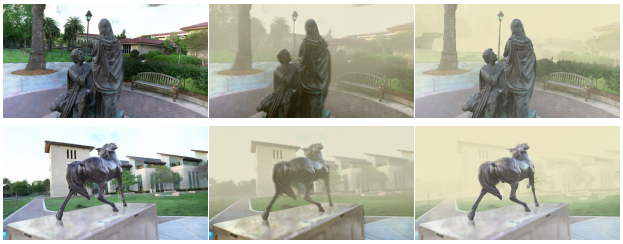
3.3. Representing and Rendering Climate Effects

We want to generate scenes with new physical entities – snow, water, smog – in place. We must determine where they are (the job of physical simulation) and what the resulting image looks like (the job of rendering). *How* the simulation represents results is important, because results must be accessible to the rendering process. Rendering will always involve computing responses to ray queries, so computing radiance at $\mathbf{u}$ in direction $\mathbf{v}$. We must represent simulation results in terms of densities and we must be able to compute normals and surface reflectance properties. Generally, we write $O_\phi(\mathbf{x}; F_\theta) : \mathbb{R}^3 \to \mathbb{R}$ for a density resulting from a physical simulation; $N_\phi(\mathbf{x}; F_\theta) : \mathbb{R}^3 \to \mathbb{S}^2$ for normals; and $B_\phi(\mathbf{x}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i; F_\theta) : \mathbb{R}^9 \to \mathbb{R}$ for BRDF. Each depends on the existing scene F_θ. Choice of B_ϕ can simulate various effects, including the atmospheric effect of smog, refraction and reflections on water surfaces, and scattering of accumulated snow. $\{O_\phi, N_\phi, B_\phi\}$ differs drastically across different physical simulations (details per effect in Sect. 3.4).

Once the physical entities are defined by functions O_ϕ, N_ϕ, B_ϕ, we can render them realistically into the image by modeling the light transport between the physical entities and the scene. Given the query point position $\mathbf{x}$, the simulation framework estimates the density and color of physical entities at position $\mathbf{x}$ through physically based rendering:

$$\sigma_\phi = O_\phi(\mathbf{x}; F_\theta),$$
$$\mathbf{c}_\phi = \int_\Omega L(\mathbf{x}, \boldsymbol{\omega}_i) B_\phi(\mathbf{x}, \mathbf{d}, \boldsymbol{\omega}_i; F_\theta)(\boldsymbol{\omega}_i \cdot N_\phi(\mathbf{x}; F_\theta))\mathrm{d}\boldsymbol{\omega}_i, \tag{4}$$

where entity color $\mathbf{c}_\phi$ is approximated with physically-based rendering equation [33] with normal N_ϕ and BRDF B_ϕ. Importantly, we approximate the incident illumination $L(\mathbf{x}, \boldsymbol{\omega}_i)$ with radiance by tracing a ray $\mathbf{r}(t) = \mathbf{x} - t\boldsymbol{\omega}_i$ opposite to the incident direction in the learned NeRF, *i.e.* $L(\mathbf{x}, \boldsymbol{\omega}_i) = C(\mathbf{r})$.



Original ClimateGAN [66] Ours

Figure 7. **Smog simulation comparison.** ClimateNeRF simulate smog in view-consistent manner, and separate foreground objects from background better.

Depending on the surface BRDF of physical entities, we use analytical or sampling-based solutions for the integral. Note that multiple bounces can be simulated through sampling.

We follow the volumetric rendering process defined in two passes. For every point along a camera ray, we query the opacity and color of the physical entities as defined in Eq. 4. In the meantime, the system also retrieves the original density σ_θ and color $\mathbf{c}_\theta$ through Eq. 2. ClimateNeRF estimates final density and color of the simulated scene by:

$$\sigma_{\text{final}} = \sigma_\theta + \sigma_\phi, \quad \mathbf{c}_{\text{final}} = \frac{\sigma_\theta \mathbf{c}_\theta + \sigma_\phi \mathbf{c}_\phi}{\sigma_\theta + \sigma_\phi}, \tag{5}$$

Once $\sigma_{\text{final}}, \mathbf{c}_{\text{final}}$ are estimated for each ray points $\{\mathbf{x}_i\}$, the pixel color is calculated by volume rendering. Fig. 3 depicts the entire physically inspired simulation and rendering process.

3.4. Climate Effect Details

ClimateNeRF is able to simulate smog, flood, and snow through various choices of $\{O_\phi, N_\phi, B_\phi\}$.

Smog Simulation We assume that smog is formed by tiny absorbing particles, uniformly distributed in empty space. In empty space the NeRF density $\sigma_\theta = 0$. The Beer-Lambert law (originally [4, 40]; in [23]) means we can model smog density in free space by simply adding a non-negative constant to the density. Inside high density regions of the NeRF, adding the constant does not significantly change the integral, so we have

$$O_\phi(\mathbf{x}; F_\theta) = \sigma_{\text{smog}} \tag{6}$$

where σ_ϕ is a controllable parameter that decides the density of the smog. Smog particles have a constant color $\mathbf{c}_{\text{smog}}$.

Original | ClimateGAN++ [66] | Stable Diffusion [61] | Ours

Figure 8. **Flood simulation comparison.** Both ClimateGAN++ [66] and Stable Diffusion [61] inpaint the flood using a water mask from our reconstructed geometry, but ClimateGAN++ [66] does not generate accurate reflections or ripples. Stable Diffusion [61] yields better water texture but scatters irrelevant vegetation on the water. In contrast, ClimateNeRF simulates photorealistic reflection and ripples, which are also consistent across views.

Both $\mathbf{c}_{\text{smog}}$ and σ_{smog} are controllable parameters. Fig. 12 depicts the effects of various smog densities.

Flood Simulation The water surface of the flooded scene is approximately a horizontal plane: $\mathbf{n}_w(\mathbf{x}-\mathbf{o}_w)=0$, where the gravity direction normal $\mathbf{n}_w$ is estimated with camera poses and vanishing points detection [49], and plane origin $\mathbf{o}_w=(0,0,h)$ determines the water height. But there are water ripples, which we implement following [31] with Fast Fourier Transform (FFT) based ripples and waves. The FFT wave takes random spectral coefficients as input and outputs a spatiotemporal surface normal based on wind speed, direction, and spatial and temporal frequencies. As shown in Fig. 8, compared against still water, FFT-based water surface simulation significantly improves the realism of the water surfaces. We simulate opacity and micro-facet ripples that make the water look glossy (details in supplementary). To approximate the integral in Eq. 4, we adopt the sigma-point method [52, 78] and sample 5 rays from $\mathbf{x}$, including reflection direction $\mathbf{d}_r$ and nearby four rays. ClimateNeRF simulates Fresnel effect, glossy reflection, and wave dynamics.

Snow Simulation Snow is more likely to be accumulated on surfaces facing upward, and the deeper part of the snow is denser due to gravity. We simulate density distributions over object surfaces using metaballs [36, 56] centered on surfaces and with density σ_o at the center. The density distribution within a metaball can be formulated by kernel function $\mathbf{K}(r,R,\sigma_o)$, which leads to a smooth decrease of density as the distance r from the metaball's center grows. We follow [75] and tuned it to create a denser visual effect: $\mathbf{K}(r,R,\sigma)=\frac{315}{64\pi 1.5^7}(1.5^2-(\frac{r}{R})^2)^3\sigma$

For any point $\mathbf{x}$ in the space, we calculate the snow's density of $\mathbf{x}$ using a parzen window density estimator over N local nearest neighbors (details in supplementary). The final density of the snow surface is decided accordingly:

$$O_\phi(\mathbf{x};F'_\theta)=\frac{1}{1+e^{-a(x\sigma_{\text{snow}}-\tau_{\text{snow}})}}\sigma_{\text{snow}}, \tag{7}$$

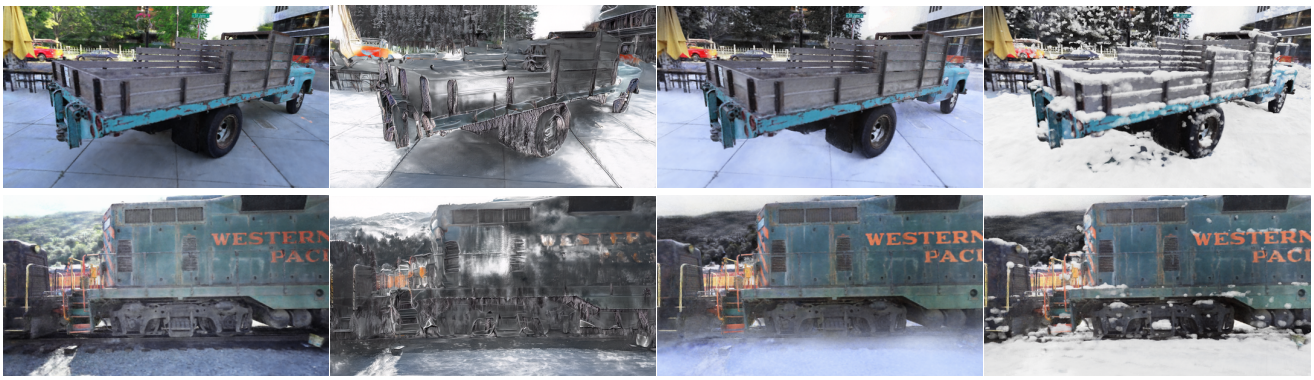
where τ_{snow} is a controllable surface truncation threshold and a is a hyper-parameter. This equation implies a point is more likely to be surface boundary if it's close or larger than the threshold. We use a spatially-varying diffuse color $\mathbf{c}_\phi(\mathbf{x}_i^{(p)})$ (which is close to pure white multiplied by the average illumination of the scene) to approximate BRDF, and apply a subsurface scattering effect [56] to light the snow's shadowed part (further detail in supplementary). Surface normal values are still calculated in a gradient based manner. A visualization of snow rendering is shown in Fig. 6.

4. Experiments

We evaluate ClimateNeRF and show simulated results over various scenes across different climate effects. We compare our results with state-of-the-art 2D synthesis and 3D stylization to show the quality and consistency of rendered frames. Experimental results demonstrate that our method is more realistic and faithful than existing 2D synthesis and NeRF model finetuned on stylized images while we also maintain temporal consistency and physical plausibility. We encourage readers to watch supplementary videos for a better demonstration of our method's quality.

4.1. Experimental Details

Datasets. We conduct experiments on various outdoor scenes: *Playground*, *Family*, *Horse*, *Truck*, and *Train* from *Tanks and Temples* dataset [38], *Garden* from *Mip-NeRF360* [3] and *Seq 00* from *KITTI-360* dataset [44]. These testing scenes vary significantly regarding scales, contents, layouts, and viewpoints.



Original | Swapping Autoencoder [57] | 3D Stylization | Ours

Figure 9. **Snow simulation comparison.** Swapping-Autoencoder [57] captures appearance changes but ignores the geometry of both truck and train. 3D Stylization preserves the geometry of original scene well but doesn't accumulate snow on horizontal surfaces. In contrast, ClimateNeRF has convincing snow accumulation both on ground and on objects. Note small snow accumulations on the bogies and running board on the train and the boards and bonnet of the truck.

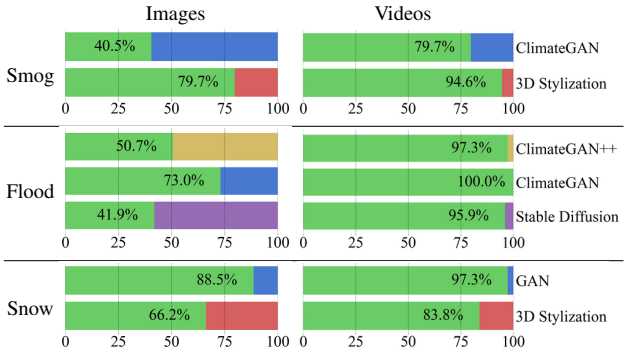


Figure 10. **User Study.** The length of bars indicates the percentage of users voting for higher realism than the opponents. The green bar with the number shows our win rate against each baseline. The video quality of our method significantly outperforms all baselines.

$t = 0$ | $t = 80$ | $t = 160$

Figure 11. **View Consistency Comparison.** We show flood simulation in Playground scene and time step t increases from left to right. ClimateGAN++ [66] (Top) cannot generate realistic reflection, Stable Diffusion [61] (Middle) synthesizes different objects in each views. Ours (Bottom) simulate photorealistic and consistent reflection and ripples.

Baselines. We compare ClimateNeRF to the state-of-the-art 2D image editing methods, such as stable diffusion inpainting [61], ClimateGAN [66], as well as state-of-the-art 3D NeRF stylization [90]. For all 2D synthesis approaches, we first build a NeRF using NGP, render at the target view, and conduct synthesis. For all the 3D methods, we re-use the improved version of NGP-based NeRF. **ClimateGAN** [66] uses monocular depth to predict masks and use GAN to inpaint the climate-related effects, including smog and flood; **ClimateGAN++** is an improved version for flood simulation using our method's water mask, yielding better geometry consistency; **Swapping Autoencoder** [57] is a photorealistic 2D style transfer method. We use the model pretrained on Flickr Mountains dataset and Flickr Waterfall dataset [57] for snow. **Stable Diffusion** [61] is the state-of-the-art guided image inpainting method based on latent diffusion model. We feed accurate water masks produced by ClimateNeRF and use text prompts of "flooding" for inpainting. **3D Stylization** leverages FastPhotoStyle [42]. To simulate white snow coverings, we stylize regions labeled *road*, *terrain*, *vegetation*, and *sky* while keeping the geometry. Please refer to supplementary materials for additional implementation details for all competing methods including ours.

4.2. Experimental Results

Qualitative Results Fig. 7 depicts qualitative results from smog simulation. Our method delivers better realism and physical plausibility (see the different transmission levels across foreground and background). ClimateGAN [66] generates visually reasonable results but fails to provide sharp boundaries. Additionally, video results further show our method is better at view consistency.

We also report flood simulation results in Fig. 8. ClimateGAN++ [66] produces waters with wrong reflection and blurry artifacts. Stable Diffusion [61] provides realistic and diverse colors and reflectance but hallucinates additional

Figure 12. **Controllable Simulation** The simulation framework is highly controllable by the users. **Top**: different smog density; **middle**: different flood levels; **bottom**: different snow accumulations; all easily adjusted by a user.

contents (e.g., cars, trees) and lacks view consistency. ClimateNeRF renders accurate reflections with Fresnel effects and simulates realistic water ripples thanks to physical simulation. Video results show that our method is view-consistent and can provide fluid dynamics.

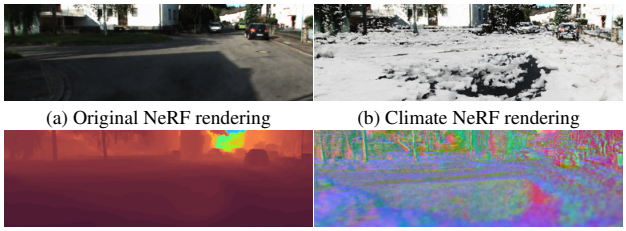
We report snow simulation results in Fig. 9. As the figure shows, 3D Stylization changes the floor texture but cannot add physical entities to the scene, limiting its realism. Swapping Autoencoder [57] changes the overall appearance but hallucinates unrealistic textures (e.g., car texture). On the other hand, ClimateNeRF simulates photorealistic winter effects, including accumulated snow and change of sky and tree colors, etc. ClimateNeRF even piles snow on tiny structures like pedals, as shown in the figure.

User Study We perform a user study to validate our approach quantitatively. Users are asked to watch pairs of synthesized images or videos of the same scene and pick the one with higher realism. 37 users participated in the study, and in total, we collected 2664 pairs of comparisons. Results are reported in Fig. 10. ClimateNeRF has consistently been favored among all video simulation comparisons thanks to its high realism and view consistency. Single image comparison does not consider view consistency. In this case, ClimateNeRF still outperforms most baselines except diffusion models on flooding, which also produces realistic water reflectances. Users find diffusion models tend to produce more reflective water surfaces and diverse ripples.

Controllability. One unique advantage of ClimateNeRF is its controllability. Fig. 12 depicts the changes in smog density, water height, and snow thickness. We show additional controllability results, such as ripple size, flood color, water reflectance, and smog color in supp materials.

Adverse Weather Simulation for Self-Driving. ClimateNeRF is a generic framework and can be applied to any NeRF scene. In Fig. 13, we show that ClimateNeRF simulates climate effects in driving scenes of KITTI-360 [44]. These results demonstrate the potential of applying ClimateNeRF to train and test self-driving autonomy under adverse weather.

Figure 13. **Simulation on Urban Driving Scenes.** ClimateNeRF simulates smog, flood, and snow on a KITTI-360 scene [44]. We anticipate automatically generated severe weather data will enhance the robustness of self-driving to weather conditions.



(a) Original NeRF rendering (b) Climate NeRF rendering

(c) Depth rendering (d) Surface normal rendering

Figure 14. **Limitations.** Snow simulation on KITTI-360 [44] dataset fails to cover a shadowed road due to wrong geometry estimations.

Limitations ClimateNeRF is dependent on the quality of NeRF reconstruction. Inaccurate geometry leads to non-ideal flood and snow simulation. Fig. 14 depicts a case that the incorrect ground surface results in artifacts in snow simulation. This also shows a future opportunity to automatically spot geometry understanding errors through physical simulation.

5. Conclusion

We propose a novel NeRF editing framework that applies physical simulation to NeRF models of scenes. Leveraging this framework, we build ClimateNeRF, allowing us to render realistic climate change effects, including smog, flood, and snow. Our synthesized videos are realistic, view-consistent, physically plausible, and highly controllable. We demonstrate the potential of ClimateNeRF to help raise climate change awareness in the community and enhance self-driving robustness to adverse weather conditions.

References

[1] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan++: How to edit the embedded images? In *CVPR*, 2020. 3

[2] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. 3

[3] 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. 2, 3, 6, 13, 14

[4] Beer. Bestimmung der absorption des rothen lichts in farbigen flüssigkeiten. *Annalen der Physik und Chemie*, 1852. 5

[5] Sagie Benaim, Frederik Warburg, Peter Ebert Christensen, and Serge Belongie. Volumetric disentanglement for 3d scene manipulation. *in arXiv*, 2022. 3

[6] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T. Barron, Ce Liu, and Hendrik P.A. Lensch. Nerd: Neural reflectance decomposition from image collections. In *ICCV*, 2021. 3, 4, 13

[7] Mark Boss, Varun Jampani, Raphael Braun, Ce Liu, Jonathan Barron, and Hendrik Lensch. Neural-pil: Neural pre-integrated lighting for reflectance decomposition. *NeurIPS*, 2021. 3

[8] NASA Global Climate Change. Vital signs of the planet. *URL: https://climate. nasa. gov/vital-signs/global-temperature/(data obrashhenija: 25.12. 2019)*, 2018. 2

[9] Daniel A. Chapman, Adam Corner, Robin Webster, and Ezra M Markowitz. Climate visuals: A mixed methods investigation of public perceptions of climate images in three countries. *Global Environmental Change*, 2016. 1, 2

[10] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. *ECCV*, 2022. 2, 4, 13

[11] Xingyu Chen, Qi Zhang, Xiaoyu Li, Yue Chen, Ying Feng, Xuan Wang, and Jue Wang. Hallucinated neural radiance fields in the wild. In *CVPR*, 2022. 14

[12] Yaosen Chen, Qi Yuan, Zhiqiang Li, Yuegen Liu Wei Wang Chaoping Xie, Xuming Wen, and Qien Yu. Upst-nerf: Universal photorealistic style transfer of neural radiance fields for 3d scene. *in arXiv*, 2022. 3

[13] Pei-Ze Chiang, Meng-Shiun Tsai, Hung-Yu Tseng, Wei-Sheng Lai, and Wei-Chen Chiu. Stylizing 3d scene via implicit representation and hypernetwork. In *WACV*, 2022. 3

[14] 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 *ECCV*, 2022. 3

[15] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 2

[16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 13

[17] Gautier Cosne, Adrien Juraver, Mélisande Teng, Victor Schmidt, Vahe Vardanyan, Alexandra Luccioni, and Yoshua Bengio. Using simulated data to generate images of climate change. *ICLR Workshop*, 2020. 3

[18] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *NeurIPS*, 2014. 14

[19] Epic Games. Unreal engine. 2

[20] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001. 3

[21] Bryan E Feldman and James F O'Brien. Modeling the accumulation of wind-driven snow. In *ACM SIGGRAPH conference abstracts and applications*, 2002. 2, 3

[22] Alain Fournier and William T Reeves. A simple model of ocean waves. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986. 3

[23] M. Fox. *Optical Properties of Solids (2 ed.)*. Oxford University Press, 2010. 5

[24] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 3

[25] Erik Glaas, Anne Gammelgaard Ballantyne, Tina-Simone Neset, and Björn-Ola Linnér. Visualization for supporting individual climate change adaptation planning: Assessment of a web-based tool. *Landscape and Urban Planning*, 2017. 1, 2

[26] Simon Green. Real-time approximations to subsurface scattering. *GPU Gems*, 1:263–278, 2004. 15

[27] John K Haas. A history of the unity game engine. 2014. 2

[28] Martin Hahner, Dengxin Dai, Christos Sakaridis, Jan-Nico Zaech, and Luc Van Gool. Semantic understanding of foggy scenes with purely synthetic data. In *IEEE*

Intelligent Transportation Systems Conference (ITSC), 2019. 3

[29] Jamie Herring, Matthew S VanDyke, R Glenn Cummins, and Forrest Melton. Communicating local climate risks online through an interactive data visualization. *Environmental Communication*, 2017. 1, 2

[30] Ove Hoegh-Guldberg, Daniela Jacob, M Bindi, S Brown, I Camilloni, A Diedhiou, R Djalante, K Ebi, F Engelbrecht, J Guiot, et al. Impacts of 1.5 c global warming on natural and human systems. *Global warming of 1.5 C. An IPCC Special Report*, 2018. 2

[31] Christopher J Horvath. Empirical directional wave spectra for computer graphics. In *Proceedings of the 2015 Symposium on Digital Production*, 2015. 2, 3, 6

[32] Liming Jiang, Changxu Zhang, Mingyang Huang, Chunxiao Liu, Jianping Shi, and Chen Change Loy. Tsit: A simple and versatile framework for image-to-image translation. In *ECCV*, 2020. 3

[33] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986. 5, 15

[34] James T Kajiya and Brian P Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH computer graphics*, 1984. 4

[35] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019. 3

[36] Leonid Keselman and Martial Hebert. Approximate differentiable rendering with algebraic surfaces. In *ECCV*, 2022. 6

[37] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM TOG*, 2017. 2

[38] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM TOG*, 2017. 6, 14

[39] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. In *NeurIPS*, 2022. 3

[40] J.H. Lambert. *Photometria sive de mensura et gradibus luminis, colorum et umbrae*. Eberhardt Klett, 1760. 5

[41] Zoe Leviston, Jennifer Price, and Brian Bishop. Imagining climate change: The role of implicit associations and affective psychological distancing in climate change responses. *European Journal of Social Psychology*, 2014. 1, 2

[42] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. In *ECCV*, 2018. 4, 7, 18

[43] Zhengqin Li, Jia Shi, Sai Bi, Rui Zhu, Kalyan Sunkavalli, Miloš Hašan, Zexiang Xu, Ravi Ramamoorthi, and Manmohan Chandraker. Physically-based editing of indoor scene lighting from a single image. *ECCV*, 2022. 3

[44] 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. 2, 6, 8, 14

[45] 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, 3

[46] Huan Ling, Karsten Kreis, Daiqing Li, Seung Wook Kim, Antonio Torralba, and Sanja Fidler. Editgan: High-precision semantic image editing. In *NeurIPS*, 2021. 3

[47] Difan Liu, Sandesh Shetty, Tobias Hinz, Matthew Fisher, Richard Zhang, Taesung Park, and Evangelos Kalogerakis. Asset: autoregressive semantic scene editing with transformers at high resolutions. *ACM TOG*, 2022. 3

[48] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020. 3

[49] Xiaohu Lu, Jian Yaoy, Haoang Li, Yahui Liu, and Xiaofeng Zhang. 2-line exhaustive searching for real-time vanishing point estimation in manhattan world. In *WACV*. IEEE, 2017. 6, 17

[50] Julien NP Martel, David B Lindell, Connor Z Lin, Eric R Chan, Marco Monteiro, and Gordon Wetzstein. Acorn: Adaptive coordinate networks for neural scene representation. *ACM TOG*, 2021. 3

[51] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *CVPR*, 2021. 13

[52] Henrique MT Menegaz, João Y Ishihara, Geovany A Borges, and Alessandro N Vargas. A systematization of the unscented kalman filter theory. *IEEE Transactions on automatic control*, 2015. 6, 15

[53] 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, 3, 13

[54] Thomas Müller. tiny-cuda-nn, 4 2021. 14

[55] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM TOG*, 2022. 1, 2, 3, 4, 13

[56] Tomoyuki Nishita, Hiroshi Iwasaki, Yoshinori Dobashi, and Eihachiro Nakamae. A modeling and rendering method for snow by using metaballs. In *Computer Graphics Forum*, 1997. 3, 6, 15

[57] Taesung Park, Jun-Yan Zhu, Oliver Wang, Jingwan Lu, Eli Shechtman, Alexei Efros, and Richard Zhang. Swapping autoencoder for deep image manipulation. *NeurIPS*, 2020. 3, 7, 8, 21

[58] Chen Quei-An. ngp_pl: a pytorch-lightning implementation of instant-ngp, 2022. 14

[59] 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. 14

[60] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *ICCV*, 2021. 3

[61] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 2, 3, 6, 7, 18, 20

[62] Viktor Rudnev, Mohamed Elgharib, William Smith, Lingjie Liu, Vladislav Golyanik, and Christian Theobalt. Neural radiance fields for outdoor scene relighting. *ECCV*, 2022. 3

[63] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. *in arXiv*, 2022. 3

[64] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *in arXiv*, 2022. 3

[65] Victor Schmidt, Alexandra Luccioni, S Karthik Mukkavilli, Narmada Balasooriya, Kris Sankaran, Jennifer Chayes, and Yoshua Bengio. Visualizing the consequences of climate change using cycle-consistent adversarial networks. *ICLR*, 2019. 2

[66] Victor Schmidt, Alexandra Sasha Luccioni, Mélisande Teng, Tianyu Zhang, Alexia Reynaud, Sunand Raghupathi, Gautier Cosne, Adrien Juraver, Vahe Vardanyan, Alex Hernandez-Garcia, et al. Climategan: Raising climate change awareness by generating images of floods. *ICLR*, 2022. 2, 3, 5, 6, 7, 18, 19, 20

[67] Pratul P Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T Barron. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *CVPR*, 2021. 3, 4, 13

[68] Michael Stokes. A standard default color space for the internet-srgb. *http://www. color. org/contrib/sRGB. html*, 1996. 16

[69] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *ACM TOG*, 2013. 2, 3

[70] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 3, 4, 13, 14

[71] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Improved direct voxel grid optimization for radiance fields reconstruction. *at arXiv*, 2022. 13

[72] 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. 2

[73] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. Optimized spatial hashing for collision detection of deformable objects. In *Vmv*, 2003. 4, 13

[74] Jerry Tessendorf. Simulating ocean water. In *SIGGRAPH*, 2001. 3

[75] Kees van Kooten, Gino van den Bergen, and Alex Telea. *Point-based visualization of metaballs on a gpu*. Addison-Wesley Longman, 2007. 6

[76] 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. *CVPR*, 2022. 3

[77] 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. 4, 13

[78] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium*, 2000. 6, 15

[79] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Clip-nerf: Text-and-image driven manipulation of neural radiance fields. In *CVPR*, 2022. 3

[80] Yilin Wang, Junjie Ke, Hossein Talebi, Joong Gon Yim, Neil Birkbeck, Balu Adsumilli, Peyman Milanfar, and Feng Yang. Rich features for perceptual quality assessment of ugc videos. In *CVPR*, 2021. 18, 19, 22

[81] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn.

Nex: Real-time view synthesis with neural basis expansion. In *CVPR*, 2021. 3

[82] Liwen Wu, Jae Yong Lee, Anand Bhattad, Yu-Xiong Wang, and David Forsyth. Diver: Real-time and accurate neural radiance fields with deterministic integration for volume rendering. In *CVPR*, 2022. 2, 3

[83] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In *NeurIPS*, 2021. 4, 13

[84] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *CVPR*, 2022. 3

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

[86] Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2014. 22

[87] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 3

[88] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *at arXiv*, 2022. 13, 14

[89] 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 *CVPR*, 2022. 3

[90] Kai Zhang, Nick Kolkin, Sai Bi, Fujun Luan, Zexiang Xu, Eli Shechtman, and Noah Snavely. Arf: Artistic radiance fields. *ECCV*, 2022. 2, 3, 7, 18, 22

[91] Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. Physg: Inverse rendering with spherical gaussians for physics-based material editing and relighting. In *CVPR*, 2021. 3

[92] Qiang Zhang, Seung-Hwan Baek, Szymon Rusinkiewicz, and Felix Heide. Differentiable point-based radiance fields for efficient view synthesis. *in arXiv*, 2022. 3

[93] 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. 3

[94] Shuaifeng Zhi, Tristan Laidlow, Stefan Leutenegger, and Andrew Davison. In-place scene labelling and understanding with implicit scene representation. In *ICCV*, 2021. 13

[95] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017. 3

[96] Peiye Zhuang, Oluwasanmi Koyejo, and Alexander G Schwing. Enjoy your editing: Controllable gans for image editing via latent space navigation. In *ICLR*, 2021. 3

[97] Károly Zsolnai-Fehér. The flow from simulation to reality. *Nature Physics*, 2022. 2

[98] Yiming Zuo and Jia Deng. View synthesis with sculpted neural points. *in arXiv*, 2022. 3

Supplementary Material

In this supplementary material, we describe implementation details of the simulation framework in Sect. A; and provide ablation study to validate our design choices in Sect. B. ClimateNeRF is highly controllable and is demonstrated in Sect C; and more qualitative and quantitative results are included in Sect. D and Sect. E, respectively. Please also refer to our project page for more interactive demonstrations.

A. Implementation Details

3D Scene Reconstruction We use spatial hashing grid [55] to represent the 3D scene. The entire space contains a multi-resolution feature grid $\{\text{enc}(\mathbf{x};\theta^l)\}_{l=1}^{L}$, where L is the total number of resolution levels; θ^l are learnable parameters at each level. For a point $\mathbf{x}$, we index grids by a spatial hash function [73] and fetch feature by interpolation and concatenation: $\gamma = \text{cat}\{\text{interp}(\mathbf{x}, \text{enc}(\mathbf{x};\theta^l))\}_{l=1}^{L}$. In order to keep spatial features intact at the stylization stage and separate gradient flows between geometry and color, we develop a disentangled version of instant-NGP inspired by [10, 70]. Each voxel maintains a geometry code γ_σ and appearance code γ_{app}. γ_σ encodes opacity and γ_{app} contains color, semantic and normal information:

$$\gamma_\sigma = \text{cat}\{\text{interp}(\mathbf{x}, \text{enc}(\mathbf{x};\theta_\sigma^l))\}_{l=1}^{L_\sigma}, \quad \gamma_{\text{app}} = \text{cat}\{\text{interp}(\mathbf{x}, \text{enc}(\mathbf{x};\theta_{app}^l))\}_{l=1}^{L_{app}} \tag{8}$$

where interp stands for linear interpolation, cat denotes concatenation.

We use shallow MLPs to predict densities σ, colors $\mathbf{c}$, semantic logits $\mathbf{s}$ and surface normal values $\mathbf{n}$ respectively from geometry features γ_σ and appearance features γ_{app}. We also incorporate appearance embeddings $\{\ell_i^{(a)}\}_{i=1}^{N}$ [51] to balance different lighting conditions across images. With hash grids and MLPs, we have the following function to reconstruct the original scene:

$$(\sigma, \mathbf{c}, \mathbf{s}, \mathbf{n}) = F_\theta(\mathbf{x}, \mathbf{d}, \ell_i^{(a)}, \gamma_\sigma, \gamma_{\text{app}}) \tag{9}$$

Following volume rendering and alpha blending [53], we render the color $C(\mathbf{r})$ for ray $\mathbf{r}$ and its semantic logit $S(\mathbf{r})$ [94].

$$C(\mathbf{r}) = \sum_{i=1}^{N} T_i(1-\exp(-\sigma_i\delta_i))\mathbf{c}_i; \quad S(\mathbf{r}) = \sum_{i=1}^{N} T_i(1-\exp(-\sigma_i\delta_i))\mathbf{s}_i, \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right) \tag{10}$$

We apply softmax to semantic logits $S(\mathbf{r})$ to obtain semantic probabilities $\{p(\mathbf{r})^l\}_{l=1}^{L}$ for all labels. During training, we perform MSE loss L_C and cross-entropy loss L_S for rendered colors and semantic logits using ground truth color $C_{gt}(\mathbf{r})$ and 2D semantic logits $\{\hat{p}(\mathbf{r})^l\}_{l=1}^{L}$ predicted by segformer [83] pretrained on cityscape dataset [16]. Moreover, we detach densities σ when rendering $S(\mathbf{r})$ since we leverage pseudo-semantic labels. Like Ref-NeRF [77], we use the density gradient normals $\hat{\mathbf{n}} = -\frac{\nabla\sigma}{\|\nabla\sigma\|}$ [6, 67] to guide the predicted surface normals $\mathbf{n}$ using a weighted MSE loss.

$$L_C = \sum_{\mathbf{r}\in\mathcal{R}} \|C(\mathbf{r}) - C_{gt}(\mathbf{r})\|_2^2 \tag{11}$$

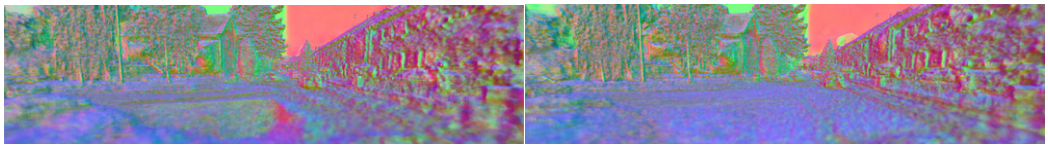
$$L_S = -\sum_{\mathbf{r}\in\mathcal{R}}[\sum_{l\in L}\hat{p}(\mathbf{r})^l \log p(\mathbf{r})^l] \tag{12}$$

$$L_{\mathbf{n}} = \sum_{\mathbf{r}\in\mathcal{R}}\sum_{i=N}^{N} w_i\|\mathbf{n_i} - \hat{\mathbf{n}}_i\|_2^2 \tag{13}$$

where $w_i = T_i(1-\exp(-\sigma_i\delta_i))$ denotes the detached weight in Eq.10. We also leverage distortion loss L_{dist} [3, 71] to mitigate floaters in the reconstruction results:

$$L_{\text{dist}} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} w_i w_j \left|\frac{t_i+t_{i+1}}{2} - \frac{t_j+t_{j+1}}{2}\right| + \frac{1}{3}\sum_{i=0}^{N-1} w_i^2\,(t_{i+1}-t_i) \tag{14}$$

However, ngp model tends to create big 'blobs' in the sky with distortion loss. We alleviate this by applying a simple penalty $L_{sky} = \sum_{\mathbf{r}\in\mathcal{R}} \mathrm{e}^{-D(\mathbf{r})}\cdot\mathbb{1}\{\hat{p}(\mathbf{r}) = \hat{p}_{sky}\}$ where $D(\mathbf{r})$ denotes the depth following Eq. 10 [88] and $\mathbb{1}\{\cdot\}$ is an indicator function



(a) w/o monocular dense depth D_{mono} (b) w/ monocular dense depth D_{mono}

Figure 15. **Ablation on monocular dense depth supervision** Normal estimations in (b) shows that leveraging monocular dense depth removes artifacts on the road.

using 2D predicted semantic logits $\hat{p}$. Moreover, we incorporate opacity loss $L_O = -\sum_{\mathbf{r}\in\mathcal{R}} O(\mathbf{r}) \log O(\mathbf{r})$ [58] to encourage ray opacity being either 0 or 1 to avoid semi-transparent regions in reconstruction results. During our training time, our model's total loss is a weighted sum of aforementioned losses:

$$L = L_C + \lambda_S L_S + \lambda_{\mathbf{n}} L_{\mathbf{n}} + \lambda_{dist} L_{dist} + \lambda_{sky} L_{sky} + \lambda_O L_O \tag{15}$$

Transient Object Occlusion In order to occlude transient objects like pedestrians or vehicles across views in tanks and temples dataset [38], we follow [11] and create per-image learnable masks:

$$M_i(u, v) = F_\psi(u, v, i, \gamma_M) \tag{16}$$

where $(u, v) \in \mathbb{R}^2$, i denotes image coordinate and image index in all training images. F_ψ denotes a shallow MLP and γ_M is the output of hash grids.

In such case, we change color reconstruction loss 11 following [11]:

$$L_C = \sum_{\mathbf{r}\in\mathcal{R}} M(\mathbf{r}) \|C(\mathbf{r}) - C_{gt}(\mathbf{r})\|_2^2 + \lambda_M (1 - M(\mathbf{r})) \tag{17}$$

where the second term is used to prevent the mask from predicting everything transient.

Geometry improvements As mentioned in the limitation subsection in the main paper, ClimateNeRF strongly relies upon high-quality geometry. To improve geometry estimations for KITTI-360 dataset [44], we further leverage the monocular dense depth D_{mono} and depth loss:

$$L_{\text{depth}} = \sum_{\mathbf{r}\in\mathcal{R}} \|((wD(\mathbf{r}) + d) - D_{\text{mono}}(\mathbf{r}))\|_2^2 \tag{18}$$

where w and d are used to align the predicted depth from NeRF $D(\mathbf{r})$ and monocular depth cue $D_{\text{mono}}(\mathbf{r})$ with a least-square criterion [18, 59, 88]. As can be seen in Fig.15, holes on the ground are "filled" due to supervisions from monocular dense depth.

Training Details Our implementations of the hash grids and distortion loss follow [54, 58]. For the scale and resolution of hash grid in our model, we get the inspiration from [70] where appearance features allocate more grids with finer resolutions. Geometry hash grids have 16 levels and 2^{19} entries at most per level while appearance hash grids have 32 levels and 2^{21} entries at most per level. For Tanks and Temples dataset [38] and MipNeRF-360 dataset [3], side length of hash grids is 16. Geometry MLP and appearance MLP both have 128 neurons per layer and 1 hidden layer while semantic MLP and normal predicting MLP has 32 neurons per layer and 1 hidden layer. For more details about network architecture, we recommend readers read supplementary materials. To balance between losses at reconstruction stage, we set the weights of image reconstruction loss and cross-entropy loss to 1 and 4^{-2} respectively while weights for opacity loss λ_O, normal loss $\lambda_{\mathbf{n}}$, distortion loss λ_{dist} and sky loss λ_{sky} are 2^{-4}, 7^{-4}, 3^{-4} and 1^{-1} respectively.

A.1. Flood Simulation

Given that water is mostly muddy and non-transparent, we approximate the opacity by simply checking a point above or under the water's surface:

$$O_\phi(\mathbf{x}; F'_\theta) = \begin{cases} \infty & \text{if } \mathbf{n}_w(\mathbf{x} - \mathbf{o}_w) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

Water is highly reflective, yet the microfacet ripples sometimes make the water look glossy. To simulate these effects, we leverage a Spherical Gaussian (SG) to approximate the BRDF on the reflective water surface:

$$B_\phi(\mathbf{x}, -\mathbf{d}, \boldsymbol{\omega}_i, N_\phi(\mathbf{x})) = \exp^{\lambda(-\boldsymbol{\omega}_i \cdot \mathbf{d}_r - 1)} \tag{20}$$

where SG lobe axis is $\mathbf{d}_r = \mathbf{d} - 2(\mathbf{d} \cdot N_\phi(\mathbf{x}))N_\phi(\mathbf{x})$,, and $\lambda \in \mathbb{R}_+$ is the lobe sharpness, controlling the glossy effects.

We use sigma point-based sampling for rendering water. Specifically, the camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ is cast from the camera and hits the water surface at position $\mathbf{x}$. The observed color is decided by a sampling-based approximation of the rendering equation:

$$\mathbf{c}_\phi = (1 - R)\mathbf{c}_w + R \sum_{i=1}^{5} L(\mathbf{x}, \boldsymbol{\omega}_i) e^{\lambda(-\boldsymbol{\omega}_i \cdot \mathbf{d}_r - 1)}, \tag{21}$$

where $L(\mathbf{x}, \boldsymbol{\omega}_i) = C(\mathbf{r})$ is the NeRF ray color (Eq. 10), representing the incident light color hitting $\mathbf{x}$ along direction $\boldsymbol{\omega}_i$. $R \in (0, 1)$ is the reflectance index determined by viewing direction d and normal $N_\phi(\mathbf{x})$, which enables the system to simulate the Fresnel effect on the water. To approximate the integral in rendering equation [33], we adopt the sigma-point method [52,78] and sample 5 rays from $\mathbf{x}$, including reflection direction $\mathbf{d}_r$ and nearby four rays. In short, ClimateNeRF simulates Fresnel effect, glossy reflection, and wave dynamics.]

Fresnel Effect When the light hits the water surface, the amount of reflection and transmission is determined by the incident and normal directions and is described by the Fresnel effect. The angle between normal and incident rays is denoted by θ_i, and the angle between normal and refracted ray in water is θ_t. According to Snell's Law: $r_i\theta_i = r_t\theta_t$, where $r_i = 1$ is the refraction index of air and $r_t = 1.33$ is the refraction index of water in our experiments, which is also consistent with real-world water properties. Next, the reflectance R in Eq. 21 is computed by:

$$R = \frac{R_s + R_p}{2}, \quad R_s = \left[\frac{\sin(\theta_t - \theta_i)}{\sin(\theta_t + \theta_i)}\right]^2, \quad R_p = \left[\frac{\tan(\theta_t - \theta_i)}{\tan(\theta_t + \theta_i)}\right]^2 \tag{22}$$

where R_s and R_p are the reflectance for s-polarized light and p-polarized light respectively. Modeling the Fresnel effect in our flood simulation pipeline makes the water far from the camera (larger θ_i) have higher R and looks more like a mirror; and the water nearby (smaller θ_i) has lower R and shows watercolor, which enhances the realism of the simulation.

A.2. Snow Simulation

For any point $\mathbf{x}$ in the space, we calculate the snow's density of $\mathbf{x}$ in a particle-based manner. We first figure out a set of N particles as metaballs' centers $\{\mathbf{x}_i^{(p)}\}_{i=1}^N$ with densities $\{\sigma_i^{(p)}\}_{i=1}^N$ and metaball radius $\{R_i^{(p)}\}_{i=1}^N$ around $\mathbf{x}$. Then we sum up the densities calculated by kernel function $\mathbf{K}(r, R, \sigma_o)$.

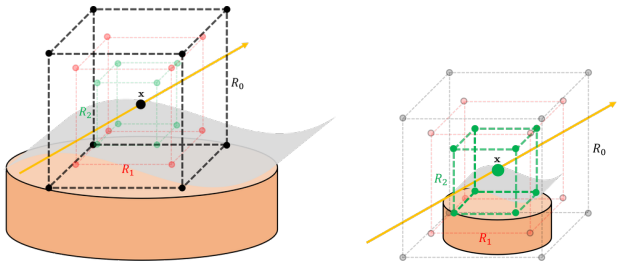
$$\begin{aligned} \sigma_{\text{snow}}(\mathbf{x}) &= \sum_{i=1}^{N} \sigma_{\mathbf{K}}(\mathbf{x}, \mathbf{x}_i^{(p)}), \\ \text{where } \sigma_{\mathbf{K}}(\mathbf{x}, \mathbf{x}_i^{(p)}) &= \mathbf{K}(\|\mathbf{x} - \mathbf{x}_i^{(p)}\|_2, R_i^{(p)}, \sigma_i^{(p)}) \end{aligned} \tag{23}$$

where $\sigma_i^{(p)}$ is defined by weights during volume rendering 10 for σ_θ of F'_θ. More details are shown in Section A.3. During rendering, we identify snow surface by a threshold τ_{snow} and a hyperparameter a:

$$O_\phi(\mathbf{x}; F'_\theta) = \frac{1}{1 + e^{-a(x\sigma_{\text{snow}} - \tau_{\text{snow}})}} \sigma_{\text{snow}}, \tag{24}$$

The BRDF of snow particles is set as spatially-varying diffuse color $\mathbf{c}_\phi(\mathbf{x}_i^{(p)})$ close to pure white multiplied by the average illumination of the scene. Furthermore, since the snow is semi-transmissive, the subsurface scattering effect [56] will light the snow's shadowed part. To simulate such effect, we leverage warp lighting function [26] $\Phi(\mathbf{n}_\mathbf{K}, \mathbf{n}_l, \gamma_\Phi)$ based on normalized surface normal $\mathbf{n}_\mathbf{K}$, light vector $\mathbf{n}_l$ and hyperparameter γ_Φ. For an arbitrary point $\mathbf{x}$ in space, the color of point $\mathbf{x}$ is a weighted sum of $\{\mathbf{c}_i^{(p)} \in \mathbb{R}^1\}_{i=1}^N$ based on kernel function:

$$\mathbf{c}_\phi(\mathbf{x}) = \frac{\sum_{i=1}^N \sigma_{\mathbf{K}}(\mathbf{x}, \mathbf{x}_i^{(p)}) \frac{\mathbf{c}_i^{(p)} + \mathbf{c}_0}{1 + \mathbf{c}_0}}{\sum_{i=1}^N \sigma_{\mathbf{K}}(\mathbf{x}, \mathbf{x}_i^{(p)})} \Phi(\mathbf{n}_\mathbf{K}(\mathbf{x}), \mathbf{n}_l, \gamma_\Phi) \tag{25}$$



(a) Sample near a large surface (b) Sample near a tiny surface

Figure 16. **Visualization of voxel+nest sampling for $N_n = 3$.** Instead of storing metaball information in point cloud, we approximate metaball's center density distributions with G_{ϕ_w}. When $\mathbf{x}$ is close to a large surface (shown in (a)), snow surface induced by the largest voxel's 8 vertices will dominate the surface at $\mathbf{x}$. When $\mathbf{x}$ is close to a tiny surface (shown in (b)), only snow surface induced by the smallest voxel's 8 vertices contributes the surface at $\mathbf{x}$.

where $\Phi(\mathbf{n}_\mathbf{K}(\mathbf{x}), \mathbf{n}_l, \gamma_\Phi) = \frac{\mathbf{n}(\mathbf{x})\cdot\mathbf{n}_l+\gamma_\Phi}{1+\gamma_\Phi}$ and $\frac{\mathbf{c}_i^{(p)}+\mathbf{c}_0}{1+\mathbf{c}_0}$ is used to approximate a high albedo for snow and $\mathbf{c}_0$ is a hyperparameter. Surface normal values are still calculated in a gradient-based manner.

A.3. Extensions

Bake Editing When simulating physical entities ,especially snow, rendering will be time-consuming if we straightly let snow fall from the sky and do collision detection. Moreover, due to a lack of supervision in a bird's eye view, depth estimation for rays cast from the sky is not accurate. To mitigate the aforementioned issues, we fit the distribution of metaballs' densities and colors in a new model and fetch them in a particle-based manner. The new model outputs high densities where metaballs locate. We use $w_i = T_i(1 - \exp(-\sigma_i\delta_i))$ in Eq. 10 for surface detection since $w(\mathbf{x}) \in [0, 1]$ is close to 1 when $\mathbf{x}$ is close to surfaces. To identify surfaces where snow accumulates, we incorporate surface normals $\mathbf{n}$ and vertical axis $\mathbf{n}_\perp$ to figure out metaballs' density weights: $w_i^{(p)} = \frac{1}{1+e^{-a'(\mathbf{n}_i\cdot\mathbf{n}_\perp-\cos(\theta_0))}} w_i$ where θ_0 is a hyperparameter. We then bake $w_i^{(p)}$ into a new model G_{ϕ_w}:

$$w_{\phi_w}^{(p)}(\mathbf{x}) = G_{\phi_w}(\mathbf{x}) \tag{26}$$

We also bake the gray scale [68] of $\mathbf{c}_\theta$ from F_θ into a new model $G_{\phi_\mathbf{c}}$ to capture an approximation for light intensities and shadows:

$$\mathbf{c}_{\phi_\mathbf{c}}^{(p)}(\mathbf{x}) = G_{\phi_\mathbf{c}}(\mathbf{x}) \tag{27}$$

Then, we leverage the pre-trained G_{ϕ_w}, $G_{\phi_\mathbf{c}}$ and do voxel sampling to fetch $\{\sigma_i^{(p)}\}_{i=1}^N$ and $\{\mathbf{c}_i^{(p)}\}_{i=1}^N$ from 8 vertices. Also, to automatically alter metaballs' radiuses according to the size of the surface, we sample nested grids with different side lengths defined in a geometric progression. Moreover, we define metaballs' radiuses by girds' side lengths. Hence, Eq. 23 and Eq. 25 can be rewritten as:

$$\sigma_{\text{snow}}(\mathbf{x}) = \sum_{i=1}^{8N_n} \sigma_\mathbf{K}(\mathbf{x}, \mathbf{x}_i^{(p)}); \quad \mathbf{c}_\phi(\mathbf{x}) = \frac{\sum_{i=1}^{8N_n} \sigma_\mathbf{K}(\mathbf{x}, \mathbf{x}_i^{(p)}) \frac{\mathbf{c}_i^{(p)}+\mathbf{c}_0}{1+\mathbf{c}_0}}{\sum_{i=1}^{8N_n} \sigma_\mathbf{K}(\mathbf{x}, \mathbf{x}_i^{(p)})} \Phi(\mathbf{n}_\mathbf{K}(\mathbf{x}), \mathbf{n}_l, \gamma_\Phi) \tag{28}$$

where N_n is the number of nests. We calculate density $\sigma^{(p)}$ and albedo color $\mathbf{c}^{(p)}$ for metaball centered at $\mathbf{x}^{(p)}$ by $\sigma^{(p)} = w_{\phi_w}^{(p)}(\mathbf{x}^{(p)})\sigma_0$ and $\mathbf{c}^{(p)} = \mathbf{c}_{\phi_\mathbf{c}}^{(p)}(\mathbf{x})$ where σ_0 is a hyperparameter. See Fig. 16 for a visualization of this sampling strategy. If stylization is done on the scene, we leverage the stylized model F'_θ and finetune the $G_{\phi_\mathbf{c}}$ to match new illumination conditions while remaining G_{ϕ_w} intact since F'_θ shares the same spatial information with F_θ.

Anti-Aliasing When rendering with simulation, the high-frequency normal map changes on the physical entity surface would lead to an aliasing effect. To alleviate such artifacts, we can render four times larger images with higher resolution, and perform anti-aliasing downsampling to the original resolution.

B. Ablation Study

To justify our design choices, we perform an ablation study of flood simulation, and the results are shown in Fig 17. Specifically, we report the simulation results without certain technical components depicted in Sec. A.1 of the main paper.

(a) Inaccurate plane geometry (b) No wave simulation (c) No Fresnel effect

(d) No glossy effect (e) No anti-aliasing (f) Full simulation

Figure 17. **Ablation Study of Flood Simulation.** (a) Without accurate plane geometry estimation with vanishing point detection [49], the water surface deviates from gravity direction. (b) The surface is perfect planar without wave simulation, which is not natural. (c) Fresnel effect makes the water far from the camera (with a larger incident angle) have higher reflectance, and is consistent with physical rules. (d) The glossy effect makes the reflection more blurry and realistic (e) There is much high-frequency noise around the water border without an anti-aliasing trick. (f) Our full flood simulation results.

Pure White Snow Only spatial variant colors Only diffuse model Full model

Figure 18. **Ablation Study for Snow Simulation** Though spatial variant colors capture local illumination conditions, they fail to offer a sense of depth for snow. Moreover, the diffuse model cannot simulate snow's scattering effects.

Fig 17 shows that all components are essential for realism. For example, vanishing point detection [49] makes the water plane follow gravity direction; wave simulation adds ripples to the water surface; the Fresnel effect makes the water reflectance view-dependent and physically plausible; the Glossy effect mimics realistic microfacet water surfaces with ripples; anti-aliasing removes far-away high-frequency noises. In short, all components contribute to the realism of the simulation.

We also perform an ablation study on snow simulation to validate our approximate scattering rendering in Fig. 18. We compare 1) pure white metaballs with spatial variant colors, 2) metaballs in a fully diffuse model, and 3) our full simulation. Results demonstrate that our choice provides a more realistic rendering of accumulated snow.

C. Controllability

We further demonstrate that ClimateNeRF is highly controllable during the simulation process. In Fig. 19, our method simulate different colors of smog and flood, varying spatial frequency of water ripples, and distinct heights of accumulated

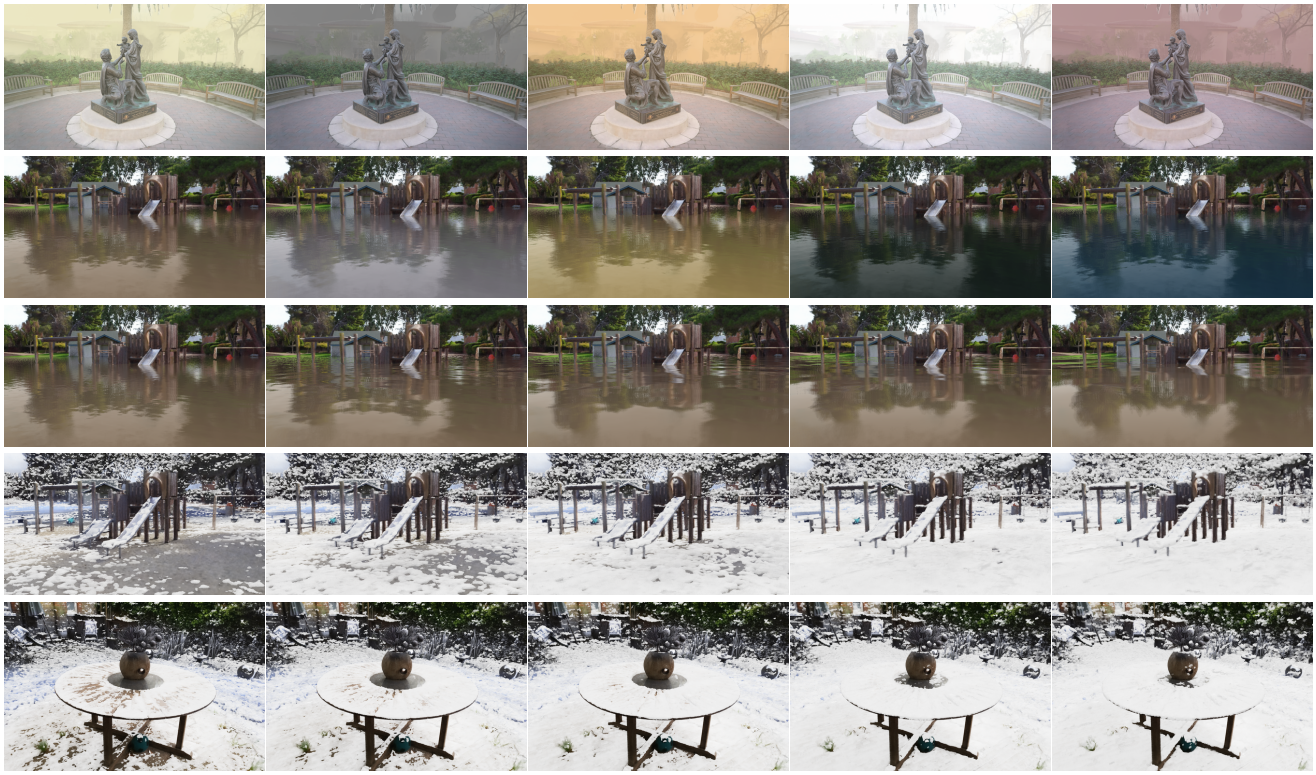


Figure 19. **Controllable Climate Simulation.** From the top row to the bottom row, we control (1) smog color (2) watercolor (3) spatial frequency of the wave. The frequency decreases from left to right. (4) snow height. We control snow height by adjusting the threshold density τ_{snow}.

snow. The results show that our simulation framework is highly controllable by the users. Consequently, scientists can use this framework to simulate accurate climate conditions depending on the projected climate in the future and visualize the consequences corresponding to different actions taken by policymakers and the general public.

D. Qualitative Results

We demonstrate more qualitative results is Fig. 20, Fig. 21, Fig. 22, and Fig. 23. For smog scene images in Fig. 20, ClimateGAN [66] generates visually plausible results but fails to provide sharp boundaries, and 3D stylization attempts to change the surface texture but makes the images overall darker. Our method simulates realistic visibility reduction effects caused by smog, thanks to the geometry reconstruction.

The flood images are shown in Fig. 21. ClimateGAN++ [66] cannot reconstruct realistic reflection on the water surface, Stable Diffusion [61] synthesize realistic water appearance but also produce random objects (e.g., cars, signs) in the scene, which is not consistent across views. ClimateNeRF simulates realistic reflection and water ripples while being view-consistent. This is better demonstrated in the supp video and website

We also compare our FastPhotoStyle [42] based stylization method with Artistic Radiance fields [90]. As shown in Fig. 24, we sustain more appearance details from the original scene.

E. Quantitative Results

No automatic quantitative score can holistically evaluate the quality of our weather-simulated movies. In this project, we evaluate the synthesized videos with the state-of-the-art video quality assessment model UVQ [80] and report the results in Table 1. The score ranges between interval $(1, 5)$, where 1 indicates the lowest quality and 5 indicates the highest quality. As the table shows, our smog simulation outperforms all other baselines, while it does not win Stable Diffusion [61] in flood simulation and ClimateGAN [66] in snow simulation. That being said, UVQ prefers sharp videos instead of measuring holistic realism. As shown in Fig. 22, baselines get a better quality score despite providing low-quality snow simulation

Original | ClimateGAN [66] | 3D Stylization | Ours

Figure 20. **Smog simulation comparison.**

results, suggesting UVQ might not be a good metric for our task. Hence, despite demonstrating UVQ [80] results, we want to emphasize that such metrics mainly focus on measuring the amount of low-level degradation (such as blurriness and noise), which cannot faithfully reproduce human evaluation on realism. Having a good video quality score on simulation remains an open topic.

Original | ClimateGAN++ [66] | Stable Diffusion [61] | Ours

Figure 21. **Flood simulation comparison.**

Original | Swapping Autoencoder [57] | 3D stylization | Ours

Figure 22. **Snow simulation comparison.**

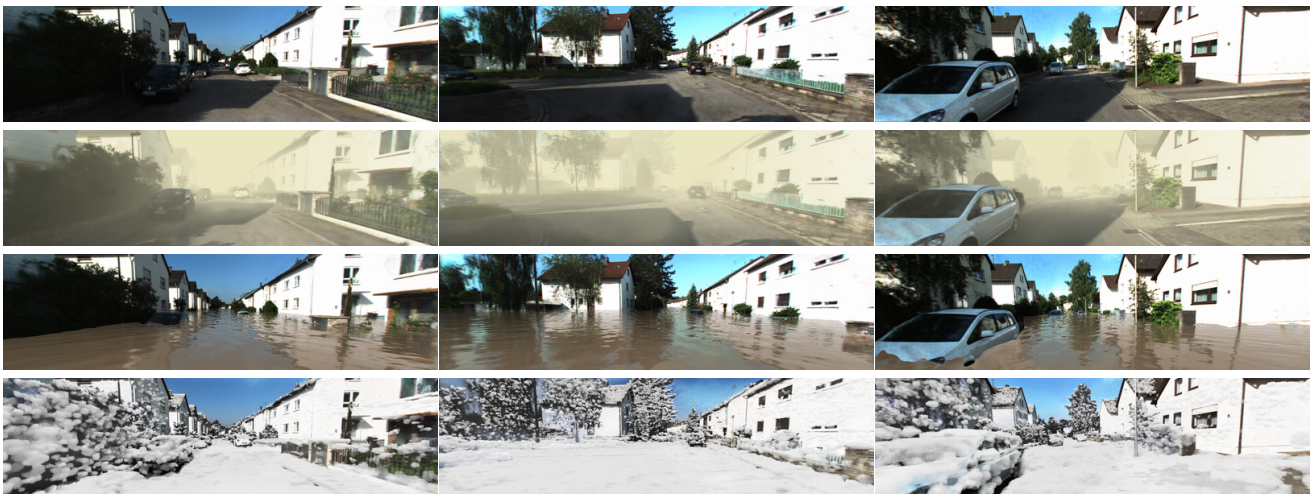


Figure 23. **Simulation on Urban Driving Scenes.**

Style Image from [86] Original ARF [90] 3D stylization

Figure 24. **Comparison with Artistic Radiance Fields [90]**

		Smog			Flood				Snow		
	Original	GAN	3D Style	Ours	GAN	GAN++	Diffusion	Ours	GAN	3D Style	Ours
Family	3.434	3.426	3.425	**3.437**	3.408	3.413	3.416	**3.424**	**3.434**	**3.434**	3.429
Horse	3.426	3.422	3.422	**3.432**	3.409	3.412	3.416	**3.422**	**3.435**	3.424	3.420
Playground	3.409	3.402	3.403	**3.412**	3.400	3.402	**3.409**	3.404	**3.427**	3.412	3.416
Train	3.406	3.396	**3.407**	**3.407**	3.401	3.402	**3.410**	3.407	3.417	3.411	**3.418**
Truck	3.425	3.424	3.424	**3.431**	3.404	3.403	**3.424**	3.413	**3.431**	3.424	3.417

Table 1. **Video Quality Assessment.** We evaluate the video quality with Google's Universal Video Quality (UVQ) model [80].