THE UNIVERSITY OF CHICAGO

ON ALPHA INVARIANCE AND THE INVERSE SCALING BETWEEN DISTANCE AND VOLUME DENSITY IN NEURAL RADIANCE FIELDS

A THESIS SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY
JOSHUA AHN

CHICAGO, ILLINOIS
APRIL 2024

Dedication Text

Epigraph Text

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Acknowledgements text

ABSTRACT

Scale-ambiguity in 3D scene dimensions leads to magnitude-ambiguity of volumetric densities in neural radiance fields, i.e., the densities halve when the scene size doubles, and vice versa. We call this property alpha invariance. For NeRFs to better maintain alpha invariance, we recommend 1) parameterizing both distance and volumetric density in log space, and 2) using a discretization-agnostic initialization strategy to guarantee high transmittance and consistently high rendering quality. We revisit several popular radiance field models from the literature and find that these architectures use various heuristics to address issues arising from scene scaling. We test their behaviors and demonstrate that our recipe is more robust to changes in scene size.

CHAPTER 1

INTRODUCTION

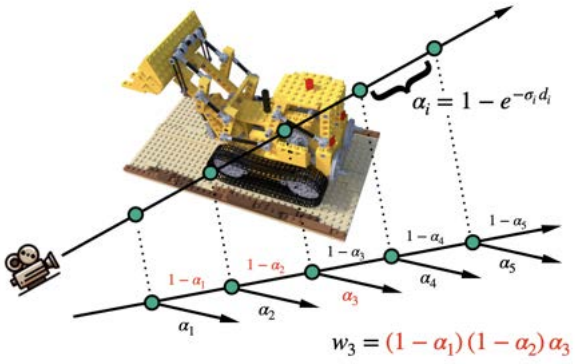

Figure 1.1: A discretized view of volume rendering. **Top**: a ray is cut into intervals, each with a density $\sigma_i \geq 0$ and interval length d_i. **Bottom:** illustration of the weight given to the 3rd interval, computed through alpha compositing. The rendered color is obtained by weighting all the interval colors with their w_is. If we scale each d_i by a constant k, scaling σ_i by $\frac{1}{k}$ renders the identical color.

3D computer graphics and vision are fundamentally scale-ambiguous. Lengths of objects and scenes are often unitless and measure up to a constant ratio. This is fine for routines such as projection, triangulation, and camera motion estimation. A common practice is to normalize the scene dimension or the size of a reference object to an arbitrary number.

Volume rendering [17] used by Neural Radiance Fields (NeRFs) [20] presents a complication due to explicit integration over space. Since the distance scaling is arbitrary, the volumetric density function $\sigma(x)$ must compensate for the multiplicative factor to render the same final RGB color. In other words, if the scene size expands by a factor k, it is *sufficient* for the learned σ to shrink by $1/k$ to be invariant to the change. The same applies to integration over cones or more general spatial data structures [1, 3]. Note that in addition to scene dimensions, the magnitude of σ is also influenced by the number of samples per ray and ultimately the sharpness of change in local opacity.

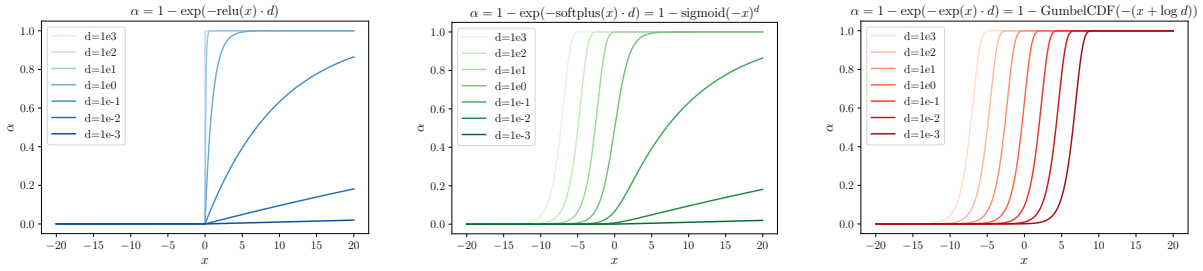

Figure 1.2: α as a function of the raw input x and d. We focus on α as a function of x, with different activation functions $\sigma(x)$, for a set of fixed values of interval lengths d. The `exp` activation (right plot) leads to a smooth, sigmoid-like transition from low to high α values regardless of the interval length d. The function $\exp(-\exp(-x))$ is the CDF of Gumbel distribution, and it is a numerically stable recipe that we recommend over `trunc_exp` because log density and log distance naturally cancel out each other before exponentiation.

There is no single "correct" size for a scene setup. A robust algorithm should be able to perform consistently across different scalings. We investigate how this notion of invariance manifests itself in practice, discuss how the hyperparameter decisions affect it, and propose a solution that ensures robustness to distance scaling across the NeRF methods. We revisit and experiment with a few popular NeRF architectures: Vanilla NeRF [20], TensoRF [6], DVGO [30], Plenoxels [8], and Nerfacto [31], and find that many systems use tailored heuristics that work well at a particular scene size. We analyze the impact of some critical hyperparameters which are often overlooked or not highlighted in the original papers.

In our testing of these models we identify two main failure modes. When the scene is scaled down (short ray interval d), some models struggle to produce large enough σ values for solid geometry. When the scene is scaled up (long interval length), the σ at initialization is often too large, resulting in cloudiness that traps the optimization at bad local optima. We therefore propose 1) parameterizing both distance and volume densities in log space for easier multiplicative scaling; 2) a closed-form formula for σ value initialization that guarantees high ray transmittance and scene transparency. We show that these two ingredients can robustly handle various scene sizes. We are not the first to use `exp` activation for volumetric densities. It was described in Instant-NGP [21] and adopted by many subsequent works. However,

existing informal discussions for why `exp` should be used are somewhat unsatisfactory. We believe that a clearer understanding of its needs would be of benefit. In addition, since distance and densities compensate for each other, moving both to log space i.e. the GumbelCDF form naturally provides numerical stability without the need for truncated `exp` [32].

Our contributions:

- We discuss the notion of alpha invariance, and clarify the relationship of inverse multiplicative scaling between volumetric densities σ and scene size. It is a basic property that applies to volume rendering and radiance fields.
- We survey and ablate popular NeRF architectures on their modeling decisions related to alpha invariance.
- We provide a robust, general recipe that enables NeRFs to achieve consistent view synthesis quality across different scene scalings.

CHAPTER 2

BACKGROUND

Volume rendering. In a space permeated by fog, it is unlikely for a faraway photon to arrive at the sensor. On a ray $\mathbf{z}(t) = \mathbf{o} + t\mathbf{d}$, the probability a photon comes from beyond t i.e. (t, ∞) is modelled by a decaying transmittance function $T(t) = \exp(-\int_0^t \sigma(s)\, ds)$, which vanishes with larger t. The volume density σ is some local light blocking factor, and the cumulative distribution function (CDF) is $1 - T(t)$. The prob. density $w(t)$ of a photon starting at *exactly* time t is thus $\partial_t(1 - T(t))$. With an infinite number of photons, by law of large numbers, the deposited color at the pixel is the sample mean which converges to the expectation

$$\mathbb{E}[\mathbf{c}(t)] = \int_0^\infty w(t)\, \mathbf{c}(t)\, dt = \int_0^\infty \underbrace{\sigma(t)\, T(t)}_{=\partial_t(1-T(t))}\, \mathbf{c}(t)\, dt. \tag{2.1}$$

A NeRF optimizes a mapping from spatial location $\mathbf{z}$ to color and volume density $f_\theta : (\mathbf{z}, *) \to (\mathbf{c}, \sigma)$, where $*$ denotes auxiliary inputs such as viewing direction [20], time [27], scene specific embeddings [16], etc. To render an image, we compute the expectation by integrating $(\mathbf{c}, \sigma)$ over points on each ray.

For a ray discretized into segments each with length d_i, assuming constant volume density and color within the segment, volume rendering takes on the form of alpha compositing. where the "over" operation [26] is applied in a back-to-front order. See 1.1 for a tree-branching analogy.

$$\alpha_i = 1 - e^{-\sigma_i d_i} \quad \text{with} \quad w_i = \prod_{j<i}(1 - \alpha_j) \cdot \alpha_i. \tag{2.2}$$

The final color is produced by the expectation w.r.t. the probability mass function $\sum_i w_i\, \mathbf{c}_i$.

Current practices on σ parameterization. A scalar x predicted by the underlying neural field is converted to volume density $\sigma(x)$ by a non-linear activation function. The choice of

Algorithm 1 Python pseudocode for our 1) GumbelCDF density activation and 2) high transmittance initialization. It is *numerically stable* since log densities, log distances, and high transmittance offset naturally cancel out one another when scene size L or the ray sampling strategy changes.

```
class DensityField():
    def __init__(self, L, tau=1.0, T=0.99):
        # L could be far - near of a typical light ray
        # tau: expected std of the neural field output
        self.offset = \
            log(log(1/T)) - log(L) - (tau**2)/2
        self.encoder = YourDensityEncoder()

    def compute_volrend_ws(self, xyz_locs, deltas):
        # xyz_locs, deltas: shape [N_rays, N_samp]
        log_densities = self.encoder(xyz_locs)
        density_delta = exp(
            log_densities
            + log(deltas)   # GumbelCDF
            + self.offset   # high transmittance
        )
        # log(deltas) + offset invariant w.r.t. L
        trans = append_zeros_in_front(
            cumsum(density_delta[..., :-1], dim=-1),
            dim=-1
        )
        trans = exp(-trans)
        alphas = 1. - exp(-density_delta)
        weights = alphas * trans
        return weights
```

this activation is one of the differences between NeRF models, and the focus of our analysis. It determines how α in Eq. (2.2), which is a function of both d and $\sigma(x)$, depends on x.

MLP-NeRF [20] parameterizes the volume density $\sigma(x)$ with a `ReLU` activation. Mip-NeRF [1] advocates for the use of `softplus` activation, out of concern that `ReLU` might get stuck since there is no gradient if inputs are negative. DVGO [30] fixes the local *interval* size. TensoRF [6] scales the local interval size by a constant. We refer the readers to GitHub issues where these are discussed[1][2]. Plenoxels [8] uses `ReLU` with a very large learning rate on $\sigma(x)$

1. https://github.com/sunset1995/DirectVoxGO/issues/7
2. https://github.com/apchenstu/TensoRF/issues/14

at the beginning of optimization before decaying it as convergence improves[3]. Instant-NGP uses `exp` activation. The motivation is not explained, besides a brief comment by the author on GitHub[4]. Some of these decisions have been adopted by more recent works. For example, HexPlane [5] and LocalRF [18] follow TensoRF's interval scaling strategy, while works building on top of Instant-NGP tend to use the truncated `exp` activation for numerical stability.

3. https://github.com/sxyu/svox2/issues/111

4. https://github.com/NVlabs/instant-ngp/discussions/577

CHAPTER 3

RELATED WORK

Alpha and Bad Weather. Early works on weather modeling [22, 23] use volume rendering to characterize the effect of rain and fog. Dehazing methods [7, 9], apart from separating the base scene albedo from the foggy airlight, use the estimated alpha mask to compute an up-to-scale depth map proportional to $-\log(1-\alpha)$. The notion of alpha invariance is implied in this step.

NeRF vs MPI. NeRF uses volume rendering [17] for view synthesis [20] and other inverse rendering tasks . Unlike prior works [25, 29, 39, 40] that directly model the alphas of a fixed set of multi-plane images (MPI), NeRF optimizes the continuous volumetric densities $\sigma(x)$. MPI fixes the discretization in advance, whereas the density parameterization in NeRF makes it easy to adjust discretization and resample along a ray. The flipside of this flexibility, however, is that the magnitude of $\sigma(x)$ is tied to the domain of integration. Our work emphasizes that even though $\sigma(x)$ and distance change to compensate for each other, the alpha value of a particular discretized segment should remain constant. The opacity is an invariant property of local geometry.

Volume-rendered SDF. Signed distance function (SDF) is suitable for procedural content authoring and surface extraction. SDF rendering used to rely on finding surface intersections by sphere tracing [24, 36], but recent methods such as NeuS [33] and VolSDF [37] move to the "fuzzier" volume rendering for easier optimization. To perform volume rendering, the distance function is first transformed into volumetric densities before alpha compositing. VolSDF learns scaling and shrinking coefficients on the distance-transformed volume densities. NeuS instead demands the CDF of volume rendering to match the shape of a scaled, horizontally flipped `sigmoid` i.e. $1 - T(t) = \texttt{sigmoid}(s \cdot -\texttt{SDF}(t))$, so that the CDF's derivative, in other words the weighting function $w(t)$, has its local maxima located at SDF value 0 for precise surface level-set extraction. The learned parameter s in the `sigmoid` acts as a global scaling

coefficient on the implied volume densities. In this sense both formulations are alpha invariant by default, with the extra constraint that the magnitude of volume density function is globally tied to its sharpness. The assumption is restrictive but fine for most use cases.

Gaussian Splatting. Gaussian splatting [12, 15, 35] renders an image by alpha compositing point primitives. Since the scene representation is by nature discrete, the concept of volume density does not apply, as is the case with MPIs. These explicit primitives can be efficiently rasterized using GPU pipelines. Whereas NeRF could use ray sampling strategies on the continuous density field to refine local details and discover missing structures, 3D gaussians / metaballs [12–14] are less flexible; it relies on SfM initialization in some cases, and needs to explicitly optimize point shape, location, and periodically remove, repopulate and merge-split the primitives. 3DGS [12] uses an exponent activation to scale the shape of each primitive, and it shares a similar motivation in terms of handling arbitrary scene scaling.

CHAPTER 4

ALPHA INVARIANCE

As reviewed in Sec. 2, $w(t) = \sigma(t)T(t)$ forms a probability density function. However, due to the scale-ambiguity of a 3D scene, the size of the support for $w(t)$ is arbitrary. For example in the original NeRF blender synthetic dataset, a light ray travels over $[2.0, 6.0]$, corresponding to ray length (scene scale) $L = 4.0$. If we were to expand the support by $2\times$ to $L = 8.0$, then by change of variables, it is *sufficient* for the probability density $w(t)$ and thus the volume density $\sigma(t)$ to scale down by $\frac{1}{2}$ to integrate to 1 and render the same color. Of course for a given 3D scene, no matter how the distance unit changes, the cumulative opacity of a fixed piece of ray segment is constant. In other words, a desired property of a model is that values of α in Eq. (2.2) should be invariant with respect to (arbitrary) scene scaling. We refer to this desired property as **alpha invariance**.

The magnitude of volume density is also affected by the sampling resolution on each ray. Most NeRF variants use some form of importance sampling to iteratively focus the computation on the fine, detailed structures that would likely be missed by the initial uniform samples. Consider an extremely coarse sampling with only 2 sampled points: each interval is unusually large, and the density needed to achieve a high alpha would be small. On the other hand, with fine-grained importance sampling, large densities are needed to create a sharp, solid geometry within the small interval. We can calculate the volume density required to achieve a certain alpha value by $-\frac{1}{d}\log(1-\alpha)$. Some example values are shown in Tab. 4.1, where we set the overall ray length again to $L = 4.0$ and vary L, the sampling resolution, and the alpha threshold.

The three activation functions mentioned in Sec. 2 as used in practice to compute $\sigma(x)$

	$d = \frac{L}{64}$	$d = \frac{L \times 2}{64}$	$d = \frac{L}{128}$	$d = \frac{L}{64 \times 128}$
$\alpha = 0.5$	11.1	5.5	22.2	1419.6
$\alpha = 0.99$	73.7	36.8	147.4	9431.4
$\alpha = 0.999$	110.5	55.3	221.0	14147.1

Table 4.1: Some example σ values given desired alpha level and interval size, using $\sigma = -\frac{1}{d}\log(1-\alpha)$. Length of ray L is set to 4.0, which is blender synthetic dataset default, and we vary the number of samples per ray. Doubling L halves the σ, while higher sampling resolution increases σ. In the extreme case where all the 128 importance samples fall within one of the initial 64 uniformly sampled bins, the magnitude of σ can get very large.

lead to the following form for α as a function of x and d:

$$\texttt{ReLU} : \alpha = 1 - \exp(-\texttt{ReLU}(x) \cdot d) \tag{4.1a}$$

$$\texttt{softplus} : \alpha = 1 - \texttt{sigmoid}(-x)^d \tag{4.1b}$$

$$\texttt{exp} : \alpha = 1 - \exp(-\exp(x) \cdot d) \tag{4.1c}$$

where we use the identity $\exp(-\texttt{softplus}(x)) = \texttt{sigmoid}(-x)$. To more clearly compare the effect of `exp` against `ReLU` and `softplus`, in Fig. 1.2 we visualize the behavior of $\alpha(x, d)$ with these activations, for different segment length d from 10^{-3} to 10^3. When d is small, both `ReLU` and `softplus` struggle to produce large α values; the values of x that the networks must predict become extreme. In contrast, with the `exp` activation $\sigma = \exp(x)$ a constant offset on x results in a direct scaling on σ.

We also note that the function $e^{-e^{-x}}$ is the CDF of Gumbel distribution,

$$\begin{aligned}\alpha &= 1 - \exp(-\exp(x) \cdot d) \\ &= 1 - \exp(-\exp(x + \log d)) \\ &= 1 - \texttt{GumbelCDF}(-(x + \log d)).\end{aligned} \tag{4.2}$$

This connection is related to the fact that the volume rendering equation is a restatement of

the Exponential distribution. The PDF and CDF functions in Eq. (2.1) are the PDF and CDF of the Exponential distribution assuming a constant "rate". This can aid in numerical stability, a commonly cited issue with using exp to parametrize σ. Large density σ is needed primarily to create sharp opacity change in a small distance interval. If we move the distance multiplication into the `exp` as a $\log d$ addition, then there should not be an overflow issue. Note that the derivative of $\exp(-\exp(x))$ is not symmetrical about y-axis even though its shape resembles `sigmoid`.

While `exp` activation has no problem producing large densities when scene scale L is small, a different challenge occurs when L is large. Given the same sampling strategy, larger L and therefore larger interval d, with the same initial volume density will produce larger alpha values. This manifests as an opaque initial scene. Optimization frequently fails since the images can be explained away by dense, cloudy floaters in front of each camera. It is important to guarantee that upon initialization the scene is transparent, i.e. each ray has high transmittance.

Since ray transmittance is related to volume density by $T(L) = \exp(-\int_0^L \sigma(s)\,ds)$, we may want the network predicting the density to be initialized so that the "average volume transmittance" in the scene is T', a value like 0.99. In practice the initial volume density value along a ray is not a constant, but a random variable produced by the underlying field function such as voxels or an MLP. We make the simplifying assumption that the sampled values are i.i.d. (admittedly imperfect, since neighboring values tend to be correlated by feature or voxel sharing). The integral $-\int_0^L \sigma(s)\,ds$ gives the Monte Carlo estimate that converges to $-L \cdot \mathbb{E}[\sigma]$. Our recommendation can then be written as

$$\exp(-L \cdot \mathbb{E}[\sigma]) = T'. \tag{4.3}$$

In case of $\sigma(x) = \texttt{exp}(x)$, assuming the pre-activation field output x is drawn from $\mathcal{N}(\mu, \tau^2)$, σ follows a log-normal distribution with mean $\mathbb{E}[\sigma] = \exp(\mu + \frac{\tau^2}{2})$. Plugging this into Eq. (4.3),

taking log, and rearranging, we get

$$\exp(\mu + \frac{\tau^2}{2}) = \frac{1}{L} \log \frac{1}{T'}, \tag{4.4}$$

and solving for μ we obtain

$$\mu = \log \log \frac{1}{T'} - \log L - \frac{\tau^2}{2}. \tag{4.5}$$

This is the desired target for initializing the density-predicting network. Setting the mean μ of the pre-activation field output to this value can be done with an additive offset. This strategy is discretization agnostic, and a factor multiplied onto L can be undone by a logarithmic shift. The merged expression for local alpha as a function of field output x and interval length d is

$$\begin{aligned} \alpha = &\quad 1 - \exp(-\exp(x + \log d + \mu)) \\ = &\ 1 - \exp\left(-\exp(x + \log \tfrac{d}{L} + \log \log \tfrac{1}{T'} - \tfrac{\tau^2}{2})\right), \end{aligned} \tag{4.6}$$

where $\log \frac{d}{L}$ is invariant w.r.t. scene scaling. See Alg. 1 for Python pseudocode.

For a real-life scene where ray length varies, a possible choice is to set L to the longest ray distance. One interpretation of $\log \frac{d}{L}$ is that we are working with distance ratios and are thus effectively hardcoding the overall scene size to 1.0. But it's only possible with the extra offset terms, without which, assuming $\tau = 1.0$, $T' = 0.99$, the scene size L needs to be set to 0.006.

Analogous recipes can be made for `ReLU` and `softplus`, although it's harder to write down closed-form expressions [34]. We could attempt it using Monte Carlo estimates of their mean. Recall our goal is to achieve $\mathbb{E}[\sigma] \cdot L = \log(\frac{1}{T'})$. Assuming the pre-activation neural field output is drawn from $\mathcal{N}(0, 1)$, the expectations of the post-activation random variables can be approximated by sample mean. When $T' = 0.99$, the RHS is 0.001. We want to *avoid* directly scaling down density or L to meet this target, since that would make it even harder

for `relu` and `softplus` to achieve large density needed at solid regions. Instead the high transmittance should be achieved by shifting the activation function profile to the right, i.e. adding negative offset to the input. The correct offset can be tuned numerically, and only needs to be done once.

CHAPTER 5

EXPERIMENTS

Scene size L is fundamentally an arbitrary decision for NeRF optimization. A robust algorithm should be able to achieve consistent view synthesis quality regardless of the value of L. We analyze different types of NeRF systems including pure MLPs to Voxels to Hashgrid-MLP hybrids by training them with a scaling factor k applied on the default scene size L, and leave the rest of the optimization hyperparameters such as sampling strategies unchanged. In our experiments, unless otherwise stated, we set the desired transmittance $T' = 99\%$ for the longest ray in each scene, and set k to range from 0.1 to up to 25, report the PSNR metrics, and provide qualitative visualizations. We find that these methods are unable to maintain high rendering quality especially when k is more extreme, whereas our recipe achieves a consistently high quality across all k.

5.1 How volume density changes in practice

Findings 1. Empirically σ changes by a factor close to $\frac{1}{k}$ when scene size is changed by k. It is most noticeable on object surfaces.

Inverse scaling between volume density and distance is sufficient to guarantee identical renderings for a radiance field. However, this might not be necessary in practice if the end goal is to achieve high PSNR values. A scene is typically dominated by either empty space or solid regions; semi-transparent structures occur much less frequently, and volumetric densities tend to take on extreme values. As such, a change in the scale of the interval length might not substantially alter the local alpha values. To get a better understanding of their exact empirical behaviors, we train vanilla-NeRF on the lego and drums scenes from the blender dataset, and Nerfacto [31] on the garden and bicycle scenes from the Mip-NeRF 360 dataset, with $k = [0.1, 1.0, 10.0]$. Here we use our proposed Alg. 1.

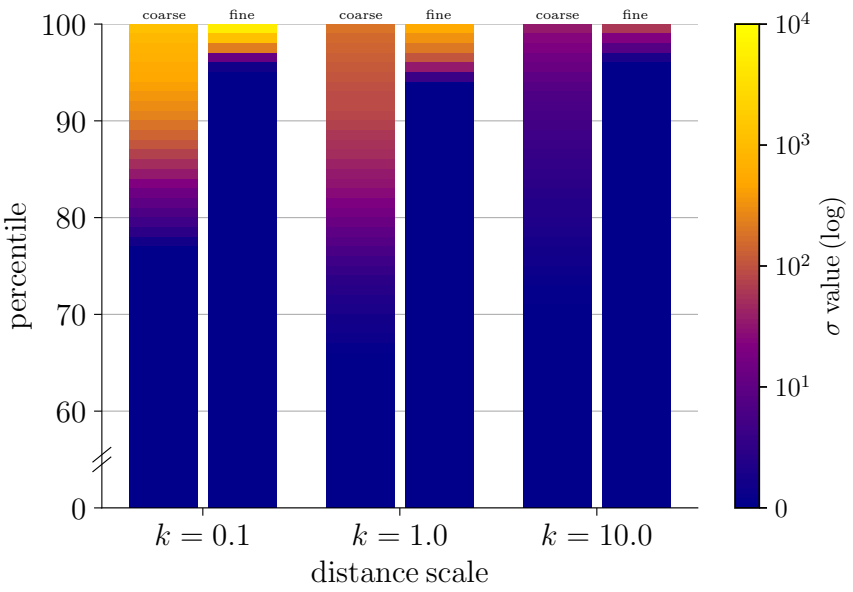

Figure 5.1: Distribution of volume density σ in the lego bulldozer scene, queried via a uniformly sampled dense grid of points from both the coarse (left columns) and fine (right columns) MLP networks of vanilla-NeRF for different k. Color represents the average σ value in each percentile of the sorted σ distribution. The fine MLPs produce larger σ than the coarse MLPs, and as k increases, the magnitude of σ decreases for both networks.

Volume statistics. For the coarse and fine MLP networks in vanilla-NeRF, we query a dense grid of 200^3 uniformly sampled points in the scene box, and summarize the sorted σ distribution with bar plots depicting the average σ value within each percentile in Fig. 5.1. As most of the scene is empty ($> 60\%$), we focus on the distribution of values in the upper percentiles and observe that with increasing k values, the numerical range of σ does decrease as we hypothesized in both the coarse and fine MLP networks. In addition, we observe that the fine MLPs produce a very small amount of significantly larger σ values than the coarse MLPs due to sharper geometry being captured by the importance sampler over the course of optimization.

Surface statistics. Since large volume density occurs near the object surface, we visualize the changing surface densities in Fig. 5.2. At a given camera pose, we shoot rays through each pixel to obtain the density histograms $\{w_i\}$, and query the spatial location at the 50-th

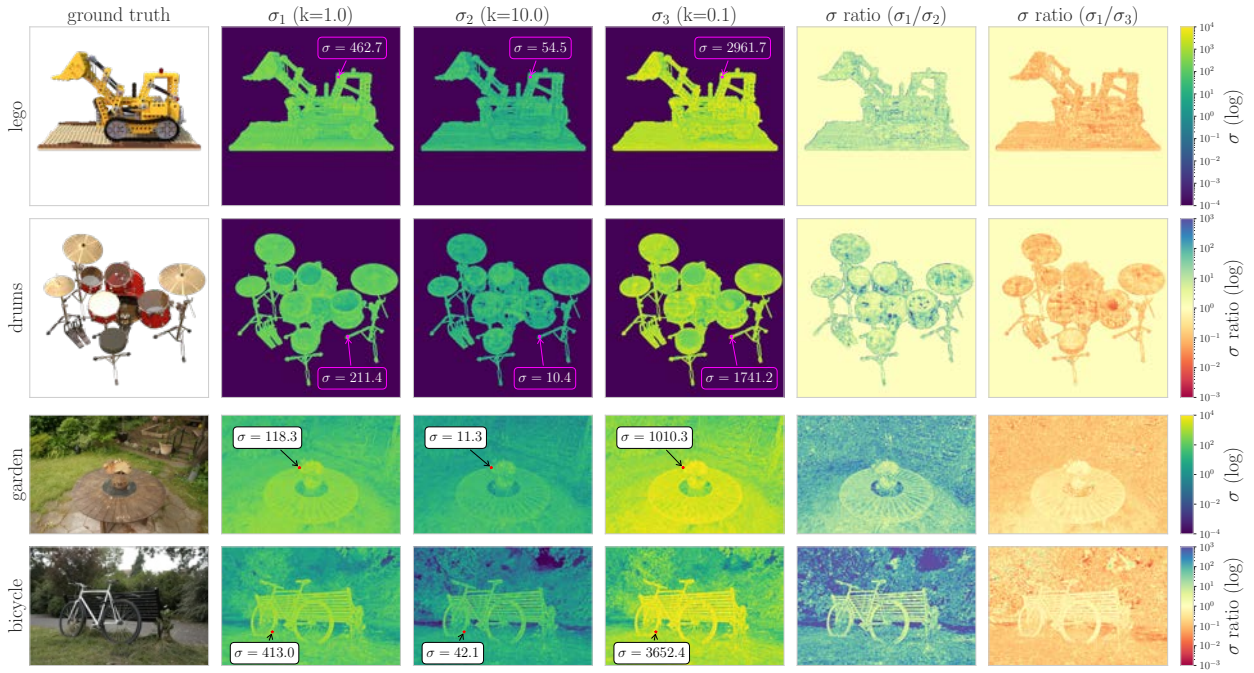

Figure 5.2: Values of volume density σ on the object surface, with models trained under different scene scalings k. On each ray, surface point is defined as the 50th percentile location of volume rendering CDF. We annotate a few prominent points, and also produce two σ division images that show the overall ratio of the numerical range of σ at different scaling factors. The blender scenes are trained with vanilla NeRF, and the Mip-NeRF 360 scenes are trained with Nerfacto. The ratio of σ is empirically close to $1/k$.

percentile of the probability CDF of each histogram for its σ value. We also display heatmaps of density ratio by dividing the σ images coordinate-wise. Values below $\epsilon = 10^{-4}$ are rounded up to ϵ for numerical stability. Based on these visualizations, we observe that inverse-scaling between density and distance does hold to a significant degree, although the factor in practice deviates from $1/k$.

5.2 Vanilla NeRF with MLPs

Findings 2. MLPs are surprisingly robust with just `ReLU` activation but they can be improved.

The canonical NeRF architecture uses an 8-layer MLP with a `ReLU` activation to produce

	chair	drums	ficus	hotdog
$k = 0.1$	30.98 / 31.19	24.25 / 24.37	28.72 / 29.26	32.22 / 34.84
$k = 0.4$	31.21 / 31.29	23.85 / 24.54	28.74 / 29.01	32.95 / 33.51
$k = 1.0$	31.26 / 31.26	12.04 / 24.58	28.76 / 28.85	33.59 / 33.71
$k = 2.5$	31.22 / 31.32	24.04 / 23.99	26.86 / 28.60	10.94 / 33.91
$k = 10.0$	14.04 / 31.36	23.84 / 24.43	28.97 / 28.92	10.39 / 33.23
	lego	**materials**	**mic**	**ship**
$k = 0.1$	30.90 / 30.64	28.27 / 28.45	28.31 / 30.96	26.70 / 27.56
$k = 0.4$	9.45 / 30.71	28.02 / 28.59	31.41 / 30.94	10.45 / 27.35
$k = 1.0$	30.71 / 31.36	13.89 / 28.57	31.35 / 31.05	25.65 / 27.21
$k = 2.5$	30.74 / 31.11	27.76 / 28.85	30.97 / 31.35	5.88 / 27.15
$k = 10.0$	30.55 / 30.84	27.97 / 28.25	31.04 / 31.44	5.88 / 26.99

Table 5.1: PSNR ↑ for NeRF-Pytorch [baseline / ours] at different scene scaling k on Blender synthetic dataset. Vanilla NeRF with `ReLU` activation is surprisingly capable of producing large σ values but the optimization does converge to poor local minima randomly. Using our recommended recipe ensures consistent convergence across all k. Note that the NeRF-Pytorch codebase is unable to exactly match the original NeRF [20] performance at 200k iterations. See appendix for details on reproducing the random failures present here.

σ, and hence must output values in the range of hundreds near object surfaces as shown in Fig. 5.2. This setup is somewhat against the conventional wisdom [4, 11] of fixing the neural network output magnitude to unit variance for stable training. We test our recipe on NeRF-PyTorch [38], train for 200k steps and present the results in Tab. 5.1. We make three observations. First, NeRF-PyTorch uses PyTorch's default linear layer initialization which does not correct for the variance gain of `ReLU`, and produces raw outputs with mean 0 and variance 0. There are random failure modes across all k. To address this, we initialize the layers with Kaiming uniform initialization [10] with $\sqrt{2}$ gain. Interestingly, we still observe random failure modes, even at $k = 1$. Finally, we tried parameterizing density with our recipe (without applying the aforementioned Kaiming uniform initialization such that $\tau = 0.0$ initially, making the task more difficult), and observed consistent performance across all k without any random failures.

	chair	**drums**	**ficus**	**hotdog**
$k = 0.1$	fail / 35.41	fail / 26.01	fail / 33.81	fail / 37.03
$k = 0.4$	35.74 / 35.96	26.04 / 25.91	fail / 34.00	37.31 / 36.93
$k = 1.0$	35.55 / 35.57	26.24 / 26.24	33.99 / 34.31	37.31 / 37.20
$k = 2.5$	35.71 / 35.88	26.26 / 26.14	34.05 / 34.15	37.09 / 37.14
$k = 10.0$	35.67 / 35.92	26.14 / 26.41	34.01 / 34.81	37.10 / 37.51
	lego	**materials**	**mic**	**ship**
$k = 0.1$	fail / 36.47	fail / 29.99	fail / 34.52	fail / 30.51
$k = 0.4$	36.20 / 35.93	30.14 / 30.10	fail / 34.31	30.81 / 30.09
$k = 1.0$	36.60 / 36.55	30.01 / 30.11	34.59 / 34.71	30.65 / 30.59
$k = 2.5$	36.66 / 36.51	30.11 / 30.21	34.48 / 34.61	30.71 / 30.80
$k = 10.0$	36.18 / 37.04	30.36 / 30.40	34.53 / 34.62	30.74 / 30.80

Table 5.2: PSNR values of TensoRF [baseline / ours] at different scene scaling k on Blender synthetic dataset. TensoRF applies an input offset -10 before `softplus`, followed by an interval multiplier of 25. The hardcoded decisions are less effective at $k = 0.1$. TensoRF is quite robust at $k = 10$ since the -10 is a rather aggressive offset. See Fig 1.2 for the effect of shifting the `softplus` activation.

	lr_init = 30.0	lr_init = 0.3	lr_init = 0.003
blender	31.66	29.51	23.51
LLFF [19]	24.51	23.22	21.09

Table 5.3: Measuring Plenoxels' performance with PSNR ↑ on the Blender and LLFF datasets with different learning rate schedules.

5.3 Voxel Variants: DVGO, Plenoxels and TensoRF

Findings 3. Voxel variants cannot converge with `ReLU` or `softplus` activations alone. Hardcoded heuristics have been used for stable training.

Unlike MLPs, direct optimization of voxel NeRFs with `ReLU` or `softplus` activations does not converge. We observe the same pattern across all three voxel model variants, and it shows the benefit of having an MLP as a global inductive bias. DVGO uses `softplus` activation, and to address the issue of divergence, it fixes a canonical scene size with the voxel size ratio such that each *interval* has length 0.5 or 1.0 depending on whether it is in the coarse or fine reconstruction stage. It then calculates a density offset such that every *local* cell's alpha value is small at initialization. We instead recommend initializing the density network based

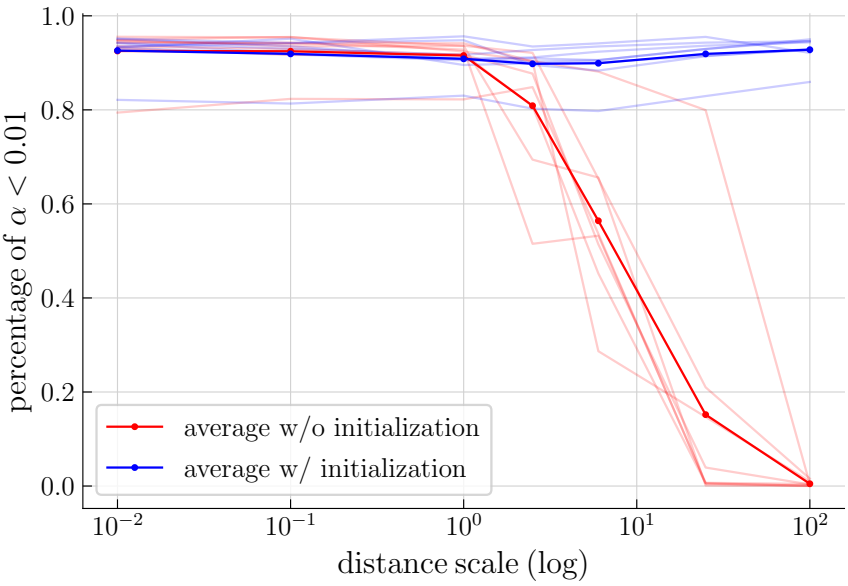

Figure 5.3: Querying a dense uniform grid of samples from TensoRF trained with exp activation. We consider a location to be empty if the local alpha is below 0.01, and apply distance scaling k ranging from 10^{-2} to 10^{2}. Each faint line corresponds to a scene from the blender dataset; the solid line is the dataset average. With our high transmittance initialization, we prevent over-densification even when k is large.

on overall ray transmittance. TensoRF applies a constant interval scaling of 25 with -10 offset on `softplus` input. We verify the robustness of our overall recipe in comparison with TensoRF's and provide numerical metrics in Tab. 5.2. We also ablate our initialization offset on TensoRF. Applying exp alone is not enough to make TensoRF robust across scene scalings; as k increases, we show that the distribution of alpha values everywhere in the scene tends towards 1. Applying our high transmittance offset enables TensoRF to converge smoothly, regardless of k. See Fig. 5.3. Plenoxels uses a total variation (TV) regularizer, and `ReLU` activation with an unusually large initial learning rate of 30.0, in addition to a reverse cosine decay on the σ coefficients. Disabling it, i.e. reducing the learning rate to a lower value such as 0.3 or 0.003, produces smaller σ, and is directly correlated with worse rendering quality. See Fig. 5.4 and Tab. 5.3.

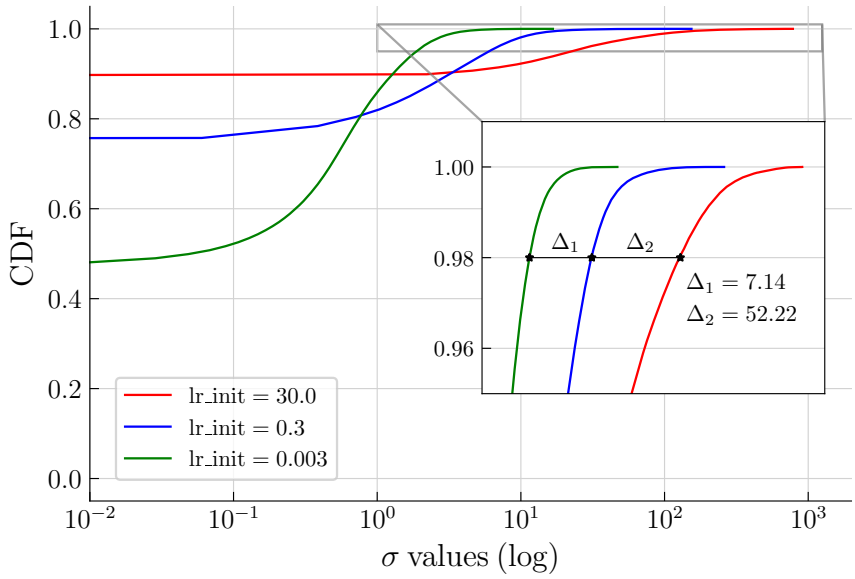

Figure 5.4: CDF of the σ distributions produced by Plenoxels on the T-Rex scene from the LLFF dataset with different learning rate schedules on the σ voxels. The default high learning rate schedule [red] is needed to produce large σ values and high PSNR. Lower learning rates lead to smaller σ and ultimately worse performance.

5.4 MLP and Hashgrid Hybrids

Findings 4. The MLP+Hashgrid hybrid with exp activation makes the model robust at small k. However, high transmittance offset is needed for large k.

Instant-NGP [21] and many follow-ups, including Nerfacto [31], already use exp activation to parameterize σ. Here we study the Nerfacto architecture, since it subsumes many of Instant-NGP's components and supports additional features like Mip-NeRF 360's scene contraction [2]. While Nerfacto is robust at producing large σ when k is small, the optimization gets stuck in the "cloudiness trap" when k is large, since the increased ray distances cause the alpha values to increase, leading to an initially opaque scene. See Tab. 5.4 for experimental results on the Mip-NeRF 360 dataset. See Fig. 5.5 for RGB-image and depth-map comparisons. Our high transmittance initialization strategy enables smooth optimization across k.

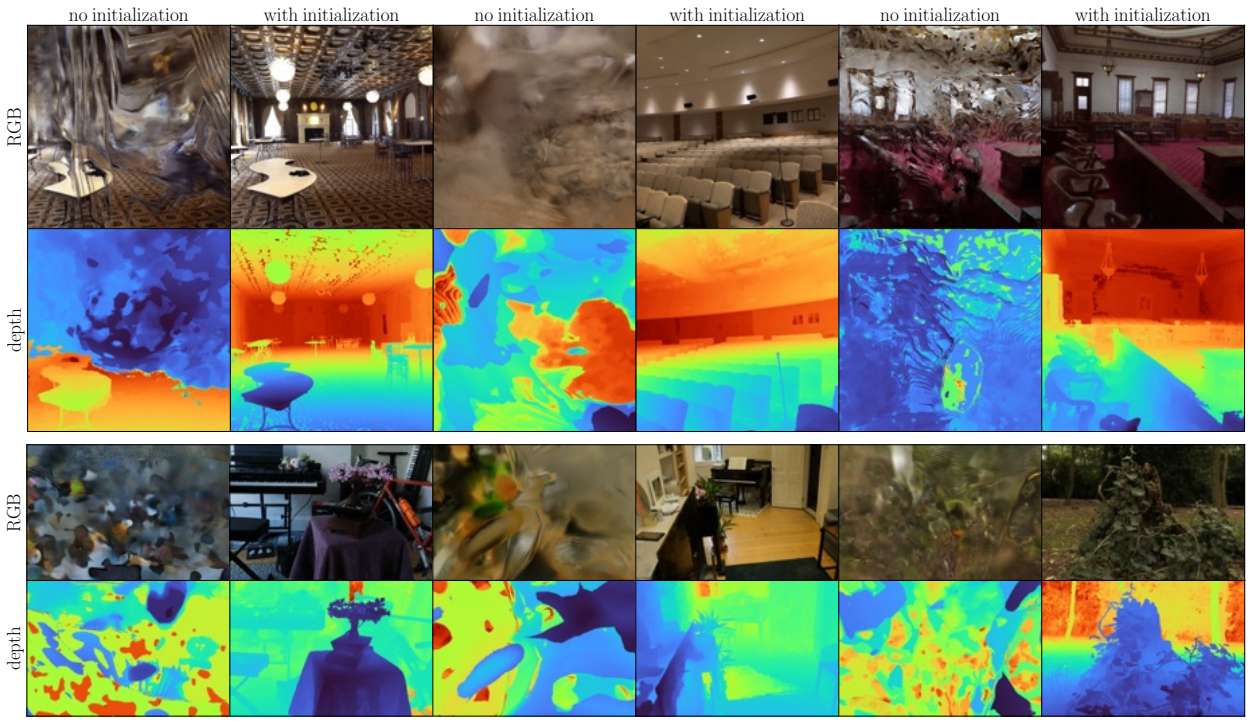

Figure 5.5: Image and depth maps produced by Nerfacto on the ballroom, auditorium, and courtroom scenes from Tanks and Temples at $k = 25$, and the bonsai, room, and stump scenes from the Mip-NeRF 360 dataset at $k = 10$. Not using high transmittance offset causes the optimization to get stuck with cloudy floaters.

5.5 Background Contraction / Disparity Sampling

To represent unbounded scenes, Mip-NeRF 360 [2] applies a contraction of $(2-\frac{1}{\|x\|})\frac{x}{\|x\|}$ if $\|x\|_2 > 1$ so that distant points with insufficient camera coverage use less model capacity. Nerfstudio [31] and MeRF [28] modify it to be more suitable for voxel grids. This scene contraction is part of the underlying NN blackbox that is in principle orthogonal to how we do volume rendering. What matters is the sampling strategy, which in this case is co-designed with samples spaced linearly in disparity. Importantly, metric distances (in t-space) *before* the disparity transform are used as the ray interval lengths d for volume rendering. Taking Nerfacto for example, the metric ray travels from 0.05 to 1000, with the largest intervals d close to 700 for samples near the outer scene boundary. It provides an inductive bias that faraway background regions have alpha values close to 1.0 at initialization. Note that Nerfacto's ray

	bicycle	bonsai	counter	garden
$k = 0.1$	22.51 / 22.38	28.71 / 28.86	25.22 / 25.16	26.26 / 26.63
$k = 0.4$	22.52 / 22.53	28.39 / 28.43	24.76 / 25.24	26.62 / 26.87
$k = 1.0$	22.62 / 22.48	28.11 / 28.71	24.71 / 24.70	26.75 / 26.61
$k = 2.5$	22.61 / 22.52	28.81 / 28.40	24.96 / 24.96	26.76 / 26.67
$k = 10.0$	12.42 / 22.40	13.91 / 28.50	24.55 / 25.26	14.35 / 26.47

	kitchen	room	stump
$k = 0.1$	28.04 / 28.24	28.70 / 28.71	23.42 / 23.48
$k = 0.4$	28.21 / 28.15	28.58 / 28.77	23.50 / 23.77
$k = 1.0$	27.31 / 28.15	28.90 / 28.91	23.45 / 23.62
$k = 2.5$	27.42 / 27.72	28.73 / 28.50	18.59 / 23.53
$k = 10.0$	27.69 / 28.21	11.27 / 28.89	15.91 / 23.81

Table 5.4: PSNR values of [baseline / ours] with different scene scalings k on the Mip-NeRF 360 dataset. The baseline is Nerfacto, which uses the `exp` activation, and thus performs well even when k is small. Using a high transmittance initialization strategy like ours makes it more robust at larger scene scalings.

	bicycle	bonsai	counter	garden	kitchen	room	stump	average
$[L_{\text{in}}, L_{\text{out}}] = [20, 10000]$	19.81	27.34	23.51	26.71	27.46	28.60	19.57	24.71
$[L_{\text{in}}, L_{\text{out}}] = [20, 40]$	22.27	28.41	25.19	26.23	28.23	28.72	23.74	26.11
$[L_{\text{in}}, L_{\text{out}}] = [20, 20]$	22.11	28.94	25.66	26.00	28.14	29.03	24.02	26.27
$[L_{\text{in}}, L_{\text{out}}] = [20, 0]$	22.25	28.46	25.03	26.11	27.77	28.44	18.42	25.21

Table 5.5: PSNR ↑ for Nerfacto on the Mip-NeRF 360 dataset at $k = 10$. $L_{\text{in}} = 2 \cdot k = 20$ (ignoring the small near-plane) and we explore different L_{out} for transmittance offset in the outer dome. The metric ray distance 10000 removes the background inductive bias as described in Sec 5.5 and degrades performance. $L_{\text{out}} = L_{\text{in}}$ is the easiest to implement and has good performance.

sampler applies uniform sampling in uncontracted space and disparity sampling in contracted space. These locations are then further updated by importance sampling. Applying scene scaling with a purely disparity based ray sampler is problematic since sample locations barely move except for the last few bins. When applying the high transmittance offset of Eq. (4.5), our initial attempt is to divide the scene based on the (max) norm of sampled location. If $\|x\|_\infty \leq 1$ we calculate the offset with $L_{\text{in}} = 2.0$ as usual. If $\|x\|_\infty > 1$, we try using $L_{\text{out}} = 0.0$, 2.0, 4.0 or 1000.0. $L_{\text{out}} = 1000.0$ would undo the background inductive bias and in practice results in floaters in near-camera regions. Not applying any transmittance offset

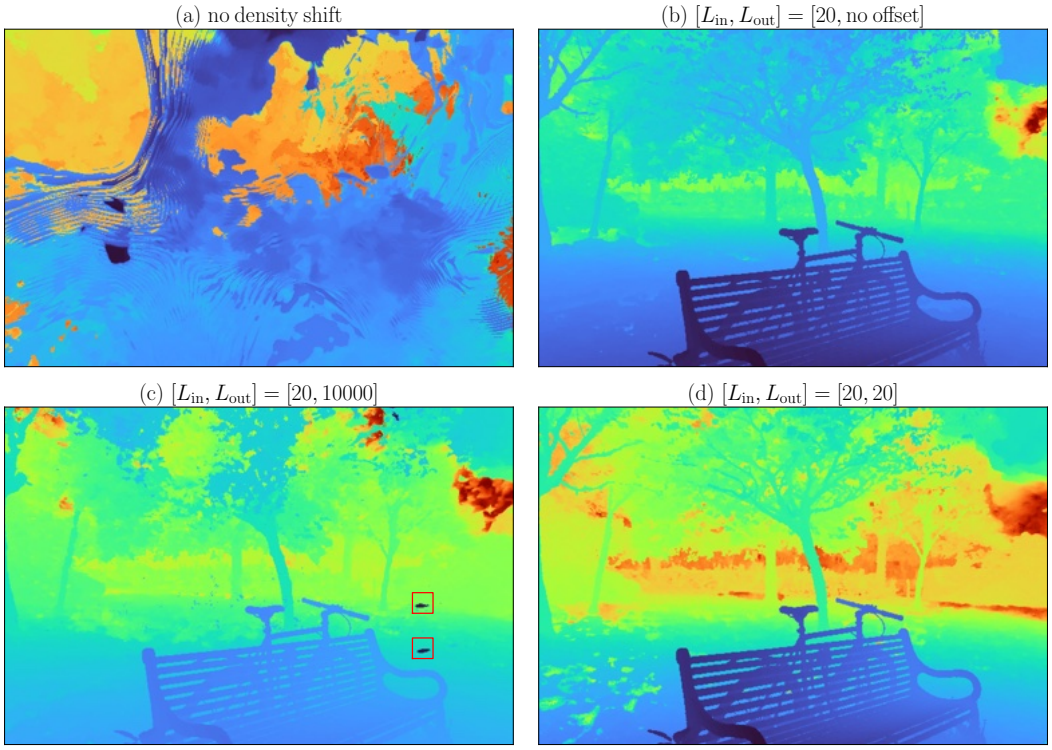

Figure 5.6: Depth maps from Nerfacto with various high transmittance offsets applied on a scene with contracted background. The dataset is "Bicycle" from the Mip-NeRF 360 with scaling $k = 10$. (a) No transmittance offset on foreground or background leads to divergence. (b) Applying the offset only on the inner dome is fine for reconstructing the foreground object, but leaves a "wall" of opacity on the background. (c) Setting L_{out} to be the metric ray distance produces misty floaters in near-camera regions. (d) $L_{\text{out}} = L_{\text{in}}$ performs well and is easy to implement.

at all i.e. $L_{\text{out}} = 0$ is not good either when scene scaling factor k is large. See Tab. 5.5 and Fig. 5.6 for comparisons of various L_{out} at $k = 10$. Our final recommendation is to simply let $L_{\text{out}} = L_{\text{in}}$, so that there is no need to write branching logic in the implementation for $\|x\|_\infty \leq 1$. The same offset is therefore applied to every point in the scene.

CHAPTER 6

CONCLUSION

We present and clarify the concept of alpha invariance. Volume density and scene size change inversely in order to maintain identical local opacities in a radiance field. It is an implication of the scale-ambiguity in a 3D scene. We recommend parameterizing both distance and volume densities in log space, along with a high transmittance offset for robust performance across scene sizes, and verify the approach on a few commonly used architectures in the literature.

One future idea we hope to pursue is applying this formulation to the domain of level-of-details (LoD) in computer graphics. VR and AR simulations operate at varying scales, and we believe that an application of our formulation to here can enable smooth transitions from large regions to zoomed-in objects without loss of reconstruction quality. In addition, we are also interested in developing improved sampling strategies that take into account the current scale of the scene in order to dynamically allocate more memory to under-observed regions so that the overall fidelity can remain high.

REFERENCES

[1] 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 *Int. Conf. Comput. Vis.*, 2021.

[2] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

[3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-NeRF: Anti-aliased grid-based neural radiance fields. In *Int. Conf. Comput. Vis.*, 2023.

[4] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[5] Ang Cao and Justin Johnson. Hexplane: A fast representation for dynamic scenes. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2023.

[6] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *Eur. Conf. Comput. Vis.*, 2022.

[7] Raanan Fattal. Single image dehazing. *ACM Trans. Graph.*, 2008.

[8] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

[9] Kaiming He, Jian Sun, and Xiaoou Tang. Single image haze removal using dark channel prior. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2010.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, 2015.

[11] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2023.

[12] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 2023.

[13] Leonid Keselman and Martial Hebert. Approximate differentiable rendering with algebraic surfaces. In *Eur. Conf. Comput. Vis.*, 2022.

[14] Leonid Keselman and Martial Hebert. Flexible techniques for differentiable rendering with 3d gaussians. *arXiv preprint arXiv:2308.14737*, 2023.

[15] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d Gaussians: Tracking by persistent dynamic view synthesis. *arXiv preprint arXiv:2308.09713*, 2023.

[16] 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 *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021.

[17] Nelson Max. Optical models for direct volume rendering. *IEEE Trans. Vis. Comput. Graph.*, 1995.

[18] Andreas Meuleman, Yu-Lun Liu, Chen Gao, Jia-Bin Huang, Changil Kim, Min H. Kim, and Johannes Kopf. Progressively optimized local radiance fields for robust view synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2023.

[19] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Trans. Graph.*, 2019.

[20] 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 *Eur. Conf. Comput. Vis.*, pages 405–421, 2020.

[21] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 2022.

[22] Srinivasa G Narasimhan and Shree K Nayar. Chromatic framework for vision in bad weather. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2000.

[23] Shree K Nayar and Srinivasa G Narasimhan. Vision in bad weather. In *Int. Conf. Comput. Vis.*, 1999.

[24] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[25] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. *ACM Trans. Graph.*, 2017.

[26] Thomas Porter and Tom Duff. Compositing digital images. In *Proceedings of Computer graphics and interactive techniques*, 1984.

[27] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural radiance fields for dynamic scenes. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021.

[28] Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. MERF: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *ACM Trans. Graph.*, 2023.

[29] 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 *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[30] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

[31] 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 preprint arXiv:2302.04264*, 2023.

[32] Jiaxiang Tang. Torch-ngp: a PyTorch implementation of instant-ngp, 2022. https://github.com/ashawkey/torch-ngp.

[33] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *Adv. Neural Inform. Process. Syst.*, 2021.

[34] Wikipedia. Rectified Gaussian distribution — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Rectified%20Gaussian%20distribution&oldid=1140140064, 2023.

[35] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d Gaussian splatting for real-time dynamic scene rendering. *arXiv preprint arXiv:2310.08528*, 2023.

[36] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. In *Adv. Neural Inform. Process. Syst.*, 2020.

[37] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. In *Adv. Neural Inform. Process. Syst.*, 2021.

[38] Lin Yen-Chen. NeRF-Pytorch. https://github.com/yenchenlin/nerf-pytorch/, 2020.

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

[40] C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 2004.

APPENDIX A

APPENDIX

A.1 Transmittance in Discrete Setting

In Sec. 4, we write down the expression for high transmittance initialization in the continuous setting. The expression is the same when the ray is cut into discrete intervals. The tree-branching analogy in Fig. 1.1 shows that the transmittance / survival probability for each segment is $1 - \alpha_i$, with overall survival probability $\prod(1 - \alpha_i) = \prod e^{-\sigma_i d_i} = e^{-\sum \sigma_i d_i} = \exp\left(-\int_0^L \sigma ds\right)$. The same steps as Eq. (4.3) from Sec. 4 follow.

A.2 Additional Results

Surface Statistics. We provide additional visualizations verifying alpha invariance in Fig. A.2. Similar to Fig. 5.2, we shoot rays through each pixel to obtain the density histograms $\{w_i\}$, and query the spatial location at the 50-th percentile of the probability CDF of each histogram for its σ value. Here, we showcase various scene types from different datasets, and generate these visualizations with different architectures, demonstrating how inverse scaling between distance and volume density is a generalizable phenomenon in radiance fields.

Voxel Variants, continued. DVGO has a strategy of fixing the scene size to some canonical scale where the interval length between each sampled point is dependent on the current voxel grid resolution's ratio to the base resolution, which empirically equates to either 0.5 or 1, depending on the stages of optimization progress. We also note that DVGO's sampling procedure is stochastic, as each ray has a potentially unique number of samples. The implication is that every ray will have a different ray length purely determined by its number of samples as the interval length between any two contiguous samples is hardcoded to either 0.5 or 1. As such, the term "ray length" loses its meaning in DVGO's context as there are no deterministic near and far planes; every ray is simply deconstructed into its constituent

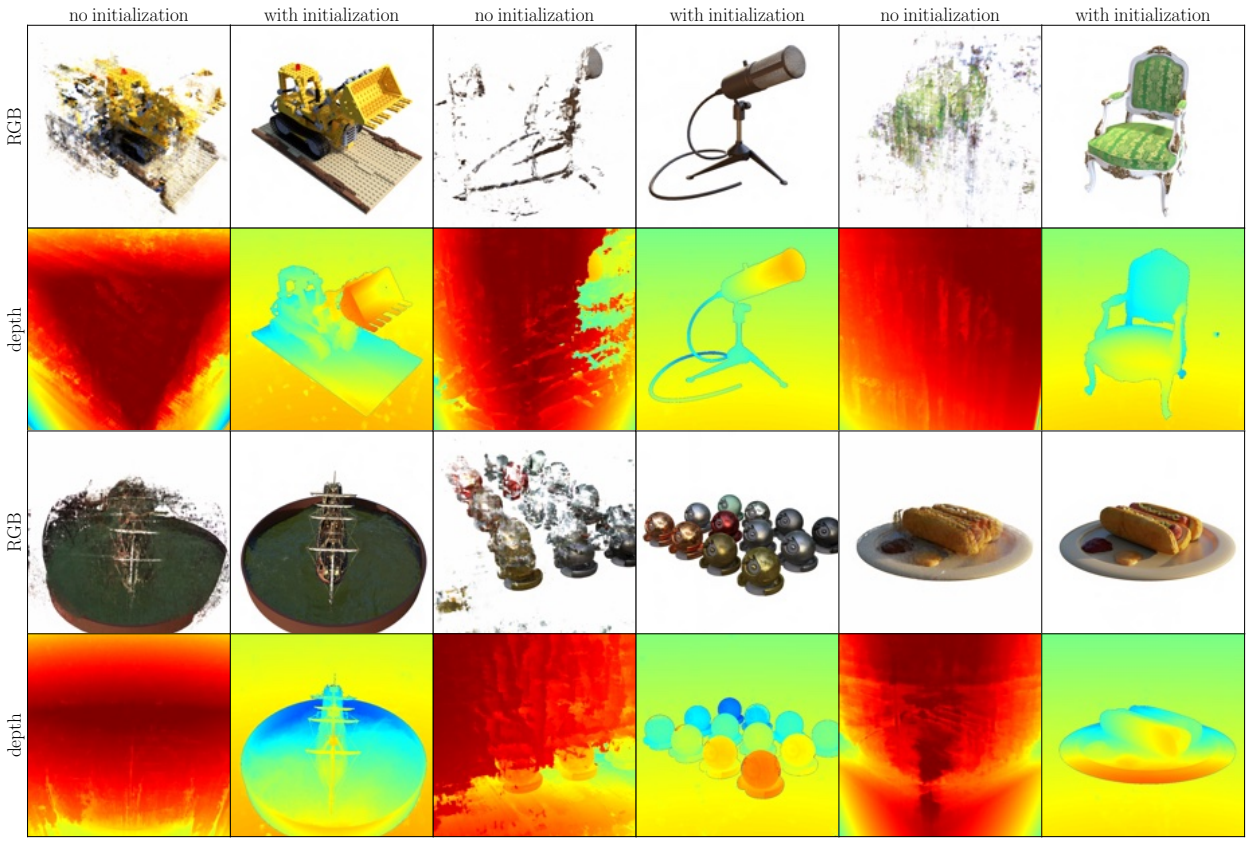

Figure A.1: RGB-image and depth-maps produced from TensoRF on the lego, mic, chair, ship, materials, and hotdog scenes from the Blender dataset at a large scene scaling $k = 25$. When using `exp` activation to parameterize σ, not using our high transmittance initialization strategy causes the optimization to get stuck with cloudy floaters.

samples. We observe that disabling this heuristic (i.e., setting the interval lengths to be the true physical distances and forcing each ray to be bounded within a predetermined global near and far plane) produces significantly worsened rendering quality, as shown in Tab. A.1.

For Plenoxels, only rectifying σ with exp is insufficient to maintain convergence at various scene scales; a high transmittance offset is needed even at $k = 1$. See Tab. A.2 for numerical results. We note that our results are worse ($\sim$ 2-3 dB) than the results presented in Plenoxels [8]. We had to significantly lower the learning rate to be more suitable for the exp activation, and believe that other changes in the hyperparameters are also needed to match the default performance. We leave this as future work; our results still demonstrate

	chair	drums	ficus	hotdog	lego	materials	mic	ship
disabled	fail	fail	fail	fail	fail	fail	fail	fail
default	34.10	25.40	32.56	36.67	34.53	29.71	33.23	28.76
	fern	**flower**	**fortress**	**horns**	**leaves**	**orchids**	**room**	**trex**
disabled	15.74	17.38	20.77	20.62	16.96	11.77	21.44	20.62
default	24.49	27.61	29.91	27.01	20.41	19.95	30.87	26.41
	bicycle	**bonsai**	**counter**	**garden**	**kitchen**	**room**	**stump**	
disabled	fail	fail	fail	fail	fail	fail	fail	
default	21.98	27.33	25.41	24.41	25.81	28.14	23.51	

Table A.1: PSNR ↑ values of DVGO on the Blender (top row), LLFF (middle row), and Mip-NeRF 360 (bottom row) datasets. DVGO sets the interval lengths d to the ratio of the current voxel grid resolution to the base voxel grid resolution in order to to make the model independent of scene size; this is referred to as 'default' in the table. We observe that disabling this heuristic (i.e., setting the interval lengths in the volume rendering equation to be the true physical distances between any two sample points) results in DVGO failing to render at a high quality. These failure modes are marked with red in the rows titled 'disabled'.

consistent rendering quality and a need for our high transmittance initialization strategy.

Benefit of High Transmittance Initialization. We provide additional RGB-image and depth-map visuals in Fig. A.1 demonstrating the benefits of our high transmittance initialization in handling large scene scales when using exp activation.

A.3 Reproducibility

In Tab. 5.1, we observe random failure modes when training the vanilla 8-layer MLP on various scenes using NeRF-Pytorch [38] at git commit hash `63a5a63`. To get a better sense of the *frequency* of these failure modes, we train Vanilla-NeRF on chair and ship scenes 5 times for each k-value, comparing the default `ReLU` parametrization on σ against our high transmittance initialization in combination with exp. Full results are provided in Tab. A.3. We attribute the random failure modes to poor initialization of the MLP layers (the initial output distribution has 0 mean and 0 variance) and the inability of `ReLU` to provide a smooth

	chair	**drums**	**ficus**	**hotdog**
$k = 0.1$	31.80 / 31.89	24.23 / 24.26	29.41 / 29.58	34.12 / 34.31
$k = 0.4$	30.33 / 31.83	23.61 / 24.23	27.65 / 29.12	31.76 / 34.46
$k = 1.0$	27.06 / 32.00	21.66 / 24.37	24.77 / 29.39	27.31 / 34.56
$k = 2.5$	17.49 / 32.43	17.24 / 24.44	18.89 / 29.83	20.20 / 34.72
$k = 10.0$	13.05 / 32.25	10.84 / 24.45	11.40 / 30.23	13.03 / 34.93
	lego	**materials**	**mic**	**ship**
$k = 0.1$	31.20 / 31.31	28.21 / 28.41	31.11 / 31.31	28.16 / 28.25
$k = 0.4$	29.61 / 31.22	26.06 / 28.02	29.04 / 31.13	27.01 / 28.13
$k = 1.0$	26.22 / 31.35	23.14 / 28.31	24.85 / 31.29	24.58 / 28.19
$k = 2.5$	16.64 / 31.53	15.51 / 28.64	15.89 / 31.61	16.67 / 28.22
$k = 10.0$	11.64 / 31.69	9.57 / 28.79	15.89 / 31.61	11.27 / 28.21

Table A.2: PSNR ↑ for Plenoxels [baseline / ours] at different scene scaling k on the Blender dataset. By default, Plenoxels applies a very large learning rate directly on the coefficients of a voxel grid, where σ is queried via trilinear interpolation, followed by ReLU to ensure non-negative density values. The baseline Plenoxels model here replaces ReLU with `exp` activation and uses a reduced learning rate ($\eta_{\text{init}} = 0.05$, $\eta_{\text{final}} = 0.005$) with no reverse cosine delay. Our model in addition applies a high transmittance offset. As shown, this transmittance offset is required for high rendering quality at various scene sizes, even at $k = 1$.

transition from low to high α values. See Fig. 1.2 for a visualization. However, even with such a poorly initialized MLP, our parametrization on σ is enough to guarantee convergence on all runs across all tested k values.

We train NeRF-Pytorch for only 200k iterations. A longer training schedule of 500k iterations would push the NeRF-Pytorch performance closer, but still slightly below the original NeRF numbers. The overall conclusions do not change.

		Default					Ours				
CHAIR	$k = 0.1$	31.20	31.14	31.04	14.04	28.82	31.20	31.27	31.07	30.99	31.25
	$k = 0.4$	28.96	31.32	31.26	31.17	31.26	31.00	31.29	31.02	31.24	31.09
	$k = 1.0$	14.04	31.32	14.04	14.04	28.82	31.24	31.30	31.11	31.39	31.23
	$k = 2.5$	14.04	31.37	28.99	31.38	31.32	31.14	31.23	31.15	31.03	31.23
	$k = 10.0$	14.04	30.97	28.76	31.00	9.75	31.29	31.19	31.16	31.10	31.22
average		25.76 ± 7.79					31.18 ± 0.10				

		Default					Ours				
SHIP	$k = 0.1$	5.88	5.88	27.59	27.61	5.88	27.20	27.06	27.56	27.51	27.31
	$k = 0.4$	27.57	27.46	27.59	27.35	27.49	27.11	27.14	27.00	27.11	27.84
	$k = 1.0$	27.56	25.64	27.27	25.57	27.60	27.18	26.99	27.16	27.30	27.11
	$k = 2.5$	27.50	27.45	5.88	27.49	27.56	27.56	27.31	27.35	27.18	27.16
	$k = 10.0$	5.88	27.57	27.35	24.12	27.43	26.99	27.00	27.16	27.27	27.28
average		22.89 ± 8.54					27.23 ± 0.20				

Table A.3: PSNR ↑ values of Vanilla-NeRF on the chair and ship scenes from the Blender dataset. Here, we run 5 experiments for 5 different k-values, and compare the results from the default NeRF baseline against our exp parametrization on σ and high transmittance initialization. We observe consistent rendering quality across all runs for both scenes with our method, but identify inconsistent rendering quality for the default configuration, with failure modes and poor convergence marked here in red.

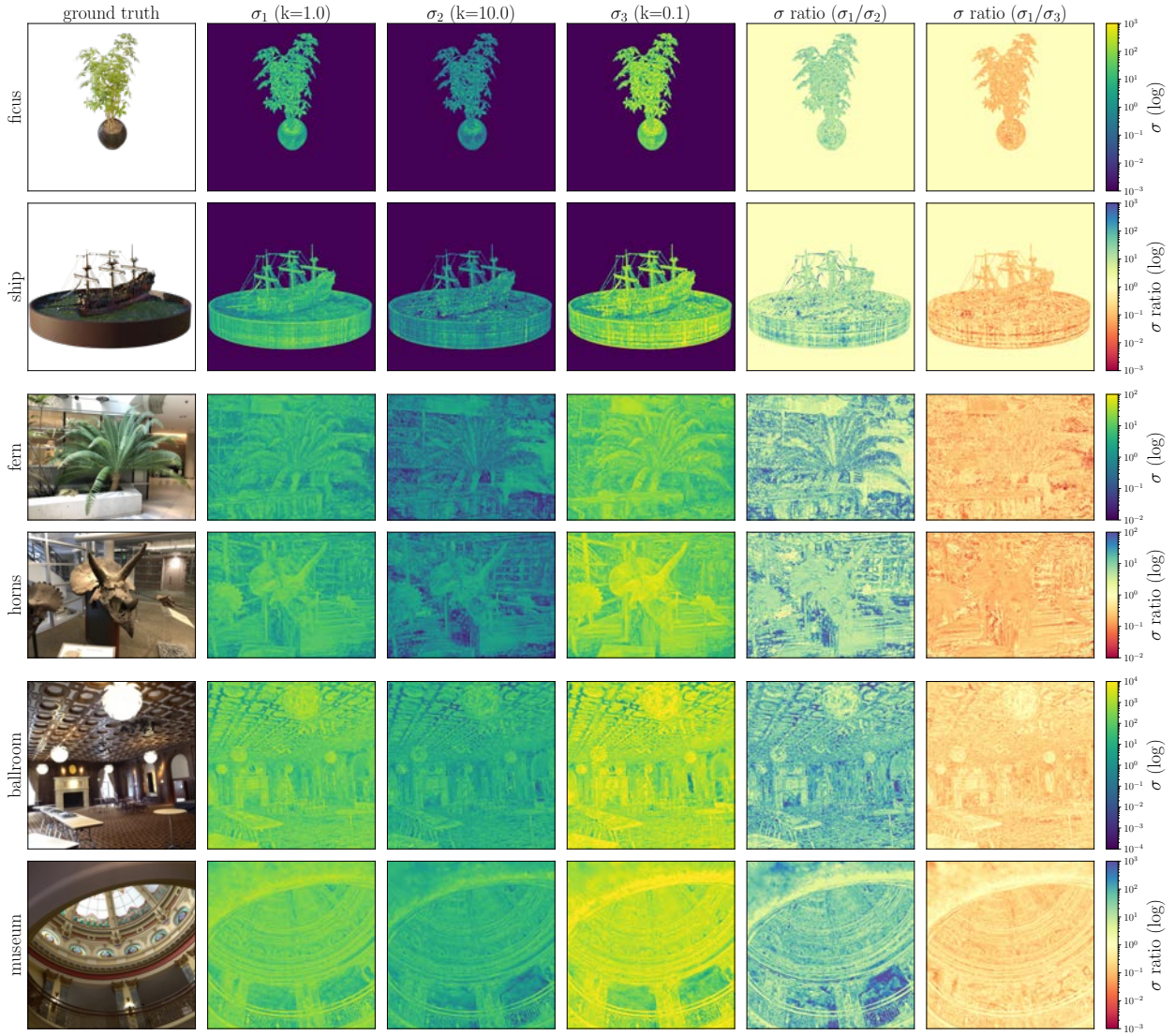

Figure A.2: Visualization of σ-image produced at the 50-th percentile location of each ray's density histogram CDF. We produce a division image that shows the global difference of numerical range across different scene scaling factor k. The first four rows are produced from the TensoRF architecture (the top two rows are the ficus and ship scenes from the blender dataset; the middle two rows are the fern and horns scenes from the LLFF dataset). The bottom two rows are produced from the Nerfacto architecture on the ballroom and museum scenes from the Tanks-and-Temples dataset. Across a variety of scenes and NeRF architectures, these visualizations demonstrate that the phenomenon of alpha invariance holds to a very strong degree.