DeRF: Decomposed Radiance Fields

Daniel Rebain[1], Wei Jiang[1], Soroosh Yazdani[4], Ke Li[2,4], Kwang Moo Yi[1], Andrea Tagliasacchi[3,4]

[1]University of British Columbia [2]Simon Fraser University
[3]University of Toronto [4]Google Research

Abstract

With the advent of Neural Radiance Fields (NeRF), neural networks can now render novel views of a 3D scene with quality that fools the human eye. Yet, generating these images is very computationally intensive, limiting their applicability in practical scenarios. In this paper, we propose a technique based on spatial decomposition capable of mitigating this issue. Our key observation is that there are diminishing returns in employing larger (deeper and/or wider) networks. Hence, we propose to spatially decompose a scene and dedicate smaller networks for each decomposed part. When working together, these networks can render the whole scene. This allows us near-constant inference time regardless of the number of decomposed parts. Moreover, we show that a Voronoi spatial decomposition is preferable for this purpose, as it is provably compatible with the Painter's Algorithm for efficient and GPU-friendly rendering. Our experiments show that for real-world scenes, our method provides up to 3× more efficient inference than NeRF (with the same rendering quality), or an improvement of up to 1.0 dB in PSNR (for the same inference cost).

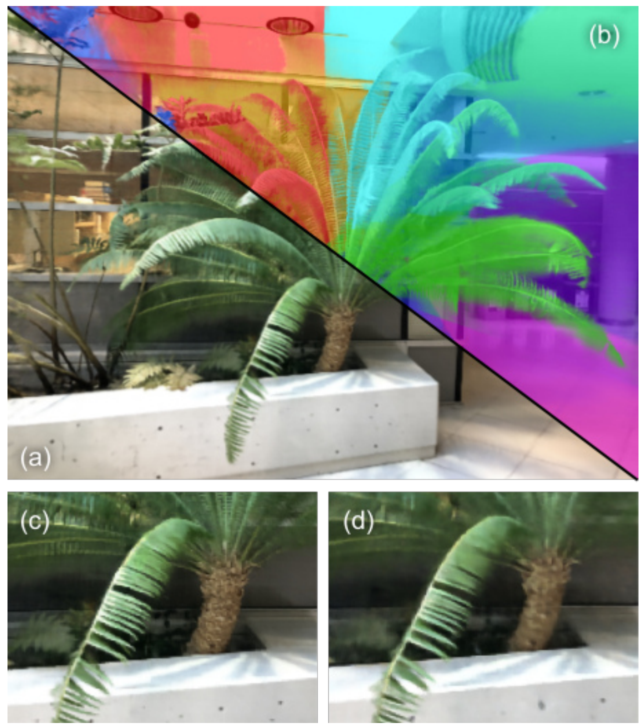


Figure 1. We render a scene (a) from a decomposed neural representation (b), consisting of a collection of spatially localized neural networks. Each of these networks render a *convex* portion of the image, and these are then composited into the output via the Painter's Algorithm. Depending on the level of decomposition, this can lead to faster rendering, or to renderings that have the same runtime, but contain sharper details. In (c), we decompose the scene into 16 parts, which leads to sharper details than (d), with similar runtime.

1. Introduction

While high-quality rendering of virtual scenes has long been associated with traditional computer graphics [16, 14], there have been promising developments in using neural networks for photo-realistic rendering [21, 9, 11, 13]. These *neural rendering* methods have the potential to reduce the amount of human interaction that is needed to digitize the real world. We believe the further development of neural scene representations will increase the viability of 3D content creation in-the-wild and continue pushing neural rendering to higher levels of life-like detail.

Among existing neural rendering methods, those that operate in 3D have lately drawn much interest [19, 9, 13]. Unlike those based on convolutional neural networks [6], these methods do not operate in image-space, and rather train *volumetric* representations of various types: they define functions that can be queried in space during a volume rendering operation. This is essential, as volume rendering introduces an *inductive bias* towards rendering phenomena, so that effects like occlusion and parallax are modeled by construction, rather than being emulated by image-space operations.

However, neural volume rendering is far from being a fully developed technology. The two development axes are

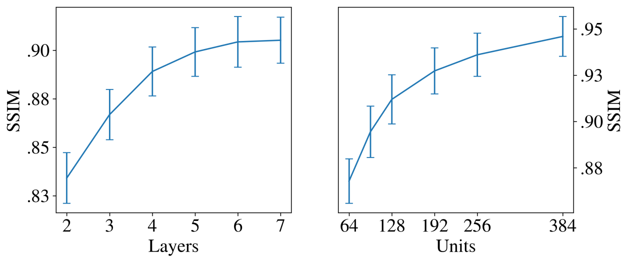


Figure 2. **Diminishing returns** – We sweep through network architectures varying by depth and width to show how the gains in quality diminish with increased capacity. The total number of network parameters varies linearly with network depth (left) and quadratically with the number of units in each layer (right). All networks trained for 300k iterations on the NeRF "room" scene.

generality (removing assumptions about the input, hence allowing their application to more general visual phenomena) and *performance* (increasing the efficiency of training and/or inference). In this paper, we focus on performance, and specifically the *inference* performance. Hence, the natural question then becomes *"why are neural volume rendering models so incredibly slow?"* Let us consider Neural Radiance Fields (NeRF) [13] as the cardinal example. These method requires hundreds of MLP invocations per pixel to compute the samples needed by volume rendering. This results in an extremely compute-intensive inference process needing $\approx 10^8$ network evaluations and *minutes* of computation to render a one megapixel image on a modern NVIDIA RTX 2080 Ti accelerator.

Naturally, to accelerate inference one could trade away model capacity, but doing so naïvely results in lower rendering quality. However, as illustrated in Figure 2, there are *diminishing returns* regarding how the capacity of neural networks (i.e. number of layers or number of neurons per layer) affects final rendering quality. We take advantage of this phenomena and accelerate neural rendering by dividing the scene into *multiple* areas (i.e. spatial decomposition), and employing a *small(er)* networks in each of these areas.

Due to hardware limitations in the memory architecture of accelerators, not all decompositions are appropriate. For example, a random decomposition of the volume would result in random sub-network invocations. Coalescing invocations so that memory access are contiguous is possible, but in our experiments we found the re-ordering operations are not sufficiently fast, and any efficiency gain from using smaller networks was lost. Hence, our question becomes *"Can we design a spatial decomposition that minimizes the chance of random memory access?"* We address this question by noting that we can elegantly overcome these limitations if we decompose space with Voronoi Diagrams [3, Ch. 7]. More specifically, the convex cells of the Voronoi diagram can be rendered *independently*, after which the Painter's Algorithm [3, Ch. 12] can be used to *composite* the final image.

We formulate our Voronoi decomposition to be differentiable, and train end-to-end to find an optimal cell arrangement. By doing so we increase the efficiency of the rendering process by up to a factor of three without *any* loss in rendering quality. Alternatively, with the same rendering cost, we enhance the rendering quality by a PSNR of up to 1.0dB (recall that Peak Signal to Noise Ratio is expressed in log-scale).

Contributions. To summarize, our main contributions are:

- We highlight the presence of diminishing returns for network capacity in NeRF, and propose spatial decompositions to address this issue.
- We demonstrate how a decomposition based on Voronoi Diagrams may be learned to optimally represent a scene.
- We show how this decomposition allows the whole scene to be rendered by rendering each part independently, and compositing the final image via Painter's Algorithm.
- In comparison to the NeRF baseline, these modifications result in improvement of rendering quality for the same computational budget, or faster rendering of images given the same visual quality.

2. Related Work

A large literature exists on neural rendering. We refer the reader to a recent survey [20], and only cover the most relevant techniques in what follows.

Image-space neural rendering. The simplest form of neural rendering resorts to image-to-image transformations via convolutional neural networks [6]. This operation can be aided by 3D reasoning [4, 17, 11, 23], producing an intermediate output that is then again fed to a CNN; regular grids [4, 17] or point clouds [11, 23] have both been used for this purpose. As these works still rely on CNNs to post-process the output, they have difficulty modeling view-dependent effects, often resulting in visible artefacts. Other, non-learned methods have also achieved impressive results by synthesizing novel views directly from the content of the input images [15, 2].

Neural volumetric rendering. Recently, researchers succeeded in integrating 3D inductive bias within a network in a completely end-to-end fashion, hence removing the need CNN post-processing. Instead, they rely on tracing rays through a volume to render an image [18, 19]. While these results pioneered the field, more compelling results were achieved via the use of fixed-function volume rendering [9, 13]. In particular, and thanks to the use of positional encoding, NeRF [13] is able to render novel views of a scene from a neural representation with photo-realistic quality. Extensions of NeRF to dynamic lighting and appearance exist [10], as well as early attempts at decomposing the complexity of the scene into near/far components [25]. With an

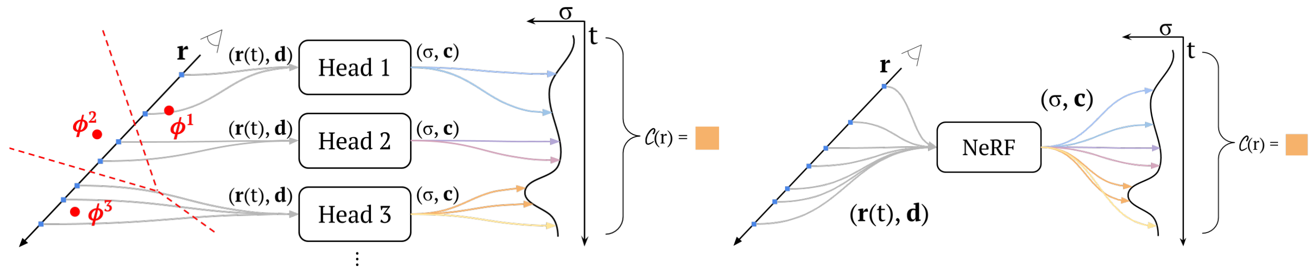


Figure 3. **Framework –** The DeRF architecture (left) consists of a set of independent NeRF (right) networks which are each responsible for the region of space within a Voronoi cell defined by the decomposition parameters ϕ. The final color value for a ray is computed by applying the volume rendering equation to each segment of radiance $\mathbf{c}$ and density σ, and alpha compositing together the resulting colors.

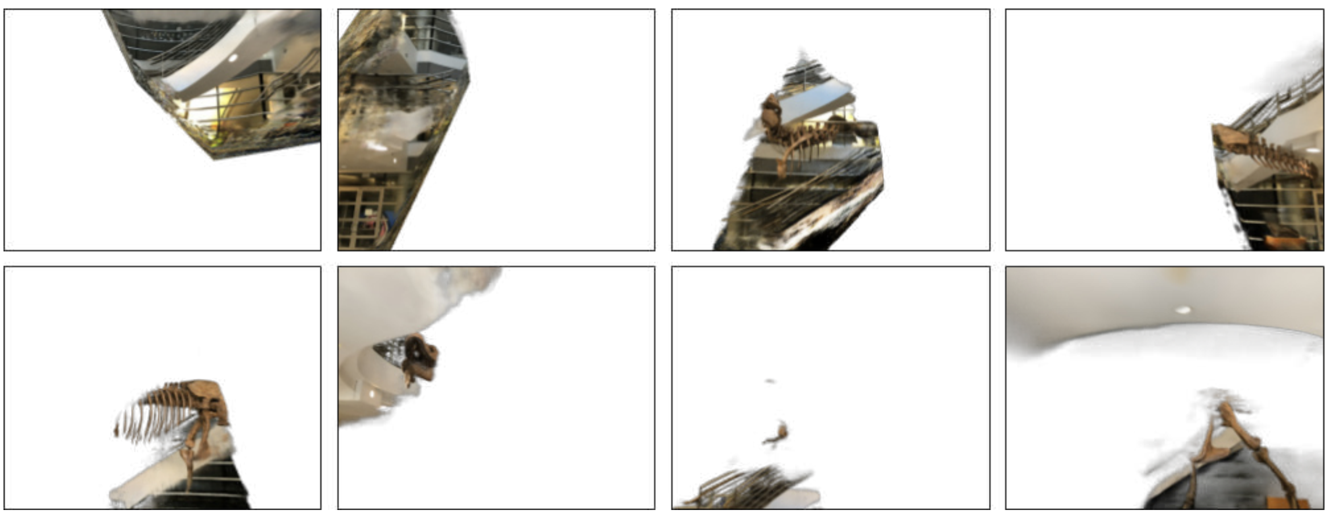


Figure 4. **Decomposed radiance fields** – We visualize each of the rendering heads individually. Note that as each head is rendered *only* the weights of *one* neural network head needs to be loaded, hence resulting in optimal cache coherency while accessing GPU memory.

objective similar to ours, in Neural Sparse Voxel Fields [8], the authors realize a $10\times$ speed-up by discretizing the scene and avoiding computation in empty areas; note this solution is *complementary* to ours. It focuses on the *sampling* part of the NeRF pipeline, and therefore can be used in conjunction with what we propose.

3. Method

We review the fundamentals of NeRF in Section 3.1, describe our decomposition-based solution in Section 3.2, its practical realization with Voronoi Diagrams and the Painter's Algorithm in Section 3.3. We conclude by detailing our training methodology in Section 3.4.

3.1. Neural radiance fields (NeRF)

To represent a scene, we follow the volume rendering framework of NeRF [13]; see Figure 3 (right). Given a camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ corresponding to a *single* pixel, we integrate the contributions of a 5D (3D space plus 2D for direction) radiance field $\mathbf{c}(\mathbf{x}, \mathbf{d})$ and spatial density $\sigma(\mathbf{x})$ along the ray:

$$\mathcal{C}(\mathbf{r}) = \int_{t_n}^{t_f} \mathcal{T}(t)\, \sigma(\mathbf{r}(t))\, \mathbf{c}(\mathbf{r}(t), \mathbf{d})\, dt \tag{1}$$

to obtain a the pixel color $\mathcal{C}(\mathbf{r})$. Here, t_n and t_f are the near/far rendering bounds, and transmittance $\mathcal{T}(t)$ represents the amount of the radiance from position t that will make it to the eye, and is a function of density:

$$\mathcal{T}(t) = \exp\left(- \int_{t_n}^{t} \sigma(\mathbf{r}(s))\, ds \right) \tag{2}$$

The neural fields $\sigma(\mathbf{x})$ and $\mathbf{c}(\mathbf{x}, \mathbf{d})$ are trained to minimize the difference between the rendered and observed pixel values $\mathcal{C}_{\text{gt}}$ over the set of all rays R from the training images:

$$\mathcal{L}_{\text{radiance}} = \mathbb{E}_{\mathbf{r}\sim R} \left[\|\mathcal{C}(\mathbf{r}) - \mathcal{C}_{\text{gt}}(\mathbf{r})\|_2^2 \right] . \tag{3}$$

Note how in neural radiance fields, a *single* neural network is used to estimate $\sigma(\mathbf{x})$ and $\mathbf{c}(\mathbf{x}, \mathbf{d})$ for the *entire* scene. However, as discussed in the introduction, it is advisable

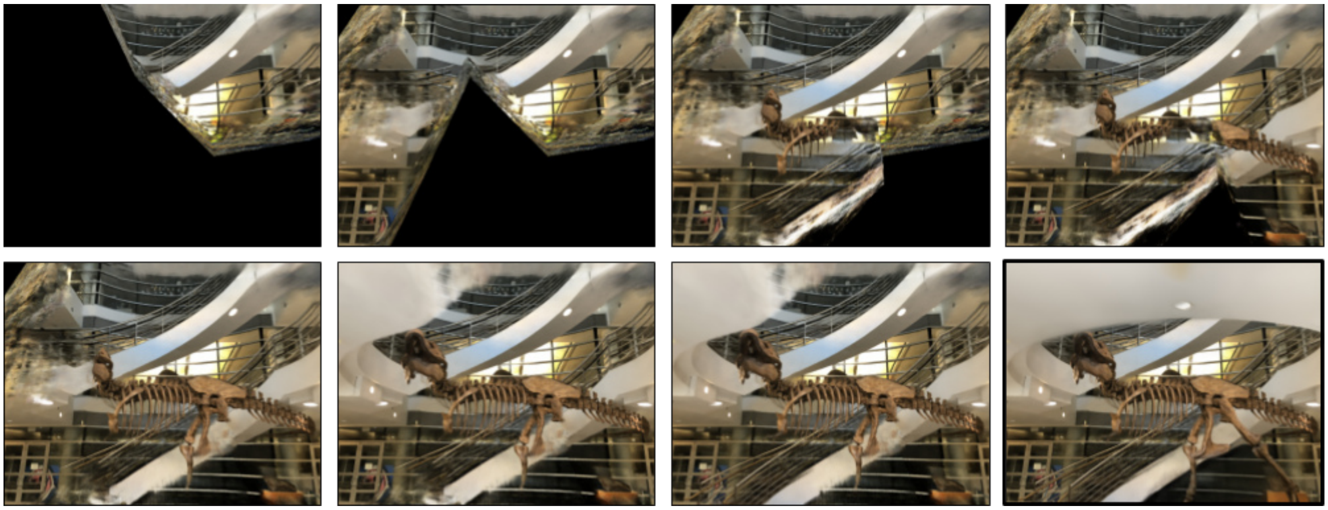


Figure 5. **Compositing with the Painter's algorithm** – Visualization of the intermediate steps of the compositing process.

to use multiple smaller capacity neural networks (heads) to compensate for diminishing returns in rendering accuracy.

3.2. Decomposed radiance fields (DeRFs)

We propose to model the radiance field functions $\sigma(\mathbf{x})$ and $\mathbf{c}(\mathbf{x}, \mathbf{d})$ as a weighted sum of N separate functions, each represented by a neural network (head); see Figure 3 (left). Specifically, the NeRF model defines two directly learned functions: $\sigma_\theta(\mathbf{x})$ and $\mathbf{c}_\theta(\mathbf{x}, \mathbf{d})$, each defined for values of $\mathbf{x}$ over the full space of $\mathbb{R}^3$, and modeled with a neural network with weights θ. Conversely, in DeRF we write:

$$\sigma(\mathbf{x}) = \sum\nolimits_{n=1}^{N} w_\phi^n(\mathbf{x}) \sigma_{\theta_n}(\mathbf{x}) \tag{4}$$

$$\mathbf{c}(\mathbf{x}, \mathbf{d}) = \sum\nolimits_{n=1}^{N} w_\phi^n(\mathbf{x}) \mathbf{c}_{\theta_n}(\mathbf{x}, \mathbf{d}) \tag{5}$$

where n denotes the head index, and $w_\phi(\mathbf{x})$:$\mathbb{R}^3 \mapsto \mathbb{R}^N$ represents our decomposition via a learned function (with parameters ϕ) that is coordinatewise positive and satisfies the property $\|w_\phi(\mathbf{x})\|_1$=1. Each head network is identical in implementation to a single NeRF model.

Efficient scene decomposition. Note how in Eq. (5), whenever $w_\phi^n(\mathbf{x})$=0 there is no need for $\sigma_{\theta_n}(\mathbf{x})$ and $\mathbf{c}_{\theta_n}(\mathbf{x}, \mathbf{d})$ to be evaluated, as their contributions would be zero. Hence, we train our decomposition w_ϕ so that only *one* of the N elements in $\{w_\phi^n(\mathbf{x})\}$ is non-zero at any position in space (i.e. we have a spatial partition). Because of this property, for each $\mathbf{x}$, only *one* head needs to be evaluated, accelerating the inference process.

Balanced scene decomposition. As all of our heads have similar representation power, it is advisable to decompose the scene in a way that all regions represent a similar amount of information (i.e. visual complexity). Toward this objective, we first introduce $\mathcal{W}_\phi(\mathbf{r}) \in \mathbb{R}^N$ to measure how much the N heads contributes to a given ray:

$$\mathcal{W}_\phi(\mathbf{r}) = \int_{t_n}^{t_f} \mathcal{T}(t)\, \sigma(\mathbf{r}(t))\, w_\phi(\mathbf{r}(t))\, dt\ . \tag{6}$$

and employ a loss function that enforces the contributions to be uniformly spread across the various heads:

$$\mathcal{L}_{\text{uniform}} = \|\mathbb{E}_{\mathbf{r}\sim R}\left[\mathcal{W}_\phi(\mathbf{r})\right]\|_2^2\ . \tag{7}$$

Minimizing this loss results in a decomposition which utilizes all heads equally. To see this, let $\mathbf{W}_\phi = \mathbb{E}_{\mathbf{r}\sim R}[\mathcal{W}_\phi(\mathbf{r})]$, and let $\mathbf{1} \in \mathbb{R}^N$ be the vector with all 1's. Recall that since $\|w_\phi(\mathbf{x})\|_1$=1 and is coordinatewise positive, we get that $\mathbf{1} \cdot w_\phi(\mathbf{x}) = 1$. Therefore, $\mathbf{1} \cdot \mathcal{W}_\phi(\mathbf{r})$ will be independent of ϕ, and hence $\mathbf{1} \cdot \mathbf{W}_\phi$ will be a constant. Finally, by the Cauchy-Schwartz inequality, we get that $\|\mathbf{1}\|_2^2\|\mathbf{W}_\phi\|_2^2$ is minimized when $\mathbf{W}_\phi$ is parallel to $\mathbf{1}$, which means that all heads contribute equally.

3.3. Voronoi learnable decompositions

We seek a decomposition satisfying these requirements:

1. it must be differentiable, so that the decomposition can be fine-tuned to a particular scene
2. the decomposition must be a spatial *partition* to unlock efficient evaluation, our core objective
3. it must be possible to evaluate the partition in an accelerator-friendly fashion

Towards this objective, we select a Voronoi Diagram as the most suitable representation for our decomposition. We employ the solution proposed in [24], which defines, based on a set $\phi \in \mathbb{R}^{N\times 3}$ of N Voronoi sites, a differentiable (i.e. soft) Voronoi Diagram as:

$$w_\phi^n(\mathbf{x}) = \frac{e^{-\beta||\mathbf{x}-\phi^n||_2}}{\sum_{j=1}^N e^{-\beta||\mathbf{x}-\phi^j||_2}} \tag{8}$$

where $\beta \in \mathbb{R}^+$ is a temperature parameter controlling the softness of the Voronoi approximation. This decomposition is: ① differentiable w.r.t. its parameters ϕ, and has smooth gradients thanks to the soft-min op in (8); ② a spatial partition for $\beta \to \infty$, and thanks to the defining characteristics of the Voronoi diagram; ③ compatible with the classical "Painter's Algorithm", enabling efficient compositing of the rendering heads to generate the final image. An example of the trained decomposition is visualized in Figure 4.

Painter's Algorithm. This algorithm is one of the most elementary rendering techniques; see [3, Ch. 12]. The idea is to render objects independently, and draw them *on top* of each other in order, from back to front, to the output buffer (i.e. image). Such an ordering ensures that closer objects occlude further ones. This algorithm has only found niche applications (e.g. rendering transparencies in real-time graphics), but it encounters failure cases when the front to back ordering of objects in the scene is non decidable; see a simple example in the inset figure, where three convex elements form a cycle in the front/behind relation graph.

Figure 6. A simple scene made of convex elements that the Painter's Algorithm cannot render.

In our solution, the scene can be rendered part-by-part, one Voronoi cell at a time, without causing memory cache incoherences, leading to better GPU throughput. We can then composite the cell images back-to-front, via the Painter's Algorithm, to generate the final image; see Figure 5. However, we need to verify that Voronoi decompositions are compatible with the Painter's Algorithm.

On the correctness of the Voronoi Painter's. Taking an approach similar to [5], we prove that our Voronoi decomposition is compatible with the Painter's Algorithm. To do so, we will show that for any Voronoi decomposition and a camera located at Q, there is a partial ordering of the Voronoi cells so that if V shows up before W in our ordering, then W does not occlude any part of V (i.e. no ray starting from Q to any point in V will intersect W). To this end, let $\mathcal{P} \subset \mathbb{R}^n$ be a set of points, and for each $P \in \mathcal{P}$ let V_P be the Voronoi cell attached to P. For any $Q \in \mathbb{R}^n$ define $<_Q$ on the Voronoi cells of

$$V_{P'} <_Q V_P \text{ if and only if } d(P', Q) < d(P, Q), \tag{9}$$

where d defines the distance between points. This clearly defines a partial ordering on $\mathcal{P}$. We now show that this partial ordering is the desired partial ordering for the Painter's Algorithm. Let $(x, x') \in V_P \times V_{P'}$ and let $x' = \lambda x + (1-\lambda)Q$ for $\lambda \in (0,1)$ (i.e. x' is on the line segment (x, Q)) and hence parts of $V_{P'}$ is covering V_P. We now need to show that $V_{P'} <_Q V_P$, or equivalently $d(P', Q) < d(P, Q)$. Let

$$H = \{z \,|\, d(z, P) < d(z, P')\} \tag{10}$$

be the halfspace of points closer to P. Note that $d(x, P) < d(x, P')$ (since $x \in V_P$) and $d(x', P) > d(x', P')$, and hence $x \in H$ and $x' \notin H$. If $Q \in H$ then the line segment (x, Q) will intersect the boundary of H twice, once between (x, x') and once between (x', Q), which is not possible. Therefore $Q \notin H$, which implies $d(P', Q) < d(P, Q)$ as desired.

3.4. Training details

When training our model, we find that successful training is only achieved when the decomposition function w_ϕ is trained *before* the network heads for density σ_{θ_n} and radiance $\mathbf{c}_{\theta_n}$. This allows the training of the primary model to proceed without interference from shifting boundaries of the decomposition. However, as shown in Eq. (6), the training of w_ϕ requires a density model σ. To resolve this problem, we first train coarse networks σ_{coarse} and $\mathbf{c}_{\text{coarse}}$ that apply to the entire scene, before the networks heads σ_{θ_n} and $\mathbf{c}_{\theta_n}$ are trained; this pre-training stage lasts ≈100k iterations in our experiments. During the pre-training stage, to stabilize training even further, we optimize θ_{coarse} and ϕ *separately*, respectively minimizing the reconstruction loss $\mathcal{L}_{\text{radiance}}$ and uniformity loss $\mathcal{L}_{\text{uniform}}$. We found through experiment that allowing $\mathcal{L}_{\text{uniform}}$ to affect the optimization of θ_{coarse} inhibited the ability of the density network to properly learn the scene structure, resulting in unstable training. Once w_ϕ is pre-trained, we keep ϕ fixed and train the per-decomposition networks σ_{θ_n} and $\mathbf{c}_{\theta_n}$ with the $\mathcal{L}_{\text{radiance}}$, as the Voronoi sites are fixed, and $\mathcal{L}_{\text{uniform}}$ is no longer is necessary. The training process can be summarized as follows:

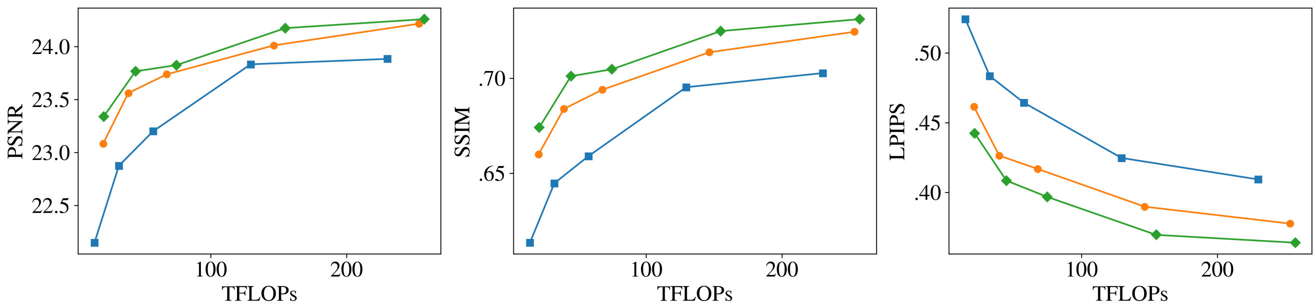
$$\theta^*_{\text{coarse}} = \underset{\theta_{\text{coarse}}}{\arg\min}\, \mathcal{L}_{\text{radiance}}(\mathcal{C}_{\theta_{\text{coarse}}}) \tag{11}$$

$$\phi^* = \underset{\phi}{\arg\min}\, \mathcal{L}_{\text{uniform}}(\mathcal{W}_{\theta^*_{\text{coarse}}, \phi}) \tag{12}$$

$$\{\theta^*_0, ..., \theta^*_n\} = \underset{\{\theta_0, ..., \theta_n\}}{\arg\min}\, \mathcal{L}_{\text{radiance}}(\mathcal{C}_{\{\theta_0, ..., \theta_n\}, \phi^*}) \tag{13}$$

The first two steps can be implemented as simultaneous optimizations to reduce training time, and form our pre-training phase.

Controlling the temperature parameter. The soft-Voronoi diagram formulation in (8) leads to differentiability w.r.t the Voronoi sites (as otherwise gradients of the weight function would be zero), but efficient scene decomposition (Section 3.2) requires spatial *partitions*. Hence, we define a scheduling for β over the training process. We start at sufficiently low value so that $w^i(\mathbf{x}) \approx w^j(\mathbf{x})$ for all i and j. As the training progresses, we exponentially increase β until it reaches a sufficiently high value ($10e9$ in our experiments), so that the decomposition is indistinguishable



Units	1 Head (□)			4 Heads (○)			8 Heads (◇)		
	PSNR ↑	SSIM↑	LPIPS ↓	PSNR↑	SSIM↑	LPIPS ↓	PSNR↑	SSIM↑	LPIPS ↓
64	22.15	0.61	0.52	23.08	0.66	0.46	23.34	0.67	0.44
96	22.87	0.64	0.48	23.56	0.68	0.43	23.77	0.70	0.41
128	23.20	0.66	0.46	23.74	0.69	0.42	23.82	0.70	0.40
192	23.83	0.70	0.42	24.01	0.71	0.39	24.17	0.72	0.37
256	23.88	0.70	0.41	24.22	0.72	0.38	**24.26**	**0.73**	**0.36**

Figure 7. **Quality vs. efficiency** – Reconstruction quality versus run-time inference cost for the "fern" scene as the network capacity (number of hidden units) is changed; the table report the data used to draw the diagrams. To make the computational requirements tractable, we lower sample counts (128 per ray) and batch sizes (512) are used than for results reported in [13], and thus are not directly comparable. For quantitative results for other scenes, please refer to the appendix.

from a (hard) Voronoi diagram; i.e. either $w^i(\mathbf{x}) \approx 1$ or $w^i(\mathbf{x}) \approx 0$ for all i.

Parameter magnitude scaling. When using an optimizer such as ADAM [7], the training process is not necessarily invariant with respect to the magnitude of individual variables. As a result, when using a common optimizer state for all trainable variables, we found that scaling the scene coordinate system such that $\phi \in [-1, 1]^{N\times 3}$ was necessary to prevent the training of the decomposition from oscillating excessively or progressing too slowly.

Units	1 Head		4 Heads		8 Heads	
	TFLOPs ↓	Time ↓	TFLOPs ↓	Time ↓	TFLOPs ↓	Time ↓
64	**14.5**	**14.4**	19.6	20.7	21.1	21.2
96	26.0	32.4	29.2	39.3	37.7	44.6
128	34.4	57.5	46.0	67.6	55.9	74.7
192	83.3	129.4	87.2	146.5	111.0	155.0
256	115.7	230.1	160.9	253.5	186.6	257.4

Table 1. Quantitative comparison of average TeraFLOPs and seconds needed to generate a frame for the "fern" scene on an NVIDIA v100 GPU; for both metrics smaller values are better.

4. Results

We now provide our empirical results. We first detail the experimental setup in Section 4.1. We then present how the theory of our method translates to practice, and how our method performs under various computation loads in Section 4.2. We then discuss other potential decomposition strategies in Section 4.3.

4.1. Experimental setup

To validate the efficacy of our method, we use the "Real Forward-Facing" dataset from NeRF [13]. The dataset is composed of eight scenes, which includes 5 scenes originally from [12], each with a collection of high-resolution images, and corresponding camera intrinsics and extrinsics. As we are interested in the relationship between rendering quality and the computational budget in a practical scenario, we focus on real images. For results on the NeRF [13] synthetic and DeepVoxels [18] synthetic datasets, see appendix.

Implementation. We implement our method in in TensorFlow 2 [1]. Due to the large number of evaluation jobs, we train with some quality settings reduced: we use a batch size of 512 and 128 samples/ray. We train each model for 300k iterations (excluding the decomposition pre-training). Other settings (including the optimizer) are the same as reported in NeRF [13]. The amount of time needed to complete training jobs varies significantly depending on configuration, but a representative example is the 8-head, 128-unit variant which is comparable in quality to a single NeRF using the original settings, and trains fully in 16 hours using two v100 GPUs.

4.2. Efficiency of DeRF

Theory vs. practice – Table 1. In theory the decomposition should not cause any significant increase in computation – the only increase would be to decompose and then merge rendering results. We validate that this is in fact the case by evaluating the number of rendered frames-per-TeraFLOP (theoretical performance) as well as the number

Figure 8. **Qualitative comparison** – A direct comparison of the results of a standard NeRF model with 256 hidden units (left), and a 16-head DeRF model with 128 units (right). The DeRF model has better *performance* and *quality* than the baseline while using networks which each have one quarter the number of parameters. In other words, despite our DeRF having $16 \times \frac{1}{4} = 4\times$ more parameters than the baseline NeRF, it executes 1.7× faster. See Figure 7 for metrics on various model combinations for this scene.

of frames-per-second (practical performance). As shown in Table 1, in terms of FLOPs, the difference between using no-decomposition and 8-decompositions is 62.8% in maximum, and 49.8% in average, whereas there is a quadratic increase in computation should more units be used.

These theoretical trends are mirrored into those of actual runtime. While there can be an increase in computation time compared to the operation count, the maximum increase when going from no-decomposition to 8-decomposition is 47.5%, and the average is 29.3%. Again, this is much more efficient than increasing the number of neurons. This highlights the efficacy of our decomposition strategy – Voronoi – allowing theory to be applicable in practice. Note that in Section 4.3, we show that this is not necessarily the case for other naïve strategies.

Quality vs. efficiency – Figure 7. We further evaluate how the quality of rendering changes with respect to the number of decompositions, and the number of neurons used. To quantify the rendering quality, we rely on three metrics:

- Peak Signal to Noise Ratio (PSNR): A classic metric to measure the corruption of a signal.
- Structural Similarity Index Measure (SSIM) [22]: A perceptual image quality assessment based on the degradation of structural information.
- Learned Perceptual Image Patch Similarity (LPIPS) [26]: A perceptual metric based on the deep features of a trained network that is more consistent with human judgement.

We summarize the results for a representative scene in Figure 7. As shown, given the same render cost, more fine-grained decompositions improve rendering quality across all metrics. Regardless of the computation, using more decomposition leads to better rendering quality.

Qualitative results – Figure 8 and Figure 9. We further show a qualitative comparison between a standard NeRF model and DeRF in Figure 8, where we show that our method outperforms NeRF in terms of both quality and efficiency. More qualitative results are also available in Figure 9 and the video supplementary.

4.3. Alternative decomposition methods

We further empirically demonstrate that naïve decomposition strategies are insufficient, and hence the importance of using our Voronoi decomposition.

Decompositions with MLPs – Table 2. An obvious first thought into decomposing scenes would be to leave the decomposition to a neural network, and ask it to find the optimal decomposition through training. To compare against this baseline, we implement a decomposition network with an MLP with a softmax activation at the end to provide values of $w^n_\phi(\mathbf{x})$. We show the actual rendering time compared to ours in Table 2. While in theory this method should require a similar number of operations to ours, due to the random memory access during the integration process, they can not be accelerated as efficiently. Consequently, their runtime cost tends to grow much faster than DeRF models as we increase the number of heads.

Decompositions with regular grids. A trivial spatial decomposition could be achieved by using a regular grid of network regions. While this would eliminate the requirement to train the decomposition, in practice it would require many more regions in total to achieve the same level of accuracy (due to the non-homogeneous structure of real scenes). Due to the curse of dimensionality, this will also result in a significant amount of incoherence in the memory access pat-

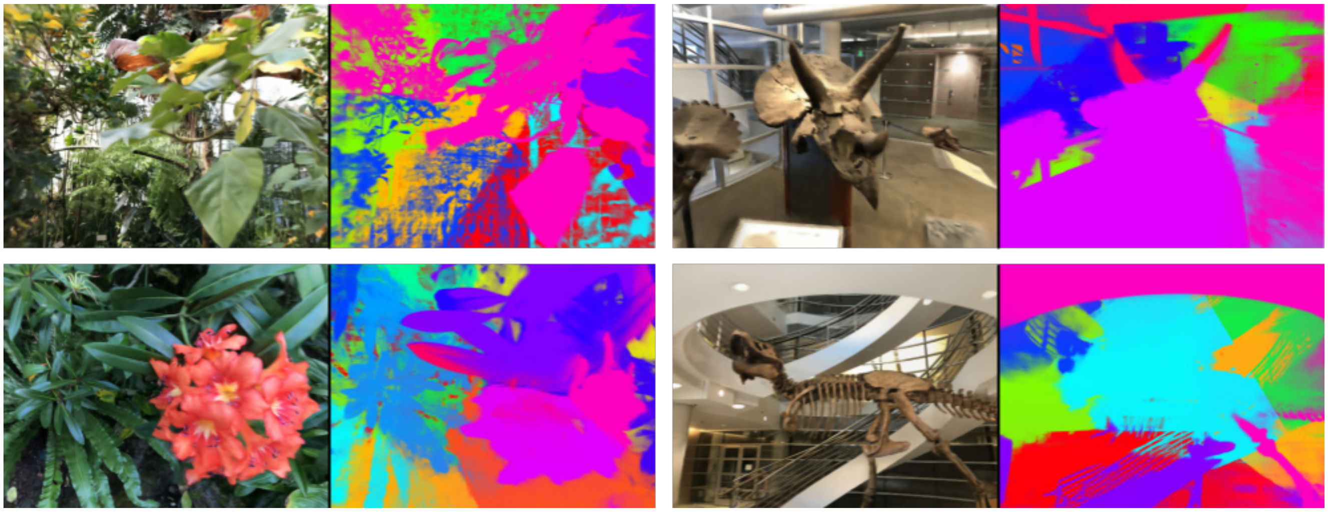
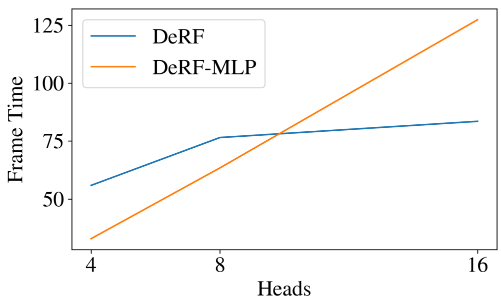


Figure 9. **Qualitative results gallery** – A sampling of DeRF renders, alongside visualizations of decompositions.



	4 Heads	8 Heads	16 Heads
DeRF	55.88s	76.51s	**83.50s**
DeRF-MLP	**32.89s**	**63.40s**	127.32s

Table 2. **Decomposition baselines –** The frame render time for the "room" scene (in seconds) for a Voronoi and MLP-decomposition. Note that as the number of heads increases, the Voronoi decomposition provides a substantial benefit to efficiency. Note models beyond 16 heads are infeasible to train due to memory requirements (for reasonable network widths).

tern for network weights, resulting in lower computational performance; see the appendix.

5. Conclusions

We have presented DeRF – *Decomposed Radiance Fields* – a method to increase the inference efficiency of neural rendering via spatial decomposition. By decomposing the scene into multiple cells, we circumvent the problem of diminishing returns in neural rendering: increasing the network capacity *does not* directly translate to better rendering quality. To decompose the scene, we rely on Voronoi decompositions, which we prove to be compatible with the Painter's algorithm, making our inference pipeline GPU-friendly. As a result, our method not only renders much faster, but can also deliver higher quality images.

Limitations and future work. There are diminishing returns with respect to the number of decomposition heads, not just network capacity; see Figure 7. Yet, one is left to wonder whether the saturation in rendering quality could be compensated by significantly faster rendering as we increase the number of heads in the hundreds or thousands. In this respect, while in this paper we assumed all heads to have the same neural capacity, it would be interesting to investigate heterogeneous DeRFs, as, *e.g.*, a 0-capacity head is perfect for representing an empty space. On a different note, the implementation of highly efficient *scatter/gather* operations could lead to an efficient version of the simple MLP solution in Section 4.3, and accelerate the training of DeRFs, which currently train slower than models without decomposition.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant, NSERC Collaborative Research and Development Grant, Google, Compute Canada, and Advanced Research Computing at the University of British Columbia.

We thank Ricardo Martin Brualla for his comments.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Gaurav Chaurasia, Sylvain Duchêne, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics*, 2013.

[3] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry*. Springer, 1997.

[4] Emilien Dupont, Miguel Angel Bautista, Alex Colburn, Aditya Sankar, Carlos Guestrin, Josh Susskind, and Qi Shan. Equivariant Neural Rendering. *arXiv Preprint*, 2020.

[5] Herbert Edelsbrunner. An Acyclicity Theorem for Cell Complexes in D Dimension. *Combinatorica*, 1990.

[6] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural Scene Representation and Rendering. *Science*, 2018.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

[8] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural Sparse Voxel Fields. *Advances in Neural Information Processing Systems*, 2020.

[9] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural Volumes: Learning Dynamic Renderable Volumes from Images. *ACM Transactions on Graphics*, 2019.

[10] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. *arXiv Preprint*, 2020.

[11] Moustafa Meshry, Dan B Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. Neural Rerendering in the Wild. In *Conference on Computer Vision and Pattern Recognition*, 2019.

[12] 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 Transactions on Graphics*, 2019.

[13] 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 *European Conference on Computer Vision*, 2020.

[14] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *ACM SIGGRAPH Asia*, 2019.

[15] Eric Penner and Li Zhang. Soft 3d reconstruction for view synthesis. 2017.

[16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2016.

[17] Konstantinos Rematas and Vittorio Ferrari. Neural Voxel Renderer: Learning an Accurate and Controllable Rendering Tool. In *Conference on Computer Vision and Pattern Recognition*, 2020.

[18] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Niessner, Gordon Wetzstein, and Michael Zollhofer. DeepVoxels: Learning Persistent 3D Feature Embeddings. In *Conference on Computer Vision and Pattern Recognition*, 2019.

[19] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations. In *Advances in Neural Information Processing Systems*, 2019.

[20] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Niessner, et al. State of the Art on Neural Rendering. *Eurographics 2020 State of The Art Report*, 2020.

[21] Justus Thies, Michael Zollhöfer, and Matthias Niessner. Deferred Neural Rendering: Image Synthesis Using Neural Textures. *ACM Transactions on Graphics*, 2019.

[22] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 2004.

[23] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. SynSin: End-to-End View Synthesis from a Single Image. In *Conference on Computer Vision and Pattern Recognition*, 2020.

[24] Francis Williams, Jerome Parent-Levesque, Derek Nowrouzezahrai, Daniele Panozzo, Kwang Moo Yi, and Andrea Tagliasacchi. Voronoinet: General functional approximators with local support. In *Conference on Computer Vision and Pattern Recognition Workshops*, 2020.

[25] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv Preprint*, 2020.

[26] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Conference on Computer Vision and Pattern Recognition*, 2018.