FastNeRF: Accelerating Ray Rendering of Neural Radiance Fields

Trevor Houchens Joshua Pierce Brandon Li Daniel Kostovetsky

Abstract

Utilizing Neural Radiance Fields (NeRF) for the representation of scenes with complex geometry and appearance achieves impressive results with regard to photorealistic view synthesis. Because NeRF represents scenes as continuous volumetric functions, it additionally holds an advantage over other 3D formats such as point clouds, voxel grids, and meshes in that its memory footprint is an order of magnitude smaller. We seek to investigate optimizations on the originally proposed NeRF paper, specifically related to rendering.

1. Introduction

Rendering photorealistic perspectives of a scene from a free viewpoint has numerous applications and has posed a long-standing challenge in computer graphics. Recent work by Mildenhall *et al*. [4] has outlined a method for implicitly representing a scene as a learned radiance function. In this work, a neural network (or *radiance field*) is trained to return a color and density given a spatial location and spherical direction from within the 3D scene. Using this function, views of the learned scene are rendered from novel camera angles via a volumetric ray-sampling technique. NeRF produces detailed, photorealistic renderings without suffering the cubic memory complexity of competing voxel-based methods. Furthermore, the inclusion of viewing direction among the network's inputs enables it to precisely handle non-Lambertian reflection. In total, NeRF was a promising innovation with potential applications in graphics, virtual reality, and robotics.

Although its results are impressive, NeRF's biggest drawback from a practical perspective is inference time: volumetric rendering requires querying the network at scores of samples along each ray for each rendered pixel. As a result, a single high-definition image can take over 30 seconds to render on a GPU, making NeRF incompatible with real-time applications.

We propose multiple efficiency improvements to the NeRF rendering pipeline. Our methods seek to decrease the required number of network queries and corresponding rendering time by more efficiently allocating samples within the 3D scene. This entails two distinct approaches: i) sparsely selecting rays and using the resulting densities to estimate the ideal sampling locations of adjacent rays; ii) efficiently allocating samples along each ray by bounding the network queries near an estimation of the scene's surface. Across these approaches we explore the trade-off between rendering speed and resolution.

2. Background

Conceptually, the task of novel view synthesis entails aligning a geometric transformation with the underlying details of the scene. Prior approaches to this task have spanned several methods of scene representations, including: voxel grids, point clouds and multi-plane images. These approaches balance trade-offs between memory demands, computation time, and resolution. Voxels [5] offer quicker rendering times, however struggle with a cubic increase in memory demand and pre-defined limits on resolution. Point clouds [1] have notably been used in large-scale scene reconstructions, however their methods are highly dependent on the quality of structure-from-motion techniques and do not offer the same level of photo-realism. Multi-plane images [6] allow for fast rendering, however are limited in the angles by which novel views can be synthesized and also have substantial space requirements.

By comparison, implicit scene representations offer a memory-compact, continuous function that can render novel views from any angle. By modelling both the source and angle of light radiance and over-fitting to a single scene, NeRF is able to generate detailed, novel views with realistic specular effects. The NeRF's volumetric rendering method underlies its efficacy towards this end. When rendering a view, each pixel corresponds to a ray passing through the scene. To render the corresponding color of a given pixel, NeRF queries the radiance field for colors and densities along this ray. The resulting colors and densities are composited via a volumetric rendering technique, where colors from any point sampled along the ray are weighted by a factor of their density and the lack of accumulated densities from the points preceding them on the ray.

$$\widehat{C}(\mathbf{r}) = \sum_{i=1}^{N} \frac{1 - \frac{1}{e^{(\sigma_i \delta_i)}}}{e^{(\sum_{j=1}^{i-1} \sigma_j \delta_j)}} c_i \tag{1}$$

In this equation for calculating the color, $\widehat{C}.$ of ray $\mathbf{r}$, N rep-

resents the number of points sampled along the ray, σ_i represents the density of point i, and δ_i represents the distance between the adjacent point on the ray. At training time, the rendered pixel colors are compared to ground-truth colors of training images, thus teaching NeRF a spatial distribution of color-densities that comport with the multiple training angles. In order to improve the focus of sampling near the surface, NeRF uses this weighted ratios of densities and accumulated densities to distribute a second pass of sampling along each ray:

$$w_i = \frac{1 - \frac{1}{e^{(\sigma_i \delta_i)}}}{e^{(\sum_{j=1}^{i-1} \sigma_j \delta_j)}}, \widehat{w}_i = \frac{w_i}{\sum_{j=1}^{N_c} w_j} \tag{2}$$

Thus, a second pass of sampling is distributed along the ray by the piecewise-constant probability density function given by $\widehat{w}_i$.

Provided the success of NeRF in rendering photo-realistic views, recent research has sought to address the limitation of its substantial rendering time. Recent work by Liu *et al*. [3] outline a method of constructing a sparse voxelization of the 3D scene and learning local radiance fields within each voxel. This entails applying local feature representations to the corners of each voxel and then using location in the voxel to learn a local radiance field from the trilinear interpolation of these voxel features. By tightly bounding the sampling of the radiance fields within the voxel space around the scene's surface, Neural Sparse Voxel Fields avoids sampling empty space and achieves speed-ups by a factor of ten compared to NeRF.

More recent work by Lindell *et al*. [2] takes a more fundamental approach towards improving NeRF's volumentric rendering. The authors define an *integral network* which will be used to return the definite integrals of estimated density along each ray, rather than densities from a single point. The structure of the corresponding *gradient network* is derived from the *integral network*. This *gradient network* is trained to generate colors and densities in a similar manner as NeRF. However at testing, the assembled *integral network* is used to evaluate definite integrals along each ray, thus obviating the need to estimate the integrated density with numerous samples per ray. This direct estimation of ray integrals results in a speed-up by factor of ten.

3. Methods

Our approach to speeding up NeRF fundamentally relies on improving the allocation of our sampling. By more efficiently querying the radiance network near the surface of the scene, fewer samples and less computation would be required to render a novel view. Towards this end, we explore two different approaches: i) sparsely selecting rays to render and using their resulting densities to interpolating the sampling location for adjacent rays; ii) extracting an estimated shape of the scene to bound the sampling depth of rays near the scene's surface.

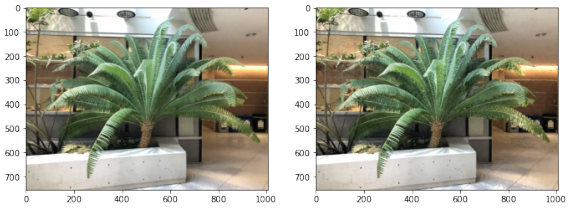

Figure 1. To demonstrate the limits in efficiency and resolution in relation to sampling location, we compare the result of the full 64-sample, first-pass volumetric rendered image (*left*) to the rendered result of only one-sample per ray (*right*), where each sample is located at the surface of the scene as estimated from the densities of the left-image. The right-image required just 2% of the rendering time.

3.1. Interpolated Ray Rendering

The regular NeRF rendering process is a two pass approach. First, each ray is sampled 64 times, and the densities at these sampled points are used to inform a second sampling. Our Interpolated Rendering approach relies on the assumption that the depth of a scene at one pixel can be estimated by the depths of its neighbors. With Interpolated Rendering we segment the novel view into tiles of equally-sized pixel blocks. The traditional first pass sampling is used to acquire density probability distributions and colors for the upper left-hand pixel of each block. We can then use the density bi-linear interpolation of the four neighboring density distributions to infer the density of rays that weren't sampled in the first pass.

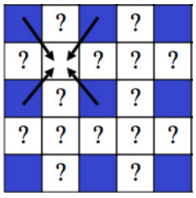

Figure 2. If the above image is a view of a scene, we may choose to render rays for the blue pixels only during the first round of sampling. Using our interpolated rendering, the white pixels can be sampled according to a probability distribution that is the bi-linear interpolation of the density distributions of the neighboring four pixels during the second round.

3.2. Point Cloud Sampling

As opposed to Interpolated Rendering, Point Cloud Sampling renders all the rays of a novel view, however this approach guides sampling depths to be focused more closely on the estimated scene surface. The first step in the process

is to generate a point cloud of the scene using the training images and their estimated depths (NeRF provides depth estimates). The point cloud produced by lifting these points into 3D space gives us an estimate of the surface of the scene. Now when rendering a novel view, we can transform the shape's point cloud into the corresponding normalized device coordinates. By iterating over the points in the point cloud, we can map their corresponding depth estimates to the aligned pixel of the 2D view being rendered. By tracking the largest and smallest depth values for each pixel, we can bound our ray-sampling between the closest and farthest point-cloud depths. If no points correspond to a given pixel, its bounds are set to (0,1) which represent the near plane to infinity in world space, and are the values normally used for rendering. If only a single point, or very few points, are mapped to a specific pixel, the bound on their depth may be very narrow (or possibly a single point). To relax the bounds, we group bounds across pixels and also modify the bounds by a constant factor, ϵ, which we set to 0.10 in our code.

$$\text{near}[i,j] = \min(\text{near}[i-2:i+2, j-2:j+2]) - \epsilon \quad (3)$$

$$\text{far}[i,j] = \max(\text{far}[i-2:i+2, j-2:j+2]) + \epsilon \quad (4)$$

Near and far bounds are clipped to be within the range (0,1). During the rendering process, rays are sampled between their near and far bound rather than across the whole range of (0,1).

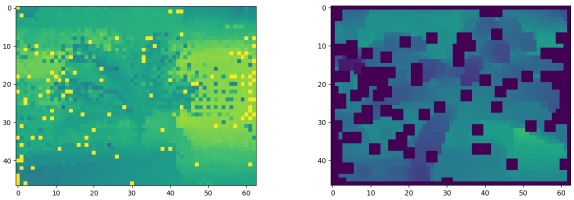

Figure 3. The left picture shows the far depth bound of the fern scene before it is modified by a local max or ϵ. The right picture shows the near bound of the fern scene after these modifications. Note that the near bound is darker in color indicating lower values. The squares are pixels that had no points from the point cloud in their field of view, so the near and far values were set to 0 and 1.

4. Results

4.1. Interpolated Rendering

We explored the results of the interpolated rendering technique across increasing levels of interpolation; in other words, across increasingly sparse initial samples rates per rendered scene. The most aggressive rate of interpolation was sampling one out of every 18^2 rays. The resulting rendered image is shown in Figure 4 and demonstrates convincing, photo-realistic results. None-the-less, NeRF did

Figure 4. Results from interpolated rendering, where only one out of every 18^2 rays relied on sampling the entire ray

report a marginally higher PSNR (shown in Fig 5). While increasing the level of interpolation, the PSNR slightly decreased. Figure 5 makes it appear that NeRF significantly

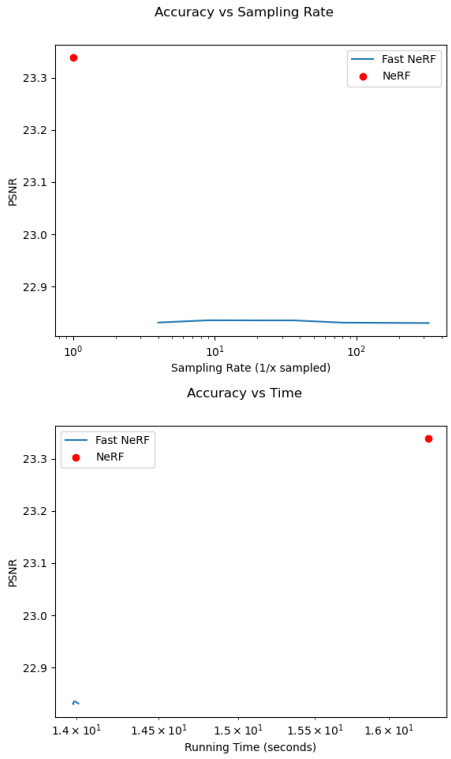

Figure 5. The first chart here shows the effect of sampling rate on the PSNR of the rendered image. The red dot represents NeRF performance, and the blue line represents NeRF with interpolated rendering, for different sparsities of rendered rays in the first pass. The second chart is similar, but running time is shown instead of the sampling rate

outperformed our interpolated rendering version, but this is mostly due to the scaling on the y-axis. Our technique actually had a very similar PSNR to NeRF, and managed to

render in only 80-85% of the time that it took NeRF. Interestingly, Figure 5 also shows that the time gain associated with sparser samplings quickly drops off. This is likely due to the fact that sample sparsity must increase quadratically, which means that most of the gains occur during the first few increases in sparsity (e.g. going from sampling $\frac{1}{1}$ points to $\frac{1}{4}$ points saves more time than going from $\frac{1}{36}$ to $\frac{1}{49}$ points).

4.2. Point Cloud Sampling

As shown in Figure 8, rendering with NeRF using point clouds achieves an output image from a generated pose virtually indistinguishable from the output of the original NeRF model. However, we found that using the point cloud made the model slower, so we chose to use smaller versions of the full point cloud, produced by rendering fewer rays for each of the training images. After selecting our point

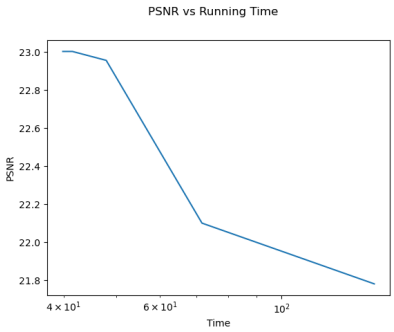

Figure 6. A comparison of the running time and PSNRs of renders using different sized point clouds. The two smallest clouds were both the fastest and the most accurate so we choose to use the second smallest cloud (downsampled x16).

cloud, we compared our rendering performance with it to our rendering performance using standard NeRF. When using point cloud sampling, we found that the model achieved similar PSNR to NeRF, but had a constant increase in time, which we attributed to the linear time processing of the point cloud. Our expectation was that as the number of samples taken along a ray decreased, the PSNR for our rendering method would remain higher, as it could better target the surfaces in the scene. However, the PSNR closely followed that of the NeRF model. One explanation for this is that since we chose such a sparse point cloud and took measures to loosen the bounds, we effectively sampled the original (0,1) range, but just added a point cloud pre-processing step that slowed down the network.

5. Discussion

Interpolated rendering proved to be a useful technique for rendering NeRF, but it was limited in the performance benefit that it provided. The render time was reduced by

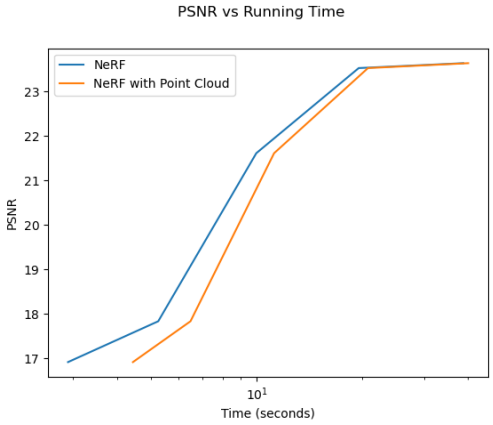

Figure 7. A comparison of the running time and PSNR of NeRF and our Point Cloud Sampling Nerf. Each vertex represents a different number of samples per ray, up to the standard number (64,128)

Figure 8. A side-by-side comparison of the output of rendering NeRF without (left) and with (right) point cloud sampling with 32 samples per ray

about 15%, and it is unlikely that similar improvements could be achieved in the same manner since the first pass of sampling only comprises 25% of the total sampling used to render an image.

Our efforts to use knowledge of the scene geometry to inform the sampling did not lead to any performance gains. This can be attributed to the speed and performance tradeoff that one makes when choosing the size of a point cloud. With too small a point cloud, we can't bound the depth at every pixel in the new view; and with too large a point cloud we spend time iterating over many points. It is likely that some other method could be used to estimate the depth of the scene with greater effect.

Our implementation of depth estimation using a point cloud could be further explored and potentially improved. Improving the propagation of values across pixels during the local max/min phase could lead to good bounds even with smaller point clouds. While our interpolated rendering provided better improvements, depth estimation is a more promising area for future work.

References

[1] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building rome in a day. In: ICCV (2009). 1

[2] David B. Lindell, Julien N. P. Martel, and Gordon Wetzstein. Autoint: Automatic integration for fast neural volume rendering. Under Review: (2020). 2

[3] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. Under Review: (2020). 2

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

[5] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. In: CVPR (1997). 1

[6] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. In: ACM Trans. Graph. (2018). 1