Radiance Fields Out of Stained Glass Pieces

Anonymous CVPR submission

Paper ID *****

Abstract

Radiance fields have achieved incredible visual results, but the implicit nature causes both rendering and learning time to be impractical for many purposes. Our work suggests a compromise between discrete representations which are convenient to render, and implicit representations, which generally enjoy better training results: instead of an implicit function which takes in 5D input based on position and angle, we use differentiable rendering on a triangle soup. This allows us to use classical rendering techniques which are fast and highly parallelizable while maintaining a promising degree of accuracy. Each triangle of the triangle soup has a learnt opacity and color attribute, hence we name it the stained glass model. Furthermore, propose a method of iteratively resampling the vertices of the triangle soup to improve the accuracy of the model.

1. Introduction

Novel view synthesis is an old task in computer vision, but it has never been so exciting. With the emergence of neural radiance fields [3], the possibilities seem endless. Neural radiance fields (NeRF) are implicit representations of a 3D object learnt as a neural network. Particularly, the neural network inputs (x, y, z, ϕ, θ) denoting the position and viewing angle of a point, and outputs (c, σ) denoting the color and opacity at that point. This implicit function is rendered into an image by raycasting and taking samples along each ray to integrate using the transmittance formula [2]. However, the original NeRF suffered from the same issue as other 3D neural implicit representations: the rendering process is too slow for real-time rendering. A high resolution image requires casting on the order of 10^6 rays, and each ray is sampled some number of times. Then each sample must be forward-propagated through the neural network. With the help of vectorization, all these computations can be done in several seconds, but this is not good enough to please the human eye, which prefers at least 60FPS. This slow rendering also results in very long training time, since rendering is a piece of the training loop. Due to this problem, a number of works have sought out to reduce the computation time [1, 4–7], primarily by reducing or factoring the neural elements of NeRF. Our work continues this trend by learning a 5D representation that doesn't involve neural networks. The most similar work is [8], which we discuss further in the next section. Like it, we prefer a discrete structure rather than a fully implicit structure. However, we choose to use a triangle soup rather than a sparse voxel grid. Here, we summarize our work's results:

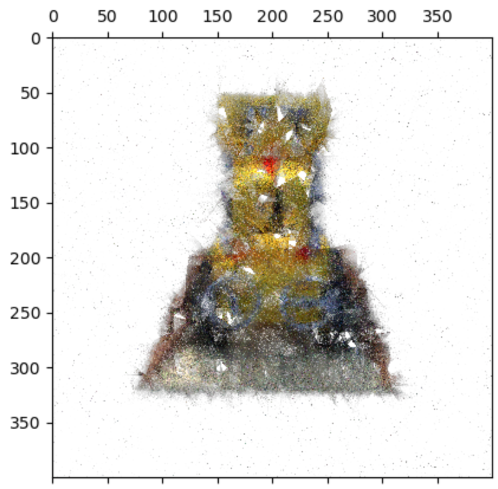

Figure 1. A teaser for the results. Our method does not come close to the state-of-the-art, but the results are somewhat promising.

- We introduce our novel stained glass model, which combines benefits of discrete structures and implicit training methods;
- We use classical rendering techniques to form a differentiable rendering pipeline;
- We give an efficient method of training for the task of novel view synthesis.

2. Related Work

Though our work does not involve any neural structures, it would be remiss not to begin by discussing neural radiance fields (NeRF) [3]. NeRF represents 3D objects as an implicit function which takes in the position and viewing direction, and outputs the density at that point and the color at that point and with that viewing direction. In order to train the network, known views were rendered from known poses. To render each image, a large number of rays were cast and samples were taken along each ray to query into the network. Thus, though this work gave excellent results in novel view synthesis, but as mentioned, it came with slow rendering time. To avoid this slow runtime, our work follows [8], which removes the neural network completely, replacing it with a voxel grid of functions in the spherical harmonic basis. The rest of their approach remained largely the same as NeRFs, with rendering being casting rays and taking samples, but removing the neural network allowed them to achieve orders of magnitude better runtime. Their conclusion was that the radiance field was the primary benefit of NeRF and that the neural representation was secondary. Our paper aims to modify the learning process to improve runtime.

3. Method

Notation. The dataset for radiance field training contains known camera poses and relevant parameters as the input. Particularly, we have K 4×4 transformation matrices $P_1, P_2, \dots, P_m$ representing our camera poses as well as the image dimensions H, W and focal length f. The outputs are the $H \times W$ images $I_1, I_2, \dots, I_m$ seen by the camera at each respective pose. We can instead express our inputs and outputs as the $K \times H \times W$ triplets $(o, r, \mathcal{C})$ where $o + tr$ is a ray from the camera and $\mathcal{C}$ is the color of the resulting pixel as RGBA. Our structure, which we denote as F, attempts to solve the problem $F(o, r) = \mathcal{C}$. Note that novel view synthesis is a straightforward application of such a model.

Stained glass model. Our model is parameterized by a triangle soup with m triangles over n points with color and opacity attributes. Let us denote our points as $p_1, p_2, \dots, p_n \in \mathbb{R}^3$. Then the m triangles take their vertices as triplets of these points which are denoted as $(a_1, b_1, c_1), \dots, (a_m, b_m, c_m) \in [n]^3$. Each point p_i has some color s_i and opacity σ_i and the triangles interpolate these attributes following barycentric weighting. When rendered with random initializations, the model appears like pieces of stained glass strewn across space (Fig. 2). The color attributes s_i are implicitly dependent on the view direction as they are parameterized with spherical harmonics coefficients.

To initialize the model, the points are chosen uniformly

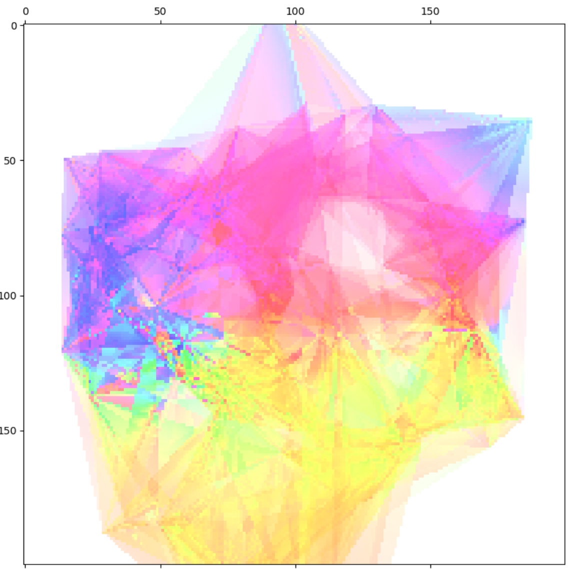

Figure 2. An example of the model's random initialization. Unlike standard rasterization, multiple triangles influence the color of each pixel.

at random from a bounding box which the object is given to be in. The initial color, parameterized by spherical coefficients, is random while the initial opacity is chosen to be a low value of 0.01. The triangles are chosen by computing the k-nearest neighbours of the vertices for $k = 10$ and forming triangles randomly within these. This helps ensure that there are no gaps, but it does mean that are no guarantees on the connectivity or topology of the triangle soup and these are poor in practice. However, for the purpose of rendering, these are not necessary.

Rendering pipeline. To render a ray (o, r), we first use the MT algorithm to compute all m ray-triangle intersections $o + t_i r$ as well as the barycentric coordinates u_i, v_i, w_i. We discard any triangles that are not actually intersected by the ray, then sort the remaining ones by t_i. For the ith triangle that remains and assuming that it is sorted, we compute its opacity σ_j^* and color s_i^* at the point of intersection:

$$s_i^* = u_i s_a + v_i s_b + w_i s_c,$$
$$\sigma_i^* = u_i \sigma_a + v_i \sigma_b + w_i \sigma_c.$$

Then using the single-scattering radiance formula, we compute

$$T_i = \exp\left(-\sum_{i=1}^{i-1} \sigma_i^*\right),$$
$$F(o, r).rgb = \sum_{i=1}^{m} T_i (1 - e^{-\sigma_i^*}) s_i^*,$$
$$F(o, r).a = 1 - T_m.$$

Again, the view direction, represented as spherical coordinates (θ, ϕ) is implicit in the above equations. The spherical harmonic coefficients are linear so we only need to evaluate the spherical harmonics with these angles after summing the coefficients over the triangles.

Training. To train, we use gradient descent to optimize over the opacity and color of each point. We choose to use gradient descent as the output is non-linear in terms of σ (though it is an overconstrained linear system in terms of the color attributes s). Also, backpropagation is relatively fast for this setup. Though the rendering pipeline is involved, backpropagating is shallow as the opacity and color are only present in the final step of the rendering pipeline. In particular, the t, u, v, w values computed during rendering are not dependent on these learnt parameters.

During the training routine, we batch pixels from our images I to cast rays from and render. Then our loss is the mean squared error between the rendered RGBA values and the ground truth RGBA values.

After each epoch during training, the points are resampled so that the new points are biased to be sampled close to the old points with high opacity. This helps learn surface details better. In particular, we choose $0.5n$ points with replacement from the old points and perturb them slightly with Gaussian noise. Then we append $0.5n$ completely new points sampled uniformly at random from the original bounding box.

4. Experiments

Setup. The experiments are solely performed on the toy Lego model dataset [3]. This is a synthetic model with 400 training, validation, and test views, each giving an 800×800 image with RGBA values between 0 and 1. The experiments were run on a NVIDIA Geforce RTX 3050 GPU. Due to memory limitations, we downsampled all images to 100×100, and used a batch size of 200. In other words, there were $100 \times 100 \times 100$ pixels in the training set and each batch rendered from 200, giving 5000 iterations per epoch. We use an Adam optimizer with a learning rate of 10^{-3}.

For our stained glass model, we choose $n = 5000$ points and generate $m = 50000$ triangles from these vertices as described above. The spherical harmonic coefficients are up to order two. The Lego dataset has relatively simple lighting conditions so it is unlikely that this choice of hyperparameters would have a large effect.

Results. Our results generally had poor visual quality. This is in part due to technical limitations as we had to downsample the images to 100×100. It is also unfortunate that we could only use $n = 5000$ points due to memory limits. For comparison, [7] used $256^3 \approx 1.7 \times 10^7$ voxels. It is likely that this directly limited the model's ability to represent the 3D scene. Even with these reduced parameters,

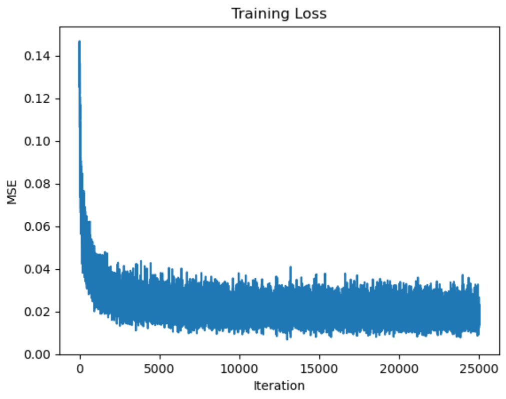

Figure 3. Training loss for training loop without resampling. As can be seen, it plateaus quickly, but fails to reach a good MSE.

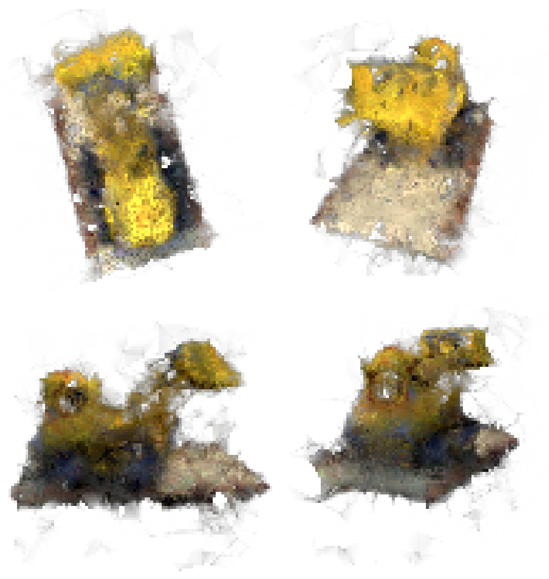

Figure 4. Results on the validation set after 5 epochs. No resampling was used for these results.

each epoch still took approximately 175 seconds. Hence, it takes 1.75 seconds on average to render a single 100×100 image of a view. However, in the following ablation study, we find that some of our techniques helped improve the visual quality.

Ablation study. There are artifacts in the results without using the resampling trick, as shown in Fig. 4, Fig. 5. We can see that there are ghost triangles where there should be empty space. However, with the exact same setup, but now resampling after every epoch, we remove these artifacts and find overall cleaner results as shown in Fig. 6, Fig. 7.

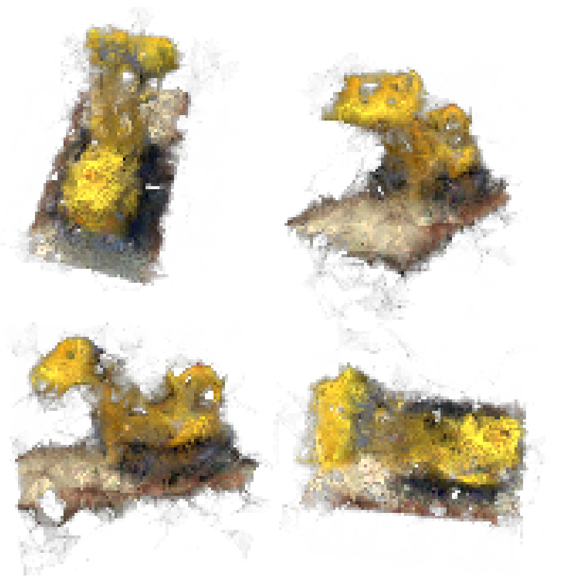

Figure 5. Results on the training set after 5 epochs. No resampling was used for these results.

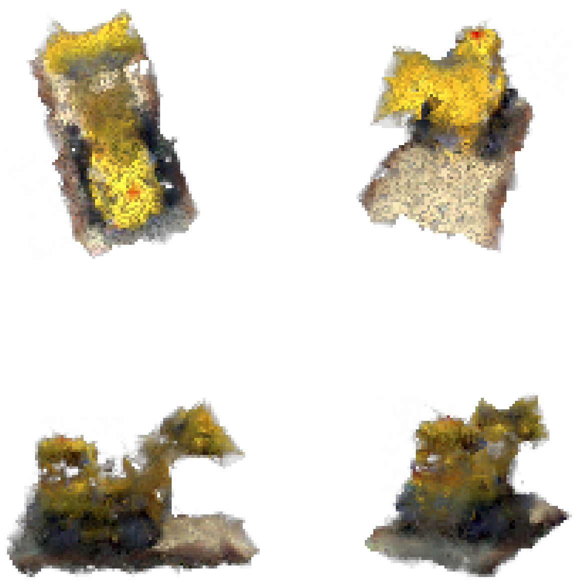

Figure 6. Results on the validation set after 5 epochs. Resampling was used for these results, and fewer artifacts can be seen.

5. Conclusion

Our work proposes a novel radiance field structure based on a triangle soup. While we do not obtain state-of-the-art results, we hope that our progress can inspire future directions to improve this work. In particular, we believe that by building on more modern pipelines for rendering triangles, we can accelerate our differentiable rendering pipeline to be competitive.

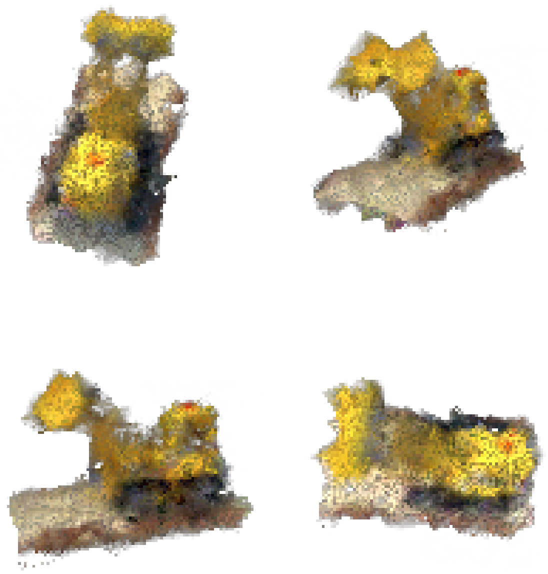

Figure 7. Results on the training set after 5 epochs. Resampling was used for these results, and fewer artifacts can be seen.

References

[1] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355, 2021. 1

[2] James T Kajiya and Brian P Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH computer graphics*, 18(3):165–174, 1984. 1

[3] 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*, pages 405–421. Springer, 2020. 1, 2, 3

[4] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H Mueller, Chakravarty R Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. Donerf: Towards real-time rendering of compact neural radiance fields using depth oracle networks. In *Computer Graphics Forum*, volume 40, pages 45–59. Wiley Online Library, 2021. 1

[5] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021. 1

[6] Vincent Sitzmann, Semon Rezchikov, Bill Freeman, Josh Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. *Advances in Neural Information Processing Systems*, 34, 2021. 1

[7] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenox-

els: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021. 1, 3

[8] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021. 1, 2