InstantNerfW for hand-held object reconstruction Machine Learning for Computer Vision Project Work

Samuele Bortolato and Riccardo Paolini
Master's Degree in Artificial Intelligence, University of Bologna
samuele.bortolato@studio.unibo.it, riccardo.paolini5@studio.unibo.it

Abstract

In this work we fuse NeRF in the Wild with InstantNeRF to speed up the training and rendering of NeRFs with occlusions, proposing a new formulation for the transients to further speed up the computation. We then applied the method to the reconstruction of hand-held objects rotated in front of the camera, analyzing the additional artifacts and proposing some possible solutions.

1 Introduction

The goal of this project is to train the NeRF (Mildenhall et al., 2020) of objects moved in front of the camera, without hands or background, in a light enough manner that it can be run on a less powerful machine, such as the laptops we have available for the project. The task presents multiple problems:

- The training and rendering of a NeRF with the original implementation is both too computationally heavy and too slow to be effectively trained on a laptop;
- The reconstruction of an hand-held objects requires the removal of the hands, that are not static and coherent with the scene we want to reconstruct;
- The presence of the background that is not coherent with the object coordinate system makes it impossible to correctly reconstruct it, but at the same time remaining nearly unchanged in all the training views it tends to still match creating a colored fog.

Following what has been done in NeRF in Wild (Martin-Brualla et al., 2020), we modeled occlusions as view-dependent transient properties to be trained together with the underlying static scene. To speed up training and rendering, and make it light enough to be trained on a laptop, we modified the original work by fusing it with what was done in Instant Neural Graphics Primitives with a Multiresolution Hash Encoding (Müller et al., 2022). We have also modified the original transient formulation from spatial integration to a 2D modulation of color loss, allowing even faster training and inference as well as enabling efficient pruning of the acceleration occupancy grid. Finally, to handle the non-matching static background, we introduced a trainable background color for the ray integration and proposed two regularization losses on the density of the static scene. We implemented the work using the Kaolin-Wisp library (Takikawa et al., 2022)

2 Background

2.1 NeRF

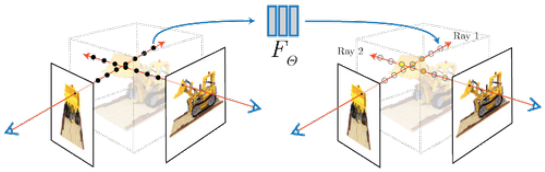
Neural Radiance Fields (NeRF) (Mildenhall et al., 2020) is a novel framework to represent a 3D scene with a neural implicit function, where a neural network $F_\theta(\mathbf{x}, \mathbf{d})$ maps a 3D point $\mathbf{x} = (x, y, z)$ and a unit viewing direction $\mathbf{d} = (\theta, \phi)$ to a volume density σ and RGB color $\mathbf{c}_i$, which gets integrated through a ray following classical volumetric rendering to determine the color of the corresponding pixel.

$$\hat{\mathbf{C}}(\mathbf{r}) = \sum_{k=1}^{K} T(t_k)\alpha(\sigma(t_k)\delta_k)\mathbf{c}(\mathbf{t_k})$$

where $T(t_k) = exp\left(-\sum_{k'=1}^{k-1}\sigma(t_{k'})\delta_{k'}\right)$, $\delta_k = t_{k+1} - t_k$ is the difference from two sample points along a ray, and $\alpha(x) = 1 - exp(-x)$ computes the opacity of a point based on the density.

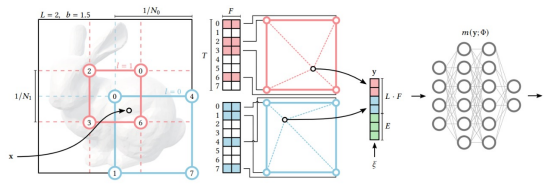
The NeRF is trained by minimizing the MSE loss between the rendered pixel color and the ground truth of the training images.

$$L_i(r) = ||\mathbf{C}_i(r) - \hat{\mathbf{C}}_i(r)||_2^2$$



2.2 Instant NGP

To speed up the computation Instant Neural Graphics Primitives with a Multiresolution Hash Encoding (Müller et al., 2022) proposes to introduce a trainable input encoding that allows the use of much smaller networks. The trainable encoding is organized in levels, at each level there is a grid over the input space with different resolutions. At each level the input is encoded by interpolating the features of the corners of the corresponding voxel of the grid. Instead of storing a dense grid that would waste a lot of memory by storing features in locations where are not needed, the features are stored in a much smaller hash table. The table doesn't explicitly handle hash collisions but instead let's the model learn features that keep into account the collisions, which works well enough thanks to the grids having different resolutions at each level. Finally to speed up the computation even more they introduced an auxiliary acceleration structure that allows to sample only important regions by keeping track of the occupancy of the voxels on the space.



2.3 NeRFW

NeRF in the Wild (Martin-Brualla et al., 2020) introduces a framework to learn a NeRF from unstructured collections of photos, which poses two main problems:

- variations of radiance of the objects due to both different time and weather conditions and different cameras;
- presence of transient objects (e.g. tourists) that occlude the underlying static scene.

They assigned to each image an appearance embedding to account for photometric variations, conditioning the prediction of the color on a latent embedding space. To deal with the transient occlusions they introduced a second NeRF conditioned on another set of per-image transient embeddings, which predicts transient colors $\mathbf{c}_i^{(\tau)}$ and transient densities $\sigma_i^{(\tau)}$. The color of the ray during training is then computed integrating both the static and transient colors.

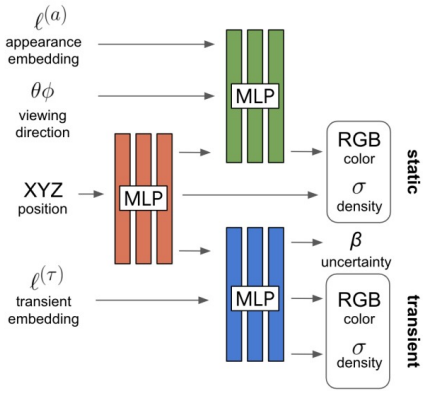
$$\hat{\mathbf{C}}_i(\mathbf{r})=\sum_{k=1}^{K} T_i(t_k)\Big(\alpha(\sigma(t_k)\delta_k)\mathbf{c}(t_k)+\alpha(\sigma_i^{(\tau)}(t_k)\delta_k)\mathbf{c}_i^{(\tau)}(t_k)\Big)$$

where $T_i(t_k)=exp\Big(-\sum_{k'=1}^{k-1}\Big(\sigma(t_{k'})+\sigma_i^{(\tau)}(t_{k'})\Big)\delta_{k'}\Big)$

Finally, they modeled the color of each pixel as a random variable with isotropic normal distribution, making the transient part of the model also predict the variance β_i. The loss for the ray r is

$$L_i(\mathbf{r})=\frac{||\mathbf{C}_i(r)-\hat{\mathbf{C}}_i(r)||_2^2}{2\beta_i(\mathbf{r})^2}+\frac{log\beta_i(\mathbf{r})^2}{2}+\frac{\lambda_u}{K}\sum_{k=1}^{K}\sigma_i^{(\tau)}(t_k)$$

where the first two terms are for the negative log likelihood of $\hat{\mathbf{C}}_i(r)$ of the normal distribution and the third term is an L_1 regularizer with multiplier λ_u to discourage the model from using the transient also from static phenomena.



3 System description

Our proposed architecture combines the features of NeRFW and InstantNGP, in fact we want to reconstruct scenes with occlusions while significantly reducing the training time.

The structure of the network is similar to the one used in NeRFW, we use three Multilayer Perceptrons (MLPs), two of which encode the "static" part of the radiance field, while the third network is used to encode the "transient". Following the improvements proposed by InstantNGP, we applied

multiresolution hash encoding that improves the approximation quality and training speed. In particular, we used two different hash grids to encode the inputs for the "static" and the "transient" networks.

3.1 Reformulation of the transient

The simple implementation of the hash grids, while speeding up training, didn't allow the use of the acceleration structure employed in instantNGP, because pruning the different occupancy grids (one for each sampled image) required sampling too many points, actually slowing down training.

The thing that allowed us to use the acceleration structure, achieving a substantial speed-up, was the reformulation of the transient from a 3D space to a 2D variation of the image.

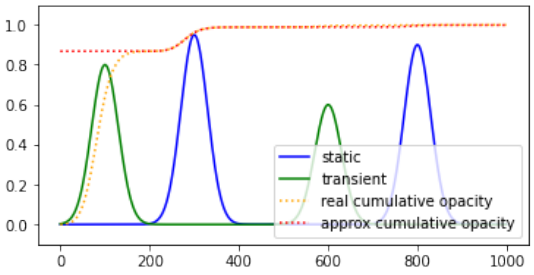


Figure 1: Complete integration vs approximaton

Since each transient space is sampled only from the point of view of the corresponding training image, instead of computing the whole spatial integration over the ray with both the static and transient scene, we integrated only the static scene and handled the transient as a variation over the ray color. Obtaining the final color then becomes

$$\hat{\mathbf{C}}_i(\mathbf{r}) = \mathbf{c}_i^{(\tau)} \alpha(\sigma_i^{(\tau)}(r)) + \hat{\mathbf{C}}(\mathbf{r})(1 - \alpha(\sigma_i^{(\tau)}(r)))$$

Matematically this is equivalent to put only an impulsive transient at the origin of the ray. Fig.1 show the original and approximated cumulative opacity $(1 - T_i(t_k))$, while Fig.2 stretches the axis to show the difference.

It's worth to notice that this approximation is completely equal to the original if the objects in the static scene are completely opaque. In fact, in this case the transient behind the first object is exactly zero and it's equal to having only the transient in front of the objects, which is then a constant for all the points belonging to the object. In contrast, if the objects are not completely opaque, we can still approximate the transient as a color change over

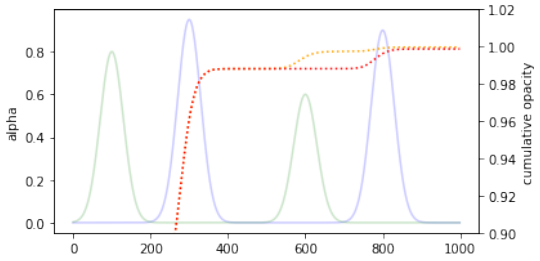


Figure 2: Closeup of the difference

the static ray, but the color and opacity is slightly different.

This allows us to compute the transient color and density only once per ray and, removing completely the transient spaces, we can store and prune an acceleration occupancy structure only for the static part, making it feasible to use accelerated ray marching.

The additional *transient loss* that discourages the model from using transient opacity to explain away static phenomena becomes

$$L_{tran}(r) = -\log(1 - \alpha_\tau(r))$$

By trying to minimize the error on ray color while minimizing the use of the transient, the model pushes the transient color to the edges of the availabe range, to get the same variation using a lower density. This is not what we wanted, so we chose to fix the transient color to the corresponding pixel color. With this modification the loss can be rewritten as a weighting of the color loss function based on the transient opacity.

$$L_{rgb}(r) = (1 - \alpha_\tau(r)) \cdot ||\hat{\mathbf{C}}(r) - \mathbf{C}(r)||_2^2$$

where $\hat{\mathbf{C}}(r)$ and $\mathbf{C}(r)$ are respectively the predicted and the ground truth color of the ray r, and $\alpha_\tau(r)$ is the predicted opacity of the transient.

In short we made the model just predict the opacity of the transient of the ray and used it to re-weight the color loss. In our tests this reformulation didn't lead to any loss of performance, while achieving a significant speedup, both because it makes the integration lighter and because we can now use occupancy structures for accelerated ray marching as in instant-NeRF.

To predict the transient density we wanted to use a two-dimensional hash grid, but since the Kaolin-Wisp library we used does not yet support it, we again used three-dimensional ones by sampling them on the surface of the unit sphere. Instead of using a different hash grid for each input feature we

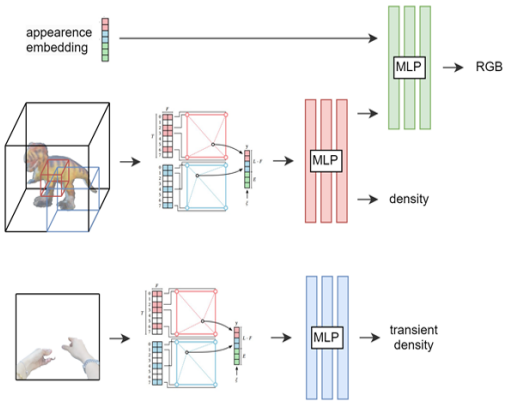


Figure 3: Complete architecture

used the same one for all by offsetting the sampled surface based on the index. This lets the model choose where to allocate resources for representing the transient. Still, using a bi-dimensional hash grid with shared table would be more appropriate and could make better use of memory. It should be updated with the new library versions.

Finally, it should be noted that Nvidia's Kaolin-Wisp library is an experimental-oriented library, is modular and made mostly in python to simplify the modification process for testing new ideas. The downside of this approach is that it's implementation is way less efficient than the instant-NeRF implementation that was written in plain Cuda. In particular, because we were running the training on a laptop with only 4 GB of RAM, it was not possible to place the entire dataset in GPU memory, and the training time due to all these inefficiencies was on the order of tens of minutes instead of minutes. Without the loss reformulation, however, training could easily require a couple of hours.

3.2 Trainable background and auxiliary losses

Both the original NeRFW and our modified version are still unable to reconstruct handheld objects without artifacts. The incoherent background cannot be reconstructed since it's not static with respect to the reference system of the object. On the other hand, since we are rotating the object in front of the camera, the background remains nearly completely unchanged leading to a strong matching of the colors. The result is that the NeRF tends to create a diffused fog of the same color of the background all around the object we are trying to reconstruct.

To reduce this effect, we introduce a trainable background color for the ray, so we can try to push the uniform part of the background into the trainable background of the rays, while the non-uniform part is handled quite well by the transient. Additionally, we introduced two regularization losses to further push the model in this direction.

Entropy loss: This term forces the network to predict density values that are either very low or very high. This is consistent with our perception of free-space and non-transparent objects. It works by pushing density values towards 0 or 1.

In particular, it is calculated on a per-sample basis so that it is more effective in its intent of removing the 'fog' since it is not affected by the presence of the object which, in a 'per-ray' calculation, would add the missing share of density to reach high values.

$$L_{entr}(r) = -\sum_{k=1}^{K} \alpha(t_k) \cdot \log(\alpha(t_k))$$

with

$$\alpha(t_k) = 1 - \exp(-\sigma_s(t_k))$$

where K is the number of samples taken along the ray r, and $\sigma_s(t_k)$ represents the density predicted by the 'static' head at point t_k.

Background loss: To compute this term we made it possible for the network to learn the background color of the images. By doing so, it becomes possible to penalize the 'static' opacity of the ray if the color of the pixel from which we cast the ray is similar to the adaptive background color.

$$L_{back}(r) = \alpha(r) \cdot \exp(-\beta_{sel} \cdot ||\mathbf{C}_{bg}(r) - \mathbf{C}(r)||_2^2).detach()$$

where $\alpha(r)$ is the opacity for the 'static' part of the ray r, β_{sel} is a hyper-parameter which varies the selectivity for the background, $\mathbf{C}_{bg}(r)$ and $\mathbf{C}(r)$ are the color of background and the ground truth color, respectively.

Total loss: With the changes that have been made, the new loss function will be a linear combination of the losses described previously, as shown below:

$$L(r) = L_{rgb}(r) + \lambda_t \cdot L_{tran}(r) + \lambda_e \cdot L_{entr}(r) + \lambda_b \cdot L_{back}(r)$$

where λ_t, λ_e and λ_b are respectively the transient, entropy and background loss multipliers.

4 Tests and results

To check whether the transient reformulation works correctly, the best thing would be to use the same

dataset used by NeRF in the Wild and try to reproduce their results. The problem is that they used a huge dataset of nearly 800 images, and we have absolutely no way to perform the training even with the optimized algorithm. We tried to run it on a much smaller subset, but the low number of rays we are able to cast with our machines makes the convergence rather noisy, and we could not get usable results in a reasonable time.

Since training a model with the original NeRFW was completely unfeasible, the only option we had was to test our initial implementation (which simply added the hash encoding to the NeRFW formulation) and compare it with our final version. For the same computational reason, we chose not to use the Photo Tourism dataset.. We chose instead to modify the already existing LEGO dataset for which we already had the camera positions and we didn't have to use colmap. We added random colored patches to the input images to try to simulate occlusions.

Figure 4: Lego input images

The problem with this first version of the dataset is that since there are no intersections between rays projected by cameras with similar viewpoints, the model learns to place these occlusions just in front of the cameras. In fact, it can simply assign to one side the color of the input image and to the other the color of the background. By doing so, cameras pointing at these points from the opposite viewpoint are unable to distinguish them from the background, making it impossible to remove the artifacts. One possible solution would be to remove the dependency from the viewing direction, but that would also remove specular effects of the model.

Then we tried a second version of the dataset, in which each input image is presented twice with two different occlusions, hoping to solve the problem in this way. But again all the models we tested (instant-NeRF, NeRFW and our own) produced the same kind of artifacts, as if the transient part was almost completely ignored. When using the transient the artifacts were slightly less opaque, but still present. We double-checked the code and

Figure 5: Lego results

doesn't seem to be an error. Our best guess was that those formulations of the losses where not enough to remove occlusion from a small set of overlapping images. Results with NeRFW are shown in fig.5, where the left image shows the whole space, while the right one renders only the central part.

For all these reasons we were not completely able to prove that our reformulation of the transient is indeed equivalent to the original NeRFW and further investigation should be carried on with more appropriate hardware. On our tests the two models produced really similar results, so we used mainly the new one for the speed.

Figure 6: Greendino input images

The second dataset is taken directly from the task we wanted to solve: reconstructing an hand-held object rotated in front of the camera. To create the dataset, we used images captured with Rayban Stories. The object we tested was a green dinosaur, so we named the dataset "greendino". We computed the camera positions by matching the features of the object using Colmap. To match only the features of the object, we used a simple mask obtained from the approximated depth map that was available. The correct estimation of the cameras was particularly difficult to compute: we could either match a lot of cameras with a significant cumulative error or settle with a lower number of well aligned views. Only well-aligned cameras were used in our experiments, although we were only able to correctly estimate camera poses for one side of the object we were trying to reconstruct. For better results, additional attention should be paid to slowly rotate the object to ensure that the interme-

diate images are not blurred, so as to simplify the camera estimation process as much as possible.

As mentioned before, reconstructing the object without explicitly handling the background causes the model to make a diffused fog around the object (Figure 7).

Completely masking the background would allow to solve the issue, but that would require a pretty precise mask. Using an approximate mask causes the model to create a small but noticeable aura around the object (the small section of fog corresponding to the part of the background that was not masked correctly) (Figure 8). In addition, if occlusions are not also masked, the model seems to have a harder time distinguishing them from the object.

As explained previously, we opted instead to use a trainable background color with additional regularizers and training directly from the original images. Fine-tuning the regularization losses is particularly delicate, so we introduced sliders in the graphical interface to make the process easier by tuning them while the model is training, and rendering the results in real time. The results we were able to obtain are showed in (Figure 9).

Figure 7: Base model

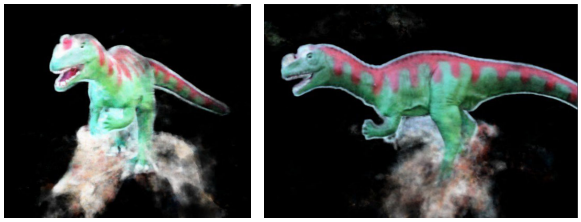


Figure 8: Masked input

Figure 9: Trainable background

Finally we tested the model on the yellowdino dataset (the only other dataset for which we were able to compute decent camera poses to train the model). As we can see from Figure 11 even in this case Colmap was not able to accurately match the images, in fact we can notice a doubling of the object along the edges. On the other hand, since the color of the object is quite different from the background color, we were able to completely remove the background by increasing the empty loss. Unlike greendino, here we do not have artifacts due to the hands, as the training images have more variation on how the object was held.

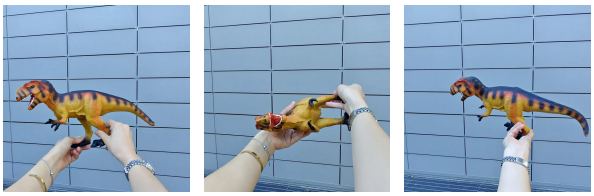


Figure 10: Yellowdino input images

Figure 11: Yellowdino results

We implement all experiments in Pytorch using the Nvidia's Kaolin-Wisp library. For each scene, we extracted the camera poses and intrinsic parameters using the Colmap package. Then we train a model initialized with random weights. We optimize the network for 1000 epochs with a batch size of 4 on a single GPU (3050ti laptop) using Adam optimizer (with hyper-parameters β_1=0.9, β_2=0.99 and ϵ=10^{-15}).

5 Discussion and future work

In this work we experimented using just images as input to the model, and for this reason we formulated all the losses based solely on pixel colors. This approach has limitations, especially when the object we are trying to reconstruct has colors similar to the background, forcing us to decrease the regularization and potentially producing more artifacts (as in greendino). The only real solution to the problem would be to use additional information other than pixel colors, such as separating the mask completely from the image, to implement it within the regularizers instead of in the target. Moreover, this would require a less precise mask, or even a

soft mask.

Another even more general solution, which does not require the creation of a mask, would be to use depth information to better distinguish occlusion and background from the object (and speed up training).

All these ideas are left for follow-up works.

6 Conclusion

We have presented our method for combining features of NeRF-W and InstantNGP, these allowed to reduce training time from 1-2 days to 30-60 min without worsening the quality of the results, reformulating the transient framework to decrease the computational cost. We then showed that NeRFW alone cannot handle the inconsistent backgorund properly and proposed to solve it with a trainable background color and some regularizers. This is only a first attempt to address the problem, and many adjustments can be made to improve the results. However, we emphasize that the most crucial part for improving the results is solving the problem of retrieving camera positions reliably, and particular care should be taken during image acquisition to facilitate Colmap in matching the scene, especially when changing side of the object.

7 Links to external resources

Github repository: https://github.com/samuele-bortolato/instantNeRFW

References

Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. 2020. Nerf in the wild: Neural radiance fields for unconstrained photo collections. *CoRR*, abs/2008.02268.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. *CoRR*, abs/2003.08934.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15.

Towaki Takikawa, Or Perel, Clement Fuji Tsang, Charles Loop, Joey Litalien, Jonathan Tremblay, Sanja Fidler, and Maria Shugrina. 2022. Kaolin wisp: A pytorch library and engine for neural fields research. https://github.com/NVIDIAGameWorks/kaolin-wisp.