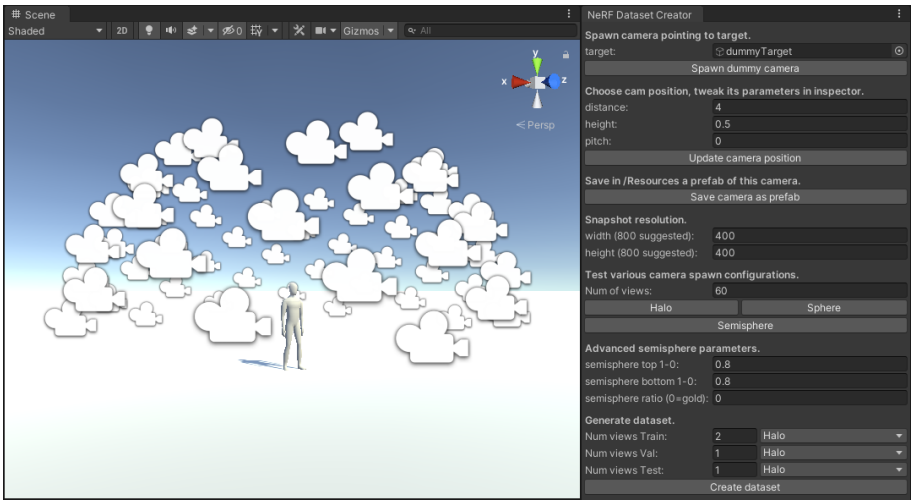
Unity Plugin for Creating Synthetic NeRF Datasets

Computer Graphics Project

Andrea Masciulli

University of Trento, Italy
andrea.masciulli@studenti.unitn.it

Abstract. A Unity plugin for easy creation of synthetic datasets for NeRF is presented. The plugin spawns cameras around a target, capturing images from them. A simple GUI is provided to help the user choose: which game object to target; how many cameras are spawned and in what configuration; how many images are wanted for training testing and validation. The output file contains the captured images, the ground-truth position and orientation of such cameras, depth and normal maps for final NeRF evaluation purposes. The output dataset file is identical to the Blender data-sets used in the original NeRF paper.



https://github.com/AndreaMas/nerf-dataset-creator-plugin

1 Introduction

View-Synthesis is the problem of synthesising novel views of a scene given a limited number of existing views. To this day, this is a challenging problem to tackle, and one which can be solved in many interesting ways. State-of-the-art results for view-synthesis can be achieved by representing scenes as Neural Radiance Fields [1]. This novel approach represents a static scene as a continuous 5D function using a fully-connected deep neural network. The function takes as input a single continuous 5D coordinate, which encodes spatial location (x, y, z) and viewing direction (θ, ϕ). Its output is the volume density and view-dependent emitted radiance at that spatial location. NeRF gained a lot of attention and many papers have been published recently which build upon it [2].

NeRF takes as input images and their intrinsic and extrinsic parameters. In real world scenarios we often don't know the intrinsics of a camera, and it is even less likely to know relative camera positions. Thankfully, this is a non-problem, since Structure-from-Motion (SfM) algorithms are able to estimate these camera parameters reliably. Do note that this is not always the case, especially if the camera taking the images can be approximated to a pinhole camera model (this usually isn't the case), if images are few, if camera moves too much from one image to another or if not enough high-level detail is present of the target (e.g. plain color wall, clear sky). However, if working in a simulated environment, there is no need for such estimation, given that we know ground-truth camera positions and intrinsics. This enables us to better understand the potential of NeRF, avoiding problems that could be given by an erroneous camera position estimation for example.

In the following, a plugin implemented in Unity for easy and fast creation of custom synthetic datasets is presented. First, a general understanding of NeRF's dataset structure is given. Then, the general framework followed by the plugin to create the dataset is presented. Finally, insights over the use of such plugin and final conclusions are made.

2 NeRF's Dataset Structure

When NeRF is about to be trained, a number of configurations parameters need to be set to best fit the training to the scenario at hand. One of the configurations parameters is called "dataset-type" and it's the parameter responsible for telling NeRF whether we are using a dataset that has been generated by Blender (i.e. synthetically) or by using real-world images, estimating intrinsics and extrinsic trough Structure-from-Motion. Of course, the user is free to implement its own dataset structure. However, for better ease of use, the dataset file structure generated by the presented Unity plugin is identical to the one the Blender data-sets uses in the original NeRF paper. Therefore, to use the data generated, it's sufficient to set the "dataset-type" parameter to "blender".

NeRF stores camera parameters in a json file [Fig. 2], which contains the number of views around the target acquired, the field-of-view of the camera, and

for each image the camera position in the virtual space. The camera position is stored as a camera-to-world matrix. This matrix embeds within itself the information to roto-translate the camera from a default camera position to its actual position in the virtual world. In computer graphics this default camera position and orientation may vary. The standard followed by OpenGL, Unity and other is to have as the default camera position a camera placed in the origin, facing toward the negative z axis, with the positive x axis at its left and the positive y axis on top.

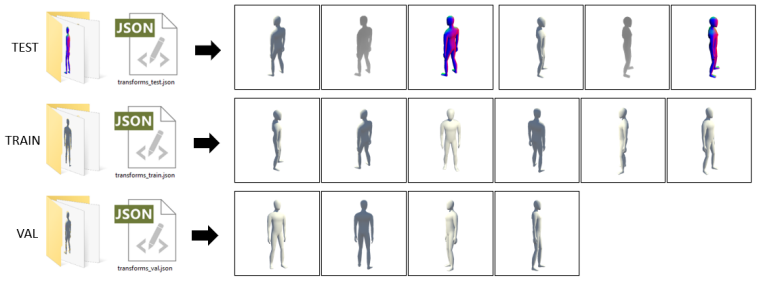


Fig. (1) Dataset structure produced by the plugin. In this particular case, the training set has six views, whereas the validation set has four. The test set is made up by two views of the target, depth and normals are also acquired. Finally, three json files for each set store the extrinsics for each view.


```
{
  "camera_angle_x": 0.6981317,
  "frames": [
    {
      "file_path": "./train/r_0",
      "rotation": 10.16641,
      "transform_matrix": [
        [0.0, -0.7, 0.714143, 2.8566],
        [1.0, 0.0, 0.0, 0.0],
        [0.0, 0.714143, 0.7, 2.8],
        [0.0, 0.0, 0.0, 1.0]
      ]
    },
    {
      "file_path": "./train/r_1",
      "rotation": 10.16641,
      "transform_matrix": [
        [0.67549, 0.510997, -0.531597, -2.1264],
        [-0.7373, 0.468115, -0.486986, -1.9479],
        [0.0, 0.720938, 0.693, 2.772],
        [0.0, 0.0, 0.0, 1.0]
      ]
    }
  ]
}
```


Fig. (2) Json file containing the information relative to a training set. In this example, the training set is made up by two images, named r_0 and r_1. The "camera angle x" variable represents the synthetic camera's field of view, while the transform matrix tells us the camera position and orientation in space.

3 General Steps for Creating the NeRF Synthetic Dataset

In order to test out NeRF with multiple custom made synthetic datasets, a plugin for the Unity game engine has been implemented. The plugin consists in a couple of scripts that spawn cameras around a target, giving as output the images captured by the cameras, as well as files containing information about their intrinsic and extrinsic parameters. This is repeated three times to create train, val and test sets. Do note that the following steps are a possible framework to build NeRF datasets, not strict guidelines. In fact, instead of creating multiple virtual cameras, the same camera could be used and moved around the target. The author opted for more cameras for better visualization. The plugin can be found in the project's GitHub page. .

More specifically, the script works as follows:

1. first, the user is required to make the following choices:
   - user chooses a target, the desired 3D model to capture,
   - script spawns dummy camera to let the user customize its intrinsics (e.g. FoV),
   - user chooses the distance from where to take the images of the target,
   - user chooses number of images desired (for train, val and test sets).
   - user chooses in what formation to spawn cameras around target: halo, sphere or semi-sphere (for train, val and test sets)
2. a "Dataset" folder is created, as well as "Train", "Val" and "Test" sub-folders within it. These will be used to store images belonging to each different set [Fig. 1],
3. the following operations are repeated for each set (train, val and test):
   - cameras spawn around the target, with the chosen intrinsics, at the given distance and in the chosen formation.
   - intrinsic and extrinsic parameters of each camera are saved in a json file,
   - each camera captures an image (note: only during the image acquisition for the test set, also depth and normal images are captured, these will not be used by NeRF but might be useful to the user to better estimate NeRF's view synthesis quality),
4. the program ends, the user can move and store the Dataset folder where he pleases.

4 Insights over the use of the Plugin

If the reader chooses to use the presented Unity plugin, the following guidelines and insights might be useful:

- The plugin can be integrated into Unity by drag-and-drop inside the Unity's Assets folder.
- To help the user, a GUI has been implemented, in the form of an Editor window [Fig. 3]. The window can be opened from Unity's Window tab (top left).

- Before the plugin can be used, it is necessary to create a new Unity Tag called "ProCam".
- The target gameobject can be assigned to the target variable trough drag-and-drop. When cameras will spawn, these will point to this gameobject.
- The user is free to assign as target an empty gameobject, using it as a dummy. This might be useful if the user wants cameras to point not exactly to the object's center. Cameras will point to the dummy's location instead, which can be placed wherever the region of interest is.
- The button "spawn dummy camera" spawns a camera which the user can customize to choose the right intrinsics (such as the correct field of view) and in general to better fit its needs. In fact, if desired, additional components can be added to the camera. By clicking "save camera as prefab" the camera is saved as a prefab. It's this camera that will be spawned around the target and used to capture images of it.
- If NeRF is trained using "blender" as "dataset-type", then the images given as input are required to only have the target rendered, not the background (i.e. background must be transparent). The plugin automatically captures a transparent background (whatever the Clear Flags value in the Camera component). However, be sure to capture only the desired target (by disabling unwanted components, or by playing with the Culling Mask of the camera component).
- If any bug or error shows up, most of the times deleting the camera prefab stored in "Resources" solves the problem.
- Users can choose the resolution of the images acquired. Although any resolution is fine, working with 800x800 images is preferable when testing out NeRF for the first time. Alternatively, use 400x400 or 1600x1600 if the default resolution is respectively too low or high.
- Sphere and semi-sphere formations place cameras around the target using the golden ratio sphere sampling algorithm.
- The advanced semi-sphere parameters should be changed only if the existing camera formations around the target are not satisfactory [Fig. 4]. By tweaking these parameters, the user can for example place cameras only on a subsection of the semi-sphere, avoiding to capture the target from the top. Although changing these parameters produces non-intuitive changes in the camera formation, many different configurations can be generated by playing with these. Do note that setting the variable semi-sphere ratio to 0 actually sets it to the *golden ratio* constant.
- Once the "create dataset" process is done, the Dataset can be found in the Assets folder. Suggestion is to move such file out of the Assets folder quickly enough, otherwise Unity will start creating the relative meta files for each image, stalling the engine for a good amount of time and filling the dataset of unwanted files.

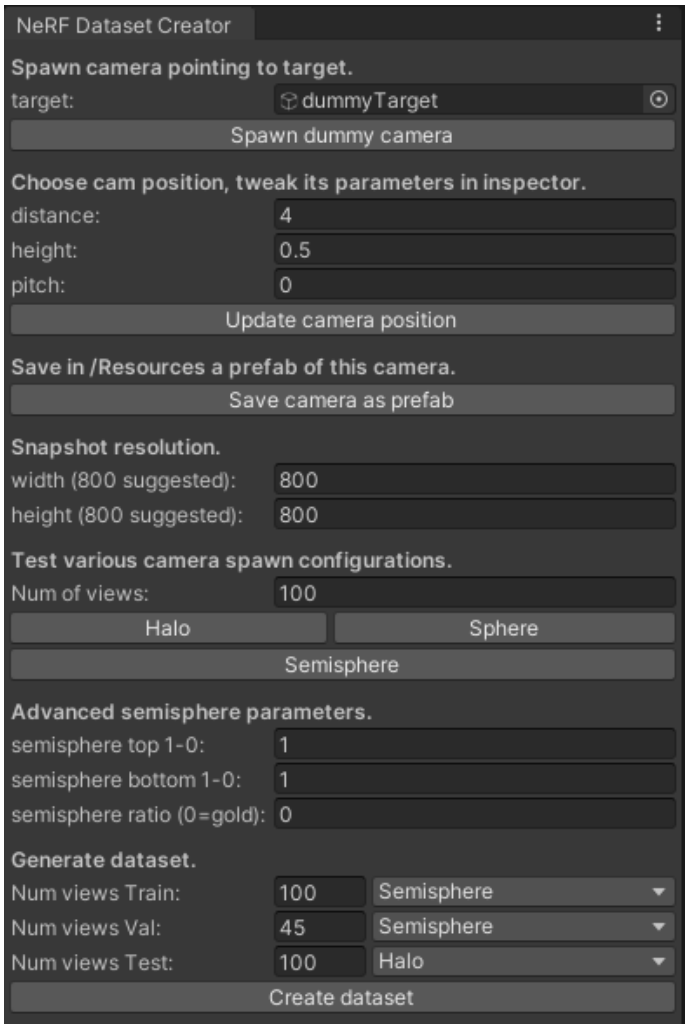
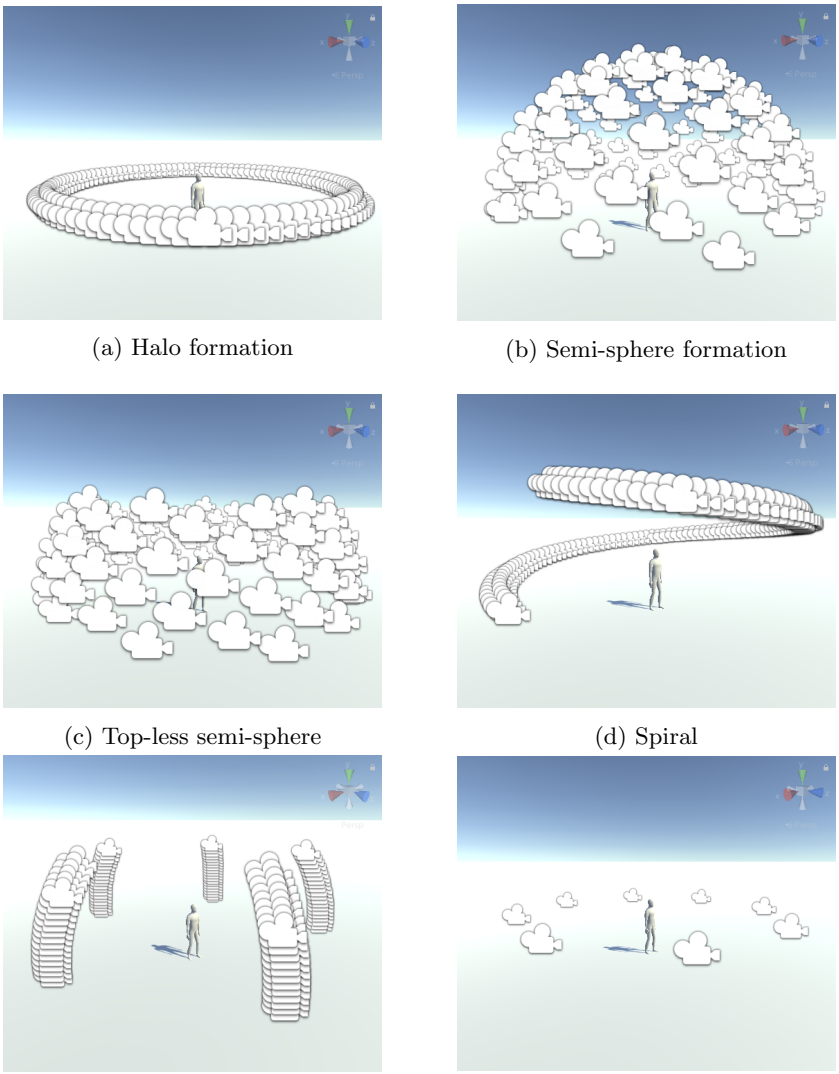


Fig. (3) Screenshot of NeRF's Dataset Creator GUI, with default and suggested values. The GUI suggests the user which steps to follow with brief explicit text descriptions and by grouping related variables and buttons within a same subsection.



(a) Halo formation

(b) Semi-sphere formation

(c) Top-less semi-sphere

(d) Spiral

(e) Column stacked

(f) Eight cameras

Fig. (4) Different possible camera spawning configurations. All cameras are equally far from target and look towards it. Except for halo formation, camera positions are chosen following the golden ratio sphere sampling algorithm. Such different configurations are created modifying this algorithms parameters through the GUI in the "advanced semiphere parameters" section.

5 Results

Hereafter I present some results obtained by NeRF on a custom made dataset. Results have been obtained by running the original tensorflow implementation of NeRF on an Nvidia RTX 2070 Ti. The dataset has 100 images for the training set, 15 for the validation set and 3 for the test set [Fig. 5]. The training was stopped at iteration 10000, with parameters $Nsamples = 64$ and $Nimportance = 128$. The training lasted about 5 hours. Results obtained on the test set are shown in Fig. 6.

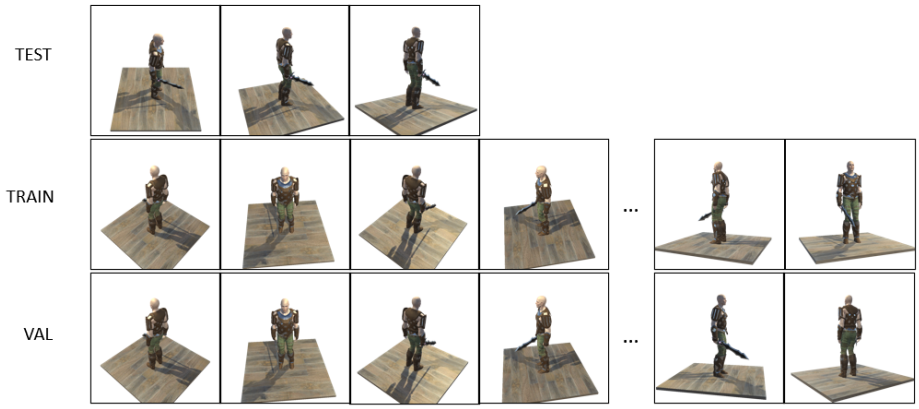


Fig. (5) Screenshot of NeRF's Dataset Creator GUI, with default and suggested values. The GUI suggests the user which steps to follow with brief explicit text descriptions and by grouping related variables and buttons within a same subsection.

6 Possible Improvements

The plugin, although being useful and serving its purpose well, doesn't feel intuitive at first use, especially in the camera creation process.

Furthermore, once all images are captured and the Dataset is successfully created, if the user does not move them out of the Assets folder quickly enough, Unity will start creating the relative meta files for each image. The author has not found any way to avoid this unnecessary process, which stalls the engine for a good amount of time (depending on the amount of images taken) and fills the dataset of unwanted files.

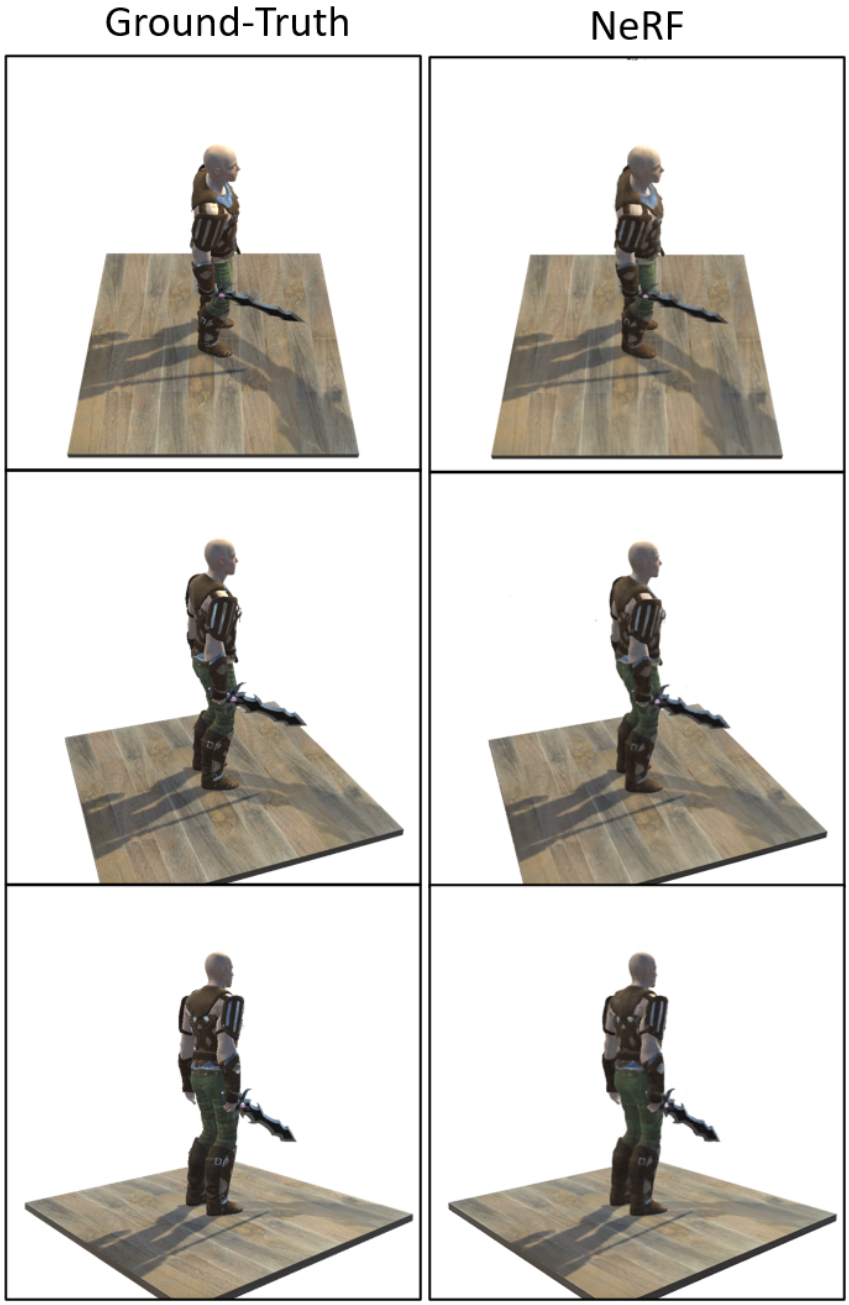


Fig. (6) Screenshot of NeRF's result over the test set. We can see that, aside for some high level detail lost, NeRF reconstructs successfully the scene seen from views on which it was not trained on. Therefore NeRF did indeed understand the 3D structure of the scene and its view dependent lighting.

References

1. Ben Mildenhall and Pratul P. Srinivasan and Matthew Tancik and Jonathan T. Barron and Ravi Ramamoorthi and Ren Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis", 2020, arXiv, cs.CV, 2003.08934.
2. @miscdellaert2021neural, title=Neural Volume Rendering: NeRF And Beyond, author=Frank Dellaert and Lin Yen-Chen, year=2021, eprint=2101.05204, archivePrefix=arXiv, primaryClass=cs.CV