NeRF - Fixing GIFs

Sharon Rotgaizer, Guy Dayan

September 5, 2023

Introduction

The NeRF method achieves state-of-the-art results for synthesizing views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of input views.

Today, 3 years after the original paper came out[1], there are many models that expand the usage of nerf, with wide range of applications.

Project Description

In this research project, we present a new application of nerf using nerfstudio, fixing gifs. Suppose we have some video in hand, we can (if want) to synthesize perfectly closed-loop clip.

Nerfstudio is a library that gives a simple API, which allows for a simplified end-to-end process of creating, training, and testing NeRFs. We use it for training and re-rendering existing video into the desired gif.

We assume that the first and last frame have partial common view, mostly because of nerfstudio limitations to places it hasn't seen. From the other side, we expect it to work, if the missing part was seen by some frame.

Methods Used

We used the following infrastructure:

1. COLMAP - this library takes images and calibrates all the cameras in the 3d space.
2. Nerfacto - this is the neural network model we use to train the 3d scene. Among the available models of nerfstudio (which suggest the major ones), it is the best to train on real data.

[1] NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis: https://arxiv.org/abs/2003.08934

Algorithm

Our algorithm takes a video, learns the 3d scene from it and uses the cameras calibration (both intrinsic and extrinsic) to generate smooth curve between the end and start of a clip, creating the smooth gif.

1. COLMAP - we run colmap to calibrate all the cameras and their position + orientation.

2. Training - we run the training with nerfstudio on Nerfacto model.

3. Loop detection - We define overlapping frames as pair of frames with the highest amount of (SIFT) matches, but one frame is relatively close to the start and the other is far enough from the start. We apply this heuristic twice – one time relative to the first frame and another time relative to the last frame. We measure "far enough" from the start by demanding the frame's id to be at distance of at least $\mathcal{F}/3$ from the start, where $\mathcal{F}$ is the total frames number. The "far enough" property will assure us that if two frames are matched, they probably define some circular shape.

   After having two pairs, (first frame, x), (y, last frame), we choose the pair of frames with the smaller euclidean distance, in order to cover larger part of the cycle. We remove all the frames with ids that are not in the selected range

4. Outliers detection - we take the remain frames from the previous step. If no loops were detected, all the frames will be selected. We than generate spline that should pass through our frames but yet connect the last frame to the first one. Frames at the end & start that are too far from the spline[2] are removed.

5. Spline generation - we create a smooth curve between the last frame and first frame with equal spacing. We generate n new frames by:

$$\frac{\mathcal{L}_{\text{new}}}{\mathcal{L}_{\text{origin}}} = \frac{n}{\mathcal{F}}$$

   where $\mathcal{L}$ is the curve length and $\mathcal{F}$ is the total frames number.

   In order to calculate $\mathcal{L}$, say we are given a spline s in finite range, and we can generate points on it, using $s(u)$, where $u \in [0, 1]$. Actually, the spline is given in vector form, $\overrightarrow{s}(u) = \langle f(u), g(u), h(u) \rangle$ (s.t. $x = f(u)$) and the curve length from a to b given by:

$$\mathcal{L} = \int_a^b \sqrt{f'(u)^2 + g'(u)^2 + h'(u)^2}\, \mathrm{d}u$$

6. Camera path rendering - we using all the positions (original and new) and orientations, to generate camera matrices. We than save them in the format required by nerfstudio. We also estimate the fov of the video by the formula

$$f = 2 \cdot \arctan\left(\frac{h}{2 \cdot f_y}\right) \cdot \frac{180}{\pi}$$

   where h is the frame height and f_y is the focal length along y axis.[3]

[2] We determine this by heuristic and hyper-parameter

[3] OpenCV, calibration implementation, here

7. Rendering - we use the camera path to render the final frames and GIF. We also render:
    - Illustration to the camera travailing among the frames.
    - frames plotting at top view, which shows the original frames, those deleted by the loop detector, those deleted since being outliers and the new generated ones.

Evaluation

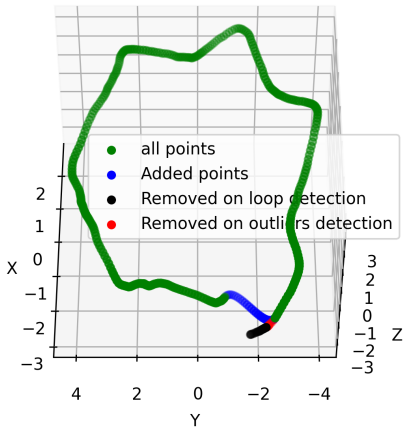
We used four datasets:

- Soldier - a person walks around chair with soldier model on it, creating 340° non stable cycle.
- Chair - a person walks around chair with water bottle on it, creating 1.5 non stable cycle.
- Egg - a person walks around chair with PlayStation console on it, creating little more than one complete non stable cycle at egg (ellipse) shape.
- Room - a person is moving around itself recording his room, with up & down movements, creating less than one cycle.

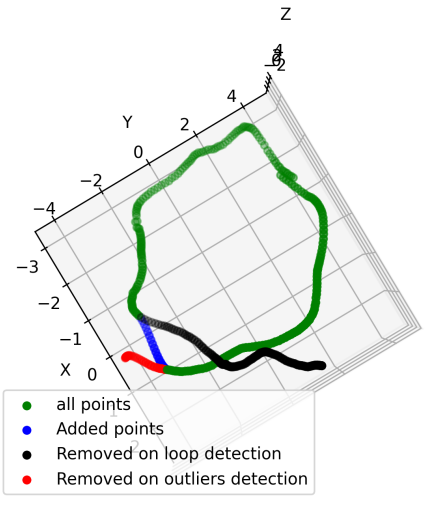
For best results, we trained on nerfacto-huge model. All the code and datasets are available in our Git repository, along with installation guide.

Results

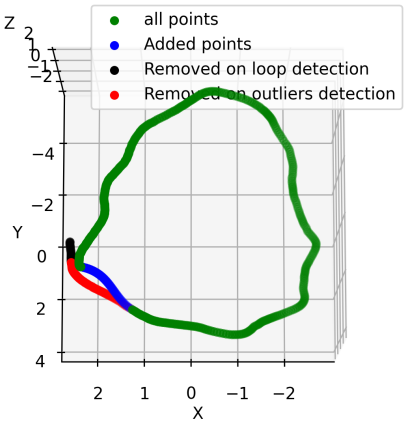
We present here the original frames (r,g,black), those deleted by the loop detector (black), those deleted since being outliers (red) and the new generated ones (b).



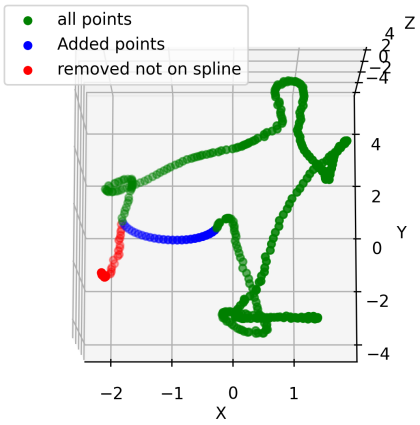
(a) Soldier frames



(b) Chair frames



(c) Egg frames



(d) Room frames

Nerf vs reality

One problem came to us when rendering with nerf is that nerf is very sensitive to colors, transparency, occlusions and distortion. We could improve the quality of the gif by copying the original remained frames, but it will damage badly the embedding with the new generated frames.

Figure 1: in the left image there is an original frame and in the right side, a generated frame from the same position and angel.