MATHÉMATIQUES
VISION
APPRENTISSAGE

Master MVA

Nuage de points et modélisation 3D

View Synthesis with sculpted neural points - Project Report

Student :
Ugo Insalaco

Code is available at: `https://github.com/Ugo-Insalaco/MVA-SNP`
SNP Author's code: `https://github.com/princeton-vl/SNP`
March 17, 2024

1 Article presentation

1.1 Task:

The article addresses the task of novel view synthesis. The objective is to take as input several images of a common scene and be able to infer new views from this same scene. The original approach proposed by the authors is to represent the underlying scene by an explicit point cloud that is then reshaped (or sculpted) before being rendered through a differentiable rendering process.

1.2 Point cloud extraction

The point cloud extraction is done using a MVS network based on Raft ([3]), trained on the DTU dataset ([1]), which produces dense depth maps that are next un-projected to 3D points. MVS (Multi-View Stereo) is the task of extracting a dense point cloud from given images and camera parameters.

1.3 Point Sculpting

The MVS network produces point clouds that are often incomplete or with outliers, this is why the authors propose to enhance this point cloud before rendering by removing certain outliers (pruning) and filling holes with rays of points (adding). Using the paper notation, for $H \times W$ RBG images $\{I_1, \dots I_m\}$ and cameras $\{C_1 \dots C_m\}$ with $C = (K_C, R_C, t_C)$ the intrinsic parameters, rotation and translations. The projection functions Π and its inverse are defined by:

$$\Pi(P, C) = [K_C(R_C P + t_C)]^{\downarrow}; \Pi^{-1}(p, d_p, C) = R_C^{-1}\left(K_C^{-1} d_p \begin{bmatrix} p \\ 1 \end{bmatrix} - t_C\right) \tag{1}$$

where $[X, Y, Z]^{\downarrow}$ is defined by $[X/Z, Y/Z]^T$, P is a $3D$ point, p a $2D$ point and d_p the associated depth.

Intuitively, **pruning** removes point that are isolated in front of some cameras and far from the surface. Interestingly, it does not consider points that are far away behind the object, not visible from any cameras which may leave trails behind the final point cloud if it was not captured from all angles. Formally, a pixel p in view i and depth d_p^i is kept in the point cloud if it verifies:

$$\bigcap_{j=1}^{m} [D^j \Pi^{-1}(p, d_p^i, C^i) \geq \delta_p d_q^j] \tag{2}$$

with D^j the depth of a $3D$ point in view j and q the point corresponding to p in view j: $q = \Pi(\Pi^{-1}(p, d_p^i, C^i), C^j)$

The **adding** part fills holes by casting rays from each camera and each pixel, and by keeping equally spaced points on the ray if the new points don't mask any existing point

from any view. In practice, during training the pruning step is followed by an optimization step, then the adding step, and a final optimization step.
One thing to note is that the added points have no particular color associated: they are always added in red, and count on the next featurization step to acquire the color meaning.

1.4 Feature extraction and differentiable rendering

From each point in the cloud, a learnable feature vector f_i and opacity o_i are associated. Spherical rendering ([8]) on the feature space then uses an input direction to refine the view-dependent effects into new features s_i, and finally a soft rasterization algorithm called Pulsar ([2]) transforms that feature vector into a 2D feature map, which is eventually fed into a modified U-Net ([5], referred as "shading_arch" in the code) that produces the resulting image.

1.5 Evaluation

Evaluation of the newly created images from the point cloud is made using PSNR, SSIM and LPIPS Metrics. PSNR (peak signal to noise ratio) is computed via: $PSNR = 10 \log_{10}\left(\frac{d^2}{MSE}\right)$ where d is the maximum value of the signal (e.g 255) and MSE the mean squared error between the target and the reconstructed image: $MSE = \frac{1}{hw}\sum_{i,j}\left(I_0(i,j) - I_r(i,j)\right)^2$
Intuitively, PSNR measures the image quality with a reference pixel by pixel, and is high if the images are closer. SSIM is a metric aiming at computing the structural similarity between two images rather than a pixel by pixel difference. It uses global metrics on the images such as mean, variance and covariance. LPIPS corresponds to the cosine similarity in the feature space of a pre-trained CNN (usually VGG). If two image present the same features in the network space, then the LPIPS metric will be high.

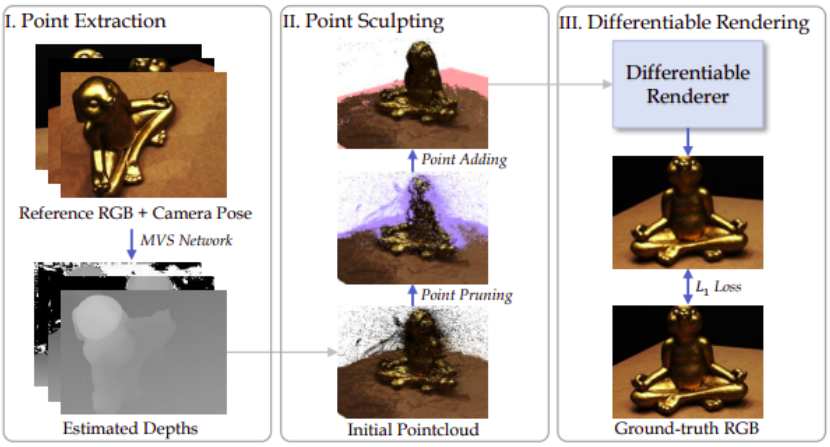


Figure 1: Pipeline to novel view synthesis using Sculpted Neural points [9]. The MVS network if based on Raft [3]. The differential renderer uses learnable feature vectors, followed by Spherical Harmonics functions [8], Pulsar [2] and a modified version of U-Net [5]

2 Reproducing the LLFF Examples

The LLFF dataset [4] was created for evaluating the task of Multi-view Stereo. It consists of a dozen of scenes captured from 20 to 50 different views, associated to their respective depths. Joint with their article, the authors provide well formatted data, pre-trained models and refined point clouds. The first task of this project is to reproduce some of their experiments, which results are visible in fig.2. We observe images that are visually pleasing for newly generated views, and that are consistent with the authors' versions, we also obtain metrics close to the average evaluation of the initial article on the LLFF dataset tab.1.

(a) fern (b) leaves

Figure 2: Novel view synthesis for two LLFF examples I obtained by running an evaluation with the author's models

Scene	PSNR	SSIM	LPIPS
fern	23.73	0.782	0.225
leaves	18.89	0.647	0.298
authors	25.32	0.817	0.229

Table 1: Obtained metrics for two LLFF examples. Authors line refers to the average results they obtained on the LLFF dataset [9]

3 Contribution

The main contribution of this project is oriented towards reproducibility. In particular, I provide Docker images and code to easily run the novel view synthesis on the LLFF examples and to create brand new views from any user's custom data using the code provided by the authors. More precisely, after setting up a usable environment and fixing code issues, I reproduced the different steps (fig.3) to infer novel views from a set of original images by:

1. Running Colmap on the data to produce a raw point cloud and depths maps as well as finding the camera parameters.

2. Transforming the Colmap output into DTU format which is understood by the authors code
3. Fine-tuning the Raft model to obtain refined depth maps
4. Training the Pulsar model, which yields a sculpted point cloud and trained rendering model
5. Rendering novel views from the Pulsar model

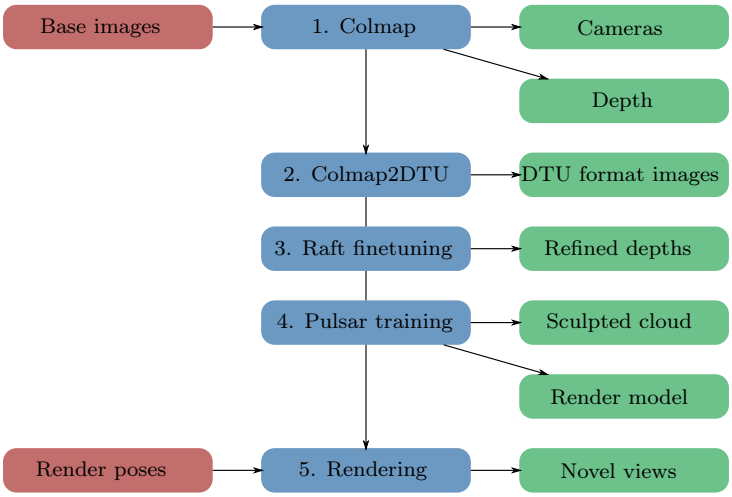


Figure 3: Contribution: Reproducing the full novel view synthesis pipeline based on Sculpted Neural Points. In red are the inputs, blue the different algorithms or steps applied in the pipeline and green some outputs provided by these algorithms

3.1 Environment setup: Docker & Google Colab

Docker is virtualization software, providing the ability to create containerized environments which are useful for reproducibility across platforms. The use of Docker aims at solving several issues.

Pytorch3d: Pytorch3d provides utilities for various 3d engineering tasks. However it is subtle to install, with specific environment dependencies. Using Docker allows to control such environment and install a pre-built wheel for this package.

Nvidia compatibility: Along with Pytorch3d, Nvidia CUDA can cause compatibility issues. Fortunately Nvidia has a toolkit which provides Docker an interface with CUDA and pre-built GPU drivers.

Colmap: Being a wildly used software, Colmap [6, 7] also has a pre-built image which is faster to use than rebuilding all the binaries from scratch. Colmap will be used later for inference with custom data.

Independently from the environment issues, the algorithms involved in Sculpted Neural Points can have large GPU requirements. Along with the Docker image and scripts, I

also repeated some parts of the training process on Google colab which gives free access to more GPU power. However these environment are not fully controled, they are still limited in their usage and could be broken in the future. Finally, the scripts I provide aim at automatically handling the large and diverse amount of files produced during the entire pipeline of novel view synthesis. It also automatically downloads the prebuilt models and point clouds given by the authors on Google Drive. As for now, downloading files from Google Drive still has one issue due to the limitation on the number of files that can be downloaded at once.

3.2 Code fixes: Colmap2DTU & Render poses

There were two limiting factors to reproducibility within the code:

Colmap2DTU: The code provided by the authors had a flaw at the second step to convert Colmap output into DTU compatible format. The written algorithm relied on a function which goal was to use the Colmap *fused.ply.vis* file. This file saves the different point visibility status from each camera. The broken algorithm was designed to transforms this visibility file into mask images for each camera, each pixel being black or white according to the visibility of the targeted point. I fixed this issue and obtained the disired masks (fig.6)

Render poses: The second limiting factor to reproducibility was the lack of a simple manner to create new render poses for the inference. Usually in the examples provided by the authors, a *render_poses_raw.npy* file was given, consisting of a set of camera parameters, arranged in a circle around existing cameras. This file is used at evaluation time to create new images from the corresponding views. My contribution is to provide a simple jupyter notebook to help users choose a reference camera and create such a file automatically (fig.10).

3.3 Novel View Synthesis

In this section, I start from original images of a rose (fig.4) and apply the different transformations and model trainings to obtain novel views. My data is composed 19 pictures of size 4608 x 3072. All point clouds depicted are available in the source code next to the report images.

3.3.1 Colmap

In this first step, I ran a Colmap SfM / MVS pipeline with the same parameters as the authors to obtain the raw point cloud and the camera parameters (fig.5). A first observation on the fused point cloud is that even if the point cloud seems well reconstructed, it still shows some holes adn sparsity, which justifies the use of a refining Raft model.

3.3.2 Colmap2DTU

The Colmap output then needs to be converted to a usable DTU format. This step leverages my implementation of the mask computation for each camera (fig.6).

Figure 4: Custom data to be processed

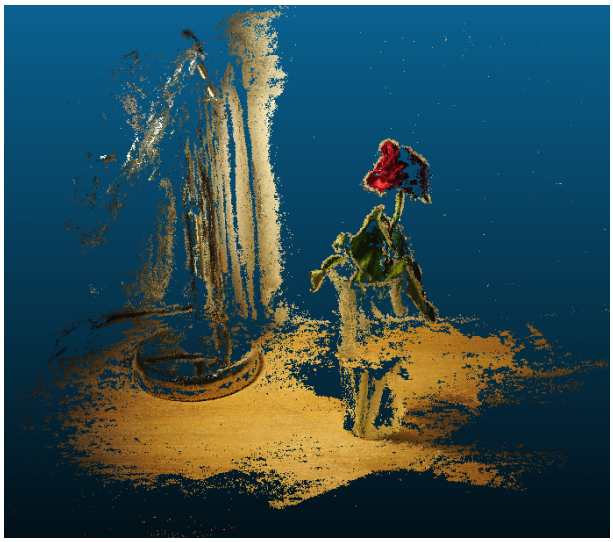


Figure 5: Fused pointcloud obtained by Colmap

3.3.3 Raft Refining

The DTU formatted data is then used to finetune Raft in order to produce a denser point cloud and dense depth maps (fig.7). During this process Raft uses the Colmap point cloud at two scales. However an issue with the code or the method made use of a wrongly computed ground truth point cloud for the second scale (fig.8 (b)). Luckily this didn't have a large influence on the rest of the pipeline and allowed me to continue.

3.3.4 Pulsar Training

Training the Pulsar model required a large amount of resources and was made on Colab. The authors setup their code to run for 50000 epochs, however with my low amount of computing power I only managed to run it for 5000 epochs which, just started yielding interesting results. A part of the training consisted of applying the point adding method, which result is shown on fig.9. I observe here that the majority of the points were added to the border of the point cloud, elongating existing trails or creating new ones. In this

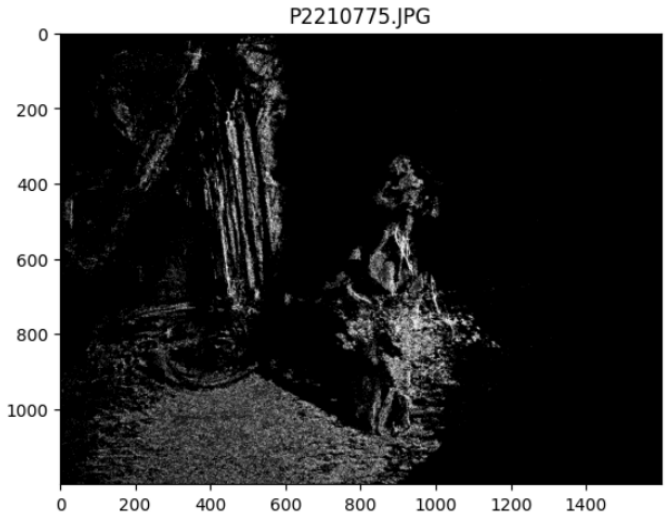


Figure 6: Mask obtained using my modified Colmap2DTU function

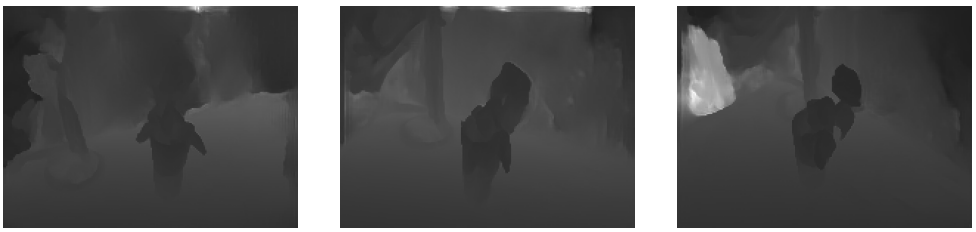


Figure 7: Refined depths maps using Raft

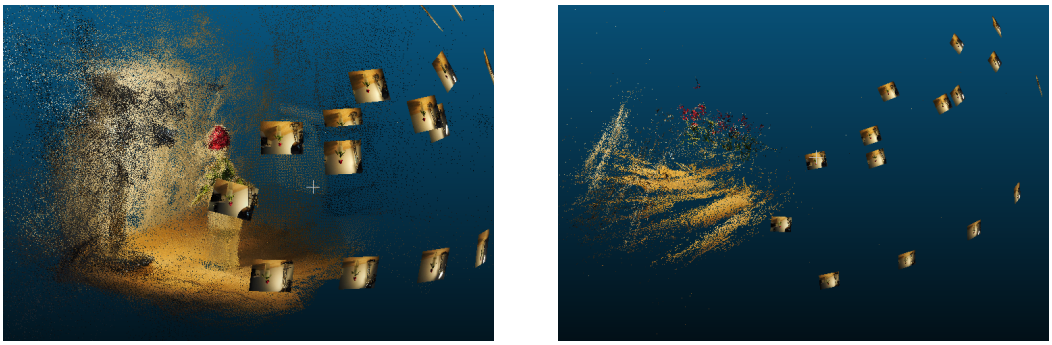
example it raises the question to the relevance of this step that visually didn't seem to have filed any holes in the point cloud and even added irrelevant point to the foreground.

3.3.5 Rendering

Before rendering I ran my notebook to create new poses for the inference (fig.10). Final results can be seen on fig.11. At first sight, according to the camera angle of the rendered views, the notebook seems to be functional. However the image quality is low compared to the example results (tab.2) and is, in my interpretation, a direct consequence of the very low number of training steps. Looking at the gif version (available in the code) we can also observe that the lamp was not well reconstructed in the views as its position bounces from one image to another.

epochs	PSNR	SSIM	LPIPS
2500	19.04	0.790	0.597
5000	18.96	0.791	0.579

Table 2: Reconstruction quality metrics obtained after 2500 and 5000 training steps of the Pulsar model



(a) (b)

Figure 8: Refined pointclouds using Raft at scale 1 (a). Reference pointcloud for training Raft at scale 2. A wrong computation in the process made this ground truth erronated (b)

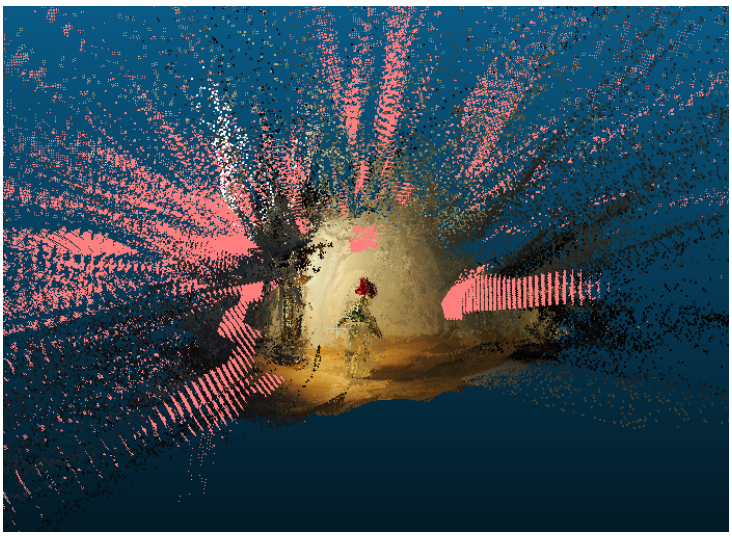


Figure 9: Added point on the refined pointcloud. Few points seem to contribute to the quality of the pointcloud.

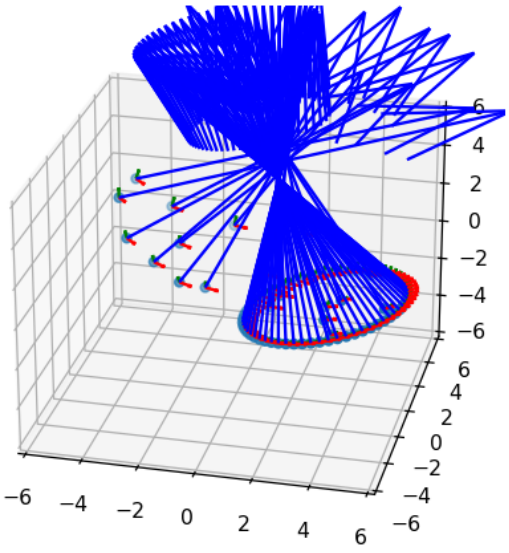


Figure 10: New poses automatically computed using my jupter notebook. In blue are the local z axis that are converging towards a common object. The created circle has consistent parameters with the existing cameras.

Figure 11: Final render for novel view synthesis (gif version is available in the code)

4 Conclusion

We have seen during this project a way to tackle the challenging task of novel view synthesis using point cloud reconstruction, sculpting and differentiable rendering. Although this project doesn't propose any new improvement of the method, it provides future users a way to reproduce the authors work simply, using docker and a fully working pipeline for original data.

If I were to refine this project with time and resources, I would focus on four axis:

- Solving the scale issue when finetuning the Raft model (fig.8 (b))
- Training an entire rendering model, fully automate the pipeline with a single script, and better handle the hyper parameter configuration of this pipeline (which for now uses exactly the same hyper-parameters than the authors at each step).
- Study how to use the color at the point adding step, maybe by using the neighbors of the added points and the effect on the final rendering.
- Compare the actual effectiveness of the adding step in my example, that seemed limited.

This project still has some limitation: the docker image created relies on a large amount of packages and files which makes it heavy (about 10Gb) and can probably be improved. Due to the nature of the method, this novel view synthesis technique is also heavy in terms of resources and time. In my experience a low bound for reproducing the entire pipeline, with full trainings, would be around 2 hours for Raft finetuning + 12 hours for the full Pulsar training using T4 GPUs.

References

[1] Rasmus Ramsbøl Jensen et al. "Large Scale Multi-view Stereopsis Evaluation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 406–413. URL: https://api.semanticscholar.org/CorpusID:18412989.

[2] Christoph Lassner and Michael Zollhöfer. "Pulsar: Efficient Sphere-based Neural Rendering". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), pp. 1440–1449. URL: https://api.semanticscholar.org/CorpusID:235601887.

[3] Zeyu Ma, Zachary Teed, and Jia Deng. "Multiview Stereo with Cascaded Epipolar RAFT". In: *ArXiv* abs/2205.04502 (2022). URL: https://api.semanticscholar.org/CorpusID:248665984.

[4] Ben Mildenhall. "Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines". In: 2019. URL: https://api.semanticscholar.org/CorpusID:260545139.

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *ArXiv* abs/1505.04597 (2015). URL: https://api.semanticscholar.org/CorpusID:3719281.

[6] Johannes Lutz Schönberger and Jan-Michael Frahm. "Structure-from-Motion Revisited". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[7] Johannes Lutz Schönberger et al. "Pixelwise View Selection for Unstructured Multi-View Stereo". In: *European Conference on Computer Vision (ECCV)*. 2016.

[8] Alex Yu et al. "PlenOctrees for Real-time Rendering of Neural Radiance Fields". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), pp. 5732–5741. URL: https://api.semanticscholar.org/CorpusID:232352425.

[9] Yiming Zuo and Jia Deng. "View Synthesis with Sculpted Neural Points". In: *ArXiv* abs/2205.05869 (2022). URL: https://api.semanticscholar.org/CorpusID:248721740.