DEEP MULTI-VIEW STEREO FOR GTSFM

A Thesis
Presented to
The Academic Faculty

By

Ren Liu

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science
School of Interactive Computing
College of Computing

Georgia Institute of Technology

May 2022

DEEP MULTI-VIEW STEREO FOR GTSFM

Thesis committee:

Dr. Frank Dellaert
Advisor
Georgia Institute of Technology & Google Research

Dr. James Hays

Georgia Institute of Technology & Argo AI

Dr. Cedric Pradalier

Georgia Institute of Technology

Date approved: April 26, 2022

ACKNOWLEDGMENTS

I would never be able to accomplish this work without the help of my thesis advisor, Professor Frank Dellaert. He has spent a lot of time giving me the most professional mentoring I have ever received. Being a member of Frank's team is my honor and an unforgettable experience that will benefit me a lot.

I am really grateful for Frank's consideration to approve my request to work in his team and for my degree thesis. I had almost given up hope and confidence to complete my Master's degree in the thesis option until Frank sent me the email just before the thesis request deadline. Although working in his team and reporting the updates in a weekly meeting is never an easy thing, I really appreciate Frank's "bottom line": to present what you are sure is true clearly to every team member even if they do not have the preliminary knowledge. I believe this criterion helps me so much to gradually become a qualified graduate student from a kid only with enthusiasm and interest. As my thesis work includes many surveys on the latest works, I am lucky to have the opportunity to read a lot of papers with Frank. I really enjoy the experience we dug into the origin of a theorem, a formula, or even a symbol or terminology. His idea, his willingness, and his dedication teach me how to think and act professionally in computer vision.

I would also like to present my thanks to all my collaborators in Frank's lab: John, Akshay, Ayush, Jing, Adi, Travis, and Jon. The experience we work for building and improving GTSFM is so valuable to me that now I keep using GTSFM's contributors' standard in any other project of mine. Your technical and academic depth broadened my horizons and taught me a great deal.

I also acknowledge the care and support of my girlfriend Kelin. She always encourage me to overcome the communication barrier and to challenge myself.

I would also like to appreciate all the members of my thesis committee, Frank, James, and Cedric, for their great help to this work.

Finally, the author appreciates the support offered by Georgia Tech. Any experiments result, point of view, and conclusions contained in this thesis are collected and drawn by the author.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

FOV Field of View

GTSFM Georgia Tech Structure from Motion

MLP Multi-layer Perceptron

MVS Multi-view Stereo

NeRF Neural Radiance Field

NGP Neural Graphics Representations

SDF Signed Distance Field

SFM Structure from Motion

SUMMARY

Structure-from-Motion (SFM) is a very important technique in our daily life. It estimates 3D information from a collection of 2D images, including the camera parameters and the sparse correspondences. But this is not the end. To help SFM process to build an almost complete connection between 2D images and 3D worlds, Multi-view Stereo (MVS) and view synthesis are two indispensable techniques. These two techniques often take the output results of SFM pipelines as input data. MVS process aims at building dense correspondences of the foreground objects, while view synthesis focuses on generate photo-realistic images from novel viewpoints. These technologies have been widely used in our daily lives, such as reconstructing traffic conditions for automated vehicles, adding virtual but reasonable assumptions to the real world by augment reality (AR) or build virtual 3D spaces for human users via virtual reality (VR).

Current SFM pipelines integrated with MVS module often applies traditional MVS methods. These algorithms cannot be well parallelized and can be slow when the data size and the resolution increase. For view synthesis, however, there are seldom SFM pipelines integrating it. Therefore, this thesis focuses on how to integrate MVS and view synthesis efficiently into SFM pipelines, especially for the latest deep learning approaches. We first do a thorough survey in both domains, compare the advantages and disadvantages of the latest studies, and select the best fit approach for our distributed SFM pipeline, Georgia Tech Structure from Motion (GTSFM) [1]. We implement the deep multi-view optimizer with PatchmatchNet [2] and integrate it into the working graph of GTSFM. We also design an algorithm to boost a novel deep view synthesis algorithm, Instant-NGP [3], by forcing the reconstruction region on the overlapping Field-of-Views. This also enables us to extract high-quality dense polygon meshes of foreground object directly from the reconstructed depth field. Experiments run on DTU [4] and Skydio Crane Mast datasets suggest our MVS approach is more efficient than some popular SFM pipelines with MVS implemented.

CHAPTER 1
INTRODUCTION AND MOTIVATION

1.1 Background

Computer vision algorithms can build tight connections between 2D images and 3D objects. The process of estimating the locations of 3D points from multiple images is commonly known as *structure from motion* [5], or *SFM* in short [6]. This converse process to build connections from 2D to 3D space [7] brings great convenience when the 3D models are complicated or dynamic. SFM process estimates fundamental parameters for a scene, including camera poses, object point coordinates and so on, which is an important step for building stereo vision. Further processes like *multi-view stereo* (or in short *MVS*) and view synthesis can be performed based on SFM results. With the estimated camera parameters from SFM process, MVS reconstructs a complete 3D object model from a collection of images taken from known camera viewpoints [8], while view synthesis can create images of the estimated scene as it would appear from novel viewpoints [9].

There are still many challenges for these processes, though. One of the most common problems is how to keep high computational efficiency while the resolution of images and the size of dataset increases. The popularization of photo-sharing applications and high-quality mobile image acquisition devices presents both an opportunity and an challenge [10]. Parallelization is a common approach. With the emerging and development cloud computing [11], scientists have been starting to apply distributed computing strategy to improve the scalability of existing computer vision systems [12, 10], and have made great progress. Another method is to apply pre-trained neural networks to perform faster and better in complex tasks. For example, object recognition [13, 14, 15] and image segmentation [16, 17, 18].

However, state-of-the-art SFM systems, from OpenMVS [19], Theia [20] to OpenSfM [21], OpenMVS [22] to COLMAP [23], Meshroom [24], very little of them has applied with either distributed system or pre-trained deep learning methods into their SFM pipelines. That is where Georgia Tech Structure-from-Motion (GTSFM) [1] was born. GTSFM is an end-to-end global SFM pipeline designed to natively support parallel computation using Dask.

In this work, we demonstrate how deep learning approaches can be integrated into GTSFM for MVS and view synthesis tasks. For each task, we will show a detailed survey and comparison on existing advanced deep learning approaches, how we integrate the selected method into GTSFM, and a number of advantages that our implementations bring to GTSFM compared with other state-of-art approaches.

1.2 Unsolved Problems in SFM pipeline

1.2.1 Challenges in MVS: Dense Correspondences Reconstruction

Multi-view stereo (MVS) algorithms [25, 26, 8] has been a popular topic of research for decades. In the SFM process, 3D points on objects estimated are filtered among tracks of matched features detected by descriptors. MVS, however, requests to produce complete 3D surfaces [7], of which the output result is usually a 3D volumetric descriptions (e.g. density field) or the reconstructed surfaces (e.g. a dense polygon mesh) for the target 3D object. Smooth surfaces can be obtained from a dense point cloud with normals [27, 28, 29], and a dense point cloud can be simply obtained by fusing reconstructed depth maps from multiple views. Therefore, how to efficiently make use of SFM results to assist MVS algorithms to compute optimized dense depth maps can be a meaningful topic.

Among state-of-art SFM pipelines with MVS process, Meshroom [24] applies Semi-Global Matching (SGM) method [30, 31], which is based on pixel-wise matching of mutual information and estimating a global and smooth depth map. COLMAP [23] and OpenMVS [19] both apply algorithms combining ideas from both PatchMatch algorithm [32] and

plane sweeping stereo [33, 25, 34]. These matching-based can output great optimized depth maps from all calibrated cameras in SFM, and can be parallelized in view level because each depth map is reconstructed independently. However, all these algorithms cannot be parallelized in pixel-wise computation. The efficiency of SGM method will be influenced by the number of image pixels and depth/disparity hypotheses. Zheng's EM-based PatchMatch [35] used by COLMAP can be further parallelized between image rows, but such complexity will still suffer from the increasing image resolution and the increasing size of dataset.

Recently, remarkable progress has been made in the domain of deep-learning based MVS [36, 37, 38, 39, 2]. Pixel-wise computation can be parallelized and reconstruction performances are improved. But until now, there are few SFM pipelines integrating these latest approaches to obtain dense correspondences. Also, how to leverage SFM results to help provide MVS algorithms with better speed and quality still remains unexplored.

1.2.2 Challenges in View Synthesis: Neural Geometry and Radiance Field Reconstruction

Traditional view synthesis algorithms rely on a dense sampling of views to build photo-realistic novel views by interpolating light fields [40, 41, 42]. For novel view synthesis with sparser view sampling, geometry and texture representations of the target object are estimated by observed images. Gradient-based mesh optimization by differentiable rasterizers [43, 44] or pathtracers [45, 46] can be solutions but the implementations are often difficult and complicated. Their performances are also largely influenced by the quality of initialized template meshes, which is unavailable for unconstrained real-world scenes [47].

With the success in studies on neural graphics representations, the emerging of neural presentations for surface [48] using signed distance field (SDF) and for volume using density field [47] are proposed and neural rendering by reconstructing radiance field [47] makes it possible to implement gradient-based view synthesis by deep neural networks. Since 2020, Neural Radiance Field (NeRF) related studies [47, 49, 50, 3] have been mak-

ing deep-learning based view synthesis algorithms much faster. These novel techniques have not been integrated to SFM pipelines because of long scene-dependent training time and high computational capabilities requested.

1.3 Thesis Contributions

Table 1.1: Thesis contributions towards integrating MVS and View Synthesis approach

Process	Surveyed Approaches	Selected Approach
MVS	MVSNet [36] , R-MVSNet [37] Point-MVSNet [39], Fast-MVSNet [38] CasMVSNet [51], PatchmatchNet [2]	PatchmatchNet [2]
View Synthesis	NeRF [47], MVSNeRF [49] Instant-NGP [3], Plenoxel [50]	Instant-NGP [3]

My thesis work mainly focuses on surveying latest studies on MVS and view synthesis, selecting the best-fit approach and integrating it into GTSFM. In this thesis, I demonstrate a thorough review on many of the latest techniques (See in Table Table 1.1) that dominates multiple benchmarks [4, 52, 53, 54] and choose the approach to integrate by comparing their advantages and disadvantages with the demand of GTSFM. I will also present some improvements I made to enable the selected approach to have even better performance user GTSFM with experiment statistics. In detail:

For MVS process, I decide to integrate PatchmatchNet because of its fast execution speed, great performance on most benchmarks, and its potential to be further parallelized. To make it fit with GTSFM pipeline, I implement the depth-range estimation and view selection modules for PatchmatchNet from SFM results computed by GTSFM. To make PatchmatchNet a complete MVS process, I apply post-processing techniques including normal estimation, redundant point removal, and surface reconstruction to the dense correspondences output by PatchmatchNet.

For view synthesis, I select Instant-NGP for its high efficiency without complex neural network architectures. To make better use of SFM results to accelerate Instant-NGP, I im-

plement an overlapping Field-of-View (FOV) estimation module from the calibrated cameras to limit the inference and rendering space for Instant-NGP. This module also makes it possible to extract neat dense point clouds for target foreground objects from the predicted density field by Instant-NGP.

1.4 Conclusion

In conclusion, latest deep learning methods have beaten the traditional MVS (Patchmatch-Net) and view synthesis (Instant-NGP) algorithms in both efficiency and performance. We select best-fit deep-learning based MVS and view synthesis approaches for modern SFM pipelines and integrate them into our distributed GTSFM pipeline. Finally, we make improvements on both algorithms to make them even faster and more robust.

CHAPTER 2

DEEP MULTI-VIEW STEREO FOR GTSFM

In this chapter, we integrate a deep-learning based Multi-view Stereo (MVS) approach into our Structure-from-Motion (SFM) pipeline, Georgia Tech Structure From Motion (GTSFM). Before integration, we focus on comparing different latest deep-learning MVS methods with great performances on most popular benchmarks and analyze which one fit best with our distributed SFM pipeline. We evaluate all methods using scenes from DTU dataset[4]. We also make COLMAP [23]'s MVS results as a baseline in the experiments. We show that PatchmatchNet [2] has the best performance in reconstruction quality, speed, and scalability. Then we modulate PatchmatchNet to fit best with GTSFM's work graph and design a set of subsequential processes including normal estimation and redundant point removal specially for the modulated PatchmatchNet. We also develop a set of metrics to evaluate the MVS results for GTSFM.

2.1 Introduction

MVS is an important module for a complete SFM pipeline. It builds complete models of foreground objects in the scene as a dense volume or a polygon mesh, which often plays the role to combine and present the final SFM results to user's interface at the last stage of the whole SFM pipeline. Many popular SFM pipelines have implemented MVS module. For example, COLMAP [23] and OpenMVS [19] use Patch-based stereo algorithms [35, 55] to compute dense point cloud for surface reconstruction algorithms like Poisson [27] and Delaunay [56].

Since the sucess of MVSNet [36] in 2018, more and more efficient deep-learning based MVS approaches emerge and lead most of the MVS benchmarks [4, 52, 53]. We are interested in how these deep-learning based methods extends ideas from traditional MVS

algorithms and how they are different with each other in implementation and performance. As a distributed SFM pipeline built for large datasets, time efficiency, scalability and robustness of the integrated MVS methods are key factors for GTSFM to consider.

In this way, we make a thorough survey of existing deep-learning based MVS methods and integrate the most suitable method into GTSFM in order to produce a complete object reconstruction. Our contributions include:

- We show by complexity analysis and experiments that PatchmatchNet [2] performs the best among tested deep-learning MVS approaches and is suitable to be integrated into SFM pipelines.
- We integrate PatchmatchNet into GTSFM as a MVS module. We also parallelize its inference loop and implement extra operations to build polygon meshes from the output dense point cloud, including normal estimation, redundancy removal, and surface reconstruction.
- We show by experiments that GTSFM performs more efficiently than existing SFM pipelines in MVS process.

2.2 Related Work

In this section, we briefly introduce approaches in traditional and deep-learning based MVS, dense point cloud compression and simplification, and surface reconstruction.

Traditional Multi-view Stereo (MVS). Traditional MVS approaches without deep learning often requires to do image rectification first and then query the best disparity by comparing the left and right views. For example, block matching [57] algorithm shifts a window along epipolar lines on the right-view image to find disparities with maximum window similarity. To accelerate the exhaustive searching, local search algorithms are proposed. PatchMatch stereo [58] extends the PatchMatch algorithm [32], which assumes that there will be at least one randomly initialized disparity that is close to the accurate dispar-

ity and pixels in the same patch are likely to be located on the same surface. With these hypotheses, PatchMatch stereo can propagate these "correct" disparities to their neighbor pixels (spatially), pixels in the matched patch in other views (among different views), and pixels in the preceding and consecutive frames (temporally). Other approaches that do not require image rectification often warp images from the source views to the reference views by homography matrices, aggregate errors between corresponding pixels for each depth hypothesis, and query the best depth with best *photoconsistency* [7] according to the computed cost volume for each pixel. Plane sweeping [33, 59, 60] performs such query in a list of discrete depths. Shen [55] and Zheng [35] both combine the advantages of plane sweeping and PatchMatch to design high-efficiency PatchMatch stereo algorithms without image rectification.

Deep-Learning based MVS (DL-MVS). The concept of MVSNet is first proposed by Yao et al. [36] in 2018. MVSNet applies the differentiable homography on warping multi-scale features to calculate the cost volume and uses a 3D-CNN to regularize the encoded cost volume to be the depth probability volume. Since Yao's MVSNet [36] proves to have great performance on both accuracy and completeness, a series of related research works have been starting and made great progress. Recurrent MVSNet [37], Point MVSNet [39], Fast MVSNet [38], CasMVSNet [51] are some of the most famous models based on the concept of MVSNet. These latest DL-MVS methods are influenced heavily by the plane sweeping algorithm mentioned. PatchmatchNet [2], however, follows and improves a set of PatchMatch stereo algorithms [58, 55, 35] to cut down the size of cost volume and breaks the shape constraint of a patch. In addition to fast parallel computing in neural layers, deep learning methods also benefit from image feature pyramid [61] so that the *photoconsistency* computation will not only be based on the pixel colors but also the extracted multi-scale features, which help improve the robustness.

Point-Cloud Compression and Simplification. There have been lots of works on compressing/simplifying the dense point cloud. In many of them, a tree-like structure will

be used to encode the spatial structure of the origin dense point cloud [62, 63]. Another idea is to downsample the original point cloud to obtain a subset or averaged point cloud with smaller quantities of points. Uniform voxel sampling is a basic and widely used downsampling technique, which returns the average coordinates of all points in every voxel grid. The voxel size is a customized parameter provided by the users. Popular libraries like Open3D [64] and Point Cloud Library (PCL) [65] all have integrated the uniform voxel downsampling. Furthest Point Sampling (FPS) is another popular downsampling algorithm that selects the furthest point from the current central every step. It is much slower so researchers have been working hard to make a fast approximation of it [66, 67, 68]. With the development of neural networks and the improvement of computation power of generally-used hardware, deep learning models like [69, 70] can provide great performance on cleaning and simplifying the point clouds.

Surface Reconstruction. Three of the most widely used techniques to estimate normals and reconstruct surfaces are the marching cubes (MC) algorithm [29], the ball pivoting algorithm (BPA) [28] and the Poisson surface reconstruction method [27]. MC builds polygonal meshes by decide how a surface intersects its logical cube created from eight neighboring pixels. BPA is a surface reconstruction method that is related to alpha shapes. Suppose there is a 3D ball with a given radius dropped on the point cloud. If it hits any 3 points and it does not fall through those 3 points, a triangle will be created. Then, the algorithm starts pivoting from the edges of the existing triangles, and every time it hits 3 points where the ball does not fall through we create another triangle. Poisson surface reconstruction can be preferable because it produces smooth results. Recently, deep learning methods like Multi-View Stereo Distance Field (MVSDF) [48] are proposed. MVSDF optimizes the implicit surface representation (encoded as a neural network) by reconstructing the signed distance field (SDF) and the surface light field, which corresponds to the reconstruction of surfaces and texture / materials separately.

2.3 Survey on DL-MVS approaches

In this section, we will first briefly go over two popular non-DL MVS approaches: plane sweeping and PatchMatch stereo. Then we will focus on comparing DL-MVS approaches based on these two approaches by both time and space complexity analysis and experiment performances.

2.3.1 Plane Sweeping Stereo

The projective transformation maps 3D world point ${}^{w}\tilde{\mathbf{X}} = [X, Y, Z, 1]^T$ into screen coordinates ${}^{s}\tilde{\mathbf{X}} = [x, y, 1, 1/z]^T$, where z is the depth to the image plane. In two view triangulation, $1/z$ is related with *disparity* d [71]:

$$d = BF\frac{1}{z} \tag{2.1}$$

where B is the baseline and F is the focal length.

The 4×4 invertible projection matrix $\tilde{\mathbf{P}}$ us defined as:

$$\tilde{\mathbf{P}} = \begin{bmatrix} \mathbf{K} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{2.2}$$

where $\mathbf{K}$ is the 3×3 camera calibration matrix (intrinsic), and $(\mathbf{R}, \mathbf{t})$ is camera extrinsic. Then for a pixel of homogeneous coordinates ${}^{s}\tilde{\mathbf{x}} = [u, v, 1]^T$, if the depth hypothesis is z, its corresponding screen coordinates is ${}^{s}\tilde{\mathbf{X}} = [u, v, 1, 1/z]^T$, its world coordinates ${}^{w}\tilde{\mathbf{X}}$ can be calculated by:

$${}^{w}\tilde{\mathbf{X}} = \tilde{\mathbf{P}}^{-1}\,{}^{s}\tilde{\mathbf{X}} \tag{2.3}$$

For a scene with k cameras, the projection matrix of camera i (c_i) is denoted as $\tilde{\mathbf{P}}_i$. Assume c_0 is the reference view and the other $(k-1)$ cameras are source views. To estimate the depth z for ${}^{s}\tilde{\mathbf{X}}$, plane sweep algorithm first generate a list of depth hypotheses $D = \{z_1, z_2, ..., z_n\}$. Under each depth hypothesis z_j, the corresponding screen coordinates

${}^s\tilde{\mathbf{X}}_{j0} = [x, y, 1, 1/z_j]^T$ is assumed to be on the fronto-parallel plane at depth z_j, and can be re-projected to source view c_i by:

$$ {}^s\tilde{\mathbf{X}}_{ji} = \tilde{\mathbf{P}}_j \tilde{\mathbf{P}}_0^{-1} {}^s\tilde{\mathbf{X}}_{j0} \tag{2.4} $$

where both ${}^s\tilde{\mathbf{X}}_{ji}$ and ${}^s\tilde{\mathbf{X}}_{j0}$ are homogeneous screen coordinates and can be transferred to corresponding pixel coordinate (u, v).

Then we can estimate the depth of ${}^s\tilde{\mathbf{X}}_0$ by querying the depth hypotheses list:

$$ \hat{z}({}^s\tilde{\mathbf{X}}_0) = \text{argmin}_{z_j \in D} \sum_{i=1}^{k} \rho(I_0({}^s\tilde{\mathbf{X}}_{j0}) - I_i({}^s\tilde{\mathbf{X}}_{ji})) \tag{2.5} $$

where $I_i({}^s\tilde{\mathbf{X}}_{ji})$ stands for the color of pixel ${}^s\tilde{\mathbf{X}}_{ji}$ in image captured by camera c_i. ρ stands for a customized error aggregation function. If we unfold the $I_i({}^s\tilde{\mathbf{X}}_{ji})$, we can find pixels of all warped images can be stacked into a volume $I(x, y, i, j)$ indexed by pixel (x, y), depth hypothesis z_j and camera c_i. The error volume calculated by ρ is called cost volume C.

We can also use homography to better interpret the plane sweeping stereo. The homography $\tilde{\mathbf{H}}_{i_1 i_2}$ to map 2D pixel coordinates from view i_1 to view i_2 is a 3×3 matrix. Annotate the depth hypothesis as z_j for the reference view, the homogeneous pixel coordinates on reference camera c_0 as $\tilde{\mathbf{x}}_{j0}$, and the corresponding pixel coordinates on the source camera c_i as $\tilde{\mathbf{x}}_{ji}$. From Equation 2.2 and Equation 2.4, we get

$$ \begin{aligned} \tilde{\mathbf{x}}_{ji} &\sim \tilde{\mathbf{P}}_j \tilde{\mathbf{P}}_0^{-1} {}^s\tilde{\mathbf{X}}_{j0} \\ &= \begin{bmatrix} \mathbf{K_i} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R_i} & \mathbf{t_i} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R_0^{-1}} & -\mathbf{R_0^{-1}}\mathbf{t_0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{K_0^{-1}} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_{j0} \\ 1/z_j \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{K_i}\mathbf{R_i}\mathbf{R_0}^{-1}\mathbf{K_0}^{-1}\tilde{\mathbf{x}}_{j0} - \frac{1}{z_j}\mathbf{K_i}\mathbf{R_i}(\mathbf{R_0}^{-1}\mathbf{t_0} - \mathbf{R_i}^{-1}\mathbf{t_i}) \\ 1/z_j \end{bmatrix} \end{aligned} \tag{2.6} $$

where $\sim$ indicates equality up to scale.

As the distance between ${}^s\tilde{\mathbf{X}}_{j0}$ to the reference image plane is z_j, the vector from camera c_0's center to ${}^s\tilde{\mathbf{X}}_{j0}$ is $z_j{\mathbf{K_0}}^{-1}\tilde{\mathbf{x}}_{j0}$. Assume the plane f_p where ${}^s\tilde{\mathbf{X}}_{j0}$ is located has the normal $\mathbf{n}$, because f_p is a fronto-parallel plane to c_0, the image plane is parallel to f_p. So the projection of $z_j{\mathbf{K_0}}^{-1}\tilde{\mathbf{x}}_{j0}$ on $\mathbf{n}$, which represents the distance from c_0's center to plane f_p, is also equal to z_j.

$$\begin{aligned} z_j &= z_j{\mathbf{K_0}}^{-1}\tilde{\mathbf{x}}_{j0} \cdot \mathbf{n} \\ 1 &= \mathbf{n}^T{\mathbf{K_0}}^{-1}\tilde{\mathbf{x}}_{j0} \end{aligned} \tag{2.7}$$

Combine Equation 2.6 and Equation 2.7,

$$\begin{aligned} \tilde{\mathbf{x}}_{ji} &\sim \begin{bmatrix} \mathbf{K_i}\mathbf{R_i}{\mathbf{R_0}}^{-1}{\mathbf{K_0}}^{-1}\tilde{\mathbf{x}}_{j0} - d_j\mathbf{K_i}\mathbf{R_i}({\mathbf{R_0}}^{-1}\mathbf{t_0} - {\mathbf{R_i}}^{-1}\mathbf{t_i})\mathbf{n}^T{\mathbf{K_0}}^{-1}\tilde{\mathbf{x}}_{j0} \\ 1/z_j \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{K_i}\mathbf{R_i}(I - \frac{1}{z_j}({\mathbf{R_0}}^{-1}\mathbf{t_0} - {\mathbf{R_i}}^{-1}\mathbf{t_i})\mathbf{n}^T){\mathbf{R_0}}^{-1}{\mathbf{K_0}}^{-1}\tilde{\mathbf{x}}_{j0} \\ 1/z_j \end{bmatrix} \end{aligned} \tag{2.8}$$

Therefore, the homography of pixel coordinates from c_0 to c_i with depth hypothesis z_j is:

$$\tilde{\mathbf{H}}_{0i}(z_j) = \mathbf{K_i}\mathbf{R_i}(I - \frac{1}{z_j}({\mathbf{R_0}}^{-1}\mathbf{t_0} - {\mathbf{R_i}}^{-1}\mathbf{t_i})\mathbf{n}^T){\mathbf{R_0}}^{-1}{\mathbf{K_0}}^{-1} \tag{2.9}$$

2.3.2 DL-MVS by Plane Sweeping

Since MVSNet [36] started a new era for deep learning MVS approaches, plane sweeping has been one of the most popular support theorem that lie behind the success of DL-MVS. In this section, we will take MVSNet as an example to show how plane sweeping stereo contributes to modern deep learning MVS architectures.

As Figure 2.1 show, there are four steps in the architecture of MVSNet: feature extraction, differentiable homography, Cost volume regularization, and depth map refinement. We can consider the first three steps as an extension to the original plane sweeping stereo.

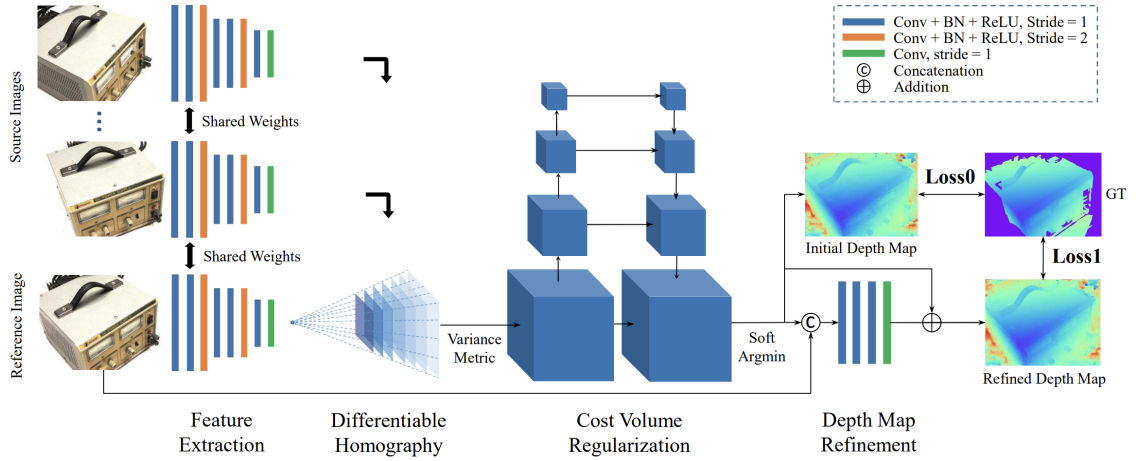

Figure 2.1: Architecture of MVSNet [36]. For each inference, multi-scale features are extracted from input images through 2D feature extraction network. Features of source images will be warped to the reference image by differentiable homography to build cost volume. Finally, the cost volume will be regularized to probability volume and generate depth map for the reference image. The depth map is refined with the reference image.

1. *Feature extraction:* Deep convolution neural networks (Deep CNNs) are well-known for its multi-scale feature extraction capability. Compared with only using color dissimilarity to calculate the cost volume, using multi-scale features extracted and encoded by trained CNNs can be more informative and robust. Besides, as the extracted features are downsized, it significantly boosts the reconstruction both in efficiency and quality.

2. *Differentiable homography:* This step is very similar to the warping process of the plane sweeping stereo (Equation 2.9). The only difference is to make the homography differentiable, where bilinear interpolation on extracted feature maps is used when warping from source images to the reference views.

3. *Cost Volume Regularization:* Instead of directly find the disparity with the minimum dissimilarity, a multi-scale 3D CNN is used to aggregate neighboring information [36] from the cost volume. The 3D CNN then decodes the depth propability volume to produce the initial depth map.

4. *Depth Map Refinement:* To deal with over-smoothing caused by cost volume regularization, a depth residual network is applied to remind the depth map of the input image colors.

Similar to MVSNet, many other DL-MVS approaches also compute the cost volume in the same way based on plane sweeping stereo. They also bring novel designs to make this basic architecture more powerful. R-MVSNet [37] uses recurrent neural network units (RNN units) to avoid caching the large cost volume. For the same purpose, Fast-MVSNet [38] first computes sparse cost volume and then uses propagation on the output sparse depth map to obtain dense depth map. Point-MVSNet [39] and CasMVSNet [51] both adopt the coarse-to-fine concept.

2.3.3 PatchMatch Stereo

PatchMatch [32] is originally proposed as a randomized correspondence algorithm for structural image editing. A patch is a region of pixels on the image. The algorithm first assigns each patch a random correspondence. Then for each patch, check if its neighbors' correspondences are better matches. If so, propagate the best fit correspondence to the current patch. Similar to many local search algorithm, PatchMatch also add each patch random perturbations to enable random search for improvements in concentric neighborhoods [32]. In this section, we will introduce how PatchMatch algorithm is imported in traditional MVS approaches.

PatchMatch Stereo with Rectified Image Pairs

Traditional MVS algorithms for rectified images like block-matching [57] search exhaustively for the best disparity. To make it faster, PatchMatch algorithm is imported to replace exhaustive searching with random local searching. PatchMatch stereo [58] is such a fast MVS algorithm. Instead of building the cost volume, PatchMatch stereo queries the best disparity by propagation. PatchMatch Stereo first assigns each pixel on each frame a ran-

dom disparity d and a random normal $\mathbf{n}$. Together with baseline B and focal length F, we can calculate the plane f_p where ${}^s\tilde{\mathbf{X}} = [x, y, 1, 1/z]^T$ located.

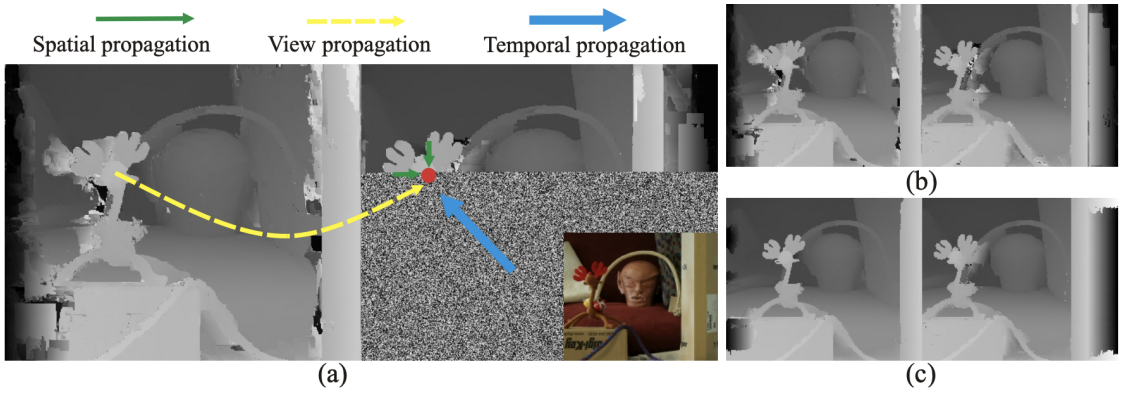

Figure 2.2: Different steps of the PatchMatch stereo [58]: (a) Three types of propagation. (b) Results at the end of propagation. (c) Results after post-processing.

The PatchMatch Stereo algorithm holds the assumption that world points of all pixels within one patch lie on the same plane as the point at the center of the patch does. For each iteration, the algorithm will traverse all patches in certain orders. Assume p is the center pixel of the current patch and its current plane is f_p. The propagation follows (Figure 2.2):

1. **Spatial propagation**: Assign p to its neighbor q's current plane f_q if it decreases the cost of the patch.

2. **View propagation**: As for p's matched point in other views p', assign p to p's current plane $f_{p'}$ if it decreases the cost of the patch.

3. **Temporal propagation**: As for p's same-location point on its preceding/consecutive image p'', assign p to p's current plane $f_{p'}$ if it decreases the cost of the patch.

Different from plane sweeping, the cost here is calculated by a spatial aggregation:

$$m(p, f) = \sum_{q \in W_p} w(p, q) \cdot \rho(q, q - d_{qf}) \tag{2.10}$$

where W_p is denotes a 2D square window centered on pixel p (will be 3D if considering

temporal stereo). $w(p, q)$ is a weight function can be customized for certain kind of purposes. d_{qf} shows the disparity of q in the image pair if it is on plane f. Hence $q - d_{qf}$ denotes the corresponding point of q in the other image. ρ is the error function to calculate dissimilarity.

The propagation works under the assumption that after random initialization there will be at least one pixel of the region, of which the plane is close to the correct one [58]. In this way, the random initialization builds a pool of disparity hypotheses to take the place of the discrete disparity list used in plane sweeping. After propagation, Plane refinement and post-processing will be applied to optimize the disparity map.

PatchMatch Stereo without Rectification

Because image rectification algorithms relies on the accuracy of matched feature point and can only be calculated between two images at a time, MVS methods that require pre-rectifying and then matching can be inefficient when the dataset is large. Therefore, there are many improved PatchMatch MVS algorithms [55, 35] that combines the advantages of PatchMatch algorithm and the plane sweeping algorithm. To remove pre-rectifying, these methods warp source images to the reference image using Equation 2.9 to query depth hypotheses instead of disparities.

Compared with simple plane sweeping stereo, PatchMatch Stereo algorithms without rectification have the following advantages:

1. Use randomized depth map as random depth pool instead of creating a list of depth hypotheses for each pixel. Avoid caching a large cost volume.

2. Patch-based error functions (often Normalized Cross Correlation, NCC) can be more robust on noise, images under different illuminance, and so on.

3. Improved PatchMatch Stereo algorithm such as Zheng's General Expectation Maximization (GEM) algorithm can be further parallelized along rows/columns.

2.3.4 DL-MVS by PatchMatch Stereo

PatchMatch Stereo without rectification are widely used in the existing SFM pipelines, such as COLMAP [23] and OpenMVS [19], there is seldom DL-MVS approach implemented based on it. PatchmatchNet [2] is the most popular and successful one of such methods. In this section we will focus on introducing PatchmatchNet.

Traditional PatchMatch stereo workflow (See subsection 2.3.3) often includes *random initialization*, *propagation*, *evaluation*, and *refinement & post-processing*. There will be a small number of iterations covering *propagation* and *evaluation* until the depth map converges. PatchmatchNet follows these steps and extends the original algorithm to 1) run faster, 2) support multiple source views, 3) break the patch shape limitation, and 4) fit with the deep neural network framework (See Figure 2.3):

1. *Initialization:* In PatchMatch stereo, each pixel is assigned a single random depth hypothesis. Patchmatchnet uniformly samples the complete disparity space to assign each pixel a list of initial hypotheses to further guarantee that close guessed hypotheses will be propagated.

2. *Adaptive Propagation:* Origin PatchMatch stereo adopts a scanning-order (from top-left pixel to bottom-right pixel) propagation, which is time-consuming. To make use of the parallel characteristic of neural network, PatchmatchNet uses a deformable convolutional network (DCN) [72] to learn to adaptively select proper pixels' depth as candidate hypotheses for the current pixel. This adjustment not only significantly accelerates the propagation step, but also breaks the shape constraint of a patch, which is usually a fixed square.

3. *Adaptive Evaluation:* Like other DL-MVS, PatchmatchNet adopts the concept of differentiable homography to warp the source image feature into reference view. This invention addresses the constraint that PatchMatch stereo can only work on a pair of images. Instead of directly calculate the cost volume, PatchmetchNet is inspired

by group-wise correlation [73] to group the features and use the inner product of grouped feature vectors to calculate a similarity volume for the selected depth hypothesis. Extend to the pixel-wise weight $w(p, q)$ in the original PatchMatch stereo, PatchmatchNet uses a pixel-wise view weight [23, 74] network to evaluate similarity confidence of each source view, and aggregates each view's similarity volume by applying weighted sum.

4. *Adaptive Cost Spacial Aggregation:* PatchmatchNet transforms the weighted-sum (by source views) similarity volume to a cost volume by a 3D-CNN. Then similar to Equation 2.10, it spatially aggregates the cost but makes two changes: 1) adaptively select W_p by learned network so that W_p will not simply be a 2D square window, 2) compute the weight $w(p, q)$ by feature and depth similarity.

Besides, PatchmatchNet also applied coarse-to-fine concept to improve depth map quality.

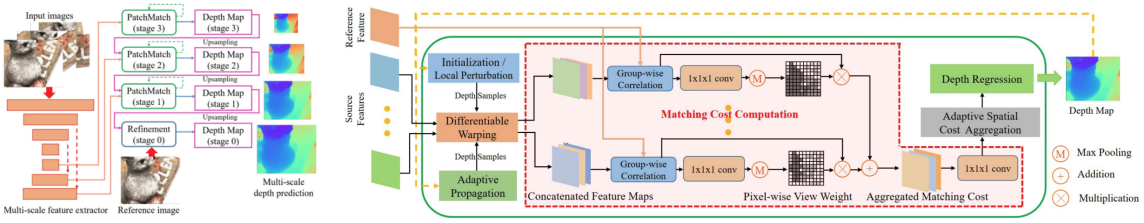

Figure 2.3: Left: Overall architecture of Patchmatch Net. Right: Detailed structure of learned Patchmatch: In the first iteration, depth hypotheses in *Initialization* are used. In the following iterations, depth hypotheses are obtained from *Adaptive Propagation* by learned offsets. *Local Perturbation* adds randomness to the hypotheses to help with the local search. The learned pixel-wise view weight is estimated in the first iteration of Patchmatch [2].

2.3.5 Complexity Analysis

The time and space complexity analyses are performed under the following assumptions:

- All frameworks run with same number of images with the same resolution.
- All neural networks are fully parallelized by the same device.

For all DL-MVS we tested, multi-scale feature extraction is a common step. Usually the feature extraction will have three stages, downsampling the original resolution. From Table 2.1 we can learn that in terms of the number of convolutional layers, MVSNet, R-MVSNet, and Fast-MVSNet will be faster than the others. CasMVSNet and PatchmatchNet apply the feature pyramid [61] so there will be extra convolutional layers for upsampling. For the number of output feature channels, Point-MVSNet and PatchmatchNet request more features. For the edge scales, all models use a 3-stage multi-scale feature, and MVSNet and R-MVSNet require the larger resolutions.

Table 2.1: DL-MVS Complexity Analysis: Feature Extraction

Approaches	**#(Conv layers)**	**#(Features)**	**Edge scale**
MVSNet	8	$[8, 16, 32]$	$[1, 1/2, 1/4]$
R-MVSNet	8	$[8, 16, 32]$	$[1, 1/2, 1/4]$
Point-MVSNet	11	$[16, 32, 64]$	$[1/2, 1/4, 1/8]$
Fast-MVSNet	8	$[8, 16, 32]$	$[1/2, 1/4, 1/8]$
CasMVSNet	13	$[8, 16, 32]$	$[1, 1/4, 1/16]$
PatchmatchNet	16	$[16, 32, 64]$	$[1/2, 1/4, 1/8]$

Cost volume (similarity volume for PatchmatchNet) is another key module that all included methods have. MVSNet, Point-MVSNet and Fast-MVSNet use the cost volume in the traditional plane sweeping way. To deal with MVSNet's memory inefficiency, Point-MVSNet builds a low-resolution cost volume and then uses point flow module to predict and interpolate the depth map. Fast-MVSNet builds a sparse cost volume and then propagates sparse depth maps to dense depth maps. R-MVSNet uses RNN to compute a cost map for one hypothesis a time, which significantly improves the memory efficiency but also brings with a lot of time consumption. CasMVSNet and PatchmatchNet both use a cascading structure, in which there are several stages. In each stage, features generated by the feature pyramid at the corresponding scale and coarse depth map from the previous stage will be used to build a cost volume and predict for a new depth map with larger resolution. Because stages with larger resolutions serve as refinement to previous stages, there will be fewer hypotheses and features required. In this way, the trade-off between

time and space complexity can be well addressed. PatchmatchNet also benefits from fewer feature channels by group-wise correction. Table 2.2 shows more details about the largest cost volume (in volume size) in each methods.

Table 2.2: DL-MVS Complexity Analysis: Largest Cost Volume

Approaches	**Edge scale**	**#(Hypotheses)**	**#(Features)**	**Volume Size**
MVSNet	$1/4$	256	32	$512 \times H \times W$
R-MVSNet	$1/4$	1	32	$2 \times H \times W$
Point-MVSNet	$1/8$	96	64	$96 \times H \times W$
Fast-MVSNet	$1/4$ (sparse)	96	32	$48 \times H \times W$
CasMVSNet	1	8	8	$64 \times H \times W$
PatchmatchNet	$1/2$	16	4	$16 \times H \times W$

In additional to this two stages, we also list modules for each architecture that could be the bottleneck for either time or space complexity (Table 2.3). One of the structures that most of these methods have is 3D-CNN, which is used to transform the cost volume to the depth probability volume. The time and space complexity of 3D-CNN can be influenced by the input resolution, kernel size, and the architecture. For example, MVSNet uses a four-scale 3D-CNN with $3 \times 3 \times 3$ kernels and a large-resolution cost volume, which can cost more time and memory than PatchmatchNet, which uses a simple sequential 3D-CNN with $1 \times 1 \times 1$ kernels and a much smaller cost volume.

Table 2.3: DL-MVS Complexity Analysis: Other Modules

Approaches	**Module name**	**Major concerns**
MVSNet	Multi-scale 3D-CNN	time & space
R-MVSNet	RNN	time
Point-MVSNet	Flow Iterations	time
Fast-MVSNet	-	-
CasMVSNet	Cascade Iterations	time
PatchmatchNet	Propagation Iterations	time

In conclusion, by analyzing time and space complexity into consideration, we select Fast-MVSNet, CasMVSNet and PatchmatchNet to be candidates for GTSFM. Next we will presents data from real experiments to take performance into consideration.

2.3.6 Performance Analysis via Experiments

For MVS algorithms, DTU dataset [4], tanks and temples dataset [52], and ETH3D dataset [53] are three popular datasets for benchmark. DTU dataset is even more widely used in DL-MVS studies. To evaluate quality of the reconstruction results, there are three acknowledged metrics [75, 52] for ETH3D dataset [53]:

- *Accuracy (%):* The fraction of the reconstruction which is closer to the ground truth than the evaluation threshold distance.
- *Completeness (%):* The fraction of the ground truth which is closer to the reconstruction than the evaluation threshold distance.
- F_1 *score (%):* The harmonic mean of accuracy and completeness, used to rank methods based on both metrics.

To make the experiments fair, we evaluate all these DL-MVS approaches together with COLMAP [23]'s MVS [35] on 5 different sample scans from DTU test dataset. All the experiments ran on a personal laptop with Ubuntu 20.04.1, Intel(R) Core(TM) i7-8750H CPU (16GB RAM), and Nvidia GeForce GTX 1070 Mobile GPU (8GB RAM). The run-time here is the inference time in seconds per image loop. To make the comparison fair, we make all MVS methods use the same SFM results generated by COLMAP. COLMAP, R-MVSNet, CasMVSNet, and PatchmatchNet can run full resolution (1600×1200), while Fast-MVSNet can only run in (1024×768) and MVSNet in (768×576). This result proves the complexity analysis above.

Table 2.4: DL-MVS Performance Analysis: DTU dataset

Approaches	**Acc.(%)** $\uparrow$	**Comp.(%)** $\uparrow$	F_1 **score(%)** $\uparrow$	**Run-time(s)** $\downarrow$	**GPU Mem.(MB)** $\downarrow$
COLMAP-MVS	80.67	59.42	68.44	6.475	1520
MVSNet	47.41	57.73	52.07	1.126	8081
R-MVSNet	31.39	32.02	31.70	5.450	7577
Fast-MVSNet	22.06	48.59	30.33	3.051	7876
CasMVSNet	75.51	66.37	70.64	1.576	7289
PatchmatchNet	79.15	65.68	71.79	1.066	6093

The experiment results show DL-MVS approaches run faster and result in better quality than the state-of-art non DL-MVS approach implemented by COLMAP. The run-time and GPU memory results also match with the complexity analyses. Among all included DL-MVS approaches, PatchmatchNet have the both relatively high running speed and reconstruction quality. Therefore, we decided to integrate PatchmatchNet into our SFM pipeline, GTSFM.

2.4 Approach: Dense Multi-view Optimizer for GTSFM

In this section, we will in detail introduce how we design and implement our dense multi-view optimizer framework for GTSFM.

2.4.1 System Overview

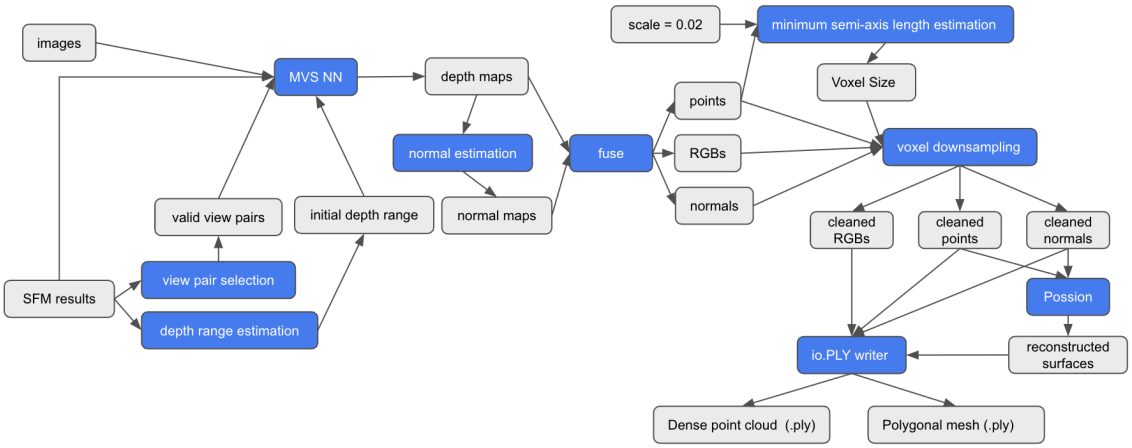

Figure 2.4: Dense Multi-view Optimizer for GTSFM

Multi-view Stereo (MVS) computes a complete object model (both geometry and texture) from a collection of images from multiple calibrated cameras. Since SFM pipelines have already calibrated valid cameras and built sparse correspondences in filtered tracks, modulated MVS algorithms in SFM pipelines need to perform the following tasks:

1. **Depth range estimation**: Estimate depth (or disparity [71]) ranges for each view by

analysing the tracks and the sparse correspondences.

2. **View pair selection**: For each view, evaluate all other views to get $N-1$ candidate views that are proper to make image pairs with the current view.

3. **MVS NN**: Establish dense correspondences by calculating and optimizing the depth maps for each view with deep neural networks.

4. **Depth Map Filtering**: Filter the output depth maps by geometry threshold and photoconsistency threshold to get valid masks.

5. **Normal estimation**: Estimate normals for each pixel from depth maps.

6. **Fuse**: Filter the depth map to get a mask for reasonable pixels and project the corresponding points for these filtered pixels to the world frame. Fuse all 3D points from valid views in the world frame to build a dense point cloud.

7. **Voxel downsampling**: Clean up the dense point cloud and remove redundancy.

8. **Poisson**: Surface reconstruction from point coordinates and normals.

9. **PLY writer**: Write both fused dense point cloud and mesh into output files.

Figure 2.4 shows how these tasks get input and output data and collaborate with each other, working as a dense multi-view optimizer module in GTSFM architecture **??**. Here the blue boxes denote processes and the gray boxes denote the input / output variables.

This framework is specially designed for DL-MVS approaches, where the **MVS NN** module can be replaced with inference neural network $\mathcal{F}$ of any other DL-MVS approach that obeys the following formula:

$$\hat{\mathbf{D}} = \mathcal{F}(\mathbf{I}, \mathbf{\Phi}, \mathcal{M}) \tag{2.11}$$

where input $\mathbf{I}$ is the image vector and $\mathbf{\Phi}$ is the camera parameter vector. $\mathcal{M}$ is a $|I| \times (N-1)$ matrix, the i-th row of which represents the $(N-1)$ selected source views when camera

c_i is the reference view. The output $\hat{\mathbf{D}}$ is the predicted depth maps for each view. With this framework, new DL-MVS approaches with better performance can be easily integrated into GTSFM and can perform complete MVS tasks without further parameter tuning.

We divide the work graph of the deep multi-view optimizer into three consecutive parts: pre-processing, inference, and post-processing. In the following three subsections, we will introduce how we implement each part in detail.

2.4.2 Pre-processing

The pre-processing part consists of depth range estimation and view pair selection. It is an important step to parse SFM results from the output of bundle adjustment module.

Since all tested DL-MVS methods uses plane sweeping to propose and evaluate depth hypotheses, a reasonable depth range for each view should be estimated for initializing coarse hypotheses. A good depth range should let the foreground object(s) completely lie between the nearest (minimum depth) and the farthest (maximum depth) planes, so measurements in the tracks can be great support evidences to estimate it. For track t_j with measurement m_k on image captured by camera c_i, we calculate its depth d_{ki} from the camera by transforming track t_j's sparse correspondence point s_j into c_i's frame and collect it as a support depth for c_i. After traversing all tracks, for each view, we compute the 1st percentile of all its support depths as the minimum depth, and the 99th percentile as the maximum depth to filter out background and outliers.

As plane sweeping enables handling multiple source views by Equation 2.9, for DL-MVS approaches, we need to find a set of source views for each valid view as the reference view, which is called view selection [36]. The evaluation method was inspired by the function proposed by Furukawa1 et al. [76]. The function is designed to measure the expected reconstruction accuracy of a 3D point P from a set of images I, and depends on the camera baselines and pixel sampling rates. The view selection algorithm used by most DL-MVS approaches included focus on the first factor, camera baselines. In two view

triangulation, for world point p with depth z for the reference camera, there is a relationship baseline b, ϵ_z is the depth error, ϵ_d is the matching error in pixels, and f is the focal length in pixels [77]:

$$\epsilon_z = \frac{z^2}{bf} \cdot \epsilon_d \tag{2.12}$$

Hence, with a narrower baseline, the same matching errors will cause much larger depth errors. It is also mentioned in [76] that views with close baselines will yield a noisy reconstruction. Meanwhile, if the baseline is too large but the depth is small, even a large matching error will result in a small depth error. Therefore, a Gaussian function that favors a certain baseline angle θ_0 is applied [76, 78]:

$$g(\theta_P) = \exp(-\frac{(\theta_P - \theta_0)^2}{2\sigma_{\theta_P}^2}) \tag{2.13}$$

where θ_P is the baseline angle between two viewing rays emanating from the world point P in a common track towards the two camera centers [76]. We use the settings in MVSNet where $\theta_0 = 5°$, σ_θ is 1 when $\theta < \theta_0$ and is 10 otherwise. Then for every two view pair (I_i, I_j), the score will be calculated by aggregating all points in shared tracks.

$$S(I_i, I_j) = \sum_P g(\theta_P) \tag{2.14}$$

After calculating the score matrix for all views, for each view as the reference view, select top $(N-1)$ views with highest scores to be source views.

2.4.3 Inference with PatchmatchNet

Different from other parts, this part should be implemented according to different DL-MVS methods, including how to pack batched data $(\mathbf{I}, \mathbf{\Phi}, \mathcal{M})$ and how to infer with the neural network. In this part, we apply the same way as original PatchmatchNet [2] program does.

2.4.4 Post-processing

For DL-MVS methods fit with Equation 2.11, the output results will be a depth map for each view as the reference view. The post-processing part starts from fusing 3D world points from these depth maps and ends by writing the refined dense point cloud and reconstructed polygonal mesh into output files.

Depth Map Filtering

After inferring DL-MVS methods, the output results will be predicted depth maps for each valid view. To evaluate the predicted depth maps to get masks for valid depths, there are often two kinds of thresholds for DL-MVS methods: geometry threshold and photoconsistency threshold.

The geometry threshold τ_{geo} focuses on evaluating re-projection errors. For depth map for camera c_{i_1}, the predicted depth for pixel $\tilde{\mathbf{x}}_{i_1}$ is $\hat{z}_{i_1}$. Its re-projection error towards camera c_{i_2} can be computed by:

$$\epsilon_{rproj}(\tilde{\mathbf{x}}_{i_1}, i_1, i_2) = ||\mathbf{K_{i_1}}(\mathbf{R_{i_1}}\mathbf{R_{i_2}^T}(\hat{z}_{i_2}\mathbf{K_{i_2}^{-1}}\tilde{\mathbf{x}}_{i_2} - \mathbf{t_{i_2}}) + \mathbf{t_{i_1}}) - \tilde{\mathbf{x}}_{i_1}||_2 \tag{2.15}$$

where $\tilde{\mathbf{x}}_{i_2}$ is the projected screen point in view i_2 and $\hat{z}_{i_2}$ is the bilinear interpolated depth for $\tilde{\mathbf{x}}_{i_2}$ from the predicted depth map of view i_2.

$$\tilde{\mathbf{x}}_{i_2} \sim \mathbf{K_{i_2}}(\mathbf{R_{i_2}}\mathbf{R_{i_1}^T}(\hat{z}_{i_1}\mathbf{K_{i_1}^{-1}}\tilde{\mathbf{x}}_{i_1} - \mathbf{t_{i_1}}) + \mathbf{t_{i_2}}) \tag{2.16}$$

The geometry mask $\mathcal{M}_{geo}$ on depth map i_1 for pixel $\tilde{\mathbf{x}}_{i_1}$ is true if $\tau_{geo} > \epsilon_{rproj}(\tilde{\mathbf{x}}_{i_1}, i_1, i_2)$ for more than $\lfloor \frac{N-1}{2} \rfloor$ source views.

The photoconsistency threshold τ_{photo}, the photoconsistency mask $\mathcal{M}_{photo}$ on depth map i_1 for pixel $\tilde{\mathbf{x}}_{i_1}$ is true if average cost $\bar{C}(\tilde{\mathbf{x}}_{i_1}, \hat{z}_{i_1}) < \tau_{photo}$.

The final mask $\mathcal{M}_{final}$ is the joint mask considering both geometry and photoconsis-

tency thresholds, which is an element-wise *AND* operation between two masks

$$\mathcal{M}_{final}(u,v) = \mathcal{M}_{geo}(u,v)\mathcal{M}_{photo}(u,v) \tag{2.17}$$

Normal Estimation

Unlike some of the MVS algorithms that also optimize the normal map for each dense point [58, 55, 35], recent DL-MVS methods often do not estimate the normal maps. Therefore, we implement a simple normal estimation module for our deep multi-view optimizer.

Most DL-MVS methods rely on Equation 2.9 to warp the source image features, so similar to traditional plane sweeping stereo algorithms, the corresponding virtual planes for each depth hypothesis are all parallel with the reference image plane. Hence, the normal of pixel $\tilde{\mathbf{x}}_i$ in view i is:

$$\mathbf{n}_{\tilde{\mathbf{x}}_i} = \mathbf{R_i}^T[0,0,1]^T \tag{2.18}$$

which is equal to the direction of camera c_i's optical axis $\mathbf{z_i}$. However, this simple normal assignment could not smoothly interpolate slanted surfaces. To make use of valid neighbor depths and produce a smoother normal map, we design and implement a improved normal estimation algorithm. Suppose there is a 3×3 patch $\mathcal{Z}$ in depth map $\hat{\mathbf{D}}_i$ centered at pixel $\tilde{\mathbf{x}} = [u,v,1]^T$, pixel $\tilde{\mathbf{x}}' = [u+du, v+dv, 1]^T$ is a neighbor of $\tilde{\mathbf{x}}$ with offset (du, dv). The predicted depths for $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ are $\hat{\mathbf{D}}_i(u,v)$ and $\hat{\mathbf{D}}_i(u+du, v+dv)$, respectively. Assume $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ are on the same slanted surface, of which the normal can be calculated by:

$$\mathbf{n}_{\tilde{\mathbf{x}},\tilde{\mathbf{x}}'} = \mathbf{v}_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}'} \times (\mathbf{v}_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}'} \times \mathbf{z_i}) \tag{2.19}$$

where $\mathbf{v}_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}'} = K_i^{-1}[\hat{\mathbf{D}}_i(u+du, v+dv)\tilde{\mathbf{x}}' - \hat{\mathbf{D}}_i(u,v)\tilde{\mathbf{x}}]$ is the vector from $\tilde{\mathbf{x}}$ to $\tilde{\mathbf{x}}'$ in camera c_i's frame.

Considering the edge-fattening problem [79], we assign a weight $w_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}'}$ to $\mathbf{n}_{\tilde{\mathbf{x}},\tilde{\mathbf{x}}'}$ which is

similar to [58]:

$$w_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}'} = \exp(-\frac{||I_i(\tilde{\mathbf{x}}) - I_i(\tilde{\mathbf{x}}')||_1}{\gamma}) \tag{2.20}$$

where $I_i(\tilde{\mathbf{x}})$ stands for RGB color of pixel $\tilde{\mathbf{x}}$ in image I_i, γ is a user-customized parameter that is set to 10 in default.

Then the interpolated normal for point at pixel $\tilde{\mathbf{x}}$ is:

$$\mathbf{n}_{\tilde{\mathbf{x}}_i} \sim \frac{\sum_{\tilde{\mathbf{x}}' \in \mathcal{Z}} \mathcal{M}_{final}(u + du, v + dv) w'_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}} \mathbf{n}_{\tilde{\mathbf{x}},\tilde{\mathbf{x}}'}}{\sum_{\tilde{\mathbf{x}}' \in \mathcal{Z}} \mathcal{M}_{final}(u + du, v + dv) w'_{\tilde{\mathbf{x}}\tilde{\mathbf{x}}}} \tag{2.21}$$

Fuse

In this steps, pixels with valid joint mask will be transformed back to coordinates in world frame using Equation 2.3, together with corresponding RGB colors and normals.

Redundant Point Removal

For most dense point clouds fused by depth maps from multiple views, redundant points are common when more than one cameras can see the exactly same world point. These redundancies cause the result fused dense point cloud to be unnecessarily large in size. To address the issue, we propose two approaches. one is to solve a bipartite graph minimum vertex cover problem which is complete but slow. The other is based on voxel grid downsampling, which is not guaranteed to be optimal but is very fast.

Before introducing the algorithms proposed, w need to figure in which case two points appear redundant in the dense point cloud. One intuitive and feasible method is filter out reconstructed points that have too small distances between each other. However, to set up the distance threshold ϵ_{rr}, we need to first estimate the scale of the reconstructed object.

We apply to use semi-axes lengths of the minimum volume ellipsoid [80] of the output dense point cloud. Suppose there are n points in the dense point cloud $\mathbf{X}$, to fit the

minimum volume ellipsoid, we need first compute the corvariance matrix Σ:

$$\hat{\Sigma} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{X}_i - \bar{\mathbf{X}})^T(\mathbf{X}_i - \bar{\mathbf{X}}) \tag{2.22}$$

where $\mathbf{X}$ is of shape $(n, 3)$, $\mathbf{X}_i$ is the i-th point of shape $(1, 3)$. $\bar{\mathbf{X}}$ is the average coordinates:

$$\bar{\mathbf{X}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{X}_i$$

. The corvariance matrix is a 3×3 semi-positive definite matrix. Similar to Principal Component Analysis (PCA) [81], we calculate the eigenvalues $\mathbf{v} = [a^2, b^2, c^2]^T$ of $\hat{\Sigma}$. Each eigenvalue represents a squared semi-axis length. With the scale of the dense point cloud computed, we set the distance threshold to be:

$$\epsilon_{rr} = k \cdot \min\{a, b, c\} \tag{2.23}$$

where k is set to 0.02 in default in our implementation.

1. Bipartite Graph Minimum Vertex Cover

1) Problem Description: For each image pair (i_1, i_2), $\mathcal{S}_{i_1}$ and $\mathcal{S}_{i_2}$ are set of world points fused from depth map $\hat{\mathbf{D}}_{i_1}$ and $\hat{\mathbf{D}}_{i_2}$, respectively. Create a new graph $G = (V, E)$ with vertex set as $V = \mathcal{S}_{i_1} \cup \mathcal{S}_{i_2}$. For vertex $v_{i_1} \in \mathcal{S}_{i_1}$ and $v_{i_2} \in \mathcal{S}_{i_2}$, there is an edge $(v_{i_1}, v_{i_2}) \in E$ if the Euclidean distance between v_{i_1} and v_{i_2} is smaller than threshold ϵ_{rr}. Therefore, the maximum set of vertex remain after redundant point removal will be $V - VC(G)$, where $VC(G)$ is the minimum vertex cover of graph G.

2) Algorithm: Since all edges in G are between $\mathcal{S}_{i_1}$ and $\mathcal{S}_{i_2}$, G is a bipartite graph. Hopcroft Karp algorithm [82] is an efficient approach to solve bipartite graph minimum vertex cover problem in $O(|E|\sqrt{|V|})$ time in the worst case, $O(|E|\log(|V|))$ in random sparse graph in high probabilities.

2. Voxel Downsampling

From another point of view, if we set the minimal voxel size as ϵ_{rr} and simply perform the voxel grid downsampling across the dense point cloud volume, there will be three cases for a single voxel:

1) If there is no point in the voxel, the voxel will remain empty after downsampling.

2) If there is only one point in the voxel, the voxel will keep the original point unchanged after downsampling.

3) If there is more than one points in the voxel, the voxel will only keep the average coordinates after downsampling.

Although voxel downsampling cannot guarantee that there will be no two points from different views that are closer than ϵ_{rr}, it solves the most cases, runs much faster and considers information from all output points.

Besides these two methods, we also test other popular methods mentioned [62, 63, 64] by experiments. To evaluate the quality of the cleaned point cloud, we adopt a Peak Signal to Noise Ratio (PSNR) metrics proposed in [62]:

$$\text{PSNR}(\mathcal{P}, \mathcal{Q}) = 20 \log_{10} \frac{d_B}{\max\{\text{RMS}(\mathcal{P}, \mathcal{Q}), \text{RMS}(\mathcal{Q}, \mathcal{P})\}} \tag{2.24}$$

where $\mathcal{P}$ is the original point cloud, $\mathcal{Q}$ is the output point cloud after redundancy removal, d_B is the diagnose voxel scale $2\sqrt{a^2 + b^2 + c^2}$, and $\text{RMS}(\mathcal{P}, \mathcal{Q})$ is the root mean square of distances to the closest neighbor in $\mathcal{Q}$ of points in $\mathcal{P}$.

Here are some of our experiments of redundant point removal on the dense point cloud from the door dataset [83] reconstructed by PatchmatchNet.

Table 2.5: Performances of redundant point removal approaches on the door dataset [83]

Approaches	**Run-time(s)** $\downarrow$	**Compression Ratio (%)** $\uparrow$	**PSNR (dB)** $\uparrow$
KD-tree + Hopcroft Karp	> 480	**371**	63.3872
Open3D[64] Voxel Downsampling	$<$ **5**	362	64.0950
PCL[65] Octree Compression + Downsampling	$<$ **5**	189	63.9650
CloudCompare OctreeCleanup	< 60	304	**65.1750**

Consider all the experiments results above, we finally decide to integrate voxel down-

sampling implemented by Open3D into GTSFM.

2.4.5 Metrics Design

To quantify the performances of the implemented deep multi-view optimizer for GTSFM, we design a set of metrics. Based on the different phases of the dense reconstruction described in subsection 2.4.1, we divide the metrics into two categories: run-time metrics and post-reconstruction metrics.

Run-time Metrics

Run-time metrics are collected and computed during deep multi-view optimizer is working.

1. Inference by PatchmatchNet:
    - Number of valid views / number of PatchmatchNet loops
    - Elapsed time per loop
2. Depth Map Filtering:
    - Geometric mask valid ratio $\bar{\mathcal{M}}_{geo}$
    - Photoconsistency mask valid ratio $\bar{\mathcal{M}}_{photo}$
    - Average re-projection error $\bar{\epsilon}_{rproj}$
3. Voxel Downsampling
    - Voxel Size ϵ_{rr}
    - Point cloud size before downsampling $|\mathcal{P}|$
    - Point cloud size after downsampling $|\mathcal{Q}|$
    - Compression ratio $r = |\mathcal{P}|/|\mathcal{Q}|$
    - PSNR (Equation 2.24)

Reconstruction Metrics

Reconstruction metrics focues on evaluating output dense point cloud given the ground truth (scanned volume, etc.). We follows the *Accuracy*, *Completeness*, *Overall* metrics used in analyzing DTU dataset (See subsection 2.3.6). Before compute the matrics, iterative closest point (ICP) algorithm [84] should be run first to align the output point cloud with the ground truth point cloud.

2.4.6 Scalability

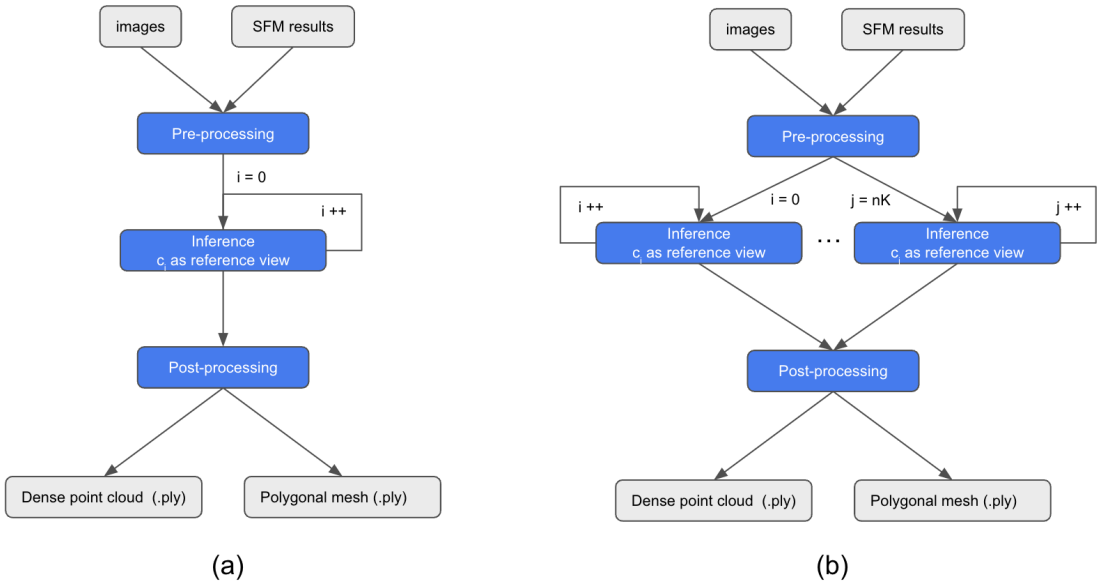

Figure 2.5: Scalability of Dense Multi-view Optimizer for GTSFM. (a) Regular optimizer. (b) Distributed optimizer.

With regular dense multi-view optimizer (Figure 2.5 (a)), if there are $|I|$ valid images, there will be $|I|$ PatchmatchNet inferences, which will be a burden when the dataset size is large. Hence we propose a distributed version of depth multi-view optimizer (Figure 2.5 (b)), for GTSFM to deal with extremely large datasets with a cluster of computing devices. For example, if there are n devices, then each device only need to do $K = \lfloor |I|/n \rfloor$ inferences simultaneously and then fuse all predicted depth maps together. To fit with the design of GTSFM, we also use Dask [85] to implement such parallelization.

2.5 Experiment Results

2.5.1 A Complete MVS Results of Door dataset [83]

In this experiment, we demonstrate a complete MVS result reconstructed by the implemented deep multi-view optimizer modulated in the GTSFM pipeline. We will use Lund Door dataset [83] as an example SFM dataset. The dataset has 12 images from different views.

Inputs. The input data will be the image set and the SFM results including calibrated camera parameters and sparse correspondences.

Figure 2.6: 12 observed images from the Lund Door dataset [83].

Depth maps output by Inferring PatchmatchNet. Here we show some of the predicted depth maps and its corresponding images. The right image is the fused point cloud.

Post-processing Results. In post-processing, we estimate the normal distribution, then use Poisson [27] to reconstruct the surfaces.

Metrics HTML. All the metrics will be output to a generated HTML file with metrics from other processes. The diagrams are implemented by PyPlot, which are interactive.

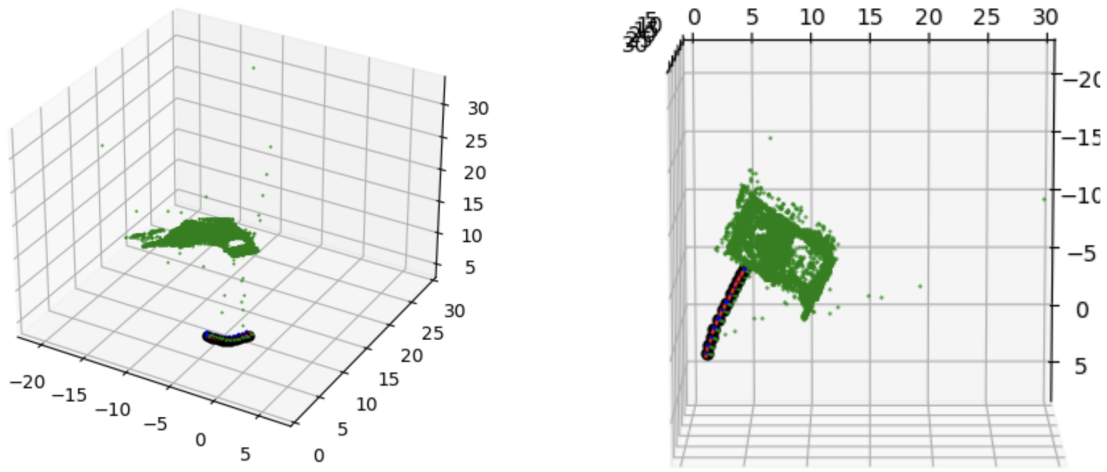

Figure 2.7: SFM results computed for the Lund Door dataset by GTSFM. Red points denote the camera locations and poses, and Green points show the sparse correspondences. (a) Oblique view. (b) Top view.

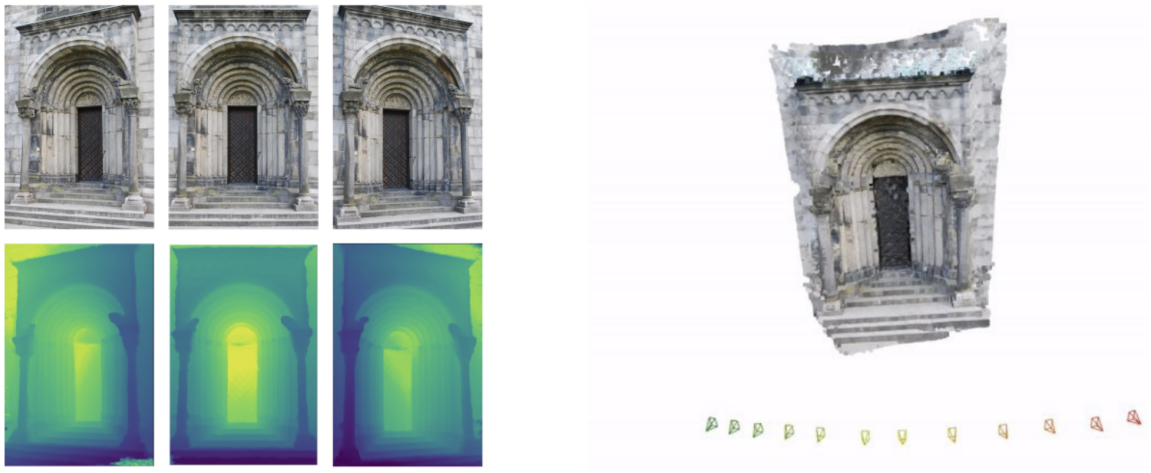

Figure 2.8: MVS results of Inference. Left: Output depth maps computed by the modulated PatchmatchNet. Right: Fused dense point cloud.

2.5.2 Experiments on Skydio Crane Mast dataset

We test the performances of latest SFM pipelines with MVS module implemented on Skydio Crane Mast dataset. All the experiments ran on a personal laptop with Ubuntu 20.04.1, Intel(R) Core(TM) i7-8750H CPU (16GB RAM), and Nvidia GeForce GTX 1070 Mobile GPU (8GB RAM).

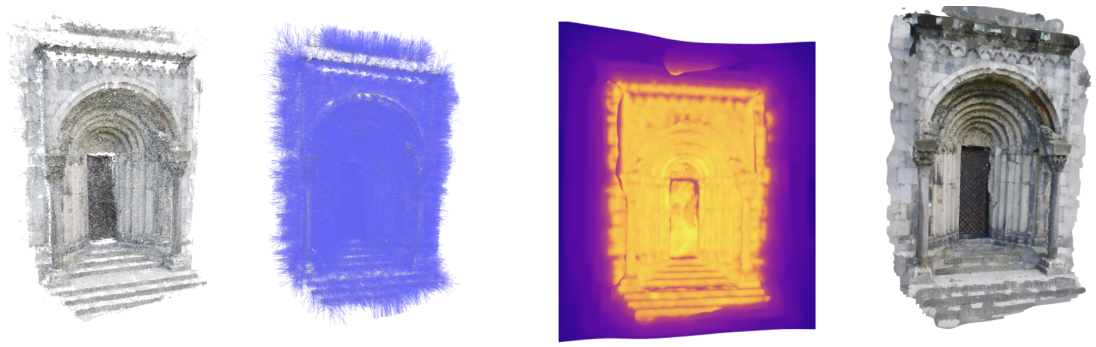

Figure 2.9: MVS results of Post-processing. From left to right are the dense point cloud, the estimated normal maps, the interpolated density map during Poisson reconstruction, and the final polygon mesh.

Multi View Stereo

Metric name	Value
num_valid_reference_views	12
mean_elapsed_time_per_ref_img(sec)	0.457049
mean_geometric_mask_valid_ratios	0.921456
mean_confidence_mask_valid_ratios	0.557354
mean_joint_mask_valid_ratios	0.536536
mean_reprojection_errors	0.217427

Voxel downsampling metrics

Metric name	Value
voxel size for downsampling	0.0175235
point cloud size before downsampling	5.51954e+06
point cloud size after downsampling	749813
compression ratio	7.36122
downsampling PSNR	59.518

Figure 2.10: MVS results of GTSFM. Left: Output depth maps computed by the modulated PatchmatchNet. Right: Fused dense point cloud.

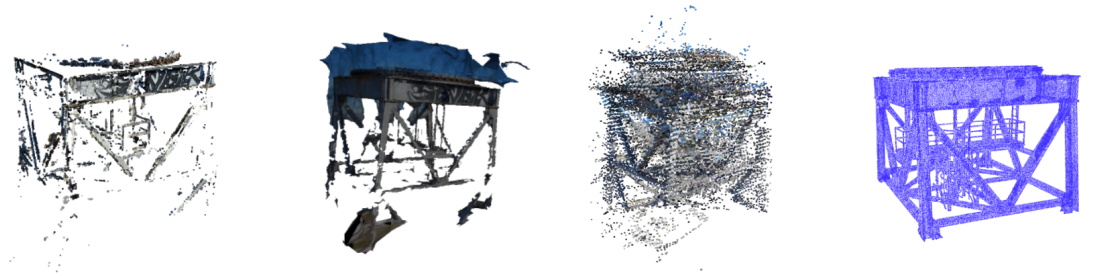

Figure 2.11: MVS results of the Skydio dataset. From left to right are COLMAP [23], Meshroom [24], GTSFM (ours), and the ground truth.

Table 2.6: Latest SFM pipelines comparison: Skydio Crane Mast dataset

Approaches	**SFM Run-time(s)** $\downarrow$	**MVS Run-time(s)** $\downarrow$
COLMAP [23]	208	3298
OpenMVG [22] + OpenMVS [19]	144	-
Meshroom [24]	419	1258
GTSFM [1]	1165	194

The results in Table 2.6 show that our MVS module is the fastest one among all other popular SFM pipelines integrated with MVS process. $-$ means the corresponding step failed during reconstruction.

2.6 Conclusion

Surveying and integrating DL-MVS approaches is my major work in this thesis. In this section, we present a thorough analysis on latest deep-learning MVS methods. Most of these novel works are based on the idea of plane sweeping and the differentiable image warping (homography). Great progress has been made on both processing time and result quality since MVSNet [36] was proposed.

We show by complexity analysis and experiments how PatchmatchNet [2] beats the other approaches and how we modulate PatchmatchNet in our depth multi-view optimizer, as the last step of our SFM pipeline. Pre-processing and post-processing methods and algorithms are implemented to make it a complete MVS work flow.

By comparing with other popular SFM pipelines, we find that integrating Patchmatch-Net significantly boosts the MVS step. However, there is still some problem and limitation.

Although PatchmatchNet [2] has already been small compared with other MVSNets, for neural network approaches the memory can always be a problem. Compared with the non-deep-learning Patchmatch methods implemented by COLMAP, our MVS module still needs more scalability.

CHAPTER 3
DEEP VIEW SYNTHESIS FOR GTSFM

In this chapter, we make a survey on latest deep-learning based view synthesis approaches, especially focusing on the reconstruction of both density field (geometry) and radiance field (lighting). We append Instant-NGP [3] as an auxiliary third-party module for GTSFM. In additional to the view synthesis, we also boost the original instant-NGP by forcing inferring and rendering to only focus on overlapping field of views (FOVs) area. With this help, we can extract a density field with foreground objects only, from which we can extract reconstructed dense point clouds and meshes of high quality.

3.1 Introduction

Compared with MVS, the goal of view synthesis is to render images of the estimated scene as it would appear from unseen viewpoints [9]. Although there is no need for view synthesis to reconstruct complete object model, like dense correspondences, view synthetic algorithms should take both object geometric information and lighting information into consideration. With reconstructed lighting information, the system can even generate synthesized videos from customized camera motions, which provides the users with large freedom. In this perspective, view synthesis can be considered as a collaboration task between computer vision and computer graphics. Therefore, the attempt to integrate latest view synthesis approaches into SFM pipelines is meaningful.

The difficulty for view synthesis comes from that 2D images merge the geometry (object shape, material and texture) and lighting (environment light sources, ambiance light) information together into RGB colors on pixels. Hence, great efforts are required to decouple these kinds of information apart. Studies on deep learning based computer graphics / computer vision approaches, such as MVSNet [36], suggest that trained deep learning

methods can efficiently analyze and express geometric primitives.

To get a better knowledge of latest deep-learning view synthesis methods, we make a thorough survey and append the most suitable method into GTSFM. We also explore the possibility to obtain smooth MVS results as a by-product. Our contributions include:

- We show by complexity analysis and experiments that Instant-NGP [3] performs the best among tested deep-learning view synthesis approaches and is suitable to be work with SFM pipelines.
- We boost the Instant-NGP by forcing the inferring and rendering region to focus on the overlapping FOVs, which also makes it possible to extract smooth dense polygon mesh from the reconstructed density field.

3.2 Related Work

In this chapter, we will briefly introduce traditional and deep-learning based view synthesis methods.

Traditional view synthesis algorithms. Traditional view synthesis approaches often rely on sampling and interpolation techniques. A dense sampling of views is required to interpolating light fields [40, 41, 42] and render images from novel views that look like reality. When dense sampling is not available, approaches need to only use observed images to estimate geometry representations of the target objects and environment lighting. Using a differentiable renderer to optimize the scene by gradient-based methods with loss between rendered images and the observed images can be a straightforward methods. This kind of gradient-based mesh optimization approaches can be divided into two genres: differentiable rasterizers [43, 44] and pathtracers [45, 46]. However, in addition to the complicated implementation of the differentiable renderer, their performances are also significantly influenced by the quality of initialized template meshes (sometimes called the initial guess), which is unavailable for unconstrained real-world scenes [47] in most cases. Plenoxels [50]

makes use of the differentiable volume rendering algorithm to learn density and spherical harmonic coefficients at each voxel by gradient descent, which manages to simplify the gradient-based optimization algorithm without neural networks.

Deel-Learning view synthesis algorithms. Compared with traditional algorithms, neural graphics representations (NGP) is a novel approach to use neural networks to implicitly represent the object geometry. This implicit representation is often a mapping from a spatial coordinates to the target value. For example, neural presentations for surface [48] is signed distance field (SDF) which means the signed distance from a world point to its closest surface. And neural presentations for volume is a density field [47]. For lighting, neural radiance field (NeRF) [47] is proposed and fit well with the differentiable volume rendering, which make it possible to do gradient descent when training the neural networks. This logic is very similar to the usage of differentiable homography in DL-MVS methods [36, 51, 39, 2]. After NeRF [47] was proposed in 2020, related studies [86, 49, 87, 88, 3] have been making deep-learning based view synthesis algorithms much faster. These novel techniques have not been integrated to latest SFM pipelines yet.

3.3 Survey on deep view synthesis approaches

In this section, we will go over the fundamental algorithm for most gradient-based view synthesis methods, differentiable volume rendering. Then we will focus on comparing three of the most popular deep view synthesis approaches to find the most suitable methods for GTSFM.

3.3.1 Volume Rendering

If we consider the whole scene as a volume, the task to generating a photorealistic image from a novel view can be regarded as volume rendering from the given view. One solution is to apply a widely acknowledged traditional volume rendering algorithm [89] based on ray tracing under Blinn's Low Albedo approximation [90].

Blinn's Low Albedo approximations

Volume density scattering model was first brought into Computer Graphics by Blinn in 1982 [90]. In Blinn's work, a volume to be rendered is considered as a space diffused with small particles. It is necessary to create a simpler model to simulate light interacting with particles, so Blinn proposed the *Single Scattering Cloud Model*. It is intuitive to have a measurement on the density of particles in the target volume, which can be used to evaluate the probability whether an incident ray to interact with any of a particle or simply pass through the volume. With the Low Albedo approximation that the primary effect is from the interaction of a ray of light with a single particle [90], we only consider the first ray-particle interaction. If the particles are solid so that the ray can only be scattered but cannot go through it, the scattered light can be seen only if 1) the incident light directly hits the particle, 2) the scatter light escapes from all other particles through the scattering channel (See Figure 3.1).

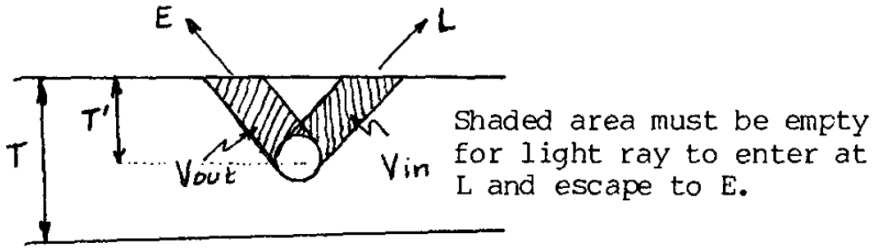

Figure 3.1: The Scattering Conditions [90]. $\mathbf{T}$ and $\mathbf{T}'$ are the depth of the volume and the depth of hit particle, respectively. V_{in} and V_{out} are the volumes of the light's incident and scattering channels. The radius of the particle and the channels is p.

Then we can calculate the attenuation of light traversing through a incident/scattering cylinder channel of radius p and volume V by computing the statistical probability of 0 particles in the volume V. Suppose there are n particles in a unit volume, the expect number of particles in the given volume will be nV. When n is small, it is close to the Poisson distribution [91], and the probability of occurrences $k = 0$ under expect number of

$\lambda = nV$ is

$$P(0;V) = \frac{(nV)^0}{0!}\exp(-nV) = \exp(-nV) \tag{3.1}$$

Then the brightness B of the hit particle can be calculated by:

$$B = \frac{bn\pi p^2}{\mu}\int_0^T P(0;V)dT' \tag{3.2}$$

where $\pi p^2/\mu$ is the projected viewing area, b is the particle brightness, ndT' is the expected number of particles per unit area.

Ray tracing algorithm for the low albedo case

When the previous method comes to the ray tracing algorithm proposed in [89], instead of using the number of particles in a unit volume, a spatial distributed density function $\sigma(x,y,z)$ is imported. To calculate the expected number of particles $N(V)$ in volume V:

$$N(V) = \int_V \sigma(x,y,z)dxdydz \tag{3.3}$$

For ray $\mathbf{r}(t) = \mathbf{o}+t\mathbf{d}$ starting at t_1 and ending at t_n, the particle brightness can be calculated by integrating continuous tracing time intervals until finishing the whole spreading path.

$$B(\mathbf{r}) = \int_{t_1}^{t_n} e^{-\int_{t_1}^{t}\sigma(\mathbf{r}(u))du}\sigma(\mathbf{r}(t))\frac{b\pi p^2}{\mu}dt \tag{3.4}$$

3.3.2 NeRF

For a general scene volume, the particles can be different from each other. What's worse, because of the heterogeneous object material and the environment lighting, the color of the particle can be view-dependent [47]. That is where a radiance field is necessary. Similar to the density filed $\sigma(\mathbf{x})$ where $\mathbf{x} = (x,y,z)$ stands for a spatial coordinates in the volume, the radiance field $\mathbf{c}(\mathbf{x},\mathbf{d})$ maps a coordinates and view direction pair, $(\mathbf{x},\mathbf{d})$, into RGB color

vector (r, g, b). If we use the radiance field to take the place of projected particle brightness $\frac{b\pi p^2}{\mu}$, and change brightness B to color C, we get

$$C(\mathbf{r}) = \int_{t_1}^{t_n} e^{-\int_{t_1}^{t} \sigma(\mathbf{r}(u))du} \sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \tag{3.5}$$

And this is the volume rendering formula with density and radiance field. Then to make the continuous integral discretized, a straitified sampling approach is designed to sample t_i uniformly from:

$$[t_1 + \frac{i-1}{n-1}(t_n - t_1), t_1 + \frac{i}{n-1}(t_n - t_1)]$$

while $1 \leq i \leq n-1$. Compared with sampling with even space, this sampling method covers the complete continuous space while training. Therefore, the discretized volume rendering formula is:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{n-1} e^{-\sum_{j=1}^{i} \sigma(\mathbf{r}(t_j))\delta_j} (1 - \sigma(\mathbf{r}(t_i))\delta_i)\mathbf{c}(\mathbf{r}(t_i), \mathbf{d}) \tag{3.6}$$

where $\delta_i = t_{i+1} - t_i$. According to this formula, $\hat{C}$ is differentiable to both density field σ and radiance field $\mathbf{c}$. That is to say, suppose there is a parameterized density function $\sigma(x, y, z)$ and σ is differentiable to all its parameters, then we can use gradient-based optimization to fit parameters in σ to learn a density field and so does the radiance field.

Because in real cases it is almost impossible to come up with a concrete mathematical model for either density or radiance field, using neural layers to implicitly represents these fields can be a feasible option. That is where neural radiance field (NeRF) born. NeRF uses two MLPs to implement σ and $\mathbf{c}$, respectively. It synthesizes new images by performing the volume rendering (Equation 3.6) along traced rays towards a spatial location $\mathbf{x} = [x, y, z]^T$ in direction (θ, ϕ). More details can be seen in Figure 3.2.

The idea of NeRF opened a new era for deep view synthesis approaches. Because NeRF requires long-time training with sufficient observed images, studies to boost NeRF [49, 50,

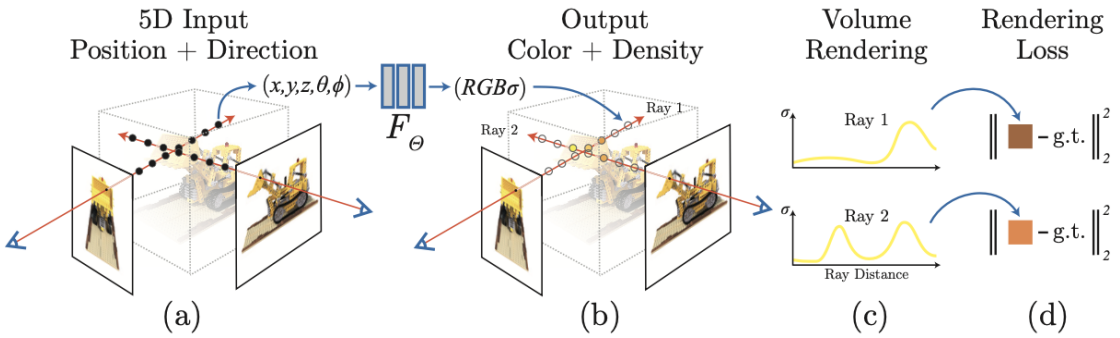

Figure 3.2: An overview of NeRF [47]. To synthesize images from a novel viewpoint, it first sampling 5D coordinates (spatial coordinates and viewing direction along traced rays. Then query the coordinates from the density field MLP to get volume density, query the coordinates plus direction from the radiance field MLP to get a color vector. Then it uses the obtained colors to perform differentiable volume rendering and combine results from all traced rays into a generated image. At last, it learns the MLPs by minimizing the loss between the ground truth images.

3], to enable sparse observed images [49], and to improve output quality [86, 88] have been emerging and making great progress.

3.3.3 MVSNeRF

The success of MVSNet [36] proves the network architecture of MVSNet can efficiently extract geometry information (dense correspondences) from images from multiple calibrated views. MVSNeRF [49] makes great use of the architecture of MVSNet to construct a cost volume from input images and camera parameters. Then the 3D convolutional layers further encode the cost volume to be a spatial field of geometric features. These features will help the following MLPs to predict the density and radiance. Details can be seen in Figure 3.3.

One differences for MVSNeRF from other NeRF approaches is MVSNeRF uses observed images for both training and inference. In this way, only small number of calibrated views (sparse sampling) are needed for inference. Another major advantage of MVSNeRF is that it does not force to train for every single scene. Only a fine-tuning for the MLP layers can be applied for higher quality synthesis. There are two possible reasons: 1) scene

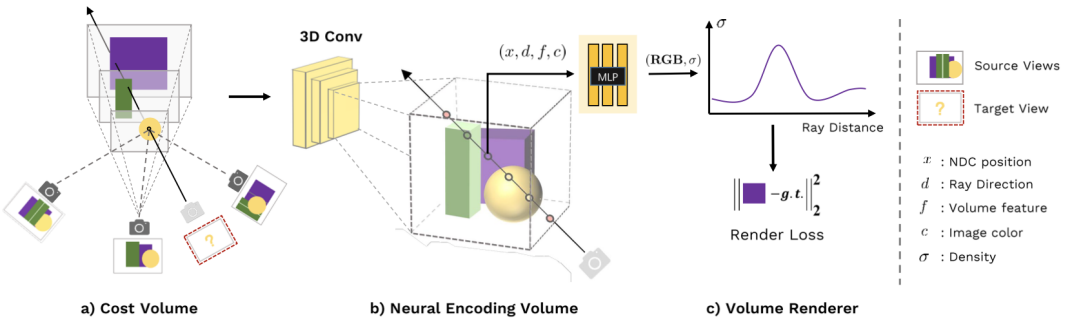

Figure 3.3: An overview of MVSNeRF [49]. It synthesizes new images by 1) constructing the cost volume like MVSNet, 2) applying several 3D convolutional layers to the cost volume and get the neural encoding volume, 3) obtaining encoded features for target location $\mathbf{x}$ by querying the neural encoding volume, 4) combining location, ray direction, features and image color to query a MLP to get density and color, 5) volume rendering as NeRF.

images are used for inference, so the network does not need to remember the whole scene. 2) the similar structure used in MVSNet has been proven to be workable for unseen scenes.

3.3.4 Instant-NGP

Many NeRF related studies mention encoding spatial location $\mathbf{x} = [x, y, z]^T$ and view direction $\mathbf{d} = [\cos\theta\cos\phi, \cos\theta\sin\phi, \sin\theta]^T$ efficiently into higher dimension before querying density and color information is helpful. For example, NeRF applies an improved scalar position encoding function used in transformers [92] to encode each scalar values in normalized $\mathbf{x}$ and $\mathbf{d}$:

$$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), ..., \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)) \tag{3.7}$$

. In MVSNeRF, the feature obtained from the neural encoding volume by $\mathbf{x}$ can also be regarded as an encoding of $\mathbf{x}$.

Instant-NGP [3] designs a highly efficient but powerful multi-resolution hash encoding, which enables Instant-NGP to be trained in a matter of seconds. Detailed steps can be seen

in Figure 3.4. The spatial hash function [93] is

$$h(\mathbf{x}) = (\oplus_{i=1}^{d} x_i \pi_i) \mod T \tag{3.8}$$

where $\oplus$ is the bit-wise XOR operation, π_i are unique and large prime numbers, T is the hash table size. The great design here is to avoid hash collision by multi-resolution complement and dominating training samples.

Multi-resolution Complement: Lower resolution levels with coarse features have little chance to collide while higher resolution levels with fine features are likely to have hash collision. It is unlikely to have collision simultaneously at every level for a given $\mathbf{x}$ because the collision happens pseudo-randomly.

Dominating Training Samples: The gradients during training process are dominated by samples that are important to the output synthesized images (visible locations with high density) rather than by the collision average and aliased table entry.

Considering its fast training speed and high synthesis quality, we decide to append Instant-NGP to our GTSFM as a third-party tool for view synthesis tasks.

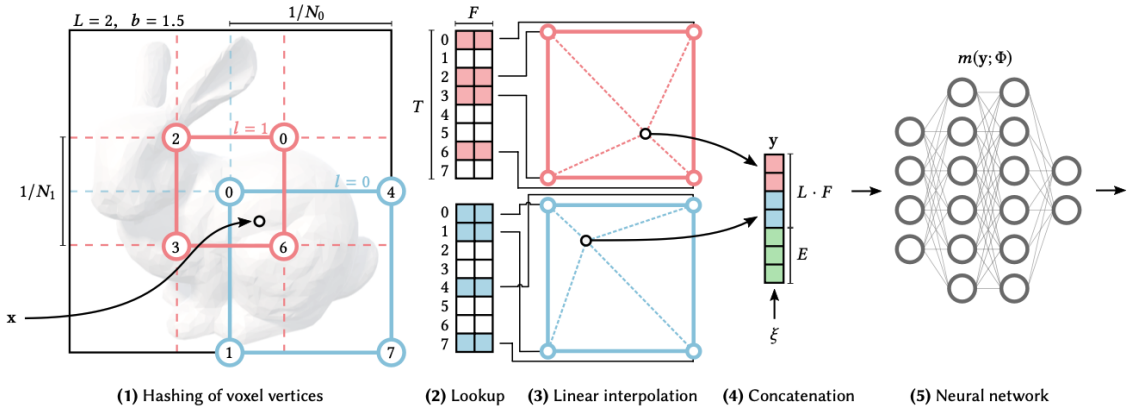

Figure 3.4: An overview of Instant-NGP's multi-resolution hash enccoding in 2D [3]. 1) for a given coordinates $\mathbf{x}$, at multiple resolution levels, find the surrounding voxels. 2) calculate their indices by hashing their integer coordinates. 3) look up their corresponding features from the hash tables for each resolution level. 4) linear interpolate the feature by the corner coordinates to make it differentiable. 5) concatenate the interpolated features to be the encoded result $\mathbf{y}$.

3.4 Approach: Boost Instant-NGP by Focusing on Overlapping FOVs

Instant-NGP can train for each scene in seconds on GPU, but there is still room for further acceleration. For example, we find that Instant-NGP will waste a lot of time and memory resources in optimizing the background volume. Although this kind of optimization offers better view synthesis quality in background, learning density and color for invisible locations and background in infinite distance is unreasonable. Meanwhile, the goal of SFM pipelines often focuses on reconstructing the foreground objects. Hence, it is meaningful to find a way to force Instant-NGP to only pay attention to the foreground area.

The field of view (FOV) is the frustum that a camera can image. Its location and direction are decided by the camera pose, while its size is decided by the focal length and the sensor size. Cameras with overlapping FOVs image a shared world space. For SFM datasets, the shared world space among all validate cameras is often where the foreground object is located. We then design an approach to force Instant-NGP only to infer and render the overlapping Field-of-View region. It not only accelerates the training process of Instant-NGP, but also enables to obtain the dense polygon mesh from the reconstructed density field.

3.4.1 System Overview

To boost the Instant-NGP and fit it better with GTSFM, we design the following steps:

1. **Overlapping FOV Estimation**: Compute overlapping FOVs bounds.
2. **Scene Transformation**: Translate and scale the estimated scene to make the overlapping FOVs bounds fit into the unit cube.
3. **Instant-NGP Training**: Train Instant-NGP with images and transformed scene.
4. **Density Field Extraction**: Make a dense sampling of the learned density field. Assign solid points with the corresponding averaged colors at sampled locations where

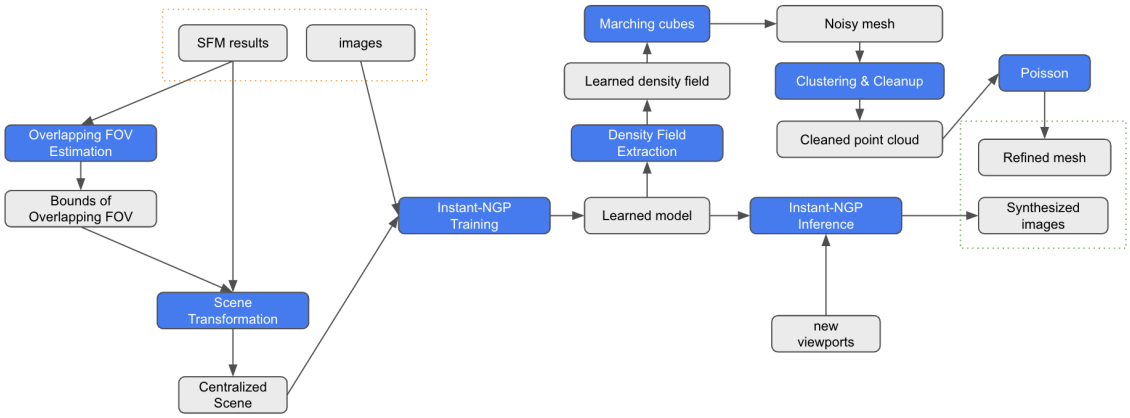

Figure 3.5: Boosted Instant-NGP. Variables in the orange dashed box are the input SFM results computed by previous modules. Variables in the green dashed box are the final output results of the integrated Instant-NGP.

the densities are larger than the threshold.

5. **Marching Cubes**: Use marching cubes algorithm [29] to generate coarse surfaces.

6. **Clustering & Cleanup**: Cluster the generated point cloud, filter out and remove isolated clusters with only a small number of points.

7. **Poisson**: Surface reconstruction from point coordinates and normals.

8. **Instant-NGP Inference**: Input target viewpoints to obtain the synthesized image.

3.4.2 Fast Overlapping FOVs Estimation

To accurately detect two overlapping FOVs, because the shape of FOV (a frustum) is a convex polyhedral, the separating axis theorem [94] can be applied. However, to enable a fast overlapping FOVs estimation, we can use a sampling based algorithm.

Assume there n cameras, the camera parameters for camera c_i is $\mathbf{R}_i$, $\mathbf{t}_i$, and $\mathbf{K}_i$. The image resolution for each camera is (w_i, h_i). A unified sampled 3D cube volume V with resolution $L \times L \times L$ and grid voxel size a. V is large enough to cover the complete scene. Therefore, the number of sampled voxels are $N = (L/a)^3$. Then the 3D mask $\mathcal{M}_{ofov}(\mathbf{x})$ for sampled locations that stays in the overlapping FOVs among all n cameras

can be calculated by:

$$\mathcal{M}_{ofov}(\mathbf{x}) = \bigcap_{i=0}^{n} (0 \leq u_{\mathbf{x},i} < w_i) \cap (0 \leq v_{\mathbf{x},i} < h_i) \tag{3.9}$$

where

$$[u_{\mathbf{x},i}, v_{\mathbf{x},i}, 1]^T \sim \mathbf{K}_i(\mathbf{R}_i\mathbf{x} + \mathbf{t}_i)$$

Then the location and bounds of the overlapping FOVs are the bounds of voxels filtered by $V[\mathcal{M}_{ofov}]$. Then we need to transform the whole scene so that the overlapping FOVs can be translated and scaled to fit into the unit cube (See Figure 3.6).

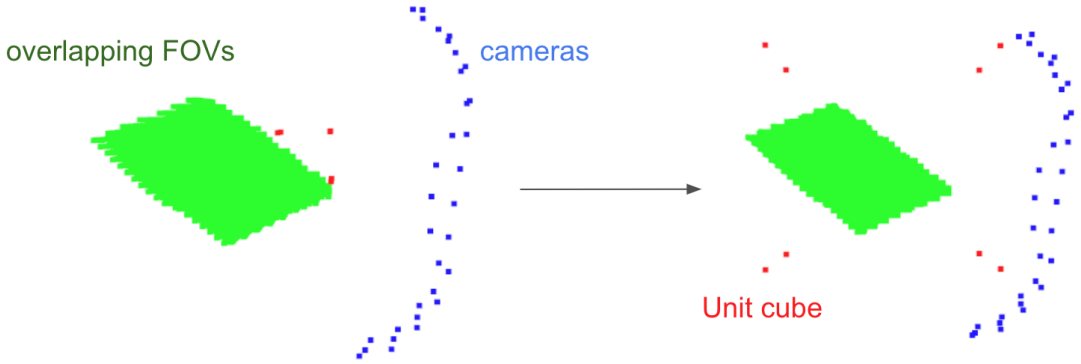

Figure 3.6: Scene Transformation with overlapping FOVs bounds. Green points denote locations within the overlapping FOVs. Red points denote eight corners of the unite cube. Blue points denote the camera locations.

Assume the nearest left bottom corner of the overlapping FOVs bounds is $\mathbf{x}_0$, the dimensions of the bounds are (a_x, a_y, a_z). Then the transformation matrix $\mathcal{T}$ will be

$$\mathcal{T} = \begin{bmatrix} \mathbf{I} & -\mathbf{x}_0 \\ \mathbf{0}^T & \frac{1}{\max\{a_x, a_y, a_z\}} \end{bmatrix} \tag{3.10}$$

Here the transformation will be performed on every camera poses and sparse correspondences included in the SFM results.

3.4.3 Object Mesh Cleanup

There is still possibilities to have noisy in the dense point cloud if we extract the point cloud directly from a density threshold. One solution is to remove isolated triangles as well as its related points from the dense mesh reconstructed by the marching cubes algorithm [29]. Then re-mesh the cleaned up dense point cloud by Poisson algorithm [27]. Given the strict computing resources requirements by Instant-NGP, we do these experiments on a Windows 10 Desktop with a Nvidia RTX 3080Ti video card.

3.5 Experiment Results

In this experiment, we continue working on the Skydio Crane Mast dataset with 32 images. We compare the both training time and the quality of extracted mesh to see whether our boosted Instant-NGP meets the expectation.

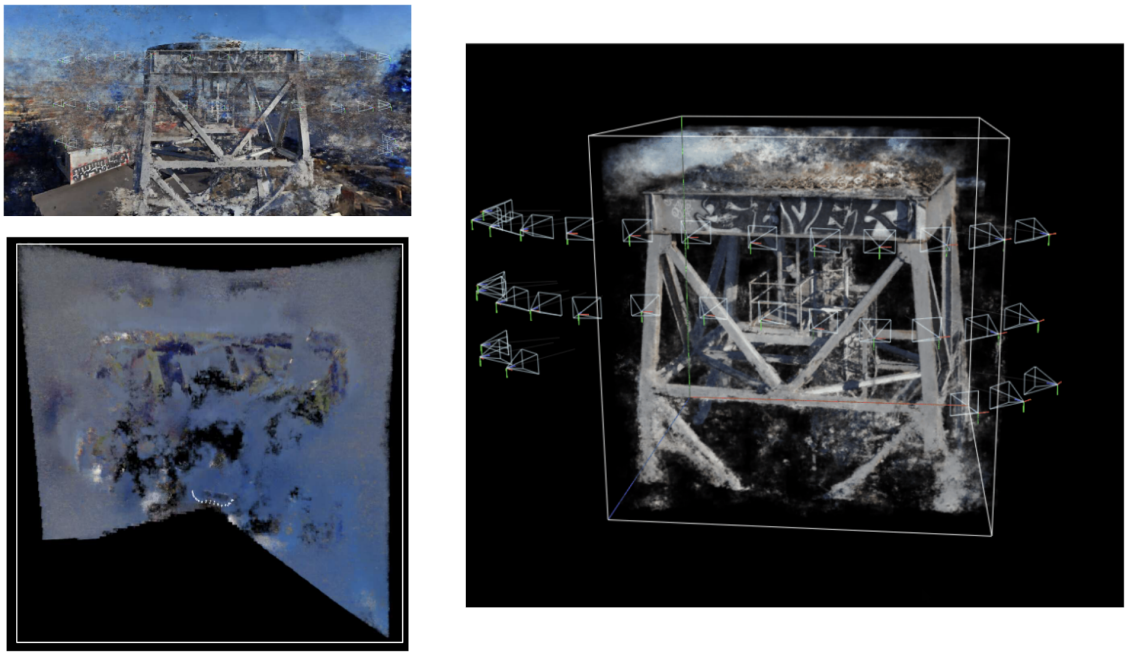

Figure 3.7: Comparison of regular Instant-NGP training and boosted Instant-NGP training. Left is the regular Instant-NGP's results and on the right is the result that boosts the Instant-NGP by forcing the reconstruction to stay inside the unit cube.

When we extract the foreground object's mesh, if we use the regular Instant-NGP, we

need to cut off all the environment (or background) volumes. We can also notice that many occupied voxel is exactly empty in the real world. Hence, instead of cropping the environment after training, keep the background distance out of the training and rendering area can be a better approach. Figure 3.8 also shows that our boosted Instant-NGP has less noise point than the other two methods.

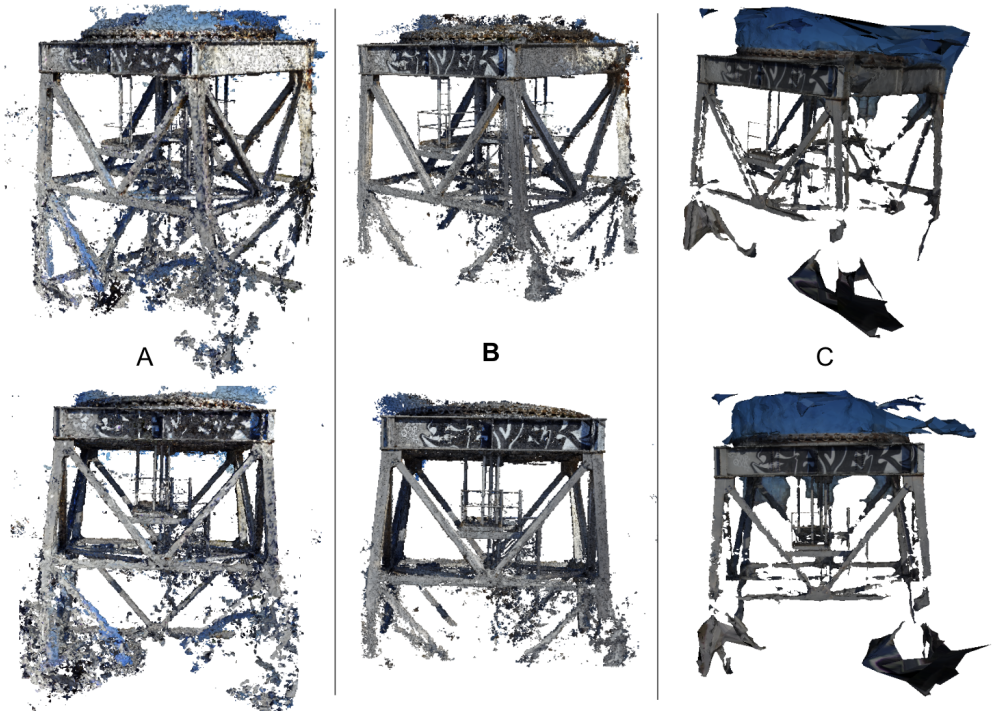

Figure 3.8: Comparison of cleaned meshes of regular Instant-NGP training, boosted Instant-NGP training, and the results of Meshroom.

For the execution time, the dense reconstruction time for Meshroom is about 20 minute and the training process for regular Instant-NGP is 5 minite. Because there is no need for boosted Instant-NGP to think about the background, the result converged very fast in about 2 minute.

3.6 Conclusion

In this chapter, we go over the latest deep-learning based view synthesis approaches. Because the invention of differentiable volume rendering, these methods can be implemented

in neural network architectures, and can be learned by gradient optimization. Compared with traditional view sampling algorithms, these methods are more simpler to implement and can be translated to different scenes by re-training the network.

Given the design that view synthesis will both learn the density field (geometry) and the radiance field (lighting), we manage to extract and optimize good polygon meshes from trained Instant-NGP. It is meaningful because if the dense correspondences can be seen as the by-product of view synthesis, we don't need a separate MVS module any more.

However, there is still limits and constraints on the reviewed approaches. A very long time is needed for training a deep view synthesis model, unless the strict requirements on computing and storage devices are met. Another problem is the model's scene dependence. When it turns to another thing, the model need to be re-trained. Although MVSNeRF [49] has already wanted to make a difference, its performances under pre-trained model are still need to be improved.

CHAPTER 4
CONCLUSION

In this work, we explored and discussed present limits and constraints in performing highly efficient MVS and view synthesis tasks in SFM pipelines. We also proved that by integrating deep-learning based methods, both performances and time consumption can be improved. However, there is still a lot of room for future domain studies. Here, I will present some of my humble ideas that may have the potential to make a difference.

4.1 Future Work

4.1.1 Novel Architectures of DL-MVS and Deep View Synthesis.

Today's DL-MVS approaches rely heavily on the traditional MVS algorithms, like plane sweeping [25] and PatchMatch [32]. A common step is to discretize the continuous computation and make it differentiable so that simple gradient-based optimization can address complex tasks. In deep view synthesis methods, lots of works have done to improve the encoding efficiency. In this perspective, deep learning methods' success sits a lot in the better expressiveness of encoded features and the speedup from highly parallelized computations. In the future, more and more novel and efficient approaches will be designed to break the limitation and constraints of traditional algorithms.

4.1.2 Explicit vs. Implicit Representation

Most deep learning approaches mentioned in NeRF apply implicit neural representation, which makes neural layers a function where we query the layer some vectors in the space, it will tell us the corresponding vector or scalar. The advantage of using implicit representation is that it avoids the complicated structure and the concern of dimension explosion

when the search domain is very large. But it also results in remaining in a black box that we do not know whether the neural layers have sufficient capability to learn and interpolate the target fields. To make implicit representation more expressive, complicated neural architectures are imported, which will significantly slow down the training and inferring process.

Two very fast view synthesis approaches, Plenoxels [50] and Instant-NGP [3] are two great examples. Plenoxel uses the explicit representation by directly learning density and radiance variables on each voxel, while Instant-NGP uses a strong and high-efficiency hashing encoding to avoid the complexity of network structure in the following implicit representations of density and radiance fields. These two examples also make us think about the question when to use explicit representation is better and when is not.

4.1.3 Generalized MVS Task

From our study to deep view synthesis, we find that it is possible to obtain great dense correspondences and meshes from the reconstructed depth field. It brings me with the idea that in the future, will there be a general computer vision task, that solves different kinds of computer vision tasks all in oneself. Instant-NGP [3], in my opinion, is the initial template of such generalized MVS tasks. With the same hashing encoding mechanism, Instant-NGP can perform multiple tasks like Gigapixel Image approximation [95], SDF and NeRF reconstruction. Developing such a generally used architecture can be very helpful and attractive.

REFERENCES

[1] A. Baid *et al.*, *GTSFM: Georgia tech structure from motion*, https://github.com/borglab/gtsfm, 2021.

[2] F. Wang, S. Galliani, C. Vogel, P. Speciale, and M. Pollefeys, "Patchmatchnet: Learned multi-view patchmatch stereo," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 194–14 203.

[3] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *arXiv:2201.05989*, Jan. 2022.

[4] R. Jensen, A. Dahl, G. Vogiatzis, E. Tola, and H. Aanæs, "Large scale multi-view stereopsis evaluation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2014, pp. 406–413.

[5] S. Ullman, "The interpretation of structure from motion," *Proceedings of the Royal Society of London. Series B. Biological Sciences*, vol. 203, no. 1153, pp. 405–426, 1979.

[6] D. Forsyth and J. Ponce, *Computer vision: A modern approach.* Prentice hall, 2011.

[7] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[8] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, IEEE, vol. 1, 2006, pp. 519–528.

[9] D. Scharstein, *View synthesis using stereo vision*. Springer, 2003.

[10] H. Agrawal *et al.*, "Cloudcv: Large-scale distributed computer vision as a cloud service," in *Mobile cloud visual media computing*, Springer, 2015, pp. 265–290.

[11] T. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges," in *2010 24th IEEE international conference on advanced information networking and applications*, Ieee, 2010, pp. 27–33.

[12] B. Resch, H. Lensch, O. Wang, M. Pollefeys, and A. Sorkine-Hornung, "Scalable structure from motion for densely sampled videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3936–3944.

[13] Y. Tian, G. Yang, Z. Wang, H. Wang, E. Li, and Z. Liang, "Apple detection during different growth stages in orchards using the improved yolo-v3 model," *Computers and electronics in agriculture*, vol. 157, pp. 417–426, 2019.

[14] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[16] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[17] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[18] Z. Liu *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[19] D. Cernea, "OpenMVS: Multi-view stereo reconstruction library," 2020.

[20] C. Sweeney, T. Hollerer, and M. Turk, "Theia: A fast and scalable structure-from-motion library," in *Proceedings of the 23rd ACM international conference on Multimedia*, 2015, pp. 693–696.

[21] M. Adorjan, "Opensfm: A collaborative structure-from-motion system," Ph.D. dissertation, Wien, 2016.

[22] P. Moulon, P. Monasse, R. Perrot, and R. Marlet, "Openmvg: Open multiple view geometry," in *International Workshop on Reproducible Research in Pattern Recognition*, Springer, 2016, pp. 60–74.

[23] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4104–4113.

[24] C. Griwodz *et al.*, "Alicevision Meshroom: An open-source 3D reconstruction pipeline," in *Proceedings of the 12th ACM Multimedia Systems Conference - MMSys '21*, ACM Press, 2021.

[25] S. M. Seitz and C. R. Dyer, "Photorealistic scene reconstruction by voxel coloring," *International Journal of Computer Vision*, vol. 35, no. 2, pp. 151–173, 1999.

[26] K. N. Kutulakos and S. M. Seitz, "A theory of shape by space carving," *International journal of computer vision*, vol. 38, no. 3, pp. 199–218, 2000.

[27] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006.

[28] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE transactions on visualization and computer graphics*, vol. 5, no. 4, pp. 349–359, 1999.

[29] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[30] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, IEEE, vol. 2, 2005, pp. 807–814.

[31] I. Ernst and H. Hirschmüller, "Mutual information based semi-global stereo matching on the gpu," in *International Symposium on Visual Computing*, Springer, 2008, pp. 228–239.

[32] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "Patchmatch: A randomized correspondence algorithm for structural image editing," *ACM Trans. Graph.*, vol. 28, no. 3, p. 24, 2009.

[33] R. T. Collins, "A space-sweep approach to true multi-image matching," in *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, 1996, pp. 358–363.

[34] D. Gallup, J.-M. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys, "Real-time plane-sweeping stereo with multiple sweeping directions," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2007, pp. 1–8.

[35] E. Zheng, E. Dunn, V. Jojic, and J.-M. Frahm, "Patchmatch based joint view selection and depthmap estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1510–1517.

[36] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan, "Mvsnet: Depth inference for unstructured multi-view stereo," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 767–783.

[37] Y. Yao, Z. Luo, S. Li, T. Shen, T. Fang, and L. Quan, "Recurrent mvsnet for high-resolution multi-view stereo depth inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5525–5534.

[38] Z. Yu and S. Gao, "Fast-mvsnet: Sparse-to-dense multi-view stereo with learned propagation and gauss-newton refinement," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1949–1958.

[39] R. Chen, S. Han, J. Xu, and H. Su, "Point-based multi-view stereo network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1538–1547.

[40] M. Levoy and P. Hanrahan, "Light field rendering," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 31–42.

[41] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 43–54.

[42] A. Davis, M. Levoy, and F. Durand, "Unstructured light fields," in *Computer Graphics Forum*, Wiley Online Library, vol. 31, 2012, pp. 305–314.

[43] M. M. Loper and M. J. Black, "Opendr: An approximate differentiable renderer," in *European Conference on Computer Vision*, Springer, 2014, pp. 154–169.

[44] S. Liu, T. Li, W. Chen, and H. Li, "Soft rasterizer: A differentiable renderer for image-based 3d reasoning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7708–7717.

[45] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen, "Differentiable monte carlo ray tracing through edge sampling," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pp. 1–11, 2018.

[46] M. NIMIER-DAVID, D. VICINI, T. ZELTNER, and W. JAKOB, "Mitsuba 2: A retargetable forward and inverse renderer–supplemental material,"

[47] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *European conference on computer vision*, Springer, 2020, pp. 405–421.

[48] J. Zhang, Y. Yao, and L. Quan, "Learning signed distance field for multi-view surface reconstruction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 6525–6534.

[49] A. Chen *et al.*, "Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 14 124–14 133.

[50] Sara Fridovich-Keil and Alex Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," in *CVPR*, 2022.

[51] X. Gu, Z. Fan, S. Zhu, Z. Dai, F. Tan, and P. Tan, "Cascade cost volume for high-resolution multi-view stereo and stereo matching," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2495–2504.

[52] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[53] T. Schöps *et al.*, "A multi-view stereo benchmark with high-resolution images and multi-camera videos," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[54] B. Mildenhall *et al.*, "Local light field fusion: Practical view synthesis with prescriptive sampling guidelines," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–14, 2019.

[55] S. Shen, "Accurate multiple view 3d reconstruction using patch-based stereo for large-scale scenes," *IEEE transactions on image processing*, vol. 22, no. 5, pp. 1901–1914, 2013.

[56] J.-D. Boissonnat and B. Geiger, "Three-dimensional reconstruction of complex shapes based on the delaunay triangulation," in *Biomedical image processing and biomedical visualization*, International Society for Optics and Photonics, vol. 1905, 1993, pp. 964–975.

[57] O. Faugeras *et al.*, "Real time correlation-based stereo: Algorithm, implementations and applications," Inria, Tech. Rep., 1993.

[58] M. Bleyer, C. Rhemann, and C. Rother, "Patchmatch stereo-stereo matching with slanted support windows.," in *Bmvc*, vol. 11, 2011, pp. 1–11.

[59] R. Szeliski and P. Golland, "Stereo matching with transparency and matting," in *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, IEEE, 1998, pp. 517–524.

[60] D. Gallup, J.-M. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys, "Real-time plane-sweeping stereo with multiple sweeping directions," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2007, pp. 1–8.

[61] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[62] R. Schnabel and R. Klein, "Octree-based point-cloud compression.," in *PBG@ SIGGRAPH*, 2006, pp. 111–120.

[63] T. Golla and R. Klein, "Real-time point cloud compression," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 5087–5092.

[64] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3d: A modern library for 3d data processing," *arXiv preprint arXiv:1801.09847*, 2018.

[65] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE international conference on robotics and automation*, IEEE, 2011, pp. 1–4.

[66] C. Moenning and N. A. Dodgson, "Fast marching farthest point sampling for implicit surfaces and point clouds," *Computer Laboratory Technical Report*, vol. 565, pp. 1–12, 2003.

[67] ——, "A new point cloud simplification algorithm," in *Proc. int. conf. on visualization, imaging and image processing*, 2003, pp. 1027–1033.

[68] ——, "Intrinsic point cloud simplification," *Proc. 14th GrahiCon*, vol. 14, p. 23, 2004.

[69] J. Qi, W. Hu, and Z. Guo, "Feature preserving and uniformity-controllable point cloud simplification on graph," in *2019 IEEE International Conference on Multimedia and Expo (ICME)*, IEEE, 2019, pp. 284–289.

[70] M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, and M. Ovsjanikov, "Pointcleannet: Learning to denoise and remove outliers from dense point clouds," in *Computer Graphics Forum*, Wiley Online Library, vol. 39, 2020, pp. 185–203.

[71] M. Okutomi and T. Kanade, "A multiple-baseline stereo," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 15, no. 4, pp. 353–363, 1993.

[72] J. Dai *et al.*, "Deformable convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 764–773.

[73] Q. Xu and W. Tao, "Learning inverse depth regression for multi-view stereo with correlation cost volume," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 12 508–12 515.

[74] ——, "Multi-scale geometric consistency guided multi-view stereo," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5483–5492.

[75] H. Aanæs, R. R. Jensen, G. Vogiatzis, E. Tola, and A. B. Dahl, "Large-scale data for multiple-view stereopsis," *International Journal of Computer Vision*, vol. 120, no. 2, pp. 153–168, 2016.

[76] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski, "Towards internet-scale multi-view stereo," in *2010 IEEE computer society conference on computer vision and pattern recognition*, IEEE, 2010, pp. 1434–1441.

[77] D. Gallup, J.-M. Frahm, P. Mordohai, and M. Pollefeys, "Variable baseline/resolution stereo," in *2008 IEEE conference on computer vision and pattern recognition*, IEEE, 2008, pp. 1–8.

[78] R. Zhang, S. Li, T. Fang, S. Zhu, and L. Quan, "Joint camera clustering and surface segmentation for large-scale multi-view stereo," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2084–2092.

[79] K.-J. Yoon and I.-S. Kweon, "Locally adaptive support-weight approach for visual correspondence search," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, IEEE, vol. 2, 2005, pp. 924–931.

[80] S. Van Aelst and P. Rousseeuw, "Minimum volume ellipsoid," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 1, pp. 71–82, 2009.

[81] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[82] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows," 1988.

[83] C. Olsson and O. Enqvist, "Stable structure from motion for unordered image collections," in *Scandinavian Conference on Image Analysis*, Springer, 2011, pp. 524–535.

[84] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, Spie, vol. 1611, 1992, pp. 586–606.

[85] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th python in science conference*, Citeseer, vol. 130, 2015, p. 136.

[86] K. Zhang, G. Riegler, N. Snavely, and V. Koltun, "Nerf++: Analyzing and improving neural radiance fields," *arXiv preprint arXiv:2010.07492*, 2020.

[87] M. Niemeyer, J. T. Barron, B. Mildenhall, M. S. Sajjadi, A. Geiger, and N. Radwan, "Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs," *arXiv preprint arXiv:2112.00724*, 2021.

[88] H. Turki, D. Ramanan, and M. Satyanarayanan, "Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs," *arXiv preprint arXiv:2112.10703*, 2021.

[89] J. T. Kajiya and B. P. Von Herzen, "Ray tracing volume densities," *ACM SIGGRAPH computer graphics*, vol. 18, no. 3, pp. 165–174, 1984.

[90] J. F. Blinn, "Light reflection functions for simulation of clouds and dusty surfaces," *Acm Siggraph Computer Graphics*, vol. 16, no. 3, pp. 21–29, 1982.

[91] S. D. Poisson, "Probabilité des jugements en matière criminelle et en matière civile, précédées des règles générales du calcul des probabilitiés," *Paris, France: Bachelier*, vol. 1, p. 1837, 1837.

[92] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[93] M. T. B. H. M. Müller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," Technical report, Computer Graphics Laboratory, ETH Zurich, Switzerland, Tech. Rep., 2003.

[94] C. Liang and X. Liu, "The research of collision detection algorithm based on separating axis theorem," *Int. J. Sci*, vol. 2, no. 10, pp. 110–114, 2015.

[95] J. N. Martel, D. B. Lindell, C. Z. Lin, E. R. Chan, M. Monteiro, and G. Wetzstein, "Acorn: Adaptive coordinate networks for neural scene representation," *arXiv preprint arXiv:2105.02788*, 2021.