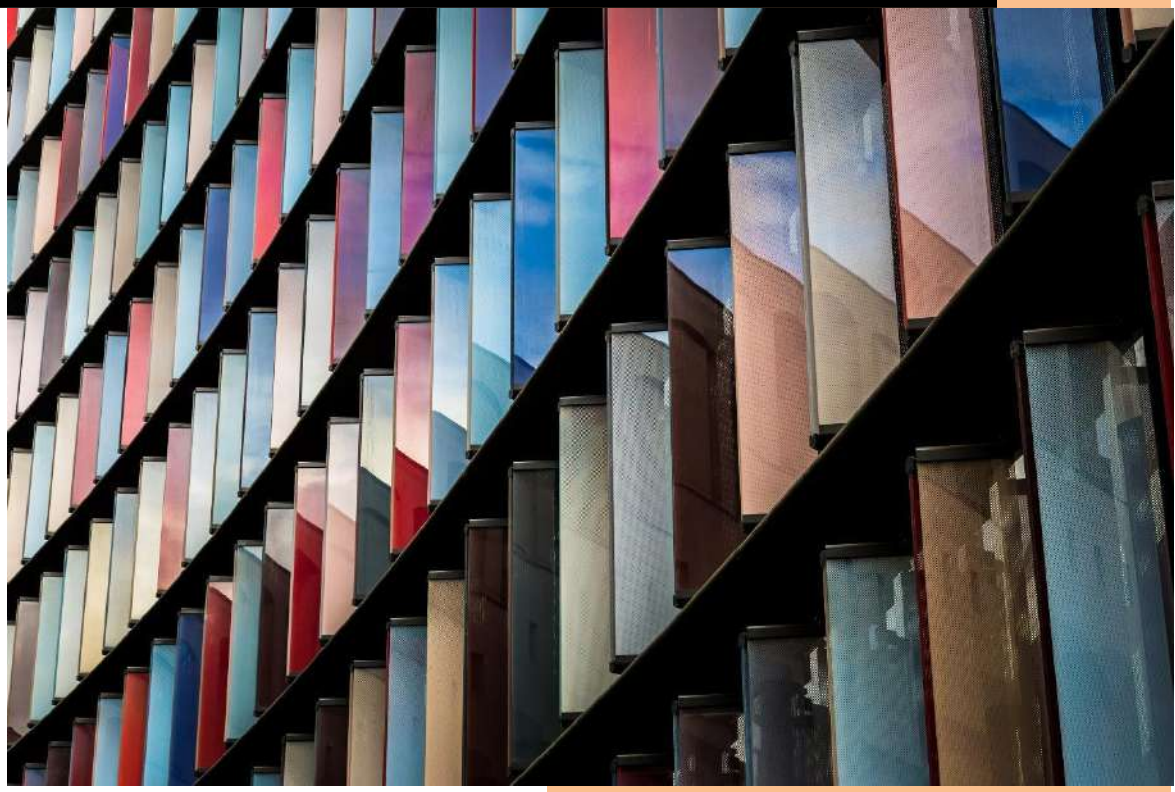
Hellenic
Mediterranean
University

3D view generation and view synthesis based on 2D data

Evgenia Lisitsa



M.Sc. in Informatics Engineering

2023

Supervisor:

Dr. A. Malamos

Table of Contents

Contents

Table of Figures

1. Introduction and Motivation

The simplification of view synthesis and complex geometry generation is a constantly evolving topic in computer science. The need to develop an efficient methodology to recreate real-world data based on a simple dataset, such as a sequence of images, has become even more crucial with the recent technological advances that allow higher-resolution graphics even on simple devices. Extraction of 3D data from a sequence of photos (photogrammetry) is a challenging, process-intensive, and extremely time-consuming procedure. Usually, photogrammetry requires taking a great number of sample photos to efficiently extract the geometry in the form of point clouds and then to be able to post-process the geometry to generate views.

On the other hand, when the target is not geometry extraction, but instead to produce a 360-degree representation of a scene, then the process may be simplified to a 360 video capture that may satisfactorily present the subject. In this thesis we research the intersection between photogrammetry and 360-degree video capturing, that is, based on a few photos captured around a subject, to create a detailed 360-degree video. Thus we focus on the creation of images directly from voxels-based data representations [15], postponing the intermediate step of extracting geometries. This problem appears in the literature as the “Novel View Synthesis” problem [57][8][7].

Attempts to approximate the solution of the “Novel View Synthesis” problem have appeared through the years using techniques such as photogrammetry, radiance fields, and neural networks[14][15][56][13][57][58]. However, there are always limitations to these methods that concern great time and resource costs, especially for the neural network training, as well as the inaccuracy and level of detail of the produced outputs [57][13]. These limitations make the view synthesis an extremely hard-to-approach technology that, however, is expected to change in the near future. Technologies, such as the Neural Radiance Fields (NeRF) (2020) [57], achieve photorealistic view-synthesis using multi-layer perceptron (MLP) Neural Networks, a simple neural network to predict complex geometry, with the cost of an extended time-consuming training process. This methodology was later advanced to PlenOctrees (2021)[8], a plenoptic octree-based[2][4] approach to radiance fields that achieve significant optimization in the training and rendering time and provide a much faster neural network training process. Even though PlenOctrees allow to reduce the neural network training time, it is still a slow process that is far from the desired speed. A similar approach using voxel grid Octrees was proposed by Hane et.al in 2017 [14] with the goal to predict object geometry using a limited dataset. This methodology uses hierarchical surface prediction (HSP), which is a framework that makes it possible to predict surfaces in high resolution from a variety of data types such as images and depth maps.

Plenoptic voxels or Plenoxels [7], proposed by the University of California, Berkeley in 2021, is one of the most recent approaches to the topic. It is a methodology that extends both NeRF and Plenoctrees and explores the idea of broadening the use of neural networks and voxel-based rendering to synthesize novel views and reduce the training and rendering time even further.

Photogrammetry can be a complex and time-consuming process, requiring significant user intervention and computation power. Therefore, there is a growing need to simplify the process and reduce the user intervention and computation power required for photogrammetry.

The main goal of this master's thesis is to understand and examine the state of the art in photogrammetry and novel view synthesis. Novel view synthesis is a problem that involves generating new images of a scene or object from a limited set of images. The focus of this thesis is to examine the creation of images directly from voxel-based data representations, postponing the intermediate step of extracting geometries. This problem appears in the literature as the "Novel View Synthesis" problem.

In addition to examining the state of the art, this thesis aims to find out the limitations that currently exist in photogrammetry and novel view synthesis approaches. By identifying these limitations, it will be possible to improve future studies and develop new techniques to reduce the user intervention and computation power required for photogrammetry. This master thesis aims to contribute to the advancement of photogrammetry and novel view synthesis by examining the current state of the art and identifying limitations in the current approaches. By simplifying the process and reducing the user intervention and computation power required for photogrammetry, this thesis has the potential to make photogrammetry more accessible to a wider audience and increase its applicability in various fields.

In this thesis, we will present and explore view synthesis methodologies such as NeRF and Plenoctrees described above. Moreover, we will meticulously examine the latest and most promising octree-based methodology called Plenoxels, which provides an efficient solution to the generation of synthetic and natural views, achieving outstanding quality that approaches the ground truth. We will evaluate and compare the results of numerous tactics used for different datasets, such as synthetic or real-world images and perform a robust examination of the produced outputs. The main goal is to define the points where this technology can be optimized and evaluated, creating a place for more innovations. During the extensive examination of the Plenoxel methodology, we will attempt to explore the capability of this method to produce alternative outputs, such as point clouds, that will make possible future advances of the technology and its applications in a variety of fields.

1.1 Spatial positioning

The ability to recognize the relation in position between objects in space is called spatial positioning. This includes the ability to identify the proportions, direction, rotation, volume, and size of the objects [3]. By combining these measurements it is possible to position and describe an object creating this way its accurate replication.

Photogrammetry can be described as a perspective projection of the scene that uses rays to identify points of intersection with the planes. The camera lens, in this case, gathers the rays from the object at one point and the photographic film or the sensor in modern cameras plays the role of the intersecting plane [3]. By recovering the rays that form the photograph, we can reconstruct the relations between objects and hence recover and reproduce the scene [3]. In other words, we can obtain each point of the scene in space by identifying the intersections of two rays [2]. This can be achieved by taking two photographs of the same object from two different angles or camera positions. Here we can add that human vision works in a very similar way to the one described above. Both eyes, due to their slightly different position, act like two different cameras, using ray intersections for the brain to triangulate the distances. This ability of the brain can be used to recreate real-world perception by projecting a slightly

different image to each eye separately. This method is called stereoscopic vision and is used with great success in Virtual Reality (VR) devices that achieve this way incredible immersion for the viewer.

Photogrammetry is a technique that gathers information and spatial measurements from real-world information by analyzing the data from a large number of images taken from a different angle of the same object. Photogrammetry techniques use perspective data, photo analysis, computer vision, and advanced computational geometry algorithms with the main goal to generate a fully textured spatial 3D model [3, 18]. A greater number of data allows to achieve precise calculations and produce detailed output. Generally, photogrammetry can be named any process that requires measurements of an object or can be photographed [3].

According to C. L. Miller et.al [3], there are three basic requirements that establish a surveying system. First, there is the scale of the objects that can be identified by the distance between two points of the object. The second requirement is that of the orientation and it is possible to obtain by the bearing of the line between two points of the scene. The third and last requirement for the surveying system is to ensure that the horizontal planes of the object in the scene will remain horizontal in the produced three-dimensional model. When working with topography, sea level can be used as a reference point to identify the horizontal line. However, when working on other objects, especially smaller scales, the process of establishing horizontal planes becomes more challenging. By achieving these three measurements, C. L. Miller et.al. insist that it is possible to reproduce the model on the *XYZ* axis without any intermediate computations.

Photogrammetry can be applied to objects of different scales, starting from large ones such as scanning the earth's surface, and going to smaller scales such as surfaces and materials [3]. The main reason photogrammetry techniques are becoming more and more popular during recent decades is the ease of use and accessibility, as well as the ability to pick up texture and color data of the object. Large-scale object reconstruction such as buildings, monuments, and landscapes using the methodology often requires the use of aerial devices and such is called aerial photogrammetry. It Very often utilizes drones that take a number of photographs of the object or environment from the air making it possible to create large-scale 3D reconstructions that are not otherwise possible to make from the ground. Photogrammetry can achieve a level of detail and realism that is not feasible using any 3D modeling tools since it is possible to capture even the smallest details of the surface of the object [1].

Figure 1 Examples of 4K renders, photoscanned materials by https://www.textures.com/

1.2 Quality of output

Quality in photogrammetry usually depends on the quality and resolution of the data, as well as the number of photos taken of the subject. A larger amount of photo data makes it possible to capture more information and allow the creation of extremely detailed results. Another factor that influences the quality of the output is the algorithms used to process the data and the machines used for it. The approach can also be resource-heavy and quite slow depending on the hardware used to process the data.

However, no matter the number of entries in the dataset, the algorithms used in data processing are the driving force of photogrammetry since most modern photography devices, even mobile phones can create high-quality photographs of objects. It is not possible to document every single detail and viewing point of the examined object, so finding clever ways to compensate for the missing data has accompanied the technology from the beginning. The main goal of the research surrounding photogrammetry is to discover more and more efficient ways to combine the smallest possible dataset into a complete highly-detailed three-dimensional object.

1.2.1 Lighting

Lighting is a very important aspect to consider in photogrammetry since it can drastically change the quality of the produced output. It is important that the subject is sufficiently lit with diffuse light with no hard shadows and no variations in brightness to achieve the best possible result. Specular highlights can also reduce the level of detail on the affected areas or can even result in a complete data loss of certain areas. Lighting is also the reason why usually smaller objects are the ones being photoscanned, since they are easy to relocate to the environment with controller lighting, allowing this way better quality and detail of the produced output [1].

For large-scale scans like buildings or locations, weather conditions play a significant role since the objects cannot be moved to a controlled lighting environment. Cloudy weather is usually beneficial since it results in diffuse light and allows to eliminate the harsh shadows and capture as many details as possible.

1.3 Applications of Photogrammetry

Photogrammetry has been used in a variety of forms during the last century with the main goal to describe and represent real-world objects and locations. Nowadays it is mostly used to recreate the shape, color, and texture of an object that can be later used in fields such as architecture, design, game development, engineering, and more [18].

1.3.1 Analog photography and photogrammetry:

In the years when complex mathematical computations were nearly impossible to achieve, analog solutions were the ones that were used to record representations of the world. The invention of analog photogrammetry was related to the need to describe geometric relationships that exist in the physical world between objects, shapes, and images. Essentially, the objects were being examined and investigated by a physical system [3]. Analog photogrammetry usually deploys film-based cameras to capture the necessary image data, which is then processed in order to identify overlapping pairs and create a 3D representation. The main applications of analog photogrammetry included map creation, object measuring, and more.

1.3.2 Photogrammetry in Engineering, and Architecture

Photogrammetry has become a vital tool in many fields as it relies on accurate measurements when building complex structures. Architecture, being a field that relies a lot on visual representation and imagination requires a lot of accurate data from the physical world. Gradually architects shifted from 2D representations to 3d as it allows more efficient representation of perspective and variety in point of view and started relying more on technology and software. In this case, photogrammetry provides architects with a lot of value as it allows sight planning, taking measurements, and accurate rendering before real-world building. It is also useful in the design process, to generate structures, buildings, and interiors. The methodology can also be beneficial when tracking the building process.

1.3.3 Photogrammetry in the movies

Photogrammetry often goes hand in hand with the cinematic industry since it gives filmmakers more freedom in the set design or planning of complex shots. Having access to more complex technology allows for the improvement of special effects and CGI and enables the creation of more sophisticated and unique environments. With photogrammetry, filmmakers can seamlessly blend computer-generated objects and environments with captured material, producing results that are impossible to achieve using only live-action filming. Objects and whole environments can also be photoscanned and then integrated into virtual sets with the use of game machines such as the Unreal Engine.
Another use of photogrammetry techniques in cinematography is motion capture, which is used to capture the motion of the actors on the scene and can be then transferred to animated characters or creatures.

The famous Matrix movie trilogy was one of the first to utilize photogrammetry in cinematography. Probably the most iconic use of this technique is the so-called “bullet scene” which gave its name to the “bullet time” effect in cinematography [55].

In this scene, a green screen,120 still cameras, and two film cameras around the actor were used to achieve a slow-motion effect while preserving the normal speed of camera movement. The images captured from the still cameras were then placed sequentially to produce the slowed movement. This effect is impossible to achieve by using regular slow-motion techniques due to the fact that the camera would have to move extremely fast [55]. However, the images from the still cameras were not enough to achieve the necessary number of frames, so the motion interpolation technique was used to produce the missing intermediate frames.

Figure 2 Shot from “The Matrix Behind the scenes” (1999)

According to Borshukov et. al [54], for the “Matrix Reloaded” a unique back-in-the-day approach was developed to create realistic renders of human facial expressions applying image-based techniques. The goal was to recreate the movement of the actors in a 3-dimensional space and then to play and show it from a variety of angles. It was achieved using ambient lighting and multiple cameras to capture the image data and then photogrammetric reconstruction was sued to identify the positions of the cameras.

A vertex of the model was projected on each of the cameras and then the motion in 2-dimensional space was tracked, estimating the 3D position at each frame using triangulation methods.

An interesting note of the method used in this movie is that no facial markers were used. Even though it made the process of feature tracking more difficult, it allowed variations of the facial texture that added to the realism of the facial rendering. To achieve even better realism, the facial animations were enhanced with highly detailed scans of the faces of the actors, which allowed the addition of the necessary details such as wrinkles and pores. Dynamic features such as additional wrinkles were added by layering on the texture map [54].

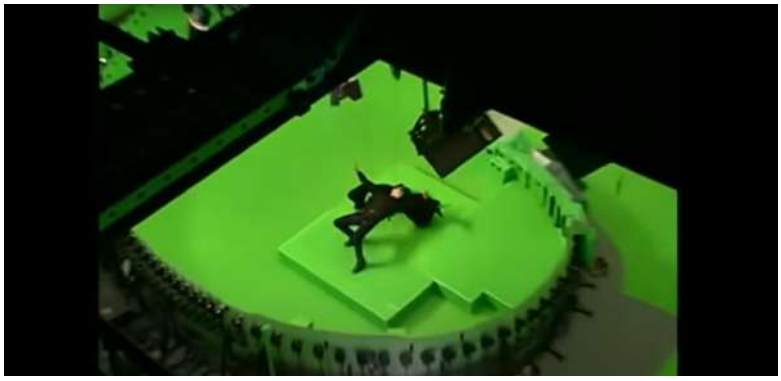


Figure 3 Shot from “The Matrix Behind the scenes” (1999)

Even though the techniques used in the Matrix movies were not entirely new and had been prior applied in cinematography on a lesser scale [55], the Matrix movies marked the point where movies were no longer limited to live-action techniques but merged with computer-generated imagery and graphics to the point where the human eye finds it impossible to notice the differences.

1.3.4 Photogrammetry in video games

Use of photoscanned assets, models, and environment Photogrammetry has been quite recently introduced in video games to create detailed and high-quality assets and realistic environments. One of the techniques used is taking overlapping photos of an object and then taking measurements from them to create 3d models and assets. It provides great realism and it is estimated that the photogrammetry approach is significantly faster than modeling 3d assets from scratch using 3d modeling software. Produced with the method of photogrammetry meshes, however, are not ready to use right away and still require modifications, refinement, and cleanup.

Another great advantage of using photogrammetry in video games is that it allows for avoiding tiling textures. Tiling textures are a technique that is traditionally used in video games and 3D modeling which uses a repetition of the same 2D texture on an object or environment. It allows the reduction of performance costs for the rendering process, however, it may be visible to the player and reduce the realism of the result. Additionally, tiling textures may create a repetitive pattern on objects, especially those with big flat surfaces like a wall or a simple terrain, that becomes boring and lacks details that add to the immersion and realism of the game.

Photogrammetry, on the other hand, allows for achieving a great realism of the produced assets since it replicates and reproduces real-world objects and environments creating unique results. The disadvantage of this technique in video games is the fact that it produces assets with expensive geometry and often huge numbers of polygons. Video games usually use a lot of assets, so an increased number of polygons significantly reduces the performance of the game and frame-drop on modern devices. Solving this issue requires heavy optimizations and polygon reduction. 3D models that are produced using photogrammetry are usually not use-ready and require clean-up and adjustments [18]. After the reconstruction process, the produced model should be optimized and issues concerning color and mesh inconsistencies should be resolved in order to create a 3D object that can be then placed in a virtual environment. This is one of the main reasons high-quality 3D models created from scratch are very often preferred compared to photoscanned ones.

Even though photogrammetry is a great tool to provide photorealistic assets, it is still mostly used for smaller props and objects since big-scale photogrammetry such as in architecture or terrains is technically challenging in terms of capturing and processing [1]. Photoscanned objects usually include natural props such as rocks and vegetation that serve as building blocks for the environment, as well as man-made objects including furniture, statues, and more [1].

Vanishing of Ethan Carter

One of the games that use photogrammetry techniques is “Vanishing of Ethan Carter”, developed in 2014 by The Astronauts company, which caused a lot of excitement among game developers and players with their groundbreaking results. For creating the game assets and environments, the

developers took multiple photos and relied heavily on photogrammetry, which was previously used almost solely for character design in video games [1].
The Vanishing pushed the previously not well-known technique of photogrammetry to new standards. It helped to achieve significant advances and improvements in the following years making more and more game developers switch to photogrammetry techniques to produce their assets.

Figure 4 "The Vanishing of Ethan Carter" by the Astronauts https://www.theastronauts.com/2014/03/visual-revolution-vanishing-ethan-carter/

1.4 Methodology for datasets production

Currently, one of the most widely used approaches in photogrammetry involves utilizing a vast collection of photographs captured from various perspectives of the subject matter, with the goal of comprehensively capturing all its details from numerous angles.

Categories of photogrammetry include aerial photogrammetry which is a convenient method for large-scale scans such as buildings, roads, and hard-to-reach locations [1]. It is one of the most common techniques involving aircraft and drones to capture overlapping data. It usually uses vertical or tilted photographs to capture the necessary input. Information about location coordinates and distance is captured as well and used to recreate accurately scaled output. This method is often used for maps, topography, and land surveying since it allows accurate and large-scale data capture.
Another category includes photo scanning and ground photogrammetry which requires the capture of data using cameras. It is necessary to acquire information about all the possible angles of the object in order to produce a high-quality output.

2. Background

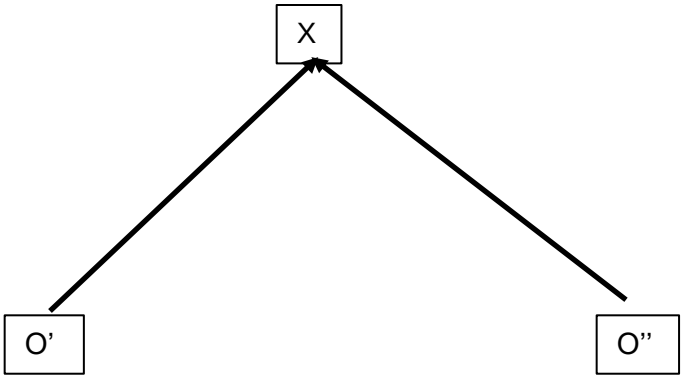
2.1 Dataset collection

Photogrammetry is a complex process and requires a variety of measures to be taken at all stages starting from the initial data collection. To ensure accurate and reliable results it is necessary to eliminate as many inconveniences that can occur due to the limitations of the hardware used to capture the photographic dataset.
During the process, calibration errors may occur due to measuring instruments, which require corrections to be applied. This can be done by using grid intersections and comparing the values to known and established measurements [2]. Deformations and distortions can also be present due to dimensional changes in the photographic data. It is necessary to calibrate and solve these issues to produce the most accurate results possible. Moreover, it's essential to consider lens imperfections and distortions produced by the instruments used to capture photographic data. Factors like image quality, sharpness, and focal point can significantly influence the output of the scanning.

Chromatic aberration is another phenomenon that can affect the quality of the produced model and complicate the processing phase [2]. This aberration is caused by the refractive properties of the wavelength of light, where shorter wavelength radiation, such as blue light, has a higher refraction compared to the longer wavelengths of red light.
Addressing these factors and implementing quality control measures can help ensure that the results of photogrammetry are accurate and reliable, providing valuable insights and solutions in various applications.

2.2 Fundamental Matrix & Epipolar Geometry

Epipolar geometry is often used to describe geometric relations in pairs of images and it allows us to predict the corresponding points in the most efficient way. It is possible to reduce the search space from 2D which is the whole image to a simple one-dimensional line. This way the computations become faster and reduce the possibility of a mistake.
In computer vision, when we have a point x of one image, it is often necessary to find the corresponding point *x'* in the second image. So usually, we have two cameras located in their projection center and a point where the rays sent from both cameras intersect.



When we have a point in *image1* we want to find the corresponding point in *image2* and the goal is to find the optimal way to locate it without searching the whole image. It is possible to map one point of the first image to a line in the second image, reducing this way the area that needs to be searched in order to identify the corresponding point.

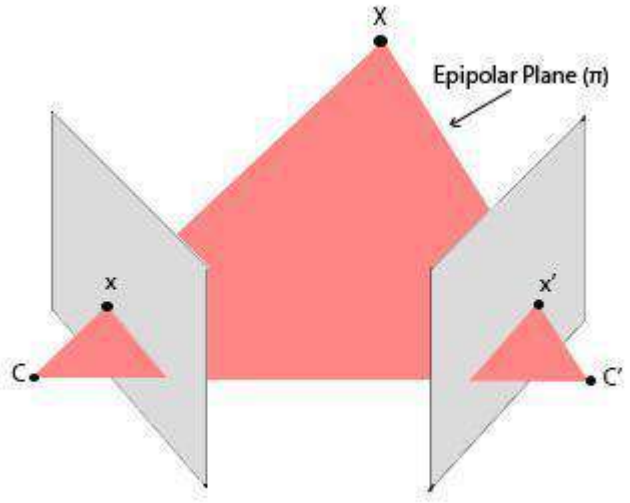


Figure 5 Epipolar plane

The big triangle that we see in the image above is called the **Epipolar plane** and represents the plane that occurs when we connect the center of the two cameras with the examined point. Given a particular point in the *Figure 5*, we want to find a line, called the Epipolar Line, in *Figure 5* on which lies the corresponding point of interest. This way it is possible to find the position of a specific point on multiple images.

If we have a point on the first image, we can be sent a ray from the camera through that point which is passing through the 3D world point. A point on a line can be found using the Least Squares method which is a technique for state estimation. To use this method, it is necessary to be able to map the parameters or the unknown values into a predicted observation which can be done using the observation function. This function takes a state as an input and produces predicted observations.

$$f(x) = \hat{z}$$

Where *f* is the observation function, *x* is the state, and $\hat{z}$ is the predicted observation.
The next step is to compare the predicted observations with the actual observations that we have and then adjust the parameters of the observation function to produce the predicted observations that are as close as possible to the actual values. This way, the generated predicted values will be in line with the actual values. The error that we try to minimize in this method is the sum of the squares of those errors where the error is the difference between the values we expect to observe and the actual values.

$$\min_{x} \sum \left(z - f(x)\right)^2$$

Where *x* is the state and $z - f(x)$ is the error
This method is used in a large variety of problems such as camera calibrations, bundle adjustments, and in our case, photogrammetry. In the basic form of the Least square method, we assume that there is a linear dependency between the actual values and the predicted observations.

If we have a camera point *x*, we can find another point *x'* that lies on the same ray sent from the initial 3D point, using the Least square method. It is possible then to create a projection of this point on the second image plane and get a second point. Given these two points, it is possible to draw a line and assume that the point of interest, which is the point that corresponds to our initial 3D point on the first image lies on this line. The projection of the Camera point to both images creates the Epipolar plane. In case we want to find a different point on the planes, we can rotate the epipolar plane [4].

In case every point *x* on the image corresponds to an epipolar line *l'* on *image 2*. Given the *x'* and ***e'*** we can find the *l'* using cross product.

$$l' = e' \times x'$$

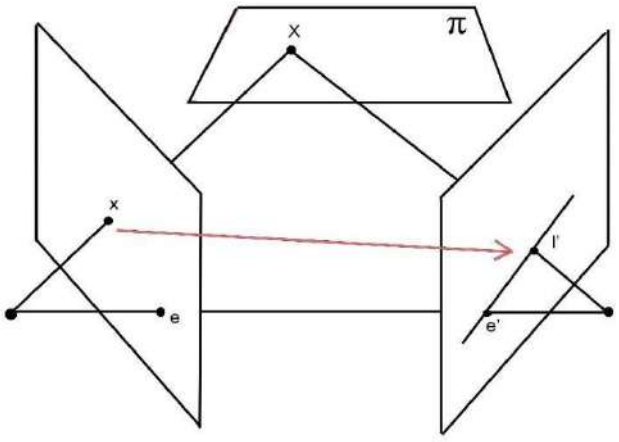


Figure 6 Epipolar line

In the 2-dimensional space this equation has the format *(x, y, w)* in a homogeneous coordinate system where ***e'*** is a vector with three elements:

$$e' = [e_1\ e_2\ e_3\ ]$$

Then we can create a skew-symmetric matrix and then multiply it with the *x'* using the cross product:

$$[e]_x = [0\ -e_3\ e_2\ e_3\ 0\ -e_1\ -e_2\ e_1\ 0\ ]$$

$$e' \times x' = [e] \times x'$$

Note: determinant = 0

$$|[e]x| = |0\ -e_3\ e_2\ e_3\ 0\ -e_1\ -e_2\ e_1\ 0\ | = -e_3(e_2e_2) + e_2(e_3e_1) = 0$$

Using homographic transform Hπ from the image above that can produce *x'* from *x*

$$x' = H_\pi x\ l' = [e] \times H_\pi x\ l' = Fx$$

Fundamental matrix *Fx* gives you the relation between a point and an epipolar line. So given a point, we multiply it by the fundamental matrix and get an epipolar line. *l'* is also formed by two points ***e'*** and *x'* which is the projection of the point on the epipolar line.

$$l' = e' \times x'\ l' = [e] \times P'P^+x\ F = [e]_xP'P^+$$

Where *P* is the projection matrix of *camera 1* and ***P'*** is the projection matrix of *camera 2.* P+ in this equation is the Least Square estimate and then after projecting the point *X* on the image plane we get the *x'* point. This way we manage to create a map between *l'* and *x*. ***PP'*** is the ***Hπ*** matrix from the previous equation [4].

2.3 One step further: Novel View Synthesis

Today we have achieved to a great point the recognition of the relations between points from an image dataset and recreating the 3-dimensional representation of the object using these points. However, the next question comes to mind: what is in between? What if it was possible to predict the data that exists where there is a lack of data? Well, we are on the verge of great change when it comes to answering these questions and this is where the novel view synthesis comes in.

Novel view synthesis is a process of creating additional views based on a photographic dataset. It allows the reduction of the number of data needed for the photogrammetry to render volumetric geometry. Classic methods to achieve novel views are mesh-based and represent a 3D scene with surfaces, creating Lambertian or diffuse scenes as well as non-Lambertian surfaces. It is a method mostly used these days in computer graphics since they are easy to render. However, the optimization process of this method in complex geometry is often challenging.

Another method to achieve novel views is photogrammetry mentioned earlier in this work, which allows highly detailed rendering. The disadvantage of this method includes the limitations of the viewing angle, high storage demands, and challenges in the editing of the scene.

Volumetric rendering, on the other hand, includes methods with a long history, using volumetric representations of a scene with multi-plane images (MPIs) and voxels is an alternative method for rendering. It allows the production of renders with a high level of detail but is not broadly used due to high memory demands.

2.4 Voxels

Voxels are volumetric representations or volumetric elements that represent the smallest pieces of data that a computer can render, similarly to pixels, but in a 3-dimensional space [9]. The concept of voxel-based geometry has been around for many decades, however, due to hardware limitations and rendering complexity, their use was mostly limited to medical image synthesis [10]. In recent years, with the advances in computer graphics and rendering methods using GPUs, attention to voxels has increased significantly. It is related to the fact that voxel-based geometry can produce more realistic renders and provide significantly improved solutions for complex geometry, compared to the polygon-based meshes that create only hollow models and require additional information to correctly display the data.

Global illumination, which provides realistic lighting to 3D scenes is another area where voxels have become an important research field. Companies such as NVidia and Unreal technologies, leaders in graphics and game development, have invested greatly in voxel-based techniques and have published a number of works in recent years, pushing the technology even further [10].

Cube-based visualization is one of the most common approaches to voxels, however, it is not necessary for the voxels to have that geometrical structure. In this technique, voxels are represented as cubes with predetermined sizes which allows a simpler visualization and easier understanding of the concept of voxels [10]. This approach, however, requires a lot of resources, especially when it comes to lighting due to the fact that all the cube's geometry should be rendered. This means that hidden geometry is rendered as well, making the process extremely computationally heavy. To solve this, ray tracing and path tracing techniques are often applied to cast rays at the target and examine if the geometry is visible to the camera [10]. Ray tracing, however, is also a very slow and computationally heavy process that requires a lot of optimization techniques.

Even with the advances in GPU rendering, voxel-based models are still very costly in terms of resources creating a need for techniques that will allow faster real-time rendering. Even a quite simple representation requires a lot of data and a large number of voxels. The solution to this problem can be approached using GPU data streaming to increase memory use. This technique is used to keep only certain data that is needed for the current part of the representation in the GPU memory.

Other approaches that are often used with voxels include tree structures that help to increase the details of rendering data only in the areas with a lot of information. This allows us to significantly reduce rendering costs. One such technique is octal tree voxel representation or octrees [11] which we will examine in detail later in this work.

Octree is one of the common tree data structures that employ the partition of a 3-dimensional space. Its structure consists of nodes with the root one describing the cube of interest in 3-dimensional space and the internal nodes containing the description of the subdivision of the current node [14].

The sparse octree data structure is also broadly used which allows for storing voxel data in the memory of the GPU. In this data structure, each node represents a voxel which can then be further subdivided

into smaller voxels including both parents and children in the data structure. The advantage of this approach is that it allows the reduction of the memory footprint and can be efficiently applied in ray casting methods [15].

Voxel representation usually contains the scale, position, and color data. Additionally, it can be used to store information such as normal vectors which can be highly beneficial for scene lighting and provide this way more advantages compared to polygon meshes. Due to the fact that all the necessary information can be stored directly in the voxels, it can eliminate the need for textures and mip mapping. This way it is possible to produce physically correct results that can be applied in computer visualization. Another advantage of voxels is that they can represent atoms, making objects destructible. Applying raycasting techniques to create a voxel hierarchy is considered to be efficient and simple since little data needs to be carried with the ray during the raycasting process [15].

Unfortunately, voxel-based rendering is yet not broadly supported by GPU manufacturers and hence requires custom technology for visualization.

2.5 Neural networks for computer visualization

As has already been mentioned, view synthesis, and especially novel view synthesis require a lot of computational resources to process a huge amount of photographic data. One of the methods that have been suggested is to utilize neural networks to predict volumetric representations, with the aid of techniques such as voxel grids [7]. Most methods so far, typically make use of a 3D CNN for the volumetric predictions, rendering then novel views using rays in a differentiable, easily optimizable way. Unfortunately, these methods are extremely slow and require significant storage space for computations and storing the voxel grids.
Another approach is to use continuous mathematical functions to represent the volumetric shapes using neural networks to represent the surface of an object in a 3D space. This method can achieve impressive results regarding speed and storage requirements, but it cannot achieve the photorealistic result that voxels can provide.
Neural Radiance fields or NeRF, introduced by Yu et. al. [7] use neural networks to encode volumetric representations of a scene in a continuous way, avoiding this way the limitations of discretized grids and achieving great results compared to previous works done in the field.
The general idea is to send a ray from the camera, in a similar way it is done in raytracing methods until a hitting point is found. The location and the viewing direction of each point are then used as input in a neural network which outputs the color and the opacity of the point. Next, the ray from the camera is marched from the back to the front and it is possible this way to compose the image by combining the density and the color of the points on top of each other.

Continuous 5D function
The key feature of the NeRF approach is the representation of a scene as a continuous 5D function which includes the *xyz* coordinates and the direction using a neural network. This is done using a

simple fully connected network with 9 layers and 256 channels on each layer. The neural network output gives both a volume density of the particles σ at a specific location and RGB color representation of the radiance of the particles in the location of the input and its direction.

$$(x, y, z, \theta, \phi) \rightarrow F\Omega \rightarrow (r, g, b, \sigma)$$

Where *x,y,z* is the spatial location, θ,φ represent the viewing direction, $F\Omega$ is the fully-connected neural network with 9 layers and 256 channels, *r,g,b* represent the output color and σ is the output density.

Summary of the neural network:
Input: coordinates in space and the looking direction of the camera
Output: Density of a small point in space and its color value. The density value ranges from 0 where the point is empty and goes towards infinity in the case where it is fully dense.

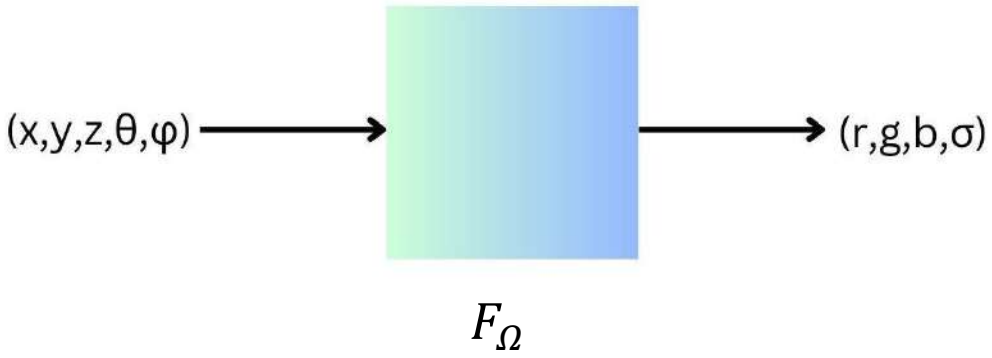


Figure 7 Neural Network summary

Volumetric rendering is used for novel view synthesis. Having the viewing direction of each point allows the creation of representations of non-Lambertian surfaces, meaning that the reflection on the surface changes based on the direction it is examined, creating extremely realistic results in terms of lighting and texture.

Optimization in this method is one of the key points that require a lot of attention since it is very performance costly. Optimization is done with the use of rendering loss without prior training of the neural network. This can be done by optimizing the neural network's parameters using stochastic gradient descent. Training the neural network this way allows the reduction of the points on locations with lower volume density over network iterations and increases the data and the color accuracy where the density is higher. Manipulating the input data allows the network to represent more details in the locations where it matters more, allowing it to represent more detailed geometry and complex scenes. Positional encoding is also applied to the inputs of a neural network to recover frequency details.

2.6 Plenoctrees

After the significant achievements with the Neural Radiance Fields, Yu et. al [11] took a step forward, proposing a method of rendering NeRFs in real-time using an octree-based representation of the 3D environment, so-called Plenoctrees. The most crucial element that the team tried to resolve was to achieve view-dependent effects and reflections on a surface while reducing the use of neural networks that require a lot of time to train.
Plenoptic voxels or Plenoxels [7], proposed by the University of California, Berkeley in 2021, is a methodology that extends both NeRF and Plenoctrees and explores the idea of broadening the use of neural networks and voxel-based rendering to synthesize novel views and reduce the training and rendering time even further. Plenoxels create a volumetric representation based on 2D images using a sparse voxel grid [7][57] to represent the geometry and spherical harmonics to represent the color of each point, achieving this way to synthesize photorealistic novel views with an excellent level of detail. Trilinear interpolation is used to create a continuous geometry and create a plenoptic function that represents it and allows the reconstruction of geometry based on a limited dataset. This approach creates an opportunity to reduce the initial images of the scene needed for training. Volumetric rendering techniques are then used in Plenoxels to reconstruct the geometry from the voxel grid [57], using the spherical harmonics stored in the grid to accurately produce the colors on the surface. This way, it is possible to achieve view-dependent rm meaning that the reflections on the surfaces of the objects are realistically depicted depending on the viewing direction. Additionally, state-of-the-art optimization techniques, such as Total Variation Regularization [59] and coarse-to-fine approach are used to reduce the noise of the generated output and reduce the training and rendering time. Regularization techniques in computer vision are methods used to prevent overfitting in machine learning models and These techniques lead to a significant improvement of the methodology performance, optimizing memory requirements and allowing the reconstruction of a smooth and detailed output.
These impressive results are made possible using CUDA [60][61], developed by Nvidia, a programming interface and API that allows the deployment of parallel computing on a graphics processing unit (GPU), saving this way precious training and rendering time. CUDA allows the use of a variety of industry-standard programming languages such as Python and C++ to perform complicated calculations and large data processing, making the process accessible to experts from different fields and backgrounds. Additionally, Anaconda [62] software is used to achieve faster Python computation such as those performed by the numpy library, and is broadly used in Machine Learning tasks.

Neural networks were mostly used to predict the representation of the radiance of a surface point using spherical harmonics, removing the viewing directions from the input of the neural networks. Spherical harmonics in this case are a basis used when defining functions on a surface of a sphere. This allows for a significant reduction in the training time when combined with appropriate optimization techniques maintaining the level of detail on the novel views and on the final render as well.
The key point of the Plenoctrees method is the use of hierarchical volumetric representation of a 3D space and pre-sampling the NeRFs into vie-dependent columns called Plenoctrees. More specifically, Yu et. al [11] in their work propose a method to use sparse voxel-based trees where every leaf stores

data about the appearance using RGB values at a specific location using spherical harmonics (SH) and density of a point that is necessary to model the radiance. Spherical harmonics, in this case, allow the retrieval of the view-dependent color of a surface. This is achieved by modifying the Neural Radiance FIeld network to predict the spherical harmonics, producing coefficients for their functions, and then storing these values into plenoctrees, instead of using RGB values. After that, the values that are stored are optimized allowing the production of high-quality renders.

2.6.1 Neural Radience Field in Plenoctrees

The first approach to neural radiance fields (NeRFs) was to produce 3-dimensional representations that render novel viewpoints and allow the capture of continuous geometry and achieve view-dependent effects. A multilayer Perceptron (MLP) is used in this method which is a class of artificial neural networks where nodes do not form cycles in their connections. The input to the MLP in Plenoctrees consists of a point position (*xyz*) in the space and a viewing direction d, producing the RGB color estimation values and the density of the point as an output. In their work on PlenOctrees [11], Yu et. al used a modification of this method, called NeRF-SH, to allow the NeRF network to produce spherical harmonic coefficient *k* instead of the RGB color values. This approach allows for the removal of the viewing direction of the point from the input to the network and still predicts the color values and produces view-dependent results with fewer network evaluations.
After the NeRF-SH network training, the output is converted into a sparse octree representation with density and SH coefficients at each leaf that allows real-time rendering. Rendering of the PlenOctree is done by determining intersections between rays and voxels in the octree data structure, producing segments of continuous color and density values. The achievement of this method is the ability to skip larger voxels in fewer steps, while still rendering the smaller ones.

Conversion from NeRF-Sh to PlenOctrees
The process of conversion includes three steps.
1. Network evaluation on a grid to obtain density values on a uniformly spaced 3-dimensional grid.
2. Filtering the grid from step 1 using a threshold to produce a sparse voxel grid sufficient for scene representation, removing voxels that have a lower weight than the threshold which are those that are non-visible.
3. The final step is to sample points from each of the remaining voxels and produce their average values and obtain SH-coefficients that are then stored as leaves in an octree. The Octree stores a vector with spherical harmonics coefficients and a density of each of the RGB channels.

To visualize and make the results of the research available to the public, Yu et. al [11] created a web-based renderer that allows to view the result of the PlenOctree method and interact with it in a web

browser. It is done by rewriting the CUDA-based PlenOctree renderer into a WebGL fragment shader and compressing the result so it can be viewed in a browser.
To convert the output produced by NeRF into a mesh, the technique called Marching cubes was used. Marching cubes is an algorithm used in computer graphics that allow the production of a polygonal mesh from voxels which runs through the voxels grid and takes eight neighbors at a time, forming a cube and then determining the polygons that are required in order to produce the surfaces that pass through each cube. The polygons produced are then formed into a mesh.

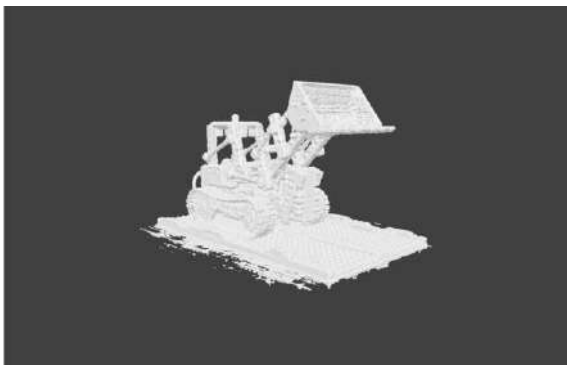


Figure 8 Mesh poduced using PlenOctrees algorithm *https://alexyu.net/plenoctrees/#demo-section*

The result of this research allows the rendering of NeRFs in real-time producing non-Lambertian surfaces and view-dependent effects with high compression degree maintaining, however, the level of detail.

2.7 Plenoxels: Evolution of the Pleoctrees into radiance fields without neural networks.

The use of Neural networks for producing novel views [8] showed promising photorealistic renders, allowing to capture of the geometry of a 3D scene with impressive quality. However, this method cannot be easily used due to the time-consuming neural network training process which can require more than a day to complete on a single GPU.
Yu et. al [7] continued their research and proposed a new method to solve the issue and to train radiance fields without the use of neural networks maintaining, however, the high quality of Neural Radiene fields (NeRFs) [8] to render novel views from calibrated 2D photographs. This method is called Plenoxels due to the use of plenoptic volume elements that have the ability to describe the amount of light in a space and all the possible rays of that space are given by a 5D plenoptic function with a magnitude of each ray described by radiance. Plenoxels use a sparse voxel grid, a computer graphics rendering technique similar to sparse voxel Octree (SVO) used in PlenOctrees [11] where each voxel stores data about the opacity and the spherical harmonics. The spherical harmonic coefficients are then used to produce a 5D plenoptic function that represents the light intensity from every viewing position and direction in 3D space, using interpolation techniques.
The core model of Plenoxels is bound to a voxel grid, however, it is possible to render unbounded scenes using normalized device coordinates (NDC) that allow representing a scene in a screen-independent coordinate system using a cube with *xyz* coordinates and range from -1 to 1. Another way to achieve the rendering of unbounded scenes is by creating a multi-sphere image around the grid to encode and represent the background in 360 degrees scenes.

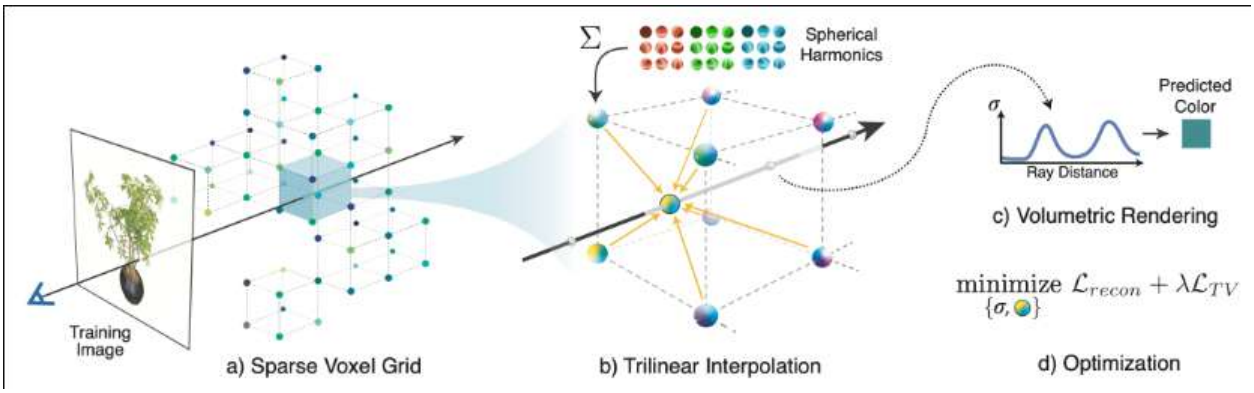


Figure 9 Plenoxels Algorithm summary, https://alexyu.net/plenoxels/

Some of the key components of the Pleoxels approach:

1. **Data representation**
2. **Forward modeling** uses a model to simulate the output.
3. **Regularization** function is a method that uses penalty values in the error function.
4. **Optimizer** to improve the produced output.
5. **Total Variation** to remove unnecessary details and noise from the output preserving the sharp
6. **Image-based modeling** for plenoptic function reconstruction based on image data set
7. Custom **CUDA implementation**

2.7.1 Differentiable volumetric renderer

Differentiable volumetric renderer is a method that allows 3D geometry reconstruction from 2D image data set similar to the one used in Neural Radience Fields to achieve continuous geometry and texture representation and can be used in the single and multiview 3D reconstruction. It is used to obtain analytic gradients for a depth map maintaining the space and texture parameters [12]. It allows the creation of a differentiable renderer for texture and shape representation from a multi-view image dataset. The surface prediction is performed using occupancy evaluation for each camera matrix and Texture information is used to enhance the visual results [12]. This method usually includes a 2D image encoder which then allows the production of a 3D points dataset. This point dataset is then usually parsed to a neural network that predicts the density and the color values of each point.
The next step is to use a ray marching technique and a threshold in order to identify the density changes and remove those that are below the threshold value.

Integration of the samples taken along the ray is done:

$$\hat{C}(r) = \sum_{i=1}^{N} \quad T_i(1 - exp\ exp\ (-\sigma_i \delta_i)\ )c_i \quad (1)$$

$$Where\ T_i = exp\ exp \left(- \sum_{j=1}^{i-1} \quad \sigma_j \delta_j \right) \quad (2)$$

Where *Ti* represents how much light is transferred through the ray *r* to a sample *i.*
exp(...) is the light contribution of sample *i*
c_i is the color of sample *i*
$(1 - exp\ exp\ (-\sigma_i \delta_i)\ )$ shows the light contribution of sample i
σ_i shows the opacity of the sample i
δ_i represents the distance to the next sample
This formula assumes single scattering and that the values between the samples are constant, making it an approximation. However, the formula is differentiable and allows to update the 3-dimensional model taking into account the error of each ray.

2.7.1 Sparse 3D grid with spherical harmonics

In the Plenoxels method, a sparse voxel grid is used in a similar way as in NeRFs to the geometry model. However, to make the process of trilinear interpolation easier, Yu et. al [7] at their work propose the use of a dense 3-dimensional index array that stores pointers to a separate array that stores data about the occupied voxels. Only the occupied voxels then store data about the opacity and spherical harmonic coefficients for each of the color channels.
Spherical harmonics in this case are used as an orthogonal base that allows the definition of functions over a sphere. The lower spherical harmonics coefficient values encode smooth and more Lambertian variations of colors and the higher harmonics values are used to encode specular effects. To produce the color of sample Ci, Yu et. al use a sum of the harmonics for each color channel, evaluated at the viewing direction and weighted by optimized coefficients.
The color and the opacity of each point along the ray are computed using the trilinear interpolation technique of the opacity and spherical harmonics coefficients of the 8 neighboring voxels. This allows to increase the resolution of the produced output and to create a continuous function that is then used in the optimization step. Most of the gaps between neighbors are closed this way in the Plenoxels method when using trilinear interpolation.

2.7.2 Neural volumes

The neural volumes are used to transform the input image dataset into 3-dimensional volume representations. Lombardi et. al proposed a method to generate 3D representation by trying to match the input images in the best way possible [13]. The approach assumes that the geometry of a scene consists of flat surfaces which allows it to capture highly detailed geometry and materials, overcoming the limitations of classical multi-view synthesis (MVS) methods. The method is based on volume ray marching methods but introduces a warp field that allows encoding the locations within the voxel grid into a 3D warp field and then using it to sample color values and opacity. This approach allows to increase the resolution of the voxel grid and to model motion.
Lombardi et. al [13] also propose a limited use of machine learning methods to produce RGBA volume and then use marching algorithms without learning parameters to produce volumes that allow advanced viewpoint generalization.
One of the methods proposed to approach a volume function is that of the Multilayer perceptron (MLP) decoder that avoids the limitations of the voxel grid resolution and the storage space requirements.

However, this approach requires MLP evaluation at every step in the ray marching along the ray which restricts its use in real-time applications. Another decoder that can be used to approach this issue is that of a Voxel grid. Voxel grid decoders model volume function as a discrete 3D voxel grid instead of trying to model the entire volume as it is done when using MLP decoders.

Based on the differentiable volume rendering introduced in NeRFs and then advanced in the PleOctrees method, Plenoxels extend these approaches to execute end-to-end sparse voxel grid optimization using spherical harmonics which performs the training and produces the outputs two orders of magnitude faster compared to the NeRF method.

2.7.3 Plenoxels approach

The Plenoxels approach uses a sparse voxel grid which is a method that uses ray tracing or raycasting techniques for the octree data representation. In this method, each of the occupied voxels stores the opacity and spherical harmonic coefficients for each of the color channels. At an arbitrary position, the color, the opacity, and the viewing direction are computed using trilinear interpolation which is a method that allows the estimation of new data points on a 3-dimensional grid based on a set of known data points in the space.

The Coarse to Fine technique is used in Plenoxels to achieve high-resolution output. The approach of this method is to start with a dense grid of lower resolution, pruning the voxels that are not needed and then gradually increasing the resolution, repeating this process. In each evaluation the voxels that remain are subdivided into half of their dimension, achieving this way further optimization in each step. Voxel pruning is done in a similar way it is done in PlenOctrees, using a threshold on the maximum weight of each voxel for each of the rays.
Another issue that needs to be solved is that of naive pruning in trilinear interpolation. Since the values of the voxels interpolate with their neighbors, pruning should be performed only in case both the voxel of interest and its neighbors are not occupied.

2.7.4 Optimization

One of the most crucial steps for the performance of the Plenoxels is that of optimization. It is done by using mean square error (MSE) on the spherical harmonic coefficients and opacity with total variation regularization (TV). Mean square error is a technique that finds the average value of the squares of the errors, where the error is the difference between the expected value and the actual output, allowing the measurement of the quality of the produced output. MSE always has a positive value and when its value decreases, the error approaches zero.

MSE:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$$

Where:

n is the number of data points
Y_i are the observed values
$\hat{Y}_i$ are the predicted values

Loss function:

$$L = L_{recon} + \lambda_{Tv} L_{Tv}$$

L_{recon}*:* MSE reconstruction
L_{Tv}*:* Total variation regularizer

$$L_{recon} = \frac{1}{|R|} \sum_{r \in R} \|C(r) - \hat{C}(r)\|_2^2 L_T v$$

$$L_{TV} = \frac{1}{|v|} \sum_{v \in V} \sum_{d \in [D]} \sqrt{\Delta_x^2(v,d) + \Delta_y^2(v,d) + \Delta_2^2(v,d)}$$

$\Delta_x^2(v,d)$: the squared difference between the dth value in a voxel $v := (i,j,k)$
(i + 1, j, k): Dth value in a voxel

In the Plenoxels method the weights and values of the Spherical harmonic coefficients are fixed for each of the scene types: bounded, forward-facing and 360-degree.

Stochastic sampling is a Monte Carlo technique that allows a nonuniform sampling of the image and is used in the Plenoxels method on the rays to evaluate the Mean Square error and on the voxels to evaluate the Total variation in the optimization step. The learning rate which controls the changes of the model according to the error values is fixed for all the scene types in the main experiment part.
In approaches such as Plenoxel, the optimization step is extremely challenging due to the huge number of values that need to be optimized (high-dimensional) and the poorly conditioned objective. Second-order optimization algorithms are usually used to solve poor conditioning, but Yu et. al propose the use of Root Mean Squared Propagation (RMSProp) which extends the gradient descent method that adapts the step size of each of the parameters and restricts the rapid changes in the vertical direction allowing to increase the learning rate and taking bigger steps in the horizontal direction.

2.7.5 Unbounded Scenes

One of the major goals of every view-synthesis technique is to extend it beyond synthetic scenes to real forward-facing or 360-degree scenes. For forward-facing scenes, Yu et.al [7] use a similar voxel grid to the one used in NeRFs [8] with normalized device coordinates. When facing 360-degree scenes it becomes a challenge to render the background and to solve this issue in the Plenoxels method, a multi-sphere image (MSI) is used for the background model instead of the sparse voxel grid that is used for the foreground. The background model

uses trilinear interpolation between the spheres with the same effect as the foreground model, with the only difference being that the voxels are wrapped on the spheres using projection methods. For memory conservation in the background, only the RGB channel values for the colors are stored using an opacity threshold.

2.7.6 Regularization

Regularization is a process that allows the addition of necessary information in order to solve ill-posed problems in the optimization step and avoid overfitting. Total Variation regularization which is used in Plenoxels assists smoothness and is used in all scene types with additional regularizers suitable for each type.
In forward-facing and 360-degree scenes, sparsity prior is used which gives priority to zero values using Cauchy loss:

$$L_s = \lambda_s \sum_{i,k} log\ \left(1 + 2\sigma\left(r_i(t_k)\right)^2\right)$$

$r_i(t_k)$: opacity of sample *k* along a ray *i*

In each evaluation of optimization in forward-facing scenes, the loss term is evaluated at each of the samples. This approach is similar to the sparsity loss that is used in PlenOctrees which urges the voxels to remain empty and saving this way memory and reducing the quality loss during the upsampling step. During the Upsampling step, the final goal is to increase the resolution to match the original input data.
For the most advanced scenes, which are the 360-degree ones, an additional beta distribution regularizer is used which employs the probability distribution function and allows the computation of the probability of possible outcomes on an interval [0,1] for each subset of the dataset or minibatch. Using this technique allows the creation of a background-foreground decomposition with the foreground being empty or fully opaque.
Beta loss:

$$L_\beta = \lambda_\beta \sum_r \ (log\ log\ (T_{FG}(r)\ ) + log\ log\ \left(1 - T_{FG}(r)\right)$$

r: training rays
$T_{FG}(r)$ foreground transmittance with values in the range [0, 1]

2.7.7 Plenoxels Implementation:

Yu et. al [7] provide a custom PyTorch CUDA library for differentiable volume rendering and a slower JAX implementation. The scenes are separated into three categories: Synthetic, real forward-facing, and real 360-degree scenes. Each one uses a similar base but requires a different approach due to the specific characteristic of each type.

Data tables are used with each entry to store data concerning density and Spherical Harmonic coefficients for RGB color channels. Another data structure that is used in this approach is a dense grid, with each cell having a value of null or including a pointer to the data table. This approach to data structure allows the enhancement of the trilinear interpolation process and maintains sparsity, which is beneficial due to the large storage requirements of the method to store SH coefficients and other values. Batches of 6000 rays are used and then the optimization is done with the RMSProp method.
CUDA rendering allows parallelization across rays, spherical harmonic coefficients, and color values allowing faster iteration and creating highly merged access to the global memory.

Synthetic scenes: These scenes include 100 training photo datasets for each of the 8 objects that are included. It is the same dataset that was used in NeRF experiments with a resolution of 800x800 pixels with known camera positions that are randomly distributed in the upper hemisphere while facing an object.
For the synthetic scenes, the initial resolution is 256^3, then voxels are pruned and the upsampling process follows after 38400 steps, which is similar to 3 epochs to a resolution of 512^3. The optimization process takes 128000 steps, which is similar to 10 epochs. Epochs are used as a term from machine learning to indicate the number of evaluations performed during the training process.

Real forward-facing scenes: The above method is extended to forward-facing scenes with the help of normalized device coordinates (NDC) which are screen-independent and can be used to define the viewing volume in a cube with *xyz* coordinates between [-1, 1]. TV regularization is also used in this method with adjusted stronger weight in the optimization process. Each scene includes 20 to 60 photo datasets which are taken by a cellphone with a resolution of 1008x756 pixels. From each dataset, 7 out of 8 images are used for the training process and the remaining one is used as a test dataset.
For forward-facing scenes, the resolution on the z-axis remains relatively low since the Plenoxels method does face these types of scenes as more 2.5D than 3D objects. It gives more weight to the resolution on the x and y axis to produce highly detailed results with relatively smaller depth. Starting with the resolution of 256x256x128 pixels, after pruning and upsampling after 38400 steps the resolution becomes 512x512x128 pixels and the process is repeated until the resolution of the *x* and *y* axis matches the resolution of the initial dataset, remaining unchanged on the *z* axis. Optimization of total 12800 steps is applied to produce the final grid.

Real 360-degree scenes: The most important step is to be able to extend this method to unbounded 360-degree scenes which allows full freedom of the rendering objects. In Plenoxels, it is achieved by using a sparse voxel grid for the foreground model and surrounding it with a multi-sphere image (MSI) that was mentioned in the background model. For each one of the background spheres, the data is represented as a simple voxel grid using trilinear interpolation to achieve continuous output.
Starting with the smallest resolution compared to the previous 2 types, that of 128^3 pixels, a similar process of pruning and upsampling is followed with 25600 steps between each

upsampling until it reaches 640^3 pixels. The optimization process then follows 102400 steps. The difference in the 360-degree scenes is that in the optimization of the background, the thresholding method is used as well as exponential decay, leaving the coarse-to-fine technique. This allows the faster optimization of the background at the beginning of the process.

3. Research & Methodology

According to Ryan et. al [18] some objects produce a better photogrammetry output than others depending on their size, texture, and material properties. For example, highly reflective or metallic surfaces are extremely difficult to reproduce using photogrammetry techniques due to the lack of information or their ability to appear different depending on the viewing angle. Large-scaled objects such as buildings also appear to be a challenging subject in photogrammetry due to the difficulty in obtaining sufficient data to support the 3-dimensional output.
In this work, we tried to challenge the Plenoxel method created by Yu et al. [7] and measure the effectiveness of the approach when applied to different object categories and scales.

Yu et. al [7] in their work address the challenges associated with training complex and computationally intensive neural networks in photogrammetry. To overcome these limitations, the authors propose a novel method called Plenoxels for training radiance fields without the use of neural networks, leveraging the concept of plenoptic volume elements. The Plenoxels approach represents scene information and the amount of light using a 5D plenoptic function. Yu et. al [7] employ a sparse voxel grid where each voxel stores data related to opacity and spherical harmonics of the color channels. By utilizing trilinear interpolation, it is possible to compute the color, opacity, and viewing position at any arbitrary position. This technique allows for the estimation of new data points on a 3-dimensional grid based on known data points in space. Notably, the Plenoxels method can be applied to render unbounded scenes by using normalized device coordinates (NDC) to represent the scene in a screen-independent coordinate system ranging from -1 to 1. Alternatively, a multi-sphere image can be created around the grid to represent the background of unbounded scenes.
To achieve high-quality output, coarse-to-fine technique is used in the Plenoxels approach, initially starting at a lower resolution and progressively increasing it by pruning unnecessary voxels and subdividing the remaining voxels. The optimization process in Plenoxels poses a significant challenge due to the vast amount of data involved. To address this, the authors utilize Root Mean Squared Propagation (RMSProp) as an optimization technique, which adapts the step size of each parameter and restricts rapid changes to enhance the learning rate. Additionally, total variation regularization (TV) is used, which is a regularization technique used to solve ill-posed problems in optimization and mitigate overfitting. The implementation of Plenoxels is performed using a custom PyTorch CUDA library, enabling efficient parallelization across rays, spherical harmonics coefficients, and color values. One of the major goals of Plenoxels is to extend its capabilities to 360-degree scenes, which offer users the freedom to render objects of their choice. The method begins with a small resolution of 123^3 and is then upsampled to achieve higher precision and detail.

The primary objective of this master's thesis is to explore and evaluate the Plenoxel method within the domain of photogrammetry. Our aim was to comprehensively analyze the pipeline and workflow of the Plenoxel method and implement them in our research. We sought to examine and apply the workflow of the method, by considering various factors such as data acquisition, preprocessing, and post-processing techniques. Furthermore, we aimed to determine the ideal conditions for dataset production to ensure high-quality and detailed results. This involved investigating the quantity and quality of input data required to achieve the desired level of accuracy and fidelity in the reconstructed scenes. Additionally, we aimed to identify the potential applications of the Plenoxel method in various fields and applications and its ability to offer novel services. Alongside these investigations, we carefully examined the limitations and constraints of the Plenoxel approach, including its complexity, requirements, and any trade-offs between accuracy and efficiency. Finally, based on our findings and insights gained from the research, we aimed to propose potential future improvements and enhancements to the Plenoxel method, addressing its limitations and expanding its capabilities to further advance the field of photogrammetry.

3.1 Methodology

The research in this work was conducted using a systematic approach. Firstly, we tested the Plenoxel approach using objects of various sizes to determine their suitability for objects of different scales. Next, we tested the same objects under different lighting conditions to evaluate the effect of light on the Plenoxel approach and compare them to traditional photogrammetry methods. In addition, we made modifications to the Plenoxel algorithm to produce more suitable data, such as modifying the data output format. Finally, we compared the results obtained using the Plenoxel method to systematically determine the strengths and limitations of the Plenoxel method and to identify areas for improvement.

In this section, we delve into the core scripts utilized within the Plenoxels method, which are integral to dataset processing, training, and evaluation. We provide detailed insights into these scripts, outlining their functionalities and discussing their significance in the overall workflow. Throughout our research, we also made minor modifications to some of these scripts, aiming to enhance their performance and expand their capabilities. These alterations allowed us to conduct a comprehensive analysis, enabling us to gain a deeper understanding of the potential of these scripts within the Plenoxels method. By exploring and experimenting with these scripts, we were able to refine and fine-tune their functionalities, tailoring them to better suit the specific requirements of our research objectives. This iterative process empowered us to conduct more thorough evaluations and assessments, ultimately contributing to a more robust investigation of the Plenoxels method.

In this section we cover some of the main scripts used in the Plenoxels method, for dataset processing, training, and evaluation. Some of them undergo minor changes later in this work helping us to further examine their capabilities.

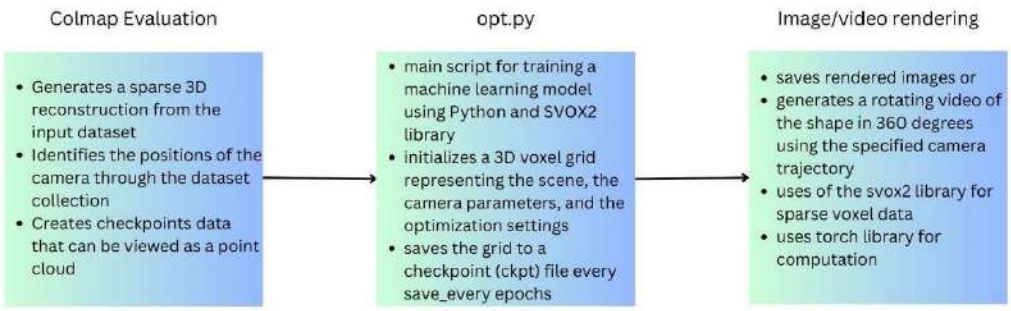


Figure 10 Plenoxels Algorithm pipeline

3.1.1 Colmap evaluation

The first step that the image dataset is going through in the Plenoxels method is to implement the core of the COLMAP pipeline and generate a sparse 3D reconstruction from the ground truth dataset. This step identifies the positions of the camera through the dataset collection and creates checkpoints data that can be viewed as a point cloud. The effectiveness of the results produced in this step is tightly connected to the way the initial image data was taken, the camera positions chosen by the creator, the lighting conditions, and the background choices. Another factor that can influence the quality of the output of this step is the abundant coverage of the object details that are required to produce a sufficient result.

run_Colmap.py
run_colmap function implements the core of the COLMAP pipeline and generates a sparse 3D reconstruction from the input dataset.
The **read_colmap** function reads camera parameters and 3D points from a COLMAP reconstruction (a 3D reconstruction software) and returns two dictionaries: objCameras and objPoints. objCameras is a dictionary that contains camera parameters for each image in the reconstruction, including intrinsic and extrinsic parameters, focal length, principal point, and lens distortion. objPoints is a list of 3D points with their location and color information.
run_colmap function is responsible for running the COLMAP pipeline, which includes feature extraction, matching, and reconstruction. It takes three arguments: vid_root, args, and factor.
The function first creates a directory called "sparse" inside the vid_root directory if it does not already exist. Then, it constructs a command string that will run the feature extraction step of the COLMAP pipeline. This step extracts SIFT features from the resized images and stores them in a database file. Finally, the function constructs a command string to run the mapper step of the pipeline. This step triangulates the matched features to generate a 3D point cloud and a sparse reconstruction.

Create_split.py
This script automates the splitting of a dataset into training and testing sets for use in machine learning or computer vision applications. The script takes as input a root directory where the dataset is located and some optional parameters to customize the splitting process. The script uses the argparse library to parse command-line arguments. The arguments include the root directory of the dataset, the

frequency at which images should be used for testing, and options for running a dry run, automatically answering yes to prompts, and choosing the split randomly rather than at a fixed interval.

3.1.2 Training - Opt.py

This is the main script for training a machine learning model using the Python programming language. It is used for training a 3D scene synthesis model using the SVOX2 library. It initializes and sets up the grid (a 3D voxel grid representing the scene), the camera parameters, and the optimization settings. The script starts by initializing the grid's spherical harmonics (SH) data, density data, and background data to 0.0 or a specified initial value. It then sets up the optimization method for the basis functions of the grid, either using 3D texture or a multi-layer perceptron (MLP). The script then enables gradient computation for the grid and sets up the render options.

The script then defines several learning rate functions for adjusting the learning rate of different parameters during training (sigma, sh, basis, background, color) over time.
It will iterate through multiple epochs of training, shuffling the training data and updating the grid's parameters based on the rendered images and the corresponding ground truth images. The script also has a feature to enable randomness during training, which is normal for certain types of training (LLFF & scenes with background).
The script starts by using the *argparse* library to parse command line arguments and then uses the maybe_merge_config_file function from the *config_util* module to potentially merge a configuration file with the parsed command line arguments. The script then loads a JSON file specified in the reso argument and assigns the result to the reso_list variable. It also initializes the reso_id variable to 0. It then opens a file named args.json in the train_dir directory, writes the args.__dict__ to the file, and saves the current file as opt_frozen.py in the same directory.

Then, the script loads the dataset specified in the dataset_type argument and data_dir argument, using the factor and n_images arguments. The script also creates a svox2.SparseGrid object with specified parameters such as reso, center, and radius that set the parameters and the quality of the training results.

Next, this script is applying various regularization techniques to a grid. The grid has several properties, including density, spherical harmonics (sh), background, and basis. The regularization techniques applied include total variation (TV), sparsity, L2 regularization, and TV of the basis.

The lambda_tv variable controls the strength of the TV regularization applied to the density property. tv_sparsity controls the sparsity of the density property. tv_logalpha and ndc_coeffs are also used in the TV regularization of the density. The *lambda_tv_sh* variable controls the strength of the TV regularization applied to the sh property. tv_sh_sparsity controls the sparsity of the sh property. ndc_coeffs is also used in the TV regularization of the sh. If use_background is true, the background property is regularized, lambda_tv_background_sigma controls the strength of the TV regularization applied to the background density, while lambda_tv_background_color controls the strength of the TV

regularization applied to the background color. tv_background_sparsity controls the sparsity of the background property.

The optim_density_step, optim_sh_step, and optim_background_step functions are used to update the density, sh, and background properties, respectively, using a manual stochastic gradient descent (SGD) or RMSprop step.

Finally, the code saves the grid to a checkpoint (ckpt) file every save_every epochs, and upsamples the grid every upsamp_every steps.

3.1.3 Rendering

Render_imgs.py

This code appears to be a script for evaluating a pre-trained 3D model, specifically one that uses the Sparse Voxel CNN (SVox2) architecture, on a given dataset. It uses the PyTorch library and the SVox2 package, which is a library for training and evaluating 3D models using the SVox2 architecture.
The script starts by importing a number of necessary libraries and modules, including torch and svox2 for PyTorch and SVox2, respectively, as well as numpy, os, argparse, imageio, and cv2 for various other functionality.
It then defines command line arguments that can be passed to the script when it is run, such as the location of the checkpoint file for the pre-trained model, the number of images to evaluate, and whether to use the train or test set. The script then loads the pre-trained model from the checkpoint file and sets the device to be used as 'cuda:0' (a GPU). It then loads the dataset specified in the command line arguments, and then uses SVox2 to render images from the dataset using the pre-trained model.
It then uses imageio library to save the rendered images and if not specified in the command line arguments, it also calculates metrics such as PSNR, SSIM, and LPIPS, which are all measures of image quality. It also creates a directory where the images are saved.

Render_imgs_circle.py

This code loads a 3D shape from a saved checkpoint and is used with a custom image dataset. It generates a rotating video of the shape in 360 degrees using the specified camera trajectory. It makes. Here is a brief explanation of the code:

args = parser.parse_args() parses the command-line arguments and stores them in args variable.
config_util.maybe_merge_config_file(args, allow_invalid=True) loads the configuration file if specified in args.
device = 'cuda:0' sets the device used for computation to the GPU. If no GPU is available, it uses the CPU.
dset = datasets[args.dataset_type](args.data_dir, split="test", **config_util.build_data_options(args)) loads the dataset to be rendered. The dataset is specified by the dataset_type argument.
if args.vec_up is None: sets the up direction of the camera based on the dataset orientation, or using the specified vector in the vec_up argument.

if args.traj_type == 'spiral': generates camera trajectories in a spiral path, and if not, it generates trajectories in a circular path.
c2ws = np.stack(c2ws, axis=0) stacks the camera matrices generated in step 6 into a single tensor.
if args.vert_shift != 0.0: shifts the camera position vertically by a specified amount.
grid = svox2.SparseGrid.load(args.ckpt, device=device) loads the saved 3D shape checkpoint using the svox2 library.
config_util.setup_render_opts(grid.opt, args) sets up the rendering options based on the arguments passed.
with torch.no_grad(): temporarily disables gradient calculation for performance.
for img_id in tqdm(range(0, n_images, img_eval_interval)): loops through the generated camera trajectories to render each frame of the video.
frames.append(img.cpu().numpy()) appends the rendered image to a list.
skvideo.io.vwrite(render_out_path, np.stack(frames)) saves the rendered video to disk using the skvideo library.

Some of the main differences between **Render_imgs.py** and **Render_imgs_circle.py**:

1. Additional imports: This code imports additional libraries like imageio, cv2 and tqdm which were not present in the previous code.
2. Additional logic: This code has additional logic for handling the additional arguments that are defined. For example, it handles the vec_up argument by automatically determining its value if it is not provided.
3. Additional functionality: This code has additional functionality for creating video output from the rendered images, whereas the previous code only rendered images.
4. Additional library: This code uses the svox2 library which was not present in the previous code, it is used for 3D model rendering.

3.1.4 Checkpoints to mesh

After finishing the training part, we applied a simple marching cube algorithm to the produced checkpoints to create a mesh.

```python
import svox2, mcubes, torch, numpy, argparse

parser = argparse.ArgumentParser()
parser.add_argument('ckpt', type=str)
args = parser.parse_args()
targetpath = args.ckpt[:-3]+"obj"
grid = svox2.SparseGrid.load(args.ckpt)

resfactor = 1 #increase/decrease to get higher or lower resolution meshes. Be very careful when increasing. 0.3 is nice for previewing

resx = int(grid.shape[0]*resfactor)
resy = int(grid.shape[1]*resfactor)
resz = int(grid.shape[2]*resfactor)

densitygrid = numpy.zeros((resx,resy,resz))
print("converting densities to non-sparse numpy array")

for x in range(resx):
    print("Progress:%i %%"%(100*x/resx)) #progress
    xvals = [-0.5+x/resx]*resx # an array with x-size size and filled with current x value, by default ranges -.5 to .49
    xvals = numpy.array(xvals)
    for y in range(resy):
        yvals = [-0.5+y/resy]*resy # an array with y-size size and filled with current y value
        yvals = numpy.array(yvals)
        zvals = numpy.array(range(0,resz))/resz
        zvals = zvals-0.5
        samplepos = numpy.dstack([xvals,yvals,zvals])[0] #get a full z-line at x,y coords worth of 3d coordinates.
        u = grid.sample(torch.Tensor(samplepos),want_colors=False)[0] #only get densities, no colors
        v = u.detach().numpy() #turn tensor into numpy array.
        vt = v.T[0] #turn array from shape (128,1) to (1,128)
        densitygrid[x,y] = vt #feed the extracted line into our density array

#from matplotlib import pyplot as plt
#plt.imshow( densitygrid[int(resz/2)], interpolation='nearest')
#plt.show()

#finally producing object.
v,t = mcubes.marching_cubes(densitygrid,20) #adjust value to your scene. start with 0.
mcubes.export_obj(v,t,targetpath)
```

Figure 11 Marching cubes, code snipset

3.2 Algorithm Evaluations on datasets

3.2.1 Synthetic scenes

The synthetic scenes can be tested much easier since they remove the limitations of the human factor concerning the collection of the initial image dataset. Synthetic scenes provide with enough necessary images to cover the object of interest from every angle, with an optimal lighting, and without background.

Since our interest in this work lies in the 360-degree scenes, we will just briefly cover these scene type for the sake of comparison of the process and the output results.

Drumset
Dataset: 100 images

To compare this scene type with the 360-degree scenes we used the COLMAP step on the dataset wich is not required with the synthetic datasets. Our goal was to observe the positions and points created in case we have a strictly controlled dataset and compare it to the 360-degree ones. This helps us to identify what are the desired camera positions and the level of details that can be achieved in the best-case scenario.

Figure 12 Synthetic scene, drumset, ground truth

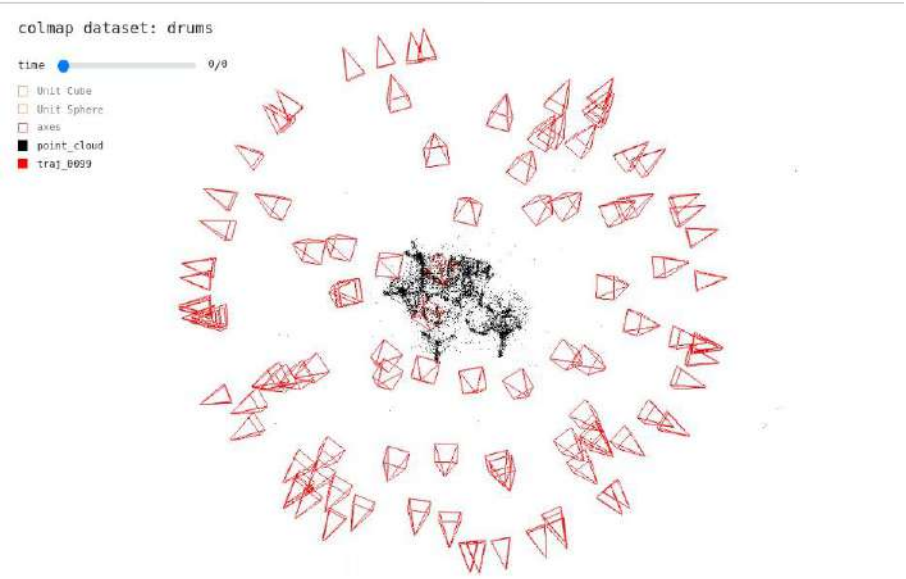


Figure 13 Synthetic scene, drumset, Colmap evaluation results 1, from above

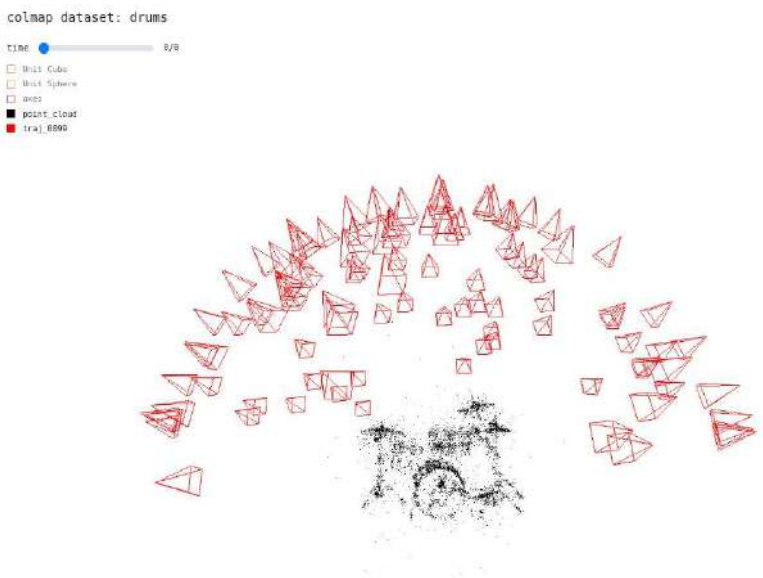


Figure 14 Synthetic scene, drumset, Colmap evaluation results 2, front view

We can see the perfectly justified in space camera positions (red) that cover the object from every angle.

Arguments:

config	syn.json
epoch_size	12800
white_bkgd	true
step_size	0.5
background_brightness	1.0
near_clip	0.0
reso	[[256, 256, 256], [512, 512, 512]]
upsamp_every	38400
upsample_density_add	0.0
background_nlayers	0
background_reso	512
n_iters	128000
batch_size	5000
sigma_optim	rmsprop
sh_optim	rmsprop
bg_optim	rmsprop
basis_optim	rmsprop

eval stats: {'psnr': 25.064694645286583, 'mse': 0.0033806120802182702}
Time : 7.261256516666666

Images 1 and 3 show us the view-dependent effects that can be achieved using the Plenoxels method. Specifically, in this case, we can see the reflections that occur on the surface of the glossy dark material on the front of the drumset which changes as we rotate our viewing point and the reflections in the transparent materials on the top part of the drumset.

Figure 15 Synthetic scene, drumset, ground truth (left) and Plenoxel output (right)

3.2.2 360 Scenes

3.2.2.1 Data structure

Step 1: Colmap evaluation

After proc_colmap.sh step data is organized into:

- Images folder: original images are resized to a smaller size, renamed in sequential order 0_00001.jpg, 0_00002.jpg, etc. Their format is also changed to jpg.
- Pose folder: includes txt folder with coordinates for each image from the dataset
- Raw folder: original images
- Sparse folder:cameras.bin images.bin points3D.bin project.ini
- VIsual folder: index.html file that includes the GUI for the python view_data.py <img_dir> step
- volrend.draw .npz file
- Database.db file:
- intrinscs .txt file:
- Points.npy file:

python view_data.py <img_dir> uses colmap
Step 1 produces a sparse point cloud that can be visualized using nerfvis
Which launches a server at localhost where we can see the produced point cloud of the object and the identified camera positions for each of the images in the dataset. The point cloud shows the object that has been identified through the process.

Step 2: Training

./launch.sh <exp_name> <GPU_id> <data_dir> -c configs/custom.json
This step splits the dataset into 8 batches, then normalizes the data using camera data, scales the scene using intrinsics, it then generates rays of scaling factor 1 and then performs evaluations for 8 epochs.

```
40 epoch 0 psnr=7.10:   0%|          | 12/12800 [00:00<01:51, 114.73it/s]
41 epoch 0 psnr=7.10:   0%|          | 26/12800 [00:00<01:40, 127.40it/s]
42 epoch 0 psnr=8.49:   0%|          | 26/12800 [00:00<01:40, 127.40it/s]
43 epoch 0 psnr=8.49:   0%|          | 41/12800 [00:00<01:34, 134.82it/s]
14570 epoch 7 psnr=28.99: 100%|██████████| 12798/12800 [02:25<00:00, 88.10it/s]
14571 epoch 7 psnr=29.13: 100%|██████████| 12798/12800 [02:25<00:00, 88.10it/s]
14572 epoch 7 psnr=29.13: 100%|██████████| 12800/12800 [02:25<00:00, 88.04it/s]
```

Figure 16 Plenoxels training process snipset

The output is the ckpt.npz file which includes the checkpoints of the training process, the details of which can be seen in the log file.

- Checkpoints in .npz format
- Log file with information regarding the training process
- Time_mins.txt which records the time required for the training process

- Args.json which includes arguments data used in the training.

config	configs/custom.json
epoch_size	12800
white_bkgd	true
step_size	0.5
background_brightness	0.5
near_clip	0.35
reso	[[128, 128, 128], [256, 256, 256], [512, 512, 512]]
upsamp_every	25600
background_reso	1024
n_iters	102400

Step 3: Video Rendering
Python render_imgs_circle.py <checkpoints.npz> <data_dir>
Renders.mp4 is the video produced after the rendering process. In the case of the synthetic dataset, a test_renders folder is created which includes the frames of the rendered video. Since the custom 360 datasets rendering does not include this function, we modified the code to save the video frames as images and bypass this limitation. This way we can have a more detailed image about the results and the data that may be missing.
Step 4: Mesh Reconstruction
from checkpoints using Marching Cubes algorithm

3.2.2.1 Examples from PLENOXEL datasets

First we examined the datasets provided by the Plenoxels team. We aimed to identify the data format and the process used to create the most efficient output. The team provided image datasets of different objects and scales. In this work, we showcase two of the examples that concern bigger objects since they are usually more difficult to photograph and render due to the limitations of the objects' sizes.

M60

Dataset 313 images (1077 x 546 pixels) of a tank by hand from different angles
LIghitng conditions: indoors, strong window light
Downsampled and resized images: 313 images (1077 x 546 pixels), no downsampling performed
Step 1: Colmap evaluation

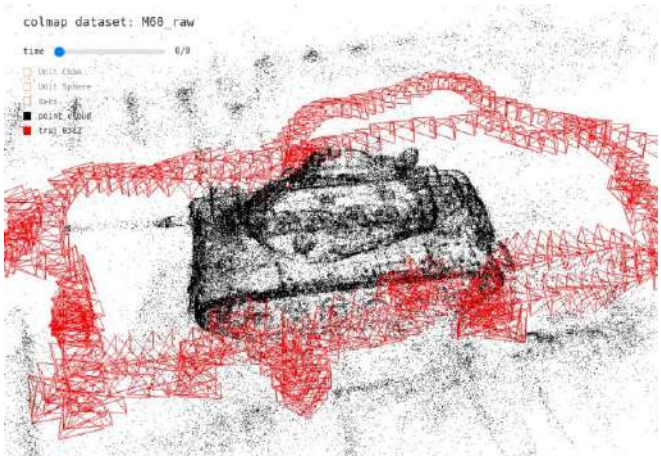


Figure 17 360 scene, m60, Colmap evaluation results 1

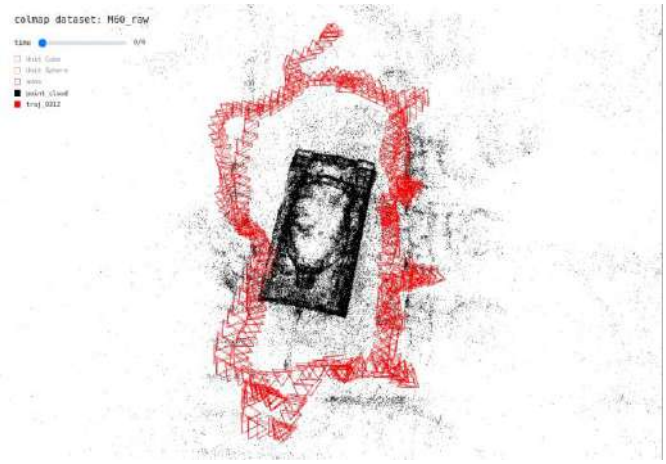


Figure 18 360 scene, m60, Colmap evaluation results 2

Step 2: Training

Evaluation time :21.04 min

Evaluation Stats:
signal-to-noise ratio (PSNR) : 23.421766534684636,
mean-square error (MSE): 0.005727972707245499

Step 3: Video Rendering
Ground truth and Renders comparison

Figure 19 360 scene, m60, ground truth (left) and Plenoxels algorithm output (right) comparison

Lighting variations due to the viewing point and strong light are visible in the render. It however does not limit the point cloud (step 1) or the mesh produced.

Notes:

Due to the size of the object some of the data is missing from the top part of the object

Figure 20 360 scene, m60, Plenoxels algorithm output 1

Strong background light from the window reduces the quality of the renders in some parts

Figure 21 360 scene, m60, ground truth 1

Playground

Dataset: 307 images (1008 x 548 pixels) of a playground
Lighting conditions: soft sunlights, partially cloudy
Downsampled and resized images: 307 images (1008 x 548 pixels), no downsampling performed

Step 1:Colmap evaluation

Figure 22 360 scene, playground, Colmap evaluation results

Step 2: Training
Evaluation time: 19.439854866666668 min

Evaluation Stats:
signal-to-noise ratio (PSNR): 22.395309038883262
Mean-square error (MSE): 0.0061263062059879305}
Epochs number: 7

Step 3: Video Rendering

Video render: 1200 frames

Ground truth and Renders comparison

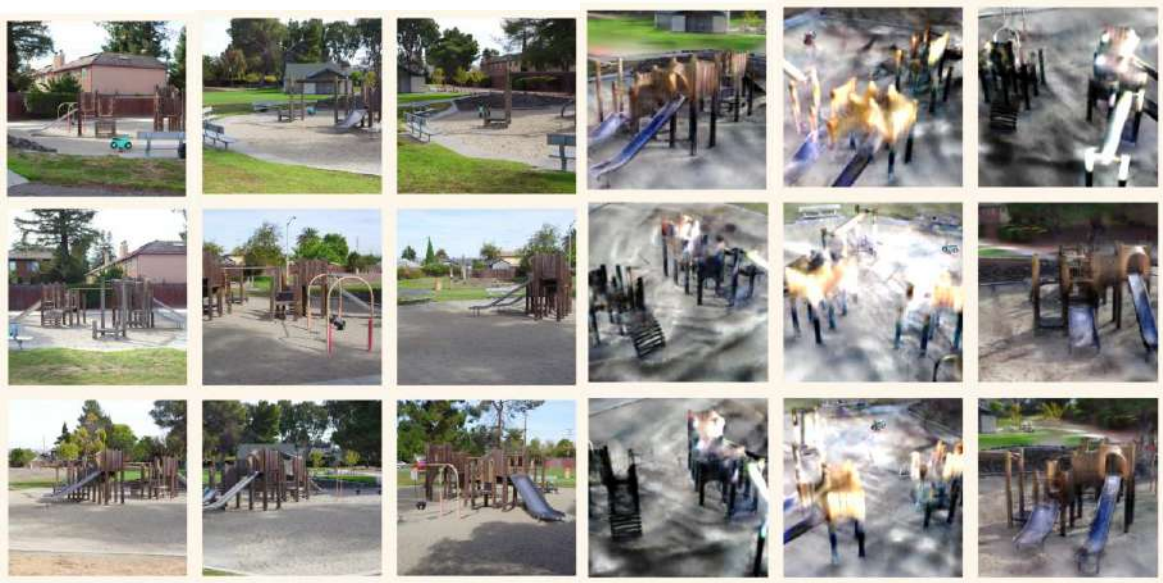


Figure 23 360 scene, playground, ground truth (left) and Plenoxels algorithm output (right) comparison

4. Experiments and results

4.1 Custom Datasets

4.1.1 Person without background

Input dataset 84 images png (822x822 pixels) captured with hand-held camera
Downsampled and resized images: 84 images (768 x 768 pixels)

Lighting conditions: controlled studio lighting, 2 lighting sources from front-left and front-right

Step 1: Colmap evaluation

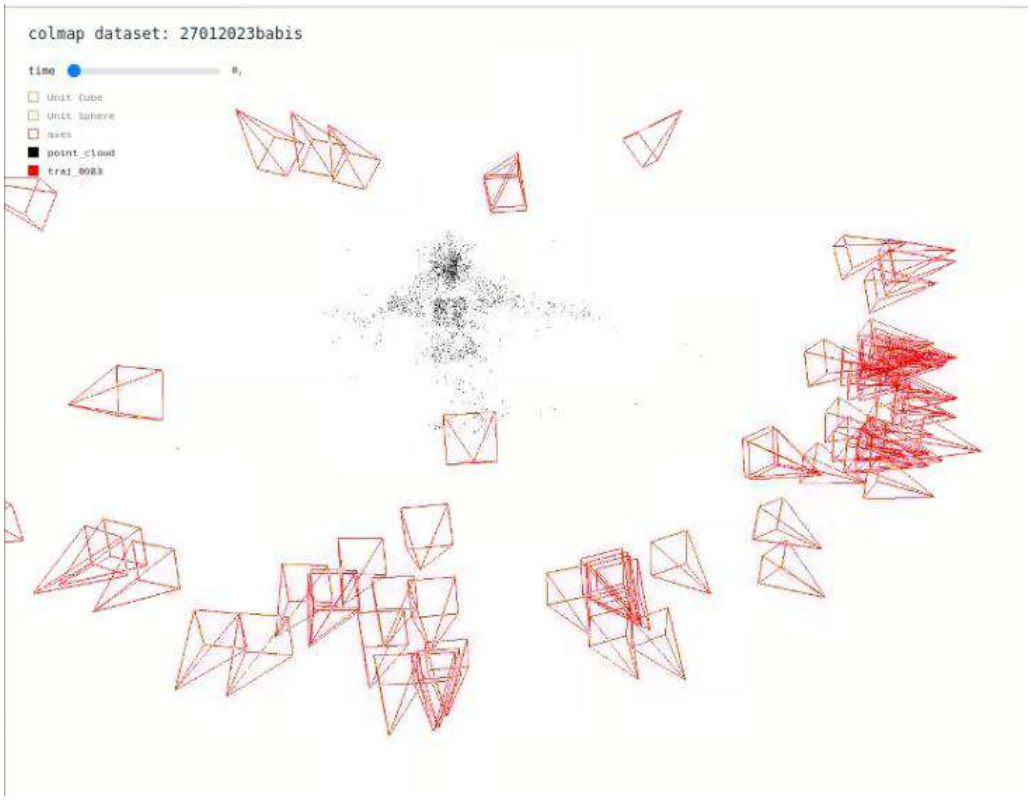


Figure 24 Custom 360 scene, person 1, Colmap evaluation results 1

Step 2: Training

Evaluation time: 16.438759916666665 min

Evaluation Stats:
Signal-to-noise ratio (PSNR): 20.42303401731458,
Mean-square error (MSE): 0.009593649844949445
Epochs number: 7

In photogrammetry, PSNR (Peak Signal-to-Noise Ratio) and MSE (Mean Squared Error) are both metrics used to evaluate the quality of reconstructed images or 3D models. The main difference between the two is that PSNR takes into account both the signal and noise, while MSE only considers the noise. MSE is a measure of the average squared difference between the original image and the reconstructed image, and it provides a quantitative value for the overall image quality. On the other hand, PSNR is a logarithmic measure of the ratio between the maximum possible power of the signal and the power of the noise that affects the fidelity of its representation. In other words, PSNR calculates the ratio of the signal power to the noise power in decibels (dB).

Step 3: Video Rendering

Figure 25 Custom 360 scene, person 1, ground truth (left) and Plenoxels Algorithm output (right) comparison

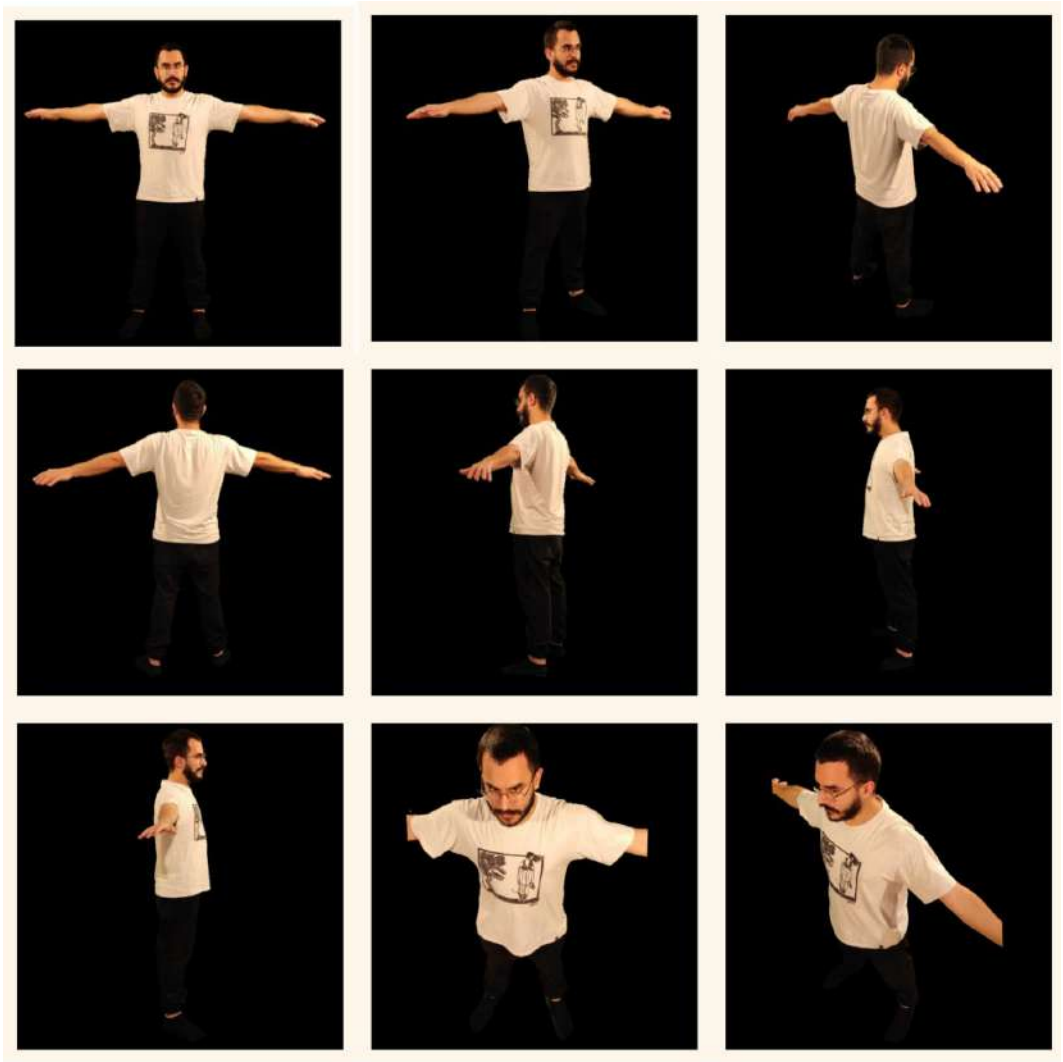


Figure 26 Custom 360 scene, person 1, Ground truth

Step 4: Mesh Reconstruction

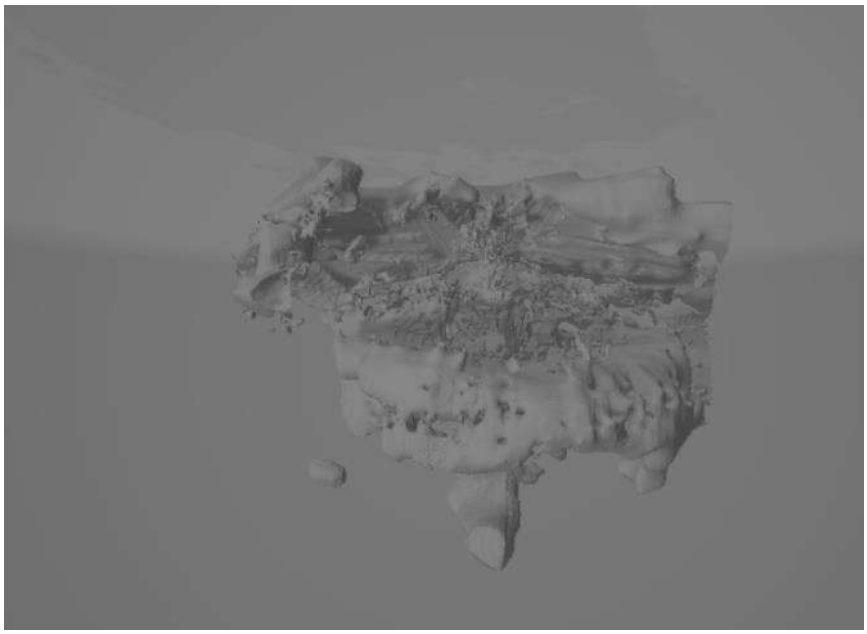


Figure 27 Custom 360 scene, person 1, mesh produced using Marching Cubes algorithm

For custom 360 scenes this steps outputs only the rendered video, so we modified the given code to save the frames of the video as separate images

4.1.2 Person without background - Alternative configuration

Same Dataset as in previous example.

We skip the 1st step of Colmap evaluation since it was already done in the previous example and no changes are needed to use custom_alt.json for configuration parameters

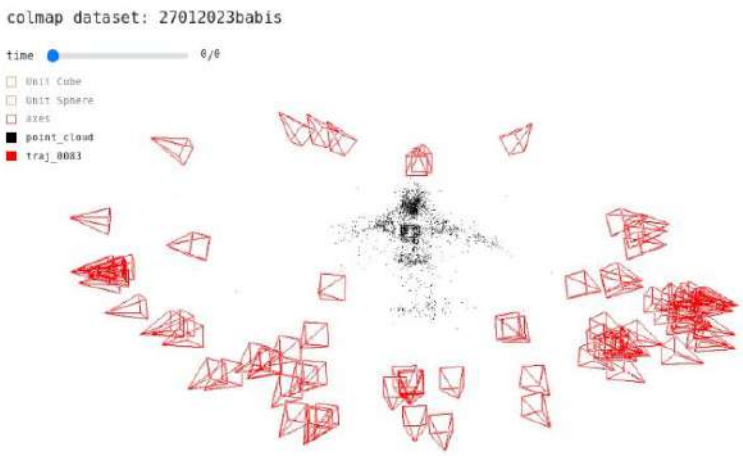


Figure 28 Custom 360 scene, person 2, Colmap evaluation results 1

Evaluation Time: 10.3162038 min
Evaluation Stats:
Signal-to-noise ratio (PSNR): 19.831963863447836
Mean-square error (MSE): 0.0160240230228131}

Figure 29 Custom 360 scene, person 2, Plenoxels algorithm output

Step 4: Mesh Reconstruction

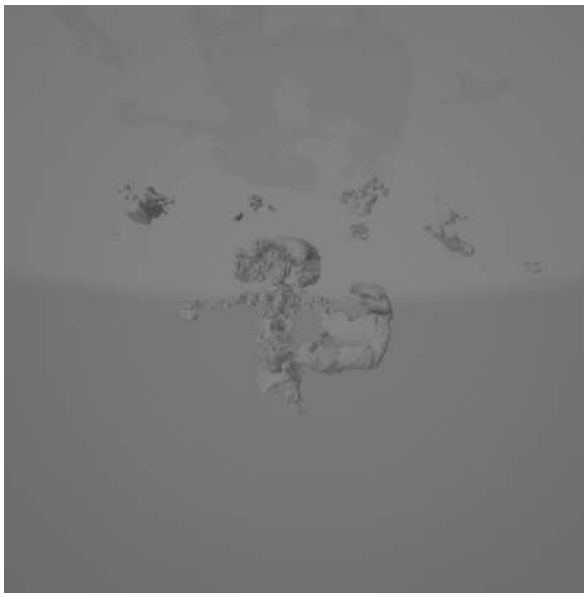


Figure 30 Custom 360 scene, person 1, mesh produced using Marching Cubes algorithm

Notes:
Most differences between custom and custom_alt are visible to the eye when we use marching cubes to render a mesh. Custom_alt despite its minor changes to the configuration managed to remove a lot of unnecessary noise that existed and produced a much cleaner result with emphasis on the figure. Even though the original dataset does not include background, custom.json settings still recognize some of the information as background and record them in the checkpoints compared to the custom_alt.

Differences between custom and custom_alt configurations

custom.json	custom_alt.json
"cam_scale_factor": 0.9,	"cam_scale_factor": 0.95,
"near_clip": 0.35,	-
"lr_sigma": 3e1,	"lr_sigma": 3e1,
"lr_sh": 1e-2,	"lr_sh": 1e-2,
"lr_sigma_delay_steps": 0,	"lr_sigma_delay_steps": 35000,

"lambda_tv": 5e-3,	"lambda_tv": 5e-5,
"lambda_tv_background_sigma": 5e-3,	"lambda_tv_background_sigma": 1e-3,
"lambda_tv_background_color": 5e-3,	"lambda_tv_background_sigma": 1e-3,
"background_brightness": 0.5,	"background_brightness": 1.0,
"tv_decay": 0.5	-

4.1.3 Object 1 - Plant

Initial dataset: 35 jpg images 4640x2088px captured with hand-held mobile phone camera
Lighting conditions: Natural daylight in shadow
Downsampled and resized images: 35 png images 345x768px approximate size 400kb

Step 1: Colmap evaluation
We can see that due to increased background information in the image dataset, the point cloud produces a spherical representation of the environment around the object. It may be more difficult to identify the shape of the object compared to the previous test where no background information was present on the initial dataset.

Figure 31 Custom 360 scene, Plant, Colmap evaluation results 1

Step 2: Training
Evaluation Time: 20.49500985 min
Evaluation Stats:
Signal-to-noise ratio (PSNR) : 20.356335902811903,
Mean-square error (MSE): 0.00953012735893329
Epochs number: 8

Step 3: Video Rendering

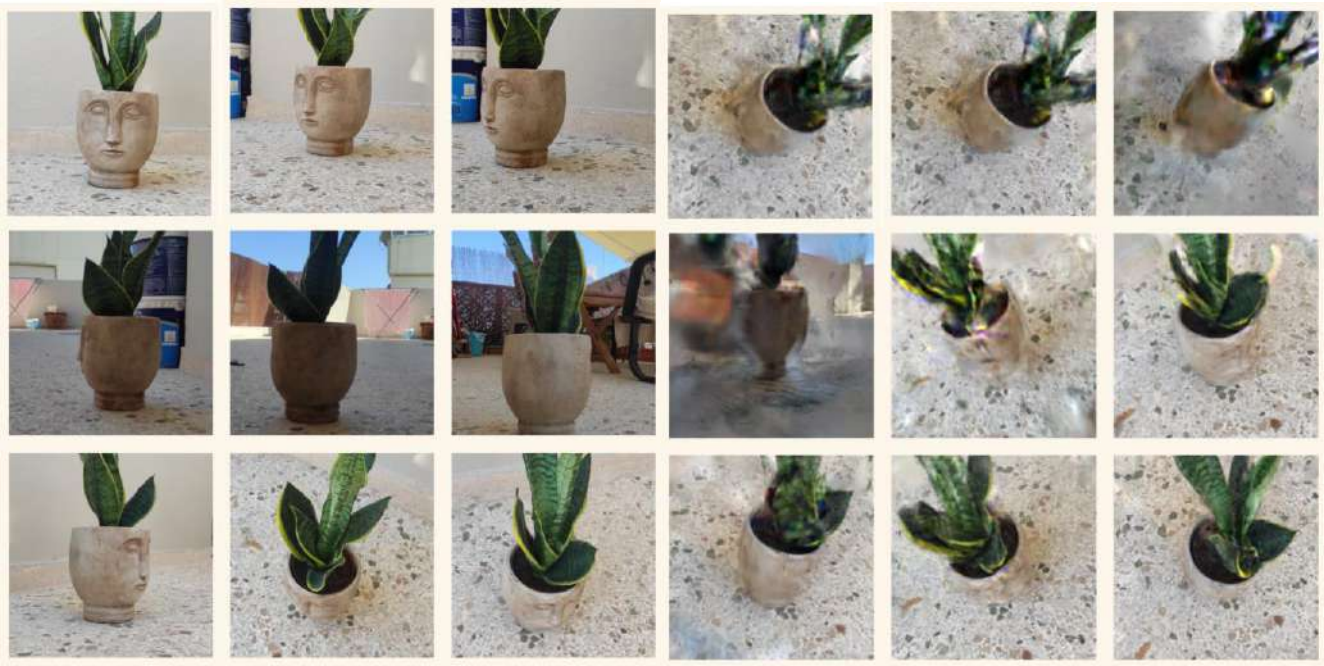


Figure 32 Custom 360 scene, Plant, ground truth (left) and Plenoxels algorithm (right) comparison

Step 4: Mesh Reconstruction

Figure 33 Custom 360 scene, Plant, mesh produced using Marching Cubes algorithm

Altternative

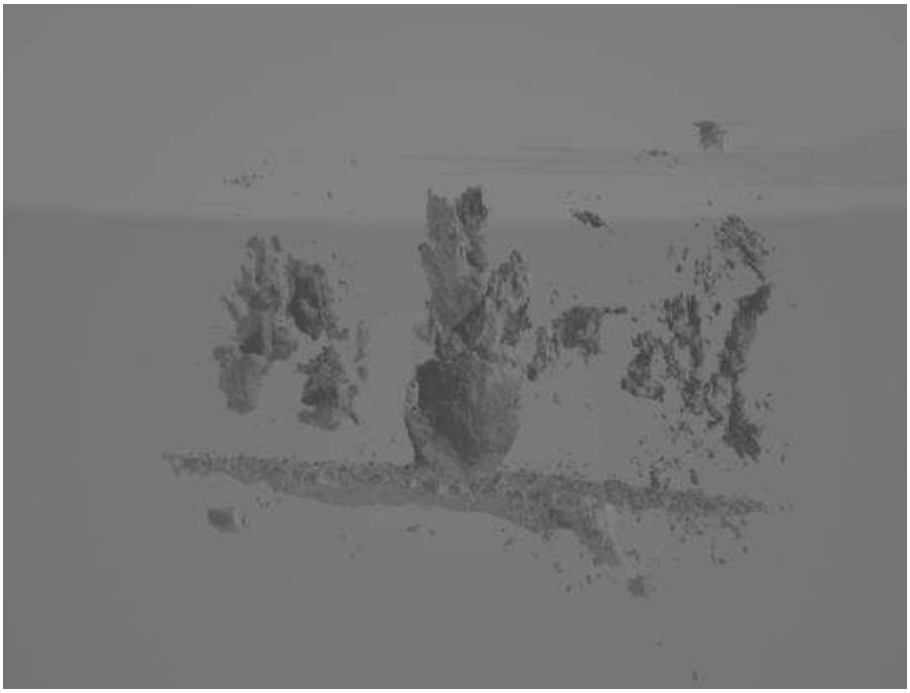


Figure 34 Custom 360 scene, Plant alternative, mesh produced using Marching Cubes algorithm

Notes
As we can see in the rendered images 3 and 4, due to inconsistent lighting and information-heavy background a significant amount of information was lost in the results. Even though the object was well photographed from all angles, the data was inconsistent which led to the loss of more than 30% of the object in the final result.

4.1.4 Object 2 - Vase

Dataset: 31 images (4640 x 2088 pixels) captured with hand-held mobile phone camera
Lighting conditions: natural diffuse light
Downsampled and resized images: 31 images (345 x 768 pixels)

Figure 35 Custom 360 scene, Vase, Gound truth

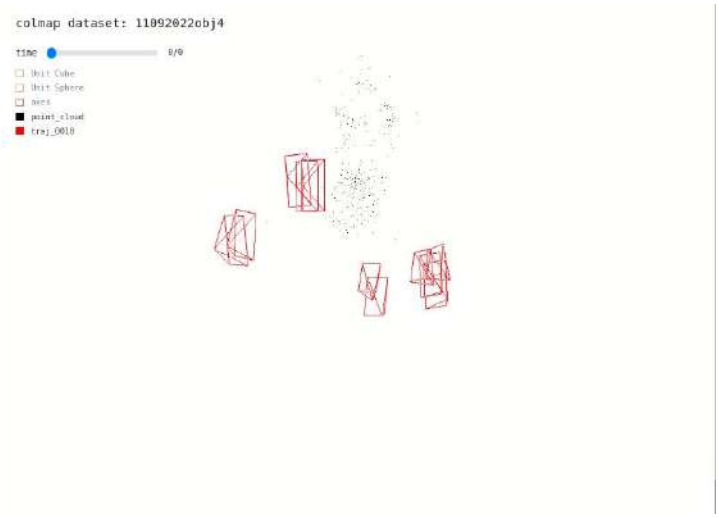


Figure 36 Custom 360 scene, Vase, Colmap evaluation results

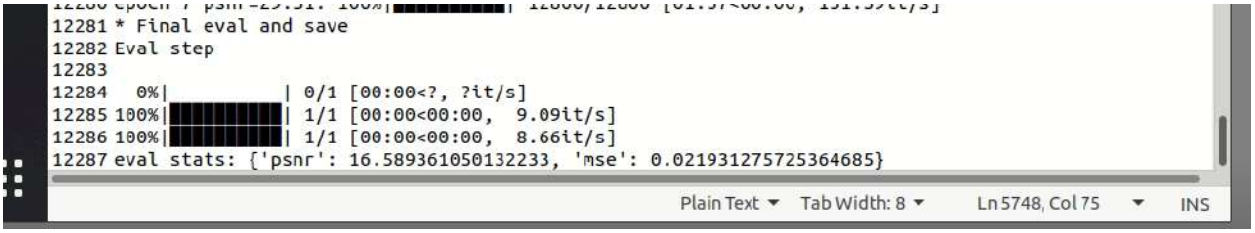


Figure 37 Custom 360 scene, Vase, training process snipset

Having an object with colors similar to the background did not produce the expected result. The dataset could not be successfully evaluated since not enough data was extracted using the Colmap step.

4.1.5 Object 3 - Toy

Object size: medium, approximately 40cm
Lighting conditions: natural diffuse light, cloudy
Dataset: 118 images (3472x4640 pixels) captured with hand-held mobile phone camera
Downsampled and resized images: 118 images (574x768 pixels)

Step 1: Colmap evaluation

Figure 38 Custom 360 scene, Toy, Colmap evaluation results

Step 2: training
Training step:
Evaluation Time: 21.6579343 min
Evaluation Stats:
Signal-to-noise ratio (PSNR) : 30.643722010923486
Mean-square error (MSE): 0.0009465880830248352
Epochs number: 7
Video render: 1200 frames

Step 3: Video Rendering

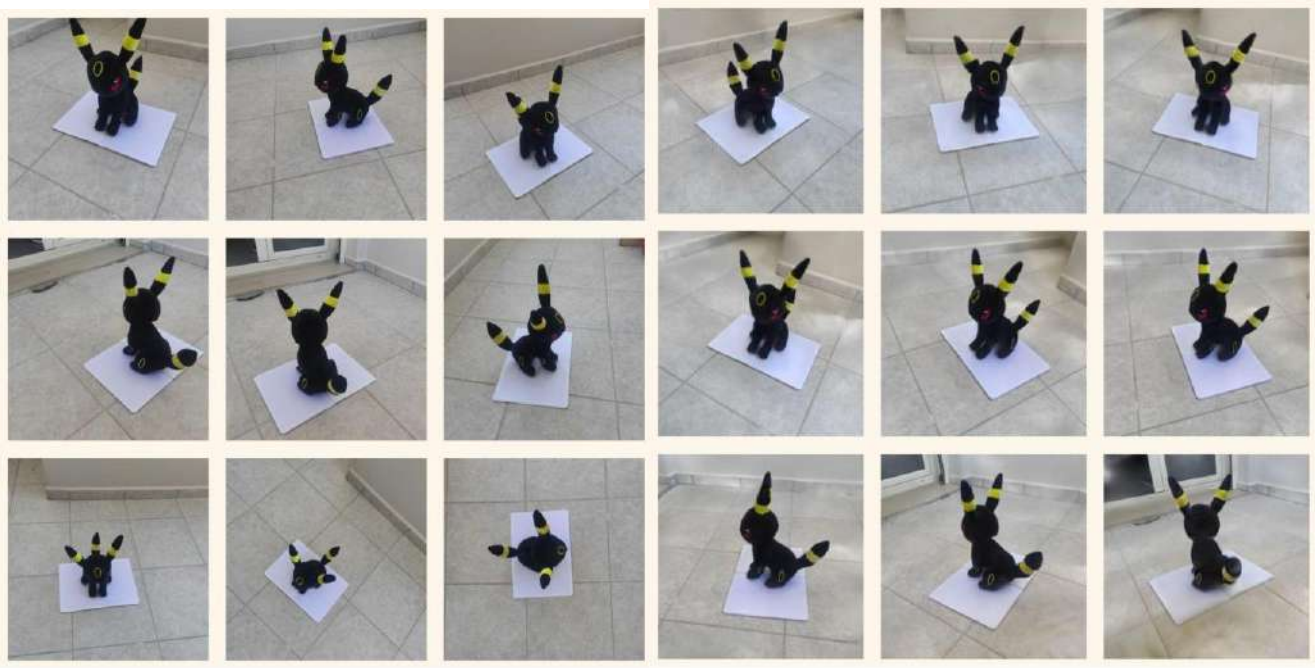


Figure 39 Custom 360 scene, Toy, ground truth (left) and Plenoxels algorithm renders (right) comparison

Step 4: Mesh Reconstruction

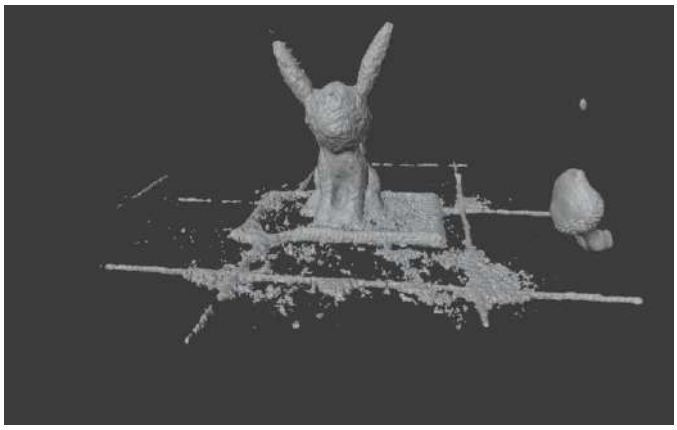


Figure 40 Custom 360 scene, Toy, mesh produced using Marching Cubes algorithm

4.1.6 Object 4 - Skull sculpture

Object size: small, approximately 15cm
Dataset: 190 images (4640 x 3472 pixels) captured with hand-held mobile phone camera
Lighting conditions: Natural diffuse light, no hard shadows
Downsampled and resized images: 190 images (574 x 768 pixels)

Step 1: Colmap evaluation

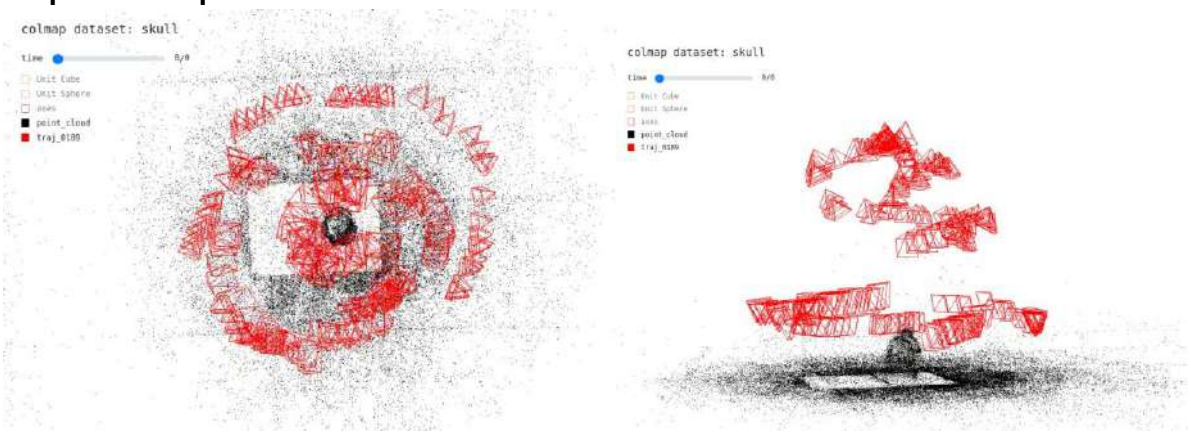


Figure 41 Custom 360 scene, Skull sculpture, Colmap evaluation results

Step 2: Training

Evaluation Time: 21.564470916666668 min

Evaluation Stats:
Signal-to-noise ratio (PSNR) : 32.402391280628116
Mean-square error (MSE): 0.0006009265416651033
Epochs number: 7

Step 3: Video Rendering
Video render: 1200 frames

Figure 42 Custom 360 scene, Skull sculpture, ground truth (left) and Plenoxel results (right) comparison

Step 4: Mesh Reconstruction

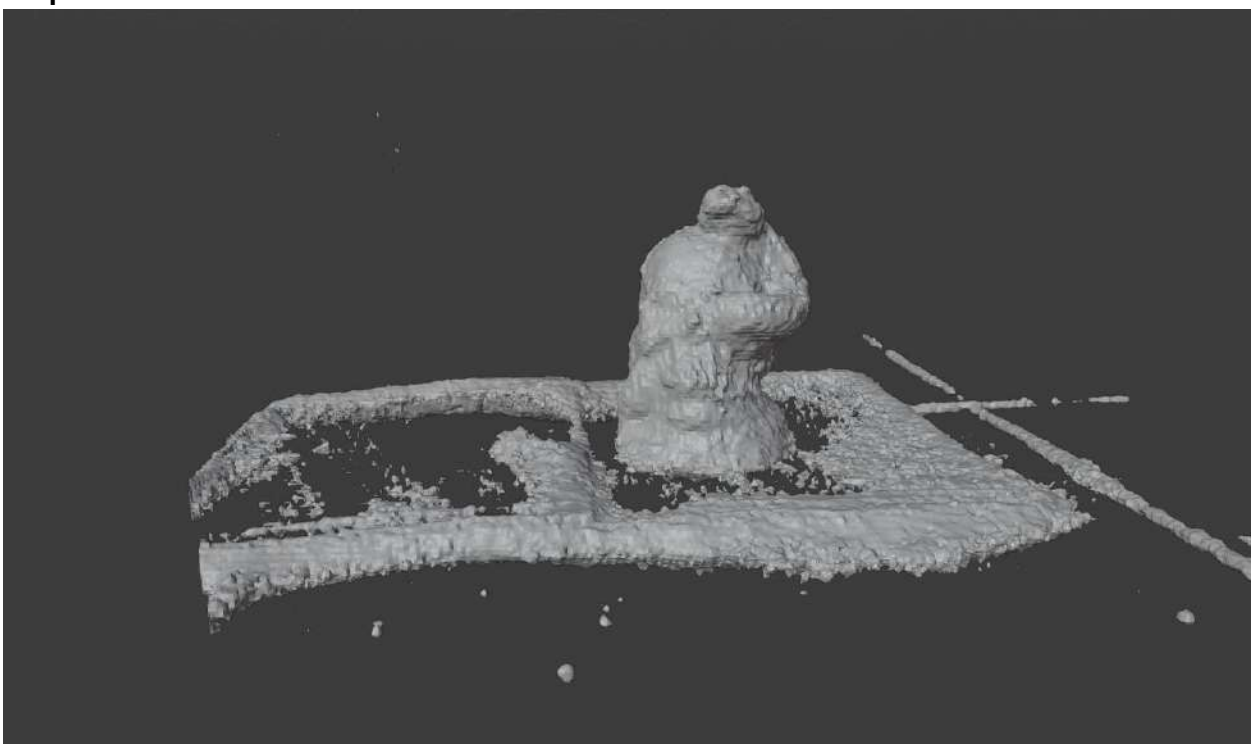


Figure 43 Custom 360 scene, Skull sculpture, Mesh produced using Marching Cubes algorithms

4.1.7 Object 5 - Group of objects

Large size: medium, approximately 3x3m
Dataset: 297 images (3472x4640 pixels) captured with hand-held mobile phone camera
Lighting Conditions: natural diffuse light
Downsampled and resized images: 297 images (574 x 768 pixels)

Step 1: Colmap evaluation

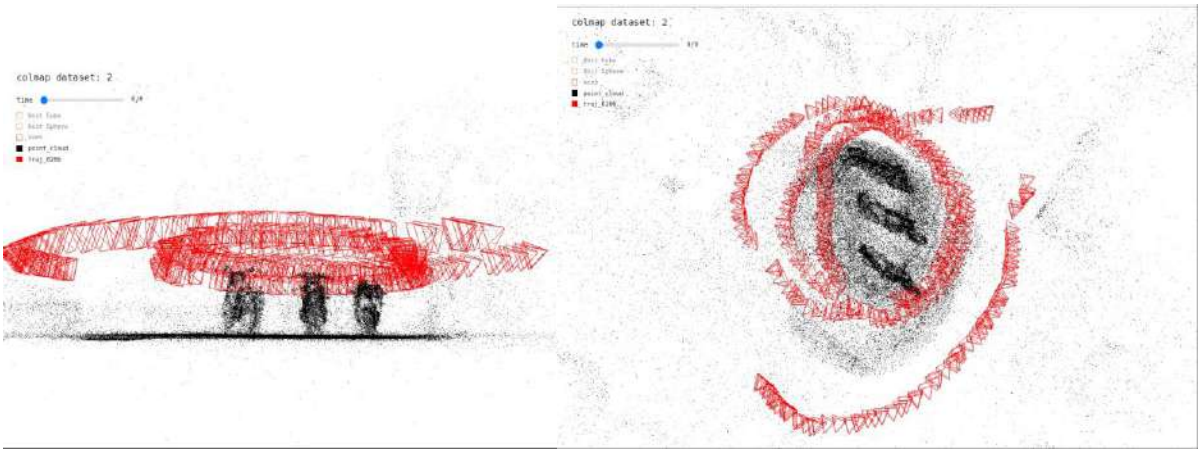


Figure 44 Custom 360 scene, Group of objects, Colmap evaluation results

Step 2: Training

Evaluation Time: 21.023919616666667 min
Evaluation Stats:
Signal-to-noise ratio (PSNR) : 23.7736436598887
Mean-square error (MSE): 0.00443241076492187
Epochs number: 7

Step 3: Video Rendering

Video render: 1200 frames

Figure 45 Custom 360 scene, Group of objects, ground truth (left) and Plenoxel results (right) comparison

Step 4: Mesh Reconstruction

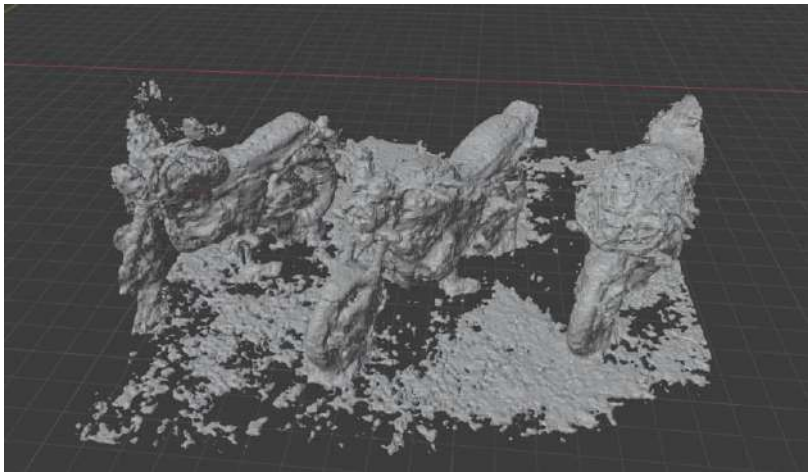


Figure 46 Custom 360 scene, Group of objects, mesh produced using Marching Cubes algorithm

4.1.8 Object 6 - Lighting conditions comparison

Object size: large, approximately 2 x 1 m
Dataset: 174 images (4640 x 3472 pixels) captured with hand-held mobile phone camera
Lighting conditions: sunset, strong light, long shadows
Downsampled and resized images: 174 images (574 x 768 pixels)

Step 1: Colmap evaluation

Figure 47 Custom 360 scene, lighting conditions comparison, Colmap evaluation results

Step 2: Training

Evaluation Time: 20.061042866666664 min
Evaluation Stats:
Signal-to-noise ratio (PSNR) : 23.531332530063693
Mean-square error (MSE): 0.004729033424519002
Epochs number: 7

Step 3: Video Rendering

Video render: 1200 frames

Figure 48 Custom 360 scene, lighting conditions comparison, ground truth (left) and Plenoxel results (right) comparison

Step 4: Mesh Reconstruction

Figure 49 Custom 360 scene, lighting conditions comparison, mesh produced using Marching Cubes algorithm

Lightings setting 2

Dataset: 166 images (4640 x 3472 pixels) captured with hand-held mobile phone camera
Lighting conditions: natural diffuse lighting
Downsampled and resized images: 166 images (574 x 768 pixels)

Step 1: Colmap evaluation

Figure 50 Custom 360 scene, lighting conditions setting 2, Colmap evaluation

Step 2: Training
Evaluation Time: 19.776202633333334

Evaluation Stats:
Signal-to-noise ratio (PSNR) : 23.531332530063693
Mean-square error (MSE): 0.004729033424519002
Epochs number: 7

Step 3: Video Rendering
Video render: 1200 frames

Figure 51 Custom 360 scene, lighting conditions setting 2, ground truth (left) and Plenoxel results (right) comparison

Step 4: Mesh Rendering

Figure 52 Custom 360 scene, lighting conditions setting 2, mesh produced using Marching Cubes algorithms

4.2 Proof of concept and comparison:

It is stated that in photogrammetry and related techniques, lighting plays a crucial role when it comes to the details quality of the captured data. In this section, we performed a proof of this concept using the last two datasets. Both datasets depict the same object in significantly different lighting conditions. We proceeded to compare the results of each step of the Plenoxels method between the two datasets with the main goal to capture the differences in details between the two results.

Step 1: Colmap evaluation

No significant differences on the point cloud after Colmap evaluation (step 1). However, if we look closely, we notice that due to lighting conditions, the amount of noise (sparse points around the more dense main subject) is visible around the object

Figure 53 Conclusion 1

Step 3: Video Rendering

On the video render (step 3) the differences that are caused due to the lighting conditions become more apparent. Some information is lost due to the lighting in the first example. In point 1 (highlighted) of the image, we can see bright sunlight coming from the front of the object that generates the noise visible in the rest of the images. Part of the object is hidden behind a “cloud” that occurs in the direction that the sunlight is hitting from. Due to the inability of the system to identify the nature of the object, the sunlight is perceived as one having volume, so it is also rendered in the final video.

Figure 54 Conclusion 2

Step 4: Mesh Reconstruction

When rendering a mesh, we observe that our assumption was true and lighting rays were perceived as objects generating additional noise (top image). This explains the data loss in the video render since the lighting rays are considered an object and hence rendered, limiting some of the viewpoints. Comparing the output to the cleaner mesh produced from the diffuse lighting dataset proves the importance of good lighting conditions when it comes to capturing the

data. The lighting directly impacts the number of details that can be captured and transferred to the digital form.

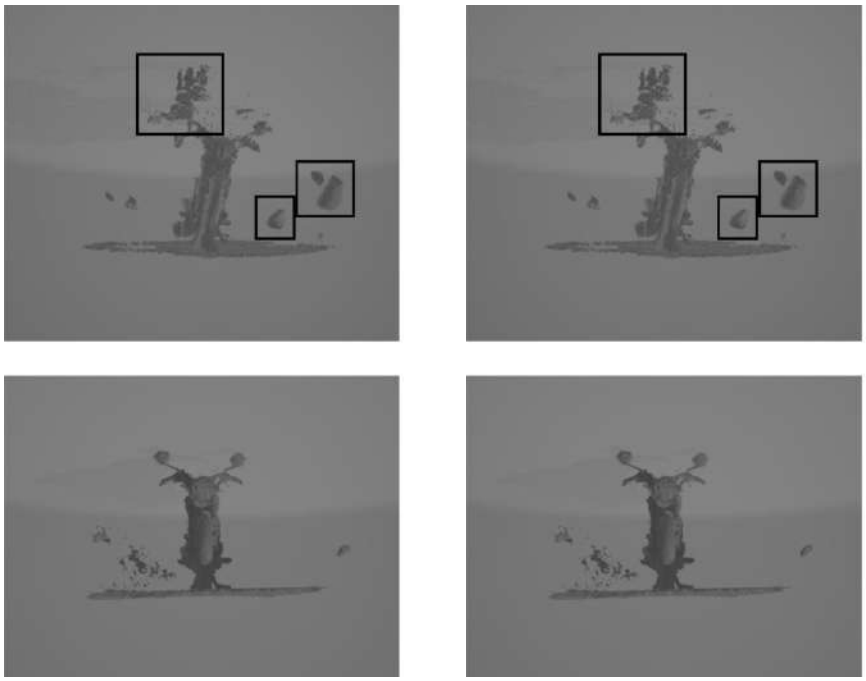


Figure 55 Conclusion 3

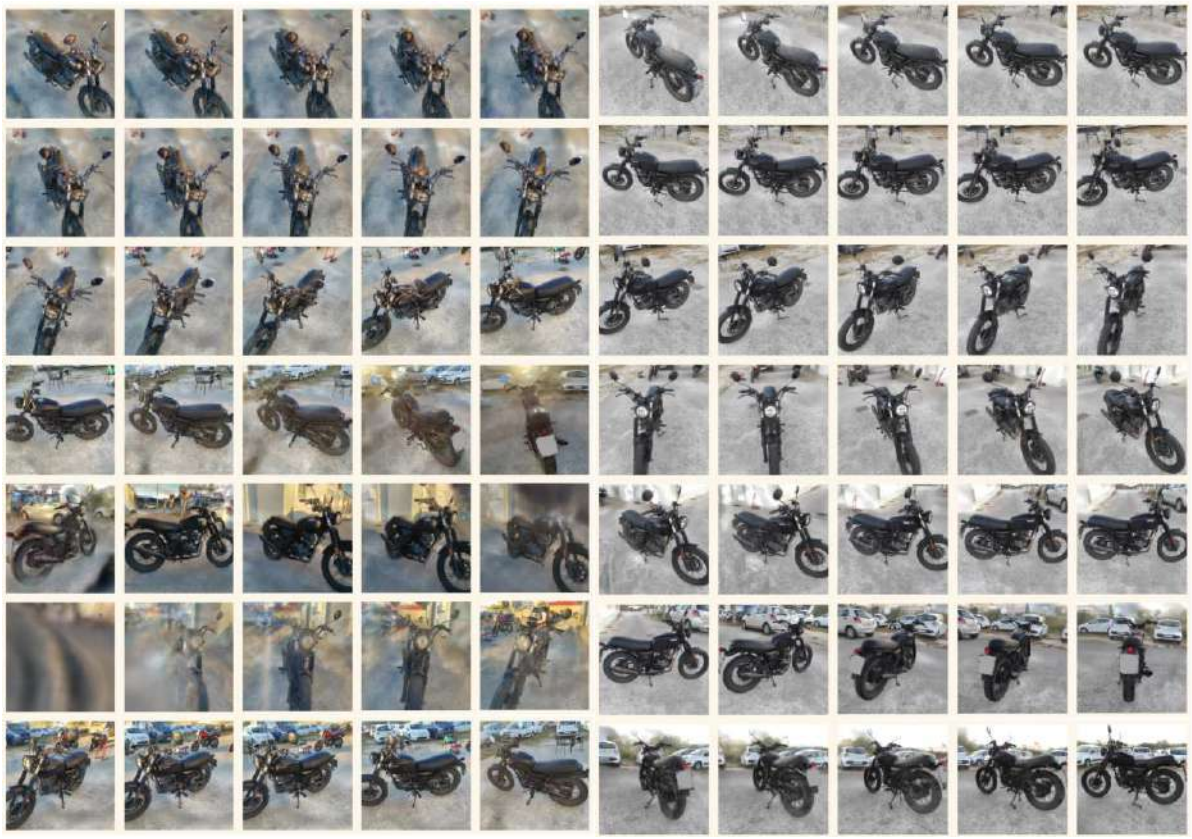


Figure 56 Custom 360 scene, lighting conditions comparison, Plenoxel results comparison

Conclusion:
Light plays a crucial role in photogrammetry as it directly impacts the level of detail that can be captured in the images. The quality and intensity of light determine how well the camera can capture the surface details and textures of the objects being imaged. Adequate lighting is necessary to ensure that the camera captures sharp, detailed images that allow precise measurement and modeling.
When the lighting conditions are good, the camera can capture images with high contrast, sharp edges, and fine details. This is important because photogrammetry relies on identifying corresponding points on multiple images, which are then used to triangulate the 3D positions of the points in space. The more details that can be captured in the images, the more accurately the points can be identified and the higher the level of detail in the final product. By controlling the lighting conditions and using appropriate techniques and tools, it is possible to create highly detailed and accurate results.

5. Conclusion

Dataset production: during the image capture step, the most complete results were achieved when the camera followed a spiral shape around the object. Any other attempt did not produce the desired result, led to a more pure output, or failed. For example, if the object was rotated instead of moving the camera position, the data from the side that was not covered by the cameras was lost. Taking photographs more sparsely often led to missing some of the object's details which may be visible in the output data.
In many cases due to the insufficient dataset, the colmap step failed, unable to extract the necessary image pairs and produce the data for the point cloud required for the "training" process. Trying to apply the Plenoxels method to ready datasets found online was mostly unsuccessful due to the failure in the Colmap step and the inability to extract the data for the image rendering.
Another limitation of the Plenoxels method resurfaces when we tried to apply it to a dataset in a less controlled environment and lighting data. An image dataset taken outside resulted in pure results due to hard shadows present in the dataset because of the sunlight. This limits Plenoxels applications, especially on bigger objects and environments that cannot be transferred to a controlled environment. Lighting conditions proved to be crucial to the quality of the produced output.

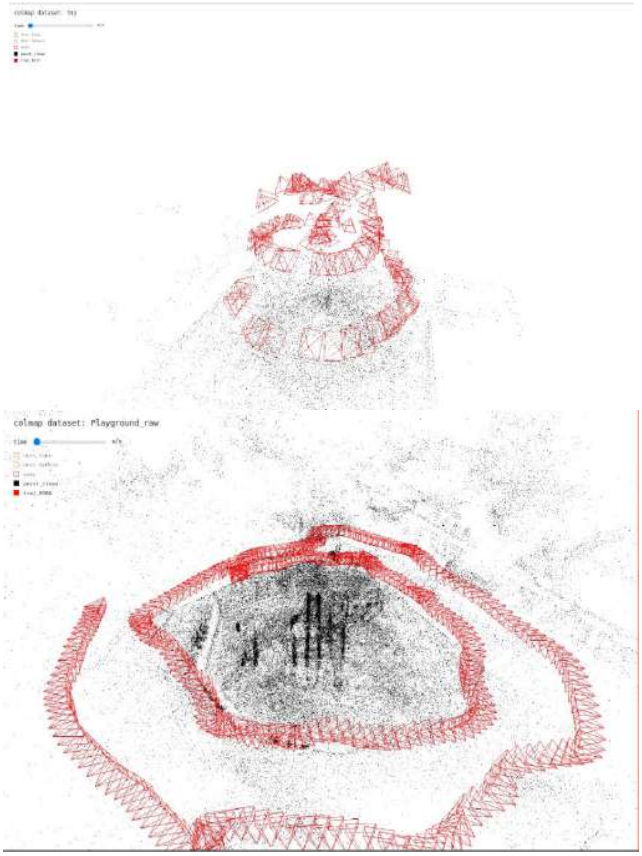


Figure 57 Conclusion 4

This led to the conclusion that the Plenoxels method cannot be easily applied to the majority of datasets created for photogrammetry purposes since they often do not follow the required camera position pattern. For example, it is very common practice in image dataset capture used in photogrammetry to rotate the object around its axis keeping the position of the camera stable. This is mainly done with smaller objects that are photographed in a controlled environment with consistent lighting to capture the maximum amount of details. However, evaluating these datasets with the Plenoxels method is not possible as it is, due to the limitations of the Colmap step.

This master thesis aimed to extract potential limitations that exist in the state of the art in photogrammetry. Through a comprehensive review of relevant literature and analysis of case studies, we have identified several areas where improvements and advancements can be made. By highlighting these limitations, we hope to contribute to the ongoing research in photogrammetry and provide a roadmap for future studies. We believe that this thesis has provided valuable insights into the challenges and opportunities present in this field, and we hope that our findings will inspire further research and development in the years to come. Ultimately, our goal is to contribute to the

advancement of photogrammetry as a powerful tool for 3D reconstruction and modeling in various industries, including architecture, engineering, and cultural heritage preservation.

In addition, our study focused on the Plenoxels methodology, and we identified several areas where this approach could be improved in future studies. One area of improvement for the Plenoxels methodology that we identified is the time-consuming step that uses Colmap software to extract the necessary information from the photographic data. Colmap is a powerful software tool used in the Plenoxels methodology to extract necessary information from photographic data for 3D reconstruction. However, this process can be quite time-consuming and requires a significant amount of computational resources. The Colmap pipeline uses a general-purpose reconstruction algorithm that relies on incremental structure-from-motion to recreate 3D structures from an image dataset taken from various viewpoints. This pipeline involves the extraction of main features and their matching, followed by a geometric verification step that produces a scene graph used for model reconstruction. Starting with a two-view reconstruction, new images are incrementally added, and point triangulation and bundle adjustment are used to refine the model. Despite its power, the time-consuming nature of the Colmap step is a significant area for improvement in the Plenoxels methodology. This step is a bottleneck in the Plenoxels pipeline and can significantly slow down the overall process. Therefore, optimizing this step could greatly improve the efficiency of the Plenoxels method.

Furthermore, the Plenoxels method inherits the light-dependency of photogrammetry, which can affect the quality and accuracy of the final model. As such, there is a possibility to improve the method by addressing this issue, such as by using additional light sources or developing new algorithms to mitigate the effects of light variation.

Additionally, the Plenoxels method has the potential to take advantage of the hardware capabilities of modern devices, such as smartphones and tablets, which often have multiple cameras. This can allow for the collection of a greater number of images from different angles, leading to a more detailed and accurate 3D model.

Finally, we believe that the Plenoxels methodology has the potential to create new services and applications, such as the ability to combine data from a variety of photos of the same place to create a more comprehensive and detailed model. With further development and optimization, we hope that the Plenoxels methodology can become a valuable tool for various industries that rely on photogrammetry for 3D modeling and reconstruction.

In conclusion, the simplification of view synthesis and complex geometry generation is a crucial and constantly evolving topic in computer science, especially in light of recent technological advances. While traditional photogrammetry methods require a great number of sample photos and extensive post-processing to extract geometry, the intersection between photogrammetry and 360-degree video capturing offers a promising solution. This thesis explores the creation of detailed 360-degree videos directly from voxels-based data representations, bypassing the intermediate step of geometry extraction. By focusing on the "Novel View Synthesis" problem, we hope to contribute to the development of more efficient and practical methodologies for recreating real-world data based on simple datasets.

6. Appendix 1: Glossary & Background

Coordinate Systems:

Polar coordinate system: is a mathematical coordinate system in two-dimensional space. It represents a point in a plane using two values: a radial distance and an angle. In this system, instead of using the traditional x-y Cartesian coordinates, a point is located by specifying its distance from a fixed point called the origin and the angle between a fixed reference direction.

Geographic (geodetic) system coordinate system is a system used to specify the location of a point on the surface of the Earth. It is based on a three-dimensional model of the Earth and uses latitude, longitude, and altitude to define the position of a point. The geographic coordinate system is used in various applications, including navigation, surveying, and mapping. It is also the basis for the global positioning system (GPS), which uses a network of satellites to determine the location of a receiver on the surface of the Earth.

Cartesian (rectangular) geometric system: also known as the Cartesian coordinate system, is a mathematical system used to describe the position of a point in a plane or in space. Photogrammetry is mostly performed using a right-handed cartesian system. [2]

Projection:

Projection is a method used for image creation that creates a two-dimensional representation of a three-dimensional environment or scene which is reduced by one dimension [4]. It uses two points joined by a straight line and this way one point is projected on the other. This straight line is referred to as a ray [2].

Projection types:

Central projection: also referred to as perspective projection, is when the points are placed on a straight line in three dimensions and are projected on a single point that is positioned outside the examined figure[2]. In other words, a ray is sent from a point in space from a three-dimensional environment through a specific point in space that is called the center of projection. This point is called the perspective center or the projection's center [4].

Parallel projection: central projection with a projection center at infinity.

Orthographic projections: when a parallel projection is intersected by a line or plane at a specific angle. Characteristic examples of orthographic projections are maps.[2]

Point clouds:

Point clouds are usually created by 3D scanning technology and represent all the points of the coordinate system that have been scanned. It usually consists of multiple points in space that can be later used to create a 3D model.
A point in the point cloud has the *(x, y, z)* format which represents its position in the coordinate system. Another measurement is the vector to the point that gives the orientation of the point in the space as well as its color usually in RGB format. Another possible value that a point can contain is intensity which describes how bright a certain point is.
Usually, the data from a point cloud is used to create a 3D polygon mesh, which is created by connecting the points into triangles since most 3D software works with polygon meshes.

Sensors and cameras used for photo-scanning use geometric properties of the scanned subject to accurately represent it. These measurements include object scale and proportions, parallel lines, vanishing points, their relation to the surroundings, and perspective [4]. The projective geometry which includes the features described to define a scene and the observation angle is crucial to recreating the geometry and understanding the structure of the examined object or scene [4].
When it comes to photogrammetry, the method is the same but this time we have multiple input data that allow better understanding and scene reconstruction from multiple angles [4]. The challenging aspect of this method, however, is to identify which points of the input correspond to the same point of the depicted object.

The simplest case that we can examine is when we have two images. It is common to examine the image as a set of points, the point clouds that we mentioned earlier, and the term “point” is often used later in this thesis when referring to a particular part of the object or image that has an insignificantly small surface.
For the multiple-view reconstruction, we consider that there is correspondence between two points of the two images.

$$x_i \Leftrightarrow x_i'$$

It is hypothesized that there are two camera matrices P and P’ that produce the correspondence $PX_i = x_i$and $P'X_i = x_i'$ which means that the point Xi then projects to the two provided data points. However, the points and the matrix are not known, and the challenging part is establishing them.
If we know nothing about the camera calibration or have no information about the location the images were taken, it is still not possible to predict or identify where the photos were taken, the object world orientation leading to a certain ambiguity of the reconstruction. In this case, it does not matter whether we have many images of the object or just a few [4].
According to Hartley et. al. [4] it is possible to reconstruct a point set from multiple views data if there is a sufficient number of points that do not lie on just one of the various critical configurations. The reconstruction can be done using fundamental matrices that show the constraints of the points x and x’ in case they represent the same 3D point.

Signal-to-Noise Ratio (SNR)

With the term SNR, we can describe a ratio between the desired output and the undesired background noise.
Usually, Root Mean Square (RMS) is used to calculate the SNR:
Noise = σ(S) where *σ* is the standard deviation and S is the signal.
S can symbolize a signal in any of the RGB channels, raw channels RGrBGb etc.
Signal-to-noise ratio SNR is a relationship between signal and noise and very often is more important than the noise itself. SNR can be defined in a variety of ways, depending on the definition of the S signal.

It is often challenging to measure the quality of an image when compared to another one. Even if it is possible to describe one image as sharper or better quality. However, when the alterations are small it may even become impossible. To achieve a convenient scientific description of the quality of the image, a method called the Signal-to-Noise ratio is used, allowing us to identify the slightest difference in the quality compared to the original one.

$$SNR = P \,/\, \sigma = \frac{P}{\sqrt{P}} = \sqrt{P}$$

SNR is a method that can be applied to different problems, especially when signals are concerned. In imaging, it is used to describe the quality and the sharpness of an image. It can also be a tool used to measure the effectiveness of an image sensor or other image-capturing tools. It can also be used to compare the original image to the one produced using, for example, a compression algorithm.
In other words, Signal-to-noise allows measuring how rough or grainy an image is. A larger SNR number indicates that the output image is closer in quality to the original than that with a smaller SNR number.

$$SNR = \frac{signal\ power}{noise\ power}$$

One of the methods to measure SNR is to use Standard Deviation (STD), which is a method that describes the number of variations in a predefined set of values. Lower STD values mean that the measured values are close to the mean and are called expected values. On the other hand, the higher the value of STD, the greater is the spread of the outputs.

In this work, we will use SNR to compare the output of the Plenoxels method, based on the number of input images and the algorithm used. SNR allows categorizing the noise that appears on the output image and deciding optimal settings and input dataset size to produce the best possible output.

Triangulation: in computer vision is a process that produces a point in 3-Dimensional space based on input images. Another definition for triangulation is the process that separates a surface into a set of triangles.

Feature extraction: is one of the most significant points when it comes to image reconstruction from a dataset. It allows the identification of the important points in the images and their interpretation upon which the reconstruction is then based [25]

Global features: in image processing, global features are used to describe the image as a whole and are produced during the feature extraction step. It is the method usually performed prior to the feature-matching process in the reconstruction pipeline [25].

Local features: in image processing, local features are used to identify significant keypoints in the dataset that can later be used to create an accurate reconstruction. They are produced during the feature extraction step. It is the method usually performed prior to the feature-matching process in the reconstruction pipeline [25].

Scale Invariant Feature Transform (SIFT): is an algorithm used in computer vision to perform image-matching developed by David Lowe. It allows point matching through various views in a 3-dimensional scene and performs object recognition based on the view. According to Linderberg [26], this algorithm proves to be robust in real-world conditions of image matching.

Inlier: despite minor inconsistencies in terminology, the term inliers usually refer to data that is in error but is often difficult to differentiate from the correct values of the dataset.

Outlier: usually refers to data that is distant from the other points in the dataset, but is not necessarily an error

Homography: is a relationship between two images of the planar surface and is used in image registration or for reconstruction.

Bundle Adjustments (BA) : is a method that can be applied in order to optimize the reconstruction by reducing the errors of the re-projection following the feature matching process [22].

Random Sample Consensus (RANSAC): is an algorithm used in computed vision and developed by Fischler and Bolles that allows the estimation of general parameters and deals with a high numbers of outliers in the input dataset. This approach produces candidate solutions using the smallest possible set of data points. The algorithms first selects a random number of points to identify the parameters of the models and then solves for those parameters in order to select a number of points that agree with a predefined tolerance value. In case of a large number of inliers that exceeds the threshold value, the algorithm reestimates the parameters' values with the inliers found and then terminates. In the opposite case, it repeats the process until the desired output is reached. [27].

Self-calibrating metric reconstruction systems: this is a method used for first systems of unorder photo collection on the Internet and for urban scenes [20].

Extrinsics

In computer vision, extrinsic parameters describe the position and the orientation of the camera or image sensor in relation to the scene in a 3D space, represented by a rotation matrix and a translation

vector. These parameters can be used to represent the relationship between the coordinates of a point in the 3D environment and its corresponding coordinates in the 2D image captured by the camera. Extrinsic parameters can be determined by using various techniques such as stereo reconstruction, Structure from Motion (SfM) and Visual Odometry (VO). These parameters are used to align the images captured by the cameras, to build 3D models from multiple images, or to estimate the motion of the camera or an object in the scene.

Intrinsics

In computer vision, intrinsic parameters are used to describe internal parameters or a property of a camera or image sensor that is independent of the scene in a 3D space. These parameters include things such as the focal length of the lens, the size of the image sensor, and the location of the principal point (the point at which the lens's optical axis intersects the image sensor). They can be used to describe the relationship between the coordinates of a point in 3D space and its corresponding coordinates in the 2D image that is captured by the camera. Typically, they are determined by calibrating the camera using a calibration pattern and then be used in various computer vision algorithms such as image rectification, stereo correspondence, and 3D reconstruction.

Scalar Field

A 3D scalar field is a mathematical function that assigns a scalar value (a single numerical value) to each point in a 3D space. It is a type of mathematical function that is defined over a 3D domain and maps to a scalar value. It is commonly used in physics and engineering to model physical phenomena such as temperature, pressure, and density.

In computer vision, a 3D scalar field is used to represent an implicit surface, which is a surface defined by a scalar function. The surface is the set of points where the function has a particular value, typically zero. The Marching cubes algorithm is one of the methods that generates a polygonal mesh from an implicit surface representation, such as a 3D scalar field.

It is important to note that a scalar field is different from a vector field, which assigns a vector (a set of numerical values) to each point in a 3D space and can be used to represent things like velocity, acceleration, or force.

Marching cubes

Once the vertices on the surface have been identified, the algorithm generates a set of triangles that connect the vertices to approximate the surface. This is done by considering each voxel as a "cube" and then connecting the vertices on the surface to form "edges" on the cube. The algorithm then looks at the combination of vertices on the surface and generates the triangles that are needed to approximate the surface.

The algorithm is efficient and generates a smooth polygonal approximation of the surface, and it is widely used in 3D computer graphics, scientific visualization, and medical imaging. Marching cubes is a type of implicit surface rendering algorithm, and it is one of the most widely used methods for isosurface extraction.

It was first published by Lorensen and Cline in 1987, and it quickly became a popular method for generating polygonal models from 3D scalar fields. Since then, many variations and optimizations of the algorithm have been proposed, such as the Marching Tetrahedra, Dual Contouring, and Surface Nets.

Marching cube in point Clouds

The marching cubes algorithm can be used to generate a polygonal mesh from a point cloud. The process of using the marching cubes algorithm on a point cloud is often referred to as "marching cubes on point clouds" or "MCPC".

The basic idea is to convert the point cloud into a 3D scalar field, in which each voxel in the grid is assigned a scalar value based on the density of points in that voxel. Then the algorithm is applied to the scalar field, just like it would be applied to any other implicit surface representation.

The density of the point cloud can be approximated by using techniques such as kernel density estimation, which constructs a smooth function that estimates the density of the point cloud at each point in 3D space. Other methods to convert point clouds to voxels, such as the use of an occupancy grid, can also be used.

It is worth mentioning that, due to the sparsity of point clouds, this method tends to create a lower-resolution mesh than if it were applied to a dense scalar field. However, it can still be useful for generating a polygonal approximation of the surface represented by the point cloud.

Keep in mind that the MCPC method may not necessarily produce the most accurate representation of the surface, and other methods, such as Poisson surface reconstruction, may give better results in some cases.

Regularization techniques

Regularization techniques in computer vision are methods used to prevent overfitting in machine learning models. Some examples of regularization techniques include:

L1 and L2 regularization, which add a term to the loss function that penalizes large weights in the model.
Total Variation (TV) regularization is a technique used in image processing and computer vision to smooth images while preserving edges. It is based on the idea that an image with a lot of "texture" or

"structure" is not smooth and that adding a penalty for this texture to the optimization problem used to recover the image will make it smoother.

The total variation regularization term is the sum of the absolute differences of the image intensity values for each pixel and its neighbors. This term encourages the solution to have a piecewise constant or piecewise linear behavior, which leads to sharp edges and piecewise constant regions in the solution. This technique is used to denoise images and improve their visual quality by reducing the amount of noise and preserving edges and fine details. It is also used to restore images that are degraded by noise or blur. It is particularly useful for images with piecewise constant or piecewise smooth regions, where traditional smoothness-based regularization methods may not work well.

It is used in various applications like image denoising, deblurring, super-resolution, and inpainting [51].

Sparsity

In computer vision, sparsity refers to the property of a signal, image, or model that has a small number of non-zero or non-negligible values. This can be beneficial in certain image processing and computer vision tasks, such as image denoising, inpainting, and compression, as well as in the development of models for object recognition and tracking [52].

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is an optimization algorithm for updating the parameters of a model, such as a neural network, based on the gradient of a loss function with respect to the parameters. The algorithm proceeds by iteratively updating the parameters in the opposite direction of the gradient, with the step size determined by a learning rate. The "stochastic" in the name refers to the fact that the algorithm uses a random sample, or subset, of the training data, called a "mini-batch", to compute the gradient at each iteration, as opposed to using the full dataset. This can make the algorithm more efficient and can also introduce some randomness, or "noise", into the optimization process, which can help the algorithm escape local minima and converge to better solutions [53].

7. Appendix 2: Experimental testbed & Technical details

NVIDIA

NVIDIA is a leading provider of state-of-the-art tools for computer vision, offering efficient pipelines and compatibility between APIs required to solve computer vision problems. Their tools provide memory and resources management, making it easier for developers to access and use the APIs. NVIDIA's implementation of algorithms for a variety of backends is fast, efficient and handles various image processing blocks such as feature detection and geometrical transformation. The NVIDIA Performance Primitives library is a key feature of their offerings, allowing for image and signal processing and is a part of the CUDA Toolkit. Additionally, NVIDIA offers face detection solutions, providing a comprehensive suite of tools and resources to meet the needs of developers working in computer vision [28].

PyTorch CUDA

PyTorch is an open-source framework for machine learning models that supports automatic differentiation and allows computation using graphical processing units (GPU) [34]. It is heavily used in machine learning to train neural networks and allows the performance of scientific computations with similar functionalities as CPU tensors [6]. PyTorch began as a Python library, but today most of its logic is written in C++ which allows it to achieve lower expenses when compared to most of the frameworks and allows multiple threads compared to limited parallelism in Python [34]. Multithreading is a core functionality when it comes to computations using multiple GPUs and is crucial for computation cost reduction [34].
PyTorch gives its users the ability to create custom differentiable operations. However, the downside of the automatic differentiation in PyTorch is the fact that it provides little visual data to the users [34].
One of the main uses of PyTorch is the training of machine learning models using GPUs which makes it necessary to deal with the limitations of the GPU memory capacity. PyTorch approaches this issue by freeing the intermediate values as soon as they are not needed anymore [34]. Another challenge that PyTorch solves is the reference cycles that can occur when automatic differentiation is naively approached. The framework solves this issue by recording a so-called “saved variable” which represents a pointer to a function instead of a full-fledged variable [34].

Cuda is a toolkit and programming model developed by NVIDIA that allows performing heavy computations using the parallelizing functionalities of GPUs, performing these heavy computational operations much faster than GPU.
PyTorch CUDA library allows asynchronous execution of tasks which significantly increases the number of computations that can be executed simultaneously and reduces this way the time requirements for the operations. It automatically distributes the task and distributes the data among the resources available [5]. It supports define-by-run dynamic execution where the function is differentiated by running the computation, compared to static graph-structured frameworks such as TensorFlow

which is differentiated ahead of time and then runs a number of times [34]. PyTorch additionally supports immediate eager execution which runs the tensor computations when they are encountered avoiding this way the materialization of a “forward graph” and records only the necessary data to differentiate the computation [34].

CuDNN is a Deep Neural Network library that provides the ability to deploy the power of GPUs for deep neural networks and to boost the performance of the training process of the network [32]. It accelerates some of the most popular deep learning frameworks such as TensorFlow and Pytorch [32].

In this work, PyTorch CUDA is used to perform differentiable volume rendering [7].

- Tensor computations using GPU
- Deep Neural Network (DNN) [https://developer.nvidia.com/deep-learning-frameworks]
- Flexibility, reduced training duration

TensorFlow

TensorFlow is an open-source software library under Apache 2.0 license [33] that includes both machine learning algorithms and their implementation designed for machine learning and developed by the Google Brain team within the Machine Intelligence research organization of Google [29][33]. The flexibility of the TensorFlow system allows the deployment of machine learning models on various environments including servers, mobiles, GPUs, and CPUs [31][33]. To perform the computations, TensorFlow uses flow graphs with the nodes of the graph representing mathematical operations and the edges representing tensors (multidimensional data arrays) [30] [31]. Each of the nodes in the graph has zero or more inputs and similarly zero or more outputs and is usually executed multiple times in the computations [33]. It is possible to perform these computations using CPUs or GPUS and this is the point where CUDA becomes useful allowing to accelerate GPUs power and offering parallel computations. The main components of the TensorFlow system are the client, the master, and the worker processes that can be executed locally on a single machine or rather be distributed between multiple devices using remote resources that are all managed by the scheduling system of the TensorFlow [33].

The results of the TensorFlow are visualized using TensorBoard, a suite of visualization tools that allows the TensorFlow users to better understand the behavior of their system and visualize the computation graphs in an interactive and customizable way [31][33].

PyTorch Cuda vs TensorFlow

PyTorch CUDA and TensorFlow are one of the most popular frameworks to perform machine learning computations and training these days. Both frameworks represent state-of-the-art systems and provide with a great number of features that can produce outstanding results which makes them the main

opponents in the debate of which framework is superior. The key points of the debate revolve around the availability of the state-of-the-art models that are available in each framework, the infrastructure of the deployment in the training process, and each framework's ecosystem.

Availability of the model: creating a complete deep learning model from scratch is a very complicated process that requires deep knowledge and experience from the user. The availability of pre-trained models is what becomes a crucial point in the simplification of the training process for the user and becomes a turning point when choosing a suitable framework.
Both PyTorch Hub [36] and TensorFlow Hub [31] official websites provide a variety of pre-trained models that can be deployed by users free of charge. Platforms such as HuggingFace [35] also provide open-source state-of-the-art models for machine learning and can be used to compare the number of models available for each of the frameworks. According to HuggingFace[35], more than 50 000 models are available for the PyTorch framework compared to more than 5 000 for TensorFlow which gives an advantage to PyTorch when it comes to pre-trained models.

Deployment
It is not only the availability of pre-trained models that play an important role when choosing a framework, but also the ability to deploy it on various platforms. The accessibility of each framework to users is what marks them stand out among their opponents and provides them with continuous community growth. When choosing a framework to work with, it is crucial to consider the support each one provides when it comes to the deployment of various devices and the ability to convert the models between them.
It seems like TensorFlow and PyTorch followed quite different approaches when it comes to model deployment and in this section, we try to identify the strong and weak spots of each one.

TensorFlow Serving [39] is a highly flexible serving system for the execution of Machine Learning models in the production environment and allows the deployment of the models on servers locally or using the cloud. The integration of TensorFlow models is provided with the TensorFlow serving, but according to official documentation [39], it also supports the deployment of other data and models. TensorFlow Lite is a library that allows the deployment of models on mobile devices as well as microcontrollers and other devices [40]. According to official documentation [40], it is not only possible to use one of the pre-trained models but also to convert any standard TensorFlow model to TensorFlow Lite one and easily deploy it on various devices such as Android and iOS.

The deployment functionalities of PyTorch have been much more limited, however, through recent years PyTorch Live or PyTorch Mobile [38] and TorchServe [37] have been introduced that provide solutions to easily deploy PyTorch models and reduce the need for writing custom code. PyTorch Mobile is currently in the beta stage [38] and can be compared to TensorFlow Lite since it allows the execution of Machine Learning models on mobile devices for Android, iOS, and Linux. Another recent release is PyTorch [41] which is a mobile application developed by MetaAI that allows the creation of mobile demos deployed using PyTorch. It also provides ready-to-use models that can be tested on a mobile device such as object detection and image classification [41].

Comparing both frameworks, it is safe to conclude that PyTorch is trying to cover the gap that exists between it and TensorFlow framework. Considering the recently released deployment tools it is possible that the gap that exists between the two frameworks will be significantly minimized in the near future.

Ecosystem

When it comes to comparing frameworks it is also important to consider the ecosystem of each one. That includes the tools it provides, management, training options as well as the whole environment.

PyTorch provides a variety of tools and libraries with most of them directly accessible through the official website. Some notable examples are PyTorch Hub [42] that is an official website that provides access to pre-trained models, guides, and tutorials, and PyTorch-XLA [43] which is a python library that allows the implementation of PyTorch models on XLA (accelerated linear algebra) devices such as Google's Cloud TPUs. Other libraries accessible to users are Torchvision [44] which is a library designed for computer vision and TorchAudio [45], a library designed for audio and signal processing. The majority of libraries provided by PyTorch include a variety of datasets and models designed to implement the purposes of each specific library which makes working with PyTorch an organized process and reduces the required space and process costs.

Similar to PyTorch, TensorFlow also provided its users with a repository with pre-trained models, the TensorFlow Hub [46]. Apart from the machine learning models, tutorials and guides can be found on the hub that offers users a solid base for their machine learning applications for image, audio, and video processing. Another platform provided by TensorFlow is TFX [47], which allows moving machine learning models from research to production, as well as managing the production pipeline. It is worth mentioning TensorFlow.js [48], which is a library that offers the ability to create machine learning models directly in a web browser.

Identifying which framework works best is a complex discussion since both frameworks are well-documented and provide an advanced learning curriculum. It is important to consider the goal of each individual application and the target device or platform that the application will be used on before deciding on the framework to use. In this work, we try to address some of the key points that make each framework unique and differentiate them based on the goal of each user.

It is possible to perform TensorFlow computation with Cuda and allow it to use the power of multiple GPUs.

TensorFlow:

- Well documented
- Rich community support
- Better visualization tools compared to PyTorch
- More limited support for parallelization than PyTorch

- Static execution

Pytorch/CUDA:

- Dynamic execution
- Parallelization
- Computations using GPUs
- Multithreading

Anaconda

Anaconda, developed by Anaconda Inc. and initially released in 2012, provides a distribution of R and Python languages for scientific computations. One of the main advantages of Anaconda is the simplification of data management, which can be challenging for data scientists working with complex algorithms and large data sets. Anaconda was created specifically for data science and machine learning, offering tools for the deployment of models for neural networks, training of CNNs and GANs, and machine learning computations using GPUs to reduce training time using parallelization algorithms. Additionally, Anaconda provides a variety of data visualization tools that enable users to easily explore and communicate insights from their data. [16][17]

Conda

Conda is an open-source environment and package management system that runs on a variety of platforms such as Windows and Linux allowing the installation and update of packages efficiently. It was created to run Python programs but has been expanded to support any language. It allows, for example, running different versions of Python on the same device according to users' needs, reducing the requirement to switch environment managers. Conda is included in all versions of Anaconda and Miniconda.

Structure-from-Motion & Colmap

Structure-from-Motion (SfM) is a methodology that allows computer vision in photogrammetric applications. It produces the estimation of three-dimensional objects or structures and their corresponding camera trajectories using an image dataset as an initial input [22, 23, 24].

SfM strategies

Incremental: probably the most popular method for reconstructions from unordered photo datasets [20, 24]. The main disadvantage of this method is the scalability of incremental SfM algorithms. This approach is the one used in the Colmap method and will be discussed further in this work.

Hierarchical SfM: is an approach that allows solving structure-from-motion problems using a hierarchical structure. The method starts from local reconstructions that are later hierarchically connected. This method usually allows the avoidance of iterations and the use of less accurate estimates at the beginning of the process [22]. According to Xu et. al [23], a hierarchical approach can be beneficial to large-scale reconstructions and produce more accurate results compared to incremental or global SfM techniques.

Global Sfm: is a method that produces reconstruction using all images together. Despite the high accuracy of this approach, it is mainly used for small-scale datasets due to its high computational costs and the high complexity of the averaging methods used [24].

Colmap

Colmap is a free Structure-from-Motion (SfM) and Multi-View Stereo (MVS) pipeline that allows reconstruction from an image dataset and can produce a 3D mesh [19]. The approach of Colmap is based upon the algorithm proposed by Schonberger et. al [20] and uses SfM to recreate 3-dimensional structures from an image dataset taken from various viewpoints. It uses projective geometry in an attempt to map the feature points and estimate the points between the images of the dataset. In the Plenoxels method [7], the Colmap pipeline is used to process 360 degrees scenes. The algorithm behind Colmap is an attempt to build a general-purpose reconstruction pipeline.

Incremental Structure-from-Motion is a sequential processing pipeline that uses iteration as a reconstruction component. The process usually includes extracting the main features and their matching and then a geometric verification step. This produces a scene graph that is then used for the model reconstruction, starting with a two-view reconstruction, and then adding new images incrementally. Point triangulation process then follows and the model is refined using bundle adjustment (BA).

Structure of the Algorithm used in Colmap [20]

Step 1: **Image pairs production.**
Input images are used to find scene overlaps and projections of the same points in the images. This step produces image pairs that are geometrically verified and a graph that includes projections of each discovered point.
Input images

$$I = \{I_i | i = 1 \dots N_I\}$$

Step 2: **Feature extraction.**
For each of the dataset images I_i, the goal is to find sets of local features F at locations $x_j \in R^2$ that are represented using an appearance factor f_j.

Local features:

$$F_i = \{(x_j, f_j)|j = 1 \dots N_{F_i}\} \text{ [20]}$$

SIFT algorithm is used here to provide good robustness when detecting feature points in an image.

Step 3: Matching.
In this step, the goal of the algorithm is to discover images that observe the same part of the scene. This is done using the features Fi described above. The naivest approach is to examine every pair of images for overlaps and use fj features to compare the similarity metrics of the pair. Computational complexity:

$$O(N_I^2 N_{F_i}^2)$$

Disadvantage: not applicable for large image datasets (compares every pair to each other)

Output: pairs of possibly overlapping images
Pairs

$$C = \{\{I_a, I_b\}|I_a, I_b \in I, a < b\}$$

with feature correspondences

$$M_{ab} \in F_a \times F_b$$

Step 4: Verification of overlapping image pairs C.
This process is done using projective geometry to estimate the transformation of feature points between the images. If a sufficient number of features between the images is identified, the images then are considered verified.

Epipolar geometry: uses the essential or fundamental matrix to describe the relations of camera movement. It is possible to extend this method from two to three views with the use of *trifocal tensor* RANSAC here is used as a robust parameter estimation technique.

Output: pairs of images C in a stage graph where images are the edges and the verified pairs of images represent the nodes.

Step 5: geometric reconstruction.
The input of this step is the scene graph created in the previous step and the goal is to create pose estimates for the images and a set of points.
Pose estimates:

$$P = \{P_c \in SE(3)|c = 1 \dots N_P\}$$

Set of points:

$$X = \{X_k \in R^3 | k = 1 \dots N_X\}$$

The selection of an initial image pair is an important step in the process which ensures the algorithm's effectiveness and produces the initial two-view reconstruction. Choosing a dense location with many overlapping cameras, according to Schonberger et. al [20], usually results in more accurate and robust reconstruction.

Solving Perspective-n-Point (PnP) problem results in new image registration with the use of feature correspondences in registered images. The PnP problem produces the pose estimates and for uncalibrated cameras, it additionally estimates the intrinsic parameters (focal length and optical center of the camera). For calibrated cameras, RANSAC and minimal pose solver are usually used.

Step 6: Triangulation.

The triangulation method is used in the Colmap to produce an extended point set X. In the case when at least one image is covering a scene part, a point *Xi* of a scene can then be triangulated and added to a set of points *X*. The main aspect of the triangulation in the Colmap algorithm is that it increases the stability of the model and allows the registration of new images to the set.

Bundle adjustment is used in order to perform further adjustments to Sfm and reduce the projection error. In the algorithm [20], Levenberg-Marquardt method is used to solve problems related to bundle adjustments

Projection error

$$E = \sum_j \rho_j \left(\| \pi(P_c, x_k) - x_j \|_2^2 \right)$$

Function π that projects points of the scene into the image space

Pj : loss function to reduce outliers

The contribution of the Colmap algorithm [20] mainly consists of developing a strategy to improve the robustness of both the triangulation and initialization processes described above. Schonberger et al. [20] propose a next-best view selection and robust triangulation method with the main goal to improve the scene structure and simultaneously reduce the costs of computations. Another topic that the algorithm tries to resolve is the efficient selection of parameters for bundle adjustments and produce a complete state-of-the-art system.

Verification strategy: The authors [20] propose a multi-view geometric verification strategy by estimating the fundamental matrix and examining the image pair in order to verify it geometrically when a specific number of inliers is identified. Next, the classification of the transformation is performed and a number of homography inliers are identified. When an image pair is validated, the scene graph is labeled using the model's type such as planar or panoramic, as well as the inliers of the model. The goal is to use calibrated and non-panoramic image pairs for the reconstruction in order to improve the robustness of the triangulation process.

Next best image selection method proposed by Schonberger et. al [20]

The selection of the next best image is one of the crucial points of SfM that allows to reduce the errors of the reconstruction and has a great impact on the quality and accuracy of the produced output.

Popular next best image selection strategy is choosing an image that sees a greater number of triangulated points.
Schonberger et. al [20] propose a method where they both monitor the number of points that are visible and their distribution, allowing to identify these images as better-conditioned and register them first. The points that are better distributed and the point number increase the score S of the image. This is done by separating the image into a grid with each cell having two states: full and empty. During the reconstruction process, when a point that is labeled as empty becomes visible, the state of the cell is changed to full and the score S of the image increases. The method favors the points that are more uniformly distributed and to avoid the wrong capture of the distribution, the scheme is then extended to a pyramid with levels $l = 1…L$. This is achieved by further partitioning the image on each level. Schonberger et. al [20] support that this method significantly improves the robustness and accuracy of the reconstruction.

Robust triangulation method proposed by Schonberger et. al [20]
The goal of this method is to handle the high outlier ratio that may be present in feature tracks due to ambiguous matches resulting in the false merging of feature tracks. The errors are often enhanced by inaccurate camera poses. To resolve these issues in the Colmap algorithm [20], the authors state the importance of identifying a valid set of track elements and applying a recursive triangulation before proceeding to the refinement with multiple views. To construct the problem of multi-view triangulation, the authors use RANSAC with the goal of maximizing the support of measurements compatible with a two-view triangulation. RANSAC runs a number of iterations (K) in order to ensure that at least one set without outliers has been sampled. The iteration number K is adapted using recursion, reducing this way the costs of computations.
The next step is to perform Bundle Adjustments. First, local BA is performed after each registration using the Cauchy function as a loss function, and Ceres Solver [21] which allows the solving of complex optimization problems is used allowing the sharing of arbitrary complexity camera models. Global Bundle Adjustment is performed only when the model grows by certain degree. After performing the Bundle Adjustments, observations with a large projection errors are filtered. Since the BA improves the camera parameters, the authors [20] propose the addition of pre-BA and post-BA Re-Triangulation steps to improve the completeness and accuracy of reconstruction. The main idea behind those steps is to continue the tracking process only to points that are below the threshold instead of increasing its value. The performance of Bundle adjustments, Re-Triangulation, and filtering with iteration steps of optimizations significantly improve the quality of the results and stimulate the reconstruction.
Another significant improvement of the Colmap algorithm compared to other reconstruction techniques is the development of an improved parameterization of Bundle Adjustments. It is done using incremental SfM as a base and grouping the unnecessary cameras into sets of scene overlaps and this way proposing an efficient camera grouping technique. Another improvement concerning the camera grouping, Schonberger et. al [20] propose the partitioning of a scene into many small camera groups, instead of just one big submap. The cameras of each group are then merged into one reducing the computation costs and the sizes of the camera system.

Technical approach/specs

To perform the evaluations of the Plenoxel algorithm in this thesis, we used the following:
FRU Information: NVIDIA DGX station a100
GPU Information: 4 x NVIDIA A100-SMX4-40gb

NVIDIA DGX A100 which is a part of the NVIDIA DGX™ platform, the universal system for all AI workloads. It allows for solving big datasets and employing the power of GPU to perform parallel task and achieve great performance. NVIDIA DGX Station A100 is a high-performance computing platform designed for demanding AI and data science workloads. It features 4 NVIDIA A100 Tensor Core GPUs with a total of 320 GB of high-bandwidth memory (HBM2), and a pair of 64-core AMD Rome CPUs. It also includes a high-speed NVLink interconnect that allows the GPUs and CPUs to communicate efficiently, and fast NVMe-based SSD storage for data access. DGX Station A100 is built to support a wide range of AI and machine learning tasks, from deep learning to natural language processing and computer vision, and is designed to accelerate model training, inference, and data analytics. The platform runs on NVIDIA's CUDA software development kit and supports popular deep learning frameworks such as TensorFlow, PyTorch, and MXNet. It also includes pre-installed software, including NVIDIA's deep learning software stack and the NVIDIA GPU Cloud container registry, making it easy to get started with deep learning projects [64].

NVIDIA A100 Tensor Core GPU
The A100 is the flagship product in NVIDIA's data center GPU lineup, designed to accelerate high-performance computing, deep learning training, and inference workloads. It is built on the NVIDIA Ampere architecture and features 6,912 CUDA cores, 432 Tensor Cores, and 40 GB or 80 GB of high-bandwidth memory (HBM2) with a memory bandwidth of up to 1.6 terabytes per second [65].

8. Bibliography

[1] Statham, N., Jacob, J., & Fridenfalk, M. (2020). Photogrammetry for Game Environments 2014-2019: What Happened Since The Vanishing of Ethan Carter.

[2] Derenyi, E. E. (1996). Photogrammetry: The concepts. [Doctoral dissertation, Department of Geodesy and Geomatics Engineering, University of New Brunswick]

[3] Miller, C. L. (1957). The Spatial Model Concept of Photogrammetry.

[4] Hartley, R., & Zisserman, A. (2000). Multiple View Geometry in Computer Vision. Cambridge University Press.

[5] Run.ai. (2023). PyTorch on GPU: Deep Learning with PyTorch on GPU. Retrieved from https://www.run.ai/guides/gpu-deep-learning/pytorch-gpu

[6] IBM. (2023). Getting started with PyTorch. Retrieved from https://www.ibm.com/docs/en/wmlce/1.7.0?topic=frameworks-getting-started-pytorch

[7] Yu, A., Fridovich-Keil, S., Tancik, M., Chen, Q., Recht, B., & Kanazawa, A. (2021). Plenoxels: Radiance Fields without Neural Networks.

[8] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

[9] Yalçinkaya, S., & Delikanli, B. (2020). Variable Voxel Computing Method - Innovative Approaches to Reduce the Computing Load in Voxel-based Solid Modeling and New Representation Methods. In L. Werner & D. Koering (Eds.), Anthropologic: Architecture and Fabrication in the cognitive age - Proceedings of the 38th eCAADe Conference - Volume 1, TU Berlin, Berlin, Germany (pp. 663-671). https://doi.org/10.52842/conf.ecaade.2020.1.663

[10] Mileff, P., & Dudra, J. (2019). Simplified Voxel Based Visualization. Production Systems and Information Engineering, 8, 5-18. doi:10.32968/psaie.2019.001.

[11] Yu, A., Li, R., Tancik, M., Li, H., Ng, R., & Kanazawa, A. (2021). Plenoctrees for real-time rendering of neural radiance fields. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 5752-5761).

[12] Niemeyer, M., Mescheder, L., Oechsle, M., & Geiger, A. (2020). Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision. [Conference paper, 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)]

[13] Lombardi, S., Simon, T., Saragih, J., Schwartz, G., Lehrmann, A., & Sheikh, Y. (2019). Neural volumes: Learning dynamic renderable volumes from images. ACM Transactions on Graphics (SIGGRAPH).

[14] Häne, C., Tulsiani, S., & Malik, J. (2017). Hierarchical surface prediction for 3D object reconstruction. In 2017 International Conference on 3D Vision (3DV) (pp. 412-420). IEEE.

[15] Laine, S., & Karras, T. (2010). Efficient sparse voxel octrees–analysis, extensions, and implementation. NVIDIA Corporation, 2(6).

[16] Anaconda. (2023). Retrieved from https://www.anaconda.com

[17] Anaconda Documentation. (2023). Retrieved from https://docs.anaconda.com

[18] Ryan, J. (2019). Photogrammetry for 3D Content Development in Serious Games and Simulations. [Master's thesis, Institution Name]

[19] Colmap. (2023). Retrieved from https://colmap.github.io

[20] Schönberger, J. L., & Frahm, J.-M. (2016). Structure-from-Motion Revisited. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 4104-4113).

[21] Agarwal, S., Mierle, K., et al. (2023). Ceres Solver. Retrieved from http://ceres-solver.org

[22] Zhao, L., Huang, S., & Dissanayake, G. (2018). Linear SFM: A hierarchical approach to solving structure-from-motion problems by decoupling the linear and nonlinear components. ISPRS Journal of Photogrammetry and Remote Sensing, 141, 275-289.

[23] Xu, B., Zhang, L., Liu, Y., Ai, H., Wang, B., Sun, Y., & Fan, Z. (2021). Robust hierarchical structure from motion for large-scale unstructured image sets. ISPRS Journal of Photogrammetry and Remote Sensing, 181, 367-384.

[24] Zhu, S., Zhang, R., Zhou, L., Shen, T., Fang, T., Tan, P., & Quan, L. (2018). Very Large-Scale Global SfM by Distributed Motion Averaging. [Conference paper, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)]

[25] Hassaballah, M., & Awad, A. I. (2016). Detection and Description of Image Features: An Introduction.

[26] Lindeberg, T. (2012). Scale Invariant Feature Transform.

[27] Derpanis, K. G. (2010). Overview of the RANSAC Algorithm.

[28] NVIDIA for Computer Vision and Image processing. (2023). Retrieved from https://info.nvidia.com/rs/156-OFN-742/images/Webinar_PDF-Implementing_computer_vision_image_processing_solutions_with_VPI.pdf?mkt_tok=eyJpIjoiWVRnd05tSTVaVGxoTkdaaCIsInQiOiJNa09jeDg0bUpxcG95WjAzTzBuVVdJaFRRXC90OU5HUGhpVE0rN0RiRk5LUGFZN3VHTkRNV0hCZWtYZURUWmJ4SER4Wnl1cEd3d1RqVFhJZXRENnBtR1FmSVJISk5DZytTNzZwK1d1VHRmXC95d2VkTlRBbG1SbkdGRUpKcWNlVm0yIn0%3D

[29] NVIDIA. (2023). Deep Learning Frameworks with TensorFlow. Retrieved from https://docs.nvidia.com/deeplearning/frameworks/tensorflow-user-guide/index.html

[30] NVIDIA. (2023). Deep Learning Frameworks. Retrieved from https://developer.nvidia.com/deep-learning-frameworks

[31] TensorFlow. (2023). Retrieved from https://www.tensorflow.org/learn

[32] NVIDIA. (2023). cuDNN. Retrieved from https://developer.nvidia.com/cudnn

[33] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J., Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

[34] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic Differentiation in PyTorch. NIPS 2017 Workshop on Autodiff.

[35] Hugging Face. (2023). Retrieved from https://huggingface.co

[36] PyTorch. (2023). Retrieved from https://pytorch.org/

[37] PyTorch. (2023). TorchServe. Retrieved from https://pytorch.org/serve/

[38] PyTorch. (2023). TorchScript and Mobile. Retrieved from https://pytorch.org/mobile/home/

[39] TensorFlow. (2023). Serving. Retrieved from https://www.tensorflow.org/tfx/guide/serving

[40] TensorFlow. (2023). TensorFlow Lite. Retrieved from https://www.tensorflow.org/lite

[41] PlayTorch. (2023). Retrieved from https://playtorch.dev

[42] PyTorch. (2023). Hub. Retrieved from https://pytorch.org/docs/stable/hub.html

[43] PyTorch. (2023). XLA. Retrieved from https://pytorch.org/xla/release/1.9/index.html

[44] PyTorch. (2023). torchvision. Retrieved from https://pytorch.org/vision/stable/index.html

[45] PyTorch. (2023). torchaudio. Retrieved from https://pytorch.org/audio/stable/index.html

[46] TensorFlow. (2023). TensorFlow Hub. Retrieved from https://www.tensorflow.org/hub

[47] TensorFlow. (2023). TFX. Retrieved from https://www.tensorflow.org/tfx

[48] TensorFlow. (2023). TensorFlow.js. Retrieved from https://www.tensorflow.org/js

[49] Sloan, P. P. J. (2017). Deringing spherical harmonics. SIGGRAPH Asia 2017 Technical Briefs, n. Pag.

[50] Schönefeld, V. (2005). Spherical harmonics. Computer Graphics and Multimedia Group, Technical Note. RWTH Aachen University, Germany, 18.

[51] Chambolle, A., Caselles, V., Cremers, D., Novaga, M., & Pock, T. (2010). An Introduction to Total Variation for Image Analysis. In M. Fornasier (Ed.), Theoretical Foundations and Numerical Methods for Sparse Recovery (pp. 263-340). Berlin, New York: De Gruyter. https://doi.org/10.1515/9783110226157.263

[52] Wright, J., Ma, Y., Mairal, J., Sapiro, G., Huang, T. S., & Yan, S. (2010). Sparse Representation for Computer Vision and Pattern Recognition. Proceedings of the IEEE, 98(6), 1031-1044. doi: 10.1109/JPROC.2010.2044470.

[53] Alagözlü, M. (2022). Stochastic Gradient Descent Variants and Applications. 10.13140/RG.2.2.12528.53767.

[54] Borshukov, G., Piponi, D., Larsen, O., Lewis, J. P., & Tempelaar-Lietz, C. (2003). Universal Capture - Image-based Facial Animation for "The Matrix Reloaded". 10.1145/965400.965469.

[55] Sudmann, A. (2016). Bullet Time and the Mediation of Post-Cinematic Temporality, 297-326.

[56] Max, N. (1995). Optical models for direct volume rendering. IEEE Transactions on Visualization and Computer Graphics, 1(2), 99-108. doi: 10.1109/2945.468400.

[57] Liu, L., Gu, J., Lin, K. Z., Chua, T., & Theobalt, C. (2020). Neural Sparse Voxel Fields. ArXiv, abs/2007.11571.

[58] Park, J. J., Florence, P. R., Straub, J., Newcombe, R. A., & Lovegrove, S. (2019). DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 165-174.

[59] Rudin, L. I., Osher, S. (1994). Total variation based image restoration with free local constraints. Proceedings of 1st International Conference on Image Processing, pp. 31-35 vol.1. doi: 10.1109/ICIP.1994.413269.

[60] NVIDIA, Vingelmann, P., & Fitzek, F. H. P. (2020). CUDA, release: 10.2.89. Retrieved from https://developer.nvidia.com/cuda-toolkit

[61] Cook, S. (2012). CUDA Programming: A Developer's Guide to Parallel Computing with GPUs (1st ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[62] Anaconda Software Distribution. (2020). Anaconda Documentation. Anaconda Inc. Retrieved from https://docs.anaconda.com/

[63] NVIDIA. (2023). Deep Learning Frameworks. Retrieved from https://developer.nvidia.com/deep-learning-frameworks

[64] NVIDIA. (2023). NVIDIA GDX A100. Retrieved from https://www.nvidia.com/en-sg/data-center/dgx-a100/

[65] NVIDIA. (2023). NVIDIA A100 Tensor Core GPU. Retrieved from https://www.nvidia.com/en-sg/data-center/a100/