Individual Layer Scaling and Redundancy Reduction of Singleshot Multiplane Images for View Synthesis

Max Bergfelt

Master's thesis
2023:E19

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Abstract

Image-based rendering for view synthesis is a field within computer vision that has seen a growing number of research activity within the past few years, much due to deep learning techniques emerging, allowing researchers to re-format view synthesis as a learning problem. The multiplane image (MPI) is a recently proposed learning-based layered 3D representation, created from one or a few input images for the purpose of rendering novel views. When rendering with an MPI created from a single image, scaling becomes an inherent problem due to the lack of a distance reference point. The scaling issue prohibits smooth view transitions between different MPI and combining this with the inherently large file size of the representation makes multi-image scene applications difficult with single shot multiplane images.

In this thesis, a method is proposed to individually scale the layers of an MPI to improve view synthesis results and view transitions between different MPI in multi-image scenes. The method consists of an optimization algorithm which adjusts the individual layers of an MPI using their alpha values together with extrinsic depth information in the form of a sparse or a dense depth map. For testing, an MPI viewer program was created allowing viewing of multi-MPI scenes as well as free transitioning and blending between different MPI. Evaluations of the algorithm were performed on images from both a synthetic and a real-life SLAM dataset, resulting in higher than baseline scores for the method across the three most commonly used image similarity evaluation metrics. Additionally, the constructed method allows for removal of redundant layers in each MPI, reducing file size by over 73 % for the synthetic dataset and 78 % for the SLAM dataset.

Contents

Abstract **I**

Table of Contents **IV**

1 Introduction **1**
- 1.1 Aim . . . 1
- 1.2 Research Questions . . . 2
- 1.3 Contributions . . . 2
- 1.4 Thesis Outline . . . 3

2 Background **5**
- 2.1 Image Based Rendering and View Synthesis . . . 5
- 2.2 Multiplane Images . . . 6
  - 2.2.1 Multi-Layered Scene Representation . . . 6
  - 2.2.2 Layer Positioning . . . 6
  - 2.2.3 Rendering . . . 8
- 2.3 Related Work . . . 10
  - 2.3.1 Stereo Magnification: Learning View Synthesis using Multiplane Images . . . 10
  - 2.3.2 Single-View View Synthesis With Multiplane Images . . . 11
  - 2.3.3 Local Light Field Fusion . . . 11
  - 2.3.4 COLMAP . . . 12
  - 2.3.5 Neural Radiance Fields . . . 12
- 2.4 Evaluation Metrics . . . 12
  - 2.4.1 Peak Signal-to-Noise Ratio . . . 13
  - 2.4.2 Structural Similarity Index Measure . . . 13
  - 2.4.3 LPIPS . . . 14
- 2.5 Unity Engine . . . 14

3 Method **17**
- 3.1 Pipeline Overview . . . 17
- 3.2 Data Gathering . . . 18
  - 3.2.1 Synthetic Image Generation . . . 18
  - 3.2.2 Real Image Dataset . . . 20
- 3.3 Generation of Multiplane Images . . . 22
- 3.4 Multi Plane Image Scaling . . . 23
  - 3.4.1 Optimization Algorithm . . . 23
  - 3.4.2 Linear Solver . . . 24
  - 3.4.3 Layer Merging . . . 25
- 3.5 Viewer Application . . . 25

3.5.1 Overview . . . 26
3.5.2 Unity Scene . . . 27
3.5.3 Scripts . . . 27
3.5.4 Shaders . . . 29
3.5.5 MPI Blending . . . 29
3.6 Evaluation . . . 30
3.6.1 Blending . . . 30
3.6.2 View Synthesis . . . 30
3.6.3 Procedure . . . 31

4 Results **33**
4.1 Similarity Metric Scores . . . 33
4.1.1 Overall Comparison . . . 33
4.2 Qualitative Examples . . . 34
4.2.1 Depth Renderings . . . 35
4.2.2 Image Renderings . . . 36
4.3 Image and Depth Renderings . . . 36
4.3.1 Synthetic Data Depth Renderings . . . 37
4.3.2 Synthetic Data Image Renderings . . . 39
4.3.3 SLAM Data Depth Renderings . . . 40
4.3.4 SLAM Data Image Renderings . . . 41
4.4 Layer Reduction . . . 42

5 Discussion **43**
5.1 Quantitative Overview . . . 43
5.2 Representative Examples . . . 44
5.3 Single View MPI Generator . . . 45
5.4 View Transitioning and Blending . . . 46
5.5 Algorithm Run Time . . . 46
5.6 Layer Reduction . . . 46

6 Conclusion **49**
6.1 Research Questions . . . 49
6.2 Future Work . . . 50

Bibliography **51**

A Layer Reduction Results **53**

Chapter 1

Introduction

The multiplane image (MPI) is a layered 3D representation, consisting of several semi-transparent image planes. A visual example of an MPI can be seen in figure 1.1. Creating an MPI requires a 2D image which the MPI will use as its base, and some kind of accompanying external information on scene depth. However, the recent rise in popularity of neural networks has led to methods being created that allows depth information to be inferred from the geometry of a scene, allowing creation of multi-plane images without any input other than a single image[1]. While the simplicity of 3D generation using this method opens up for new possibilities such as 3D viewing of images captured with a regular smartphone camera, inferring scene depths from geometry has its limitations. With single-view input there is no way to determine the absolute scale of the scene, leading to errors being introduced in the synthesized views. This limitation puts a stop to the usage of single-view generated MPI in any situation where scale accuracy matters. To circumvent the scaling issue while keeping the simplicity of single-view generated MPI, we have posed the question of whether it is possible and effective to correct for scale after the MPI has been generated.

1.1 Aim

The research is conducted to investigate the possibility of scaling single-view generated MPI and whether this allows for multiple MPI to be combined into a larger scene representation that utilizes blending between multiple MPI views. All previous usages of MPI have spaced each layers by a fixed value in disparity in order to maximize resolution in the depth space, but we believe that optimal layer placement is highly scene-dependent. As depth information is becoming easier to access, with many smartphone cameras having software or hardware solutions for measuring depth, we can rely on these tools to individually scale the layers of an MPI. In addition, much like every other 3D representation MPI have both advantages and limitations, one

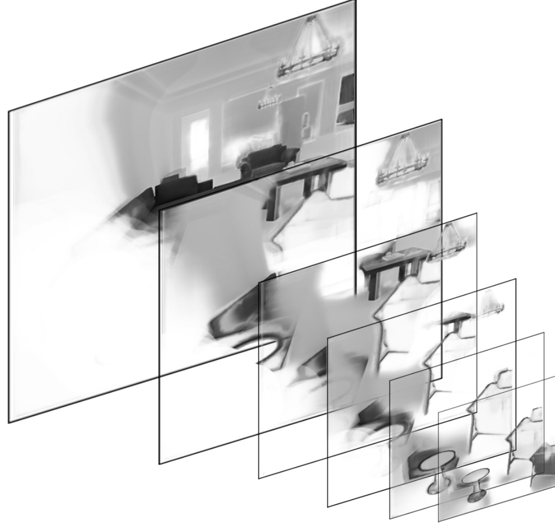


Figure 1.1: Visual example of a multi-plane image.

being the file size of the representation. We therefore also aim to find out whether re-scaling individual layers allows for redundant layers to be excluded without major quality reduction.

1.2 Research Questions

RQ1 Can layers of single-image generated multiplane images be scaled as a way to tackle the scale ambiguity problem?

RQ2 Are scaled single-image generated multiplane images a viable option for multi-image input view synthesis with MPI blending?

RQ3 If the estimated depths of two or more layers are close, can they be merged to reduce redundancy and compress the file size of the MPI representation without major loss of quality?

1.3 Contributions

The research conducted for this thesis contributes to the field of view synthesis in computer vision by investigating the following topics.

C1 Individual layer MPI scaling

C2 Consistent view transitions over multiple single shot MPI

C3 Compression of the MPI scene representation.

1.4 Thesis Outline

In chapter 2 of the report, all relevant background will be given needed to understand the content of this thesis, including what experiments have been done, how and what results they have given. A background will be given on what multiplane images are and what they are used, an introduction to relevant previous research will be given and the software and models used in this research will be explained. Chapter 3 contains an explanation of the work that has been done for this thesis with separate sections for each part of the pipeline. In chapter 4 of the report, the results given by the procedure explained in section 3 are presented, followed by chapter 5 containing a discussion in which the finished work of this thesis is analyzed by looking comparing the results with the method applied in this report and previous work.

Chapter 2

Background

2.1 Image Based Rendering and View Synthesis

Ever since it emerged, image based rendering has become a very popular application of computer vision. Unlike traditional computer graphics which utilize 3D meshed to render images, image based rendering techniques use 2D image captures of a single scene in order synthesize and render the scene from additional views. A practical example of an image-based rendering that many people may have used is Google map's street view navigation function. Some classic applications of image-based rendering include light fields [2], which are mathematical representations of a collection of light rays traveling through a scene and are used to generate realistic images from different viewpoints, or view interpolation [3] which allows interpolated image rendering between image pairs using precomputed depth maps.[4]

Recently, the rising popularity of technologies which could benefit from applications of image-based rendering such as virtual and artificial reality has increased the incentive for image based rendering. In addition, the rise of deep learning methods have brought new possibilities on how to perform view synthesis for image-based rendering, yielding impressive results. These two advancements have helped in revitalizing the field, which is now a hot topic within computer vision research. One recent technology for performing view synthesis which have benefited from such technological advancements is the multiplane image (MPI) representation [5]. Because of its simplicity and apparent potential for various applications, it has become popular and many research papers have been published on the topic during recent years.

2.2 Multiplane Images

Multiplane images were first introduced by Zhou et al. [5] as a way of tackling stereo magnification, the problem of extrapolating views from images captured by narrow baseline cameras. By representing scenes as a pair of fronto-parallel planes the authors managed to create a new image representation that outperformed many previous methods of synthesizing novel views from input images. Advantages with multiplane images over many other methods for view synthesis is that the scene representation can be predicted once and then used to generate novel views with minimal computation, instead of each separate view having to be predicted on its own. Another advantage with layered scene representations is the ability to model occluded content. This effectively allows "peeking behind corners" when rendering new images. View synthesis using MPIs has since it was first introduced been continued in several different works, such as Tucker and Snavely [1] and Srinivasan et al. [6].

2.2.1 Multi-Layered Scene Representation

An MPI is a three-dimensional image consisting of a fixed number of ordered two-dimensional RBGA images, referred to as layers or planes. Given a RGB image of an arbitrary scene with accompanying pixel-wise scene depth information, an MPI can be generated. Each plane in an MPI is close to a copy of the original image and one another with regards to the RGB-components, but has an alpha component that varies over the different layers. A specific layer in an MPI can therefore be referred to by its color and alpha components C_i and α_i, each with a resolution $W \times H$ matching that of the input image as well as 3 and 1 channels respectively.

An MPI is generated by first assigning each prospective plane to a depth value in the original image, $d_1, ..., d_D$, ranging between the minimum and maximum depth value of the scene so that $d_1 = d_{far}$ and $d_D = d_{near}$. The alpha value for every pixel in each plane is decided by the difference between the depth assigned to the plane and the relative depth value for the pixel, extracted from the accompanying pixel-wise scene depth information. As a result, planes assigned to a lower depth value will be less transparent at pixels depicting objects that are closer to the camera and planes assigned to higher depth values will be less transparent at objects further from the camera. For each pixel in each layer, the transparency is decided by the difference between the layer depth and the actual depth of the object depicted by the pixels. A visual example of a multiplane image can be seen in figure 2.1.

2.2.2 Layer Positioning

Previous research on multiplane images, Zhou et al. [5], Tucker and Snavely [1], Mildenhall et al. [7] and Flynn et al. [8] have all utilized the same strategy for determining the layer depths of the MPI. The strategy used is to space the individual back-to-front ordered layers of each MPI equally in disparity, which is defined as the distance in image location of an object viewed from two different positions. Given a scene such as the one shown in 2.2 containing two cameras with focal length f and baseline b, as

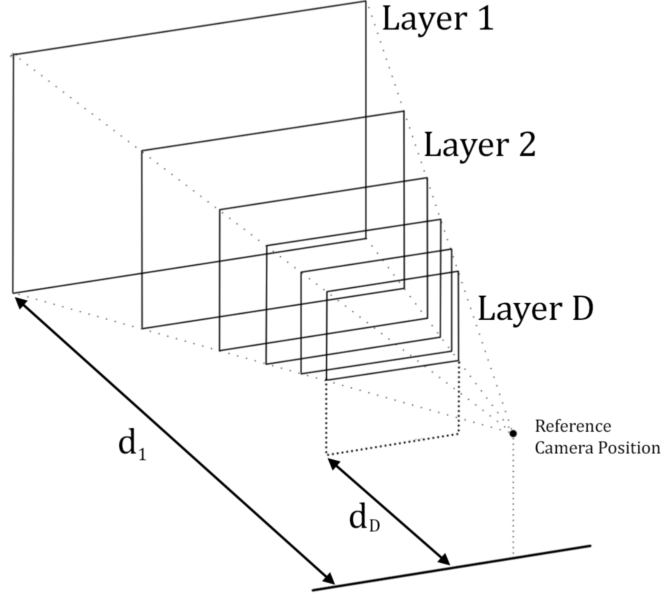


Figure 2.1: Drawing of an MPI being viewed from a reference camera position.

well as an object placed at a position (x, z), the disparity can be written as $(x_l - x_r)$, defined as the difference between the projected position of the object in the left-hand camera image plane, x_l, and the right-hand camera image plane, x_r. On inspection of the scene, it can be noted that for each camera, the right-angled triangle between the camera, the object and the point at a distance z in front of the camera is similar to the triangle between the camera, the projected object position and the point at a distance f in front of the camera. This gives results in the following relations

$$\frac{x}{z} = \frac{x_l}{f}, \quad \frac{x - b}{z} = \frac{x_r}{f} \tag{2.1}$$

Using these two equations, the relation between x_l and x_r can be derived as

$$\frac{x_l}{f} = \frac{x_r}{f} + \frac{b}{z}. \tag{2.2}$$

Finally, we can rewrite this expression, giving the relation between depth and disparity.

$$x_l - x_r = \frac{1}{z} f b \tag{2.3}$$

This shows that disparity is inversely proportional to depth and that the perspective effects between two views decrease inversely when the distance of the objects in the scene increases. Spacing the layers of an MPI linearly in disparity works as a general way to maximize view synthesis performance by covering as much of the disparity spectrum as possible. Likewise, an increase of the number of layers can be seen as increasing the disparity space resolution of the MPI.

However, this does not take into consideration the actual structure of most scenes and the typical depth value ranges of objects in most images. Spacing MPI layers equally in disparity places many of the layers at a small distance in front of the camera.

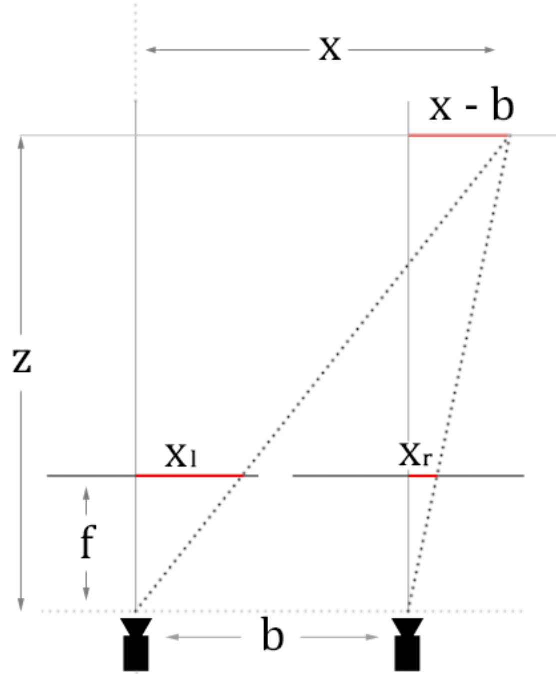


Figure 2.2: Setup of a scene with a single object seen through two cameras. f denotes the focal length of the cameras and x_l and x_r the position of the object in the image planes of the left and right camera respectively.

While this allows for correct rendering of potential textures at these distances in the scene, most data that the MPI models were trained on does not contain any textured objects at the distances of these layers. This leads the deeper layers having to do a lot more of the "heavy lifting" while many of the more shallow layers are left almost fully transparent, something that can be seen for most MPI generation methods, especially the ones creating more accurate MPI using more data such as DeepView[8]. The result is a waste of both disparity space resolution and file size of the representation.

2.2.3 Rendering

Novel images can be rendered from MPI's at arbitrary viewpoint positions. Given a source viewpoint T_s and a target viewpoint T_t, this is done by transforming each layer of the MPI, $\{C_i, \alpha_i\}$, from T_s to T_t with a homography H, giving transformed layers

$$C'_i = H_{T_s \to T_t}(C_i, d_i), \quad \alpha'_i = H_{T_s \to T_t}(\alpha_i, d_i). \tag{2.4}$$

In practice, this is equivalent to placing each layer at their corresponding depth d_i in front of the source image camera position in 3D space and re-scaling each plane so that it covers an equal amount of the field of view from the source viewpoint. An example of an MPI placed in 3D space is shown in figure 2.3.

After each layer has been transformed to the target viewing position, the image is rendered using the over operation as seen in equation (2.5).

$$\hat{\boldsymbol{I}}_t = \sum_{i=1}^{D}(C'_i\alpha'_i \prod_{j=i+1}^{D}(1-\alpha'_j)) \tag{2.5}$$

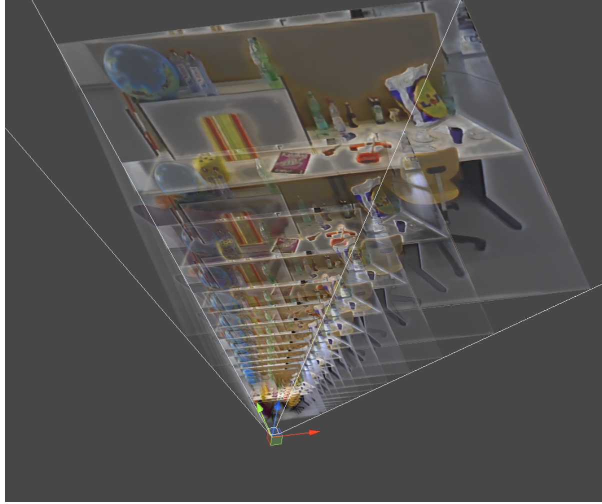


Figure 2.3: Multiplane image placed in a three-dimensional space.

Figure 2.4: Example of the "stack-of-cards effect" caused by rendering from a baseline that is too large.

This way of rendering is also referred to as alpha-compositing and takes a weighted average of the foreground and background colors, with the weight being determined by the alpha value. The equation reflects the intention of mixing the colors based on the degree of transparency in the foreground and allows creation an illusion of overlapping 3D behaviour in a 2D image.

Due to a limit to the amount of visual data that can be extrapolated from a single MPI, the baseline between T_s and T_t cannot be too large, while still retaining visual quality with minimal artifacts. Rendering an image with a baseline that is too large will result in visual artifacts, often through a "stack-of-cards effect" arising from a visial duplication of the layers edges that are not being correctly aligned. A example of a scene rendering using an MPI from the original camera position and from a baseline that is too large can be seen in figure 2.4.

Mildenhall et al. [7] proposed a guideline for sampling distance aimed at local light fields. Multiplane images can be seen as encodings of local light fields and these guidelines therefore allow direct application to multiplane images as well. The upper sampling distance limit Δ_u from Mildenhall et al. [7] for multiplane images is thus after slight modification giving by equation (2.6). In the equation, z_{min} and z_{max} denotes

the minimum and maximum scene depths, f the focal length, D the number of layers and Δx the camera spatial resolution (pixel size, mm/px). In summary, an MPI is rendered in two steps, layer positioning and alpha compositing.

$$\Delta_u \leq \frac{D\Delta_x}{f(1/z_{min} - 1/z_{max})} \tag{2.6}$$

2.3 Related Work

In the following section, previous research on multiplane images is presented, which has either been used directly in the work for this thesis through public code libraries or indirectly by following the procedures described in their academic papers.

2.3.1 Stereo Magnification: Learning View Synthesis using Multiplane Images

The first model trained to generate multiplane image representations presented in the original paper by Zhou et al. [5] used a fully convolutional encoder-decoder neural network. As input, the network took two images taken with a narrow baseline, I_1 and I_2 together with their camera parameters c_1 and c_2, consisting of position and orientation such that $c_i = (p_i, k_i)$. The network predicted the alpha components of the layers, α_i directly and the RGB image of the layers, C_i indirectly. The authors found that directly predicting C_i for every layer resulted in an over-parameterized output and instead opted to predict a background image $\hat{I}_{bg}$ as well as pixel-wise blending weights w_i for each layer. For the output of the predicted background image of the MPI $\hat{I}_{bg}$ and layer weights and alpha components w_i and α_i, the RGB image of each layer was found according to equation (2.7)

$$C_i = w_i \odot I_1 + (1 - w_i) \odot \hat{I}_{bg} \tag{2.7}$$

Training data In order to gather training data with accompanying ground truth, the authors built a pipeline capable of mining frames from Youtube videos, generating a dataset named RealEstate10K. The videos used came from a number of manually selected Youtube channels uploading real estate videos. The authors used the pipeline to identify videos with a moving camera and a static scene and to find favorable clips (low motion blur, distortion, editing effects, etc) within the videos. The resulting dataset generated consisted of over 7000 clips between 1 and 10 seconds length. A SLAM method, more specifically ORB-SLAM2 by Mur-Artal et al. [9] was used for generating the poses for the frames.

From the generated data, the authors extracted frame triplets $\langle I_1, I_2, I_t, c_1, c_2, c_t \rangle$ for training. The triplets were assigned randomly from sequences of 10 frames. This meant that depending on how the frames in the sequence were assigned, the network would have to either interpolate or extrapolate between the two source images for a

varying distance of frames. In addition, the variations in camera movement between scenes and sequences resulted in the baseline distance in both input and output to vary greatly between different training data points.

2.3.2 Single-View View Synthesis With Multiplane Images

In the work presented by Tucker and Snavely [1], which has been used as a basis for this thesis, the authors presented a way of generating MPI from single image input. The model created by the authors was a DispNet-style convolutional network [10] taking a single source image, I_s as input and outputting an MPI, $\{(C_1, \alpha_1), ..., (C_D, \alpha_D)\}$. The RealEstate10K dataset generated by Zhou et al. [5] was used for training with similar procedures. Video frames together with poses from SLAM was used, with the addition of a point set $\boldsymbol{P} = \{(x, y, d), ...\}$. Similarly, the output of the model were layer alpha values α_i and a predicted background image $\hat{I}_{bg}$. The RGB value of the layers, C_i was retrieved in the same way, by blending the source image I_s with $\hat{I}_{bg}$ according to (2.7). However, since the network did not output any blend weight w_i, they were given by the level of occlusion of the pixel in the corresponding layer, derived from the alpha values α_i according to equation (2.8) below.

$$w_i = \prod_{j>1} (1 - \alpha_j) \tag{2.8}$$

This effectively allows the network to predict the background layer $\hat{I}_{bg}$ so that occluded pixels in certain layers are used for inpainting.

Scaling Ambiguity When working with single images and no external information, scaling becomes ambiguous. The network will learn a scale appropriate for the data it is trained on, but as the scale of any single set of input data is arbitrary, correct scaling of any new input cannot be guaranteed. During training, this problem prohibits correct evaluation and since the generated MPI is not to scale with the scene it is generated from and rendering from the MPI at the pose of the ground truth will most likely not minimize the loss. To combat this problem, the authors introduced what they chose to call *scale-invariant view synthesis*. What this implies is the introduction of a scaling factor σ which is calculated from the point set $\boldsymbol{P}_s$ and a disparity map generated from the MPI. Including this scaling factor during rendering to scale the whole MPI up and down allowed the authors to bypass the scaling ambiguity. However, it should be noted that when generating images using MPI created by a pre-trained model, there is no way to avoid the scaling ambiguity without any additional external depth data to calculate the scaling factor σ.

2.3.3 Local Light Field Fusion

In the paper *Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines* by Mildenhall et al. [7], the authors present a method for performing view synthesis using a set of images irregularly sampled from a grid. Each

image in the grid is promoted to a multiplane image, acting as local light field representation, using the 4 most adjacent images in the grid. With a resulting grid consisting of several MPI, view synthesis is performed by a weighted blend between views individually rendered from a set of multiplane images with a reference position close to the current view position. This allows the scene to be rendered from with photorealistic results from any interpolated position within the grid. These results shows that multi-MPI scene representations are a viable option for performing view synthesis with excellent quality and that blending between views from multiple MPI can be used to increase the maximum baseline of the reference view at which the scene can be rendered with good quality.

2.3.4 COLMAP

COLMAP[11], is a general-purpose software tool for performing Structure from motion and reconstructing 3D datasets from images. COLMAP can be used to create sparse and dense scene reconstructions, generate depthmaps and more. COLMAP can be used either from its graphical user interface or from the terminal using its command-line interface. COLMAP offers several methods for feature extraction and mapping as well as different preset settings for camera parameters depending on the degree of information the user has about the scene.

2.3.5 Neural Radiance Fields

Another 3D representation for view synthesis that has gained a lot of popularity recently are neural radiance fields (NeRF) first introduced by [12]. The original NeRF consist of a neural network that has been trained on a couple of dozen input images taken of the same scene at different viewing positions. The input to the network is a single 5D coordinate consisting of camera position and rotation, and the output of the network is the volume density and the emitted radiance at the corresponding position which can be projected to a RGB image using classic volume rendering techniques. The NeRF model is therefore trained on a single scene and the scene itself is therefore stored in the weights of the network. An advantage of NeRF to MPI is very good rendering quality from a larger baseline with relatively few input images. However, the classical NeRF method takes a long time to train and has difficulties rendering images in real-time. Since it also utilizes a neural representation of the scene, it is not as visual of a representation as MPI and it is not user-editable in the same way.

2.4 Evaluation Metrics

When evaluating the similarity between two images, a common and simple strategy is to evaluate the difference between each pixel in the two images, one by one. However, in many scenarios we ideally want to use a metric which evaluates images similarly to how the human visual system does, which quickly becomes a more difficult task. For per pixel evaluation, the L1 norm has been shown to be preferable over the L2

norm[13], but these metrics are often deemed insufficient. The human visual system is highly capable of identifying structural patterns in images and evaluation metrics that can mimic this behaviour are preferable. In this chapter, a few common metrics for image similarity that have been used for evaluating the work in this thesis are introduced.

2.4.1 Peak Signal-to-Noise Ratio

Peak signal-to-noise ration (PSNR) is a metric for evaluating the noise between two images. PSNR is most simply defined using mean squared error and for 8-bit images with peak pixel values $2^8 = 256$, the formula is given as

$$\mathrm{PSNR}(f, g) = 10 \log_{10}(255^2/\mathrm{MSE}(f, g)). \tag{2.9}$$

.

As seen from the equation, PSNR is simply a logarithmic measure of inverse MSE. Since PSNR is a logarithmic quantity, it is defined in dB and a low MSE gives a higher PSNR score, meaning less noise between the images evaluated.

2.4.2 Structural Similarity Index Measure

Developed by Wang et al. [14], the structural similarity index measure (SSIM) is a measure of image similarity on a scale from 0 to 1, designed for comparing structural differences in images. It is given by the product of three factors being loss of correlation, luminance distortion and contrast distortion. Given a reference image f and a test image g, the SSIM with each of the three factors weighted evenly is given as

$$\mathrm{SSIM}(f, g) = l(f, g)c(f, g)s(f, g). \tag{2.10}$$

Each of these factors are defined using the mean pixel values μ_f and μ_g, the standard deviances σ_f and σ_g as well as the covariance σ_{fg} as

$$l(f, g) = \frac{2\mu_f\mu_g + C_1}{\mu_f^2 + \mu_g^2 + C_1} \tag{2.11}$$

$$c(f, g) = \frac{2\sigma_f\sigma_g + C_2}{\sigma_f^2 + \sigma_g^2 + C_2} \tag{2.12}$$

$$s(f, g) = \frac{\sigma_{fg} + C_3}{\sigma_f\sigma_g + C_3} \tag{2.13}$$

where the constants $C_1, C_2, C_3 \ll 1$ are included to avoid instability for denominator values close to zero.

2.4.3 LPIPS

One of the most recent metrics for evaluating image similarity was introduced Zhang et al. [15] and is called Learned Perceptual Image Patch Similarity (LPIPS). Unlike PSNR and SSIM, which are rather simple algorithms, LPIPS computes similarity by comparing the activations of two images for some predefined deep neural network. It was found that utilizing these deep features of a neural network resulted in a metric that might be more similar to the complex way humans evaluate similarity between images. The implementation released by the authors include implementations for the SqueezeNet, AlexNet, and VGG networks, with AlexNet being recommended. Since it was introduced, LPIPS has become a very popular metric for evaluation in image based rendering research, especially related to view synthesis.

2.5 Unity Engine

Unity Engine [16] by Unity Technologies is a cross-platform 3D engine used to create content such as simulations, visualizations or games. The Unity Engine is a useful platform in development of applications built on image-based rendering as it provides a number of features and tools that are specifically designed for working with 3D graphics and image data. Some of these features include

- A highly customizable rendering pipeline
- Integration with external libaries (OpenCV, Tensorflow, etc)
- Support for scripting and image processing tools

To give a basic understanding on how Unity works, some of the core features of its architecture are explained below.

Scene The Unity scene is an essential part of an application hosting content. Every application contains at least one scene which might contain part of or the whole application.

Game Object Each object in a Unity scene is a Game Object. They can be seen as containers that do nothing on their own but fulfill different functions depending on what components they have.

Component Component are functional elements that can be attached to Game Objects. The component attached to a Game Object define its behaviour and each component might contain a number of editable properties.

Transform Standard component attached to all Game Objects. The Transform components contains properties corresponding to position, rotation and scale.

Script A script is custom user-created component whose behaviour is defined by C# code. Scripts can be used to modify an application in various ways such as interacting with Game Objects, components and user input.

Camera A camera is a Game Object with a Camera component attached to it. Cameras are used for capturing the state of the world and displaying it to the player. A scene can contain one or several cameras.

Shader Shaders in Unity are pieces of code containing instructions on how to render objects. Scripts interact with objects through a Material component.

Compute Shader Compute Shaders is a type of shader containing arbitrary code written to be ran on the GPU.

Scene Hierarchy In the Unity Editor, a list of all Game Objects existing in the scene. Game Objects are organized through parent-child hierarchies where a child Game Object transform can be defined relative to its parents transform.

Texture A 2D image file used for giving a game object a visual effect during rendering. Textures can be assigned as properties to certain components.

RenderTexture A texture that can be rendered to by cameras during runtime or through code. Useful for applying image-based rendering effects.

Chapter 3

Method

3.1 Pipeline Overview

To evaluate a scaling algorithm for single view generated multiplane images, a reliable way of generating and viewing MPI is required. In addition, since scaling requires depth information, a method of extracting this information is needed for any dataset used. For this purpose, a pipeline was constructed consisting of the following steps

0. (Synthetic image generation)
1. Depth information extraction
2. Single-view MPI generation
3. MPI Scale Correction Algorithm
4. Unity MPI Viewer application

Since many of the trials for this thesis has been done with synthetically generated data, the step for generating said data is included in the pipeline as an optional step 0. If data is instead imported, the first step is extracting depth information for use in the scaling algorithm. After the images have been scaled using the depth information extracted in the previous step, the multi-MPI scene can be viewed in the MPI viewer application. The viewer allows blending between several MPI and exporting rendered images that can then be evaluated. A visual overview of the pipeline can be seen in figure 3.1. In the following part of this section, each part of the pipeline and how they were constructed is explained in more detail.

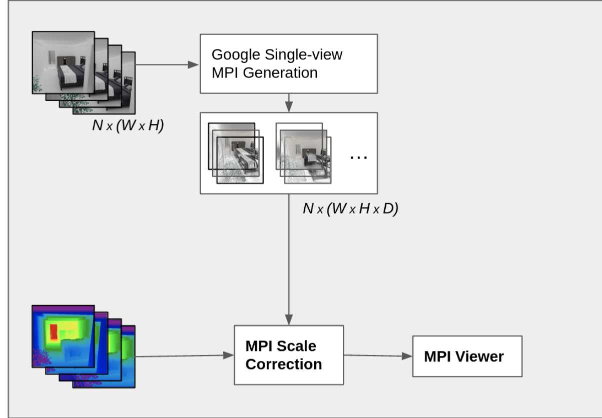


Figure 3.1: Overview of the pipeline for scale correction and viewing of multiplane images.

3.2 Data Gathering

For testing and evaluating the scaling algorithm, two separate datasets were used. To prove that the developed model has potential real world application, a dataset of real images, with corresponding SLAM-mapped camera positions was used. However, in order to test in an environment where errors introduced by the actual capture and mapping of a real image dataset can be avoided, an additional dataset consisting of synthetically generated images was introduced.

3.2.1 Synthetic Image Generation

A disadvantage with using real data taken by a handheld camera is that many testing parameters, such as distance intervals between images, camera positions and camera angles cannot be controlled accurately. Since our algorithm requires depth information, another disadvantage is that this depth information may not be estimated accurately enough. All of these factors can potentially contribute to errors in the evaluation of the scaling algorithm. This makes it difficult to evaluate any error to a specific part of the pipeline. Since capturing a large scale dataset while controlling these parameters would require extensive equipment such as depth cameras and a capturing setup, utilizing synthetically generated images provides a good compromise, given that the single-view MPI generation model performs similarly well on these images. Generating images synthetically allows for great flexibility and precise control over all image and camera parameters as well as accurate and dense depth information. For this reason, much of the experiments in this research has been performed with synthetically generated images.

Camera parameters To ensure that generated synthetic images would work well with model by Tucker and Snavely [1] for multiplane image generation, the camera parameters were set up to mimic the training data for the model, the youtube videos in RealEstate10K. Upon the camera intrinsics given in RealEstate10K dataset, most clips seemed to use a resolution corresponding to a 16:9 aspect ratio, such as 1920×1080,

Figure 3.2: Examples of images from the synthetic dataset.

1280×720 or 640×360, which is consistent with the standard aspect ratio for Youtube videos. Combining this with the focal lengths in the dataset showed that the typical field of view for the cameras was 90° horizontally and around 59° vertically. The synthetic images were generated with a 512×512 resolution and to make this consistent with the training data, both the horizontal and vertical field of view was set to 59°. Doing this allows the resulting images to be identical to images generated with a 16:9 aspect ratio at a 90° horizontal and 59° vertical field of view that are then cropped to a $1:1$ aspect ratio. This exact resolution was chosen since it resulted in a good balance between high image quality, low computation times for pixel-wise operations and total dataset file size. In addition, setting both the horizontal and vertical resolution to a multiple of 128 allowed the model to be used with the MPI generator without any padding or cropping.

BlenderProc For generation of synthetic images, BlenderProc2 by Denninger et al. [17] was used. BlenderProc2 is a pipeline for the open source computer graphics software Blender [18]. It allows photorealistic rendering of Blender scenes for the sake of training convolutional neural networks. Using Blender and BlenderProc2, images can be rendered from various dataset of synthetic scenes and be exported together with exact pose information and depthmaps.

Dataset The scenes used to generate the synthetic dataset was gathered from the synthetic indoor scene repository 3D-FRONT (3D Furnished Rooms with layOuts and semaNTics) by Fu et al. [19]. The scenes in the 3D-Front repository consist of various fully furnished high-quality texture indoor environments. The dataset was chosen because of its similarities to the RealEstate10K dataset [5] which the single-view MPI generator was trained on. Every scene from the 3D-Front repository used to generate the dataset was individually selected and the camera trajectories for each scene were chosen manually. A few example renderings from the 3D front dataset can be seen in figure 3.2.

The synthetic dataset generated from 3D-Front consists of 17 different camera trajectories selected from different scenes. The camera trajectories follow a straight line perpendicular to the forward axis of each camera. Along each trajectory, images were generated with a strict distance interval of 5 cm resulting in a total of 615 MPI. More detailed information on the dataset as well as description on the views along each

Table 3.1: Information on MPI dataset generated from 3D-Front using BlenderProc

Scene number	Number of MPI	Scene description
1	24	Transition from a bedroom view to a living room view.
2	38	Bedroom view with bedside table and bed.
3	26	View of a reflective texture dinner table and its scenery.
4	31	View of a living room table, its contents and scenery.
5	43	Living room view from a hallway.
6	20	Bedroom view from the edge of the room.
7	36	Living and dining room view from a hallway.
8	37	Living and dining room view from the living room.
9	37	Hallway and living room view from the living room.
10	32	Living room view from the edge of the room.
11	37	Low living room view from the edge of the room.
12	49	View of furniture from a slight distance away.
13	48	Child's bedroom view from the middle of the room.
14	34	Dining/Living room view from the middle of the room.
15	38	Open floor plan apartment view.
16	55	Dining table view from a slight distance away.
17	30	Kitching and dining table view from the hallway.
Total	**615**	

trajectory can be seen in table 3.1.

3.2.2 Real Image Dataset

For real image data, part of the RealEstate10K dataset[5] was used. The reason that this dataset was used, rather than capturing and mapping a new dataset with SLAM, or utilizing any other pre-existing SLAM dataset, is that this was the dataset was used for creating the single-view MPI generator[1]. Because the authors used the dataset both for training and evaluation of the model, we know that the single-view MPI generator is adapted to input from this dataset and that it will be able to generate good output. Using a different dataset will introduce more uncertainty regarding if the model used for generating the MPI is well adapted to the input data or not.

The RealEstate10K dataset does not contain images directly and consists of a collection

Figure 3.3: Examples of images from the real image dataset, RealEstate10K.

of text files, each containing a youtube video url together with a number of mapped camera poses and corresponding timestamps in the video. For each text file and video url, the mapped poses form a single camera trajectory chosen from the video. The camera trajectories varied between different clips resulting in a lot of variation in camera distances between adjacent frames of the video as well as relation between the rotation of the camera and the direction of the trajectory. Likewise, the trajectories in the RealEstate10K dataset did not necessarily follow a straight line, as they do for the synthetic dataset.

Data Extraction and Formatting To extract data for MPI generation, the text files in the dataset were processed and the corresponding frames were downloaded from the youtube videos. As mentioned in 2.3.1, the pipeline created by the authors of the dataset was designed to select sequences of each video favorable for training and evaluating the MPI model. This meant that a lot of blurry and distorted sequences, as well as some clips containing editing effects were sorted out. However, some of the downloaded frames still contained large font descriptive texts, and frames towards the end of the sequence often contained transition editing effects. These frames were therefore manually sorted out from the data. For each processed text file, every 5th frame was downloaded, resulting in between 10 and 25 images per sequence. The reason for this was to mimic the test data that Tucker and Snavely [1] used to test their model, which consisted of images extracted from every 5 and 10 frames in the video sequences. The resulting subset of RealEstate10K that was extracted consisted of 667 MPI over 38 scenes. The exact length of the sequences in each scene can be seen in table 3.2 and a few example images can be seen in figure 3.3

External Depth Extraction Using the camera poses for the images extracted from the RealEstate10K dataset, text data files for camera parameters and image poses were created in a format readable by COLMAP[11]. The created data files were then imported together with the corresponding images into COLMAP through the command-line interface and used to perform a sparse reconstruction of the scene, preserving the imported camera poses for each image. For every image, the sparse reconstruction yielded number of extracted features, many of which had been matched between several images and had a corresponding triangulated 3D position estimation. Information regarding these 3D mapped feature points, as well as the 2D positions in the images in which they were matched was then exported to be processed externally. The external processing finding the distance between the 3D position of the points and the corresponding image plane. This allowed a sparse depth map to be created,

Table 3.2: Information on the subset of RealEstate10K used as the SLAM dataset

Scene number	Number of MPI	Scene number	Number of MPI	Scene number	Number of MPI
1	12	14	21	27	16
2	21	15	17	28	12
3	24	16	22	29	15
4	17	17	22	30	23
5	17	18	16	31	20
6	16	19	23	32	22
7	15	20	17	33	15
8	12	21	14	34	19
9	23	22	21	35	19
10	15	23	21	36	16
11	11	24	12	37	20
12	16	25	23	38	16
13	14	26	12		
Total	**667**				

with a width and height resolution consistent with that of the input images.

3.3 Generation of Multiplane Images

For generation of single-view multiplane images, the method presented by [1] was used. The tensorflow libraries released with the paper to Google's Google Research repository was used together with their model trained on the RealEstate10K dataset. As the model release by Google only is compatible with images that have a width and height dimension that is a multiple of 128, the real-life data images were padded to match this resolution following the procedure used by Tucker and Snavely [1]. Since the synthetic images were generated with a 512×512 resolution, they already fulfilled this requirement and no resizing was needed. After the MPI had been generated, the padded part of the MPI's was cropped out, returning the images to their original resolution. Another limitation of the model is that it is only capable of generating multiplane images consisting of 32 layers, which is the number of layers for all multiplane images used in the experiments presented in this report.

3.4 Multi Plane Image Scaling

Using the external depth information extracted from the scene using tools such as COLMAP or SLAM, or ground truth depth maps in the case of synthetic images, the layer depths of each MPI were readjusted. As shown by Tucker and Snavely [1], an MPI can be used to generate generate both depth and disparity maps, using the over operation in equation (2.5) for non transformed layers and replacing the color value of each pixel color value C_i by the corresponding layer depth value d_i or inverse depth value d_i^{-1}, respectively. Doing so to estimate depth map $\hat{D}$ for an MPI with 32 layers, yields the following expression/

$$\hat{\boldsymbol{D}} = \sum_{i=1}^{32} \left(d_i \alpha_i \prod_{j=i+1}^{32} (1 - \alpha_j) \right) \tag{3.1}$$

The strategy for scaling MPI depth values applied in this work utilizes this equation in conjunction with a externally generated depth map D, used as a ground truth to minimize the error between the estimated depth map and the ground truth, $\|\hat{\boldsymbol{D}} - \boldsymbol{D}\|$

3.4.1 Optimization Algorithm

By analyzing (3.1) we see that each element of the sum yields an expression consisting of the current index depth value multiplied by a element-wise product of matrices. However, since the alpha values α_i of each layers are known, we can simply replace the matrix product in each element of the sum by a constant matrix c_i. Doing so results in the expression being simplified to

$$\hat{\boldsymbol{D}} = \sum_{i=1}^{32} d_i c_i \tag{3.2}$$

Each matrix can now be flattened into its own vector of dimension $W \times H = N$. These vectors can then be further combined into a single matrix $\boldsymbol{A}$ of dimensions $N \times 32$. Now, gathering the depth values d_i into a column vector $\boldsymbol{x}$ allows the equation to be re-written as a compact matrix-vector product,

$$\text{flatten}(\hat{\boldsymbol{D}})^T = \left[\text{flatten}(c_0)^T, \;\; \text{flatten}(c_1)^T, \;\; ..., \;\; \text{flatten}(c_{32})^T\right] \boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}. \tag{3.3}$$

Flattening the ground truth depthmap into a vector $\text{flatten}(\boldsymbol{D})^T = \boldsymbol{b}$ allows the expression $\|\hat{\boldsymbol{D}} - \boldsymbol{D}\|$ to be written as $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|$. Since this is a linear expression, it can be solved for 32 known values of $\hat{\boldsymbol{D}}$ and always has an exact optimal solution. Given any missing values in the depth map, the corresponding pixel values in $\boldsymbol{b}$ and $\boldsymbol{A}$ can be set to 0 or masked to avoid any of these values to contribute to the final solution.

Linear Constraints Optimizing the depth of any MPI using the method discussed above without any constraints might result in MPI depth values taking on

negative values or reordering of the MPI layers. While the optimal solution containing layer depths placed at 0 might be a good indicator that the corresponding layers are redundant, layers placed at negative depths do contribute to the loss while not contributing to any visual change in the rendering of the MPI and should thus be avoided. Reordering of layers might result in a better solution, but due to the scaling algorithm operating on the assumption of a fixed rendering order, reordering of the layers without recreating $\boldsymbol{A}$ with the new rendering order will cause a non-optimal solution to be found. The discussed constraints are therefore included in the optimization problem which then takes on the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\| \\ \text{s.t.} \quad & x_i \geq 0 \\ & x_i \geq x_{i+1}. \end{aligned} \tag{3.4}$$

The strict inequality is not used for the zero constraint to allow redundant layers to be removed from rendering. Likewise, strict inequalities are not used for the ordering constraint either. As long as the rendering order of the layers is preserved, layers placed at the same depth do not cause a problem. It should also be noted that an upper limit constraint for the depths would be excessive. Since all rows of $\boldsymbol{A}$ sum to 1, any layer depth value is naturally constrained to the maximum value of $\boldsymbol{b}$, the highest value of the ground truth depth map.

Sparse Depth Information When optimization using sparse depth information, the overall procedure had very few changes. The sparse depth information, extracted from the data as explained in section 3.2.2, was stored using the exact same representation as the dense depth map using a $W \times H$ matrix where each position of the grid corresponds to a certain pixel. Each sparse depth point value was stored in the most adjacent pixel position on the grid after projecting the point into the image camera. All remaining values in the matrix were set to 0. Running the optimization algorithm with these sparse depth maps caused all zero-values to be masked out, leaving only the sparse depth points to be used. While this method caused the algorithm to rely on much fewer alpha values in the MPI, the lower number of points allowed optimization to be performed much faster.

Depthmap Preprocessing Some values in the dense depth maps generated using blenderproc had values at infinity. In the images, these corresponding pixel values depicted windows and other areas where the background of the environment could be seen. Since these values would disturb the optimization process, all values over a certain threshold in the depthmaps were set to 0.

3.4.2 Linear Solver

For solving the constrained least squares problem, an implementation of the Alternating Direction Method of Multipliers (ADMM), first introduced by Boyd et al. [20] was

utilized. The ADMM optimizes one variable at a time while keeping the others fixed and uses dual variables to enforce constraints. The implementation used proximal operators and a solver was constructed using the python package `proxop` by Chierchia et al. [21]. For more information regarding proximal operators, please refer to the cited user guide.

3.4.3 Layer Merging

Due to the design of the linear optimization problem shown in (3.3) and in particular the ordering constraint, optimization often resulted in adjacent layer depths taking on the same values. Each time an optimal solution lies close enough to the edge of this constraint, the optimal solution within the constrained set will lie right on the edge, placing the layers at the same depth.

Each time this happens, all layers placed at the same depth can be merged into a single image, thus reducing the total layers of the MPI. This was done simply by rendering the new layer the same way the MPI would be rendered, using equation (2.5). For two images $\{C_1, \alpha_1\}$ and $\{C_2, \alpha_2\}$, the resulting new image would become

$$\{C_{1,2}, \quad \alpha_{1,2}\} = \left\{ \frac{C_1\alpha_1(1-\alpha_2) + C_2\alpha_2}{\alpha_2}, \quad \alpha_2 + \alpha_1(1-\alpha_2) \right\}. \tag{3.5}$$

Performing this algorithm iteratively on all layers placed at the same depth in a front-to-back manner allows an arbitrary number of images to be merged. Since this merging step can be seen as a pre-rendering for the layers of the MPI that contribute with the same perspective effects, it will result in no loss of image quality or visual effects in the MPI. Any image rendered from the MPI will be the same regardless if the layers are merged in advanced, or rendered at the same position during run-time.

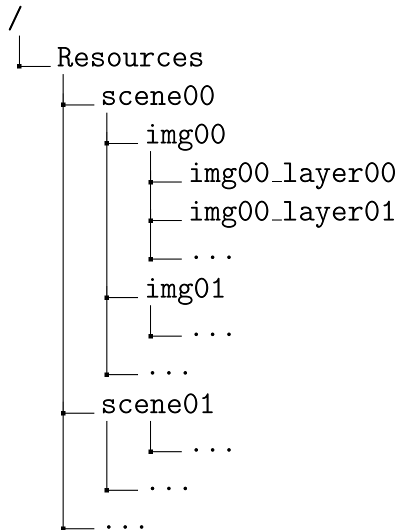
The advantage of layer merging however, is that it potentially allows compression of the MPI file size. A scene with 10 multiplane images each consisting of 32 layers results in a scene representation consisting of $10 \cdot 32 = 320$ images. For large resolution scenes consisting of many MPI, the file size might quickly become an issue and a more compact MPI scene representation is beneficial.

3.5 Viewer Application

In order to allow displaying of the scaled MPI, blending between multiple MPI and exporting rendered images for evaluation, a MPI viewer was constructed using the Unity Engine. The application was developed using version 2021.3.4f1 LTS of Unity and the rendering pipeline used was the built-in RP. In this section, the application, what it is capable of and how it works will be explained in greater detail.

3.5.1 Overview

When the application is run, it loads all MPI images for a certain scene from a specified resources folder on disk. In the folder containing the MPI, each MPI has its own subfolder containing the PNG images for each layer, ordered by their filename. An example of how the file structure needs to be set up can be seen in the directory tree below.



After the images have been loaded from disk, the reference camera poses for all MPI as well as the layer depths are read from three separate text files. In the three text files, each line contains the blank space separated floating point values for the x, y, z camera positions, w, x, y, z camera quaternion rotations or the $d_1, d_2, ...$ camera depth values, corresponding to the MPI given by the line number. It is important that each row contains the correct number of values and the number of lines corresponds to the number of MPI in the dataset.

Interface Using the loaded images and the corresponding positions, rotations and layer depths, each MPI is placed at its reference camera position in the Unity scene. A user interface is loaded consisting of two separate windows as the main camera is initially placed at the reference camera position of the first MPI. One of the windows displays the final image rendered from all the MPI in the scene, while the other window displays a 2D user interface showing each MPI reference camera position as well as the current camera position. An image of the user interface can be seen in figure 3.4.

Navigation Both windows are intractable and by pressing a reference MPI position in the 2D user interface, the camera moves from the current position to the selected position, rendering views by blending between different MPI during the transition. Hovering over the window with the MPI rendering, the camera position is changed locally around the camera reference position of the current MPI. This works the similarly to the interactive example presented on the project pages of the research by Tucker and Snavely [1] and Flynn et al. [8], the difference being that rotational movement is not included. Likewise, hovering over the window and scrolling will move the camera forward or backward, zooming in or out on the scene.

Figure 3.4: Image showing the user interface of the MPI viewer application.

3.5.2 Unity Scene

The scene hierarchy of the application contains four parent Game Objects responsible for all content in the scene, each hosting a number of child Game Objects. The functions of each of these game objects are

Main Camera The main camera which the final blended rendering of all MPI is output to. The camera is rendered into a rendertexture displayed by the leftmost window in the interface shown in 3.4. On runtime, is assigned a number of MPI Camera child game objects corresponding to the total number of multiplane images in the scene.

- *MPI Camera* Camera used to render an image from a single MPI which the camera is assigned to.

Multiplane Image Renderer Game object containing logic for loading MPI layers, depth and spacial information, as well as placing every MPI correctly in the scene. At runtime, generates a MPI game object as a child for each MPI in the scene.

- *MPI* Empty game object placed at the reference camera position of the corresponding MPI. Holds a child game objects for each layer in the MPI placed at its corresponding depth in front of the parent object.

User Interface Empty game object hosting child components responsible for generating the user interface. This includes everything that can be seen in figure 3.4.

Navigation Map Camera Camera responsible for rendering a 2D display of the MPI positions into the the rendertexture displayed in the rightmost window in the user interface in figure 3.4.

3.5.3 Scripts

To manage the images, allow interaction with the user interface and handle camera movement a number of custom scripts and shaders are used. The scripts are all attached to either the **Multiplane Image Renderer** or the **Main Camera** game

object. Since these scripts and shaders make up the main part of the application, an explanation of the features of the most important scripts are given below.

Multiplane Image Renderer The Multiplane Image Renderer game object has one single script attached to it named **MpiLoader** that is responsible for loading images, creating game objects and positioning the multiplane images in the scene. The sole purpose of the script is to set up the scene and its functions are called only once on the start of the application. After the script has been ran it has fulfilled its purpose and detaches itself from the game object. On the start of the application the script

- Parse the text files containing position and quaternion values
- Create all MPI game objects at their reference camera position using the parsed values
- Read all layer depths values by parsing the corresponding text file
- Loop through all MPI game objects and create layers at the corresponding depth positions using the parsed values.

Main Camera The main camera of the scene has three scripts attached related to rendering, scene navigation and image capturing. The names and functions of these scripts are as follows.

MpiBlend On starting the application creates child camera game objects for rendering each MPI. In addition, contains code that is ran each frame for rendering images from each created camera and blending them to a final output image.

MpiSelector Contains code for managing the camera rendering the image seen in the right window of the interface in figure 3.4. After all MPI have been created, calculate the optimal way to display every MPI on the grid and attach icons to every MPI, making them visible to the Navigation Map Camera. In addition, contains code for detecting when the user has clicked a new MPI camera reference position in the UI as well as code for moving the main camera to said new camera reference position.

CaptureEvalImgs Contains code for automated image capturing and exporting. Its functions can be ran through the editor in order to capture and save images for evaluation of all MPI in the current scene.

Additional Scripts The following are smaller yet essential scripts attached to various game objects

NavigationControls Attached to the UI element window on the right hand side of the interface shown in figure 3.4. Contains code that allows the user to zoom in and out on the grid as well as click and drag to change the viewing position.

MpiLocalMove Attached to the UI element window on the left hand side of the interface shown in figure 3.4. Locally changes the main camera position when the user hovers the mouse over the window. Zooms the camera in and out when scrolling the mouse wheel over the window.

WindowDrag Attached to both UI element windows shown in figure 3.4. Allows each window to be dragged to change its position on the UI.

3.5.4 Shaders

The shaders in the application consist of one regular shader for rendering the image planes of each MPI as well as one compute shader for calculating the blending of multiple MPI. The function of each shader is as follows.

MPI Shader The MPI Shader is applied to every layer of each MPI in the scene. Since the Unity rendering pipeline takes care of alpha compositing for rendering, this shader does not do anything other than output the pixel value of the texture applied to the object. The reason a custom shader is written for the MPI is to ensure that redundant rendering effects used by the standard shader are not applied.

Blend Compute Shader The blend compute shader contains code ran om the GPU using several threads for blending between multiple images. As input, it takes images rendered from a fixed number of multiplane images with a camera reference position close to the current camera position together with corresponding blending weights. It outputs a single image, retrieved by blending the input images using the input weights. Finally, it normalizes the alpha value for each pixel in the output image so that the final image has no transparent pixels. For input multiplane images $\{C_i, \alpha_i\}$ and w_i, this corresponds to

$$\begin{aligned} C_{out} &= \frac{\sum^N C_i \alpha_i w_i}{\sum^N \alpha_i} \\ \alpha_{out} &= \frac{\sum^N \alpha_i w_i}{\sum^N \alpha_i} = 1. \end{aligned} \tag{3.6}$$

3.5.5 MPI Blending

The algorithm for blending between images follows the procedure of Mildenhall et al. [7]. As mentioned in section 2.3.3 their method constructs interpolated views by combining separately rendered images from multiple MPI positioned close to the current camera position. These rendered images are the input images mentioned in the description of the compute shader responsible for handling the blending, mentioned in chapter 3.5.4.

Blending between multiple renderings of MPI, contrary to simply relying on the over

operation to render the final image correctly given a scene of multiple MPI is favorable for two reasons. First, each MPI is created with alpha values adapted for rendering images using only that MPI, containing its specific number of layers. This likely results in deeper layers not being used at all during rendering, since they are occluded by a large number of shallowly placed layers. Similarly, while the current camera position might be very far from the reference camera position of a certain MPI, its layers might still lie very close to the camera and cause occlusion of other MPI that would be favorable to use for rendering.

3.6 Evaluation

The standard way of evaluating view synthesis for a single multiplane image is to render an image of a nearby camera position for which we already have a ground truth image. Doing this allows the image rendered from the MPI to be compared directly to the ground truth using evaluation metrics such as those introduced in section 2.4. This method is utilized by most previous research using multiplane images including Zhou et al. [5], Tucker and Snavely [1] and Flynn et al. [8]. This is also the method for evaluation applied in this research.

3.6.1 Blending

The evaluation of the scaled MPI is used to find an answer to RQ1 and RQ2 in section 1.1. To paraphrase, evaluation should be useful for deciding whether individual layers of an MPI can be scaled to combat the scaling ambiguity and allow smooth transitions between different MPI in a scene environment consisting of multiple MPI. For this purpose, evaluation can be done using either images rendered from MPI individually, or blended images of renderings from multiple MPI. For the sake of simplicity, the former option was chosen. Using renderings from multiple blended MPI would introduce additional ambiguity regarding the impact of the blending algorithm. In addition, the blending algorithm does not need to be evaluated as Mildenhall et al. [7] has already proven its effectiveness through comparisons of renderings between single and multiple MPI. Scale correction will directly translate to smooth transitions between MPI during blending and there is therefore no need to evaluate blended images directly.

3.6.2 View Synthesis

Unlike data generated by capturing real scenes and generating poses with SLAM, generation of synthetic images allows strict control of camera parameters, positioning and ground truth. The synthetic dataset described in section 3.2.1, contained images generated from various scenes along trajectories with a strict 5 cm distance interval. To allow for more variation during testing, each MPI was evaluated with the ground truth at two separate distances. In addition to only evaluating each MPI with its closest neighbours, views were also synthesised at a 10 cm baseline from the reference camera position of each MPI. This compares synthesizing new views for each MPI both

at the reference camera positions of its closest neighbours and at camera positions two images apart.

The evaluation images for the RealEstate10K dataset were also synthesized in a similar manner. As discussed in 3.2.2, the images used to generate the MPI in this dataset were spaced 5 frames apart in each clip. This allowed the same method to be used as for synthetic images, generating images for evaluation generated from MPI spaced 5 frames and 10 frames apart. This follows the evaluation procedure utilized by Tucker and Snavely [1] when evaluating the single-view MPI generator model.

3.6.3 Procedure

The *CaptureEvalImgs* script described in 3.5.4 was ran for every scene in the dataset, generating and exporting ground truth images and renderings from nearby MPI for all reference camera positions in the scene. The image estimated image at each camera location was then evaluated against its corresponding ground truth, given by the original image at the camera position for which the estimated image was rendered. The metrics used for evaluation were PSNR, LPIPS and SSIM as described in section 2.4.

Cropping As discussed in section 2.2.3, rendering an MPI from a position other than its reference camera position corresponds to transforming the image with a homography. Since this homography is projective in nature, the resulting image will be warped and not strictly rectangular. When capturing the evaluation images, the rendering from the transformed planes don't fit perfectly into the strictly rectangular image, potentially leaving gap at the edges of the image, as can be seen in figure 3.5. For some evaluation metrics, such as LPIPS which is based on pixel-by-pixel MSE, this will introduce a large error. Since the layer depth values determines the extent of the warping, this will cause certain layer depths to be favored regardless of image quality. To combat this problem, we follow the procedure of evaluation in the original single-view MPI paper [1], cropping 5 % of the image at each side, guaranteeing that the resulting image does not have any empty pixel values.

Comparisons To evaluate the performance of the depth scaling algorithm, the depths values were compared against inversely spaced depths, placed at equidistant values in disparity between a far plane and a near plane. The values chosen for the near and the far plane was 1 m and 100 m. These two values were chosen as they during trials seemed to give good view transitions between MPI compared to other values and were also the standard distances used in the implementation of the original MPI paper[5]. In order to also include comparisons against some other type of scaled depths, the depths were also additionally compared against inverse depths where the position of the depth and far plane were customized for each MPI, effectively scaling the whole MPI up and down. The strategy used to generate these were to place the near place and the far plane at the minimum and maximum value from the depth information (sparse or dense depth map) for the corresponding image.

Figure 3.5: Image showing an example of plane warping during rendering.

Chapter 4

Results

Results are presented first as quantitative comparisons between the three different depths values described in section 3.6.3. In the second part of the chapter, qualitative comparisons are made in order to provide more insight into how the results vary depending on data-dependent factors. In the final part, results are presented on layer reduction of the main method. In all plots and tables, the depth values using the method presented in this thesis are refereed to as "Optimized depths" while the depth values used for comparisons, consisting of depths inversely spaced between 1 and 100 as well as the minimum and maximum value in depth information are refereed to as "Inv fixed" and "Inv custom" respectively.

4.1 Similarity Metric Scores

The PSNR, SSIM and LPIPS evaluations are presented for the images rendered from the corresponding multiplane images as explained in 3.6.3, using the three different sets of depth values. More similar images have higher PSNR and SSIM scores as well as lower LPIPS loss.

4.1.1 Overall Comparison

Each of the MPI renderings were evaluated against the neighbouring ground truth image in the trajectory with an identical camera pose. The similarity metric scores of all individual image pairs over all scenes were then averaged, giving one similarity score for each evaluation metric and each dataset (synthetic and slam). The scores are presented below.

Synthetic Data Table 4.1 below contain the summary of the results of the evaluation on the synthetic dataset. On average, the optimized depths outperformed the fixed inverse depths and the customized inverse depths over all evaluation metrics, regardless of baseline. Unsurprisingly, doubling the baseline lowered the scores for all metrics regardless of which depth values were used. On average, the unscaled depth values inversely spaced between 1 and 100 perform better than the depth values utilizing the depth information.

MPI Depths	Baseline	PSNR	SSIM	LPIPS
Optimized	5 cm	**28.9**	**0.919**	**0.085**
	10 cm	**25.9**	**0.880**	**0.115**
Inv Fixed	5 cm	26.1	0.877	0.103
	10 cm	23.4	0.833	0.164
Inv Custom	5 cm	27.1	0.890	0.095
	10 cm	24.2	0.848	0.144

Table 4.1: Table displaying average evaluation metric scores for the synthetic data
.

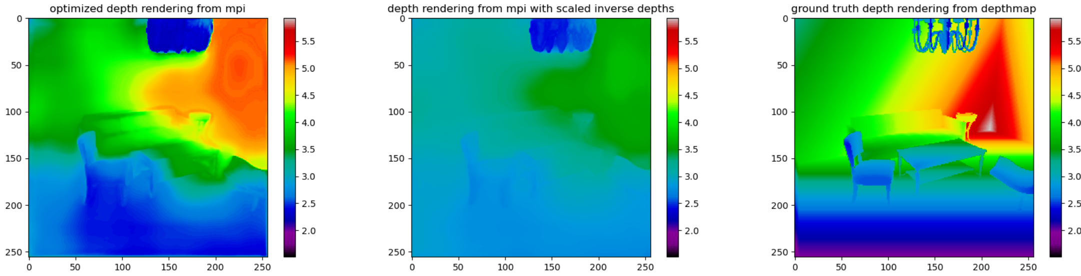
SLAM Data Table 4.2 below contain the summary of the results of the evaluation on the SLAM dataset. Similarly to the results of the tests on the synthetic dataset, the images rendered using the optimized depths have better scores over all evaluation metrics. Also consistent with the synthetic dataset, images rendered using inverse depths with far and near planes at 1 and 100 gave higher similarity scores than those using external depth information for the position of the near and the far plane.

MPI Depths	Baseline	PSNR	SSIM	LPIPS
Optimized	5 frames	**23.7**	**0.763**	**0.133**
	10 frames	**21.1**	**0.687**	**0.188**
Inv Fixed	5 frames	22.6	0.727	0.162
	10 frames	20.0	0.649	0.237
Inv Custom	5 frames	20.6	0.664	0.209
	10 frames	18.2	0.585	0.293

Table 4.2: Table displaying evaluation metric scores for the SLAM data
.

4.2 Qualitative Examples

In the following section, a number renderings are presented that can be used as representative examples. Figures depicting renderings presented in this section can be found in section 4.3.



(a) Optimized depth rendering. **(b)** Inverse depth rendering. **(c)** Ground truth.

Figure 4.1: Example of a well optimized rendering with corresponding ground truth from the synthetic dataset.

4.2.1 Depth Renderings

Using the method to generate depth from an MPI from which the depth scaling algorithm was derived as explained in section 3.4, the layers of an MPI can be used together with the corresponding layer depths in order to render depths and disparity maps. For an MPI together with a set of depth values, the depth map can is estimated using the over operation in equation (3.1). These depth renderings provide a visualization of the layers of an MPI and their corresponding depths. Comparing these estimated depth maps to the ground truth allows comparisons of the depth detail of the MPI.

Fixed Inverse Depths To be able to compare the depth renderings side-by-side, the plots of the estimated depth maps were set to use the same color scale as the ground truth depth map. This worked well for the optimized depths and the customized inverse depths. However, not being created using the ground truth or external depth information, the fixed inverse depth values, all ranging between 1 and 100, most often fell outside of the color range of the ground truth resulting in a single-colored depth map. Because of this, the depth renderings from the fixed inverse depth values were excluded from the results.

Synthetic Data For the depth renderings from the synthetic data, the ground truth depth map is the same as the dense depth map which was generated and exported using BlenderProc as described in 3.2.1. This is also the same depth information which was used for scaling the optimized depths. An example of depth map renderings from an MPI using optimized and inverse depths together with the ground truth can be seen in figure 4.1. More renderings used as representative examples in the discussion can be found in section 4.3.

SLAM Data For the SLAM dataset, there is no ground truth dense depth map and the depth renderings from each MPI using different depths can therefore not be compared side-by-side to the ground truth as for the synthetic dataset. However, the sparse depth information can still be used to set the color scale of the depth renderings

for the optimized and customized inverse depths. Plotting these two depth renderings together with the base image and corresponding point matches used to scale the MPI allows for visualization of the MPI, the two set of layer depths and the points used for optimization. An example of an image with overlaid point matches can be seen in figure 4.2. Additional depth renderings and point match visualizations from the SLAM dataset referred to in the discussion can be found in section 4.3.

Figure 4.2: Example image from the SLAM dataset with corresponding sparse depth points. Image number 8 in scene 32.

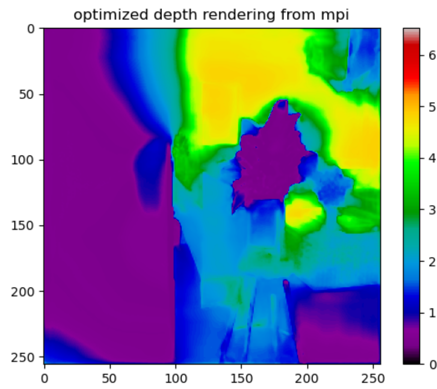
4.2.2 Image Renderings

Rendered images from several MPI are included in the results of this report as examples. These rendered images are the same images used for evaluation with the image similarity metrics. Studying representative examples, especially along with corresponding depth renderings and depth information visualizations can allow insight into the performance of the depth scaling. All depth rendering examples referred to in the discussion can be found in section 4.3.

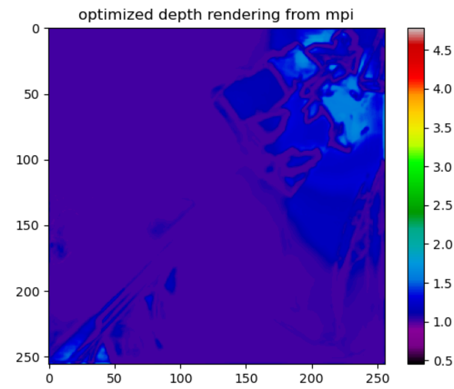
4.3 Image and Depth Renderings

In this section, a number of figures containing depth and image renderings are included. The renderings were selected as representative examples and are further commented in the discussion. The renderings are grouped by the type (depth or image rendering) and by the dataset (synthetic or SLAM).

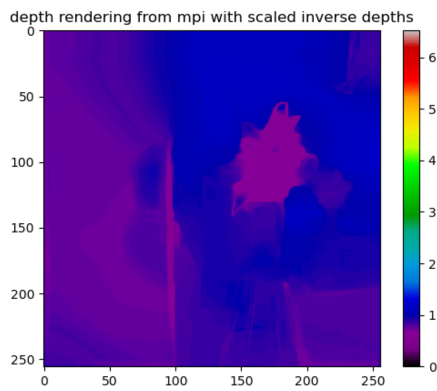
4.3.1 Synthetic Data Depth Renderings



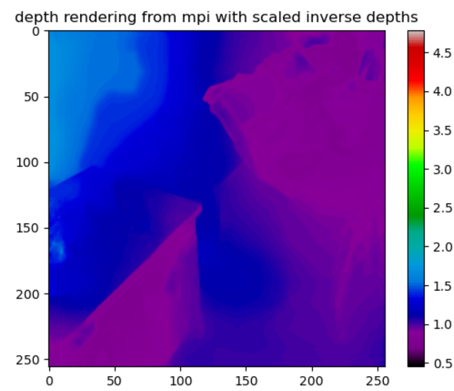
(a) Optimized depths depth rendering.



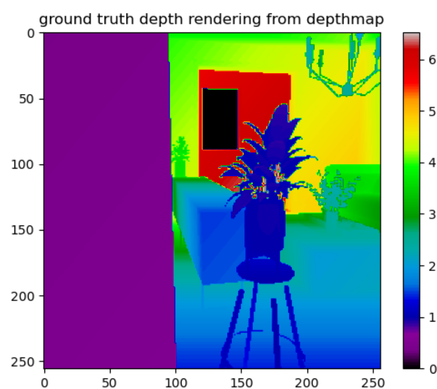
(d) Optimized depths depth rendering.



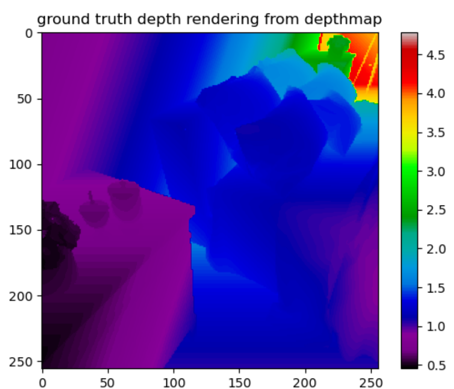
(b) Inverse depths depth rendering.



(e) Inverse depths depth rendering.

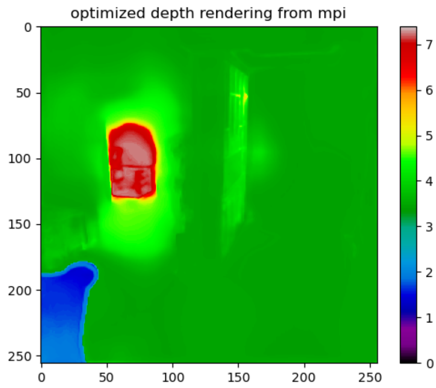


(c) Ground truth depth map.

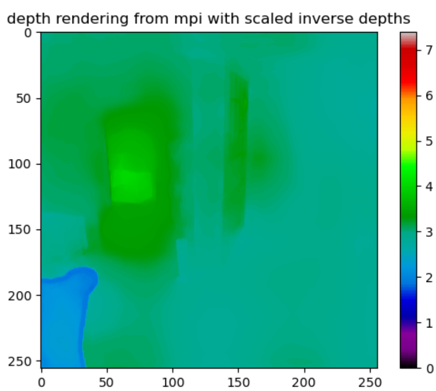


(f) Ground truth depth map.

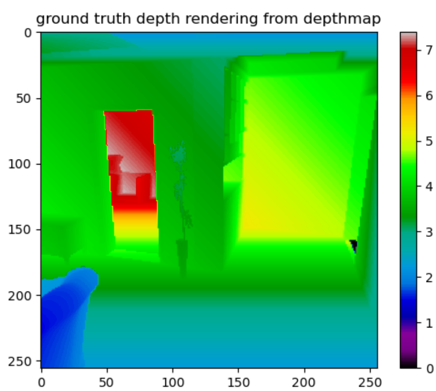
Figure 4.3: Ground truth depth map and depth renderings. The left images (a-c) show an example of how a poorly generated MPI results in bad results even after optimization. The right images (d-f) show an example of how a contrasted depth area (top right corner) can disturb the optimization algorithm.



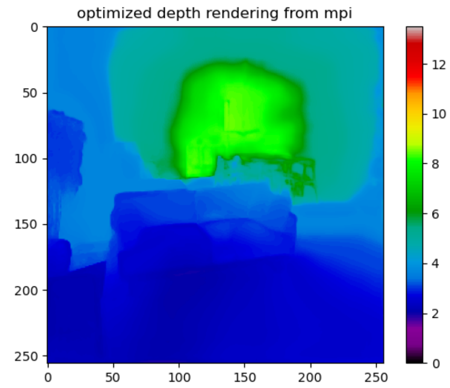
(a) Optimized depths depth rendering.



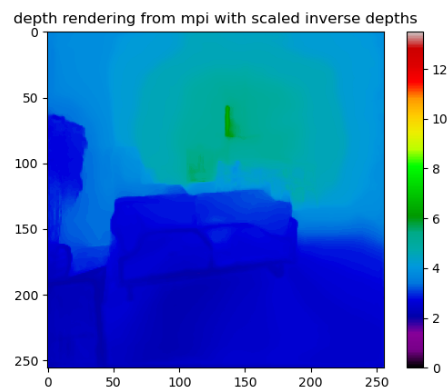
(b) Inverse depths depth rendering.



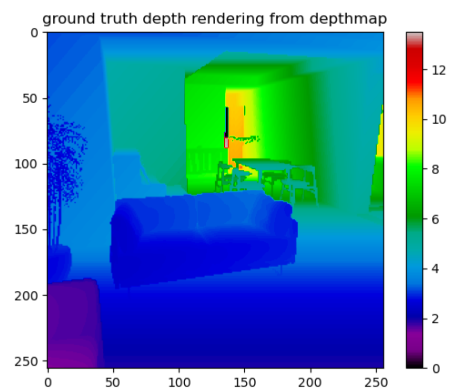
(c) Ground truth depth map.



(d) Optimized depths depth rendering.



(e) Inverse depths depth rendering.



(f) Ground truth depth map.

Figure 4.4: Ground truth depth map and depth renderings for two images in synthetic scene 9 and 8. The depth optimized depth renderings are examples of non-major adjustments being made by the scaling algorithm to improve results for a well generated MPI.

4.3.2 Synthetic Data Image Renderings

(a) Ground truth to the MPI rendering. Image 12 in synthetic scene 1.

(b) Rendering of image 12 using MPI number 10 in synthetic scene 1.

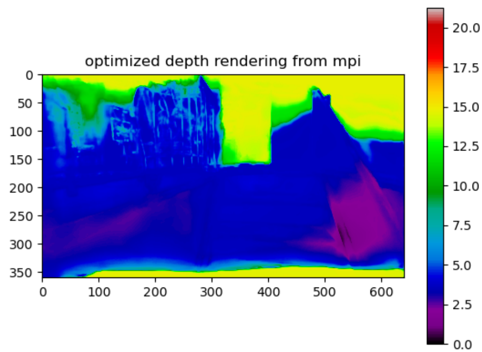
Figure 4.5: Image rendered using the same MPI as in figure 4.3 (a-c) with corresponding ground truth. Example of how poorly generated MPI results in bad rendering results after optimization.

(a) Rendering of image 13 using optimized depths.

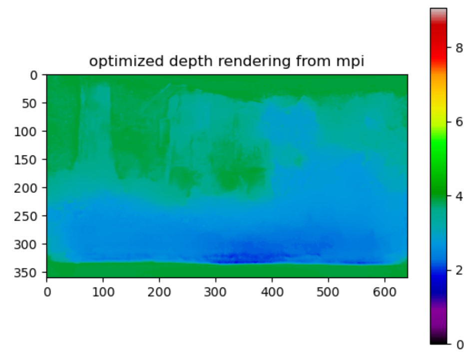
(b) Rendering of image 13 using scaled inverse depths.

Figure 4.6: Comparisons of the same image rendered using MPI number 1 in scene 13 with optimized and inverse depths. The left image shows an example of artifacts that can sometimes appear after optimization.

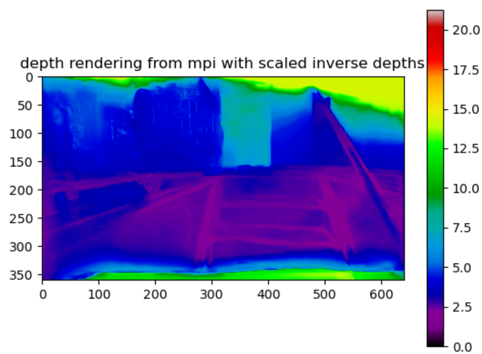
4.3.3 SLAM Data Depth Renderings



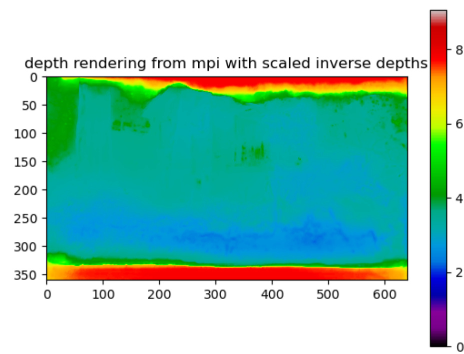
(a) Optimized depths depth rendering.



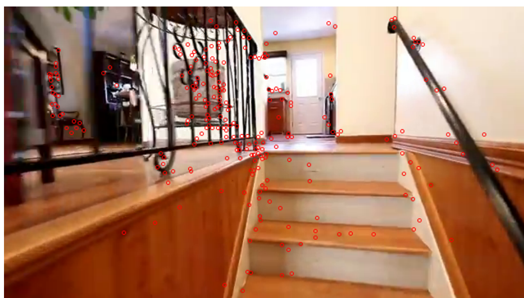
(d) Optimized depths depth rendering.



(b) Inverse depths depth rendering.



(e) Inverse depths depth rendering.



(c) Sparse depth points.

(f) Sparse depth points.

Figure 4.7: Comparisons of depth renderings using the optimized depths for two SLAM dataset scenes. The left images (a-c) show how certain well textured areas end up having more mapped points and become prioritized during optimization. The right images (d-f), show a poorly generated MPI with a great number of evenly distributed sparse depth points.

4.3.4 SLAM Data Image Renderings

(a) Ground truth to the MPI rendering. Image 11 in SLAM scene 20.

(b) Rendering of image 11 using MPI number 13 in SLAM scene 20 using optimized depths.

Figure 4.8: Image rendering using optimized depths and corresponding ground truth for the MPI shown in figure 4.7 to the right (d-f). Shows an example of how the rather poorly generated MPI leads to bad rendering results despite a great number of well dsitributed sparse depth points.

(a) Rendering of image 12 using optimized depths.

(b) Rendering of image 12 using scaled inverse depths.

Figure 4.9: Image rendering using optimized and inverse depths for the MPI shown in figure 4.7 to the left (a-c). Example of how the an sparse depth points unevenly distributed across the image lead variance in rendering quality across the image.

4.4 Layer Reduction

In table 4.3 below, a summary of the layer reduction results is shown. The full results showing the reduction of layers in each individual scene for both the synthetic and SLAM data can be found in appendix A.

Dataset		Total layers	Layer mean	Std	Reduction	Std
Synthetic	Reduced	5245	8.5	2.2	73.3 %	6.8 %
	Unreduced	19648	32	0	0%	0%
SLAM	Reduced	4300	6.4	2.6	78.7 %	8.8 %
	Unreduced	21504	32	0	0 %	0 %

Table 4.3: Table displaying summarized layer reduction results for both datasets.
.

Overall, more layer depths were placed at the same values for the SLAM data than for the synthetic data and the variance was also greater. The maximum average size reduction for a single scene in the SLAM data was **91.7%** compared to **80.1%** in the synthetic data. The minimum average reduction in a scene however was found in the synthetic dataset at **63.9%** compared to the minimum of **65.6%** in the SLAM data. In the SLAM dataset **3.9%** of all MPI were reduced to 2 layers or less while the same number in the synthetic dataset was 0.16%. The same numbers for 3 layers or less were **12.4%** and **1.0%** respectively. The minimum amount of layers in any MPI after reduction in the synthetic dataset was **15** and in the SLAM dataset, **14**. The overall distribution of the layers reduced can be seen in figure 4.10.

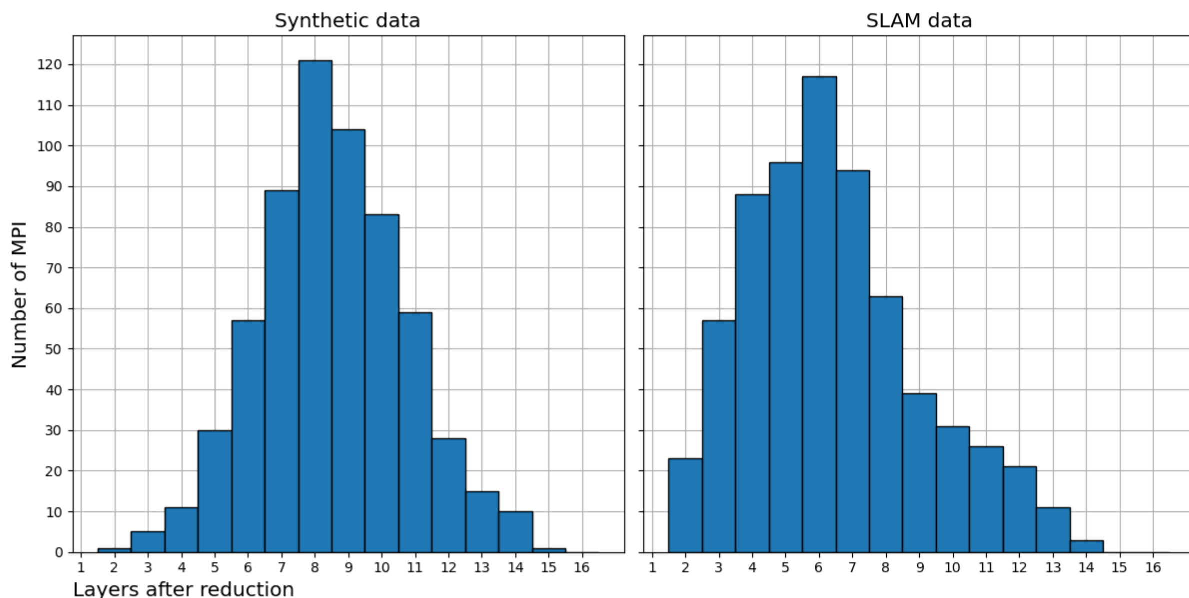


Figure 4.10: Histogram of the distribution of layer reduction in each dataset.

Chapter 5

Discussion

5.1 Quantitative Overview

As presented in the results section, evaluations on the renderings from multiplane images using the three set of depth values resulted in higher PSNR, SSIM and LPIPS scores overall for the optimized depths. What this shows is that scaling MPI layers individually using the algorithm presented in this work is a viable way of solving the scaling problem and improving smooth view transitions between multiple MPI. However, to determine how well the algorithm performs and to what extent it can be used in various applications, further analysis is needed.

For the SLAM dataset, the fixed inverse depths performed better than the custom inverse depths over all metrics and for the Synthetic data, the custom inverse depths only performed slightly better. This inconclusive result make it hard to determine whether placing the near and far planes for inversely scaled depths between the minimum and maximum value in the depth information actually reduces scaling ambiguity compared to placing the planes at suitable fixed values. This would leave the scaling algorithm which uses external depth information to be compared against a method without any access to any external depth data.

To give further insight, the results from the evaluation on the RealEstate10K SLAM dataset can be compared to the ablation study using the same evaluation metrics in the paper of the single-view MPI generator used to generate the all MPI in this research [1]. Since the authors of this paper used a scaling factor σ calculated from a sparse point cloud to scale the whole MPI up and down as mentioned in section 2.3.2, this could work as an alternative comparison to the custom inverse depths. The optimized depths in this work evaluate at lower scores over all metrics compared to their full model for the 5 frame and 10 frame baseline. However, it should be noted that their **noscale** model, which is the same as the fixed inverse depths in this work, also

scored higher than the inverse depth method evaluated for this work. The difference could be explained by the variation in the data used for testing. Since only a subset of the RealEstate10K dataset was used for testing in this thesis, it is likely that the selected scenes either had larger baselines or and/or more difficult camera trajectories on average. To draw further conclusions, additional testing is therefore required.

5.2 Representative Examples

The goal of the individual layer scaling algorithm is to adjust the position of the layers to reduce scaling error, increase rendering accuracy and reduce view transition errors between multiple MPI. As the singe-view MPI generator is trained to create MPI with inversely spaced layer depths and because inversely spacing the depths maximized the disparity-space resolution as explained in section 2.2.2, the layers of the MPI should theoretically only need minor adjustment in most cases. We can see two examples where this is the case by comparing the ground truth to the MPI depth renderings of synthetic scene 8 and 9 seen in figure 4.4. However, by looking at a few examples, we can see how that is not always the case.

Nonoptimal Layer Placements One of the main issues seem to be when MPI layers containing a lot of detail (low transparency) are placed at depths drastically different from the depths they were created to model. As an example, looking at the difference between the ground truth depth in figure 4.3c and the depth rendering in figure 4.3b, we can see that not only is there great total variance in depth, but there is also variance in depth over areas that should have the same depth when comparing with the ground truth. For this case, the MPI can be said to be incompatible with the ground truth, meaning that the MPI has layers that have low transparency for pixel values that have a high depth contrast in the ground truth depth map. This results in layers being placed at depths where they try to model multiple depth spaces of the image. For the extreme case where a layer has low transparency for very deep and very shallow depth values, the optimization will place the layer in the middle, where it is unable to model either of the depth spaces accurately. While this is not always visible from viewing the depth renderings, it is likely the cause of renderings from certain MPI with optimized depths having artifacts not visible in their non-optimized counterparts, as seen in figure 4.6.

Optimization Constraints Although the optimization constraints drives many layers to be placed at the same depths, it can also become a limitation for certain cases. Since the constraints enforce layer ordering and each layer depth value works as the upper and lower limit for the layers in front and behind, one layer placed at a nonoptimal depth can cause other layers to also be forced to nonoptimal depth values. This might be the reason that although the depths are optimized, some MPI does not seem to be able to render images with good results.

Sparse Depth Data A problem with using sparse depth maps created from feature extraction and mapping is that the distribution of points in the images might be uneven and areas with a lot of texture might be favored. This causes the resulting sparse depth map to have more depth values in certain areas causing these areas to be favored during optimization. This can be seen in figure 4.7 for the images on the left hand side. In figure 4.7c, we see that the left part of the image depicting the railing has a higher density of depth points compared to the other parts of the image. This in turn causes the depth rendering from the optimized depths seen in figure 4.7a to be very detailed around the same area, while the depth in other parts of the image are less detailed. The steps of the staircase, as an example, loses a lot of detail compared to the depth rendering using the inverse depths seen in figure 4.7b.

5.3 Single View MPI Generator

The main limiting factor of achieving good results with the layer optimization algorithm seems to be the quality of the multiplane images themselves. An example of this is synthetic scene 1 as seen in figure 4.5a, which contained camera trajectory between the entrance to two different rooms. In between the two rooms, there was a large wall relatively close to the camera which seemed to interfere with accurately creating the MPI. This resulted in the the MPI visualized with inverse depths in figure 4.3b. Even though the layer values are optimized, as seen in figure 4.3a, since the underlying MPI was not created accurately, the images rendered from the MPI are not accurate, as seen in figure 4.5b.

Network Scene Incompatibility Another example where the generated MPI was not created well, seemingly because single-view MPI generator does not work well with the scene, is scene 20 from the SLAM dataset. The image shown in figure 4.8 depicts a house surrounded by a forest-like environment. Since the MPI generator was trained on real estate footage consisting mainly of lower texture indoor environments, it was not able to generate an MPI accurately for this scene. The high-textured environment allowed a relatively large number of depth points to be distributed fairly equally across the whole image as seen in 4.7f. This in turn allowed for the layers to be adjusted seemingly well as can be seen by examining the difference between figure 4.7e and 4.7d, but the rendering using the optimized depths shown in 4.8b still ends up with artifacts in large parts of the image.

MPI Detail Another incompatibility between the single-view MPI generator and the scaling algorithm is the lack of detail in the generated MPI. Since the single-view MPI generator has infer depth from scenes without any intrinsic information, it lacks detail and depth variations over smaller areas of the images seem to be ignored. If some of these small areas of the image are in large contrast to the surrounding area, they can still have a great impact on the placement of the layers. For example, in synthetic scene 2, comparing the ground truth depth rendering seen in figure 4.3f to the MPI depth rendering using inverse depths in figure 4.3e, we see that the depth

detail in the top right corner of the ground truth is missed by the MPI. Since these depth values are in great contrast to the rest of the image and no layer of the MPI was created specifically depict these pixels at the correct depth, the optimization algorithm tries to model the depth for these pixels by using other layers that were created for more shallow depths as discussed under *Nonoptimal layer placements* in 5.2.

5.4 View Transitioning and Blending

Blending between multiple MPI resulted in seamless view transitions for [7], something that was not oberved to the same degree with the method for this thesis, although the same blending method was utilized. The reason for this might not only lie in the greater baseline between the MPI but also in how the network was trained. When blending, accumulated alpha values are considered from each MPI rendering. Our method might not draw full benefit of this compared to the method presented in LLFF [7], due to how their network was trained throughout the blending procedure. This way of training the network throughout blending encourages it to give low alpha values to pixels that it is unsure about, so that they can be filled in by pixels from other MPI during the blending procedure. Since our network was trained for single-views, each MPI will instead have pixels that accumulate to non-transparency over all layers, which is not optimal for blending.

5.5 Algorithm Run Time

Another point to discuss is the time difference required for running the optimization algorithm. The run time of the algorithm varied significantly depending on the size of the matrix $\boldsymbol{A}$ in (3.4), which grows as the dimensions of the image increases. $\boldsymbol{A}$ also grows larger the more depth points are used, which resulted in optimization for the synthetic dataset taking considerably longer than for the SLAM dataset. When running the algorithm, parameters of the linear solver needed to be adjusted manually in order to ensure that an optimal solution was found and quality did not vary significantly between various MPI. To avoid this problem all together, a fixed number of iterations was used, which was high enough to guarantee an optimal solution was reached for all MPI used. This makes it impossible to give accurate optimization times for the algorithm. However, hyper parameter engineering and its automation for linear optimization is out of the scope of this thesis and is therefore left as future work.

5.6 Layer Reduction

In table 4.3, we see that the algorithm managed to reduce the size of the synthetic and SLAM data by an average of over 70 %. While more layers were merged for the SLAM dataset than for the synthetic data, a considerable amount of MPI ended up with 2 or 3 layers after optimization which was not the case for the synthetic data.

When the number of layers are too low, even though their placement is optimized, the low disparity resolution will limit the accuracy of the images rendered from the MPI. Looking at figure 4.10, we see that the distribution of layer numbers in the optimized MPI was on average higher and much more evenly spaced around the mean value for the synthetic data. This can be explained by the synthetic using the actual ground truth depth for optimization instead of a limited sparse point set, but the actual input images themselves might have also had an impact. Although the layer numbers are low for certain MPI, it is important to note that the layer merging itself has no impact on quality once two layers are already placed at the same depth, and the evaluation results in section 4.1 still holds true whether layers merged or used as separate images.

Chapter 6

Conclusion

The algorithm developed in this research works as an effective method to individually scale layers of an multiplane images generated from single images, increasing rendering accuracy compared to fixed inversely spaced layers for the three of the most common image similarity evaluation metrics. The performance of the scaling algorithm and the rendering results are highly dependent on the quality of the multiplane images themselves which varies from scene to scene. However, the single-shot MPI generator has the benefit of being a light-weight and simple way to generate multiplane images and the MPI created from it can therefore not be expected to have the same quality as other algorithms utilizing a much great amount of input images to maximize quality, such as DeepView[8]. Further, the algorithm allows for reduction of redundant layers, by placing adjacent layers to equal depths which allows layer merging, reducing the size of the MPI representation by over 70 %.

6.1 Research Questions

RQ1 Can layers of single-image generated multiplane images be scaled as a way to tackle the scale ambiguity problem?

As shown by the results presented in, in section 4.1.1, re-scaling the layers of single image generated MPI improves image similarity to the ground truth over all similarity metrics used. This strongly shows improved render quality and in turn a reduction of scaling-ambiguity induced errors. In conclusion, individual layer scaling can be scaled as a way to tackle the scaling ambiguity problem with good results.

Answer

RQ2 Are scaled single-image generated multiplane images a viable option for multi-

image input view synthesis with MPI blending?

While scaling layers of single-image generated MPI reduces scaling ambiguity and improves results, the results are inconclusive regarding the possibility of using the re-scaled MPI for multi-image input with blending. Many images in the results showed visual artifacts and the results seemed to be heavily based on the quality of the underlying MPI. If the quality of the MPI is low, the quality of the rendered images will be low regardless of re-scaling. As of now, single-image generated multiplane images does not seem to be a equally efficient method to previously presented methods utilizing many images to render MPI such as Mildenhall et al. [7] and Flynn et al. [8].

RQ3 If the estimated depths of two or more layers are close, can they be merged to reduce redundancy and compress the file size of the MPI representation without major loss of quality?

The presented method resulted in an average size reduction of over 70 % for both datasets. Since layer merging is done for layers placed at the same position after optimization, it does not result in any loss of quality. Based on the results of the experiments conducted in this research, layer merging seems to be a viable way to size-reduce MPI with good results.

6.2 Future Work

To continue on this research, more alternative optimization algorithms could be explored. One method that could be tested is allowing layer reordering. This would require the $\boldsymbol{A}$ matrix to be recreated for each alternative layer order combination as explained in 3.4.1 and would likely lead to a more optimized depth rendering, but a lower rate of layer merging. A way of layer optimization could also be explored in which the layers are optimized to maximize RGB consistency between rendered images and ground truth, instead of optimizing the depth rendering as was done in this work. Finally, for further and more accurate evaluation of the work, comparisons could be included with the same depth scaling factor σ that was used in the original single-view MPI generator[1].

Bibliography

[1] Richard Tucker and Noah Snavely. "Single-view View Synthesis with Multiplane Images". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[2] Marc Levoy and Pat Hanrahan. "Light Field Rendering". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 1996, 31–42. ISBN: 0897917464. DOI: `10.1145/237170.237199`. URL: `https://doi.org/10.1145/237170.237199`.

[3] Shenchang Eric Chen and Lance Williams. "View Interpolation for Image Synthesis". In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA: Association for Computing Machinery, 1993, 279–288. ISBN: 0897916018. DOI: `10.1145/166117.166153`. URL: `https://doi.org/10.1145/166117.166153`.

[4] Richard Szeliski. *Computer Vision - Algorithms and Applications, Second Edition*. Texts in Computer Science. Springer, 2022.

[5] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe and Noah Snavely. "Stereo Magnification: Learning view synthesis using multiplane images". In: *ACM Trans, SIGGRAPH*. 2018.

[6] Pratul Srinivasan, Richard Tucker, Jonathan Barron, Ravi Ramamoorthi, Ren Ng and Noah Snavely. "Pushing the Boundaries of View Extrapolation With Multiplane Images". In: June 2019, pp. 175–184. DOI: `10.1109/CVPR.2019.00026`.

[7] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng and Abhishek Kar. "Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines". In: *ACM Transactions on Graphics (TOG)* (2019).

[8] J. Flynn, M. Broxton, P. Debevec, M. DuVall, G. Fyffe, R. Overbeck, N. Snavely and R. Tucker. "DeepView: View Synthesis With Learned Gradient Descent". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, 2019, pp. 2362–2371.

DOI: 10.1109/CVPR.2019.00247. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00247.

[9] Raul Mur-Artal, J. Montiel and Juan Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system". In: *IEEE Transactions on Robotics* 31 (Oct. 2015), pp. 1147 –1163. DOI: 10.1109/TRO.2015.2463671.

[10] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy and Thomas Brox. "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation". In: June 2016, pp. 4040–4048. DOI: 10.1109/CVPR.2016.438.

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

[12] In.

[13] Pawan Sinha and Richard Russell. "Perceptually-based Comparison of Image Similarity Metrics". In: *Perception* 40 (Nov. 2011), pp. 1269–81. DOI: 10.1068/p7063.

[14] Zhou Wang, Alan Bovik and Hamid Sheikh. "Structural Similarity Based Image Quality Assessment". In: *Digital Video Image Quality and Perceptual Coding, Ser. Series in Signal Processing and Communications* (Nov. 2005). DOI: 10.1201/9781420027822.ch7.

[15] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman and Oliver Wang. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". In: *CVPR*. 2018.

[16] Unity Technologies. *Unity Engine*. https://unity.com/. 2022.

[17] Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi and Harinandan Katam. "BlenderProc". In: *arXiv preprint arXiv:1911.01911* (2019).

[18] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: http://www.blender.org.

[19] Huan Fu, Bowen Cai, Lin Gao, Ling-Xiao Zhang, Jiaming Wang, Cao Li, Qixun Zeng, Chengyue Sun, Rongfei Jia, Binqiang Zhao et al. "3d-front: 3d furnished rooms with layouts and semantics". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10933–10942.

[20] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato and Jonathan Eckstein. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends in Machine Learning* 3 (Jan. 2011), pp. 1–122. DOI: 10.1561/2200000016.

[21] Giovanni Chierchia, Emilie Chouzenoux, Patrick L Combettes and Jean-Christophe Pesquet. *The proximity operator repository. user's guide, 2020.*

Appendix A

Layer Reduction Results

Table A.1: Table displaying layer reduction results of the synthetic dataset.

Scene number	Total layers before reduction	Total layers	Layer number mean	Layer number standard deviation	Size reduction mean	Size reduction standard deviation
1	768	153	6.4	2.6	80.1 %	8.1 %
2	1216	298	7.8	2.9	75.5 %	9.2 %
3	832	184	7.1	1.7	77.9 %	5.3 %
4	992	306	9.9	2.4	69.2 %	7.4 %
5	1376	344	8.0	1.5	75 %	4.8 %
6	640	183	9.2	1.6	71.4 %	4.9 %
7	1152	321	8.9	1.4	72.1 %	4.3 %
8	1184	379	10.2	1.4	68.0 %	4.4 %
9	1184	308	8.3	1.7	74.0 %	5.2 %
10	1024	244	7.6	1.6	76.2 %	4.9 %
11	1184	392	10.6	1.4	66.9 %	4.4 %
12	1568	421	8.6	1.7	73.2 %	5.4 %
13	1536	401	8.4	1.5	73.9 %	4.5 %
14	1088	256	7.5	1.6	76.5 %	4.9 %
15	1216	439	11.6	1.8	63.9 %	5.5 %
16	1760	378	6.9	1.5	78.5 %	4.6 %
17	928	238	8.2	1.2	74.4 %	3.8%
All scenes	**19648**	**5245**	**8.5**	**2.2**	**73.3 %**	**6.8 %**

Table A.2: Table displaying layer reduction results of the SLAM dataset for scenes 1 to 19.

.

Scene number	Total layers before reduction	Total layers	Layer number mean	Layer number standard deviation	Size reduction mean	Size reduction standard deviation
1	384	32	2.7	1.1	91.7 %	3.4 %
2	672	147	7.0	1.2	78.1 %	3.8 %
3	768	110	4.6	1.9	85.5 %	6.1 %
4	544	116	6.8	1.4	78.7 %	4.5 %
5	544	104	6.1	1.2	80.9 %	3.6 %
6	512	78	4.9	1.5	84.8 %	4.8 %
7	480	96	6.4	2.4	80.0 %	7.5 %
8	384	55	4.6	2.9	85.7 %	9.1 %
9	736	216	9.4	1.3	70.7 %	4.2 %
10	480	84	5.6	1.3	82.5 %	4.1 %
11	352	75	6.8	1.9	78.7 %	5.9 %
12	512	120	7.5	1.9	76.6 %	5.8 %
13	448	129	9.2	1.5	71.2 %	4.6 %
14	672	86	4.1	1.7	87.2 %	5.2 %
15	544	97	5.7	1.8	82.2 %	5.7 %
16	704	129	5.9	1.5	81.7 %	4.8 %
17	704	179	8.1	2.5	74.6 %	7.9 %
18	512	46	2.9	0.8	91.0 %	2.5 %
19	736	96	4.2	1.6	87.0 %	5.0 %

...

Table A.3: Table displaying layer reduction results of the SLAM dataset for scenes 20 to 38.

Scene number	Total layers before reduction	Total layers	Layer number mean	Layer number standard deviation	Size reduction mean	Size reduction standard deviation
20	544	105	6.2	1.2	80.7 %	3.7 %
21	448	105	7.5	1.0	76.6 %	3.2 %
22	672	132	6.3	0.8	80.4 %	2.6 %
23	672	110	5.2	1.6	83.6 %	4.9 %
24	384	39	3.3	0.5	89.8 %	1.4 %
25	736	125	5.4	2.3	83.0 %	7.0 %
26	384	66	5.5	1.2	82.8 %	3.6 %
27	512	135	8.4	2.4	73.6 %	7.7 %
28	480	63	4.2	1.1	86.9 %	3.6 %
29	736	121	5.3	0.9	83.6 %	2.7 %
30	640	139	7.0	1.3	78.3 %	4.1 %
31	544	74	4.4	1.1	86.4 %	3.3 %
32	704	215	9.8	3.0	69.5 %	9.2 %
33	480	153	10.2	1.7	68.1 %	5.3 %
34	608	128	6.7	2.4	78.9 %	7.6 %
35	608	127	6.7	1.9	79.1 %	5.9 %
36	512	176	11.0	2.2	65.6 %	6.8 %
37	640	116	5.8	1.0	81.9 %	3.1 %
38	512	176	11.0	1.3	65.6 %	4.0 %
All scenes	**21504**	**4300**	**6.4**	**2.6**	**78.7 %**	**8.8 %**

Master's Theses in Mathematical Sciences 2023:E19
ISSN 1404-6342

LUTFMA-3500-2023

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/