Real-time Building of Multiplane Images from RGBD Data

Floris van Onna[1]

Supervisor: Petr Kellnhofer[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2026

Name of the student: Floris van Onna
Final project course: CSE3000 Research Project
Thesis committee: Petr Kellnhofer, Joana Pinho Gonçalves

Abstract

Multiplane Images (MPIs) are among the simpler forms of scene representations usable for novel view synthesis. They are stacks of partially transparent RGB images which create parallax when they are not aligned with the viewpoint. Considering MPIs are a method of adding depth to images, one might expect that they would have been extensively studied in conjunction with images obtained from depth cameras (RGBD images), but the contrary is true. We propose a method for generating MPIs from RGBD images using only classical algorithms, as opposed to machine learning based methods. This makes it fast to run and hence suitable for realtime use, such as building MPIs from live video streams from depth cameras. Our method is evaluated in terms of performance using scenes adapted from the VKITTI 2 dataset. It is shown that on a modern laptop, our method far exceeds the speed of generation needed to ingest real-time data, and the process could still run at interactive framerates when generating MPIs with more than 400 planes. Since visual artifacts are very visible in the built MPIs, it is suggested that more sophisticated algorihms are investigated, with the goal of alleviating these artifacts while remaining real-time.

1 Introduction

The ability to capture and digitally reproduce and visualize scenes is the pinnacle of many computer graphics related studies. Letting anyone see a location or object interactively from their own computer has ample applications, for real-estate websites or online shops for example. Considering the complexity of real-life scenes and the (lack of) computational power available to consumers, such systems are often very complex and have to make compromises to maintain interactivity. Among the simpler of such representations is the Multiplane Image (MPI) [ZTF+18].

Multiplane Images represent 3D scenes using a set of 2D images, stacked to create the illusion of depth. If a sufficiently large number of such planes are used, the camera can be moved slightly, revealing parallax not present in the original images. Because the geometry used to represent the scene is quite simple, this method is well suited for low-power mobile devices and websites.

One useful source of input data for MPIs are depth cameras. They provide extra information about the scene, allowing a wider range of algorithms to be used for analysis and reconstruction. RGBD images do have one drawback when compared to synthetically generated alternatives, however. The combination of measurement inaccuracies and occlusion from either viewpoint of a stereoscopic depth camera leads to some pixels not having depth values. This implies that to be usable with such input data, the algorithms have to either be agnostic of this, or the missing data has to be reconstructed.

The purpose of this research is to evaluate whether MPIs can be build in real-time from RGBD data, and whether said MPIs also have acceptable quality. More precicely, the following questions will be answered:

- How can MPIs be built from RGBD data using only classical algorithms?
- Can MPIs be built with an effective speed of 30 viewpoints per second?
- Does an increase in building speed result in lower quality of the produced MPI?

To this end, we present a method using only classical algorithms which can be used to build MPIs from sets of RGBD images. The method has been tuned for speed of MPI generation and rendering. This makes it suitable for realtime building and viewing of MPIs from a live video stream from a depth camera. Since most depth cameras operate with a maximum framerate of 30 Hz, this is the performance target for our algorithm.

All code associated with this paper can be found at `https://gitlab.ewi.tudelft.nl/cse3000/2025-2026-q4/multiplane-images/floris`.

2 Background & history of MPIs

The general concept of MPIs came long before the introduction of the now standard term. First, in 1994, Wang and Adelson [WA94] proposed a method to separate individual frames from videos into layers, which could then be moved around separately, creating new views of the original scene. This method was oriented more around motion rather than depth however, and does not provide a way to simply reproject said planes based on a new viewpoint and rotation. Then, in 1998, Shade et al. [SGHS98] introduced Layered Depth Images (LDIs). LDIs do allow for novel view synthesis of captured scenes, but also relies on a more complex rendering method, as the depth samples were not necessarily planar. Such a method has the advantage that viewpoints far away from where the scene was captured can be rendered without larger artifacts compared to nearby viewpoints. This does come with the drawback that the data necessary to build an LDI must either be generated synthetically, or must be preprocessed into a voxelized representation first using another method.

2.1 Machine learning-based methods

In the last decade, the increase in available computing power has resulted in a shift of MPIs being generated using machine learning, rather than just classical hand-tuned algorithms. This started with Zhou et al. [ZTF+18], who also coined the term Multiplane Image, and was later improved upon by Mildenhall et al. [MSO+19] and then Srinivasan et al. [STB+19].

Machine learning-based methods reframes the problem of MPI generation from one of geometry to one of finding an efficient model architecture. They have

the advantage of being able to infer certain information, such as depth when only color information is used, or occluded areas when few, or just one, viewpoint is used. This comes with the disadvantage that the accuracy of MPIs built using these methods is bound not only by the accuracy of the scene data, but also the training data, and the quantity thereof. Since machine learning often requires large amounts of computational power, this also meanse these methods are often restricted to situations where said power is available, or compromises have to be made with the accuracy of the generated MPI or the time needed for the generation.

All techniques mentioned up until this point relied on multiple input images to cover angles occluded by some viewpoints and disoccluded by others. Then, in 2020, Tucker and Snavely [TS20] proposed a method suitable for generating MPIs from just a single input image. This was later improved upon further by Luvizon et al. [LCdS+21] through the introduction of the Adaptive Multiplane Image. Adaptive MPIs have non-uniformly spaced image planes, which make them more suitable in situations where little computational power is available, because less planes are needed to accurately represent a scene.

2.2 MPI-adjacent structures

The most recent improvement in MPIs was made by Zhang et al. [ZWL+23], who introduced the Structural MPI, where each image plane can be oriented freely. Compared to regular MPIs, S-MPIs have increased representation accuracy while reducing the number of planes needed for scenes with a lot of flat geometry, like indoor scenes. While this is the first application of such a representation to the field of MPIs, it had been proposed before for use in other fields in 2012. Guan et al. [GYTL12] and Holz et al. [HHRB12] each published methods for reconstructing scenes from RGBD images using a representation practically identical to S-MPIs. Though both of these methods were presented with a different (computer vision-related) purpose, they could be applicable to novel view synthesis as well.

2.3 RGBD Datasets

For the evaluation of our method, an RGBD dataset is required. Many such datasets exist, as can be seen in a compilation by Lopes [Lop22], however not all are suitable for the same purposes. Some include extra data that others do not, like intrinsic and extrinsic camera matrices, which are required for our method. VKITTI 2 [CMH20; GWCV16] was chosen for this project, as it contains RGB data, depth data, the aforementioned camera matrices, and other data not used here. Since it is a synthetic dataset, it contains perfect depth values for every pixel, simplifying the algorithms needed to process the data.

3 Method

Our method consists of a multi step process, and converts a set of input viewpoints and a few parameters, which are further explained below, into an MPI. At a high level, our method can be split into three stages:

1. **Initialization**
   The MPI is initialized as an array of two dimensional textures, with four channels per pixel. The values are initialized by taking all pixels from the main viewpoint, rounding their depth down to the nearest plane, and writing the color to that plane. All other pixels remain fully transparent.
2. **Adding viewpoints**
   The remaining viewpoints, called auxiliary viewpoints hereafter, are added to the MPI by having all pixels on every plane of the MPI projected onto the viewpoint. Then, the color value is added to the plane if its depth matches the depth sampled from the viewpoint when rounded down.
3. **Finalization**
   All pixel values of all planes of the MPI are averaged by dividing the RGB value by the alpha value, if it is not zero. Then, to account for interpolated sampling when rendering, all color values of transparent pixels are set to the average of their neighbors, while keeping them transparent.

3.1 Parameters

As mentioned before, besides the input viewpoints, a few parameters are necessary.

- **Main viewpoint**
  One of the input viewpoints has to be selected as the main viewpoint, which means the resulting MPI will be built with the planes aligning with this viewpoint's forward axis. Because this viewpoint is projected without distortion onto the image planes, it will likely have the highest representation accuracy.
- **Plane count**
  The number of planes dictates the quality of the MPI, as more having more planes means more distinct depth levels in the final representation. The consequence of this is that building time and rendering frametime will be longer, and file size and memory consumption will be larger, due to more pixels needing to be calculated and stored.
- **Depth cutoff threshold**
  To determine the depth range of the main viewpoint, a threshold is needed to exclude parts of the image that represent volumetric effects, such as clouds or the sky. These parts are usually given (near) infinite depth, and so including these values would reduce the number of planes used to represent nearby geometry, thereby reduceing the overall quality of the built MPI.

3.2 Initialization

To initialize the MPI, the main viewpoint is projected onto the planes by rounding the viewpoint's depth down to the nearest plane. For this to be possible, each plane needs to be assigned a fixed depth. To this end, the

depth range is calculated from the main viewpoint's depth texture as follows:

Let d denote a single depth value and let D_{main} denote the set of all depth values of the main viewpoint. Then, the start of the depth range is given by

$$d_{start} = \min_{x \in D_{main}} d$$

and the end of the depth range is given by

$$d_{end} = \max_{x \in D_{main}, d < p} d$$

where the p is the depth cutoff threshold.

Within the depth range, planes are distributed uniformly with respect to disparity, so the depth of a plane with index i can be calculated as

$$d_i = \left(i \frac{d_{end}^{-1} - d_{start}^{-1}}{n_{planes} - 1} + d_{start}^{-1} \right)^{-1}$$

where n_{planes} denotes the number of planes in the MPI and where i satisfies $0 \leq i < n_{planes}$.

Each plane has the same image resolution, equal to that of the main viewpoint. For each pixel in the main viewpoint, the depth value is rounded down to the nearest plane, and the color value is written to that corresponding plane with an alpha value of one, creating a 4-dimensional vector. This vector is first multiplied with a weight, equal to the number of auxiliary viewpoints. Since all auxiliary viewpoints are later added with a weight of 1, this means the main viewpoint is weighed for fifty percent of the total. This is to prevent the data added from the auxiliary viewpoints from reducing the representation accuracy of the MPI when viewed from the main viewpoint.

3.3 Adding viewpoints

Data from auxiliary viewpoints is added to fill in pixels of the MPI that are occluded from the main viewpoint. This is done by projecting every pixel of every plane of the MPI onto the auxiliary viewpoint, sampling the depth texture, and if the depth matches that of the plane when rounded down, adding the color value to the pixel with an alpha value of 1.

Let P denote the position of any pixel in the MPI's texture array, ranging from $\vec{0}$ (inclusive) to $\vec{n}$ (exclusive), where $\vec{n}_x$, $\vec{n}_y$ and $\vec{n}_z$ denote the width of the MPI in pixels, the height of the MPI in pixels and the number of planes, respectively. Let $M_{int,main}$, $M_{ext,main}$, $M_{int,aux}$ and $M_{ext,aux}$ denote the internal and external camera matrices of the main and auxiliary viewpoints, respectively.

$$P_{main} = M_{int,main}^{-1} \begin{bmatrix} P_{xy} \\ 1 \end{bmatrix} d_{P_z}$$

$$P_{aux} = \left(M_{ext,aux} M_{ext,main}^{-1} \begin{bmatrix} P_{main} \\ 1 \end{bmatrix} \right)_{xyz}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M_{int,aux} \frac{P_{aux}}{P_{aux,z}}$$

$$d_{sampled} = Sample\left(D_{aux}, \begin{bmatrix} u \\ v \end{bmatrix} \right)$$

$$\begin{bmatrix} r & g & b \end{bmatrix}_{sampled} = Sample\left(RGB_{aux}, \begin{bmatrix} u \\ v \end{bmatrix} \right)$$

$$d_{aux} = \left(M_{ext,main} M_{ext,aux}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right)_z$$

$$d_{aux,P_z} = P_{aux,z}$$

$$d_{aux,P_z+1} = P_{aux,z} \frac{d_{P_z+1} - d_{aux}}{P_{main,z} - d_{aux}}$$

Here, the sampled RGB value should only be added to the pixel at P if $d_{aux,P_z} \leq d_{sampled} < d_{aux,P_z} + 1$ is satisfied.

3.4 Finalization

At this point, the MPI is complete, but not yet suitable for rendering. The RGB and alpha values still have an arbitrary range, while most graphics applications expect them to be between 0 and 1. Since every color added from both the main and auxiliary viewpoints has had RGB values within this range and an alpha value of 1, optionally having the whole RGBA vector multiplied by a weight, the alpha value of every pixel in the MPI can be treated as the high end of the range for that pixel. This means the pixels can be normalized by dividing their RGB values by their alpha values, provided the alpha values are not 0. Since an alpha value of 0 prior to this step implies the pixel has not been written to, pixels with an alpha value of zero must remain as such.

$$\begin{bmatrix} r & g & b & \alpha \end{bmatrix} = \begin{cases} \begin{bmatrix} \frac{r}{\alpha} & \frac{g}{\alpha} & \frac{b}{\alpha} & 1 \end{bmatrix} & \text{if } \alpha \geq 0.5 \\ \vec{0} & \text{if } \alpha < 0.5 \end{cases}$$

Here, 0.5 is used as the alpha cutoff, but since the alpha value is always an integer in practice, any value between 0 and 1 will work.

After this step, all pixels in the MPI have RGBA values between zero and one, yet one artifact would still be present if the MPI were to be rendered as is. Assuming bilinear interpolation, any texture sample between transparent and opaque pixels would be blended between the color of the opaque pixels and the implicit black of the transparent pixels. This would work correctly when rendered with alpha blending, assuming premultiplied alpha is used, but for when rendered with alpha cutoff. To fix this, a final pass is applied where the color values of transparent pixels are set to the average of its opaque neighbors, and their alpha values are set to half the average of their neighbors.

Let u and v denote the pixel coordinates of any pixel in any plane of the MPI. The new color value of that pixel is then given by

$$[r \quad g \quad b \quad \alpha]_{old} = Sample\left(RGBA_{plane}, \begin{bmatrix} u \\ v \end{bmatrix}\right)$$

$$[r \quad g \quad b \quad \alpha]_{up} = Sample\left(RGBA_{plane}, \begin{bmatrix} u \\ v-1 \end{bmatrix}\right)$$

$$[r \quad g \quad b \quad \alpha]_{down} = Sample\left(RGBA_{plane}, \begin{bmatrix} u \\ v+1 \end{bmatrix}\right)$$

$$[r \quad g \quad b \quad \alpha]_{left} = Sample\left(RGBA_{plane}, \begin{bmatrix} u-1 \\ v \end{bmatrix}\right)$$

$$[r \quad g \quad b \quad \alpha]_{right} = Sample\left(RGBA_{plane}, \begin{bmatrix} u+1 \\ v \end{bmatrix}\right)$$

$$\alpha_{total} = \alpha_{up} + \alpha_{down} + \alpha_{left} + \alpha_{right}$$

$$\alpha_{new} = \begin{cases} \alpha_{old} & \text{if } \alpha_{old} \geq 0.5 \\ \dfrac{\alpha_{total}}{8} & \text{else} \end{cases}$$

$$x_{new} = \begin{cases} x_{old} & \text{if } \alpha_{old} \geq 0.5 \\ 0 & \text{else if } \alpha_{total} < 0.5 \\ \dfrac{x_{up} + x_{down} + x_{left} + x_{right}}{\alpha_{total}} & \text{else} \end{cases}$$

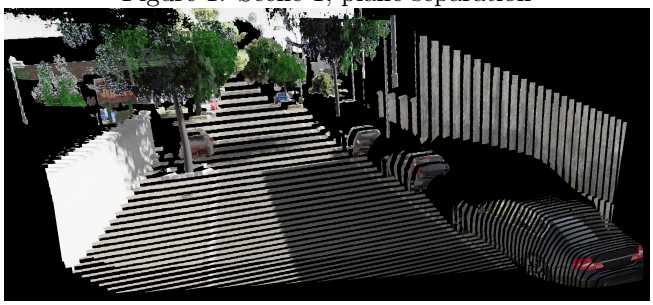
where x can be substituted for either r, g or b. Note that once again, 0.5 is used as the alpha cutoff, but any value between 0 and 1 should work.

3.5 Implementation & Rendering

We implemented our method in Rust and WGSL using `wgpu`, an implementation of the WebGPU specification. For the MPI building, the majority of the initialization stage and adding viewpoints stage were implemented in a compute shader each, and the finalization stage was implemented in two compute shaders.

With the MPI built, the rendering is comparatively simple. A mesh is built with n_{planes} quads made from two triangles, each with the aspect ratio and field of view of the main viewpoint. The planes are ordered front to back, and their depth is calculated as explained in section 3.2. The mesh can then be rendered using a standard perspective view-projection matrix, bilinear interpolated texture sampling, and an alpha cutoff of 0.5. Here, the value of the alpha cutoff does matter, since the alpha values are no longer guaranteed to be integers. Alpha blending should explicitly not be used, as the method produces alpha values of either exactly 0 or 1 almost everywhere in the MPI, and alpha values between 0 and 1 are only meant to to make the edges between planes less jagged.

Figure 1: Scene 1, plane separation



An MPI with 64 planes built from the first 4 viewpoints of scene 1, rendered from a perspective where the planes can be seen separately.

Table 1: Total MPI generation time

Number of planes	32	64	128
2 viewpoints	132.3 ms	237.3 ms	402.0 ms
4 viewpoints	152.5 ms	257.4 ms	429.1 ms
6 viewpoints	160.7 ms	267.2 ms	458.0 ms
8 viewpoints	171.8 ms	279.6 ms	483.7 ms
10 viewpoints	186.2 ms	306.2 ms	511.2 ms

The time needed to build an MPI with the given numbers of planes and viewpoints. Values are averages of 5 runs across all 5 scenes, for a total of 25 measurements.

4 Evaluation

4.1 Performance

We evaluated our method using a subset of the VKITTI 2 dataset. 5 sequences of consecutive frames from the dataset's scenes and their variants were chosen to represent the dataset as a whole. VKITTI 2 contains images of traffic conditions, and since MPIs only model static geometry, sequences were chosen that do not show movement in the scene, but only in the camera. Sequences with a length between 1 and 5 frames were used, with two cameras, resulting in scenes with 2 to 10 viewpoints. For all measurements, the viewpoint of the first frame from the left camera was chosen as the main viewpoint. The depth values in the VKITTI 2 dataset range from 0 to 655.35 meters, and therefore 655.35 was chosen as the depth cutoff (non-inclusive). Finally, each scene was built with three different numbers of planes: 32, 64 and 128. An example can be seen in figure 1.

The numbers shown in table 1 and table 2 are from tests ran on a modern laptop with an AMD Radeon™ RX 7700S. For information on the hardware used, as well as all measurement outcomes, see appendix A.

Table 2: Time per auxiliary viewpoint added

Number of planes	32	64	128
Time per viewpoint	8.8 ms	10.8 ms	14.4 ms

The time needed to add an auxiliary viewpoint (see section 3.3) to the MPI with the given number of planes. Values are averages of a total of 625 measurements taken across the same runs referenced in table 1.

As can be seen in table 2, the time per added viewpoint remains far below the upper bound of 33.3 ms for real-time use. It should be noted that this time per viewpoint does not include the main viewpoint, as the measurements were only taken during the second stage of the MPI building process. Because there is only one main viewpoint however, the time needed to include it is not relevant for the target framerate.

Since the time needed to build an MPI using our method depends linearly on both the number of planes and the number of viewpoints, an approximation for the time needed to build an MPI on the hardware used for testing can be calculated from the values in table 1 and is given by

$$T_{build} = 32ms + 2.8ms \cdot n_{planes} + (7.0ms + 0.058ms \cdot n_{planes}) \cdot n_{viewpoints} \quad (1)$$

From this, we can calculate that on this hardware our method could build MPIs with upwards of 400 planes, while still ingesting viewpoints at an interactive rate. This suggests that even significantly weaker hardware would still be able to build MPIs in real-time, given a more reasonable number of planes.

Another interesting observation is that with a low number of viewpoints, the constant factor, which arises from loading and saving data, setting up the graphics device, allocating memory, etc., far outweighs the time needed to add viewpoints. This points towards more sophisticated methods still being able to be run in real-time, given they are well optimized and run on sufficiently powerful hardware.

4.2 Visual quality

Though the method succeeded in producing MPIs at real-time, the quality of said MPIs is far from perfect. In figure 2,the main viewpoint of scene 1 can be seen as included in the dataset, and as rendered from an MPI built using our method. In several places, parts of the scene appear shadowed by a copy of itself. This happens mostly at depth discontinuities, so these artifacts probably occur because the affected objects are projected onto different parts of the planes from different viewpoints due to rounding errors. On flat, textured surfaces, another kind of artifact can be observed on the boundaries between planes. These artifacts are likely also caused by misalignment between viewpoints due to rounding.

5 Responsible Research

5.1 Reproducibility

To comply with imposed ethical requirements, the use of data within the project, as well as the reproducibility of the results, were critically evaluated. Since exclusively publicly available datasets with open source licenses were used, anyone is able to download, inspect and use the data that was used to evaluate our method. The source code implementation of our method is publicly available under a weak copyleft license, and can be obtained from our GitLab repository[1]. The `README.md` file contains instructions on how to set up the project and how to reproduce our experiments using our codebase. Results obtained from this implementation were included directly in this paper without further processing, ensuring that everything can be reproduced and verified independently. Finally, the likelihood of direct negative societal impact occurring as a result of this research was deemed extremely unlikely, and therefore no further precautions were taken.

5.2 Use of generative AI

For the implementation of our method generative AI tools, specifically Github Copilot and Le Chat, were used. No such generated code was inserted into the codebase as-is, however. All AI generated content was thoroughly checked, and, more often than not, fixed. Furthermore, no AI tools were used for research or for writing this paper. Technically, as of writing this paper Google Search has become an AI-related tool, but only the classical non-AI part was actually used. Therefore, AI tools were only used for the implementation part of this project.

6 Conclusion

Though many methods for building MPIs have been introduced by prior works, little emphasis has been put on the speed of MPI generation. Most modern methods rely on machine learning, which is usually slow, but they can infer information not explicitly present in input data. We have presented a method that can build MPIs in real-time, and is therefore suitable for use in interactive applications. Our method uses RGBD images as input, which can be obtained from depth cameras, depth inference techniques or generated synthetically. On a high-end consumer-grade laptop, our implementation was able to ingest viewpoints at a significantly higher rate than 30 per second, and therefore even significantly weaker hardware should be able to build MPIs at an interactive framerate.

6.1 Possible improvements

In its current state, our method was able to generate MPIs at speeds far beyond what is necessary for real-time use. Given some quite noticable visual artifacts are present in the built MPIs, an investigation into more complex algorithms may yield qualitative improvements in the output of the method. For example: since the weigth of all auxiliary viewpoints are equal, MPIs built with more viewpoints that are less aligned with the main viewpoint may suffer more visual artifacts due to errors arising from the rounding down of the depth to the nearest plane. Better results may be able to be obtained by weighing viewpoints according to their alignment with the main viewpoint, or by spreading out the writes across multiple planes.

[1] `https://gitlab.ewi.tudelft.nl/cse3000/2025-2026-q4/multiplane-images/floris`

Figure 2: Scene 1, main viewpoint

Top: The main viewpoint of scene 1, as included in VKITTI 2 [CMH20; GWCV16]. (Path: `vkitti_2.0.3_rgb.tar/Scene01/clone/frames/rgb/Camera_0/rgb_00000.jpg`)
Bottom: An MPI with 64 planes built from the first 4 viewpoints of scene 1, rendered from the main viewpoint, and brightened. Artifacts can be seen on the side of the car on the right, the streetlight on the far left, and the leaves of trees.

For the sake of simplicity, our method assumes all pixels to have depth values, but as mentioned earlier, this property does not hold for data obtained from depth cameras. To use such data, either the missing depth values would have to be inferred, or our method would have to be adapted to be agnostic of the missing data. A possible approach could be to simply discard any writes to the MPI if a pixel with a missing depth value was read, and then to use a set of viewpoints such that all relevant parts of the scene have depth values in at least one viewpoint.

References

[CMH20] Y. Cabon, N. Murray, and M. Humenberger, *Virtual kitti 2*, 2020. arXiv: 2001.10773 [cs.CV].

[GWCV16] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, "Virtual worlds as proxy for multi-object tracking analysis," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2016, pp. 4340–4349.

[GYTL12] L. Guan, T. Yu, P. Tu, and S.-N. Lim, "Simultaneous image segmentation and 3d plane fitting for rgb-d sensors — an iterative framework," in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2012, pp. 49–56. DOI: 10.1109/CVPRW.2012.6238914

[HHRB12] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke, "Real-time plane segmentation using rgb-d cameras," in *RoboCup 2011: Robot Soccer World Cup XV*, T. Röfer, N. M. Mayer, J. Savage, and U. Saranlı, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 306–317, ISBN: 978-3-642-32060-6.

[LCdS+21] D. C. Luvizon et al., "Adaptive multiplane image generation from a single internet picture," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Jan. 2021, pp. 2556–2565.

[Lop22] A. Lopes, *Awesome rgb-d datasets*, 2022. Accessed: Jun. 18, 2026. [Online]. Available: https://github.com/alelopes/awesome-rgbd-datasets

[MSO+19] B. Mildenhall et al., *Local light field fusion: Practical view synthesis with prescriptive sampling guidelines*, 2019. arXiv: 1905.00889 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1905.00889

[SGHS98] J. Shade, S. Gortler, L.-w. He, and R. Szeliski, "Layered depth images," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '98, New York, NY, USA: Association for Computing Machinery, 1998, pp. 231–242, ISBN: 0897919998. DOI: 10.1145/280814.280882

[STB+19] P. P. Srinivasan, R. Tucker, J. T. Barron, R. Ramamoorthi, R. Ng, and N. Snavely, "Pushing the boundaries of view extrapolation with multiplane images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[TS20] R. Tucker and N. Snavely, *Single-view view synthesis with multiplane images*, 2020. arXiv: 2004.11364 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2004.11364

[WA94] J. Wang and E. Adelson, "Representing moving images with layers," *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 625–638, 1994. DOI: 10.1109/83.334981

[ZTF+18] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely, *Stereo magnification: Learning view synthesis using multiplane images*, 2018. arXiv: 1805.09817 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1805.09817

[ZWL+23] M. Zhang, J. Wang, X. Li, Y. Huang, Y. Sato, and Y. Lu, "Structural multiplane image: Bridging neural view synthesis and 3d reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2023, pp. 16 707–16 716.

A Performance measurements of MPI building

All times listed are averages of 5 measurements rounded to tenths of milliseconds. The measurements were taken on a Windows 11 laptop in performance mode with the following hardware:

- CPU: AMD Ryzen™ 9 7940HS
- GPU: AMD Radeon™ RX 7700S
- RAM: 64GB, 5600MT/s

A.1 Scene 1

- Scene: `Scene01`
- Variant: `clone`
- Start frame: 0

Total time (ms)	1 frame	2 frames	3 frames	4 frames	5 frames
32 planes	130.1	137.7	158.4	169.2	182.5
64 planes	240.3	262.8	268.1	277.1	315.8
128 planes	420.3	410.0	473.4	503.1	522.0

Time per viewpoint (ms)	1 frame	2 frames	3 frames	4 frames	5 frames
32 planes	8.9	8.6	9.1	8.5	8.3
64 planes	11.3	10.3	10.9	10.6	10.7
128 planes	16.1	14.7	14.0	14.4	14.4

A.2 Scene 2

- Scene: `Scene01`
- Variant: `clone`
- Start frame: 160

Total time (ms)	1 frame	2 frames	3 frames	4 frames	5 frames
32 planes	125.9	158.9	165.7	166.9	194.7
64 planes	229.9	264.4	267.2	286.9	305.0
128 planes	397.9	452.1	459.2	470.8	521.6

Time per viewpoint (ms)	1 frame	2 frames	3 frames	4 frames	5 frames
32 planes	9.6	8.9	8.7	8.6	8.7
64 planes	13.0	11.0	10.7	10.2	10.2
128 planes	15.2	14.2	14.2	13.8	14.2

A.3 Scene 3

- Scene: `Scene01`
- Variant: `15-deg-right`
- Start frame: 224

Total time (ms)	1 frame	2 frames	3 frames	4 frames	5 frames
32 planes	124.7	156.3	159.3	175.8	179.8
64 planes	230.1	249.9	289.0	291.2	312.9
128 planes	397.6	429.2	448.5	475.2	498.5

Time per viewpoint (ms)	1 frame	2 frames	3 frames	4 frames	5 frames
32 planes	10.6	9.2	9.0	8.6	8.4
64 planes	12.0	10.6	10.7	10.9	10.6
128 planes	15.4	15.2	14.0	14.3	14.2

A.4 Scene 4

- Scene: `Scene06`
- Variant: `30-deg-left`
- Start frame: 236

Total time (ms)	1 frame	2 frames	3 frames	4 frames	5 frames
32 planes	130.9	153.4	153.2	169.7	185.8
64 planes	245.2	258.4	236.7	285.3	293.5
128 planes	403.7	427.7	444.3	488.9	496.6

Time per viewpoint (ms)	1 frame	2 frames	3 frames	4 frames	5 frames
32 planes	9.6	9.3	8.9	8.6	8.8
64 planes	11.5	11.0	10.6	10.6	10.8
128 planes	15.0	14.5	14.1	14.5	14.2

A.5 Scene 5

- Scene: `Scene18`
- Variant: `clone`
- Start frame: 42

Total time (ms)	1 frame	2 frames	3 frames	4 frames	5 frames
32 planes	149.7	156.4	166.7	177.4	188.3
64 planes	241.2	251.3	274.8	257.6	304.0
128 planes	390.4	426.7	464.7	480.6	517.3

Time per viewpoint (ms)	1 frame	2 frames	3 frames	4 frames	5 frames
32 planes	9.9	9.5	8.6	8.9	8.9
64 planes	11.7	11.5	11.1	11.2	10.8
128 planes	15.0	14.8	14.9	14.5	14.5