Casual 3D Photography

LOUIS MARTINEZ*, ENS Paris-Saclay, France
MOHAMED ALI SRIR*, Dauphine PSL, France

In this report, we discuss key elements of Casual 3D Photography by Hedman et al. [3]. Their method reconstructs a 3D photo: i.e. a two-layered panoramic mesh with reconstructed surface color, depth, and normals, from casually captured cell phone or DSLR images, which can be rendered and modified. In particular, we detail the implementation of the depth map reconstruction and the stitching and two-layer fusion algorithms. Throughout this report, we also emphasize some crucial implementation decisions based on our interpretation of the original article.

Additional Key Words and Phrases: 3D reconstruction, Image-based rendering

1 Introduction

For accurate reconstruction, traditional methods rely on strong assumptions about the scene or the devices used to capture it (LiDAR, LASER). Hedman et al. in Casual 3D Photography [3] address these challenges by designing an end-to-end rendering pipeline that relaxes these strict requirements. They show that high-quality scene reconstruction can be achieved using only casual cameras (DSLR or phone), with all other aspects handled through software. Building upon the approach introduced in the original paper, we tackle its three main innovations from a technical point of view.

(1) *Depth-map reconstruction.* Although the plane sweep algorithm is a well-known method in multiview stereo (MVS) for depth estimation, the framework at stake here makes its vanilla implementation less reliable. Therefore, we implement here a more sophisticated version, taking into account the narrow baseline of the captured environment.
(2) *Parallax-tolerant stitching.* We combine all captured images into a panoramic representation. Given the nature of the acquisition, which includes overlapping regions with inconsistencies, we employ strategies to determine the most accurate color and depth. Additionally, we leverage both foreground and background information to preserve details obtained from multiple viewpoints. We end up with two panoramas, one for the foreground details and one for the background details.
(3) *Two-layer fusion.* We go back to the 3D space, by converting the two panoramas to a mesh with two layers (front and back), which is a denser reconstruction of the scene, that can be rendered with different view points, relighted and used for VFX.

The article, as well as the resources we used, can be found in Project page. Our code can be found on GitHub.

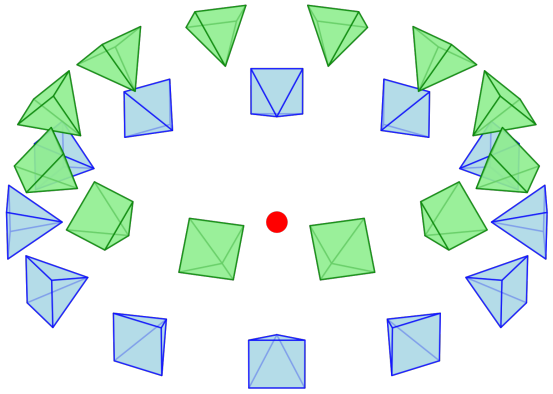

Fig. 1. **Setup configuration for capturing a scene.** Every scene of the provided datasets is captured from a central point of view (red dot), at two distinct levels (blue and green pyramids). The main motivation behind this configuration is to stick with the way a casual user would capture their surrounding environment. That way, we get a dense enough set of views to further reconstruct, sticking with a rather simple methodology.

2 Framework Definition

2.1 Images Pre-processing

Regardless of how they were acquired, every provided image follows a pinhole camera model, either natively if they were captured by a smartphone or after undistorsion. Furthermore, the intrinsic parameters are the same for all images from a given dataset since they were captured with the same device.

For consistency and to qualitatively validate our results, we used the images provided in the supplementary material of the article.

2.2 Scene parameters

The extrinsic and intrinsic parameters are determined using COLMAP2.1. Thanks to the acquisition methodology (Figure 1), we ensure that the estimated parameters are reliable enough, since there is enough overlap between views.

3 Dense depth maps reconstruction

(Louis Martinez)

In this section, we discuss two approaches for estimating dense depth maps. First, we introduce our implementation of a part of the near-envelope prior proposed by [3]. Then, we leverage a recent deep learning-based method proposed by Bochkovskii et al. [1], Depth Pro.

*Both authors contributed equally to this research.

Authors' Contact Information: Louis Martinez, louis.martinez@ens-paris-saclay.fr, ENS Paris-Saclay, Paris, France; Mohamed Ali Srir, mohamed-ali.srir@dauphine.eu, Dauphine PSL, Paris, France.

3.1 Multi View Stereo-based (MVS) approach

In the context of casually captured views of a scene, a traditional plane-sweep MVS algorithm fails to accurately construct dense depth maps. The main reasons claimed in the original article [3] are 1) the repeating patterns within images (Figure 2) and 2) large areas without texture, such as the sky. The main idea is to consider a finite number (set to $N = 220$ in the article) of possible depths for each pixel of an image.

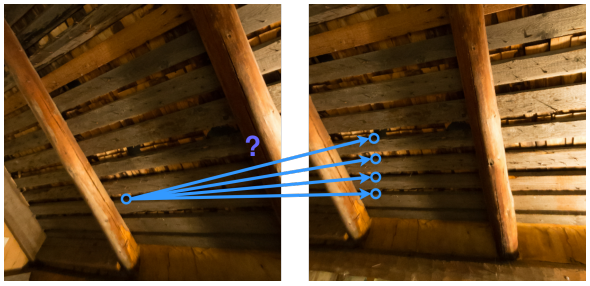

Fig. 2. **Example of ambiguity induced by repeating textures.** Both images are overlapping views of the same scene. The left image is used here as the one for which we want to construct the depth map with an MVS algorithm. Given a pixel (highlighted in blue on the left image) for which we want to estimate the corresponding position in a neighboring view, different depth hypotheses give potential candidate positions. However, since the cost function is computed based on the gradients and colors of both anchor and target images, the right candidate is ambiguous. This motivates the additional processes proposed by the author to ensure consistency across views.

3.1.1 Optimization framework. The authors introduce a *near envelope prior*. This prior aims to provide a first coarse estimate of the true depth maps. Ultimately, these maps are used as initialization for an optimization problem (Equation 1) introduced by [7]. This optimization problem is nothing but a labeling problem, where each pixel of an image at stake is attributed a depth such that the cost function reaches a minimum. Although the authors provided detailed explanations on how to implement this step of the pipeline, many key details were eluded. Therefore, we had to make concessions and adopt particular stances in our implementation. These elements are further detailed in Section 3.1.2.

$$\arg\min_{d} \sum_{i \in N} E_{data}(i) + \lambda_{smooth} \sum_{(i,j) \in N} E_{smooth}(i,j) \qquad (1)$$

We do not further detail the optimization scheme of the depth estimation pipeline since we haven't implemented it. Instead, we focus on near-envelope prior. Given a set of preprocessed images, the pipeline is as follows.

(1) *Determination of anchor pixels* A first estimate of depth maps is computed for each image. To do so, we apply a Winner-Takes-All (WTA) strategy. Let us say that we want to determine the first approximation of the depth map of an image I of shape $m \times n$. Concretely, for each depth hypothesis, every pixel of I is projected on 16 neighboring images. Then, a cost corresponding to this depth is attributed to each pixel. This cost corresponds to the term E_{data} in Equation 1. We end up with a cost volume of shape $N \times m \times n$. The coarse depth map is obtained by choosing the depth corresponding to the lowest cost in the first dimension of the volume. *We have only reimplemented this first part. The reasons for this restriction are detailed further in Section 3.1.2.*

(2) *Filtering* The first depth map estimate turns out to be very noisy. Therefore, some depths are pruned to ensure geometric consistency across all captured images. This step is the key to overcome repeated texture or textureless ambiguities (Figure 2). This process is complemented by two median filters that are responsible for removing the remaining outliers. At the end of this filtering step, the pixels that have not been pruned are the *anchor pixels*.

(3) *Depth propagation* Once outliers are removed, the depths of the anchor pixels are propagated to neighboring pixels to which no depth is attributed (former outliers). To ensure a smooth evolution of depth, the authors compute a term based on color affinity between anchor pixels and former outlier pixels.

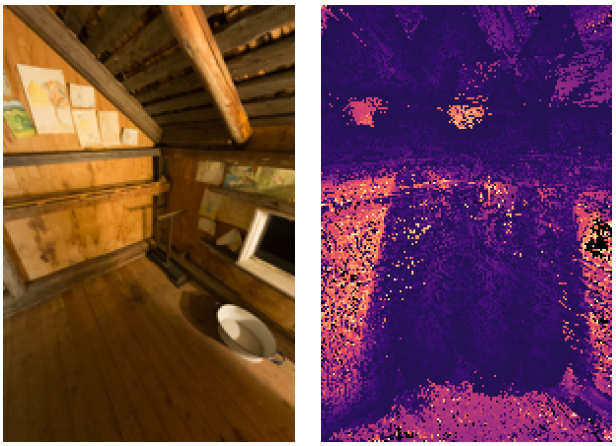

Fig. 3. **Results of our implementation of the WTA depth map estimate.** The left image is the view for which we want to determine a first coarse estimate of the near-envelope prior. The right image is the result obtained with our implementation. A direct comparison with the results of the paper is not possible since we are not provided the true maps. However, it is obvious that the obtained result is of very poor quality. Some issues explaining such result might be the heavy downscaling factor applied to the original image ($\times 16$), the choice of neighbor images, or some slight implementation choices that are not compatible with the hyperparameter values suggested by [3].

3.1.2 Implementation. Following strictly the instructions in the paper, our implementation of the WTA depth map encountered both performance and quality hits. To overcome these issues, we made several implementation choices that allowed one to at least confirm the validity of our implementation. For the remainder of this section, we reuse the notation of the original paper [3] without reintroducing them.

Scaling convention The Plane-Sweep MVS algorithm introduced in the paper involves several hyperparameters (σ_{photo}, α_{∇}, w_{min},

etc.). Their values are fixed as those providing the best results after extensive tuning by the authors. However, the authors do not explicitly specify whether the color channels are within $[[0, 255]]$ or between $[[0, 1]]$. The same question holds for computing the gradients of the images, but this is discussed further. After trying both conventions, using a range between 0 and 1 seemed to provide more meaningful results. Using the 0-255 convention leads to saturated values when computing the cost volume. The intermediate values turned out to be binary, which does not make sense.

Down-scaling factor Although the authors of [3] praise the qualities of their approach, they briefly mention its computational expense. However, this pitfall turned out to be a major issue in our experiments. For example, let us assume that we want to estimate the WTA depth map. For an image of shape 900×1350 (as used in the paper), 220 depth hypothesis, and 32-bit encoded floating values, storing a cost volume requires nearly 1GB of memory. This value does not take into account intermediate computations (such as projection onto neighbor images). In addition, despite heavy parallelization, the processing time for a single depth map exceeds 40 minutes on a standard laptop with a Ryzen 7 5700U CPU. To still be able to generate a coarse depth map, the original images (of shape 3560×2372) were reduced by a factor 16. In this case, the computations for a single depth map were reduced to 9 minutes. This choice is also what prevents us from implementing the rest of the pipeline. The authors emphasize that the filter sizes of the next steps of the pipeline were *carefully* chosen with respect to the shape of the images they process. Changing these hyperparameter values with more suitable ones would require an extensive and time-consuming parameter search.

Neighbor images selection In order to compute the cost E_{data}, each pixel is projected onto 16 neighbor images. However, automatically selecting these images can be challenging. In our experiments, we bypass this difficulty by using the spatial configuration of the captured images (Figure 1). Given an image for which we want to estimate the depth map, its neighborhood is manually selected by visually inspecting all the images. Although automatic methods based on feature matching or view frustum intersection can be setup, manually choosing them for the purpose of our experiments remains the most reliable method. Figure 4 illustrates which images are selected as neighbors.

Image gradient estimation There are several ways to estimate the gradient of a RGB image, but none of them are mentioned in the paper. As mentioned in the Scaling Conventions paragraph, gradients could be determined from images ranging either from 0 to 255 or from 0 to 1. Given the hyperparameter values provided, we use the range $[0, 1]$. We also ended up estimating the gradients on the corresponding grayscale image with a Sobel filter. The last point to clarify, for which we do not have an unequivocal answer, is whether we compute gradients before or after downscaling the input image.

3.2 Deep Learning-based Depth Estimation

In 2017, the authors of [3] mentioned the poor performance of deep neural network (DNN)-based approaches for depth estimation. The first reliable (and widely cited) method was MVSNet by Yao et al. (2018, [8]). However, recent breakthroughs in deep learning-based depth estimations lead us to evaluate one of these state-of-the-art approaches with the one proposed in [3]. Currently, Depth Pro, proposed in 2024 by Bochkovskii et al. [1] (Apple) seems to be the most promising approach. In particular, it claims accurate results on single-view, zero-shot absolute depth estimation. The main strength of Depth Pro is that it is based on a multi-scale vision transformer [5]. Transformer-based architectures have shown significantly improved results compared to legacy Convolutional Neural Networks. This approach was not available when [3] was released since Vision Transformers [2] were introduced in 2020.

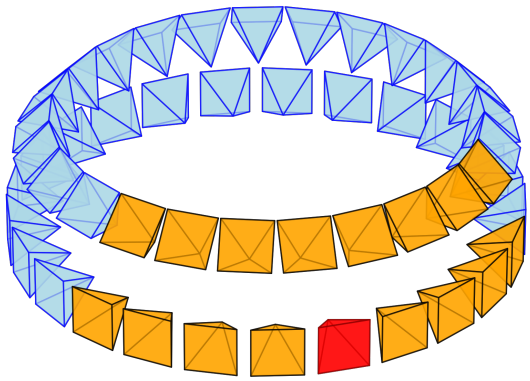

Fig. 4. **Setup configuration for capturing the scenes.** Given a view for which we want to compute the depth map (red pyramid), we select 4 views (orange pyramids) on the left, 4 on the right, and 8 from the top (resp.) bottom row. The remaining views (blue pyramids) are not taken into account.

3.2.1 Motivation. Regardless of the accuracy and reliability of deep learning-based methods, they benefit from being highly parallelizable on GPU. Therefore, we can estimate dense depth maps in a reasonable amount of time (less than 10 seconds on a V100 GPU with 32GB VRAM) in full resolution. In contrast, the method proposed by [3] is executed on a PC with 256GB RAM with lower resolution images. In addition to that, depths predicted by a DNN are continuous, thus not restricted to 220 predefined depth hypotheses.

3.2.2 Results. As illustrated in Figure 5, the depth maps obtained with Depth Pro are visually coherent and better preserve fine-grained details. In some cases, though, Depth Pro output depth maps with a lower range of depth than the MVS approach, and therefore many erroneous depths (central column).

4 Stitching and Two-layer Fusion

(Mohamed Ali Srir)

From the previous steps, we have now for each image the position of the camera, and color and depth map, so we can now reconstruct a 3D object for each view (Mesh or Point Cloud).

In the next sections, we will respectively construct a 3D object of each view, merge all these view into a single panorama, and then reconstruct a new, more compact single 3D object from it.

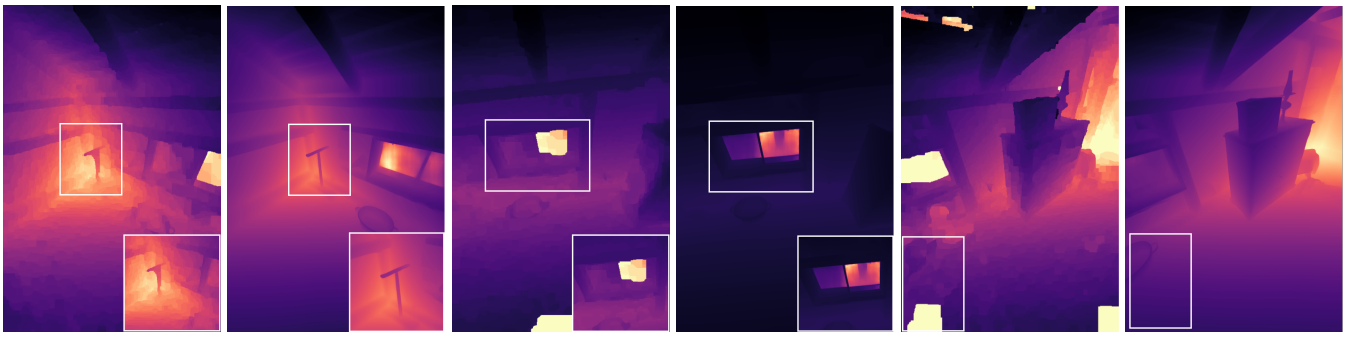

Fig. 5. **Comparison between MVS and DNN-based approaches.** For each image, we highlight key differences. As mentioned in Section 3.2, depth maps generated by Depth Pro are generated with nearly full resolution (1536 × 1536 in practice according to the model documentation). This observation appears to be obvious for the first two images on the left. Fine grained details are well conserved by Depth Pro. The center images show that the discrete depth distribution of the MVS approach fails to recover far details such as the tree behind the window. Finally, the right images illustrate that Depth Pro overcome some artefacts due to misclassified pixels.

(2D images + Depth + camera -> 3D object -> Front and back 2D Panorama + depth -> 3D Compact Mesh)

4.1 Depth map to 3D object

From a camera estimated with COLMAP, along with a color image and a corresponding depth map, we can reconstruct a 3D object as a point cloud or a dense mesh. The point cloud provides a direct representation, but often contains holes because of missing depth information. In contrast, the mesh offers a continuous surface, though it may include elongated triangles in regions with sparse depth data. In the paper, they used a mesh reconstruction.

Fig. 6. **Difference between Point Cloud and Mesh Reconstruction.** For performance, the original images are subsampled.

Figure 7 illustrates the reconstruction process from RGB-D images and camera poses, while Figure 6 highlights the differences between point cloud and mesh representations.

4.2 Front and Back Surface Warping

The transformation from a 3D point cloud to an equirectangular projection involves two main steps: Cartesian-to-Spherical conversion and Spherical-to-Equirectangular mapping. First, each point in the 3D space (X, Y, Z) is converted to spherical coordinates (radius r, latitude θ, and longitude ϕ). The radius represents the distance from the origin, while latitude and longitude define the point's angular position on a sphere. Since the camera trajectories are roughly circular as presented in figure 1, the center of the spherical coordinates is the center of all cameras. Then, these spherical coordinates are mapped onto a 2D equirectangular projection, where longitude (ϕ) determines the horizontal position and latitude (θ) determines the vertical position in the image. The equirectangular projection unwraps the sphere into a rectangular RGBD (Red Green Blue Depth) image (with the depth being r), ensuring that every point on the sphere corresponds to a unique pixel in the panoramic view. This method is commonly used in 360-degree imaging, VR, and panoramic rendering (you can put our panorama here), making it a standard format for displaying spherical data on flat screens.

In the paper, they used a mesh representation for rendering and projection, employing a specialized depth test to mitigate artifacts caused by elongated triangles. This depth test artificially pushes triangles with a grazing angle close to zero (nearly parallel to the view direction) farther away than they actually are, preventing visual distortions. In our pipeline, instead of mesh rasterization, we perform 3D point-cloud rasterization, achieving comparable results at the chosen resolution without requiring a depth test. The projection of an example image is shown in figure 8. Figure 10 shows that the effect of the modified depth is negligible, in the context of point cloud rasterization, since the number of modified points is very small. The only potential drawback compared to meshes is the possibility of holes if the point cloud is too sparse; this can occur when subsampling the source RGBD images while maintaining a high resolution for the panorama, like showcased in figure 9, but we can easily control this and it is compensated by the overlapping. We can use a median filter, for example, if the gaps are not narrow enough.

For performance constraints, our panorama resolution is 1024 by 512, while subsampling with a stride of size 6 the RGBD images of size 2300 × 3500, while the paper uses a 8192 × 4096 resolution for the panorama and full resolution.

Fig. 7. **Example of Multiview RGBD Images Reconstruction.** Every scene of the dataset, is a set of cameras with a circular trajectory, that we can reconstruct as a 3D object, here we plot in red green blue, 3 different cameras with their respective views and depth.

Fig. 8. **Difference between Paper's warping and our implementation.** On the left the projected image, on top the paper's result and in the bottom our result.

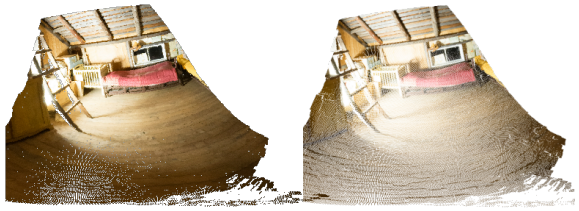

Fig. 9. **Limits of Point Cloud rasterization in High dimension with high subsampling.** The projection of a subsampled RGBD image with stride 6, on a 1024x512 pano on the left and on a 2048x1024 pano on the right, we can see that the latter one is highly discontinuous.

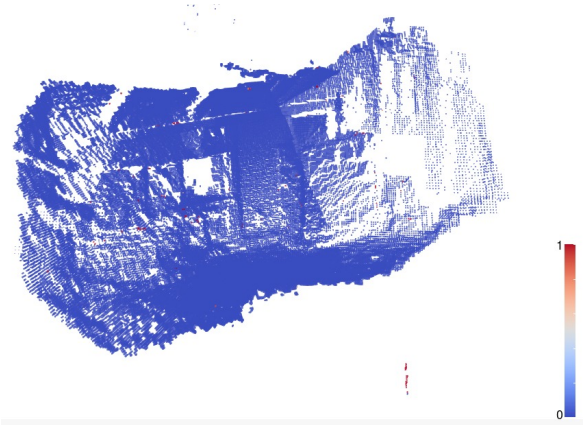

Fig. 10. **Mapping of the stretch term.** We plot the value of the stretch term as defined in the paper for the same point cloud as in figure 7. Very rare term have a stretch that is close to one, justifying that the utility of this term in Point Cloud rasterization is less useful.

4.3 Stitching

In our implementation, inspired by the Single Front and Back Layer Stitching method from the paper, we address the multilabel problem that arises when rendering a mesh or point cloud onto a panorama. Since multiple vertices of different images can project onto the same pixel, we must select the most representative label, determining the optimal depth and color among the candidates. The original approach formulates this as an energy minimization problem using graph cut optimization, with multiple terms - including a regularization term - contributing to the final decision. The stretch term plays a crucial role in discouraging stretched triangles, ensuring

that more reliable depth values are selected. The graphcut is computationally heavy. Additionally, their approach retains both the most front and back-facing layers by reversing the depth test. In our simplified version, we take a more straightforward approach by keeping only the most upfront and most backward layers. Despite this simplification, we still achieve decent results, demonstrating that a minimalistic strategy can be effective (figure 11).

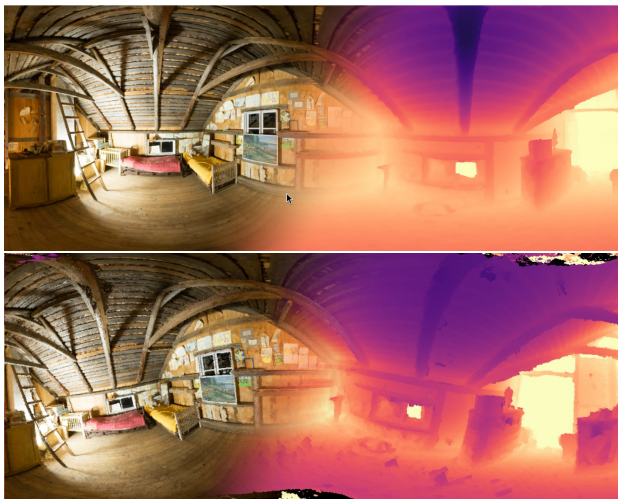

Fig. 11. **Comparison between Paper's panorama and ours.** On top paper result, in the bottom our RGBD Panorama.

Having two layers is important because we keep information of regions occluded by the front view; it is especially handy when we want to go back to the 3D domain. Several examples are given in figure 13, to see the front and back details.

4.3.1 Comparison with Depth Pro. Since we have depth maps estimated by Depth Pro, which visually seem to be accurate with those provided by the paper (Figure 5), we can use them to stitch them. Again, this experiment only provides a qualitative evaluation of Depth Pro. Quantitative comparison is not reliable since key steps of the method of the article [3], such as down / upsampling with interpolation, are not reliable, or the relatively small number of depth hypotheses introduces major discrepancies. However, if the stitched panorama obtained is coherent, it is enough to assess the potential of this approach.

4.4 Merging

Now that we have a front and a back panorama, we can render from them a new 3D object that is more compact, by rendering back to 3D space and doing some tuning on the topology (see Section 4.4.3 of the paper), the new 3D object is much more compact (from 50 point clouds with 2000×3000 points to 2 point clouds of 2048×1024). However, we can see that even if one keeps information from occluded parts (example with roof and ladder in figure 13), we do not have true view-dependence like NeRF [6] or Gaussian [4] (Figure 13, right column, we cannot reproduce view-dependent reflections properly).

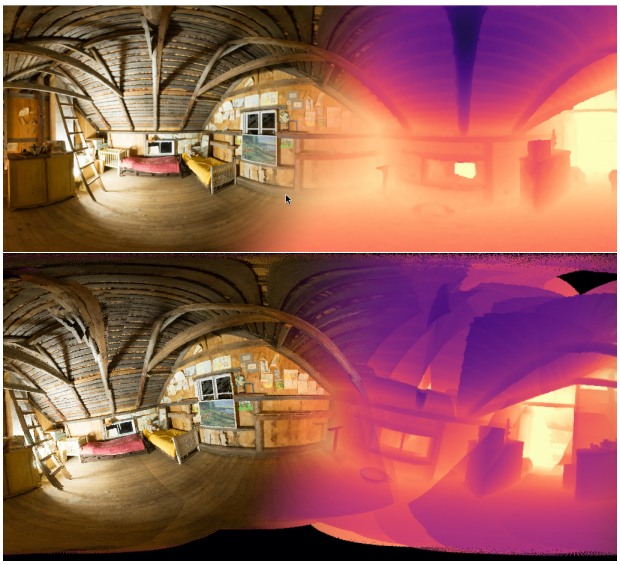

Fig. 12. **Comparison between Paper's panorama and ours+Depth Pro.** On top paper result, in the bottom our RGBD Panorama with depth maps estimated by Depth Pro.

5 Conclusion

This work focuses mainly on the implementation of some key steps of the method proposed by Hedman et al. [3]. Despite the overall detailed implementation instructions, we had to formulate many assumptions on how the authors actually had implemented their pipeline. We also formulate simplified hypotheses that allow us to accurately render a captured scene, despite obvious discrepancies with the original method.

Limitations. Our pipeline allows for accurate scene reconstruction, but at a much lower resolution than the original method. The dense reconstruction only requires 8 hours of computation, which is far from the 5 hours claimed in the paper for the whole pipeline. The use of Python instead of C++ is one of the reasons for this performance hit. Furthermore, the hardware configuration at our disposal cannot compete with the one used by the authors.

Further improvements. Our key attempt to improve this method was to delegate the estimation of the dense depth map to a DNN. For simplicity, though, we used a single-view, zero-shot approach that leads to inconstencies between estimated depth maps. The two ways to overcome this issue are to reintroduce consistency, following the idea of the original paper, or to use a neural network specifically optimized for MVS depth estimation.

References

[1] Aleksei Bochkovskii, Amaël Delaunoy, Hugo Germain, Marcel Santos, Yichao Zhou, Stephan R. Richter, and Vladlen Koltun. 2024. Depth Pro: Sharp Monocular Metric Depth in Less Than a Second. *arXiv* (2024). https://arxiv.org/abs/2410.02073

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV] https://arxiv.org/abs/2010.11929

[3] Peter Hedman, Suhib Alsisan, Richard Szeliski, and Johannes Kopf. 2017. Casual 3D Photography. 36, 6 (2017), 234:1–234:15.

Fig. 13. **Difference between Front (F) and Back (B) wrapping for different part of the Panorama.** As observed in the original paper, pixel corresponding to nearer elements in the scene tend not to appear in the background. Section 4.4 provides an in-depth interpretation of how the window in the two images on the right hand side of the figure is processed at this step.

[4] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[5] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. arXiv:2103.14030 [cs.CV] https://arxiv.org/abs/2103.14030

[6] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. arXiv:2003.08934 [cs.CV] https://arxiv.org/abs/2003.08934

[7] D. Scharstein, R. Szeliski, and R. Zabih. 2001. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*. 131–140. doi:10.1109/SMBV.2001.988771

[8] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. 2018. MVSNet: Depth Inference for Unstructured Multi-view Stereo. arXiv:1804.02505 [cs.CV] https://arxiv.org/abs/1804.02505