Point Cloud Generation from Stereo Images

Signal, Image and Video Course Project

Christian Dalvit (249988)
University of Trento
Trento, Italy
christian.dalvit@studenti.unitn.it

I. Introduction

Stereo matching is the process of taking two images and constructing a 3D model of the scene by identifying corresponding pixels in both images. The objective is to convert 2D pixel positions into 3D depth information [13]. Stereo matching techniques have a wide range of applications, including autonomous navigation, robotics, and 3D reconstruction of infrastructure and products. A robust and efficient implementation of stereo matching is essential for accurate real-time 3D perception.

The objective of this project is to develop a pipeline for generating point clouds from stereo image pairs. The stereo matching component is implemented without relying on the OpenCV library [4], using only the NumPy library [6]. The implemented algorithms are evaluated both quantitatively and qualitatively. Processing time is measured and optimized by implementing algorithms in different programming languages and leveraging parallelization. The project is available on GitHub.

II. Theory

A. Epipolar Geometry

The theoretical foundation of stereo matching is epipolar geometry. Let c_0 and c_1 be the optical centers of the left and right cameras for a given stereo setup. The baseline is the line connecting c_0 and c_1. A 3D point p projects onto the image planes as points x_0 and x_1. The key geometric constraint is that the three points c_0, p, and c_1 define a plane called the epipolar plane. Each epipolar plane intersects the image planes along lines called epipolar lines, denoted as l_0 and l_1 [7]. This implies that given a point x_0 in the left image, its corresponding point x_1 in the right image must lie on the corresponding epipolar line l_1 [7]. This constraint significantly reduces the search space for matching points. Figure 1 illustrates how epipolar geometry links the projections in the two image planes.

Although epipolar geometry applies to any roto-translation between c_0 and c_1, images are often rectified in practice [13]. Rectification involves warping the input images so that corresponding horizontal scanlines become epipolar lines [13]. This transformation results in the following relationship between the pixel coordinates $x_0 = (u_0, v_0)$ and $x_1 = (u_1, v_1)$:

$$u_1 = u_0 + d(u_0, v_0), \qquad v_1 = v_0$$

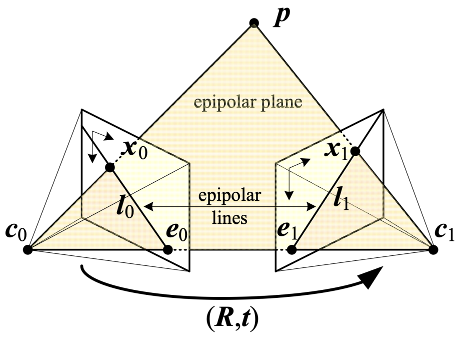

Fig. 1. The camera centers c_0 and c_1, along with the 3D point p, define an epipolar plane. The intersection of this epipolar plane with the two image planes forms the epipolar lines l_0 and l_1. These corresponding epipolar lines define the search space for matching points.

where $d(u_0, v_0)$ is called the disparity at pixel location x_0. Collecting the disparities for all image locations produces the disparity map. The task of estimating depth from a stereo image pair then reduces to computing the disparity map [13]. For the purposes of this project, it is always assumed that the input images are rectified.

B. Disparity Estimation

The problem of finding matching points can be reduced to estimating the disparity d for a given pixel location. Various approaches exist for disparity estimation, but discussing all of them is beyond the scope of this report. Therefore, this section provides an overview of the algorithms implemented in this project. Further details on these algorithms, along with their implementation, are discussed in Section IV.

Local methods constitute the first group of algorithms for estimating disparity. These methods aggregate the matching cost by summing or averaging over a support region [13]. Although, in theory, the matching pixel lies on the corresponding horizontal line in the second view, in practice, a window around the pixel is used as the support region to obtain more robust estimates. A cost function $C(u, v, d)$ is computed over this region, and the estimated disparity d^* is then the disparity that minimizes the cost:

$$d^* = \underset{d}{\operatorname{argmin}}\ C(u, v, d)$$

Local methods offer significant flexibility in choosing different matching costs and region selection strategies. For a more comprehensive overview of the various approaches, refer to [13]. A limitation of this approach is that the uniqueness of matches is enforced only in the reference image, meaning that points in the other image may correspond to multiple points unless cross-checking and subsequent hole filling are applied [13].

A different approach to disparity estimation is based on *global methods*. These methods are formulated within an energy-minimization framework, where the objective is to find a disparity function d that minimizes a global energy:

$$E(d) = E_D(d) + \lambda E_S(d) \tag{1}$$

where $E_D(d)$ measures how well the disparity function agrees with the input image pair, and $E_S(d)$ enforces smoothness constraints, i.e., that neighboring pixels should have similar disparity values [13]. Performing this energy minimization in the 2D plane is known to be NP-hard [13]. However, for the 1D case, efficient algorithms exist to solve the energy-minimization problem [3]. Combining multiple 1D optimization directions and aggregating their path costs leads to *semi-global matching* methods [8]. Semi-global matching algorithms are both efficient and perform well in practice [13].

C. Depth Estimation

After computing the disparity for each pixel, the depth of a point can be estimated using the following equation:

$$d = f\frac{B}{Z}$$

where B is the baseline (i.e., the horizontal distance between the two camera centers), f is the focal length, and Z is the depth. Since f and B are fixed parameters, the equation can be rearranged to solve for Z once d is known [13]. The world coordinates X and Y of a pixel can be computed using similar equations, provided the camera's world coordinates are known.

From the discussion so far, it is clear that the most challenging part is disparity estimation, as depth computation becomes straightforward once the disparity is known.

III. State of the Art

So far, a brief overview of the theory behind traditional reconstruction methods has been presented. Many approaches have not been discussed, such as segmentation-based methods [14] and hierarchical techniques [2, 1]. Among traditional methods, the semi-global matching (SGM) pipeline by Hirschmüller [8] is particularly popular. It achieves a good trade-off between performance and quality and is one of the default implementations in the widely used OpenCV library [4].

Beyond traditional methods, deep learning approaches have gained increasing popularity in depth estimation and point cloud reconstruction. These methods enable end-to-end learning of disparity map prediction. Supervised deep learning techniques now dominate individual benchmarks that include dedicated training sets [13]. The survey by Tosi et al. [15] categorizes deep learning approaches into five groups: CNN-based cost volume aggregation, Neural Architecture Search for stereo matching, iterative optimization-based methods, Vision Transformer-based models, and Markov random field-based architectures.

Despite the success of deep learning architectures in the field of stereo reconstruction, Tosi et al. identify several limitations across all five categories [15]: Deep learning models for depth estimation face several challenges:

- *Domain shift*: Performance drops when models trained on one domain are deployed in another, often due to variations in illumination, color, environment, or stereo camera setups [15].
- *High-resolution images*: Depth estimation models struggle with high-resolution images, though coarse-to-fine techniques have been developed to address this issue [13].
- *Non-Lambertian objects*: These objects, including those with specular reflections, challenge both deep learning and traditional methods. The lack of training data for such scenarios complicates training deep networks, but recent efforts focus on creating relevant datasets [10].

Finally, Tosi et al. argue that while foundation models are emerging for other computer vision tasks, they are still missing in the field of stereo matching [15]. Some progress has been made in single-image depth estimation [17], and recent efforts by Wen et al. [16] aim to close this gap.

IV. Implementation

This section first provides an overview of the entire reconstruction pipeline. Next, the details of the two implemented stereo matching algorithms are discussed. Finally, the sub-pixel estimation method used for both matching algorithms is explained.

The primary objective of this section is to analyze the matching algorithms and their implementation. Both algorithms were implemented without using the OpenCV library [4], relying solely on Python and NumPy [6]. Additionally, the block matching algorithm was implemented in C to explore potential avenues for performance improvements. The results and performance of the different algorithms are compared in Section V.

A. Reconstruction Pipeline

Before discussing the implemented algorithms, an overview of the reconstruction pipeline is provided. The pipeline is depicted in Figure 2. The following processing steps are applied to the rectified input images:

1) **Grayscale conversion**: Both images are converted to grayscale as they carry sufficient information while simplifying the matching cost computation.
2) **Stereo matching**: The left image is used as the reference, and corresponding points are searched for in the right image using a matching algorithm. The output of this stage is a coarse disparity map. This part is implemented without OpenCV.
3) **Median blur**: The coarse disparity map may contain many outliers. A median blur with a window size of 5 is applied to eliminate these outliers.

4) **Reconstruction**: The disparity map is converted into a 3D point cloud using OpenCV's `reprojectImageTo3D` method.

Finally, the generated point cloud is saved in the Polygon File Format (PLY) for visualization and further processing. Next, the two algorithms implemented for the stereo matching step are discussed in detail.

B. Block Matching

Block matching is one of the simplest local stereo matching approaches. In this project, block matching is implemented by moving a window along the corresponding horizontal line in the right image I_1 for every pixel (u, v) in the left image I_0. For every offset $d \in \{0, \ldots, D\}$, the cost $C(u, v, d)$ is computed, where D is the fixed maximum disparity. In the implementation, the average sum of squared differences is used as the matching cost function:

$$\frac{1}{N} \sum_{(i,j)\in R} [I_0(u+i, v+j) - I_1(u+i-d, v+j)]^2$$

The estimated disparity d^* is then the disparity with the lowest cost $d^* = \text{argmin}_d\ C(u, v, d)$.

In the project implementation, a window size of 15 and a maximum disparity $D = 64$ were used. Note that the offset is subtracted from the image coordinates. Following the projection rules for rectified images, the image coordinates of a point in the right image are farther left than the coordinates of the same point in the left image. The pseudocode for the block matching algorithm is displayed in Algorithm 1, where K denotes the window radius, $C(u, v, d)$ denotes the average sum of squared differences, and O is the output disparity map. The output disparity map is initialized with all zeros.

Algorithm 1 Block Matching

```
for $K \le y \le height(I_0) - K$ do
    for $D \le x \le width(I_0) - K$ do
        $d^* \leftarrow$ NULL
        $c_{\text{min}} \leftarrow \infty$
        for $0 \le d \le D$ do
            $c \leftarrow C(u, v, d)$
            if $c < c_{\text{min}}$ then
                $c_{\text{min}} \leftarrow c$
                $d^* \leftarrow d$
            end if
        end for
        $O(u, v) \leftarrow d^*$
    end for
end for
```

Note the boundary conditions of the two outermost loops. The y coordinate has been cropped by K to handle the window shifting. The x coordinate starts at D because the pixels on the far left of the left image are not present in the right image, assuming $K \le D$.

C. Semi-Global Matching

The Semi-Global Matching (SGM) algorithm presented by Hirschmüller [8] is based on the idea of pixel-wise matching and approximating a global 2D smoothness constraint from Equation 1 by combining many 1D constraints [8]. The implementation of the Semi-Global Matching algorithm in this project is inspired by [8], but it is not identical. The distinct processing steps of the implementation are discussed in the following paragraphs.

a) Census Transform: Instead of matching pixels by their grayscale values, the census transform [18] is applied to the left and right images. The census transform computes a bit string for each pixel, which is then used for matching. The census transform generates a bit string based on the relative intensities of the neighboring pixels. If the intensity of a neighboring pixel is less than that of the center pixel, a 1 is added to the bit string; otherwise, a 0 is added. This operation is repeated for all neighboring pixels to create the bit string. The ordering of the bits in the string contains enough information to perform an accurate and fast correspondence operation. Figure 3 shows the application of the census transform to an image patch. The census transform computation is implemented by iterating over the image and calculating the census transform for each pixel. The bit string is stored as an integer.

b) Cost Computation: After applying the census transform to the left and right images, the Hamming distance is used to compute the cost. The Hamming distance compares two bit strings and counts the number of differing bits (e.g., 101 and 110 have a Hamming distance of 2 since they differ in two bits). The computation of the cost $C(u, v, d)$ for the pixel (u, v) and disparity $d \in \{1 \ldots D\}$ can be implemented as follows:

$$C(u, v, d) = \sum CT_0(u, v) \oplus CT_1(u - d, v)$$

Where $\oplus$ denotes the logical XOR operation, CT_0 and CT_1 are the census-transformed images, and the summation is performed over the bit string. The cost computation step can be implemented by iterating over $d \in \{1 \ldots D\}$, shifting the right image CT_1 by d pixels, and computing $C(u, v, d)$ pixel-wise. The NumPy library can be leveraged to perform this computation efficiently. All costs are collected in a cost volume with dimensions $\mathbb{N}^{H \times W \times D}$, where H and W are the image dimensions, and D is the maximal disparity.

c) Cost Aggregation: Pixelwise cost calculation is generally ambiguous and noisy. As mentioned in section II-B, minimizing the energy from Equation 1 in 2D is NP-hard [13]. Therefore, Hirschmüller proposes to minimize the energy from Equation 1 in 1D for different directions r and aggregate the costs. This acts as an additional constraint that supports smoothness by penalizing changes in neighboring disparities [8]. This results in a smoother cost volume and hence a smoother disparity map. The path cost $L_r(p, d)$ for a pixel

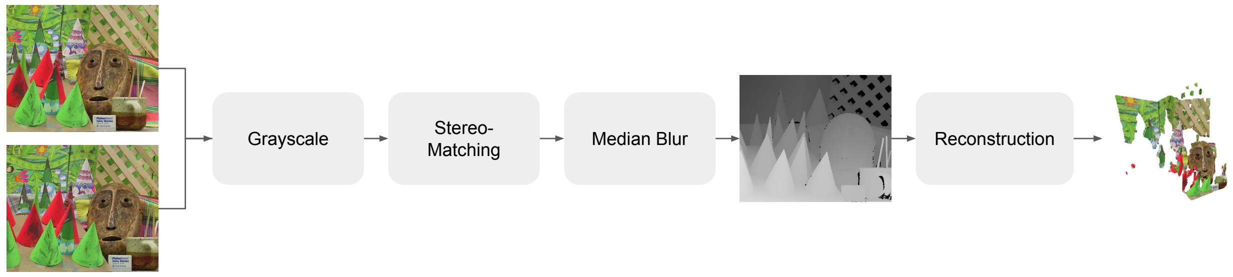

Fig. 2. The reconstruction pipeline consists of the following processing steps: grayscale conversion, stereo matching, application of median blur, and 3D reconstruction.

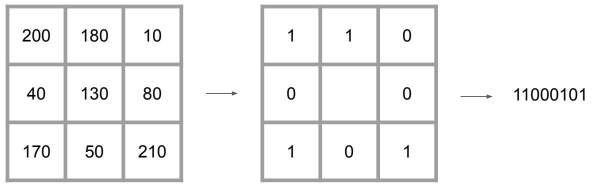

Fig. 3. The census transform compares the neighbouring intensities with the center pixel intensity. The value 1 is stored if the neighbor has a greater intensity as the center pixel and 0 is stored otherwise. After the comparison the patch is flattened into a bit string.

p and direction r is computed by the following expression:

$$C(p,d) - \min_k L_r(p-r,k) + \min \begin{cases} L_r(p-r,d) \\ L_r(p-r,d-1) + P_1 \\ L_r(p-r,d+1) + P_1 \\ \min_i L_r(p-r,i) + P_2 \end{cases}$$

The second term is used to prevent $L_r(p,d)$ from permanently increasing along the path [8]. P_1 and P_2 are constant penalties, where P_1 penalizes small disparity changes and P_2 penalizes larger disparity changes [8]. Using a lower penalty for small changes permits adaptation to slanted or curved surfaces [8]. In the project implementation, the penalties are set to $P_1 = 10$ and $P_2 = 120$.

The costs L_r are summed over paths in all directions r to obtain the aggregated (smoothed) cost:

$$S(p,d) = \sum_r L_r(p,d)$$

Although Hirschmüller recommends using 8 to 16 paths [8], Žbontar and LeCun state that they gained no accuracy improvement from using more than 4 paths [19]. Therefore, the SGM algorithm implemented in this project uses 4 paths (i.e., north, south, east, west directions). Note that the smoothed costs form a cost volume $S \in \mathbb{N}^{H \times W \times D}$. By selecting

$$d^* = \underset{d}{\operatorname{argmin}}\ S(p,d)$$

for every pixel p, the final disparity map can be extracted from the cost volume.

D. Sub-pixel estimation

So far, all algorithms computed a discrete disparity value $d \in \{1, \ldots, D\}$ [13]. When reconstructing the point cloud from the disparity map, this results in discrete levels of the disparity map. The left image in Figure 4 shows this effect. Since this does not correspond to a realistic reconstruction of the scene, sub-pixel estimation can be used to obtain a more continuous scene reconstruction. Sub-pixel estimation uses the costs of the two neighboring disparities $C(d^*-1)$ and $C(d^*+1)$ of the optimal disparity d^* to compute the minimum of the parabola interpolated through the three points:

$$d^*_{\text{ref}} = d^* + \frac{C(d^*-1) - C(d^*+1)}{2C(d^*-1) - 4C(d^*) + 2C(d^*+1)}$$

The refined disparity d^*_{ref} is then stored in the final disparity map. In the project implementation, sub-pixel estimation can be activated for both algorithms with a command-line argument.

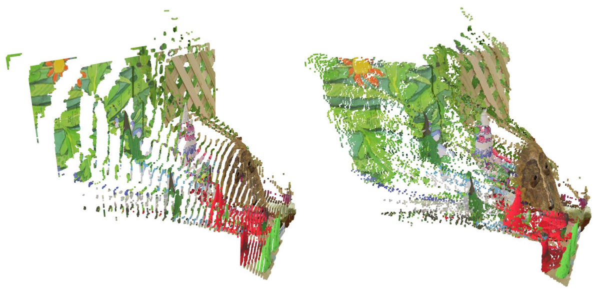

Fig. 4. *Left*: Point cloud reconstruction without sub-pixel estimation. The point cloud has clearly visible levels, caused by the discrete integer values in the disparity map. *Right*: Point cloud reconstruction with sub-pixel estimation. The point cloud is smooth, and objects like the mask appear as a continuous surface.

E. Programming Language

All algorithms described in this section have been implemented in Python using the NumPy [6] library. Although NumPy invokes C functions in the background, the nested loops in the Python code create a significant performance bottleneck. To overcome this bottleneck, the block matching algorithm has also been implemented in C. A binding for the

TABLE I
PERCENTAGE OF BAD PIXELS OF DIFFERENT ALGORITHMS

	Cones	**Teddy**
Block Matching	34.31	39.36
+ Sub-pixel	*32.46*	*37.38*
SGM	34.29	40.3
+ Sub-pixel	32.81	38.33
OpenCV	31.09	36.34

TABLE II
PROCESSING TIME OF DIFFERENT ALGORITHMS

	Time
Block Matching (Python)	30.08s
SGM (Python)	6.46s
Block Matching (C)	1.98s
Block Matching (C) + OMP	0.75s
OpenCV	0.02s

C code allows the user to invoke the C implementation from Python. Using C enabled further performance improvements. The C code can be linked with the OpenMP library [5] to parallelize the execution of the outermost loop. Performance results are discussed in Section V.

V. RESULTS

The implemented algorithms have been evaluated on two selected scenes from the Middlebury stereo dataset collection [12, 11, 9]. All images have dimensions of 450×375 pixels. The extracted disparity maps have been evaluated quantitatively by computing various metrics and qualitatively by visual inspection.

Figure 5 shows the computed disparity maps of the implemented methods without sub-pixel estimation. The OpenCV disparity map is included for reference. The SGM algorithm produces a smoother disparity map, with fewer holes (deep blue patches in the disparity map). However, the smoothing in the SGM algorithm also removes some details in the background (e.g., in the top right corner). The OpenCV algorithm performs an additional left-right consistency check by computing disparity maps for both the left and right images. This allows the OpenCV algorithm to mark occluded pixels as invalid (i.e., assigning them a disparity of zero). Both implemented algorithms do not use a left-right consistency check, and therefore, occluded pixels are assigned a disparity.

Using sub-pixel estimation produces a smoother disparity map. The effects of sub-pixel estimation on the reconstructed point cloud can be observed in Figure 4. The smoothed disparity maps are shown in Figure 6. Especially for the mask and the rounded cones, sub-pixel estimation produces more realistic reconstructions of the scene compared to disparity maps without sub-pixel estimation.

Apart from a qualitative evaluation, quantitative metrics can be computed as the Middlebury stereo datasets provide ground truths [12, 11, 9]. Table I reports the percentage of bad pixels (PBP) for different algorithms and images. PBP reports the percentage of pixels where the difference between estimated and true disparity is greater than a given threshold τ:

$$\text{PBP} = \frac{\sum_i \mathbb{I}\left(|d_i - d_i^*| > \tau\right)}{H \times W} \times 100$$

In the evaluation for this project, a threshold of $\tau = 3$ pixels was used. According to Table I, block matching with sub-pixel estimation produces the best results on the two tested image pairs. Nonetheless, it is important to note that the evaluation of the different algorithms during this project is by far not exhaustive. A larger dataset and a more systematic evaluation of the disparity map quality are needed to make justified claims about the algorithm's performance.

Overall, the generated point clouds appear to be a reasonable reconstruction of the scene. Visually, the SGM algorithm with sub-pixel estimation seems to produce the best results with only a few outliers. The final reconstructed point clouds for the Teddy scene are shown in Figure 7.

The last metric that was evaluated during the project was the processing time. Processing time can be crucial for different applications, such as real-time navigation. The processing time of the stereo matching algorithms was measured and is reported in Table II. All benchmarks have been performed on a MacBook Air with an M1 chip. The processing time for sub-pixel estimation variants is omitted, because sub-pixel estimation does not significantly affect the processing time. The data in Table II clearly shows that the OpenCV implementation is the fastest. However, implementing the block matching algorithm in C instead of Python already resulted in a speedup of 15. Parallelizing the outermost loop with OpenMP further improved the processing time by a factor of 2. Since minimizing the processing time was not the objective of this project, further optimizations were not investigated. However, one can be optimistic that advanced performance optimizations would further close the gap between the C implementation and OpenCV's block matching.

VI. CONCLUSION

This project successfully implemented a pipeline for generating point clouds from stereo images. To achieve this goal, two different algorithms with sub-pixel estimation were implemented. Finally, the implemented algorithms were compared and evaluated both quantitatively and qualitatively. The block matching algorithm was implemented in Python and C to optimize processing time, leveraging parallelization capabilities.

Future work can be pursued in various directions. Additional algorithms could be implemented, and neural network approaches could be integrated into the project. A broader benchmarking of the algorithm's performance would be beneficial, especially for challenging scenes. Furthermore, optimizations such as left-right consistency checks could be implemented to improve the handling of occluded image regions. Lastly, further improvements in processing time could be achieved by leveraging different hardware capabilities such as vectorization or GPU computing. On the software side, optimizations such as computing the integral image for cost computation could lead to improved processing times.

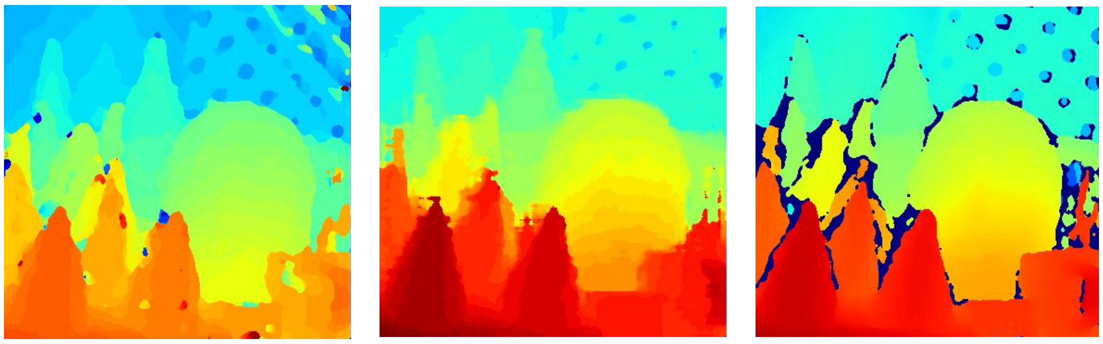

Fig. 5. *Left*: Block matching algorithm without sub-pixel estimation. *Middle*: SGM algorithm without sub-pixel estimation. *Right*: OpenCV Block matching implementation.

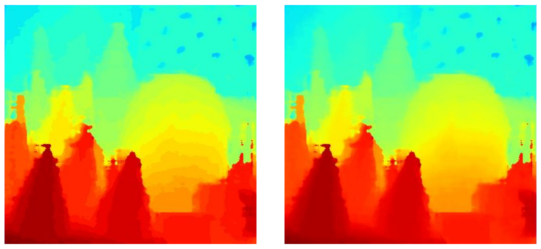

Fig. 6. *Left*: Disparity map from the SGM algorithm without sub-pixel estimation. *Right*: Disparity map from the SGM algorithm with sub-pixel estimation.

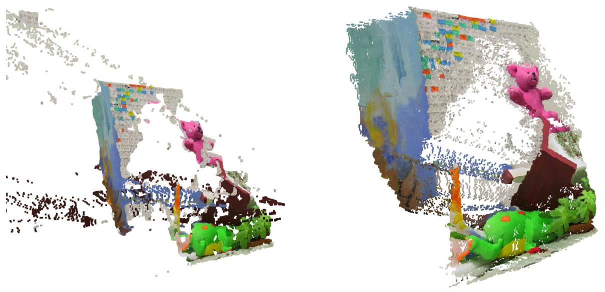

Fig. 7. *Left*: Reconstructed point cloud using the block matching algorithm. Outliers in the point cloud are clearly visible. *Right*: Reconstructed point cloud using the SGM algorithm. The SGM point cloud has significantly fewer outliers.

References

[1] John L Barron, David J Fleet, and Steven S Beauchemin. "Performance of optical flow techniques". In: *International journal of computer vision* 12 (1994), pp. 43–77.

[2] James R. Bergen et al. "Hierarchical model-based motion estimation". In: *Computer Vision — ECCV'92*. Ed. by G. Sandini. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 237–252. ISBN: 978-3-540-47069-4.

[3] Yuri Boykov, Olga Veksler, and Ramin Zabih. "Fast approximate energy minimization via graph cuts". In: *IEEE Transactions on pattern analysis and machine intelligence* 23.11 (2001), pp. 1222–1239.

[4] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[5] Rohit Chandra et al. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[6] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[7] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[8] Heiko Hirschmuller. "Stereo processing by semiglobal matching and mutual information". In: *IEEE Transactions on pattern analysis and machine intelligence* 30.2 (2007), pp. 328–341.

[9] Heiko Hirschmuller and Daniel Scharstein. "Evaluation of cost functions for stereo matching". In: *2007 IEEE conference on computer vision and pattern recognition*. IEEE. 2007, pp. 1–8.

[10] Pierluigi Zama Ramirez et al. *Booster: a Benchmark for Depth from Images of Specular and Transparent Surfaces*. 2024. arXiv: 2301.08245 [cs.CV]. URL: https://arxiv.org/abs/2301.08245.

[11] Daniel Scharstein and Chris Pal. "Learning conditional random fields for stereo". In: *2007 IEEE conference on computer vision and pattern recognition*. IEEE. 2007, pp. 1–8.

[12] Daniel Scharstein and Richard Szeliski. "High-accuracy stereo depth maps using structured light". In: *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.* Vol. 1. IEEE. 2003, pp. I–I.

[13] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[14] Hai Tao, Harpreet S Sawhney, and Rakesh Kumar. "A global matching framework for stereo computation". In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 1. IEEE. 2001, pp. 532–539.

[15] Fabio Tosi, Luca Bartolomei, and Matteo Poggi. *A Survey on Deep Stereo Matching in the Twenties*. 2024.

arXiv: 2407.07816 [cs.CV]. URL: https://arxiv.org/abs/2407.07816.

[16] Bowen Wen et al. *FoundationStereo: Zero-Shot Stereo Matching*. 2025. arXiv: 2501.09898 [cs.CV]. URL: https://arxiv.org/abs/2501.09898.

[17] Lihe Yang et al. *Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data*. 2024. arXiv: 2401.10891 [cs.CV]. URL: https://arxiv.org/abs/2401.10891.

[18] Ramin Zabih and John Woodfill. “Non-parametric local transforms for computing visual correspondence”. In: *Computer Vision—ECCV’94: Third European Conference on Computer Vision Stockholm, Sweden, May 2–6 1994 Proceedings, Volume II 3*. Springer. 1994, pp. 151–158.

[19] Jure Žbontar and Yann LeCun. *Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches*. 2016. arXiv: 1510.05970 [cs.CV]. URL: https://arxiv.org/abs/1510.05970.