https://doi.org/10.2352/ISSN.2470-1173.2017.13.IPAS-210


# Camera-to-Model Back-Raycasting for Extraction of RGB-D Images from Pointclouds

***Hani Javan Hemmat, Egor Bondarev, and Peter H.N. de With***
***Eindhoven University of Technology***
***Eindhoven, The Netherlands***

## Abstract

*Conventional raycasting methods extract 2D-images from pointclouds in two main steps. The pointcloud is voxelized and then, rays are casted from a virtual-camera center towards the model. The value for each pixel in the resulting image is calculated based on the closest non-empty voxel intersected with the corresponding ray. Both voxelizing and such raycasting limit the quality (resolution) of the extracted image and impose high memory demands. In this paper, we propose an alternative back-raycasting method, where rays are casted from the model towards the virtual-camera center and intersecting an image plane. This does not require any voxel grid to be generated. Moreover, this method allows to obtain images with any required resolution with all the points involved. Besides this, a neighbours-consistency technique is introduced to enhance the resulting image quality. The proposed method has been evaluated based on several criteria and for various resolutions. Evaluation results show that the proposed method compared to the conventional approach executes upto 49 times faster and improves PSNR and SSIM metrics for the resulting images by 26% and 12%, respectively. This improvement is beneficial for such domains as feature matching, edge detection, OCR and calibration. To enable researchers generating the same results and extend this work, the dataset and implementation codes are publicly available [1].*

## Introduction

Raycasting [2] plays a prominent role in structure of several algorithms including extracting images from pointclouds, hidden points/surface removal, non-recursive ray-tracing and volume rendering algorithms for various domains such as computer graphics, computational geometry, volumetric image processing, health-care and robotics [3, 4, 5, 6, 7, 8, 9]. In this paper, we focus on raycasting for pointclouds with the aim of extracting RGB-D images from 3D data for a specific point of view.

In general, raycasting algorithms trace rays from a virtual-camera center towards the 3D-models to generate a 2D-image. In resulting 2D-images, a ray is casted for each pixel, where the pixel value is determined based on the closest point of the model intersecting with the corresponding ray. Applying raycasting to pointclouds, the most challenging part is to intersect a ray with points of the 3D-model in the 3D space. Since pointclouds are discrete structures of disconnected points, rays may not intersect any points while passing through the pointcloud. Therefore, the conventional algorithms voxelize the pointclouds prior to the raycasting process [10]. A voxelized pointcloud is a 3D array of voxels (volume-pixels) representing the model data. The resolution of a voxelized model is determined based on its voxel size. A drawback of voxelized models compared to pointclouds is their high demands on memory. To overcome this problem, several spatial partitioning techniques have been introduced (e.g. octrees and kd-trees) to deliver the performance of the pure voxelized model, but with a significantly less memory requirement [11, 12, 13]. Figure 1 shows the a conceptual 3-D model and its corresponding voxelized and optimized models.

Conventional raycasting algorithms perform two main steps to extract 2D-images from pointclouds: (a) voxelizing the target pointcloud and (b) raycasting from virtual-camera center to the voxelized model (Figure 2). Both steps decrease the quality of the extracted 2D-image. First, voxelizing process manipulates the arrangement of the points by changing the density and distance between them, which degrades the 2D-image quality especially for the points located closer to the camera (influenced by the value of $V$ in Figure 2). Second, since rays are casting from the camera towards the model, they scatter while becoming further from the camera. This decreases the intersection chance for the voxels located further from the virtual camera, accordingly (influenced by the value of $a_r$ in Figure 2). Although decreasing the voxel size and the angel between rays can partly compensate the above-mentioned drawbacks, they require larger memory and computational resources.

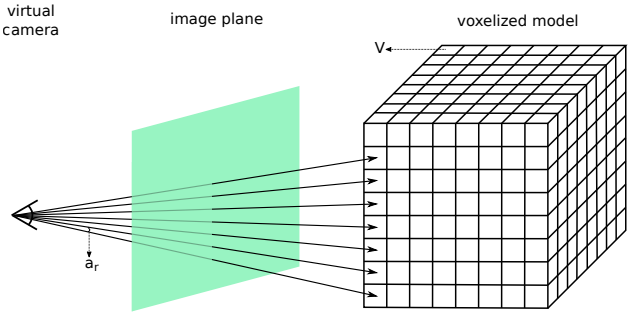


Figure 2: Conventional raycasting method ($a_r$ and $V$ are the angel between rays and voxel size, respectively).

In this paper, we propose an alternative for the conventional raycasting method to extract RGB-D images from colored pointclouds. Two main advantages of the proposed method are (1) its ability to produce higher-quality images and (2) its efficient CPU and memory usage, when compared to the conventional raycasting method. The next section introduces the proposed method in detail. Then, evaluation results are shown and discussed. And the final section concludes the paper.

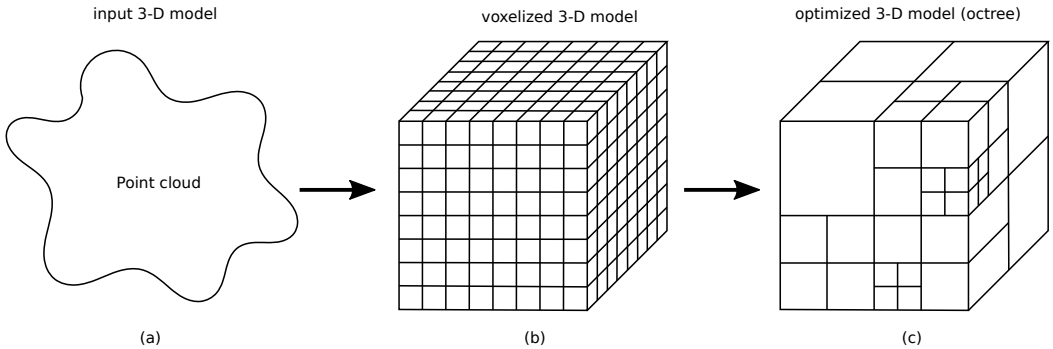

Figure 1: (a) Input 3D-model, (b) voxelized model and (c) corresponding optimized model.

## Method

We propose a Model-to-Camera back-raycasting (M2C) method as an alternative to the conventional Camera-to-Model raycasting (C2M) approach. The M2C method allows to extract higher-quality images in two main steps without any need for the voxelizing process. As demonstrated in Figure 3, the conventional reycasting method performs two main steps: voxelizing and raycasting. However, these main steps are replaced by back-reaycasting and pixelizing steps in the proposed method.

### A. Back-Raycasting

To extract images from the 3-D model, first, the rays are casted from the 3D-model towards the virtual-camera center to intersect the resolution-less image plane. This image is located at the focal length of the virtual camera and preserves the exact intersection point of each ray. In contrast to digital images, there is no resolution defined for this continuous image. Figure 4 illustrates the M2C method, where casting rays from the pointcloud to the camera center are intersecting the resolution-less image.

Each intersection point on the resolution-less image records two elements of data per ray: (1) coordinates of the intersection in the range of $[-1,1]$ per each axis and (2) the distance from the ray origin in the pointcloud to the camera. This information are processed in the pixelizing step to generate the corresponding color and depth images.

### B. Pixelizing

Extraction of images from the 3-D model continues with the second step of pixelizing, where the 2D-image with the desired resolution is generated from the obtained resolution-less image. In pixelizing technique, each pixel value is calculated by integrating data of multiple intersection points. Pixelizing process is performed in two phases. First, we apply the desired resolution as a 2D grid on the resolution-less image to determine each grid cell and the intersection points located inside it. Then, we determine the grid pixel values as follows. If a grid cell contains no intersection points, the corresponding pixel is determined as an empty pixel. The pixel value is straightforwardly calculated, if a grid cell contains a single intersection point. For a grid cell containing multiple intersection points, the pixel value is inherited from the value of the intersection point with the closest distance. Figure 5 shows the phases of the pixelizing process.

### C. Neighbours Consistency

Regardless of deploying C2M or M2C methods, the extracted images may contain undesired empty pixels. Moreover, both methods may result in sharp borders between neighbouring pixels in the extracted image, depending on the virtual-camera pose in the scene (Figure 6). To overcome the above-mentioned challenges, we introduce a neighbours-consistency technique. The aim is to perform a weighted color-fusion in order to achieve a smoother image with the least possible amount of empty pixels. The following equation updates each pixel value by combining its own value with its neighbouring values:

$$Q_i = \frac{W_d \times P_i + \sum_p (W_p \times N_p) + \sum_c (W_c \times N_c)}{W_d + \sum_p W_p + \sum_c W_c} \,. \tag{1}$$

The new pixel value, $Q_i$, is calculated based on three elements: (1) the old pixel value, $P_i$, (2) the non-diagonal neighbours, $N_p$, and (3) the diagonal neighbours, $N_c$. The user-defined weights $W_d$, $W_p$ and $W_c$ indicate the amount of influence for the old value, non-diagonal and diagonal neighbours, respectively ($W_d = 80$, $W_p = 4$ and $W_c = 1$, for our experiments). Figure 7 illustrates the neighbours-consistency technique applied to a generic pixel of the extracted image.

The following section discusses the evaluation results, where the proposed M2C method is compared to the conventional C2M method.

## Evaluation Results

**Implementation and dataset collection:** the proposed M2C method is implemented in C++ and all the codes and dataset collection are publicly available under an open-source license [1]. We have utilized OpenCV [15] and PointCloud Library [14] for the standard implementation of some well-known algorithms to generate evaluation results. The CloudCompare [18] tool has been used for the pointclouds comparison. The experimental results have been obtained utilizing a workstation with a Xeon(R) W3550 @3.07GHz CPU and 20 GB of RAM. The dataset collection consists of 40 colored pointclouds obtained from the FARO Focus$^{3D}$ laser scanner and the corresponding groundtruth images.

**Criteria:** the experimental results are obtained based on three main metrics: performance, color-image quality and depth-image quality. For performance, we compare the M2C and C2M methods in terms of their execution time. We evaluate the color-image quality by applying the following four well-known algorithms to the extracted images: edge detection, cam-

era calibration, feature detection and Optical Character Recognition (OCR). And finally, in order to evaluate the depth-image quality, we compare the pointclouds generated based on the extracted depth images to the groundtruth laser-scanner pointcloud. In all cases, we evaluate the extracted images for four resolution formats: VGA (=640×480), VGA×2 (=1280×960), VGA×3 (=1920×1440) and VGA×4 (=2560×1920).

### A. Performance

Table 1 and Figure 8 show the average execution time of the C2M and M2C methods for the four resolutions. The most important finding is that the execution time for M2C is resolution-independent. In contrast to that, the execution time for C2M grows exponentially by increasing resolution. The reason is that M2C depends only on the number of points in a pointcloud and not the image resolution. However, the image resolution influences on the performance of the C2M method through its voxelizing phase. On the average, M2C and C2M execute in 2.15 s and 47.27 s, respectively. This means that for an average resolution, M2C runs 22 times faster than C2M.

Table 1: Execution time for the C2M and M2C methods for various resolutions.

| Resolution | C2M (s) | M2C (s) |
|---|---|---|
| VGA×1 | 2.50 | 1.80 |
| VGA×2 | 12.15 | 1.93 |
| VGA×3 | 44.79 | 2.21 |
| VGA×4 | 129.62 | 2.65 |


Figure 8: Execution time for the C2M and M2C methods for various resolutions.

### B. Color-Image Quality

Figure 9 (b-d) compares the extracted images by the M2C and C2M methods for the four criteria: quality of edges, calibration results, quality of features, and OCR results, respectively. In the following, we discuss the evaluation results for each criterion in detail.

#### B.1. Edge Detection

We have applied the well-known Canny edge-detection algorithm to the images obtained by C2M and M2C. The resulting edges have been compared to the corresponding groundtruth by computing the Peak Signal-to-Noise Ratio (PSNR) and mean Structural SIMilarity (mSSIM) metrics. Table 2 and Figure 10 show the edge-detection results for C2M and M2C. For all four resolutions, both metrics indicate a consistent pattern, however, M2C outperforms C2M in terms of providing higher-quality edges. For the average resolution, C2M yields a PSNR of 17.3 dB and 89.3% of similarity with the mSSIM metric. M2C provides higher-quality edges by yielding a PSNR of 20.5 dB and 94.8% of similarity with mSSIM, when compared to C2M. This shows 15.2% and 5.8% of improvement in terms of preserving edges for M2C, based on the PSNR and mSSIM metrics, respectively.

Table 2: Comparing similarity between detected edges and the groundtruth for the images extracted by the C2M and M2C methods in terms of PSNR and mSSIM metrics for various resolutions.

| Resolution | PSNR (dB) | | mSSIM (%) | |
|---|---|---|---|---|
| | C2M | M2C | C2M | M2C |
| VGA×1 | 17.2 | 19.5 | 87.7 | 92.9 |
| VGA×2 | 18.8 | 20.6 | 92.4 | 94.7 |
| VGA×3 | 17.3 | 21.8 | 92.3 | 96.2 |
| VGA×4 | 16.1 | 19.9 | 84.7 | 95.2 |

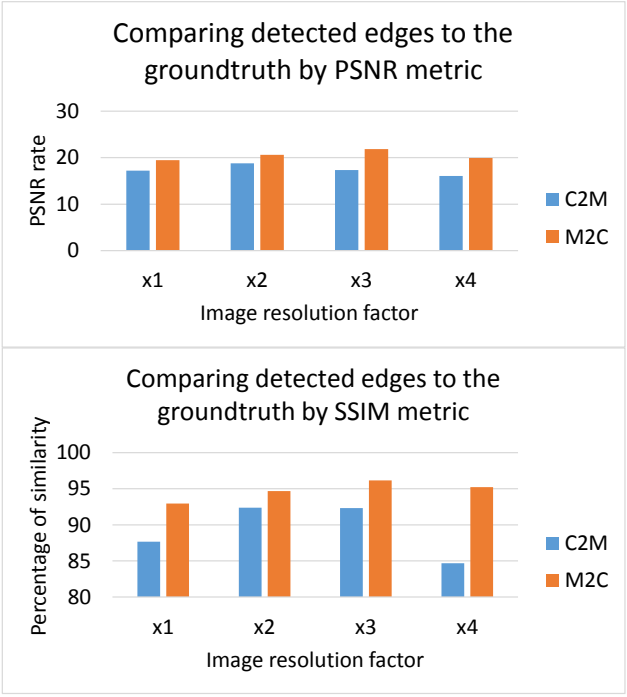


Figure 10: Comparing similarity between detected edges and the groundtruth for the images extracted by the C2M and M2C methods in terms of PSNR (top) and mSSIM (bottom) metrics for various resolutions.

#### B.2. Camera Calibration

We have evaluated the M2C and C2M methods by applying a standard camera-calibration algorithm to the images, which are extracted from the pointclouds containing checker-boards 3-D data. The resulting camera intrinsics have been compared to the virtual camera intrinsics fed to both C2M and M2C methods to generate the color images. As shown in Table 3 and Figure 11, the average calibration error for the images extracted by M2C and C2M methods are 0.36% and 11.7%, respectively. This means that M2C performs 32 times more accurately than C2M. Besides this, M2C shows a consistent behaviour independent of resolution, when compared to the resolution-dependent behaviour of C2M.

Table 3: Percentage of the average error for applying camera calibration algorithms to the images extracted by the C2M and M2C methods for various resolutions.

| Resolution | C2M (%) | M2C (%) |
|---|---|---|
| VGA×1 | 3.76 | 0.05 |
| VGA×2 | 11.03 | 0.99 |
| VGA×3 | 18.93 | 0.16 |
| VGA×4 | 13.03 | 0.26 |


Figure 11: Percentage of the average error for applying camera calibration algorithms to the images extracted by the C2M and M2C methods for various resolutions.

### *B.3. Feature Detection*

We have applied the well-known SIFT feature detector algorithm to the extracted images and the corresponding real images of the same scenes. Then, the detected features are categorized in two groups: (1) extracted images together, where we have matched resulting images together and (2) between extracted and real images, where we matched the resulting images with the real images taken by an actual camera. The average number of inlier matched features between each pair of images is used as a metric to evaluate the color-image quality. Table 4 and Figure 12 show the evaluation results for the C2M and M2C methods.

Table 4: Average number of the matched features between the extracted (ext.) images by the C2M and M2C methods in two groups: inside the extracted images and between the extracted and real images for various resolutions.

| Resolution | ext. vs ext. | | ext. vs real | |
|---|---|---|---|---|
| | C2M | M2C | C2M | M2C |
| VGA×1 | 151 | 256 | 2 | 7 |
| VGA×2 | 99 | 200 | 42 | 53 |
| VGA×3 | 138 | 216 | 62 | 105 |
| VGA×4 | 141 | 209 | 74 | 85 |

M2C represents a more steady behaviour compared to C2M, in terms of matching features in the extracted images. However, both methods perform differently for each resolution, when matching features between extracted and real images. This is due to the fixed resolution of the real images. On the average, C2M and M2C lead to 132 and 220 (67% of improvement) matched features in the extracted images, respectively. And for matching features between the extracted and real images, C2M and M2C lead to 132 and 220 (67% of improvement) matched features, respectively. These values are 45 and 62 (40% of improvement) matched features between the extracted and real images.

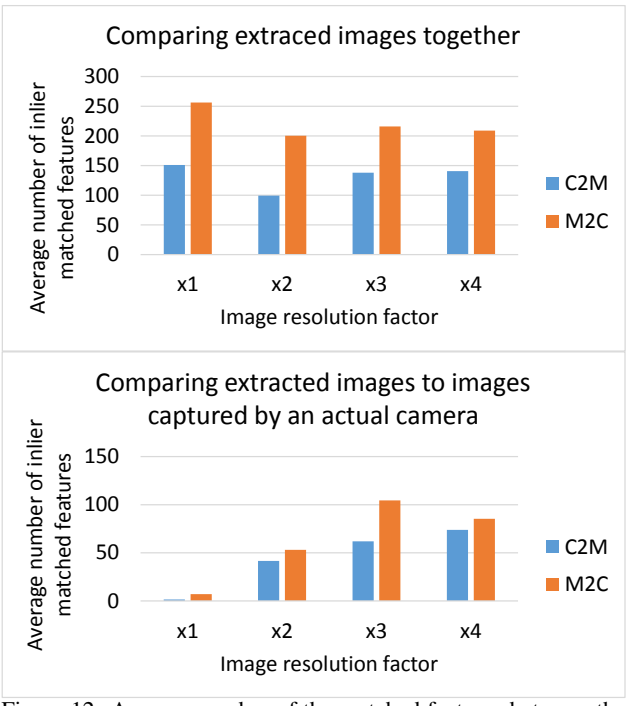


Figure 12: Average number of the matched features between the extracted images by the C2M and M2C methods in two groups: inside the extracted images (top) and between the extracted and real images (bottom) for various resolutions.

### *B.4. OCR algorithms*

We have generated a laser-scanner dataset containing text images of various font types and sizes. Then, two well-known OCR algorithms, Tesseart [16] and OnlineOCR [17], have been applied to the extracted images. As a metric for color-image quality, we have reported the percentage of correctly recognized characters by each algorithm per method and for various resolutions (Table 5 and Figure 13). As expected, both algorithms indicate resolution-dependent results with a consistent pattern for the extracted images by the C2M and M2C methods.

Table 5: Average percentage of the recognized characters in the extracted images by C2M and M2C for various algorithms and resolutions.

| Resolution | Tesseart (%) | | OnlineOCR (%) | |
|---|---|---|---|---|
| | C2M | M2C | C2M | M2C |
| VGA×1 | 39 | 44 | 39 | 46 |
| VGA×2 | 53 | 69 | 73 | 75 |
| VGA×3 | 63 | 72 | 78 | 87 |
| VGA×4 | 71 | 77 | 80 | 88 |

The Tesseart algorithm can averagely recognize 56% and 65% of characters in the images extracted by the C2M and M2C methods, respectively. The OnlineOCR recognition rate is 68% and 74%, when applied to the the images extracted by the C2M and M2C methods, respectively. These show that the M2C method leads to 16% and 10% more recognized characters by the Tesseart and OnlineOCR algorithms, when compared to the C2M method.

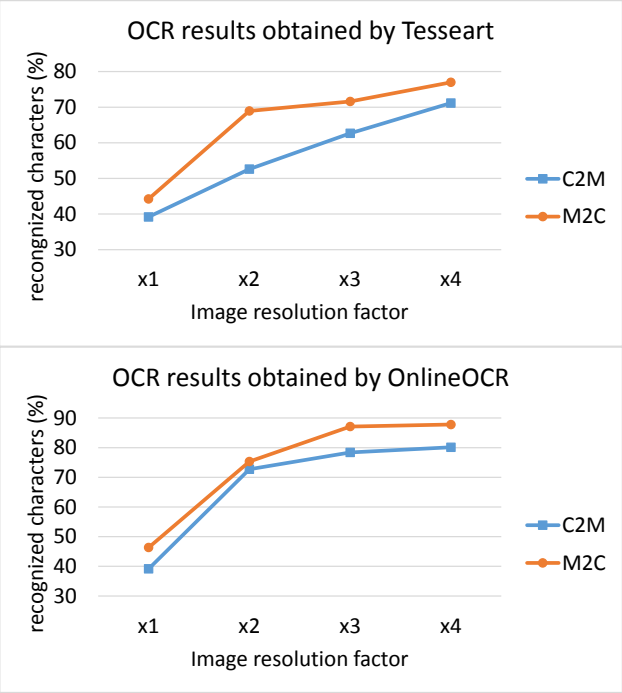


Figure 13: Average percentage of the recognized characters in the extracted images by C2M and M2C for various resolutions: results obtained by the (top) Tesseart and (bottom) OnlineOCR algorithms.

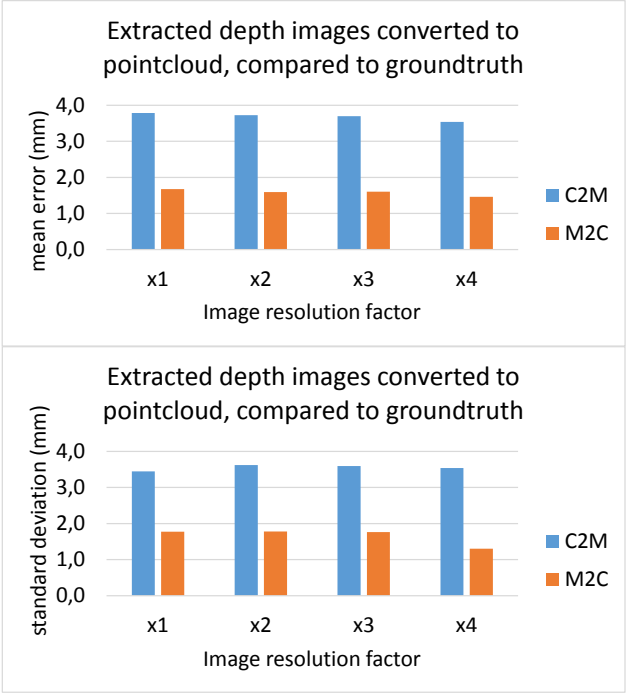


Figure 14: Average point-to-point error of the pointclouds generated based on the images extracted by the C2M and M2C algorithms, compared to the groundtruth: (top) mean error, (bottom) standard deviation.

### C. Depth-Image Quality

To evaluate depth-image quality, first, we have extracted the depth images from the laser-scanner pointclouds, based on the C2M and M2C methods. Then, the extracted images have been converted again to the pointclouds, according to the virtual camera intrinsics used for both methods. Finally, the resulting pointclouds have been compared to the corresponding groundtruth laser-scanner pointclouds. We have utilized the average point-to-point distance between pointclouds (obtained via CloudCompare [18]) as the error metric. Table 6 and Figure 14 demonstrate these comparison results of the C2M and M2C methods in terms of mean error and standard deviation.

Table 6: Average point-to-point error of the pointclouds generated based on the images extracted by the C2M and M2C algorithms, compared to the groundtruth.

| Resolution | mean error (mm) | | std. dev. (mm) | |
|---|---|---|---|---|
| | C2M | M2C | C2M | M2C |
| VGA×1 | 3.79 | 1.68 | 3.45 | 1.77 |
| VGA×2 | 3.72 | 1.59 | 3.62 | 1.78 |
| VGA×3 | 3.70 | 1.60 | 3.59 | 1.76 |
| VGA×4 | 3.54 | 1.46 | 3.54 | 1.30 |

Both C2M and M2C methods produce consistent results for all four resolutions. The extracted depth images obtained by the proposed M2C method contain the average error of 1.58 mm, which is a 57% of improvement compared to the C2M method with 3.69 mm of the average error. The same pattern appears for the standard deviation metric, where C2M and M2C result in the average deviation of 3.35 mm and 1.65 mm (53% less error), respectively.

## Conclusion

We have proposed a Model-to-Camera back-raycasting (M2C) method as an alternative to the conventional Camera-to-Model raycasting (C2M) approach. This method allows to extract higher-quality images in two main steps without any need for the voxelizing process, which is required for the conventional C2M approaches.

The proposed method consists of two main steps to extract color or depth images from the input pointclouds: back-raycasting and pixelizing. First, at the back-raycasting step, we cast a ray per each point in the pointcloud towards a resolution-less image located at the focal length of the virtual camera. This resolution-less image contains both color and depth data per point. Second, at the pixelizing step, we generate a pixel grid with a user-defined resolution to extract the color and depth images from the resulting resolution-less image. Besides these two main steps, we also apply a neighbours-consistency technique on the extracted images obtained from both C2M and M2C methods. This extra step improves the image quality in terms of filling holes and smoother transition between pixels.

We have evaluated the proposed methods in terms of performance, color image and depth-image qualities. To evaluate the color-image quality, we have utilized four well-known algorithms including edge detection, camera calibration, feature matching and OCR algorithms. The evaluation results show that the proposed M2C method outperforms the conventional C2M method for all the metrics and resolutions. In terms of execution time, M2C runs upto 49 times faster than C2M (22 times faster on the

average). Besides this, the execution time of M2C is resolution-independent, compared to the resolution-dependent behaviour of C2M. The edges generated by a Canny detector on M2C images are 15% closer to the groundtruth, compared to the detector results running on C2M images.

Applying camera calibration algorithms on the extracted images shows that, on the average, M2C provides images resulting in 32% less errors, when compared to the images extracted by C2M. The images generated by M2C lead to 67% more matched features compared to the images extracted by C2M, when utilizing SIFT algorithm. The OCR algorithms can averagely recognize 16% more characters in the images extracted by M2C compared to the images generated by C2M. And finally, depth images generated by M2C contain 57% less point-to-point error, when compared to the depth images extracted by C2M.

As a future work, we are investigating an efficient implementation of the proposed method to improve the execution time even more. To enable researchers generating the same results and extend this work, the dataset and implementation codes are publicly available under an open-source license [1].

## References


[1] M2C-related codes, docs and dataset, `https://gitlab.com/HaniJH/M2C/tree/master` Accessed 14-January-2017.

[2] Scott D. Roth, Ray Casting for Modeling Solids, journal of CGIP, Vol. 18, no. 2, 1982, pp. 109–144.

[3] Ellis, J. L. and Kedem, G. and Lyerly, T. C. and Thielman, D. G. and Marisa, R. J. and Menon, J. P. and Voelcker, H. B., The Ray Casting Engine and Ray Representatives, ACM, New York, NY, USA, 1991, pp. 255–267.

[4] Sutherland, Ivan E. and Sproull, Robert F. and Schumacker, Robert A., A Characterization of Ten Hidden-Surface Algorithms, ACM, New York, NY, USA, 1974, pp. 1–55.

[5] Katz, Sagi and A. Tal, Ayellet and Basri, Ronen, Direct Visibility of Point Sets, ACM SIGGRAPH, Vol. 26, no. 3, 2007, doi = 10.1145/1275808.1276407.

[6] Katz, Sagi and A. Tal, Ayellet and Basri, Ronen, Improving the Visual Comprehension of Point Sets, IEEE CVPR, 2013, pp, 121–128, doi=10.1109/CVPR.2013.23.

[7] Alsadik, B. and Gerke, M. and Vosselman, G., Visibility analysis of point cloud in close range photogrammetry, journal of ISPRS, 2014, pp, 9–16, doi=10.5194/isprsannals-II-5-9-2014.

[8] Woop, Sven and Schmittler, Jörg and Slusallek, Philipp, RPU: A Programmable Ray Processing Unit for Realtime Ray Tracing, ACM Trans. Graph., Vol. 24, no. 3, 2005, pp. 434–444.

[9] Solomon Boulos, Notes on Efficient Ray Tracing, ACM SIGGRAPH, 2005, doi = 10.1145/1198555.1198749.

[10] Samuli Laine and Tero Karras, Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation, NVIDIA Technical Report, 2010, NVR-2010-001.

[11] Schnabel, Ruwen and Klein, Reinhard, Octree-based Point-cloud Compression, IEEE VGTC, 2006, pp. 111–121, doi = 10.2312/SPBG/SPBG06/111-120.

[12] Huang, Yan and Peng, Jingliang and Kuo, C.-C. Jay and Gopi, M., Octree-based Progressive Geometry Coding of Point Clouds, IEEE VGTC, 2006, pp. 103–110, doi = 10.2312/SPBG/SPBG06/103-110.

[13] Sim, Jae-Young and Lee, Sang-Uk and Kim, Chang-Su, Construction of regular 3D point clouds using octree partitioning and resampling, ISCAS, Vol. 2, 2005, pp. 956-959, doi = 10.2312/SPBG/SPBG06/103-110.

[14] R. Rusu, S. Cousins, 3D is here: Point cloud library (PCL), IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 1–4, doi:10.1109/ICRA.2011.5980567.

[15] G. Bradski, OpenCV: an open source computer vision library, Dr. Dobbs Journal of Software Tools.

[16] Smith, R., An Overview of the Tesseract OCR Engine, IEEE Computer Society, ICDAR'07, Vol. 2, pp. 629–633, 2007.

[17] Free Online OCR Services, `http://www.onlineocr.net/`, Accessed 14-January-2017.

[18] T. P. EDF R&D, CloudCompare (version 2.6) [GPL software], Retrieved from `http://www.cloudcompare.org/`, (2016)


## Author Biography

**Hani Javan Hemmat** *received his B.Sc. degree in Computer Engineering from Amirkabir University of Technology (Tehran Polytechnic) in 2002. In 2006, he received the M.Sc. degree in Computer Architecture from the Computer Engineering department of the Sharif University of Technology (Tehran, Iran). His research interests include 3D reconstruction, robotics, parallel processing, hardware-software co-design, and design of hardware/software architectures for real-time implementation. Since 2012, He researches on multi-modal fusion and 3D reconstruction at Technical University of Eindhoven.*

**Egor Bondarev** *received his MSc degree in robotics and informatics from the State Polytechnic University, Belarus Republic, in 1997. In 2009 he has obtained his PhD degree in Computer Science at Eindhoven University of Technology (TU/e), The Netherlands in the domain of real-time systems with a focus on performance predictions of component-based systems on multiprocessor architectures. Currently, he is an Assistant Professor at the Video Coding Architectures group, TU/e, focusing on such research areas as multi-modal sensor fusion, photorealistic 3D reconstruction of environments and SLAM systems. Egor Bondarev is involved in several European research projects and currently he is a TU/e project leader in the PANORAMA and OMECA projects, both addressing challenges of ultra-high definition UHD and 3D image processing.*

**Peter H. N. de With** *graduated in electrical engineering (MSc., ir.) from Eindhoven University of Technology and received his PhD degree from University of Technology Delft, The Netherlands. From 1984 to 1997, he worked for Philips Research Eindhoven, where he worked on video compression and chaired a cluster for programmable TV architectures as senior TV Systems Architect. From 1997 to 2000, he was full professor at the University of Mannheim, Germany, Computer Engineering, and chair of Digital Circuitry and Simulation. From 2000 to 2007, he was with LogicaCMG in Eindhoven as a principal consultant Technical SW and distinguished business consultant. He was also part-time professor at the University of Technology Eindhoven, heading the chair on Video Coding and Architectures. In the period 2008 to 2010, he was VP Video (Analysis) Technology at CycloMedia Technology. Since 2011, he has been an assigned full professor at Eindhoven University of Technology and appointed scientific director Care and Cure Technology and theme leader Smart Diagnosis in the University Health program. He is a national and international expert in video surveillance for safety and security and has been involved in multiple EU projects on video analysis, featuring object and behavior recognition, and also surveillance projects with the Harbor of Rotterdam, Dutch Defense, Bosch Security Systems, TKH-Security, etc. He is the (co-) recipient of multiple papers awards of the IEEE CES and VCIP papers and the EURASIP Signal Processing journal paper award.*

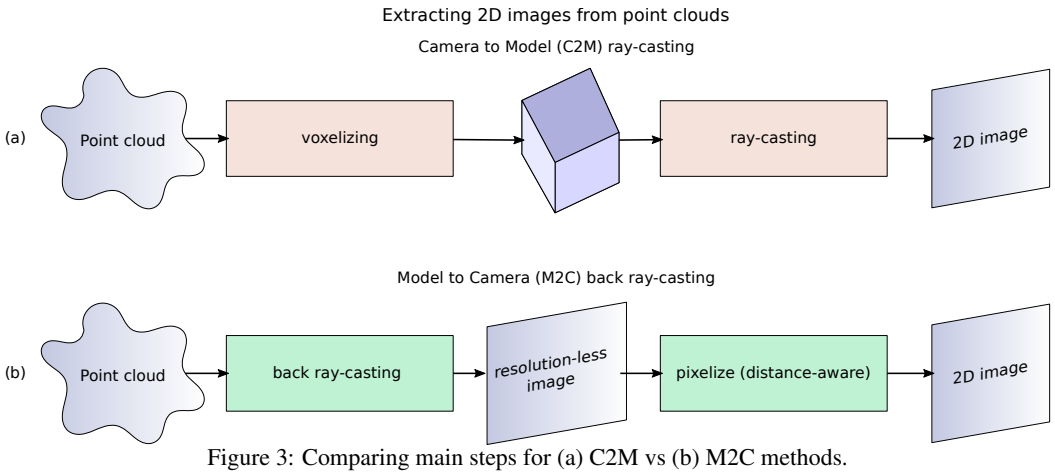


Figure 3: Comparing main steps for (a) C2M vs (b) M2C methods.

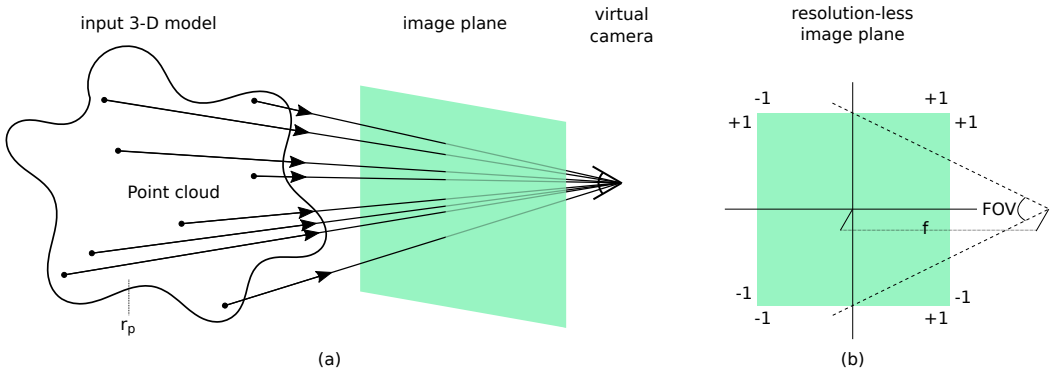


Figure 4: M2C method: (a) back-raycasting towards a (b) resolution-less image.

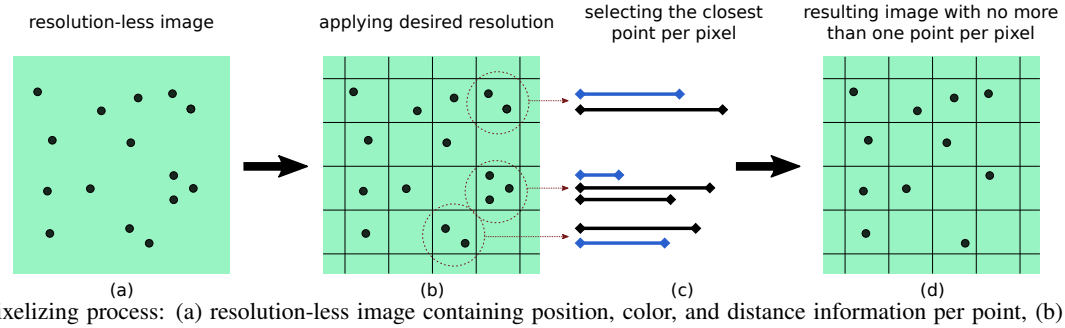


Figure 5: Pixelizing process: (a) resolution-less image containing position, color, and distance information per point, (b) user-defined pixel grid, (c) data fusion, and (d) the resulting color or depth image.

Figure 6: Before (left) and after (right) applying the neighbours-consistency technique.

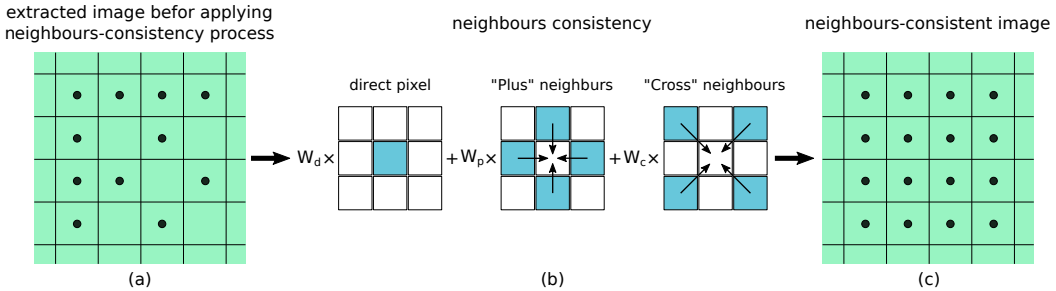


Figure 7: Neighbours-consistency technique: (a) the input image, (b) the effect of neighbours on each target pixel, and (c) the resulting image.

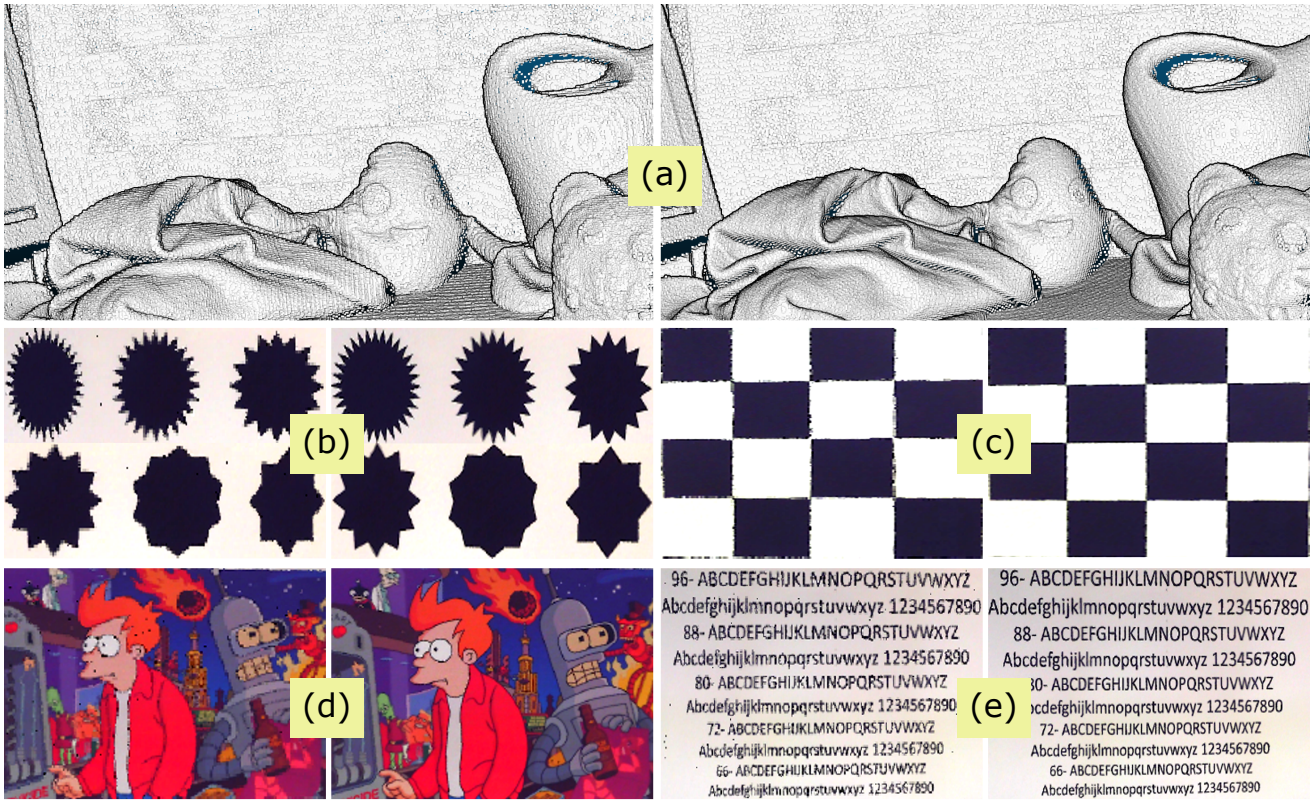


Figure 9: Comparing the extracted images by the C2M (left in each sub-image) vs M2C (right in each sub-image) methods: (a) depth images, (b) color images used for edge detection, (c) checkerboard images exploited for camera calibration, (d) color images utilized for feature matching and (e) color images used for the OCR algorithms.