High quality and efficient direct rendering of massive real-world point clouds

H. Bouchiba[1], R. Groscot[1], J-E. Deschaud[1] and F. Goulette[1]

[1] MINES ParisTech, PSL Research University, CAOR - Centre de robotique, 60 Bd St Michel 75006 Paris, France

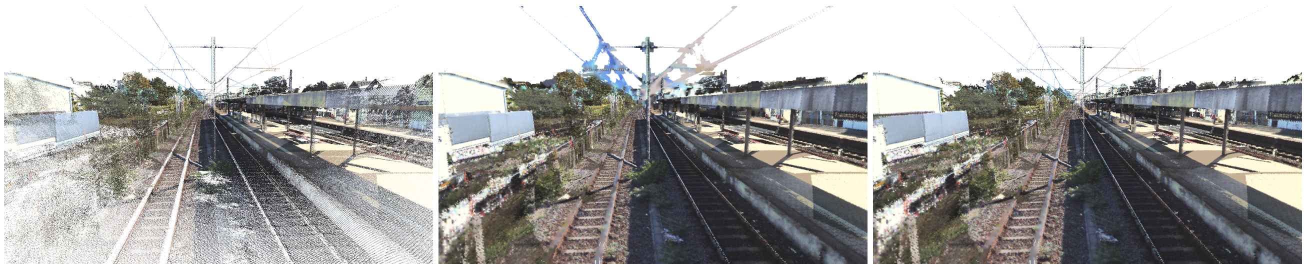

Figure 1: ***(a)*** *Rendering with one pixel per point.* ***(b)*** *Rendering with [PGA11] algorithm.* ***(c)*** *Our new rendering pipeline*

Abstract

We present a novel real-time screen-space rendering algorithm for real-world 3D scanned datasets. Our method takes advantage of the pull phase of a pull-push pyramidal filling algorithm in order to feed a hidden point removal operator. The push phase is then used to fill the final framebuffer. We demonstrate on a real-world complex dataset that our method produces better visual results and is more efficient comparing to state of the art algorithms.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Bitmap and framebuffer operations

1. Introduction

Today's 3D scanners produce massive and detailed 3D point cloud datasets. However as points do not embed any topological information they produce rendering artifacts. Figure 1(a) shows that some parts of the model that are supposed to be hidden can be seen by transparency over it, and that the feeling of a surface vanishes near the camera due to perspective. We tackle in this article the problem of getting rid of these artifacts.

[BHZK05] proposed a well-known splatting method to render points with normals and radii information. [MKC08] extended this work using screen-space pyramidal algorithm in order to reduce the complexity of the original algorithm. However normals and radii are not directly generated by 3D scanners and computing them for large point clouds is expensive and highly dependent to real-world point cloud artifacts: noise, holes, misalignment and outliers.

[PGA11] use a screen-space pipeline to render unprocessed point clouds based on a visibility operator to remove the non-visible parts of the model. The space between points is then filled by an anisotropic filter. [PJW12] use screen-space nearest neighbor queries in order to compute normals and radii to feed [BHZK05] splatting algorithm in real-time. Screen space rendering methods are only based on raw point clouds, but they do not handle well scenes with high depth differences, and scenes containing linear patches as cables or traffic signs, as it can be seen in Figure 1(b).

We propose in this paper a new real-time screen-space raw point cloud rendering pipeline that overcomes the limitations of above-cited methods, and that is also more efficient.

2. Our method

The first phase of our four-step rendering approach is an optional preprocessing segmentation step. We then perform a pyramidal pull phase which is used to feed an adaptive screen-space visibility algorithm. Finally a filled framebuffer is reconstructed with a pyramidal push phase.

DOI: 10.2312/egp.20171035

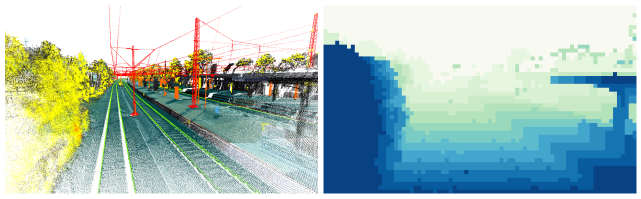

Figure 2: ***(a)*** *Segmented dataset rendered with false colors.* ***(b)*** *Visibility kernel size.*

Step 1 - Linear patches segmentation The only preprocessing step required by our algorithm is a segmentation in order to separate linear patches from the rest of the point cloud. We used [RDG16] method, the results are depicted in Figure 2(a). Only the non-linear parts are processed by the rest of the pipeline. The cables and other linear patches (in red in Figure 2(a)) are rendered with one pixel per point.

Note that this step is not necessary if the amount of linear patches in the dataset is small. It does not add any topological information to the points and is very fast to compute. For our dataset it took 143.5s to segment the linear patches and 612.5s to segment all the classes depicted in Figure 2(a) (not used in this work).

Step 2 - Pyramidal down-sampling: pull phase We perform a pyramidal down-sampling phase on the valid pixels of the framebuffer. Differently from [MKC08] for each 4x4 pixels patch we only keep the nearest from the camera (as if we were rendering to a smaller framebuffer). The coarser level of the pyramid depth buffer is then used for the next phase of the pipeline.

Step 3 - Adaptive visibility operator A visibility operator is used to remove the supposed hidden parts of the point cloud seen by transparency over it. The operator is applied to the whole image (even on the background pixels) in order to detect the holes to be filled by the next stage of the pipeline. We use an adaptive version of [PGA11] visibility operator to better handle large scenes and to reduce in the same time its computational cost. The size of the kernel is no longer constant over the whole image and is taken proportional to inverse of the depth computed on the coarse depth buffer previously obtained, as it can be seen in Figure 2(b).

This way, far from the camera the hidden-point detection is still coherent as it is less dependent to the decreasing of resolution due to perspective. The reduction of kernel size reduces in the same time the mean computational cost of the algorithm.

Step 4 - Pyramidal filling: push phase Pyramidal filling method is then used to reconstruct a filled surface as in [MKC08].

3. Results

We illustrate our work on a 35 million points Mobile Laser Scanning (MLS) dataset of 600 meter long railway. This environment has been chosen as it contains many linear patches. Each point contains only position and color attributes.

We implemented our method in C++ with OpenGL. In order to deal with massive point clouds, we simply divide our dataset in uniformly sized cubes stored in a hash map. In each cube the points are ordered in LODs laid sequentially in memory. All the images have been rendered at 1248 x 768 on a 3.4 Ghz Intel Core i7 with an Nvidia GT 640.

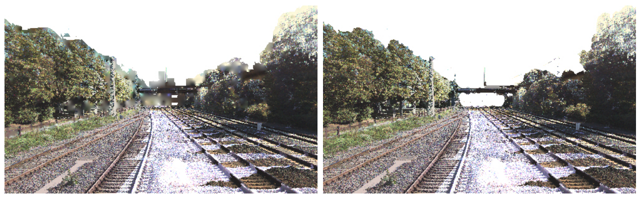

Figure 3: ***(a)*** *[PGA11] visibility with push-pull filling.* ***(b)*** *our adaptive filling. (Linear parts have been removed on both)*

visibility kernel size (px)	[PGA11] (FPS)	our method (FPS)
9×9	17.4	27.4
15×15	9.1	25.6
25×25	4.2	24.4

Table 1: *Mean frame rate of our rendering pipeline comparing with state of the art algorithm.*

Figure 3 shows the robustness of our adaptive visibility operator on the scene without linear elements. The gap between the ground and the bridge at the back of the scene is labeled as a hole with [PGA11] method whereas it is not with ours. The performance gain of our algorithm is illustrated in Table 1. The frame rates are obtained by taking the mean of 3 points of view: far from the scene, at medium altitude and close to the ground.

Acknowledgement The MLS colored railway dataset have been gratefully provided by RIEGL Laser Measurement Systems, `www.riegl.com`.

References

[BHZK05] Botsch M., Hornung A., Zwicker M., Kobbelt L.: High-quality surface splatting on today's gpus. In *Proceedings of the Second Eurographics / IEEE VGTC Conference on Point-Based Graphics* (Aire-la-Ville, Switzerland, Switzerland, 2005), SPBG'05, Eurographics Association, pp. 17–24. 1

[MKC08] Marroquim R., Kraus M., Cavalcanti P. R.: Special section: Point-based graphics: Efficient image reconstruction for point-based and line-based rendering. *Comput. Graph. 32*, 2 (Apr. 2008), 189–203. 1, 2

[PGA11] Pintus R., Gobbetti E., Agus M.: Real-time rendering of massive unstructured raw point clouds using screen-space operators. In *Proceedings of the 12th International Conference on Virtual Reality, Archaeology and Cultural Heritage* (Aire-la-Ville, Switzerland, Switzerland, 2011), VAST'11, Eurographics Association, pp. 105–112. 1, 2

[PJW12] Preiner R., Jeschke S., Wimmer M.: Auto splats: Dynamic point cloud visualization on the gpu. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization* (May 2012), Childs H., Kuhlen T., (Eds.), Eurographics Association 2012, pp. 139–148. 1

[RDG16] Roynard X., Deschaud J. E., Goulette F.: Fast and Robust Segmentation and Classification for Change Detection in Urban Point Clouds. In *ISPRS 2016-XXIII ISPRS Congress* (2016). 2