A POINT-BASED VIRTUAL REALITY SYSTEM FOR SUPPORTING PRODUCT DEVELOPMENT

Hiroki Okamoto
The University of Elector-Communications
Chofu, Tokyo, Japan

Hiroshi Masuda
The University of Elector-Communications
Chofu, Tokyo, Japan

ABSTRACT

In this paper, we discuss methods to efficiently render stereoscopic scenes of large-scale point-clouds on inexpensive VR systems. Recently, terrestrial laser scanners are significantly improved, and they can easily capture tens of millions points in a short time from large fields, such as engineering plants. If 3D stereoscopic scenes of large-scale point-clouds could be easily rendered using inexpensive devices, they might be involved in casual product development phases. However, it is difficult to render a huge number of points using common PCs, because VR systems require high frame rates to avoid VR sickness. To solve this problem, we introduce an efficient culling method for large-scale point-clouds. In our method, we project all points onto angle-space panoramic images, whose axes are the azimuth and elevation angles of head directions. Then we eliminate occluded and redundant points according to the resolutions of devices. Once visible points are selected, they can be rendered in high frame rates. Visible points are updated when the user stays at a certain position to observe target objects. Since points are processed on image space in our method, preprocessing is very fast. In our experiments, our method could render stereoscopic views of large-scale point-clouds in high frame rates.

1. INTRODUCTION

Virtual reality (VR) is useful in a variety of phases in product development. VR is a technique to create an immersive virtual environment as if the user were in the scene. In design phases, VR can be used to confirm styling by placing products in realistic usage scenes. Simulation results can also be evaluated three-dimensionally in analysis phases. VR is also effective in process planning phases to instruct operations or to confirm manufacturability in virtual production lines.

Recently inexpensive stereoscopic devices were developed for computer games. These devices can be potentially used in a variety of product development phases [1]. However, it is time-consuming to create 3D virtual scenes if the user has to create many 3D models. This problem prevents us to utilize virtual environment in casual tasks.

Point-based models are useful to easily and rapidly create virtual environment. Recently, terrestrial laser scanners (TLS) are significantly improved, and they can easily capture a huge number of points in a short time from large fields, such as engineering plants. The state-of-the-art laser scanners can capture one million points in a second within one hundred meters.

When point-clouds are densely captured, they can be used to render realistic as-is scenes. If 3D stereoscopic scenes of a huge number of points could be easily rendered on inexpensive devices, they might be involved in casual product development phases.

However, VR systems require high frame rates for rendering scenes to avoid VR sickness. It is often pointed out that frame rates for VR should be more than 75 fps. In existing methods, it is not easy to render tens or hundreds millions points captured by TLSs in high frame rates on common PCs. An easy way to render a huge number of points is to reduce the number of rendered points, but this approach pays the penalty of missing details.

So far, point-based rendering has been intensively studied in the field of computer graphics [2-4]. In typical methods, 3D space is subdivided into hierarchical sub-spaces to efficiently render point-clouds in a multiresolution manner [5-15]. Rusinkiewicz et al. [10] proposed a fast rendering method for point-clouds based on level-of-details and quantization of coordinates. Dachsacher et al. [11] extended this method using the sequential point trees (SPT). Okamoto et al. [12] clustered points using the view frustum and depth determination. Pajarola, et al. [13] proposed out-of-core multi-resolution rendering for point-clouds. Wimmer et al. [14] developed Instant Points to reduce preprocess-ing time.

However, the frame rates of these methods are not very high. In most cases, they compromise the quality of rendering while objects are moving, and then they refine details when objects stop moving. Unfortunately, this approach is not suitable for stereoscopic devices, because head directions are always changing in VR environment. Kreylos, et al. [15] realized real-time rendering of large-scale point-clouds using an out-of-core view-dependent multiresolution rendering scheme. However, their method was developed on expensive high-end VR systems.

In this paper, we propose a method for rendering a huge number of points on common PCs. Since our method does not construct hierarchical structure of points, preprocessing is very fast. We realize high

Figure 1. Head mounted display (Oculus Rift DK2)

Figure 2. Distorted parallax images on HMD

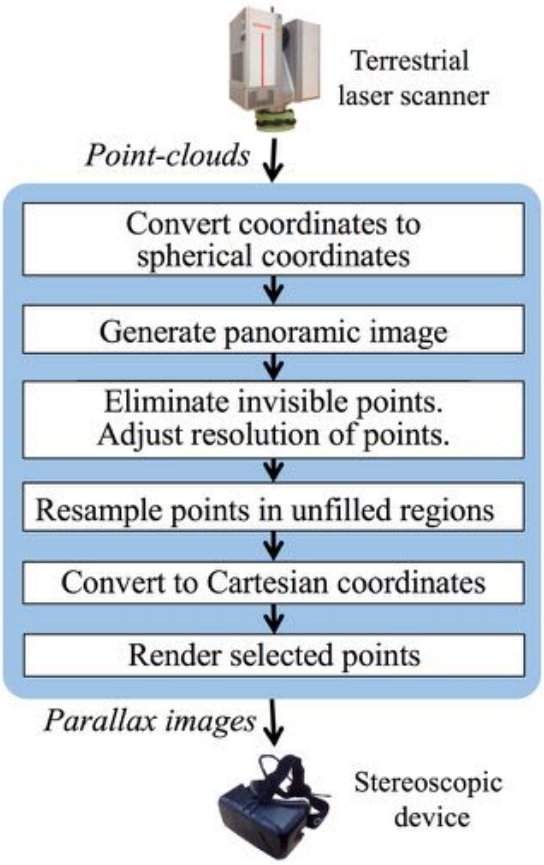


Figure 3. Process of our method

frame rates for rendering by eliminating invisible points and adjusting the resolution of points to the resolution of devices. We show that invisible points and redundant points can be easily and quickly eliminated on angle-space panoramic depth images, whose axes are the azimuth and elevation angles of head directions.

As an inexpensive device, we use the Oculus Rift DK2 (Figure. 1), which was developed by Oculus VR Inc. [16]. The resolution of this device is 1920 x 1080. The head tracking system traces head directions using a gyroscope, and the position tracking system traces the head position using an infrared camera. The device realizes a wide viewing angle by distorting parallax images using two fish-eye lenses. Figure 2 shows parallax images displayed on the screen of an Oculus HMD.

Figure 3 shows a process of our method. First visible points are selected from point-clouds. Then selected points are transferred to a stereoscopic device, and they are rendered according to head directions in real-time. In this paper, we define frame rates that are equal or higher than 75 fps as "real-time" rendering.

When the user moves forward or backward and changes the head position, visible points are recalculated and updated. This process is done in background processes while points are rendered. In the phase for selecting visible points, we transform coordinates of point-clouds to the spherical coordinate system, whose origin is the head position. Then, we create a panoramic image in angle space, as shown in Figure 4. The resolution of panoramic images is determined so that the resolution is equal or higher than the one of the stereoscopic device. When multiple points are projected on the same pixel, redundant points are eliminated. When the density of points is smaller than the resolution of the panoramic image, unfilled regions may be generated on rendered images. Then we estimate surfaces and resample points in unfilled regions.

2. GENERATION OF PANORAMIC IMAGE

2.1. Panoramic image

In our method, point-clouds are converted to a panoramic image in angle space. We define a local spherical coordinate system, in which the origin is the head position. Then each coordinate (x, y, z) in point-clouds is converted into a spherical coordinate (θ, φ, r), where θ is the elevation angle, φ is the azimuth angel, and r is the distance from the origin. A panoramic image (Figure 4) is generated by quantizing (θ, φ) and describing RGB colors of points at each pixel. In this paper, we generate panoramic images so that the resolution is equal or higher than the one of the stereoscopic device.

When the density of points is higher than the resolution of the panoramic image, multiple points are projected on the same pixel. Then the point with the smallest depth is maintained at the pixel.

Figure 4. Panoramic image generated by points

2.2. Resample points in unfilled regions

When the resolution of points is less than the one of panoramic images, unfilled regions may be generated on panoramic images, as shown in Figure 5. In this figure, any points are not projected in white regions. Unfilled regions are often generated on distant objects or inclined surfaces, on which point density becomes sparse.

Unfilled regions are also generated when laser beams are occluded by other objects. However, it is difficult to estimate arbitrary occluded surfaces. In this paper, we fill only planar regions by resampling points on planes.

We first fill salt-and-pepper gaps simply by interpolating the depths as the average of 3x3 neighbor points on the panoramic image. In the next step, we detect unfilled regions and select their boundary points. When the boundary points fit a plane approximately, the unfilled region is regarded as a continuous plane. Then we resample points in each unfilled region by calculating coordinates on the plane. In addition, colors in the unfilled region

are calculated as the average of two color values, which are interpolated linearly in each of the two

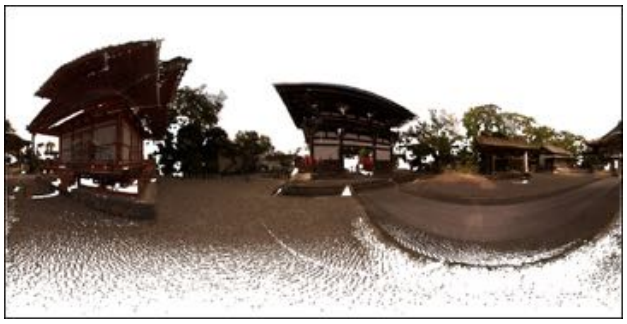


Figure 5. Unfilled regions on panoramic image

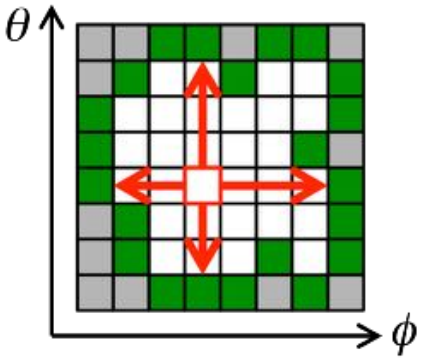


Figure 6. Linear interpolation in two directions on the panoramic image.

Figure 7. Panoramic image filled by additional points

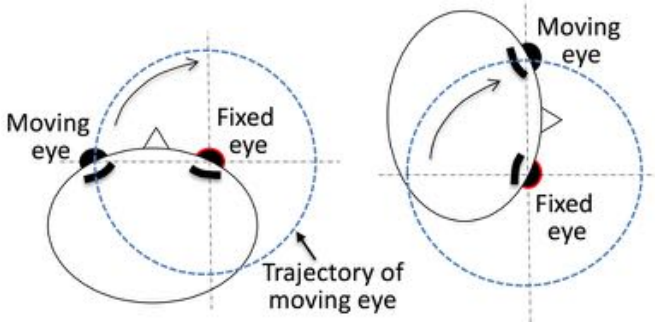
directions, lateral and longitudinal on the panoramic image, as shown in Figure 6.

Figure 7 shows an interpolated result, in which resampled points fill planar unfilled regions.

3. EXTRACTING VISIBLE POINTS

3.1. Visible points from the fixed eye

To render stereoscopic views on a HMD, two parallax images have to be generated. Since it is time-consuming to separately calculate visible points from two eyes, we select visible points only from



(a) Looking forward (b) Looking right direction

Figure 8. Fixed eye and moving eye

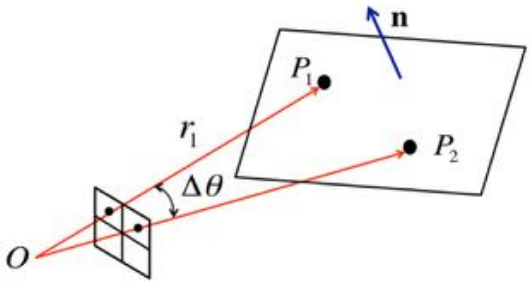


Figure 9. Two adjacent points on a surface

one eye, and then select additional points that are visible from the other eye.

To reduce computation time, we suppose that one eye is fixed, and the other eye moves around. In Figure 8(a), the head direction of the user faces forward, and in Figure 8(b), the head direction is turned to the right. In this paper, we suppose that the position of the moving eye moves horizontally. Then, visible points from the fixed eye can be simply obtained as a panoramic image, because the fixed eye is defined as the origin of the spherical coordinate system.

3.2. Detection of boundary points

When visible points from the fixed eye are obtained as a panoramic image, continuous regions can be detected from the panoramic image. Figure 9 shows two adjacent points on a panoramic image. When two points $\mathbf{P}_1$ and $\mathbf{P}_2$ are on the same surface, they approximately satisfies:

$$|\mathbf{P}_1 - \mathbf{P}_2| \approx \frac{|\mathbf{P}_1|^2 \Delta\theta}{|\mathbf{P}_1 \cdot \mathbf{n}|}, \quad (1)$$

where $\Delta\theta$ is the angular resolution of points, and **n** is the normal vector of a surface.

Since it is time-consuming to estimate normal vectors at all points, we suppose that the angle between $\mathbf{OP}_1$ and **n** is less than 75 degree. Then Equation (1) can be simply represented as:

$$|\mathbf{P}_1 - \mathbf{P}_2| < |\mathbf{P}_1|\Delta\theta/\cos 75^\circ. \quad (2)$$

When two points satisfy this condition, they are regarded as point on the same surface.

3.3. Visible points from the moving eye

In Figure 10, continuous surfaces are shown in gray, and boundary points of surfaces are shown in blue. As shown in this figure, depth values largely vary at the boundaries of continuous surfaces. When visible points from the fixed eye are given, additional visible points from the other eye can be observed between boundary points of two continuous points, as shown in Figure 10.

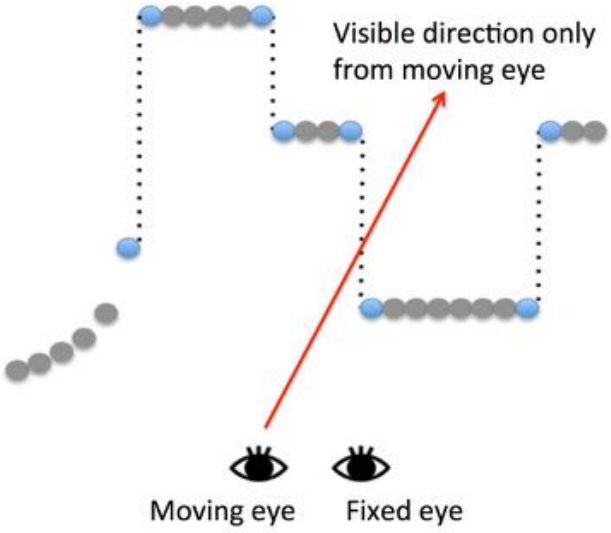


Figure 10. Points on continuous surfaces

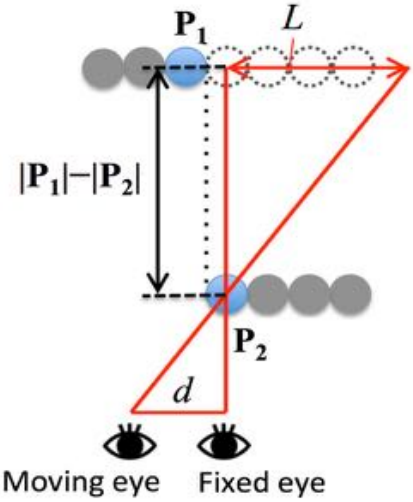
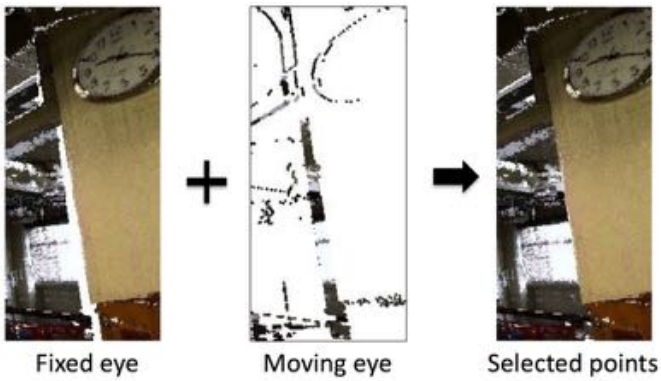


Figure 11. Additional visible points for moving eye



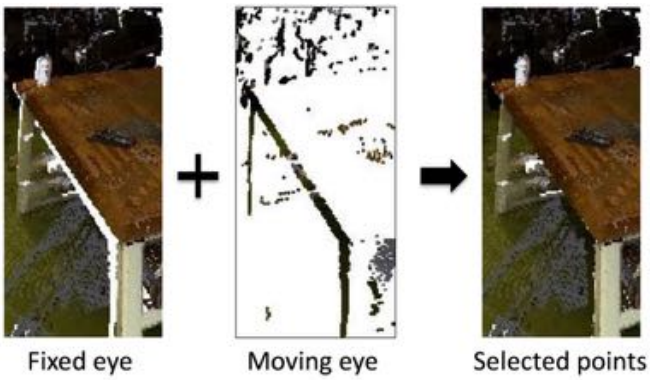


Figure 12. Visible points from two eyes

To detect points that are visible only from the moving eye, we estimate the range of additional points using two boundary points. We denote the range of additional points as L, and the distance between two eyes as d, as shown in Figure 11. The origin is the fixed eye. Then L can be represented as:

$$L = d\left(\frac{|\mathbf{P}_1|}{|\mathbf{P}_2|} - 1\right) \quad (3)$$

We project points on the panoramic image again, and select occluded points when the distance from boundary points is less than L.

Figure. 12 shows visible points from two eyes. The left figures show visible points from the fixed eye. When these points are viewed from the moving eye, gaps, which are shown in white, can be observed near boundary points. The middle figures

show additional points that are visible only from the moving eye. The right figures merged points, which are sufficient to generate parallax views for both eyes.

3.4. View frustum culling on panoramic image

When points are out of the view frustum, they do not have to be rendered. However, it is time-consuming to apply view frustum culling to a huge number of points. To efficiently culling points, we subdivide panoramic images into strips, as shown in Figure 13.

Since strip regions are defined in angle space, visibility of each strip can be quickly evaluated using head directions and viewing angles. We denote the head direction as (θ, ϕ), and the viewing angles as w and h. Then points in each strip are rendered only when the strip region overlaps with the region defined by $\theta \pm (w/2 + \varepsilon)$ and $\phi \pm (h/2 + \varepsilon)$, where ε is a constant value for parallactic angles.

4. EXPERIMENTS

4.1. Performance evaluation

We evaluated the performance of our method using point-clouds of a factory in our university. An example of a point-cloud is shown in Figure 14. This point-cloud consists of about 40 million points. We captured point-clouds at several positions using the laser scanner HDS7000, which were developed by Leica geosystems. The measurement time is about 5 minutes for setting up a laser scanner, and about 1 minute to measure a point-cloud.

Computation was performed on a PC with 16GB RAM and an Intel Core i5-4440 3.10GHz CPU. When we measured rendering performance, we calculated the average of 1000 trials. Point-clouds were rendered using OpenGL with NVIDIA GeForce GT 640. We used an Oculus Rift DK2 as an immersive head mounted display. Parallax images are generated using the Oculus SDK.

First we evaluated how many points could be rendered in high frame rates without using our method. We evaluated frame rates while changing the number of points. Figure 15 shows the results. In this evaluation, only one million points could be rendered in 75 fps.

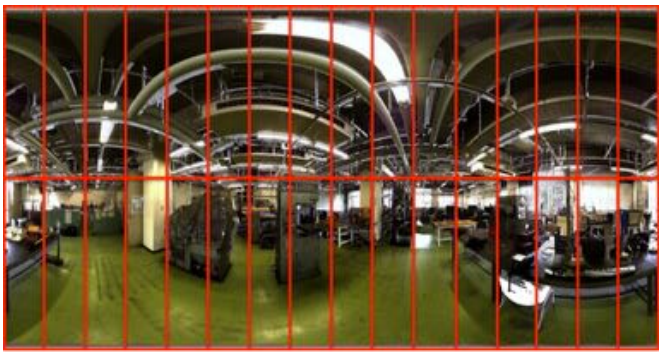


Figure 13. Subdivision for view frustum culling

Figure 14. Point-cloud of our university factory

Then we uniformly reduced the number of points to one million. Figure 16 shows the rendering result. This result shows that the rendering quality of reduced points is significantly degraded.

Then we evaluated frame rates using our proposed method. In our method, points are selectively reduced on image space by eliminating invisible points and redundant points. The number of rendered points is determined according to the resolution of stereoscopic devices. When the higher resolution is specified, the larger number of points are selected. We rendered points while changing the resolution of rendered images.

The results are shown in Figure 17. In this experiment, the frame rate exceeded 75 fps when the resolution was 1944 x 972 pixels. We note that there are about 1.9 million points on an image of 1944 x 972 pixels, but rendered points are reduced to less than one million using strip-based view frustum culling.

Even when the number of rendered points was limited to less than one million, our method could

generate much more clear images, as shown in Figure 18. This is because our method selected a small number of rendered points according to the visibility of points and the resolution of the device.

In addition, our method requires only small sizes of memory, because the system maintains only points on a panoramic image. When the resolution of the devise is 1944 x 972 pixels, the data size is about 28 MB.

We also evaluated timing for generating panoramic images while changing the sizes of point-clouds. Since all points are projected onto a

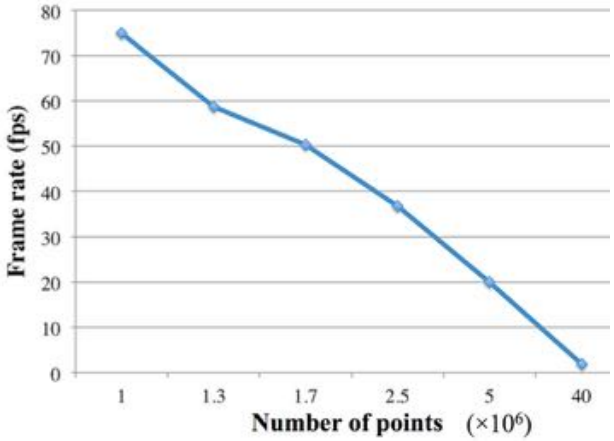


Figure 15. Relationship between frame rates and the number of points.

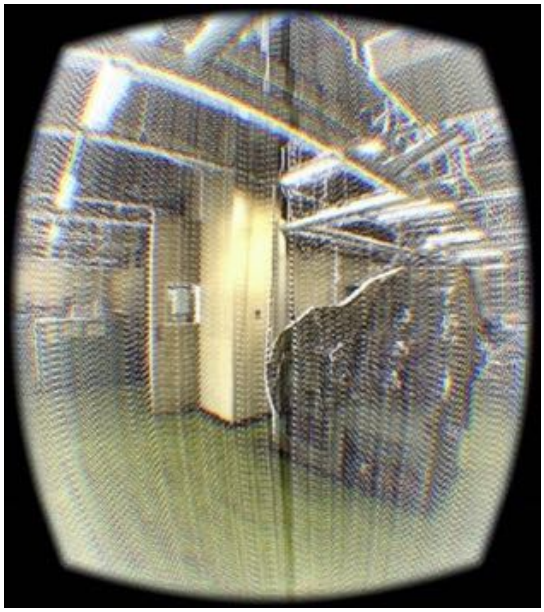


Figure 16. Points rendered in 75 fps without using our method

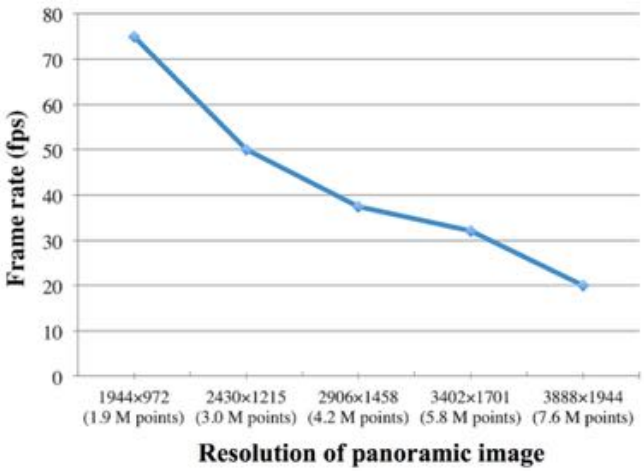


Figure 17. Frame rates of our method

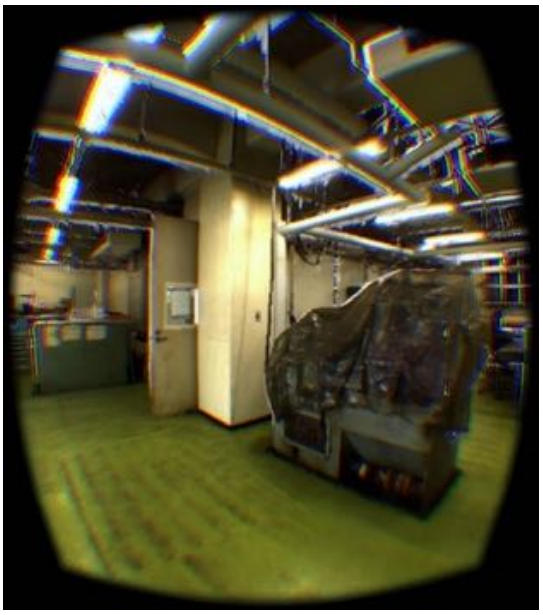


Figure 18. Points rendered in 75 fps using our method

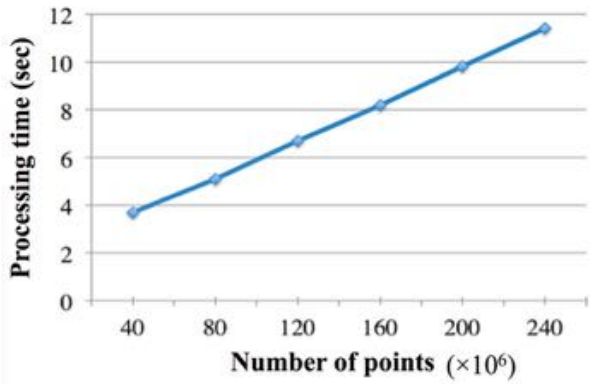


Figure 19. Pre-processing time for selecting points

panoramic image, computation time becomes longer when the number of points is larger. Figure 19 shows processing time to generate panoramic images. The frame rates were linearly decreased according to the number of points.

In this experiment, approximately 20 million points could be processed in a second. Therefore the user has to wait updating rendered points in a several second when the user moves to a new position. Once points are updated, points can be rendered in a stereoscopic way in 75 fps.

5. CONCLUSION

In this paper, we proposed a fast stereoscopic rendering method for large-scale points captured using terrestrial laser scanners. In our method, visible points are extracted on panoramic images based on the visibility of points and the resolution of the device. In our experiments, we showed that our method could render stereoscopic images of large-scale point-clouds in 75 fps.

In future work, we would like to consider methods that can generate higher resolutions of images in 75 fps. In our current implementation, the user has to wait several seconds when the user changes head positions. We would like to reduce waiting time in preprocessing. We would also like to develop method to flexibly interact with stereoscopic views using emerging devices for user interaction.

REFERENCES

1. Bharathi, A. K. B. G. and Conrad, S. T.: Investigating the Impact of Interactive Immersive Virtual Reality Environments in Enhancing Task Performance in Online Engineering Design Activities. ASME 2015 IDETC (2015)
2. Kobbelt, L., Botsch, M.: A survey of point-based techniques in computer graphics. Comput. Graph. 28(6), 801–814 (2004)
3. Sainz, M., Pajarola, R.: Point-based rendering techniques. Com- put. Graph. 28(6), 869–879 (2004)
4. Gross, M.H., Pfister, H.: Point-Based Graphics. Morgan Kauf- mann, San Mateo (2007)
5. Mario, B., Wiratanaya, A. and Kobbelt, L.: Efficient high quality rendering of point sampled geometry. Proceedings of the 13th Eurographics Workshop on Rendering (2002)
6. Mario, B. and Kobbelt, L.: High-quality point-based rendering on modern GPUs. Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on. IEEE (2003)
7. Gobbetti, E., Marton, F.: Layered point clouds. In: Proceedings Eurographics/IEEE VGTC Symposium on Point-Based Graphics, pp. 113–120 (2004)
8. Michael, W. et al.: Interactive Editing of Large Point Clouds. SPBG (2007)
9. Ruwen, S., Möser, S. and Klein, R.: A Parallelly Decodeable Compression Scheme for Efficient Point-Cloud Rendering. SPBG (2007)
10. Rusinkiewicz, S., Levoy, M.: QSplat: A multiresolution point rendering system for large meshes. In: Proceedings ACM SIG- GRAPH, pp. 343–352 (2000)
11. Dachsbacher, C., Vogelgsang, C., Stamminger, M.: Sequential point trees. ACM Trans. Graph. 22(3) (2003). Proceedings ACM SIGGRAPH
12. Okamoto, Y.: Sequential Point Clusters: Efficient Point-based Rendering Method for Huge 3D Models. IPSJ Journal, 46(1), pp. 1234-1237 (2005)
13. Pajarola, R., Sainz, M., Lario, R.: XSplat: External memory multiresolution point visualization. In: Proceedings IASTED International Conference on Visualization, Imaging and Image Processing, pp. 628–633 (2005)
14. Wimmer, M., Scheiblauer, C.: Instant points: Fast rendering of unprocessed point clouds. In: Proceedings Eurographics/IEEE VGTC Symposium on Point-Based Graphics, pp. 129–136 (2006)
15. Kreylos, O., Gerald W. B. and Louise H.: Immersive visualization and analysis of LiDAR data. Advances in visual computing. Springer Berlin Heidelberg, pp. 846-855 (2008)
16. https://developer.oculus.com