Proceedings of the 2nd International Conference on Information Technology for Application (ICITA 2004)

Image-Based Point Rendering for Multiple Range Images

Hiroaki Kawata, ***Non-Member, IEEE***, **Takashi Kanai**, ***Member, IEEE***

***Abstract*--In this paper, we describe image-based point rendering (IBPR) for multiple range images from 3D scanners. Our approach is a natural extension of the method called pull-push so as to render scanned points with some measurement errors. Several extensions for rendering range images are proposed. One is a seamless rendering of a whole object even for points that have measurement errors. The other is a high quality rendering to reduce blurring. Our method is suitable for roughly checking the shape from range images.**

***Index Terms*--Culling, Point-Based Rendering, Range Images, Semi-transparency Rendering.**

I. Introduction

ADVANCES in 3D scanning technologies have enabled the practical creation of meshes with hundreds of millions of polygons. One problem, however, is handling such large meshes. The construction of meshes from range images is a laborious work, even though various reconstruction algorithms are proposed [14]. Therefore, it has an advantage to render points directly, achieving a similar effect as surface rendering.

First, Levoy and Whitted [10] proposed to use points for rendering objects. For rendering, points are faster than surface primitives such as polygons in general. Therefore, several approaches for rendering points combined with LOD (Level-Of-Detail) techniques were proposed. For example, applications to complex geological data [8], ecosystems [2], complex scenes [11], and complex objects [1] have been discussed.

Points don't have the information needed for surface rendering such as face, connectivity, etc. To give the impression of surface rendering when using only points, one has to address the issue of filling the ``hole'' between adjacent points. There are roughly two approaches to fill such a hole; one is based on using a *splat*, the other is an image-based approach.

Rusinkiewicz and Levoy proposed an efficient method called QSplat [7] for rendering large number of points from 3D scanner. It is based on multiresolution technique to search the minimum set of points which is sufficient for shading. However, a special multiresolution-based data structure needs to be computed as a pre-process to use this approach. It is a disadvantage when the number of points is quite large.

Hiroaki Kawata is at Keio University, Faculty of Environmental Information (e-mail: t02282hk@sfc.keio.ac.jp).
Takashi Kanai is at Keio University, Faculty of Environmental Information (e-mail: kanai@sfc.keio.ac.jp).

ICITA2004 ISBN 0-646-42313-4

Zwicker et al. described a high-quality point rendering [6],[9]. In [6], the information for rendering points is called *surface element* (*surfel*). The fundamental element of surfel [6] consists of position, color, radius and normal. Surfel rendering techniques based on image-based approach or on splatting approach are used for visibility testing. In splatting approach a rectangle or a disk is used instead of a point. Zwicker et al. described a surface splatting algorithm with an anti-aliasing method [9]. Coconu and Hege proposed an efficient rendering method for complex scenes used graphics hardware [12].

Grossman and Dally [4] propose a method called *pull-push* for pseudo-surface rendering from points. This approach is based on an image reconstruction: For a screen buffer, an additional image buffer with lower resolution is prepared. Points are stored in this image buffer to be used in a hole-filling process.

In Image-Based rendering (IBR) techniques, the rendering time of IBR depends on image resolution. Gortler and He [13] proposed an efficient rendering method called LDI (Layered-Depth-Image). LDI is applied for rendering point-sampled geometry.

The main purpose of this research is to directly render range images obtained from multiple scans. We call our approach image-based point rendering (IBPR), and it is a natural extension of pull-push approach proposed by Grossman and Dally [4]. We first extend the method to the rendering of multiple range images. These data from 3D scanners sometimes include measurement errors. Because of these errors, it can occur that two boundaries of such range images do not overlap. Our method establishes a robust and high quality rendering even for such data. We then improve the quality of the rendering. We propose a novel approach so that the resulting image is clearer.

In addition, we propose an alpha blending algorithm for IBPR. We show that semi-transparency rendering can be done within our approach's framework.

II. Image-Based Point Rendering

In this section, we describe the data for IBPR and rendering process. In addition, we discuss about semi-transparent method and simple culling method for IBPR.

A. Algorithm Overview

We mainly treat range images from 3D scanners. The minimum elements required for our point rendering are the position (x, y, z) and the resolution of the target data.

If we render points with color, we need an additional color

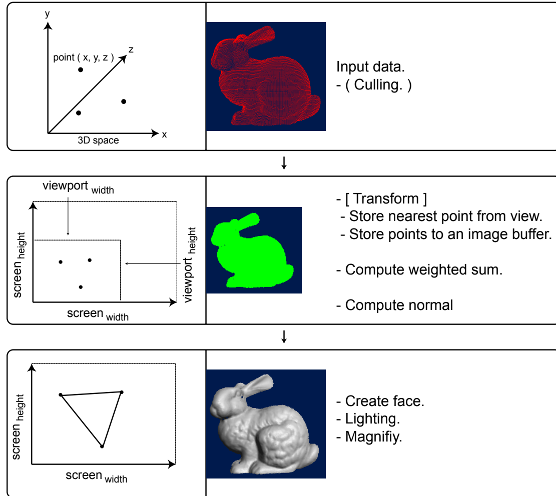


Fig. 1 Rendering process of our IBPR approach.

element (r, g, b, a) for each point. Normals can be computed on-the-fly during the rendering process (Sec.II.C).

For our approach, we prepare an image buffer to store the following elements: 2D position (u, v), original 3D position (x, y, z), a nearest point from view (x, y, z), normal (nx, ny, nz), color (r, g, b, a) and the number of points per pixel.

These buffers are allocated before the rendering process. The buffer size is determined according to the size of screen buffer. We also assume that each pixel can store a floating-point value. This floating-point pixel is used to improve the quality of the resulting image.

Fig. 1 describes a whole rendering process of our approach. For each frame, our rendering algorithm executes the following process:

1. Compute the size of an image buffer.
2. Store points to an image buffer.
3. Compute normals and colors for shading.
4. Create faces and magnify an image buffer to the size of a screen buffer.

To further to improve the quality of the resulting image, we add one more rendering pass. We will describe in details the two pass rendering in Section.II.E.

The size of an image buffer $(viewport_{width}, viewport_{height})$ is determined by the following equation,

$$viewport_{width} = \frac{screen_{width}}{s}, \quad viewport_{height} = \frac{screen_{height}}{s} \qquad (1)$$

where s is a variable determined by fov, the density of points and distance between view point and target object. $(screen_{width}, screen_{height})$ is the width and the height of the resulting image.

B. Storing Point to Buffer

For each point, we apply transformations, projections and we scale by s. The resulting 2D point is rasterized and is stored to an image buffer.

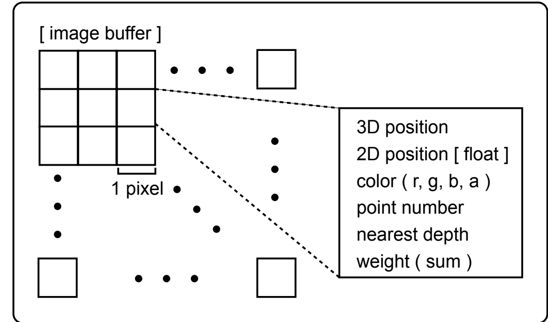


Fig. 2 The definition of an image Buffer.

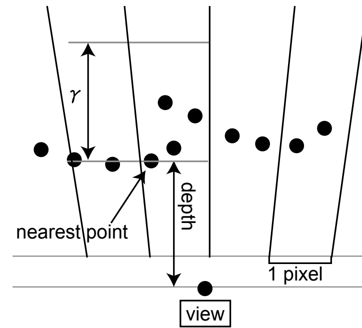


Fig. 3 The geometric meaning of γ.

The image buffer is described in Fig. 2. If there is another pixel value in a stored pixel, such a value is updated by calculating a weighted sum. The weight of each point is then calculated as follows,

$$w = \frac{|z - z_{near}|}{\gamma} \qquad (2)$$

where z is the depth value (distance to view plane) of the point of interest, and z_{near} the minimum depth value of the current pixel's point set.

γ is a threshold to determine whether a point is included in a *face* or not. It is an important parameter especially for the rendering of multiple range images. The geometric meaning of a threshold γ is described in Fig. 3. For each pixel, γ is defined as a distance from the point which is nearest from view point. If a distance between a point and the nearest point is larger than γ, this point is regarded as on another face, then it is not included to a pixel. Else, it is added to a pixel with calculating a weight described in Equation (2).

By using γ, any value of a pixel can be calculated as a weighted interpolation of points in the range of γ. This is useful when treating multiple range images. Suppose two range images have measurement errors, then points of two boundaries do not overlap. Such non-overlapping points will be treated as only one face. An appropriate value of γ largely depends on the extent of measurement errors. To our experiments, it is good to set γ to 1-3 times the interval of points on a range image.

We can apply visibility testing and can draw the surface by

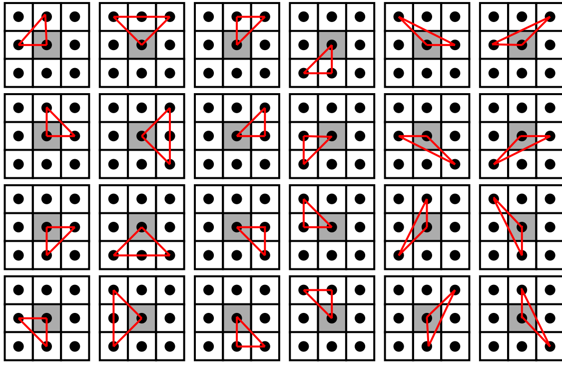


Fig. 4 Patterns for creating face.

usual methods like Z-buffer.

In IBPR, we use a weighted sum of points in the range of γ, instead of the nearest depth value to a view point in each pixel.

C. Creating Normal

Here we describe how to compute normals for shading. One algorithm has been described in Hoppe et al.'s paper [5]. For each point, this algorithm gathers neighbor points and compute the principal axis of an inertia ellipsoid. This direction is regarded as an approximate axis parallel to a normal vector. We could use [5] as pre-processing. But an additional computation to orient all normals to outer directions would be needed. In contrast, our approach described below is executed during the rendering process and is easier to compute.

We compute the normal of a pixel depending on its neighbor pixels. Fig. 4 illustrates 24 patterns for creating a face from a pixel and its neighbors. In Fig. 4, the 9 squares of each pattern denote a pixel and its 8 neighbors. A triangle in a square shows a face generated by 3D coordinates of pixels. A filled gray square denotes a target pixel. If several patterns match for one pixel, an average of normals of all these patterns is calculated.

In the above approach, if z values of neighboring two pixels are very different, a blurring between two pixels is seen in the resulting image. To overcome this issue, an evaluation of z values is added. We assume here that normal vector is not calculated when the difference between z values of two pixels is more than a threshold. We use the same threshold as described in Section.II.B.

Fig. 5 illustrates the above situation: In Fig. 5, filled squares show pixels near view point and white squares show pixels far from filled pixel. The arrows from target pixel show the region of interest. To compute a normal vector for a pixel, only a region surrounded by solid lines is considered. Fig. 6 shows the results with or without considering such a threshold. Considering a threshold decreases a burring effect.

D. Creating Face

We create faces for a given pixel by using neighbor pixels. As shown in Fig. 7, we determine whether a triangle or a rectangle (or nothing) is defined by referring neighbor three or four pixels. A point in a pixel shows that a pixel has at least one 3D position. Filled regions denote a triangle or a rectangle created by the above operation.

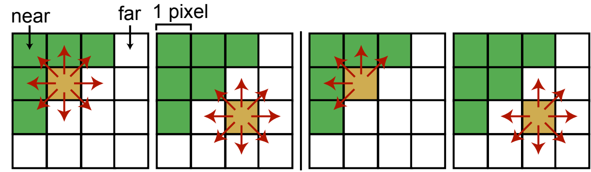


Fig. 5 Range of target pixel for creating normal.

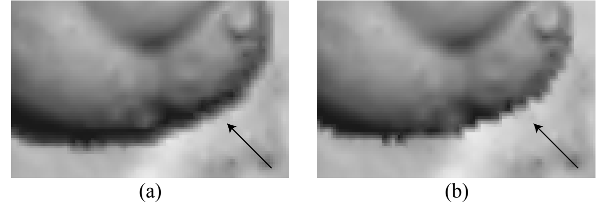


Fig. 6 The result with or without using threshold. (a) without threshold. (b) with threshold.

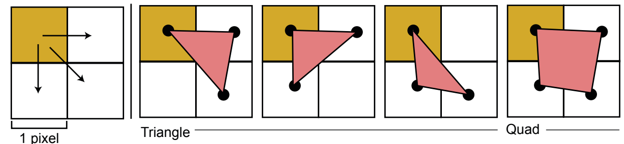


Fig. 7 Creating Face.

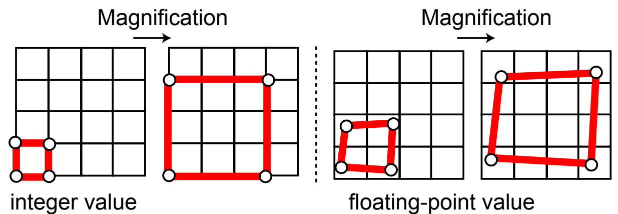


Fig. 8 Magnification using floating point values of vertices.

After this operation, the image buffer is scaled using s to fit the size of the screen buffer. The color of a scaled face (defined as a set of pixels) can be computed by a standard shading calculation.

Faces are created from the image buffer by magnification. In this process, positions that project onto 2D space (stored by floating-point values) for each pixel are used to magnify vertices of faces. If we use rasterized 2D positions (stored in integer values) for magnification, the quality of the resulting image could be decreased. Fig. 8 shows such a situation. The left and the right figures illustrate the results using integer value and floating-point value positions, respectively. This is the reason why we store the floating-point value position into a buffer during the whole rendering process.

E. Sharpness Operation

One issue of our approach is that blurring occurs in the resulting image, especially for bumpy regions. Fig. 11 illustrates this case. From our observation, more points are stored in a pixel in region where normal direction is nearly perpendicular to the view vector. A similar case can occur at silhouette edges of an object. There are some jaggies on these edges.

We solve this issue by introducing two-pass rendering. For silhouette edges, we change the size of the image buffer to store points during the rendering process (Fig. 9). In the first pass,

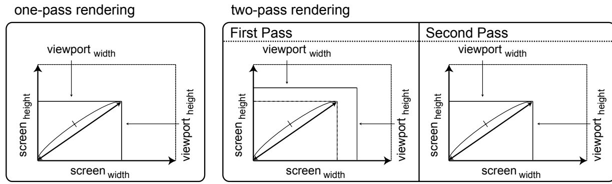


Fig. 9 The resolution of one-pass and two-pass rendering.

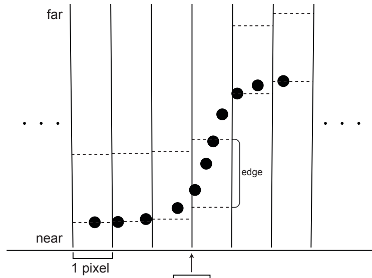


Fig. 10 The number of point of a pixel.

we set the size of the image buffer larger (then s is smaller) than that used in one-pass rendering. This reduces the number of points stored in each pixel and then blurring can also be reduced. According to our experiments, gives nice results setting s to a 0.83-0.66 times smaller value than for one-pass rendering.

For the second pass, we set the image buffer size as big as that used one-pass rendering. More concretely, we do not create faces for a pixel satisfying with either one of the following two conditions:

A) The number of points in a pixel is more than a threshold.

B) A normal direction of a pixel is nearly perpendicular to view vector (Fig. 10).

The final result is obtained by using the second pass results everywhere but in A and B-type pixels where high resolution first pass results are used.

Fig. 11 illustrates the comparison between one and two pass rendering. Fig. 11 (a), (b) are examples of silhouette edges. It can be seen from the results that two-pass rendering produces more accurate and smooth silhouette edges. Fig. 11 (c), (d) are a examples of a bumpy region including a concave shape. The image generated by two-pass rendering is sharper than the one generated by one-pass rendering.

F. Semi-Transparency Rendering

In this subsection, we describe a semi-transparency rendering technique that can be used within the framework of our IBPR.

As for semi-transparency rendering, Everitt has proposed a technique which is ordering independent semi-transparent algorithm [3]. In this method, a multi-pass rendering is needed in order to determine faces which are in the back.

Our semi-transparency rendering is based on generating two or more layers in an image buffer. Fig. 12 illustrates an example. Each pixel of an image buffer has several layers. Each layer (represented by dot lines) consists of several 2D points and has its own depth, color and 2D coordinates. The layer

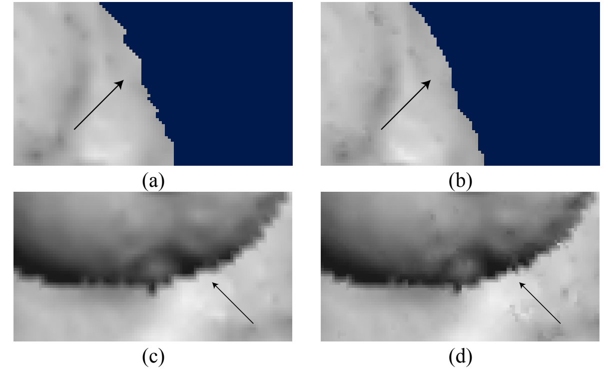


Fig. 11 Results with different rendering passes. (a), (b) silhouette edges. (c), (d) a bumpy region. (a), (c): one-pass. (b), (d): two-pass.

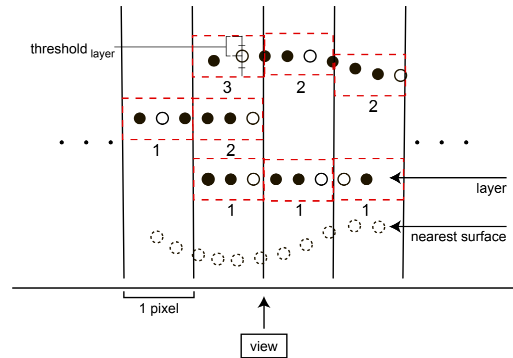


Fig. 12 Layer for semi-transparency.

attributes (color, depth, 3D position, normal) are calculated by averaging the 2D point's attributes.

Semi-transparency rendering can be done by the following procedure:

- For each rasterized 2D point, its depth value is compared to that of each existing layer. If it is outside every layer, a new layer is created and this 2D point is stored inside. If the depth value is within the range of an existing layer ($threshold_{layer}$), the point is added to this layer and the layer's attributes are updated.
- Within each pixel, layers are sorted according to depth values.
- For each layer within a pixel, a normal vector is calculated using the method described in Section.II.C and we apply shading operation to compute the layer's color inside this pixel.
- To compute final pixel color we compose the color of each layer. The blending operation is using the following equation:

$$r_0 = c_{background}(1-\alpha_0)+c_0\alpha_0, \quad r_n = r_{n-1}(1-\alpha_n)+c_n\alpha_n \quad for \quad n \ge 1 \qquad (3)$$

where r_i is the resulting color of the operation, c is color of the layer. $\alpha(0 \le \alpha \le 1)$ denotes transparency of each layers. N being the number of layers in the pixel color will be r_{N-1}.

G. Process of Culling

In this subsection, we describe a culling technique for IBPR.

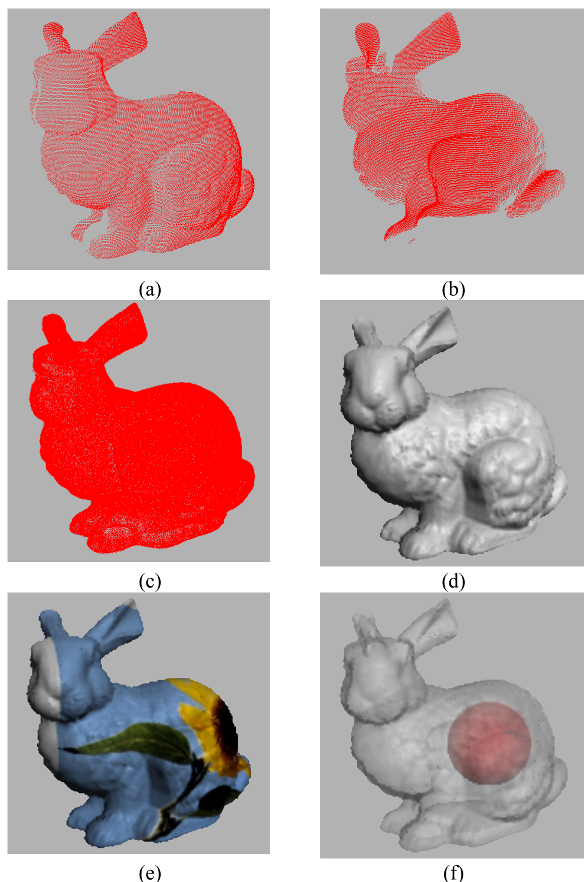


Fig. 13 The results of rendering ``Bunny'' range data.

IBPR doesn't need to combine all range scan images. We can select range images depending on the view and thus increase the rendering speed.

Here we explain a simple culling method. Our approach is the same as the general back-face culling approach used for drawing surfaces. For each range image, we calculate a dot product between the direction of scanning and the view direction, and determine whether it should be included for drawing or not.

III. Results

We used two range images directly to evaluate the efficiency of our approach. In these experiments, computation times are measured on personal computer with Athlon XP 2600+ 2.09GHz CPU.

One example is ``Stanford Bunny'' range image. Fig. 13 shows such range image and the results of point rendering.Fig. 13 (a) and (b) denote two single scan images respectively. Fig. 13 (c) shows a whole range image scanned from 10 directions (362,272 points). Fig. 13 (d) is the result of a single-pass point rendering using the data in Fig. 13 (c). It can be seen that seamless rendering can be done even for such multiple range images. Fig. 13 (e) is the result of rendering with colors. Fig. 13 (f) is the result of semi-transparency rendering. A sphere was inserted into Bunny's body for illustration purpose.

The other example uses ``Happy Buddha'''s range images. The results of rendering for points scanned from 18 directions

Model	S	Alpha	Points	Pass	Time
bunny	2.7	No	362272	1	0.20
bunny	2.7	No	362272	2	0.41
bunny	2.7	Yes	362272	1	0.44
bunny	2.7	Yes	362272	2	0.91
buddha	1.6	No	1274573	1	0.47
buddha	1.6	No	1274573	2	0.98

Table. I Computation time (second) for rendering "bunny" and "Buddha" data.

Model	S	Culling	Points	Time
Beetle	1.31	Off	559327	0.28
Beetle	1.31	On	307933	0.23

Table. II Computation time (second) for rendering "Beetle" data.

(1,274,573 points) are described in Fig. 14. In this figure, the comparison between one-pass and two-pass rendering is shown. From the comparison of two close-up views, it can be seen that the blurring effect is reduced by two-pass rendering. The computation time is describing in Table. I.

We also examine culling process using ``Beetle'' range images. This model scanned from 9 directions (559,327 points) is described in Fig. 15. This rendering result with culling process time is describing in Table. II.

IV. Conclusion and Future Work

We have proposed an image-based point rendering approach for multiple range images. Our approach is a natural extension of pull-push method. We have shown that the shading, which is close to surface rendering, can be established even for range data with some measurement errors. No special data structure for a pre-process is needed. Especially, normals needed for shading can be computed during the rendering process. We have also proposed a two-pass rendering algorithm, which enables high-quality rendering. We have also shown that the resulting image has less blurs thanks to our extension. Furthermore, we have proven that semi-transparency rendering can be done within the framework of our image-based point rendering.

One future direction is hardware acceleration of our image-based approach. We are especially interested in using programmable shaders to implement such accelerations.

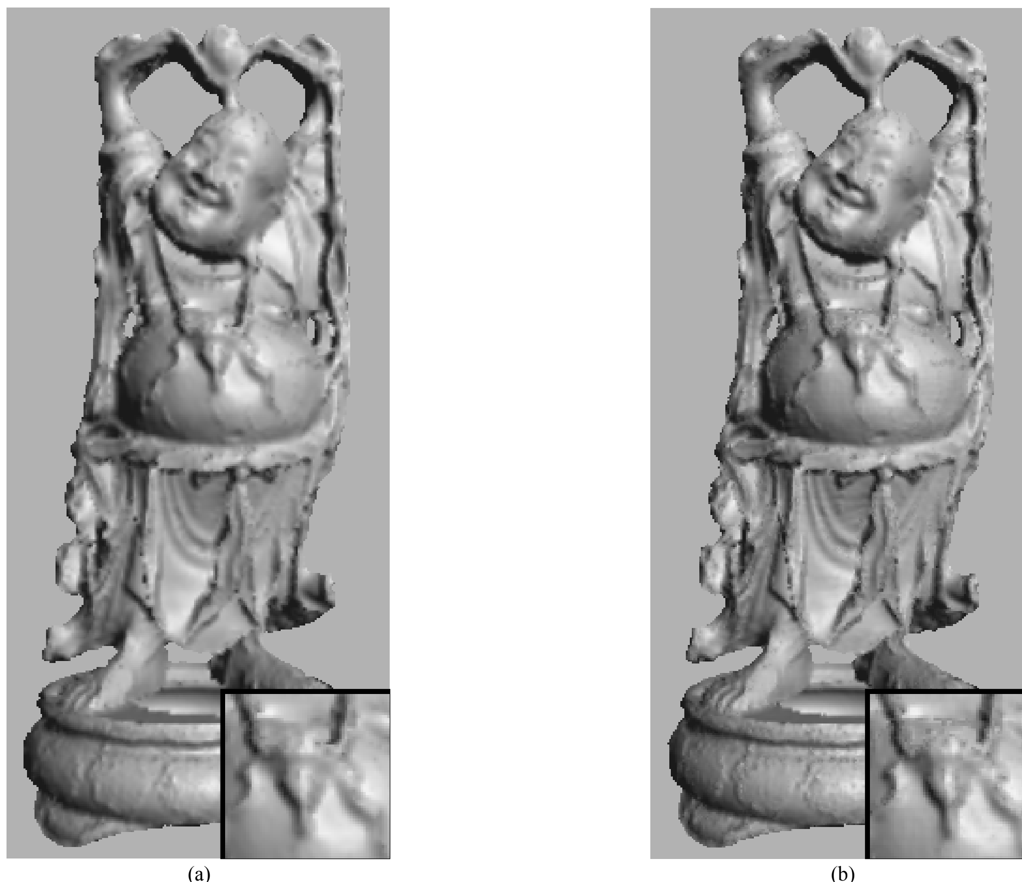
Acknowledgment

``Stanford Bunny'' and ``Happy Buddha'' range images are courtesy of Stanford University Computer Graphics Laboratory. And ``Beetle'' range images are courtesy of Prof. Kenji Kohiyama at Keio University. We would also like to thank Mr. Alexandre Gouaillard for proofreading this paper.

References

[1] B. Chen and M. X. Nguyen. Pop: a hybrid point and polygon rendering system for large data. In Proc. IEEE Visualization 2001, pages 45–52, 2001.

[2] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In Proc. IEEE Visualization 2002, pages 219–226, 2002.



(a) (b)

Fig. 14 The results of rendering ``Happy Buddha" range data. (a) one-pass rendering. (b) two-pass rendering.

(a) (b)

Fig. 15 The results of rendering ``Beetle" range data. (a) without culling. (b) with culling.

[3] C. Everitt. Interactive order-independent transparency. nVIDIA Inc., 2000. available from http://developer.nvidia.com/view.asp?IO=Interactive_Order_Transparency.

[4] J. Grossman and W. J. Dally. Point sample rendering. In Proc. 9th Eurographics Workshop on Rendering, pages 181–192, 1998.

[5] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. Computer Graphics (Proc. SIGGRAPH 1992), 26(2):71–78, 1992.

[6] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: surface elements as rendering primitives. In Computer Graphics (Proc. SIGGRAPH 2000), pages 335–342. ACM Press, New York, 2000.

[7] S. Rusinkiewicz and M. Levoy. Qsplat: a multiresoliton point rendering system for large meshs. In Computer Graphics (Proc. SIGGRAPH 2000), pages 343–352. ACM Press, New York, 2000.

[8] M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In Proc. 11th Eurographics Workshop on Rendering, pages 151–162, 2001.

[9] M. Zwicker, H. Pfister, J. van Beer, and M. Gross. Surface splatting. In Computer Graphics (Proc. SIGGRAPH 2001), pages 371–378. ACMPress, New York, 2001.

[10] M. Levoy and T.Whitted. The Use of Points as a Display Primitive. TR 85-022. Univ. of North Carolina at Chapel Hill, 1985.

[11] M. Wand, M. Fischer, I. Peter, F. Meyer auf der Heide and W. Straßer. The Randomized z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes. In Computer Graphics (Proc. SIGGRAPH2001), pages 361–370. 2001.

[12] L. Coconu and H. Hege. Hardware-Oriented Point-based Rendering of Complex Scenes. In Proc. 13th Eurographics Workshop on Rendering, 2002.

[13] S. J. Gortler and L. He. Rendering Layered Depth Images. MSTR-TR-97-09, 1997.

[14] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In Computer Graphics, Volume 21, Number 4, July 1987.