Image-Based Point Rendering and Its Application to Color Editing Tool

Hiroaki Kawata
Keio University, SFC
5322 Endo, Fujisawa-city,
Kanagawa, 252-8520, Japan.
t02282hk@sfc.keio.ac.jp

Alexandre Gouaillard
CREATIS, INSA de Lyon
69621 Villeurbanne, France
alexandre.gouaillard@insa-lyon.fr

Masahiko Morita
Keio University, SFC
masahiko@sfc.keio.ac.jp

Kenji Kohiyama
Keio University, SFC
kohiyama@sfc.keio.ac.jp

Takashi Kanai
Keio University, SFC
kanai@sfc.keio.ac.jp

ABSTRACT

Advances in 3D scanning technologies have enabled the practical creation of hundreds of millions of points. In this paper, we describe a novel image-based point rendering algorithm only using points. Most of previous point rendering algorithms has to prepare normal vectors in advance to establish shading effects. Our algorithm is based on image processing and can calculate normal vectors on the fly in each frame using principal component analysis. Also, our algorithm is familiar with various other image processing algorithms. As an example we demonstrate an interactive color-editing tool for points.

Keywords

Point-Based Rendering, Image Processing, Principal Component Analysis, Color Editing.

1. INTRODUCTION

In recent years, a large number of scanned points can be acquired thanks to the development of scanning technology via 3D range image scanners. On the other hand, geometric processing techniques such as surface reconstruction from range images [Hoppe92] are needed to utilize points in the various Computer Graphics (CG) applications. Surface reconstruction is a laborious process and often requires a try-and-error task to make polygonal surfaces from such points suitable for practical use.

Point-based rendering has been a focus of constant attention to address the above issue. There are several advantages compared to the polygon-based rendering: The image quality of point-based rendering can now be approximately the same as polygon-based rendering whereas the data representation of points keeps compact because faces are not needed to render.

Point-based rendering was firstly introduced by Levoy and Whitted [Levoy85]. In order to achieve the rendering quality as same as polygon-based rendering, a method to fill holes between points is required. One type of approaches for filling holes is to define a rectangle, a circle or an ellipsoid called as *splat* [Rusin00] or *surfel* [Pfist00, Zwick01] for each point. The other type of approaches based on image processing [Gross98, Kawat04a] has been also proposed.

On being related to the last advantage, there are several painting or surface editing approaches. [Agraw95] proposed 3D painting for scanned objects; however the algorithm is used for meshes. [Zwick02] developed an application for editing point set surfaces. In this application, color editing and surface editing for point set surface can be applied at interactive frame rate. [Adams04] proposed another painting method for point set surfaces. In this approach a virtual brush for painting, which is also defined by a point set, is used.

In this paper, we propose a color editing tool by using point based processing technique based on

Short paper proceedings ISBN 80-903100-9-5
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.

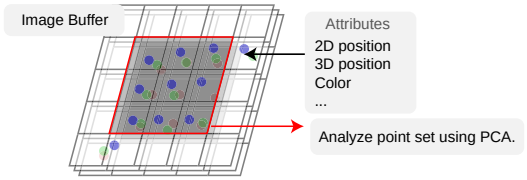

Figure 1. Image-Buffer and affecting range of PCA.

image processing. Our approach can render the image by **only** using position information of points without preparing normal vectors in advance. Our approach is called as Image-Based Point Rendering (IBPR) [Kawat04a].

The paper is organized as follows, firstly in Section 2, we describe an IBPR algorithm used in color editing tool on our approach. In Section 3, we propose color editing tools as an application of our approach. In Section 4, we show results of our approach. In Section 5, we conclude this paper and discuss about our approach.

2. IMAGE-BASED POINT RENDERING

In this section, we describe a brief overview of Image-Based Point Rendering (IBPR) algorithm described in [Kawat04a]. The input data of IBPR is only 3D positions $p_i = (x, y, z)\ (i = 1 \ldots n)$ of points. Also, color information $c_i = (r, g, b, a)$ can be attached in each point if needed. In IBPR an image buffer is provided to render points. We set the resolution of such an image buffer smaller than screen resolution (frame buffer). This approach is similar to the pull-push algorithm proposed by Grossman and Dally [Gross98]. Whereas the pull-push algorithm uses several multi-resolution images for filling holes, we use only one low-resolution image with arranging its resolution.

In the rendering process, we store each 2D coordinates $p_i = (p_i^x, p_i^y)$, color information c_i and a normal vector $n_i = (n_i^x, n_i^y, n_i^z)$ to a corresponding pixel of an image buffer. A normal vector is computed using stored 2D positions p_i in each frame (see [Kawat04a]). We determine the resolution of an image buffer (w_i, h_i) as follows:

$$w_i = w_s / s, h_i = h_s / s, \tag{1}$$

$$w = \frac{zn \cdot zf}{zf - zn} \cdot \omega + 1, \tag{2}$$

$$s = \frac{\sigma}{\tan(fov/2)} \cdot \frac{1}{w} \cdot \lambda.$$

where w_s, h_s are width and height of a frame buffer respectively. s is determined by the resolution of a point set and the distance from a view position to a point ω and a field of view fov. σ is the resolution of point set, λ is width or height, and zn, zf are the distance from a view position to a near plane and to a far plane respectively.

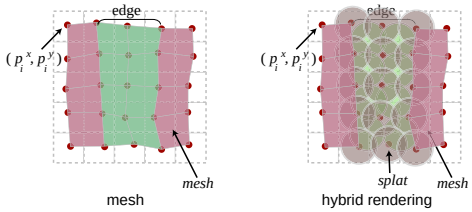

Figure 2. Hybrid rendering.

After all information of points is stored in an image buffer, we compute a normal vector and create a rectangle grid mesh from each pixel and its neighbor pixels only which points are stored. We use these meshes for shading which allow filling holes. Finally, we magnify an image buffer to the size of an original frame buffer for the actual rendering.

We describe a method to compute normal vectors for each pixel of an image buffer using principal component analysis (PCA). We use PCA here for only computing principal directions. In PCA computation process, we need to search a set of points for each pixel and its neighbor pixels. In our approach, we can search neighbor points by referring neighbor pixels of an image buffer efficiently. Figure 1 shows the range of pixels of an image-buffer needed to search neighbor points (multiple points can be stored for each element; these points are used in up-sampling process described later in Section 3).

In original IBPR [Kawat04a], holes are filled using a lower resolution image buffer. However, this process itself causes a blurring effect on a resulting image. We apply a hybrid rendering which both mesh and splat are used as drawing primitives. We describe the technique in Figure 2.

The details of the computation of normal vectors using PCA and hybrid rendering are also described in [Kawat04b].

3. COLOR EDITING

In this section, we propose a color-editing method based on IBPR with some extensions. In our approach, we can use general 2D image processing algorithms, because IBPR is much familiar with these algorithms. Since our approach uses an image buffer, general 2D image processing techniques can be applicable. In this paper, we focus on a stamp tool (Figure 3) for points. In a stamp tool, one region of an image is copied to another region. This tool can be used for the removal of noises of range images.

Figure 3. Stamp tool.

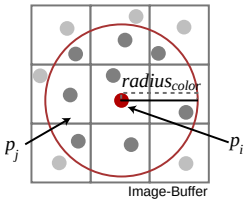

Figure 4. Color transformation.

Editing Process

In IBPR, a stamp tool is executed after all point information is stored to an image buffer (described in Section 2). We use this information for color editing directly. Next, we describe the process.

First, the user defines the source region of the image on screen space. 2D positions p_i and colors c_i included in the specified region are stored to a temporary buffer.

Second, the user defines the destination region of the image on screen space. In this region color information is updated according to the temporary buffer. Figure 4 shows such an updating process. We determine a color of p_i by using the distance between p_i and its neighbor points p_j. We also calculate a weight for each neighbor point by using a distance from p_i. A color is calculated by a weighted sum of colors in neighbor points. This calculation is done for setting a higher priority to a closer point.

We also define the region of interest $radius_{color}$. It is also used for computing a weight. The weight is computed as follows,

$$\omega_j = \frac{|p_j - p_i|}{radius_{color}}. \qquad (3)$$

Then, a color c_i of p_i is determined as follows,

$$c_i = \frac{\sum_{j=1}^{n} c_j \omega_j}{\sum_{j=1}^{n} \omega_j}, \qquad (4)$$

where value of $radius_{color}$ is defined by the user. This process gives a simple way to find neighbor points without any special data structure of point sets.

Up-Sampling

In the above process, if the density of points on the destination region is not enough, the blurring of a resulting image can occur. To address this problem, we apply up-sampling to increase the quality of a resulting image.

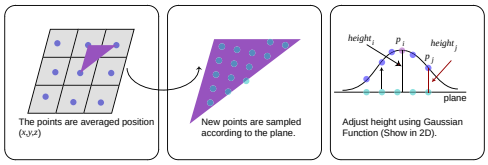

Figure 5. Up sampling process.

Figure 6. Result of up sampling.

Up-sampling is used in most researches about point-based painting. For example in [Adams04], dynamic up-sampling scheme is applied for less density regions. We use a simple method to improve the quality of destination regions in a stamp tool. In this method neighbor point regions are used to calculate new sampling points. Such regions are given by an image buffer. Our up-sampling method is described as follows:

1. We define a sampling plane based on neighbor points (Figure 5 left). The plane is defined by three positions which are taken from an image buffer.
2. Next, we calculate an interval of new sample points. Based on this interval, new points are sampled on this plane (Figure 5 middle).
3. We adjust a height from the plane of each new sampled point according to source 3D points (the positions of original 3D points) (Figure 5 right). Source 3D points are stored to an image buffer in advance. We also compute a distance $height_i$ from the plane to each source 3D point. We use a Gaussian function to determine the weight of source 3D point. A position p'_j of new sampled points are determined as follows:

$$r(d) = e^{-d^2/h^2}, \qquad (5)$$
$$p'_j = n \cdot \{r(|p_i - p_j| / range) \cdot height_i\},$$

where d is a distance from a source 3D point. h is a user-specified parameter and is set to 0.2. $range$ is also a parameter and is set according to an interval of up sampling. A direction n is a normal vector of the plane.

Figure 6 shows the comparison of resulting images between up-sampling and the original approach (up-sampling is not used). It can be seen that the pasted image is clearly displayed compared to the original approach.

(a) (b)

Figure 7. The results of stamp tool.

4. RESULTS

In this section, we show the results of our approach. We evaluate our approach on Pentium 4 3.2 GHz CPU and 1GB RAM PC. For the experiment we use an original range image of Stanford Bunny (362,272 points). Figure 3 shows the result of our point-based rendering with attached color attribute and the right upper of Figure 3 show a magnified image of the result with our stamp tool. The rendering time is 1.11 seconds.

Next, we show the results of color editing tool. We used Beetle range images with color information (559,327 points) for the experiment. We edit the regions of noises appeared in range images by pasting colors of other regions. Figure 7 (a) shows rendering results before editing operations with different view directions. Figure 7 (b) shows the results after editing operations. On the left upper of each figure, a magnified image to a modified region is shown. From these results, it can be seen that the regions of noises are corrected and are smoothly rendered.

5. CONCLUDING REMARKS

In this paper, we have described IBPR and its application to color editing tool. Our approach for point-based rendering is familiar with various 2D image processing algorithms. As one of examples, we have shown that a stamping tool can be efficiently implemented on our point-based rendering scheme. However, the rendering time is slow, but we can imagine that an image-processing part of our algorithms can be easily ported to a fragment program of GPU, which dramatically improves the rendering speed.

We have also demonstrated a stamp tool as an application to our IBPR. We think that we can implement other 2D image processing tools and also 3D geometry editing tools in the near future work.

ACKNOWLEDGMENTS

Stanford Bunny range images are courtesy of Stanford University Computer Graphics Laboratory.

REFERENCES

[Agraw95] M. Agrawala, A. C. Beers and M. Levoy. 3D painting on scanned surfaces. In Proc. 1995 Symposium on Interactive 3D Graphics, pp. 145-150, 1995.

[Adams04] B. Adams, M. Wicke, P. Dutré, M. Gross, M. Pauly and M. Teschner. Interactive 3D painting on point-sampled objects. In Proc. Eurographics Symposium on Point Based Graphics 2004, pp. 57-66, 2004.

[Gross98] J. Grossman and W. J. Dally. Point sample rendering. In Proc. Eurographics Workshop on Rendering 98, pp.181-192, 1998.

[Hoppe92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In Proc. SIGGRAPH 92, pp. 71–78, 1992.

[Kawat04a] H. Kawata and T. Kanai. Image-based point rendering for multiple range images. In Proc. 2nd International Conference on Information Technology & Applications (ICITA 2004), pp. 478–483, 2004

[Kawat04b] H. Kawata, A. Gouaillard and T. Kanai. Point-based painterly rendering. In Proc. 2nd International Conference on Cyber Worlds 2004, 2004.

[Levoy85] M. Levoy and T. Whitted. The use of points as a display primitive. Technical Report 85-022, Computer Science Department, University of North Carolina at Chapel Hill, 1985.

[Pfist00] H. Pfister, M. Zwicker, J. van Baar and M. Gross. Surfels: Surface elements as rendering primitives, Proc. ACM SIGGRAPH 2000, pp.335-342, 2000.

[Rusin00] S. Rusinkiewicz and M. Levoy. Qsplat: A multiresolution point rendering system for large meshes. In Proc. ACM SIGGRAPH 2000, pp.343-352, 2000.

[Zwick01] M. Zwicker, H. Pfister, J. van Baar and M. Gross. Surface splatting. In Proc. ACM SIGGRAPH 2001, pp. 371-378, 2001.

[Zwick02] M. Zwicker, M. Pauly, O. Knoll and M. Gross. Pointshop3D: an interactive system for point-based surface editing. In Proc. ACM SIGGRAPH 2002, pp.322-329, 20