Interactive Point-Based Painterly Rendering

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

Alexandre Gouaillard
Creatis Laboratory, Insa de Lyon
69621 Villeurbanne, France
alexandre.gouaillard@insa-lyon.fr

Takashi Kanai
Keio University, SFC
5322 Endo, Fujisawa-city, Kanagawa, Japan.
kanai@sfc.keio.ac.jp

Abstract

Non-photo realistic rendering (NPR) methods have been proposed for more than ten years in the Computer Graphic field. Separately, within past few years, several algorithms for rendering surfaces from their points, or point sets directly, were proposed. But so far, no NPR like painterly rendering has been proposed for point sets. In this paper, we propose an interactive point-based painterly rendering algorithm. The Interactive frame rate is achieved by using both a point-based approach to represent the geometry of the surface and an image-based approach for the rendering. Our algorithm achieves interactive rates and outperforms mesh-based previously reported results. A by-product of our work, we also provide a faster, higher quality IBPR algorithm. We have showed that using most sophisticated statistical approach, we could improve the rendering quality.

Index Terms–Point-Based Rendering, Non-photo realistic rendering.

1 Introduction

Non-photo realistic rendering (NPR) methods have been proposed for more than ten years in the Computer Graphic field. They enable the utilization of several artistic effects when visualizing images or 3D scenes. Separately, within past few years, several algorithms for rendering surfaces from their points, or point sets directly, were proposed. Those Point-Based Rendering techniques (PBR) allow one to visualize bigger models at interactive rates without loss of quality. They also provide a way to visualize data when connectivity is not provided. This is the case in reverse engineering for example. Finally, image-based rendering techniques are presenting great interest as they represent a natural LOD and view-dependent optimization of any rendering process. In [7], the authors proposed an Image-Based Point Rendering (IBPR) algorithm which achieved interactive frame rate on several hundred of thousands of points. Their approach was designed for range-images, but it is suitable for any PBR.

In this paper, we propose an interactive point-based painterly rendering algorithm. The interactive frame rate is achieved by using both a point-based approach to represent the geometry of the surface and an image-based approach for the rendering. Our contributions can be summarized as follows:

1. We propose an enhanced Image-Based Point Rendering engine (E-IBPR) providing same rendering quality as IBPR [7] with only one pass.

2. We propose a PCA core algorithm that provides more accurate parameters (normals of surface and orientation of brushes...) at different levels of the rendering.

3. We propose a point-based painterly rendering algorithm.

The paper is organized as follows: first, in section 2 we will present more in details the state of the art in NPR and PBR. In section 3, we will introduce our new painterly rendering algorithm as a whole. It is based on an enhanced IBPR (E-IBPR) that we are going to introduce in section 4. A new hybrid (mesh + splat) rendering algorithm is proposed that directly reduces the blurring effect seen with one-pass IBPR without requiring a second pass. Section 5 will then details the computation of several features using Principal Component Analysis (PCA) instead of usual averaging formula. PCA is applied on the neighborhoods defined

in the E-IBPR. This definition allows one to compute view-dependent PCA of point sets. It overcomes the time limitations encounter by the author's work in [5] (using octree). It also computes much accurately several parameters of the algorithm like normals and orientation of splat in E-IBPR. Painterly rendering itself is presented in section 6. Some results are shown and we will finally conclude in section 8.

2 Related Work

Several NPR approaches exists for both 2D images and 3D geometry. In [4], the author first proposed painterly rendering for 2D image and 3D geometry. They define "Brush" as patterns used to modify the appearance of the rendered image. Depending on the kind of Brush and on the number of strokes the user want to apply, the rendered image has a different aspect. Here, the quality of the image is of no interest, only the effect is important. For 3D geometry, the Brush's orientation is determined by normal. This approach is well suited for user interaction. In [14], the authors proposed a painterly rendering algorithm for 2D images. Their approach focuses on the automation of the rendering. The algorithm changes the size, the orientation and the color of the brush depending on the original image. The brushes are applied layer by layer. First the big areas are covered with big strokes, and then the details are added iteratively using brushes of decreasing size. No corresponding algorithm is provided for 3D geometry. In [10], the authors proposed a painterly rendering algorithm for 3D geometry aiming at animation. They use a particle approach to achieve the painterly rendering. The surface is randomly sampled into particles. They use the particles as support for the geometry. Their work focuses on the aspect continuity of surfaces between frames of the animation. In [8] and [6], the authors use "Geograftal" for NPR rendering. They could achieve several artistic rendering effects that include painterly rendering, and this approach is suited for edit by user. In the most recent work on 3D painterly rendering [10, 8, 6], all the authors are using particles to achieve high quality painterly rendering. When the surface is available, it needs to be sampled to get particles. Recently the meshes are (too) densely sampled. Taking the points directly as particles without resampling the surface makes sense as several points are already contained in the volume covered by one pixel of screen space.

Levoy and Whitted proposed first to use points as rendering primitive [9]. Grossman and Dally [3] proposed a PBR using image-based approach. Using pull-push algorithm they achieved faster PBR, but the quality is still improvable. Plister et al. [11] and Zwicker et al. [15] proposed high-quality PBR. In [11], the authors proposed to use surface elements (surfels) at the position of points to improve the quality of the rendering. Surfels can be seen as small primitives (oriented squares or circles) approximating the surface locally. In [15], the authors extend the notion of surfel to the notion of Splat. Splats can be seen as surfels whose parameters (position, color, and orientation) would be more accurate thanks to the usage of Elliptical Weighted Average (EWA) in screen space. The representation can also be made hole-free. QSplat [12] is a more efficient Splatting algorithm. This approach is using multi-resolution data structure and rectangular or ellipsoidal Splats. In [1], the authors proposed a hardware implementation of a Splat rendering. In [5] the authors proposed a new PBR which allow multiresolution rendering with a very high quality. Their method is unfortunately reported to be very slow.

3 Point-Based Painterly Rendering

In this section we are going to provide an overview of the complete algorithm. The rendering pipeline is made of mainly three parts. A first rendering pass using an enhanced version of IBPR [7] compute a first hole-free representation of the surface. Our experiments showed that point based painterly rendering cannot achieve hole free representation directly. Although that can be interesting for some applications, it is not desirable in our case. The advantages of using E-IBPR are numerous: we only need geometry information as opposed to connectivity and normal information. If the original geometry is disturbed by acquisition noise, the noise is automatically reduced. We can, but are not obliged to, use additional color information coming from texture mapping for example. It's fast. It's possible to add alpha blending. The details of E-IBPR are provided in next section.

It relies on a PCA core algorithm for the computation of the normal at each pixel. PCA relies on E-IBPR computed "clusters" for the analysis. The details of this symbiotic relation between E-IBPR and PCA will be explained in details farther in the paper. PCA also not only compute the normal, but also the principal directions in the tangential plane. The results of the first pass (mainly position in 3D space and color) and the results of PCA (global orientation) are used by a second rendering pass to achieved various painterly effects.

Figure 1 illustrates the process of our painterly rendering in term of rendering passes. Figure 2 illustrates the flowchart of the algorithm.

4 Enhance Image-Based Point Rendering (E-IBPR)

In this section, we describe enhanced image-based point rendering (E-IBPR). In previous approach the authors provided a framework for computing Image-based point ren-

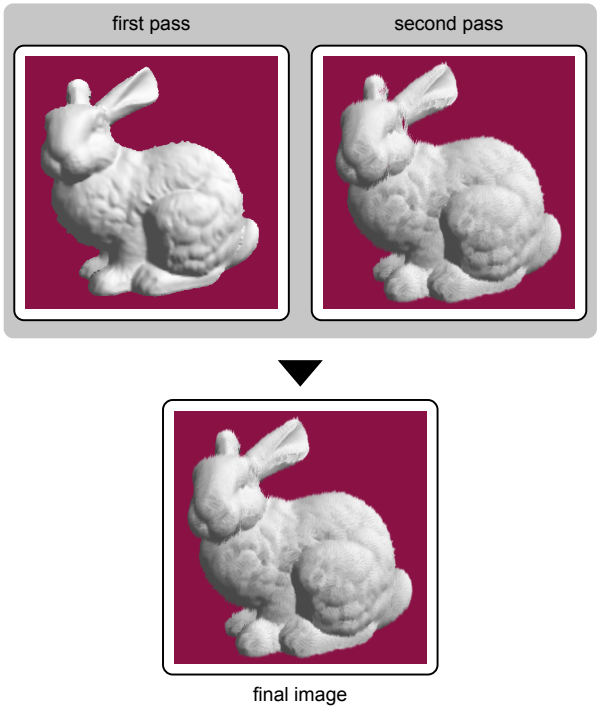


Figure 1. Process of painterly rendering using IBPR.

dering. For each frame, their rendering algorithm can be sketched as follows (for more details, refer to [7]):

1. Compute the size of an image buffer, allocate the memory.
2. Store original points information to the allocated image buffer.
3. Compute normals and colors for shading.
4. Create faces and magnify the image buffer to the size of the screen buffer.

Specifically, their definition of point clusters from the point of view allows one to render point sets in a fast, hole-free way. The definition of those clusters used as interface between original point set and for final pixels rendering is illustrated in Figure 3. As important parameter is the buffer size / screen size rate s that is computed as follows:

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

where 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. We have set γ as proposed in [7] to 1-3 times the sampling interval of points. Their approach suffers of mainly two limitations.

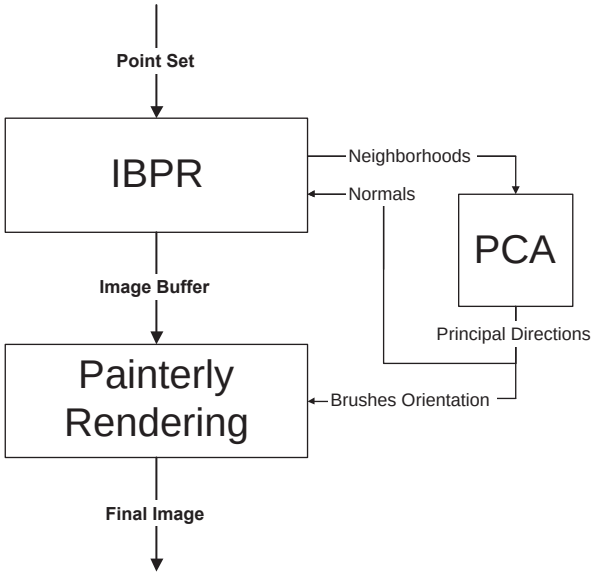


Figure 2. Overview of our algolithm.

First, a blurring effect occurs that is only reduced if using an extra pass. Then, the computation of parameters for rendering, especially the normals, is of medium quality. This is a big limitation as the resulting image is very sensible to small changes in the normal direction.

In this paper we propose one solution for each one of the above cited problems. We propose a hybrid rendering using two primitives (mesh and splat) to reduce the blurring effect directly. We then only require one pass to achieve the same quality as the previous two passes results. We also propose to use a more accurate statistical tool: Principal Components Analysis (PCA) to compute more accurately the normals. The PCA core algorithm results are of further interest than just the normal direction. They are also used during the final painterly rendering. We will explain in detail our usage of PCA later in the paper.

The proposed hybrid rendering is using mesh and splat to render the underlying surface. Mesh is used as a primitive everywhere but on ”edges” where splats are used. We define ”edges” as regions where the normal is perpendicular to the view vector. Note that the normal direction thus is of prior importance for the quality of the resulting image. That is yet another reason why we needed a more accurate computation of the normals. This definition of the edges includes the regions that would be edges in the reconstructed surface as well as the silhouette. We are improving the rendering of both of them. As illustrated in Figure 4, on ”edges” we are placing one splat at each pixel and one splat in the middle of the pixel. The first four (respectively two splats if silhouette) use pixel’s attributes as explained in previous work [7]. The only difference is that we are forcing the splats to be behind the mesh by increasing their depth. The splat in

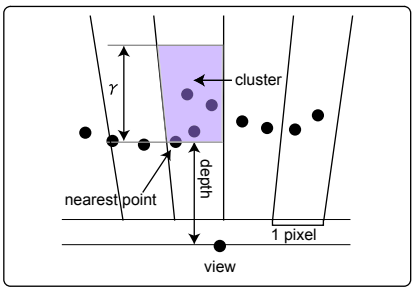


Figure 3. The geometric meaning of γ.

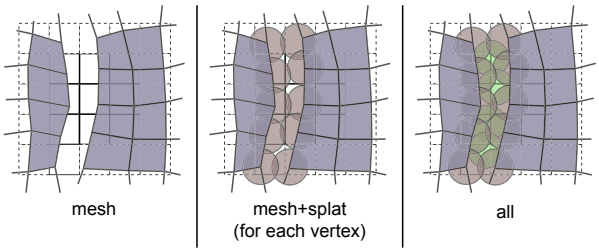


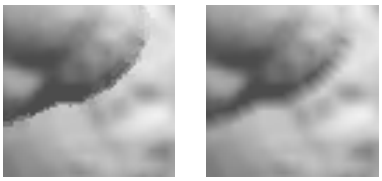
Figure 4. example of drawing by hybrid primitive.

the middle is using an average of its 4 neighbors for each attribute. We are forcing this one to be behind its four neighbors. Figure 5 compares the rendering result using IBPR and our E-IBPR. The speed itself is also increased, but as PCA computation requires more time than averaging, the frame rate using E-IBPR is approximately the same as with IBPR.

5 Principal Component Analysis Usage

PCA is a statistical tool that provides good ways to approximates sets. The PCA analysis of a collection of N points in a d-dimensional space gives us the mean ν, an orthogonal basis f , and the variance σ of the data [2] . The terms ν and σ are d-dimensional vectors and we refer to their i-th scalar value as ν^i and σ^i respectively, with $\sigma^i \geq \sigma^j$ if $i > j$. The basis f consists of (atmost) d vectors with the i-th vector referred to as f^i. Intuitively, we are going to represent a set a point by an "entity" that shares the same statistical properties [2]. More explicitally, the PCA analysis of a group of points results in an estimate of their local orientation frame, the mean and the variance along the different axes of the coordinate frame.

In the work that motivates our usage [5], they keep only the PCA results and regenerate point sets on demand. In our case, we focus on determining the principal orientations. We are using PCA to improve the quality of our rendering and to compute the orientation of our brushes in 3D space.



(a) (b)

Figure 5. Comparison of the quality of results. (a) hybrid rendering, (b) mesh rending only.

PCA is the good tool in our case as it provides directly the three vectors we need throughout the two pass rendering (one for E-IBPR, and the three of them for painterly rendering). We didn't investigate the compression part, as we are doing view-dependent clustering on-the-fly. We thus avoid building a complex data structure and traversing it. However, we share with the previous work common motivations for PCA usage. We noted that there is a very high coherence of points within a cluster. They also noted that "the accuracy required to generate a visually realistic image from a point cloud could be achieved using statistical methods on a sparse point representation" even if the statistical method trade off accuracy against determinism. We are taking this argument a little bit further using an IBPR algorithm. Intuitively, only the features that are going to be visible at the given screen resolution will be taken into account. It allows us to achieve a high quality rendering of point based surfaces at interactive rate.

It shall be noticed that even if it was not our original goal, this represents a good by-product of our work. This rendering technique combines both advantages from IBPR and PCA:

- view dependent
- Output optimal
- Multiresolution (changing resolution of the image buffer used)
- Interactive speed
- Accuracy of the result
- Anisotropy of the result (very important in painterly rendering for example).

Finally, we show the result of PCA in Figure 6, and we show the two images which are previous our normal generation approach and PCA approach result. The PCA approach could decrease the noise of point set.

(a) (b)

Figure 6. (a): The result using PCA, (b): the result using the previous approach.

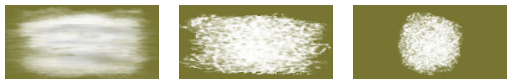
6 Painterly Rendering

Intuitively, brushes can be seen as oriented "textured splats". The painterly rendering algorithm we use as a second pass can be sketched as follows for each pixel in screen space :

1. Get corresponding 3D position,
2. Get color,
3. Get orientation of the brush,
4. Get size of the brush,
5. Compute the "3D stroke",
6. Add the 3D stroke to the scene to be rendered.

Once every pixels have been processed, we render the final scene.

To achieve painterly rendering we need the following items for each pixel: position in 3D space, orientation of normal and color. This is provided by the first pass. We then need the following informations to define the strokes: brush pattern, brush size, position in 3D space and orientation. The brush pattern and size are chosen by the user. The position in 3D space is the position of the pixel currently processed. Finally the orientation in 3D space is given by PCA. Note that we are not using the eigen values here. The original brush pattern is thus not stretched in 3D space. It will be stretched when transformed into the screen space, but this is not related to any anisotropy of the underlying surface, at least in this version of the implementation. In previous 3D approaches the computation of the orientation was less accurate (because of particle computation [6], [10]) and / or needed heavy image processing [14],[13].



(a) (b) (c)

Figure 7. Brush images.

7 Results and Discussion

All the reported results were done on Pentium 4 3.20GHz with 1GB RAM machine. Currently the algorithm has only been implemented using C++ language and DirectX primitives. We ran our algorithm against Stanford Bunny for all the main explanations of this paper. We also used bigger and more challenging models to prove the efficiency of our method. Those models are Stanford Happy Buddha and Stag Beetle model. In Figure 7, we show brushes that we used for our painterly rendering.

Figure 8 illustrates the difference between IBPR (a),(e) and painterly rendering(b)-(d), (f)-(h) using different brushes. Top row uses only the point set, bottom row uses color information from texture mapping. Figure 9 illustrates the results obtained using Stag Beetle model, and Figure 10 illustrates the results of obtained using Happy Buddha model.

The computation times are reported in the following table. The screen resolution was 512 by 512 pixel. The value of s, as defined in section 4 is also provided.

model	s	points	effect	time(sec)
Bunny	2.72	362,272	normal	1.11
Bunny	2.72	362,272	painterly	2.31
Beetle	1.91	559,327	normal	1.03
Beetle	1.91	559,327	painterly	2.03
Buddha	1.67	1,274,573	normal	2.11
Buddha	1.67	1,274,573	painterly	4.59

These timed results compares well with [6] in which the authors reported 425,700 vertices at 1.44 fps on a professional workstation. Especially our results do not depend on the input number of points, while [6] report linear dependence on the number of points.

8 Conclusion and future work

We proposed a new painterly rendering algorithm for point sets. This algorithm achieves interactive rates and outperforms previously reported results. As a by-product of our work, we also provide a fast, high quality IBPR algorithm. We have showed that using most sophisticated statistical approach, we could improve the rendering quality.

In the future, we would be interested in investigating the usage of PCA values directly to reduce the needed bandwith. We think there is an opportunity there to improve the

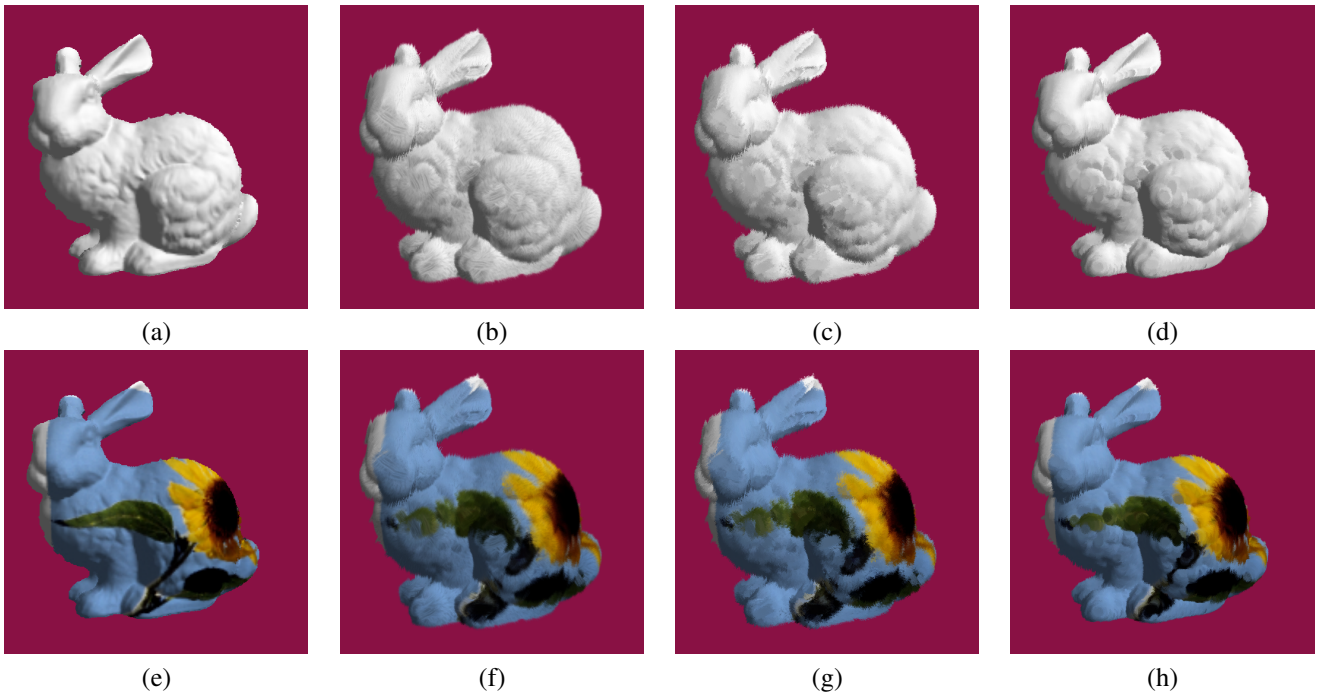


Figure 8. Results of painterly rendering (Bunny). (a) : normal rendering. (b) : brush a, (c) : brush b, (d) : brush c. (e): normal rendering with color information from texture. (f)-(h): painterly rendering of (e) using the same brushs as (a)-(c).

maximum size of data processed and the speed of the rendering. We are also thinking about proposing an implementation of the algorithm using graphics hardware to increase the speed. The final goal being to let the user interact with the model in real-time, which is not possible with our current software implementation.

Acknowledgements

We would like to thank Prof. Kenji Kohiyama for providing Stag Beetle model.

References

[1] L. Coconu and H.-C. Hege. Hardware-accelerated point-based rendering of complex scenes. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 43–52. Eurographics Association, 2002.

[2] D. Ebert, F. Musgrave, P. Peachey, K. Perlin, and S. Worley. Texturing and modeling: A procedural approach. San Diego, 3rd edition, 2002. AP Professional.

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

[4] P. Haeberli. Paint by numbers: abstract image representations. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 207–214. ACM Press, 1990.

[5] A. Kalaiah and A. Varshney. Statistical point geometry. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 107–115. Eurographics Association, 2003.

[6] M. Kaplan, B. Gooch, and E. Cohen. Interactive artistic rendering. In *Proceedings of the first international symposium on Non-photorealistic animation and rendering*, pages 67–74. ACM Press, 2000.

[7] H. Kawata and T. Kanai. Image-Based Point Rendering for Multiple-Range Images. In *ICITA 2004*, 2004.

[8] M. A. Kowalski, L. Markosian, J. D. Northrup, L. Bourdev, R. Barzel, L. S. Holden, and J. F. Hughes. Art-based rendering of fur, grass, and trees. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 433–438. ACM Press/Addison-Wesley Publishing Co., 1999.

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

[10] B. J. Meier. Painterly rendering for animation. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 477–484. ACM Press, 1996.

[11] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: surface elements as rendering primitives. In *Computer*

(a)

(b)

Figure 9. Result of painterly rendering (Stag Beetle). (a):E-IBPR rendering, (b):painterly rendering.

Graphics (Proc. SIGGRAPH 2000), pages 335–342. ACM Press, New York, 2000.

[12] 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.

[13] T. Saito and T. Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 197–206. ACM Press, 1990.

[14] M. Shiraishi and Y. Yamaguchi. An algorithm for automatic painterly rendering based on local source image approximation. In *Proceedings of the first international symposium on Non-photorealistic animation and rendering*, pages 53–58. ACM Press, 2000.

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

(a) (b)

Figure 10. Result of painterly rendering (Happy Buddha). (a) : E-IBPR rendering. (b) : painterly rendering.