A RENDERING ALGORITHM FOR HYBRID SCENE REPRESENTATION

Yen Tien

Ins. of Multimedia Engineering
National Chiao Tung University
Hsinchu, Taiwan (ROC)
southp0105@gmail.com

Yun-Fung Chou

Ins. of Multimedia Engineering
National Chiao Tung University
Hsinchu, Taiwan (ROC)
yfchou@cs.nctu.edu.tw

Zen-Chung Shih

Dep. of Computer Science
National Chiao Tung University
Hsinchu 300, Taiwan (ROC)
zcshih@cs.nctu.edu.tw

ABSTRACT

In this paper, we discuss two fundamental issues of hybrid scene representation: constructing and rendering. A hybrid scene consists of triangular meshes and point-set models. Consider the maturity of modeling techniques of triangular meshes, we suggest that generate a point-set model from a triangular mesh might be an easier and more economical way. We improve stratified sampling by introducing the concept of priority. Our method has the flexibility that one may easily change the importance criteria by substituting priority functions. While many works were devoted to blend rendering results of point and triangle, our work tries to render point-set models and triangular meshes as individuals. We propose a novel way to eliminate depth occlusion artifacts and to texture a point-set model. Finally, we implement our rendering algorithm with the new features of the shader model 4.0 and turns out to be easily integrated with existing rendering techniques for triangular meshes.

Keywords: computer graphics, mesh sampling, object-space EWA splatting, point-based model, point set

1. INTRODUCTION

Point-set model is one of the most widely concerned geometric representations. For its conceptual simplicity, unstructured nature and ease of data maintenance, its possibilities have been extensively studied for decades.

One of the most significant discussions is to combine the advantages of triangular mesh and point-set model. Since triangles can better capture the flat area and sharp features of a surface while points do better on the complex part, many works were done in mixing both representations. POP system [6], sequential point trees [8], and Cocunu L. and Hege H. C. [7] construct LOD representation and render triangles when it is a faster option. All these works blend the rendering result of triangles and of points to create a smooth transition. Guennebaud G. and Gross M. [10] further discuss the blending issues in EWA splatting algorithm. However, they did not consider the issues when blending is not desired, e.g. triangular meshes and point-set models are different objects.

In this paper, we focus on the hybrid scene which triangular meshes and point-set models are individuals. We visit the following issues:

Fig. 1: Rendering results of our techniques. The top row shows a triangular-mesh man wearing a point-set clothes. The bottom row shows two Stanford bunnies with different textures.

How to construct a hybrid scene?

Because of the long dominating history and development of related techniques, there are plenty of sources of triangular meshes. In contrast, an intuitive way to obtain a point-set model might be through a 3D scanner, which is not available all the time. Considering the popularity of packages of triangular mesh modeling tools, we propose that generating point-set models via sampling a triangular mesh seems to be both reasonable and economical way. In this paper, we propose a priority-based sampling algorithm to convert a triangular mesh to a point-set model. After sampling, we further generate the corresponding splat representation with a modified version of [28], since we render a point-set model with splatting algorithm.

How to render a hybrid scene?

In this issue, we propose several basic policies: the algorithm must be easily integrated to existing triangular mesh rendering algorithm with reasonable performance. Since the great work by Zwicker M., et. al. [29], EWA

splatting becomes one of the most popular ways to render a point-set model because of its superior quality. The original work of M. Zwicker et. al. [29] presented a software implementation. Many works were done to investigate the power of graphics hardware since then [3], [4], [5], [22]. Our algorithm follows the same spirits. We propose a novel way to implement EWA splatting based on shader model 4.0 with DirectX 10. Consequently, our system is guaranteed to be easily integrated into existing triangular mesh rendering system.

In the following sections, we first introduce related previous works of techniques related to point-set models in section 2. In section 3, we describe the sampling and the splat generation algorithm that we use to produce point-set model. Our rendering algorithm will then be described in section 4 in detail. Finally, we show our results and conclusions in chapters 5 and 6, respectively.

2. RELATED WORKS

Using points as primitives was first proposed by Levoy and Whitted [15]. Because of the development of technology of range scanners and the conceptual simplicity of a point, many works were devoted to this field since then. However, efficient rendering of point-set models was not possible until the work by Grossman J. P. and Dally W. J. [9]. They developed an image-space surface reconstruction algorithm and made a great step forward in both the rendering performance and quality. Later, QSplat [23] introduced splat with flat-shading quality and multi-resolution data structure to deal with massive point sets.

Alexa M. et. al. [1] introduced the concept of MLS (moving-least-square) fitting with respect to a plane. It soon became the main trend of the surface definition of point set because of its great approximation of the surface and indefinitely differentiability [13], [14]. Recently, Guennebaud G. and Gross M. [11] suggested to define the surface with MLS fitting with respect to algebraic surface to gain more accuracy.

Zwicker M. et. al. [29] presented a pure software implementation of EWA(elliptical-weighted-average) splatting and achieved superior rendering quality and handled transparency correctly. Many works were then devoted to develop an efficient way to implement EWA splatting on graphics hardware [3], [4, [5, [22]. Recently, Weyrich T. et. al. [27] further presented a prototype of graphics adapter for EWA splatting.

The LOD of point-set model was also investigated for efficiency. QSplat [23] organized points as bounding-sphere hierarchy and gained a great performance and memory efficiency via densely encoding node information. For solving depth order and LOD, Zwicker M. et. al. [23] and Pfister H. et. al [21] first introduced layered-depth cube (LDC), which is basically an improved version of layered-depth map [24]. Dachsbacher C. et. al. [8] developed a LOD structure which may process and select level entirely on graphics adapter.

Since point set may represent complex geometry efficiently while triangle may be a better choice for broad flat region and sharp features, hybrid representation of were investigated [6], [7], [8], [10]. They rendered and blended the surface color of triangles and point-set models when it was a faster option. Müller M. et. al. [17] expanded the definition of a splat with clipping lines to render sharp features solely with splatting algorithm.

3. MESH SAMPLING

Our sampling algorithm is basically an improved version of the stratified mesh sampling proposed in [19]. It first converts meshes to voxel approximation by constructing an octree, and then generates a sample point per voxel with respect to a radial function. It is to overcome the drawback that area-based uniform sampling algorithm often failed to spread enough sample point over complex region of the mesh [26]. With stratified sampling, we may ensure better spatial uniformity of sampling points over the whole mesh. Nevertheless, the original algorithm in [19] lacks of importance criteria. Further, it doesn't provide user with the ability to define one's desired sample count with respect to a voxel approximation level. We improve these via introducing priority during sampling and allow user to define the number of sample points in a level.

Figure 2 is an overview of the whole sampling process. By substituting the priority function and the distribution function, our system may change the perspective on importance region totally, while maintaining the spatial uniformity. Thus it can be viewed as a framework, not just an algorithm.

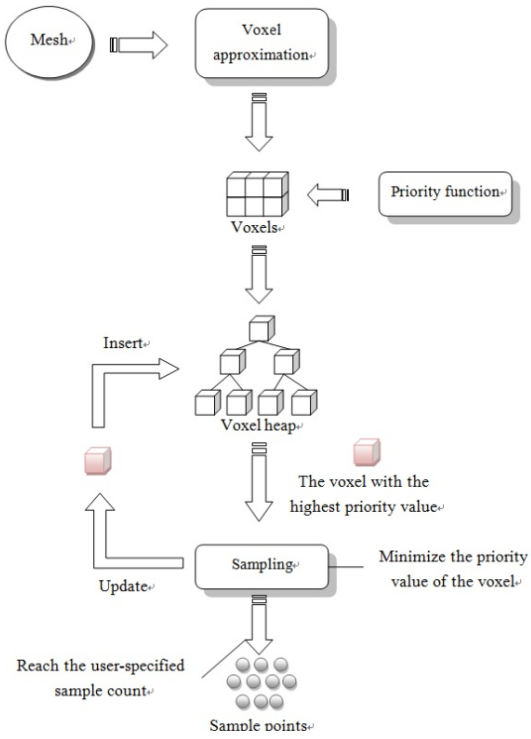

Fig. 2: Overview of the sampling process

In the following sections, we use the term "leaf cell" and

voxel alternatively, since they are the same in this context. In section 3.1, we first describe how we do voxelization. We then present our sampling algorithm and how we define attributes of each sample point in section 3.2.

3.1 Voxel Approximation

We compute the voxel approximation of the input triangular mesh via a top-down octree construction algorithm like [19]. First, the axis-aligned bounding box is computed and is taken as the root cell. Next, we recursively divide the cell with respect to the longest dimension, and store triangles which intersect with the cell. To detect whether a triangle and a cell intersect, we found that the fast triangle-box overlap testing procedure presented by T. Akenine-Mäoller [2] is very efficient and is easily integrated. The recursion stops whether the user-defined depth reached or no triangle is recorded in the cell. After the whole process terminates, each leaf cell contains the following information:

***Position*:** The center position of a voxel.

***Dimensions*:** Since our cell is axis-aligned, these are dimensions in x, y, z axis.

***Triangles*:** Triangles which intersects with the voxel.

***Priority value*:** As its name implies, it defines the importance estimation of a voxel. It is computed by the priority function pre-defined by the system. Our system then spreads the sample points according to this value. We use the number of un-sampled triangles as the default priority function.

3.2 Priority-Based Stratified Sampling

After the above process, we obtain the leaves of the octree as the voxel approximation of an input mesh. Then we spread the user-defined amount of sample points on these leaf cells, and assign attributes to each. In the following paragraph, we examine each stage in detail.

Distribute sample points

Our system allows user to input the number of sample points. We first compute the priority value of each voxel via the priority function. Then, we reorganize these voxels to a heap. With the help of heap, we may easily get the voxel with the highest priority value. Since we hope each sample point contains as much information as possible, our strategy is to minimize the priority value of a voxel after sampling. We then update the priority value and insert the sampled voxel back to the heap. In this way, we may ensure the voxel with higher priority value may produce more sample points than lower one. After the sample count reaches the user-specified value, the process terminates.

Assign attributes

For each point generated in the distribution phase, we first project it onto the surface defined by the triangles recorded in the voxel. The projected point will lie on one of the triangle. In some rare case, it will lie on edges or vertices. We then choose the first encountered one. Next, we compute the barycentric coordinates of the projected point. Finally, we interpolate the attributes with the barycentric coordinates and get the final sample point. A sample point may contain arbitrary information defined or derived on the mesh. In our implementation, each sample point contains position, normal, texture coordinates, and material information. After sampling, we estimate the radius of each point by constructing KNN graph [28].

4. RENDERING

Rendering a point-set model can be viewed in two different perspectives. In the computational-geometry point of view, the surface defined by projecting the point set to the moving-least-square surface defined by the point set itself will be an indefinitely differentiable surface [13], [14]. Thus, any implicit surface rendering techniques can be applied, e.g. ray-casting algorithm. However, it seems to be unpractical when high performance of rendering is significant because of the cost of computing implicit surface. In the view of signal processing, if we take the surface attributes of the input mesh as a spatial signal, then rendering a point-set model becomes a spatial signal reconstruction problem. We now further discuss this perspective.

EWA splatting [29] is a technique with the highest rendering quality so far in our knowledge. It is originated from the work of Heckbert [12], which applying elliptical-weighted-average filter for texture filtering. It assigns an elliptical Gaussian reconstruction filter to each splat, and convolves it with a band-limited filter, which is called the object space EWA filter. If the band-limited filter is again a Gaussian, then the projection on the image plane is still a Gaussian, which is referred to as the image space EWA filter. Projecting and accumulating these reconstruction kernels on the image plane produce the final image output. The whole process may be considered as signal reconstruction in object space or image space. For a complete derivation, we recommend the article by Zwicker M. et. al. [30].

Over the past decade, the occurrence of programmable vertex and pixel shader grab great interests on implementing hardware-accelerated EWA splatting. Many great works were done in facilitating of pixel shader to rasterize EWA filter on screen space [3], [4], [5]. The point sprite like OpenGL point [3] or NV_sprite [5] provide an ideal way to generate enough fragments to rasterize EWA filter. To rasterize the filter, the first step is to discard unnecessary fragment via the inside test,

$$u^2 + v^2 = (\mathbf{u^T} \cdot (\mathbf{q} - \mathbf{c}))^2 + (\mathbf{v^T} \cdot (\mathbf{q} - \mathbf{c}))^2 \leq 1 \qquad (1)$$

where **u** and **v** are tangent coordinates, **c** is the center, and **q** is the input point. With the position and the normal of the splat, the algorithm may then rasterize the shape of the filter correctly. However, since a point sprite is actually a billboard aligned with image plane, depth correction is necessary against incorrect depth occlusion artifacts. Botsch M. and Kobbelt L. [4] first presented an implementation with Gouroud shading quality. Phong splatting [3] soon achieved phong shading quality by

associating a linear normal field with each splat. Botsch M. et. al. [5] introduced the idea of deferred shading in EWA splatting and improved the performance further.

Ren L. et. al. [22] proposed an object-space approach. The idea was to render EWA filter with a quad textured with a unit-Gaussian map. In this way, the perspective transform is automatically accurate and is computed by hardware. The rasterization of EWA filter also did not need any special care since it was done by rasterizing the textured quad. Further, it didn't need depth correction. However, the performance was slower than most of the screen space approach because of the hardware constraints.

No matter object-space or screen-space approach is used, a common issue occurs: how to blend splats contribution? Since each EWA filter is truncated to a finite support, it is reasonable to blend only the splats which deviate in z direction of eye coordinates under some threshold values. I.e. the z-test is not simply 0 or 1 anymore; it contains a small tolerance range and thus the name "fuzzy-z test". Unfortunately, there is no way to implement this fuzzy-z test directly under current graphics hardware since the depth-stencil test stage is not yet programmable. One way to solve this problem is to apply other visibility technique. Layered-depth cube is a common choice and has been widely investigated [4], [21]. The other trend is to introduce a visibility pass to the rendering process [3], [4], [5]. It only generates depth map. One simply "moves" it along the viewing direction the amount of tolerance range [22], and then the traditional depth-stencil test will behave as fuzzy-z test.

Nevertheless, with this fuzzy-z test, rendering a scene with triangular meshes and point-set models as individuals becomes a tricky task. A naïve approach may be render the point-set model in a different render target, and then merge it back according to the depth buffer. However, depth occlusion artifacts occur on the intersection region with this approach. Figure 3 shows a point-set Stanford bunny intersects to a triangular-mesh Utah teapot. These artifacts occur because the depth value of the point-set model is computed from the tangent plane of each splat lied on, not from the surface itself.

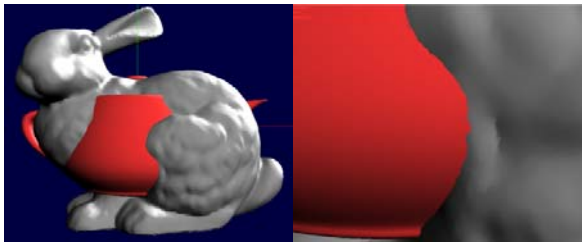

Fig. 3: The depth occlusion artifacts.

Thanks for the new shader 4.0 specifications, we may efficiently construct object-space EWA filter with geometry shader now. We further combine the deferred shading proposed in [5], and implement it with the new feature in shader 4.0 which allows us to render to multiple render targets concurrently in primitive level. We further generalize the attribute pass to deal with depth occlusion artifacts and texturing.

4.1 Pass 1: Visibility Pass

The main goal of this pass is to generate the depth map for the following fuzzy-z test. We first pack the whole splat set into a vertex buffer, and set the primitive type as point list. The vertex shader in this pass transforms the position and tangent coordinates to world space, and then passes the data to the geometry shader. The geometry shader then generates a quad corresponding to each splat, and transforms them to the projection space, as shown in Figure 4.

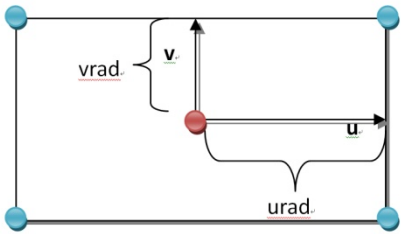

Fig. 4: The quad generated by geometry shader. The red point is the center of the splat, and blue points are points generated by geometry shader.

Before any draw call is made, we set the render target as NULL. Thus we get a depth map in this pass without affecting any previous rendering result in the framebuffer.

4.2 Pass 2: Attribute Pass

In this pass, the vertex shader transforms the position, normal and tangent coordinates to the world space. The geometry shader again expands the point into a quad, then assigns desired attributes in the color channel, and sends them to the correct render target via *SV_RenderTargetArrayIndex* semantic. The depth buffer is set NULL at the beginning and is fed as a shader resource. In the pixel shader, we first do the inside test as equation (3) to discard unnecessary pixels. Although it is an optional step for object-space approach, we found that discard these pixels may increase some performance. Next, it read depth buffer to do fuzzy-z test. Given that a value, z_b, read from the z buffer. Since it is a value defined in the normalized device space, we need to transform it back to the projection space,

$$z = \frac{FN}{F - z_b(F - N)} \qquad (2)$$

where F stands for the far clipping plane, N stands for the near clipping plane. We then use this value to perform the fuzzy-z test.

After processing two tests described above, we then render the quad with the prescribed unit Gaussian map as alpha texture. By using the floating-point precision render target and enabling alpha blending, surface attributes are accumulated and blended correctly in each render target:

$$C(\mathbf{x}) = \sum_i w_i h(\mathbf{x} - \mathbf{x_i}), \alpha(\mathbf{x}) = \sum_i h(\mathbf{x} - \mathbf{x_i}) \quad (3)$$

where $\mathbf{x}$ is the position, $\mathbf{x_i}$ is the splat center, $\mathbf{h}$ is the reconstruction kernel and thus the Gaussian in this paper. C stands for the (R, G, B) channel, and α stands for the alpha channel. We then do per-pixel normalization by dividing the color value with alpha value:

$$C(\mathbf{x}) = \sum_i w_i \frac{h(\mathbf{x} - \mathbf{x_i})}{\sum_i h(\mathbf{x} - \mathbf{x_i})} \quad (4)$$

In the original work of Bostch M. and Kobbelt L. [5], they generated a color map and a normal map in this pass. The color map was basically the blending result of the material color and the diffuse texture. The normal map was the blending result of normal vectors as the name described. Here we further generalize the application of the attribute pass to deal with texturing and eliminate depth occlusion artifacts.

Texturing and depth correction: TexZ map

As mentioned in the first paragraph, the EWA splatting is actually a spatial signal reconstruction process, and the spatial signal can be any surface attributes. Since our point-set model is obtained from sampling a triangular mesh, we may reconstruct the parameterization of the surface by sending the texture coordinates to the attribute pass. Further, we may also consider the depth value of each sample point in the projection space as a surface attribute and reconstruct the depth value of the surface. Our current implementation only considers 2D texture-space parameterization; thus we may pack the 2D parameterization and the projection-space depth value, and render it into one render target. Since it consists of 2D parameterizations for texturing and depth, we name it *TexZ map*.

4.3 Pass 3: Shading Pass

In this pass, we take color map, normal map, depth correction map, and any other possible attribute maps generated in the attribute pass to compute the final result. First, we set geometry shader and the input layout as NULL. In vertex shader, we use the system value: *SV_Vertex_ID* to generate a viewport-sized quad.

For each pixel, we first load the value from TexZ map by screen coordinates and discard it if its alpha value is zero. Of course, this check can be done with any attribute map. Next, for each pixel passing the alpha test, we compute its color value. It is basically *color_value* + *lighting_component*, where *color_value* is fetched from the diffuse map with texture coordinates in TexZ map, and the *lighting_component* is computed via the normal fetched from the normal map. All the value fetching mentioned above uses the intrinsic function Load(). Since they are all viewport-sized, we don't need any filtering. After fetch the value, we do per-pixel normalization as in [5].

Notice that we turn on the output channel to the depth buffer in our pixel shader. Since the z value stores in the TexZ map is in the projection space, we need to transform it to the normalized device space before output:

$$z_b = \frac{F}{F - N}\left(1 - \frac{N}{z}\right) \quad (5)$$

Thus any following rendering techniques will then have proper depth information of our point-set model.

5. RESULTS

Our results were measured and captured on a machine with GeForce 8800GTX card, the version of the driver was 172.20, and the screen resolution was 1024×768. The algorithm was implemented with DirectX SDK ver. March 2008. Our implementation achieves 16M splats/sec in average. Note that we did not apply any LOD technique in our experiment and thus the performance "seems to be" far slower than those pioneer works [3], [4], [5], [29].

Stanford bunnies in Figure 1 show our texturing results. Figure 5 shows the result of our depth correction. With proper depth information, we may try to make a closer interleaving scene without worries like the top row of Figure 1 and Figure 6.

Fig. 5: Before/After depth correction

Fig. 6: Bunnies in grass. The scene consists of 231986 splats and 50553 triangles. Bunnies and rocks are point-set models while grass and carrot are triangular meshes. The scene is rendered at 55 FPS on our platform.

6. CONCLUSIONS AND FUTURE WORKS

In this paper, we try to investigate some of the basic techniques of hybrid scene representation where point-set models and triangular meshes are individuals. We name two fundamental issues: data source of point-set model and

an easily-integrated rendering module. We present priority-based stratified sampling to convert a triangular mesh to a point set. Our priority-based sampling can be viewed as more a framework than simply an algorithm.

We revisit the value of object-space EWA splatting. Our current implementation rendered with a raw data set. Integrating a LOD technique will undoubtedly boost the performance. To not conflict with our basic principle: easily-integrated with exist triangular mesh rendering modules; we anticipate that a LOD technique suits for GPU, like sequential point trees [8], will be an ideal choice.

7. REFERENCES

[1] Alexa M., Behr J., Cohen-Or D., Fleishman S., Levin D., and Silva C. T., "Point Set Surfaces," IEEE Visualization, 2001.

[2] Akenine-Mäoller T., "Fast Triangle-Box Overlap Testing."

[3] Botsch M., Spernat M., and Kobbelt L., "Phong Splatting," Eurographics Symposium on Point-Based Graphics, 2004.

[4] Botsch M., and Kobbelt L., "High-Quality Point-Based Rendering on Modern GPUs," Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, 2003.

[5] Botsch M., Hornung A., Zwicker M., and Kobbelt L., "High-Quality Surface Splatting on Today's GPUs," Eurographics Symposium on Point-Based Graphics, 2005.

[6] Chen B., and Nguyen M. X., "POP: A Hybrid Point and Polygon Rendering System for Large Data," IEEE Visualization, 2001.

[7] Coconu L., and Hege H. C., "Hardware-Accelerated Point-Based Rendering of Complex Scenes," Thirteenth Eurographics Workshop on Rendering, 2002.

[8] Dachsbacher C., Vogelgsang C., and Stamminger M., "Sequential Point Trees," SIGGRAPH 2003.

[9] Grossman J. P. and Dally W. J., "Point Sample Rendering," Proceedings of Eurographics Rendering Workshop '98 page. 181-192, 1998.

[10] Guennebaud G., Barthe L., and Paulin M., "Splat/Mesh Blending, Perspective Rasterization and Transparency for Point-Based Rendering," Eurographics Symposium on Point-Based Graphics, 2006.

[11] Guennebaud G., and Gross M., "Algebraic Point Set Surfaces," ACM SIGGRAPH, 2007.

[12] Heckbert P., "Fundamentals of Texture Mapping and Image Warping," Master's Thesis, University of California at Berkeley, Department of Electrical Engineering and Computer Science. 1989.

[13] Levin D., The Approximation Power of Moving Least-Squares. Mathematics of Computation, Vol. 67, No. 224, October 1998, pages 1517-1531.

[14] Levin D., "Mesh-independent Surface Interpolation," Geometric Modeling for Scientific Visualization. Springer-Verlag, 2003.

[15] Levoy M. and Whitted T., "The Use of Points as Display Primitives," Technical Report TR 85-022, the University of North Carolina at Chapel Hill, Department of Computer Science, 1985.

[16] Marroquim R., Kraus M., and Cavalcanti P. R., "Efficient Point-Based Rendering Using Image Reconstruction," Eurographics Symposium on Point-Based Graphics, 2007.

[17] Müller M., Heidelberger B., Teschner M., and Gross M., "Meshless Deformations Based on Shape Matching," ACM SIGGRAPH, 2005.

[18] Müller M., Keiser R., Nealen A., Pauly M., Gross M., and Alexa M., "Point Based Animation of Elastic, Plastic, and Melting Objects," Eurographics/ACM SIGGRAPH Symposium on Computer Animation, 2004.

[19] Nehab D., and Shilane P., "Stratified Point Sampling of 3D Models," Eurographics Symposium on Point-Based Graphics, 2004.

[20] Pauly M., Keiser R., Kobbelt L. P., and Gross M., "Shape Modeling with Point-Sampled Geometry," ACM SIGGRAPH, 2003.

[21] Pfister H., Zwicker M., Baar J., and Gross M.," Surfels: Surface Elements as Rendering Primitives," ACM SIGGRAPH, 2000.

[22] Ren L., Pfister H., and Zwicker M., "Object Space EWA Surface Splatting: A Hardware Accelerated Approach to High Quality Point Rendering," Eurographics 2002.

[23] Rusinkiewicz S., and Levoy M., "QSplat: A Multiresolution Point Rendering System for Large Meshes," SIGGRAPH, 2000.

[24] Shade J., Gortler S. J., He L., and Szeliski R., "Layered Depth Images," In Computer Graphics, SIGGRAPH 98 Proceedings, pages 231-242. Orlando, FL, July 1998.

[25] Sankaranarayanan J., Samet H., and Varshney A., "A Fast K-Neighborhood Algorithm for Large Point-Clouds," Eurographics Symposium on Point-Based Graphics, 2006.

[26] Turk G., "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion," Computer Graphics, Vol. 25, No. 4, July 1991.

[27] Weyrich T., Heinzle S., Aila T., Fasnacht D. B., Oetiker S., and Botsch M., "A Hardware Architecture for Surface Splatting," ACM SIGGRAPH, 2007.

[28] Wu J., and Kobbelt L., "Optimized Sub-Sampling of Point Sets for Surface Splatting," In Proceedings of Eurographics 04, pages 643-652.

[29] Zwicker M., Pfister H., Baar J., and Gross M., "Surface Splatting," ACM SIGGRAPH, 2001.

[30] Zwicker M., Pfister H., Baar J., and Gross M., "EWA Splatting," IEEE Transactions on Visualization and Computer Graphics, Vol. 8, No. 3, July-September 2002.