A Hardware Accelerated Approach for Accurate Surface Splatting

Weihua An and Bingfeng Zhou

Institute of Computer Science and Technology, Peking University, Beijing, China

Abstract. Elliptical weight average(EWA) surface splatting provides high quality rendering of complex point based models. But its software based implementation can not reach real-time rendering. In order to improve its performance, this paper presents a GPU based accelerated approach, which is superior both in the rendering performance and visual quality. This approach uses the OpenGL point primitive to approximate the splat. Through analyzing the OpenGL view frustum, we deduce an equivalent projective transformation, and therefore form a uniform format to represent the EWA surface splatting based on both software and hardware implementation. According to the projective transformation, we can accurately correct each splat depth. Our experiment results have shown that our approach can render to 8M EWA filtered splats.

1 Introduction

For the connectivity and simplicity, the triangle meshes are widely adopted to represent the 3D models. However, with the increasing requirements for geometry accuracy, the triangle meshes become more and more dense. Sometimes, one or several meshes are corresponding to one pixel on the screen[1]. This huge scale data bring serious challenges to the rendering performance[2].

In contrast, the sampled point sets have superior capability to represent sophisticated models. This representation defines 3D surfaces as a series of points, and it need not maintain the connectivity and topology information. Another advantage is that the point sets can be easily organized into a level-of-detail(LOD)[3] data structure, which enables the multi-resolution rendering. However, the main challenges for point based rendering are to identify the visibility, and to fill the holes on the screen[4].

Nowadays, many rendering methods have been proposed[5][6][7]. Among them, the surface splatting technology proposed by Zwicher et al[8] is acknowledged as the best method, which not only solves the visibility and holes problem, but also avoids aliasing artifacts. However, the original version of this method is implemented by un-optimized software, and can not reach the real-time rendering. Based on this method, some accelerated measures are proposed. In order to improve the rendering speed, they degrade the visual quality to some extent.

This paper presents a GPU based rendering approach, which employs the OpenGL point primitive to implement the surface splatting. The most important

contribution is to deduce an equivalent projective transform formula from the view frustum, which leads to a uniform format to represent the EWA surface splatting based on both software and hardware implementation. According to the projective transformation, we can also accurately obtain every splat's shape and depth values, which benefit identifying the visibility.

2 Previous Works

The concept of representing objects as a series of points was first proposed by Levoy and Whitted[4]. In their work, they discussed some fundamental problems such as surface reconstruction and visibility. Based on this work, many point based rendering technologies have been proposed. For example, Grossman and Dally[9] presented a pull-push algorithm to fill the hoses on the screen, which is prone to the aliasing artifacts. Pajarola[3] organized the point sets as a LOD data structure, and adopted quantized normal representation, which benefits multi-resolution rendering.

Recently, Ewicker et al[8] presented the EWA surface splatting technique, which not only solves the visibility and holes problem, but also avoids aliasing artifacts. Because the EWA filter involves quite complex computation, this method can not reach real-time rendering.

All of the above software methods can not satisfy the real-time requirement. In order to improve the rendering speed, Rusikiwicz and Levoy[10] firstly employed graphics hardware to accelerate point based rendering. Considering the large scale datasets of Digital Michelangelo Project[2], they used a hierarchy of bounding spheres to organize the point sets. They also proposed a two-pass rendering approach to blend the overlapped splats, which is adopted by many other accelerated methods.

Stamminger et al[11] improved the rendering performance to 5M surface splats per second, and Botsch's accelerated method[12] can reach the speed of 10M splats per second. However, all of them didn't consider the aliasing problem.

Ren et al[13] implemented object space EWA surface splatting, which renders every point as a textured rectangle in the object space. Since the data amount is increased by a factor of 4, the rendering performance isn't improved remarkably.

In contrast, some methods used OpenGL point primitive to approximate the EWA filter, and didn't increasing the data sets. Among these methods[14][15], the depth correction for every pixel is not accurate, which causes negative effects on visibility identification.

3 Screen Space EWA Surface Splatting

In this section, we briefly review the surface splatting theory, which is discussed in detail in the Zwicker's paper[8]. The surface splatting is composed of two components: the object space reconstruction kernel and the screen space band-limited filter. The former is used to reconstruct a continuous surface from discrete points, and the latter is adopted to avoid the aliasing artifacts.

We use $\{\mathbf{P}_k\}$ to represent a set of discrete points, which are sampled on a continuous 3D surface. For each point, the coefficient w_k denotes its color attributes. From this point set, we define a continuous texture function on the corresponding surface, which is the reconstruction kernel. For a position $\mathbf{Q}$ on the surface, we construct a local coordinate system in its small neighborhood. Therefore, the position $\mathbf{Q}$ and the point set $\{\mathbf{P}_k\}$ have local coordinates $\mathbf{u}$ and $\mathbf{u}_k$, respectively. Then, the continuous texture function $f_c(\mathbf{u})$ can be expressed as:

$$f_c(\mathbf{u}) = \sum_{k \in N} w_k r_k(\mathbf{u} - \mathbf{u}_k) \tag{1}$$

where r_k is the basis function for the point $\mathbf{P}_k$ in the local coordinate system.

To render an object, the texture function $f_c(\mathbf{u})$ should be warped to screen space. Because the sampling frequencies in the object space and the screen space are different, a band-limited filter is required to eliminate the aliasing artifacts. Therefore, for a coordinate $\mathbf{x}$ on the screen, the texture function $g_c(\mathbf{x})$ can be expressed as:

$$g_c(\mathbf{x}) = \sum_{k \in N} w_k \rho_k(\mathbf{x}) \tag{2}$$

where $\rho_k(\mathbf{x}) = (r'_k \bigotimes h)(\mathbf{x} - m_k(\mathbf{u}_k))$.

Here, r'_k denotes the mapped reconstruction kernel, and h denotes the band-limited filter in screen space. m_k denotes the local affine approximation of the projection mapping $\mathbf{x} = m(\mathbf{u})$ for the point $\mathbf{u}_k$. After Taylor expansion, it can be expressed as:

$$m_k(\mathbf{u}) = \mathbf{x}_k + J_k(\mathbf{u} - \mathbf{u}_k) \tag{3}$$

where J_k is the jacobian matrix, and $J_k = \frac{\partial m(\mathbf{u}_k)}{\partial \mathbf{u}}$.

The filter function $\rho_k(\mathbf{x})$ in the formula (2) is the resampling kernel, which is expressed by the convolution of the warped reconstruction kernel and the band-limited filter. This approach is called surface splatting. In the paper of Zwicker et al[8], both the reconstruction kernel and the band-limited filter are expressed as an elliptical Gaussian function, and so the Gaussian resampling kernel is called the EWA filter.

4 Hardware Accelerated Surface Splatting

We use the OpenGL point primitive to approximate the splat, and adopt the Cg language[16] to program the vertex shaders and fragment shaders. The traditional two-pass algorithm[14] is implemented to identify visibility and blend every pixel colors. During the first pass, to fill the depth buffer, the points are slightly shifted against the viewer by an offset ε, and rendered without lighting. In the second pass, the points are rendered with lighting and blending, while the depth buffer is not writable. The EWA filter is also computed and stored in

every fragment's alpha component in the second pass. Finally, every pixel color must be divided by the sum of weights stored in its alpha component.

The main challenge of these two passes is to correct every fragment's depth value. In order to accurately solve this problem, a correct projection transformation is needed to compute formula (3). In the following section, we will define a projection transformation from the OpenGL view frustum.

4.1 Projection Transformation

In order to identify the relationship between a 3D point $\mathbf{P}$ and a pixel $\mathbf{X}$ on the screen, we must define the projection transformation between them. In the world coordinates, the projection transformation can be defined as:

$$\mathbf{X} = AR(\mathbf{P} - \mathbf{C}) \tag{4}$$

where

$$A = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

here, f denotes the focal length, R denotes the rotation matrix, and $\mathbf{C}$ denotes the viewpoint.

Because the projective transformation is substituted by a view frustum in OpenGL(Figure 1), we must identify the parameters f, R, and $\mathbf{C}$.

Any object position is located in the camera space, after multiplied by the model view matrix M which can be directly obtained in OpenGL. It means that the viewpoint is the origin $\mathbf{O}$ after the model view transformation. Therefore, the viewpoint in the world space can be obtained using formula (5). Meanwhile, the model view matrix M can be looked as the rotation matrix R.

$$\mathbf{C} = M^{-1}\mathbf{O} \tag{5}$$

According to the view frustum in OpenGL (Figure 1), the projection matrix can be defined as follows.

$$\begin{pmatrix} \frac{\cot(v/2)}{a} & 0 & 0 & 0 \\ 0 & \cot(v/2) & 0 & 0 \\ 0 & 0 & \frac{z_f+z_n}{z_n-z_f} & \frac{2z_fz_n}{z_n-z_f} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

where, the variable v denotes the angle of the filed of view along the y-axis, a denotes the ratio of the frustum, and z_n, z_f are the distances between the viewpoint and the two clipping planes along the negative z-axis.

Based on this projection matrix, the focal length can be defined as follows. For a point $(x, y, z, 1)^T$ in the camera space, we can multiply its coordinates by the projection matrix , and obtain the result:

$$(\frac{\cot(v/2)}{a}x, \cot(v/2)y, (z\frac{z_f + z_n}{z_n - z_f} + \frac{2z_fz_n}{z_n - z_f}), -z)^T$$

Therefore, its normalized coordinates (x', y') in the view port can be defined as:

$$x' = \frac{\cot(v/2)x}{-az} \tag{6}$$

$$y' = \frac{\cot(v/2)y}{-z} \tag{7}$$

where the value of x' and y' is in the field of [-1, 1].

If we use the variable w and h to represent the width and height of the window, its corresponding coordinates in the window can defined as:

$$x'' = \frac{\cot(v/2)x}{-az} \cdot \frac{w}{2} \tag{8}$$

$$y'' = \frac{\cot(v/2)y}{-z} \cdot \frac{h}{2} \tag{9}$$

For $a = w/h$ in OpenGL, we can further simplify the formula (8) as:

$$x'' = \frac{\cot(v/2)x}{-z} \cdot \frac{h}{2} \tag{10}$$

According to the theory of pinhole imaging, the relationship between the coordinates $(x, y, z, 1)^T$ and (x'', y'') can be described as:

$$\frac{x''}{f} = \frac{x}{-z} \tag{11}$$

$$\frac{y''}{f} = \frac{y}{-z} \tag{12}$$

Comparing and analyzing the formula (9) (10) (11) (12), we have the conclusion: $f = \cot(v/2)$. $h/2$ can be looked as the scale factor from the view port to the window. Although the focal length is imaginary, this conclusion is coherent with the projection transformation in OpenGL.

By far, we have defined a projection transformation from the known parameters in OpenGL. Based on the projection transformation, the EWA filter can be directly computed by the graphics hardware following the software method[8].

4.2 The Splat Size and Shape

In OpenGL, each splat can be rasterized into a rectangle of $l \times l$ pixels on the window. Therefore, a proper distance l must be determined to assure that all the valid fragments can be generated.

According to the sampling distance of the point set, each splat's size can be determined in the vertex shaders. As shown in Figure 2, the edge length l of the rectangle formed by splatting the point $\mathbf{P}$ can be defined as:

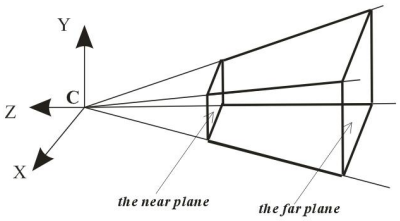

Fig. 1. The view frustum in OpenGL.

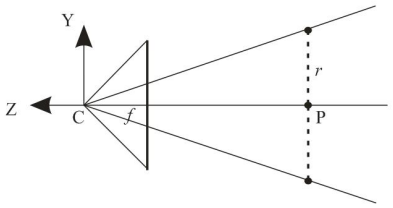

Fig. 2. Defining the edge length of a splat.

$$l = 2r\frac{f}{d_{(C,P)}} \tag{13}$$

where $d_{(C,P)}$ is the distance between the viewpoint $\mathbf{C}$ and the sampled point $\mathbf{P}$, and r denotes the sampling distance at the point $\mathbf{P}$.

The splat's shape can be identified by programing the fragment shaders to eliminate all the invalid fragments. The detail procedure is presented as follows.

Given that a fragment with window coordinate (x, y) is formed by splatting the 3D point $\mathbf{P}$, its corresponding coordinate (u, v) in the local coordinate system of the point $\mathbf{P}$ can be obtained by computing the formula (3). Ren et al[13] have provided an algorithm to compute the Jacobian matrix. With the condition of $\sqrt{u^2 + v^2} \leq 1$, we can identify whether the fragment is valid.

On the other side, according to the projection transformation discussed in last section, we can reproject each fragment into 3D space. As shown in Figure 3, the point $\mathbf{Q}_v$ on the screen is obtained by splatting the point $\mathbf{P}$ with the normal $\mathbf{n}_p$. It is reprojected into 3D space, and crossed with the plane defined by the point $\mathbf{P}$. Depending on the distance between the crossed point $\mathbf{Q}$ and $\mathbf{P}$, we can also judge the fragment's validity.

4.3 Correcting Splat Depth

For one splat, all the fragments generated by OpenGL have a common depth value. These inaccurate depth values will cause failed visibility identification[13], so they must be corrected. An accurate depth correction must follow two important points. The first point is that each fragment depth value should be along the view rays rather than the z-axis in the camera space[13]; the second point is that the nonlinear distribution used in OpenGL should be avoided(Figure 4).

According to the projection transformation, we can accurately correct the depth value. As shown in figure 3, the crossed point $\mathbf{Q}$ can be obtained by the ray-casting algorithm presented in the last section. The distance between the point $\mathbf{Q}_v$ on the screen and $\mathbf{Q}$ is the accurate depth value. Note that the corrected depth value should be normalized into the field of [0, 1], which is supported by OpenGL.

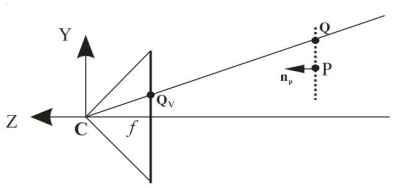

Fig. 3. Correcting the splat shape and depth value.

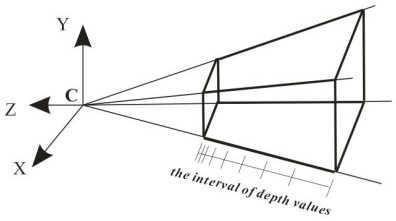

Fig. 4. The nonlinear distribution of the depth values in OpenGL.

4.4 Computing the EWA Filter

According to the theory of surface splatting, the formula (3) must be computed for each fragment. In order to improve the rendering performance, we adopt the approximate EWA filter proposed by Botsch et al[15]. They use the window with 2×2 pixels to clamp each splat, which guarantees that the radius of each splat is no less than 1. Therefore, enough fragments are generated for antialiasing purposes. This means that we should additionally consider the 2D distance between the current fragment and its splat center, when judging the fragment validity. Following this strategy, the final EWA weights can be precomputed, and stored in a 1D texture image. Therefore, each fragment weight can be obtained by sampling the texture image.

5 Implementation and Results

We have applied our approach to render various models generated from virtual and real objects, and achieved very satisfactory results. For a virtual object, we create a LDC model[17] by sampling its surfaces from three sides of a cube. This manner can generate uniformly sampled points, and reduce redundant data furthest. For a real object, an image based modeling method[18] is adopted to generate a set of points with such attributes like light fields, reflectance fields.

Various results rendered by our approach are shown in Figure 5. The Beethoven, cow and head models are rendered with only diffuse reflection, and the man model is rendered with both diffuse and specular reflection. The cup and horse models with more colorful texture are generated from real objects.

In order to test the antialiasing effect, a checkerboard shown in figure 6 is rendered by different methods. It includes 250k points, and each grid has 100 points. The top image is rendered by directly projecting each point onto the screen. The middle and bottom images are rendered without and with the EWA filter, respectively.

Octree based data structure[3] is very effective for visibility judgment. However, this loose data structure will become the bottleneck for data transfer. In our implementation, we use the interleaved vertex arrays to arrange the data, and the vertex buffer object will be the better option. The invisible points are

eliminated in the vertex shaders by judging the angle between view ray and each vertex normal. The multi render target may be the best choice to decrease the computation costs in the fragment shaders[15].

The results we present are measured on 2.8GHz Pentium4 with a NVIDIA GeForce6800 GT graphics card, running Window XP. Table 1 shows the rendering performance for different window resolutions. From this table, we can see that our method can render up to 8M filtered splats per second with 1024×1024 resolution. For different window resolution, the rectangles generated by OpenGL have different sizes. The edge length is increased from 2 or 3 to 5 or 6 pixels, when the window resolution is increased from 512×512 to 1024×1024. For higher resolution, more fragments are generated, and the computation costs for EWA filter are increased.

Table 1. Rendering performance of our approach in frames per second for several models shown in figure 5 (the splat size is 2 or 3 for the 512×512 resolution, and 5 or 6 for the 1024×1024 resolution.

objects	points	512×512 $(2, 3)$		1024×1024 $(5, 6)$	
		Unfilter	Filter	Unfilter	Filter
beethoven	565k	31.6	15.5	19.3	14.6
head	340k	50.9	25.6	34.1	17.1
cow	249k	68.1	34.5	54.7	28.3
man	168k	98.8	52.1	68.8	36.6

Compared with the work of Ren et al[13], our approach provides a projection transformation, which benefits EWA filter computation. It also shows superior rendering performance, for their method processes a rectangle for each vertex.

Guennebaud and Paulin[14] also provided an accelerated method to implement surface splatting. But in their work, the corrected depth values are nonlinearly distributed, which makes it difficult to select a proper offset for visibility identification. As shown in Figure 7, the same Venus model is located in two positions which have different distances to the viewpoint, and the same offset is selected. The nonlinear distributed depth values lead to failed visibility identification on near object, and our method avoids it effectively.

6 Conclusion

We have described an efficient accelerated approach for rendering point based models, which is based on the EWA surface splatting technology. From the OpenGL view frustum, we can deduce the equivalent projective transformation. Based on the projective transformation, the splat sizes and shapes are easily determined, and each splat depth value is accurately corrected. In the future,

we will focus on extending our method to handle more complex models, and realizing some particular effects such as transparency.

Fig. 5. Several models rendered by our approach with different lighting condition.

References

1. Deering, M.: Data complexity for virtual reality: where do all the triangles go? In: IEEE Virtual Reality Annual International Symposium (VRAIS), Seattle, WA (1993) 357–363
2. Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., L.Pereira, M.Ginzton, Anderson, S., Ginserg, J., shade, J., Fulk, D.: The digital michelangelo project: 3d scanning of large statues. In: SIGGRAPH2000 Proceedings, Los Angeles, CA (2000) 131–144
3. Pajarola, R.: Efficient level-of-detail for point based rendering. In: Proceedings IASTED Computer Graphics and Images. (2003)
4. Levoy, M., Whitted, T.: The use of points as display primitives. Technical Report TR85-022, the University of North Carolina at Chapel Hill, Department of Computer Science (1985)
5. Kobbelt, L., Botsch, M.: A survey of point based techniques in computer graphics. Computers & Graphics **28**(6) (2004) 801–814
6. Alexa, M., Behr, L., Cohen-Or, D., Fleishman, S., Levin, D., , Silva, C.: Point set surfaces. In: Proceedings of IEEE Visualisation, San Diego, CA (2001) 21–28

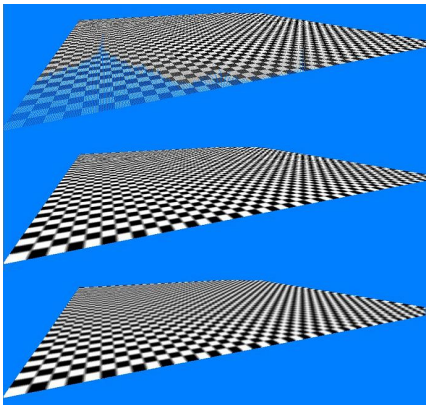

Fig. 6. Point based checkerboard rendered by different methods.

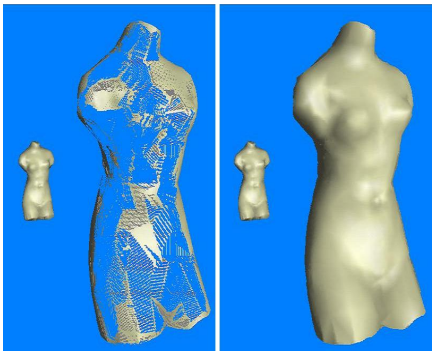

Fig. 7. The rendered results with different depth correction(Left: nonlinear depth correction. Right: depth correction of our method.

7. Sainz, M., Pajalora, R.: Point-based rendering techniques. Computers & Graphics **28**(6) (2004) 869–879
8. Zwicker, M., pfister, H., Vanbaar, J., Gross, M.: Surface splatting. In: SIGGRAPH2001 Proceedings, Los Angeles, CA (2001) 371–378
9. Grossman, J.P., Dally, W.: Point sample rendering. In: Rendering Techniques '98, Springer Wien, Vienna, Austria (1998) 181–192
10. Rusinkiewicz, S., Levoy, M.: Qsplat: a multiresolution point rendering system for large meshes. In: Proceedings of the 12th Eurographics Workshop on Rendering, London, UK (2001) 151–162
11. Stamminger, M., Drettakis, G.: Interactive sampling and rendering for complex and procedural geometry. In: Proceedings of the 12th Eurographics Workshop on Rendering, London, UK (2001) 151–162
12. Botsch, M., Kobbelt, L.: High-quality point-based rendering on modern gpus. In: Proceedings of Pacific Graphics03, Alberta, Canada (2003) 335–343
13. Ren, L., Pfister, H., Zwicker, M.: Object space ewa surface splatting: A hardware accelerated approach to high quality point rendering. In: Proceedings Of Eurographics 02. (2002) 461–470
14. Guennebaud, G., Paulin, M.: Efficient screen space approach for hardware accelerated surfel rendering. In: Proceedings of Vision, Modeling, and Visualization 03, Munich, Germany (2003)
15. Botsch, M., Hornung, A., M.Zwicker, Hobbelt, L.: high-quality surface splatting on today's gpus. In: Eurographs symposium on point based graphics(2005). (2005)
16. Mark, W.R., Glanville, R.S., Akeley, K., Kilgard, M.J.: Cg: a system for programming graphics hardware in a c-like language. ACM Transaction on Graphics **22**(3) (2003) 896–907
17. Pfister, H., Zwicker, M., Vanbaar, J., Gross, M.: Surfels: Surface elements as rendering primitives. In: SIGGRAPH2000 Proceedings, Los Angeles, CA (2001) 335–342
18. Matusik, W., Pfister, H., Ngan, A., Beardsley, P., Ziegler, R., McMillan, L.: Image-based 3d photography using opacity hulls. In: SIGGRAPH2002 Proceedings, San Antonio, Texas (2002) 427–437