A classification-based rendering method for point models

Haitao Zhang*, Arie Kaufman

Center for Visual Computing (CVC) and Computer Science Department, Stony Brook University, Stony Brook, NY 11794-4400, USA

Received 3 December 2006; received in revised form 2 May 2007; accepted 14 July 2007

Abstract

We present a classification-based high quality rendering method for large scenes with point-based models. Each model is represented by a uniformly sized point hierarchy. All the points at the same resolution in the hierarchy share the same splat radius, and the splat radius ratio between any two neighboring resolutions are the same. We use this data structure to minimize the number of rendering points with no compromise in the image quality. Compared to the standard OpenGL point primitive, it is quite expensive to rasterize the perspectively correct splat projection even when using the advanced features of modern GPUs. We propose a classification-based rendering algorithm to reduce the number of points rendered by the complicated splatting. The potentially visible points are collected and classified into two types: a pixel point, whose image projection size is within one pixel; and a multi-pixel splat, whose image projection size is larger than one pixel. Only the multi-pixel splats are rendered by a perspectively correct splatting algorithm. The point classification is integrated with the hardware accelerated rendering. By adopting the uniformly sized point hierarchy and the classification-based rendering, we can achieve effective rendering of large scenes.

Keywords: Point-based graphics; Point-based rendering; Classification-based rendering; Uniformly sized point model

1. Introduction

With the advent of graphics hardware and the 3D scanning techniques, point has become an important modeling and rendering primitive. The main difficulty in point-based rendering is how to generate a hole-free visible surface on the screen without any coherence information of the points. Surface splatting is a commonly used method, which defines a disk-shape splat on the tangent plane of the model at every point. During rendering, these splats are projected onto the image plane and overlapping visible splats are blended together to generate a smooth hole-free surface.

It is an expensive operation to compute the perspectively correct projection of a splat even when taking advantage of the programmability of modern GPUs. For a complex scene consisting of a large number of point-based models, surface splatting would be relatively slow, especially compared to the use of the standard OpenGL point primitive with one pixel size. One advantage of the surface splatting is that a high quality image can be generated directly from the point-based models from any viewpoint, which is not guaranteed by the use of the standard OpenGL point primitive.

In this paper, we present a classification-based method for efficient rendering of large scenes of point-based models. A uniformly sized point hierarchy is used for modeling, which reduces the number of points in the rendering. The points in the same resolution share the same radius value, and these radius values in a point hierarchy form a geometric sequence. An effective resolution is chosen for each model for fast rendering without any compromise in image quality. A classification-based rendering method classifies the potentially visible points into two types: a pixel point and a multi-pixel splat. The image plane projection size for the pixel point is within one pixel, while the multi-pixel splat projects to more than one pixel. Only those points classified as multi-pixel splats need perspectively correct splatting for high quality rendering. The classification process is incorporated in the hardware accelerated rendering for effective rendering.
From the point of view of modeling, our uniformly sized point hierarchy is a hybrid model hierarchy composed of points and splats. The coarser resolutions are all points, and the finest resolution can be either splats or points, which is determined by the classification-based rendering.

2. Related works

2.1. Point-based Rendering


Using the programmability of modern GPUs, hardware accelerated methods [7–10] have been introduced for efficient point-based rendering. For the GPU-based rendering algorithms, the main difficulty is how to blend only visible splats together and discard the invisible ones. Most of the GPU-based methods use a three-pass splatting method: a visibility splatting pass to get the depth, followed by a splatting pass for rendering those splats that pass the depth test, and finally a normalization or shading pass. By shifting the depth away along the view direction in the visibility splatting, overlapping visible splats are blended together in the splatting pass. In the fragment shaders, the proper splat projection shape and depth are computed, together with the weight for blending. The final image is generated by a normalization of the color buffer, or by a per-pixel shading pass, such as Botsch et al.’s [11] method, which splats point attributes into multiple frame buffers for per-pixel Phong shading. Guennebaud et al. [12] add an additional pass for rendering point indices to limit the number of points rendered by splatting. Dachsbacher et al. [13] use a sequential list to store the point tree hierarchy. Each point in the sequential point tree has an error value, and the point is rendered only when its error is within an error range computed from the view distance. Our rendering method is also based on the GPU-based three-pass splatting method. We collect the potentially visible points, and classify them according to the projected image size for better performance. Furthermore, we minimize the number of rendered points by a uniformly sized point hierarchy.

There are indirect point rendering systems [14–16] which fit a piecewise quadratic function in a local region for surface reconstruction. In the rendering, these quadratic functions are resampled with image resolution, or converted into other rendering primitives such as meshes. Ray tracing also can be used by finding the intersections between the rays and the implicit surface [17].

2.2. Hybrid model hierarchy

Hybrid model hierarchy consisting of points and other primitives can be used for rendering large models. POP [18] is a model hierarchy that uses polygons as leaf nodes and points for inner nodes. Cohen et al. [19] adaptively replace the nodes by points in a triangle hierarchy construction. Dey and Hudson [20] use both triangle hierarchy and the corresponding point hierarchy in the rendering. Proper resolution in the model hierarchy and the corresponding primitive type are chosen according to the viewing or some user defined error tolerance. Our uniformly sized point hierarchy consists of pure points. In the rendering, these points are rendered either by splats or one-pixel size points. Thus, our model can be thought of as a hybrid model with splats and points.

3. Uniformly sized point hierarchy

3.1. Motivation

Given a point-based multi-resolution model, we want to select one resolution for the best image quality with the lowest rendering time possible. Fig. 1 shows three renderings at three different resolutions of a Santa model when the splat projection is larger than one pixel for all the points. The finest resolution will always generate the best image, which should be used to guarantee the rendering quality. Fig. 2 shows the results of the same three resolutions when the splat projection is within one pixel. From the enlarged view (Fig. 2), we can see that there is no noticeable difference in the image quality among the three resolutions. If a limited number of points are blended for each pixel, such as Guennebaud et al.’s [12] rendering method, using the coarsest resolution whose splat projection is still within one pixel may even result in better filtering. This is because the attributes of a point in a coarser resolution are the averages of the corresponding points in the finest resolution.

To sum up, if the splat projection is larger than one pixel for the finest resolution of the model, this finest resolution (Fig. 2) should be rendered for the best image quality without much increase with the rendering time compared to any coarser resolution. If the splat projection is within one pixel, the coarsest resolution whose splat projection is still within one pixel (Fig. 2) is used to minimize the number of rendering primitives and get the best filtering. In order to efficiently determine the splat projection size for a point-based model, we use a uniformly sized point hierarchy.

3.2. Point hierarchy

For a uniformly sized point hierarchy, all the points at the same resolution share a same splat radius. Thus, it is
efficient to determine the size of a splat projection on the model level. Furthermore, the splat radii at different resolutions form a geometric sequence, with the splat radius at each coarser resolution is \( K \) times the value at its neighboring finer resolution. After computing the splat projection size for the finest resolution, if it is within one pixel, it is easy to find the coarsest resolution whose splat projection is still within one pixel.

Given a uniformly sized point-based model \( M_0 \) and a ratio \( K \), a point hierarchy is constructed so that the splat radius ratio between two neighboring resolutions is \( K \). There are many methods for simplifying point-based models directly [21]. Wu and Kobbelt [22] select a subset of the original points as a hole-free approximation of the model within a prescribed error tolerance. Progressive splatting [23] uses two error metrics defined on the splat distance and normal difference in the clustering. We use a clustering method based on splat growing.

The original uniformly sized model \( M_0 \) is used as the finest resolution. Assume the splat radius for \( M_0 \) is \( r \), for any coarser resolution \( M_m \) with splat radius \( K^m r \), it is generated by simplifying \( M_0 \). All the points in \( M_0 \) are clustered, and each cluster forms a point in \( M_m \), whose position is the center of the cluster. The cluster is generated by splat growing. A cluster \( C \) is initialized by randomly choosing an un-clustered point \( p_0 \) in \( M_0 \), and the corresponding initial splat is centered at \( p_0 \) with radius \( r \). Nearby un-clustered points are added into the cluster one by one to grow the splat. An un-clustered point \( p_k \) can be added to the cluster if it satisfies:

(i) Radius requirement: After adding \( p_k \), the splat center \( c_k \) is recomputed as the average of all points in the cluster, and the splat radius \( r_k \) is the maximum distance between \( c_k \) and the points in the cluster plus \( r \) for conservatively covering all the points:

\[
r_k = \max_{p \in C} (|c_k p| + r).
\]

(ii) Normal requirement: The normal difference between \( p_k \) and any point already in the cluster should be within a threshold to prevent clustering points on disconnected opposite surfaces. This threshold increases with \( m \) value to alleviate this requirement for coarser resolutions.

If there are more than one point satisfying the two requirements, the point with the minimum total distance to all the points in the cluster is chosen. The splat growth stops when no more qualifying points can be added. The attributes of the point from the cluster for \( M_m \), such as normal and color, are the averages of the original points in the cluster.

Compared with Wu and Kobbelt’s splat growing method [22], our method dynamically changes the splat center in
the growing while theirs fixes it, and our splat growing is bounded by the splat radius and theirs stops when reaching an error tolerance. This is because the two simplification methods work for different purposes. Wu and Kobbelt try to find a good approximate of the original model in terms of surface error, and our method is used for our classification-based rendering. We only care that no artifacts would appear in our classification-based rendering. While our simplified model may not be the best simplification in term of surface error, it is sufficient for our classification-based rendering. In our classification-based rendering, discussed in the next section, the simplified models are only rendered when the splat projection on the image plane is within one pixel size. This prevents any noticeable artifacts from the simplified models. Fig. 3 shows the point distribution in a portion of the Santa model at five resolutions with splat radius ratio $K = 2$.

With a model has a large number of points, or the model shape is irregular in its three dimensions, it would not be efficient in our rendering when creating the point hierarchy on the whole model. A finer resolution has to be chosen even if only a few points in the next coarser resolution project to multiple pixel size, which increases the number of rendered points dramatically. In order to deal with this problem, we segment such models into several clusters using $k$-means clustering, and create a uniformly sized point hierarchy for each cluster.

### 3.3. View-dependent resolution selection

Each model $M$ (or each cluster in the large models) is associated with several attributes: (1) a bounding sphere $S$, with $C_S$ and $R_S$ as its center and radius; (2) a splat radius $r$ for the finest resolution $M_0$; and (3) the splat radius ratio $K$ between any neighboring resolutions $M_m$ and $M_{m+1}$. In the rendering, an effective resolution is selected for $M$ if it passes a view frustum culling.

Assuming the aspect ratio of the field of view is the same as the image aspect ratio, the maximum possible splat projection size for $M_0$ on the image plane is defined as

$$d(M_0) = -h/\tan(fovy/2) * r/(Z(C_S) + R_S),$$

(2)

with $h$ as the image height, $fovy$ as the field of view angle in the vertical direction, and $Z(C_S)$ as the depth of $C_S$. If $d(M_0) > 1$, this finest resolution is selected. Otherwise, the coarsest possible resolution $M_n$ that still satisfies $K^nd(M_0) < 1$ is selected. Points in the selected resolution are sent to the GPU for rendering.

### 4. Classification-based rendering

The points in the selected resolutions of the point hierarchies are rendered by a classification-based method implemented totally on the GPU. Our rendering method shares some similarities with Guennebaud et al.’s [12] deferred splatting method. Their method renders the point indices for visible point selection, and performs EWA surface splatting on the selected points. In our method, we collect and classify the potentially visible points and store their attributes in textures using the multiple draw buffers extension (MRT). There are two types of classification: a pixel point and a multi-pixel splat, determined by the splat projection size, with the former within one pixel, and the latter larger than one pixel. EWA surface splatting is only applied to the multi-pixel splats, whose attributes are collected in the textures. The pixel points can be efficiently rendered by texture mapping, because their image positions are already computed in the point collection and
classification step. The visible point attributes are blended together, and per-pixel Phong shading is used for the final image.

There are six steps in our rendering pipeline, which are described below:

Step 1: Point collection and classification. Points are rendered using the standard OpenGL point primitive with pointsize 1. By enabling depth test, the potentially visible points are collected, one per pixel. The point classification is determined by the splat projection size. The point position and normal under the eye coordinates, the splat radius, the color, and the classification are stored in two 32-bit floating-point RGBA textures (TEX\textsubscript{PR} and TEX\textsubscript{NC}) using the MRT extension. The RGB channels of TEX\textsubscript{PR} store the point position, and the A channel stores the splat radius (which is greater than 0) if the classification is multi-pixel splat, −1 for pixel point, 0 for the background. The point normal is stored in the RGB channels of TEX\textsubscript{NC}, and the remaining A channel holds the color, which is packed into a 32-bit float using the Cg function pack\textsubscript{4ubyte}. This pass is also a part of visibility splatting to get the depth from the pixel points and the splat centers of the multi-pixel splats.

Step 2: Vertex array creation. TEX\textsubscript{PR} and TEX\textsubscript{NC} are converted into a vertex array for rendering using the vertex buffer and pixel buffer object extensions (VBO and PBO). The image size is the length of this vertex array, which consists of position (4 floats), normal (3 floats) and color (3 ubytes) attributes.

Step 3: Multi-pixel splat visibility splatting. To finish the visibility splatting, the multi-pixel splats need to be rendered by a perspectively correct surface splatting method. The vertex array generated in Step 2 is rendered, and those points with a non-positive 4th value of the point classification are culled away in the vertex shader to render the multi-pixel splats only. Only the A channel of TEX\textsubscript{PR} is rendered, which is set to 0 to erase possible invisible pixel points behind the splats.

Step 4: Pixel point rendering. The color and normal attributes of the visible points are to be blended together and stored in two textures for deferred shading. These two textures are initialized with the pixel points rendering in this pass.

For each pixel point, its attributes are stored in TEX\textsubscript{PR} and TEX\textsubscript{NC} at a position the same as its image projection. Thus, pixel point rendering becomes very simple, similar to a texture mapping of an image size rectangle: TEX\textsubscript{PR} and TEX\textsubscript{NC} are binded, the color and normal attributes are rendered if the 4th component of TEX\textsubscript{PR} is smaller than zero; otherwise, the corresponding pixels are cleared to 0.

Step 5: Multi-pixel splat rendering. The multi-pixel splats are rendered with blend enabled to accumulate the color and normal attributes, using a perspectively correct splatting similar to Step 3.

Step 6: Shading pass. Per-pixel Phong shading is performed using the color and normal values generated in the previous steps.

Step 3 and 5 can be skipped if no finest resolution is selected in the view-dependent resolution selection, because all the points would be classified as pixel points. We use the ray-casting method [9], which finds the ray-splat intersections to get the perspective correct splat shape and depth. For each fragment, the ray goes through the center of the corresponding pixel. A perspective correct splat projection on the image plane is a near ellipse shape. When the splat is nearly parallel to the view direction, the short axis of this near-ellipse will become too short to cover any pixel center, resulting in holes in this ray-casting method. In order to avoid the possible holes, we extend this method to guarantee that all the pixels that the long axis of the near-ellipse goes through will be rendered. The corresponding long axis direction \(A_{lo}\) on the splat plane and the short axis direction \(A_{so}\) on the image are computed for the splats. If the distance between the ray-splat intersection and the splat center long \(A_{lo}\) is within the splat radius, and the distance along \(A_{so}\) is smaller than \(\sqrt{2}/2\), this fragment should be rendered regardless of whether the intersection and splat center distance exceeds splat radius or not. Since this perspective correct splatting is only performed on the multi-pixel splats, there is only a tiny effect on the rendering speed from this extra computation.

5. Results

We use a 3.59 GHz PC with an NVIDIA Quadro FX 4500 graphics card to conduct the experiments for our classification-based rendering for uniformly sized point hierarchies. All the vertex/fragment shaders are written in the Cg language.

We have created two large scenes \(S_A\) and \(S_B\), which consist of 9604 and 18,480 uniformly sized point hierarchies, respectively. The total number of points is 835,670,451 for \(S_A\), and 1,608,071,219 for \(S_B\), counting at the finest resolutions. Although we use only four unique models (Dinosaur, Horse, Rabbit and Santa) and repeat them to create the scenes, in the rendering, the models are regarded as different models and rendered separately for testing purpose. A uniformly sized point hierarchy with five resolutions is created for each of the four test models, and the number of points at every resolution is shown in Table 1. The computational costs for building these 4 point hierarchies range from 1 to 3 min.

Figs. 5(a)–(c) show the renderings for \(S_A\), and Figs. 5(d)–(f) for \(S_B\). The image sizes are 800 × 800. Table 2 is the corresponding frame rate, the time spent on each of the rendering steps, and the number of points sent to the GPU from the selected resolutions of the point hierarchies. The running time for Steps 2–6 is relatively fixed by the image size. The number of classified multi-pixel splats will affect the time for Steps 3 and 5, with an upper bound also set by the image size. The running time for Step 1 is approximately linear with the number of points sent to the GPU. Since the time for Steps 2–6 is relatively fixed with a same image size, Step 1 takes most of the time in the rendering.
when the number of points is large. Once the image size is changed, the time for Steps 2–6 will be affected accordingly. The image plane splat projection size will also be altered, resulting in a possible change of the selected model resolutions, which determine the number of points handled by Step 1. The last column of Table 2 shows the corresponding frame rate when the image size is reduced to $512 \times 512$.

6. Conclusions and future work

We have implemented a classification-based high quality rendering method for large scenes composed of point-based models. A uniformly sized point hierarchy is used for point minimization in the rendering. In the GPU-based rendering, the potentially visible points are classified according to their image plane projection sizes, and effective rendering methods are applied to the different classifications.

For a large scene, the rendering speed is mainly determined by the number of points sent to the GPU. Therefore, it is essential to minimize the number of points using different culling techniques. In our rendering method, view frustum culling and level-of-detail method (the uniformly sized point hierarchy) are used. It is possible to integrate other culling methods such as occlusion culling and backface culling to further reduce the number of points.

We use 2 as the splat radius ratio for the uniformly sized point hierarchy, and five resolutions are used for each model. It is important to study how to determine the best

<table>
<thead>
<tr>
<th>Model</th>
<th>Point count at resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Dinosaur</td>
<td>100,468</td>
</tr>
<tr>
<td>Horse</td>
<td>104,764</td>
</tr>
<tr>
<td>Rabbit</td>
<td>67,038</td>
</tr>
<tr>
<td>Santa</td>
<td>75,781</td>
</tr>
</tbody>
</table>

The splat radius relative to the finest resolution is used to represent the resolution level.

Fig. 5. Renderings of the large scenes with an image size $800 \times 800$: (a)–(c) rendering of a scene consisting of 9604 models and (d)–(f) rendering of a scene consisting of 18,480 models.
splat radius ratio and the number of resolution for a given model to optimize the rendering. Since only one splat radius is used in the finest resolution, a small radius should be used if there are detailed features in some parts of the model. This is not an efficient modeling when the model has both flat regions and regions with high curvatures. We plan to extend our multi-resolution data structure to deal with these models. In the finer resolutions, more than one splat radius should be allowed to preserve the shape features while not increasing the point density in the relatively flat regions.

Acknowledgments

This work has been partially supported by NSF Grant CCR-0306438. The Dinosaur, Horse, Rabbit, and Santa models are courtesy of Cyberware, Inc.

References


Table 2

The performance for large scenes with an image size 800 × 800, including the number of points sent to the GPU, the frame rate, and the time for the six steps in the rendering.

<table>
<thead>
<tr>
<th>Image</th>
<th>Number of points sent to GPU</th>
<th>Frame rate (fps)</th>
<th>Time (ms)</th>
<th>512 × 512 image</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Step 1</td>
<td>Step 2</td>
</tr>
<tr>
<td>5(a)</td>
<td>4,052,359</td>
<td>12.8</td>
<td>45</td>
<td>31</td>
</tr>
<tr>
<td>5(b)</td>
<td>15,375,511</td>
<td>4.8</td>
<td>172</td>
<td>31</td>
</tr>
<tr>
<td>5(c)</td>
<td>11,376,964</td>
<td>6.0</td>
<td>109</td>
<td>31</td>
</tr>
<tr>
<td>5(d)</td>
<td>5,248,646</td>
<td>10.5</td>
<td>62</td>
<td>31</td>
</tr>
<tr>
<td>5(e)</td>
<td>7,580,679</td>
<td>7.9</td>
<td>78</td>
<td>31</td>
</tr>
<tr>
<td>5(f)</td>
<td>2,452,274</td>
<td>14.9</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>

The last column shows the corresponding frame rate when the image size is reduced to 512 × 512.