AN ALGORITHM FOR VISIBILITY-DETECTION IN PBR

YUEPING FENG, HUIXIANG ZHONG, HUIQIN WANG,YUNJIE PANG

College of computer science and technology, Jilin University, Changchun 130012 P.R.China

Abstract:

Point-based rendering attracts more and more interest from researchers as 3D digital scan devices are used more widely in computer graphics. Within point-based rendering, removing hidden points from a 3D object is a key process to make the rendered object look realistic. This paper presents an algorithm for visibility-detection in point-based rendering, which is designed according to the features of point-based models.

Keywords:

Point-based rendering, hidden points, back direction point, sorting of points

1. Introduction

There are many ways of representing a 3D object surface in computer graphics. Polygons, spline surface, implicit surface are all used. By using these representations, the process of producing a 3D object surface can be simplified. Among these representations, polygon or triangle representation is used most widely and considered as the most important representation since all the others such as implicit surface, NURBS, subdivision surface can be converted to it. In addition, this representation can also be easily processed by using common graphics devices. However, as the complexity of an object model to be rendered keeps on increasing, the number of triangles also increases. As Smith[1] said, "The realistic means 800,000 polygons". Thus, disposing triangles becomes the bandwidth bottleneck and leads to floating-point computation difficulty.

Recently, 3D digital scanners become very popular, and geometric details and shapes obtained through scanning also become abundant. This leads to the need of finding out an efficient way in representing, processing and rendering geometric models with a large scale and high complexity. 3D scanners create a large quantity of sampled points, and there is no relationship between any of these points. Traditional way of processing these points is to create triangle meshes by sampling these points, but with the advance of 3D scanning technology, the number of mesh points obtained through 3D scanning usually falls on the order of billion[2]. The efficiency of polygon model transformation and reconstruction becomes lower, and the limit of polygon representation also becomes dominant. As a result, with the wide applications of 3D scanning devices, the development of computer graphics needs to solve these following problems: an efficient way of rendering objects and scenes with high complexity; reduction of redundancy; saving of data storage and new rendering structure which can directly represent dense point clouds built by 3D scanners.

Due to the above reasons, Point-Based Rendering (PBR) becomes popular again [3]. This rendering algorithm does not use the traditional triangle facet method. It records the information of each point, and directly from the information it can reconstruct the object. Thus, it provides a new approach that can resolve the difficulty of fast rendering processing of abundant 3D sampled data.

The algorithm proposed in this paper is just to deal with one of the problems existing in point-based rendering. Sometimes there is no facet (such as splat) to cover the points which should have been concealed by the points in front, so they are mistakenly displayed on the 3D object surface. In order to identify these points and hide them, we have developed a visibility-detection algorithm.

2. Point-based Rendering

2.1. Background and Related Work

Point-based rendering has a long history. In 1985, Levoy and Wihtted[3] first proposed the idea of point-based rendering. They discussed about the disadvantages of using continuous scanning method to render, and pointed out that although points are simple, they are sufficient in representing any 3D object. As long as the points are dense enough on a 3D object surface, projecting each point onto the screen can still give a realistic visual effect. They used a circular splat as rendering element, and set a minimum value of the splat's size to maintain the rendering quality. With the development of PBR, other researchers also proposed their own PBR algorithms. Currently, two PBR algorithms, Qsplat[4,7,8,9] and Surfel[5,10,11,12], are most developed and the basic ideas are both to construct a small area centered around the sampled point. Qsplat uses a hierarchy of bounding spheres for visibility culling, LOD control, and rendering; and the level tree traverses in a

Manuscript received June 25, 2006.
Manuscript revised June 30 , 2006.

depth-first order. Surfel uses "Surfel" as basic element and the data structure used is called LDC (Layered Depth Cube). The rendering process of Surfel is similar to Qsplat. The cost of applying the above two algorithms is considerably high. However, the algorithm proposed in this paper uses sampled points to render directly, and thus is more cost-saving.

2.2. Point Data Structure on Objects

The surface of any 3D object can be viewed as a composition of a set of points with different degree of density. In 1974, Catmull noticed that the limit of division in geometry is point [5]. Therefore, representing a geometric object by points conforms to its intrinsic property. The point data obtained from a geometric object may differ from each other depending on how they are obtained. In our research, the point data are all obtained from polygon meshes. The data structure used to describe the point is as follows:

```
point_data{
        position;
        normal;
        color;
        visbility;
        ⋮
        }
```

where position is the coordinate of the point in world axis; normal is the normal vector of the point; color is the color or texture of the point and visibility is the visibility of the point. The value of visibility is initially assigned to false. As points are used to render in this case, the traditional visibility-detection algorithm needs to be modified as well.

2.3. Visibility-detection algorithm for PBR

Visibility-detection is a necessary process to render an image with realistic visual effect. Usually visibility-detection means to remove hidden lines or surfaces. In PBR, visibility-detection is to remove hidden points, i.e. points that are invisible. Hidden points can be subdivided into 2 groups: back points and non-back points. Back points can be identified through the angle between the vector V in the view direction and normal vector N of each point. When $V \cdot N < 0$, the point is a back point. Non-back points can be defined as those that should have been concealed by the points in front and do not satisfy the condition of being a back point. Because there is no facet to conceal the points, these hidden points can be displayed through the gap on the 3D object surface, causing failures of rendering. Therefore, different algorithms have been developed to remove these hidden points.

2.4. Removing back points

The angle between the vector V in the view direction and normal vector N of each point can be used to determine whether a point is a back point or not. When $V \cdot N < 0$, the point is a back point and the value of the visibility is set to false.

Figure 1 shows an image after the back points are removed. The arrow shows the points which should have been concealed but displayed through the gaps. This is because the object is represented by discrete points, when the density of the points is not high enough to cover all the pixels on the projected plane, there are gaps between these points. These gaps can be filled up by re-sampling. However, the problem of uneven density still exists. In addition, with different resolutions, gaps may still exist. As a result, there is a need to do a second iteration of visibility-detection based on view point.

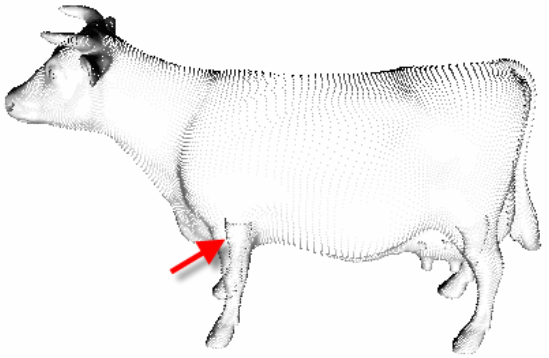


Figure 1: Image after removing back points

2.5. Visibility-detection based on view point

After removing the back points, all the invisible points that are not facing the view point have been removed. However, the above algorithm cannot remove all the hidden points in every circumstance. It can identify and remove all the hidden points for a single convex polygon, but it cannot guarantee it has removed all the hidden points for converse polygons or complicated scenes with many objects inside. Therefore, the depth-buffer method used in traditional visibility-detection has been modified to give a new visibility-detection algorithm based on view point.

The basic idea of visibility-detection based on view point is like this: as some points on a 3D object may be projected to the same pixel on the screen, each pixel thus corresponds to a point sequence (the sequence may be empty). In this sequence, the point that can be seen has the shortest distance to the view point since all the other points in the sequence are concealed by the point in front (assuming there is no gap between points). If the points inside each sequence are sorted according to the view direction, the very first point in each sequence is thus the

visible point (Figure 2). As a result, only the first point in each sequence needs to be considered when dealing with displaying or lighting effect. The sorting method is as follows:

1. The initial sequence of these points is empty.
2. For each point P in non-back point set, obtain the world coordinate, view position coordinate, normal vector and color of P.
3. Project P onto the screen, obtain the corresponding screen coordinate, i.e. the pixel point (x, y); if the corresponding sequence of point (x, y) is empty, put the point into the sequence; if not, use quick sort to find a proper position in the sequence for this point, and insert it into the sequence.

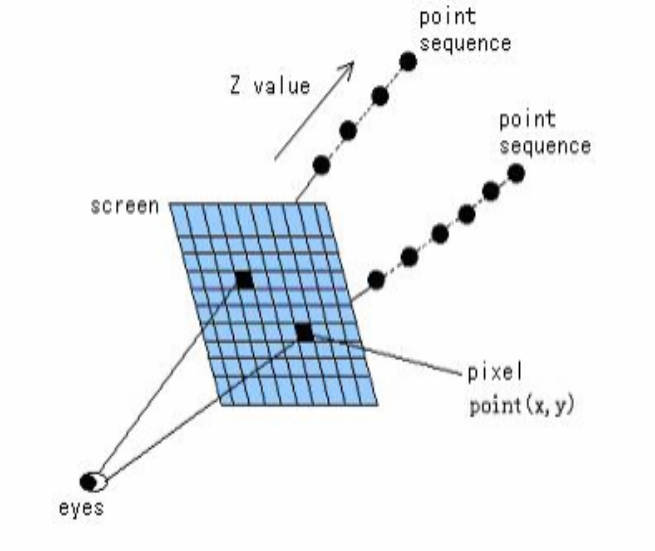


Figure 2: Sorting of points

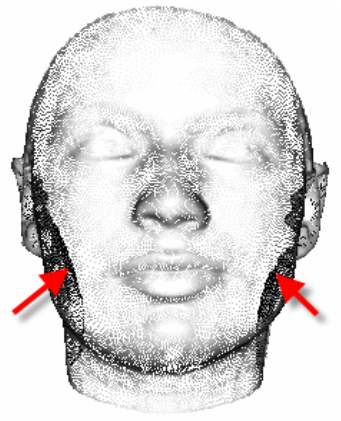
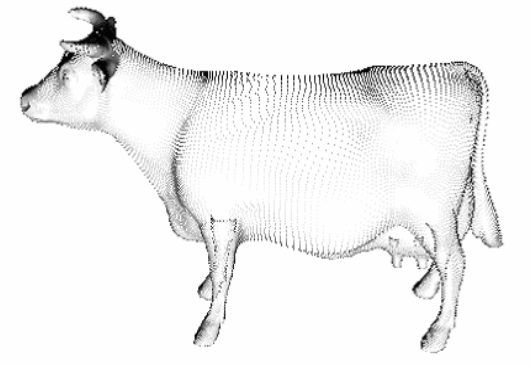
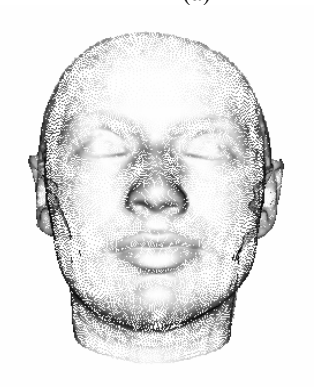


Figure 3: Image after sorting of points according to view point, arrows pointing to back-points

In each sequence, only the first point has the visibility value set to true, while all the rest are set to false. During rendering, only those points with a visibility value of true can be displayed on the screen.



(a)



(b)

Figure 4: Images after removing back points and visibility-detection

2.6. Visibility-detection based on local-dependent depth

After completing the above steps, most of the hidden points have been removed; the results are shown in Figure 4. From Figure 4, it can be observed that all the points with "true" visibility value have been displayed on the screen. But it is also noted that many points on the cow legs in (a) and the cheeks in (b) which should not have been displayed are displayed. There are 2 reasons why these points cannot be removed, one is that they are not back points; the other is that there is no point in front to conceal them, i.e., the density of points in front of these points is not large enough to conceal the corresponding pixel area on the screen. Therefore, these points can be seen from the gaps between the points in front of them (from the view direction). In order to remove these points, visibility-detection based on window-dependent depth algorithm has been developed.

As we know, according to view direction, these points are different from points around them in terms of depth (z axis). There are many points in front of them, but as they are not in line with view point, they cannot conceal the points that should be concealed. Therefore,

window-dependent depth can be used to identify which points should not be displayed.

After the above hidden points removal, a depth matrix with visible points can be obtained. The maximum values of x and y are recorded in this matrix and the depth matrix is thus defined, the size of which is x × y and all the entries have their values set to 0 initially. This matrix is used to store the distance from each point to the view point (depth). The display window is divided into 3 × 3 sub-windows in which hidden points problem is discussed. The depth of each point in these sub-windows (9 in total) is computed and the maximum depth (max) and minimum depth (min) are recorded. Threshold is defined as k × (max − min) where k is an unsigned integer; meandep denotes the average distance from each point to view point. If meandep < depth of point < threshold, the point is taken as a hidden-point. For this type of point, there are 2 methods to make them invisible: one is to record this point as invisible point; the other is to make this point an inserted point, which is used to fill up the gap on the object surface. But its normal vector needs to be changed before it can be done, so that during the calculation of lighting these points can be taken to be the same as other points around them. The method adopted to calculate the normal vector is called approximating method, which uses points around to approximate the normal vector. The method is described as shown in Figure 5:

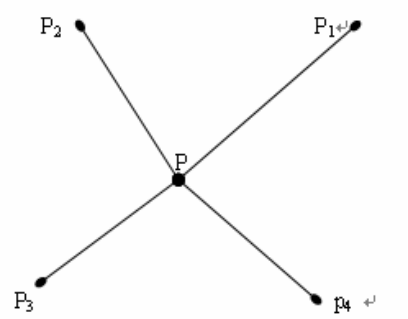


Figure 5: Calculation of normal vector

The normal vector can be calculated as follows: for a point $P(x,y,z)$, define a vector neighborhood $\delta = \{q(x,y,z) \,|\, |P-q| < \varepsilon\}$, where q is a point inside the vector neighborhood, ε is determined by the density of sampled points. Obtain 4 points P_1, P_2, P_3, P_4, from this vector neighborhood which are at the upper left, upper right, lower left and lower right of P and have the shortest distance to it, then the normal vector P_N of P can be calculated through the formula below:

$$P_N = \left[\frac{PP_1}{|PP_1|} \times \frac{PP_2}{|PP_2|} + \frac{PP_2}{|PP_2|} \times \frac{PP_3}{|PP_3|} + \frac{PP_3}{|PP_3|} \times \frac{PP_4}{|PP_4|} + \frac{PP_4}{|PP_4|} \times \frac{PP_2}{|PP_1|} \right] / 4$$

window-dependent algorithm to remove invisible points can be described as follows:

1. Record the maximum values of x and y, define a depth matrix whose size is x × y and set all initial values to 0.
2. assign the value of each entry of the matrix as the distance between the very first point to view point (depth).
3. Divide the depth matrix into 3×3 sub-matrix, compute the followings in the sub-matrix:
   maximum and minimum depth: max, min;
   average value of depth :
   meandep = (max + min)/2;
   threshold =k × (max − min);
   (where k is an unsigned integer)
4. if (meandep < depth of point < threshold)
   {
   property of this point;
   }

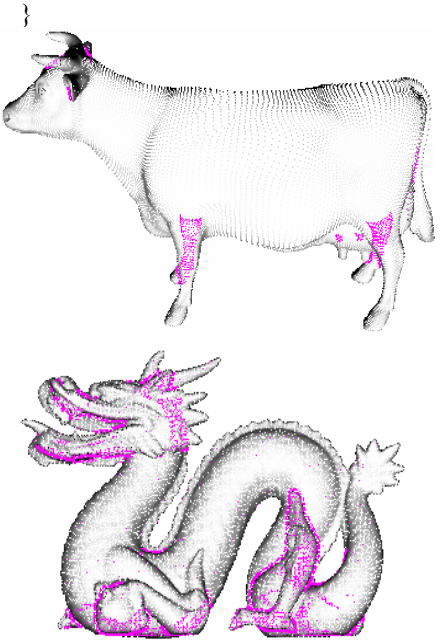


Figure 6: Hidden points detected

After applying the above algorithm, these hidden points (pink points in Figure 6) can be found and their color are then set to background value, so these points will not be displayed in the image space and the objective of hiding these points is achieved. The images with hidden points removed are shown in Figure 7. From Figure 7 it can be observed that the points on cow's leg and dragon's body have been removed. After visibility-detection, image rendering techniques have been applied to render the images with hidden points removed, and images with realistic visual effect are obtained in Figure 8.

3. Conclusions

The objective of producing realistic computer graphic images is to provide a high visual quality. Our objective is to have simple algorithms, low complexity and fast processing time. That is also why we use a pure point-based rendering algorithm. This algorithm does not require us to define area for each point, and there is also no need to determine whether the area is overlapped during painting the color. Therefore, this algorithm is better than splat or Surfel point-based rendering in terms of processing time and complexity. However, as visibility is only related to view direction, if the view direction is changed, we need to compute all the data again. We have processed the most complicated image in Figure 4 (Dragon, 100250 points) by using a Pentium IV computer, the data input time is: 4 seconds approximately, algorithm processing time: 5 seconds approximately.

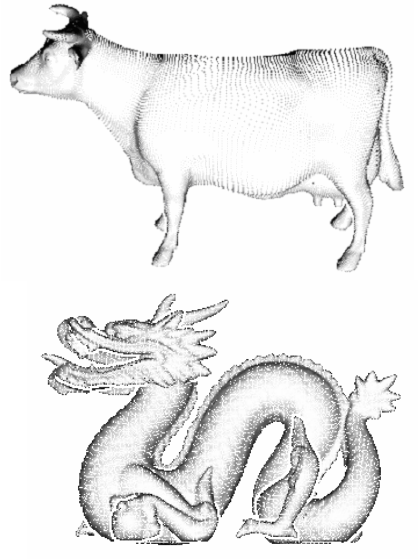


Figure 7: Images after visibility-detection

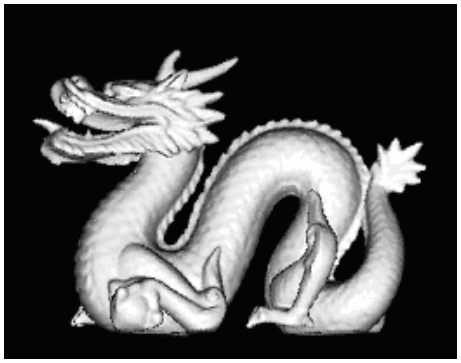


Figure 8: Images with realistic visual effect shown after removing all the hidden points

Acknowledgements

The work was supported by the Doctoral Site Foundation from the MOE, PRC (Grant No. 20010183041).

References

[1] Smith, A. R.: Smooth Operator. The Economist, pages 73–74, March 6 1999. Science and Technology Section.

[2] Levoy, M., Pulli, K., Curless, B.: Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., and Fulk, D. The Digital Michelangelo Project: 3D Scanning of Large Statues, Proc. SIGGRAPH, 2000.

[3] 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.

[4] Szymon R., Levoy, M.: "Qsplat: A Multiresolution Point Rendering System for Large Meshes", SIGGRAPH'2000 Proceedings, New Orleans, LA USA, 343-352.

[5] Pfister, H. , Zwicker, M.J. van Baar, J. and Gross, M.: Surfels: Surface Elements as Rendeing Primitives. In Computer Graphics, SIGGRAPH 2000 Proceedings, pages 335–342. Los Angeles, CA, July 2000.

[6] Catmull, E.: A Subdivision Algorithm for Computer Display of Curved Surfaces [D], University of Utah, Salt Lake City, 1974.

[7] Zwicker, M., Pfister, H., van Baar, J., Gross, M..:Surface splatting. In: Proc. of ACM SIGGRAPH 01. pp. 371–378.

[8] Zwicker, M., R¨as¨anen, J., Botsch, M., Dachsbacher, C., Pauly, M.: Perspective accurate splatting. In: Proc. of Graphics Interface 04.

[9] Wu, J., Kobbelt, L., 2004. Optimized subsampling of point sets for surface splatting. In: Proc. of Eurographics 04.
[10] Pfister, H.,Zwicker, M. ,van Baar, J.and Gross,M.:Surfels: Surface Elements as Rendering Primitives, to appear in Proc. SIGGRAPH 2000, July 2000.
[11] Schaufler, G., Jensen, H. W.: Ray tracing point sampled geometry.Proc.Eurographics Rendering Workshop (1998).
[12] Tobor, I., Schlick, C. , Grisoni, L.: Rendering by surfels. In Proceedings of Graphicon, 2000. 3, 8, 25.