Point Based Representations for Hierarchical Environments*

Kedarnath Thangudu, Lakshmi Gade, Jag Mohan Singh, P J Narayanan
Center for Visual Information Technology
International Institute of Information Technology, Hyderabad, India
{kedar@students., lakshmi_g@students., jagmohan@research., pjn@} iiit.ac.in

Abstract

The advent of advanced graphics technologies and improved hardware has enabled the generation of highly complex models with huge number of triangles. Point-based representations and rendering have emerged as viable representations for high-quality models in this scenario. These methods have been demonstrated only on high resolution, compact objects so far. They have to be adapted to environments that extend over large areas to be considered serious representations. In this paper, we present an adaptation of the point-based representation to large, hierarchical environments. We show how point-based data can be generated by sampling polygon-based representations. We also show the combination of an object hierarchy and multiresolution point representations and develop rendering algorithms for the same. The multiresolution representation is constructed during the generation process. We then show a hybrid representation in which the more complex portions of the environment are represented using points and others using the original polygon-based representation. This produces better rendering performance by keeping large, flat regions as triangles. We demonstrate the method on the model of the Fatehpur Sikri which has 14000 objects with over 500,000 triangles.

1. Introduction

Along with the changing technologies and hardware, graphics primitives have also been evolving over the years. Various building blocks have been employed for the efficient rendering of different kinds of datasets. Many complex primitives such as nurbs, polygon meshes, etc have been in use. However, in the recent years, due to the increase in the number and the complexity of primitives required for high quality rendering and the overheads involved in handling these primitives, a new trend has emerged towards exploring points as rendering primitives [8]. Research has shown that point based rendering is ideal for complex geometry. In this paper, we explore the applicability of points as rendering primitives for large environments.

*This work was partially funded by the Naval Research Board. We thank NCST/CDAC for the Fatehpur Sikri model.

1.1. Points as rendering primitives

In polygon-based graphics, linear interpolation of the properties at the vertices results in approximation of the real world object. But, in point based graphics, color, lighting and other auxiliary properties are computed for each point, hence giving a realistic appearance to the rendered object. Also, for highly complex models, using algorithms with low per-primitive computations (such as points) would be less expensive and appropriate, while polygon-based representations sometimes map to areas smaller than a pixel, causing redundant computations. Further, geometric subdivision ultimately leads to points and conceptually points would be the smallest, simplest limiting primitives that can be used for rendering. Moreover, 3-D acquisition techniques of real world objects such as stereo capture, range scanning, etc generate points as output. For the acquired data, triangulation can be inaccurate and expensive while point representations would be more natural. Also complications do not arise due to the topology of the model as connectivity information of surfaces is not maintained in point representations.

1.2. Point-based Graphics

Point-based geometry is equivalent to sampling a continuous surface to obtain various properties such as 3D positions, colors, normals etc. The main phases involved in Point based Representations (PBRs) are Acquisition, Representation, Rendering and Compression.

- *Acquisition:* There are various methods of acquiring point data. By using laser range scanners [9], structured light based scanners etc, high resolution point cloud can be obtained directly. Sampling 3-D polygon based models is another efficient way of generating points. And, stereo capture can also be used to acquire data with the help of the texture and the images of the model from different views.

- *Representation:* The point cloud obtained from these acquisition techniques is generally unordered and independent. In order to increase the rendering efficiency and flexibility of the model, an ordering of these points is required. To achieve this various hierarchical representations such as Octree structures [3], QSplat [12], BSP trees etc have been proposed.
- *Rendering:* Many optimizations applicable on TBRs (Triangle-based representations) such as LoD selection, culling etc can also be performed with points when appropriate representations are used. As opposed to triangles, rendering the model as points may result in gaps between the neighboring point samples. These gaps are filled by drawing a circle/square/ellipse, oriented in the normal direction, instead of the point. These are called splats. Each of these splats is associated with an appropriate size, large enough to cover the gap between adjacent points.

2. Related Work

A lot of work has been done in the areas of hierarchical representations for large environment models. Open scene graph [1] provides one such representation which builds an acyclic, directed graph of the model. This facilitates operations such as view-frustum culling, LoD selection etc.

Point rendering algorithms have been aiming at achieving view dependent LoD selection which involves rendering optimal number of points for a view such that they are just dense enough not to leave any holes and the projected size of the splats on the screen remains constant.

QSplat [12] is one such algorithm which describes a multi-resolution hierarchy based on bounding spheres. The points are recursively subdivided according to their spatial locations and are stored as leaves. Their properties are averaged to obtain the values at the intermediate nodes allowing continuous view dependent LoD selection.

Sequential Point Trees [4] is another data-structure based on QSplat, which transforms the hierarchical rendering process into a sequential process and transfers the load of selection of the detail levels and rendering onto the GPU. Chen and Nguyen [5] present hierarchical approaches that smoothly replace the point clouds by the original triangles for close ups.

3. Triangle to Point Based Representations

We are interested in studying point-based representations (PBRs) for large environments. Captured data of such environments are not available and hence we discuss how PBRs can be generated given triangle-based representations (TBRs).

3.1. Generation

Very high resolution models can be converted into point models by retaining the vertices as the points of the PBR. But, if the size of the triangles is slightly bigger, the vertices of the triangle would not be sufficient to represent the whole model. In such cases, the intermediate points have to be obtained by sampling the triangle-based model. The following procedure describes how this can be achieved.

Rendering is equivalent to sampling the surface points. Therefore, each triangle of the 3-D model, with its color, texture and normals, is rendered to an orthographic camera which is aligned normal to its surface. As the camera's view volume should include the whole triangle, its size is set to the longest side of the triangle and the camera is centered over the circumcenter of the triangle. Each pixel of the rendered triangle is a point on its surface. Its color is read from the frame buffer, and its world coordinates are obtained by unprojecting its screen coordinates and the depth obtained from the depth buffer. Normal at each point obtained is calculated by interpolating the normals at the vertices of the triangle.

The resolution of the point data obtained is determined by the ratio of the viewport dimensions to the view-volume dimensions. This ratio is maintained constant to generate points with the same resolution for all the triangles.

$$Resolution = \frac{Viewport\ size}{Frustum\ size}$$

As the sampling resolution is constant, the radius of the splats of all the points obtained is the same. This can be obtained as

$$Radius = \frac{1}{\sqrt{2} * Resolution}$$

Constant sampling direction has to be maintained for all the triangles in a plane. This is determined by direction of the up-vector of the camera, which has to be maintained constant for all the triangles in a given plane. OpenGL omits points on the top and right edges while rendering filled polygons. When two adjacent triangles are sampled such that their common edge is not rendered while sampling either of these triangles, holes are observed. On the other hand, if the common edge is rendered while sampling both the triangles, double points are obtained. Rendering the triangular mesh along with the filled triangle would prevent holes from appearing in the point model. But this leads to double points along all the edges of adjacent triangles. These double points can be removed by a post-processing step during which the points whose distance from their nearest neighbor is less than a threshold.

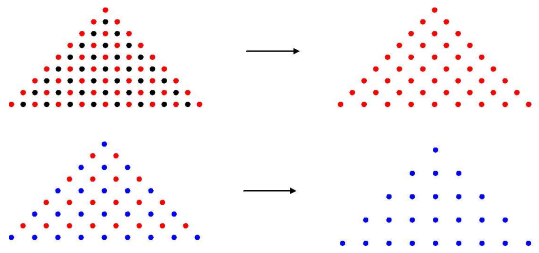

Figure 1. Illustration of obtaining (a) even LoD from an odd LoD, (b) odd LoD from an even LoD.

3.2. Multi-Resolution Representation

There are various techniques to obtain multi-resolution representation of point data. Some algorithms use a local plane of support [7, 11] for this purpose. Point set surfaces [6] use Moving Least Square (MLS), a projection operator which can be used both for upsampling and downsampling of the surface. Further a regular grid can be induced to the points using a local plane and spectral processing can be performed [10] on the point set. Optimal pairing of points using local planes can be used for the decomposition of the point cloud [14]. This is achieved [13] without using local planes by recursive reordering of the point set.

We use the method described by [13] to compute the point cloud without a local plane. Since the points are being sampled from the triangle surfaces, the inherent adjacency information can be exploited to obtain a multi-resolution model, instead of resorting to complex algorithms for pairing of points. The highest LoD is generated by sampling the model with the required resolution and all the lower LoDs are derived from it. The second LoD is derived from the first by skipping alternate points in each row and the third LoD can be derived from the second by skipping alternate rows (Figure 1). The same procedure can be followed for the subsequent detail levels.

If we consider a grid (i along x-axis, j along y-axis) over the points (i.e. the pixel positions in the frame buffer to be un-projected), the points which in the odd LoDs are present when

$$(i \text{ \% } (2^{lod/2})=0) \text{ and } (j \text{\% } (2^{lod/2})=0)$$

and points in even LoDs are present when

$$(i - 2^{lod/2-1}) \text{ \% } 2^{lod/2}=0 \text{ and } (j - 2^{lod/2-1}) \text{ \% } 2^{lod/2}=0$$
$$\text{or}$$
$$(i \text{ \% } (2^{lod/2})=0) \text{ and } (j \text{\% } (2^{lod/2})=0)$$

Since the points in a lower LoD are also present in the higher LoDs, each of these detail levels need not be stored individually. Progressively storing the lower LoDs, followed by the additional points in the next higher LoD in the same model would reduce storage space Figure 2.

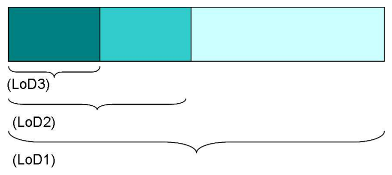

Figure 2. Progressive storage of LoDs

3.3. Procedural Rendering of Points

Each point in the PBR is rendered as a splat. The conventional splat rendering by texture mapping suffers from aliasing effects. Hence we propose to render circles on GPUs procedurally to overcome this effect. Procedural rendering is a method in which geometry is obtained by computing a function rather than from primitives. In this context, we had to draw a circle ($x^2 + y^2=0$). Conventional splatting renders each point as a circle-texture mapped triangle, with in-center as the splat position and in-radius equal to splat radius in the plane perpendicular to the normal of the splat. Here the same procedure is followed, but instead, the triangle is drawn with just the texture coordinates without the texture. Rasterization process interpolates these texture coordinates at the vertices to the pixels on the surface. In the pixel shader each pixel to be rendered is available with its texture coordinate, which is the position of the pixel in the texture space. So we render only the pixels which lie in the in-circle and discard the rest. Rendering circles in this procedure does not suffer from aliasing as, however big a splat might be rendered, we will still draw all the pixels in the circle, unlike the texture-mapping procedure where the texture size is constant resulting in aliasing when scaled.

4. PBRs for Large Environments

Environments are generally represented as triangle-based models. The challenge in handling large environments is the high variation in the resolution of the model in different regions. As they extend over large areas, the small objects which are far away, need not be drawn in full detail. When rendered, the small triangles map to very small areas and hence, using points in these regions would enhance performance. But, converting the entire model into points does not provide any tangible gain over other polygonal representations. This is because there are many large, flat areas in an environment and in order to have fine texture detail, even the points from these areas are required to be generated with high resolution. This produces huge number of points (Table 1) resulting in a considerable increase in the size of the model and poor rendering rates. Moreover, insufficient sampling of the textured, flat regions gives a spotted appearance to the model when zoomed in Figure 3.

Hence, we propose a hybrid representation for large en-

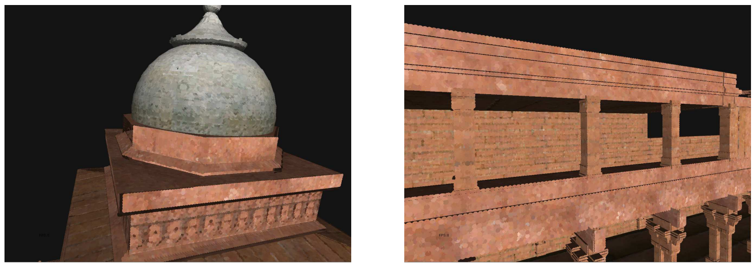

Figure 3. Complete point model

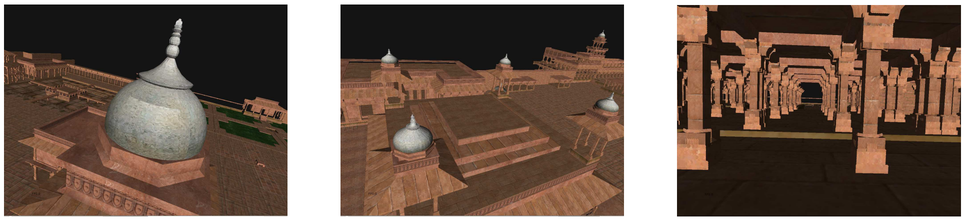

Figure 4. Hybrid Model

vironments which captures only the advantages of PBRs while leaving out its drawbacks.

4.1. Hybrid Representation

In a hybrid representation, the geometry of large flat regions is retained as triangles and only the smooth and curved surfaces are converted to points (Figure 5). Large environment models generally comprise of individual objects with a spatial hierarchy built over them, for example, Octrees, Open Scene Graph etc. So, the hybrid representation is built keeping the hierarchy intact and changing only the geometry of the objects as necessary. Each object is split into two parts, *complex* and *non-complex*. The *complexity* of the geometry could be estimated by various factors such as average size of sides of the triangles, area of the triangle etc. For example, the triangles of objects which have the average side less than a threshold can be defined as *complex* and the rest as *non-complex*. The *complex* part is converted to point based multi-resolution model as described above and the *non-complex* portion is retained as triangle based model. As the large flat regions are rendered as triangles, they retain their fine texture detail and the curved regions are rendered with high detail with per-point normal and lighting computation (Figure 4).

As only the *complex* part of the model is converted to points, we get a compact model. The statistics of the complete point model and hybrid model are given in Table 1. Figure 6 depicts the distribution of points (blue) and triangles (red) in the model in some regions.

4.2. Rendering the Hierarchy

The hierarchical model, which contains the geometry at the leaf nodes and the spatial hierarchy over it, is loaded into a tree-based

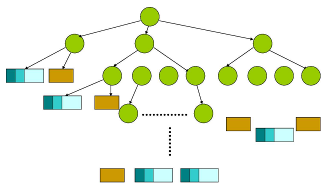

Figure 5. Hybrid scenegraph with internal nodes (green), TBRs (orange)& multiresolution PBRs (cyan).

data-structure. Each node in the tree could either be a leaf nodes or an intermediate node. As a hybrid representation is being used, the leaves could either be points-based models, or triangle-based models.

Algorithm 1 TraverseHierarchy(node)

```
if $node\ not\ visible$ then
    Skip the branch
else if $node\ is\ a\ leaf\ node$ then
    if $TBR$ then
        Render Triangles
    else if $PBR$ then
        Select $LoD$
        Render point model ($LoD$)
    end
else
    for each child in $children(node)$ do
        $TraverseHierarchy($ node $)$
    end
end
```

In each rendering pass, the hierarchy is traversed (Algorithm 1) recursively, and the objects present in the view volume of the camera are selected by performing *visibility culling* [2]. This is done

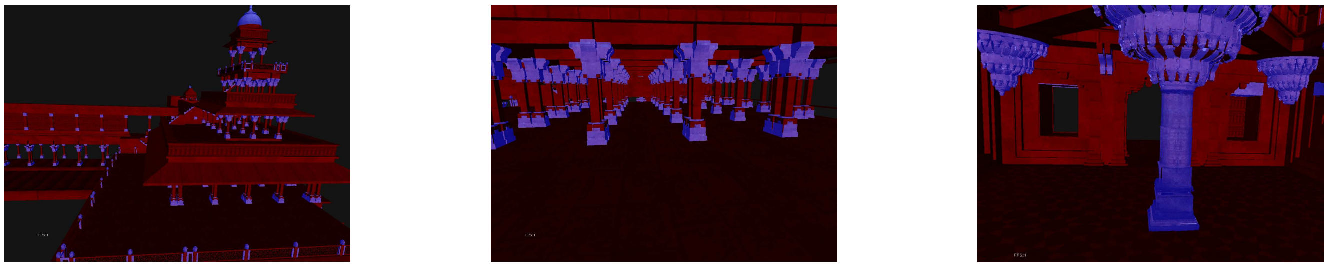

Figure 6. Distribution of points (blue) and triangles (red).

Mdl No.	Total no. of Triangles	Complete Point Model		Hybrid Model						
		No. of points	R.Time (msecs)	No.of Triangles	No. of Points					R.Time (msecs)
					LoD1	LoD2	LoD3	LoD4	LoD5	
1	509933	240237568	24050	116491	5967721	3041851	1732615	908521	574193	3000
2	187403	11583049	1200	13019	1771404	888982	454740	255008	172886	200
3	123619	14979628	1500	37939	1921202	956887	486566	263773	167562	250

Table 1. Statistics of points and triangles in complete point and hybrid models. R.Time is the rendering time

by checking the intersection of the bounding box of the nodes with the frustum and culling those nodes falling completely outside the frustum, along with their subtrees.

Further, as the point-objects are represented using multiple discrete LoDs, they are drawn with the appropriate LoD depending upon the distance of the nearest point on the object from the camera. This procedure increases the rendering speed and reduces redundancies.

5. Conclusion

Points are an excellent choice for highly detailed models. But, when used for large, flat areas, they do not provide any benefit and further, they lower the performance. Thus, a combination of points and triangles, chosen according to the size of the objects, is an ideal way of rendering different graphics models. As large environments are made of objects with diverse sizes, the hybrid model would work especially well.

References

[1] http://www.openscenegraph.org/.

[2] T. Akenine-Moller and H. Eric. *Real-time Rendering*, pages 612–614. A K Peters, 2002.

[3] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 53–64, 2002.

[4] D. Carsten, V. Christian, and M. Stamminger. Sequential point trees. In *SIGGRAPH '03*, volume 22, pages 657–662, 2003.

[5] B. Chen and M. Nguyen. Pop: A hybrid point and polygon rendering system for large data. In *IEEE Visualization*, pages 45–52, 2001.

[6] S. Fleishman, D. Cohen-Or, M. Alexa, and C. Silva. Progressive point set surfaces. In *ACM Trans. Graph.*, volume 22, pages 997–1011, 2003.

[7] S. Fleishman, D. Cohen-Or, and C. Silva. Robust moving least-squares fitting with sharp features. In *ACM Trans. Graph.*, volume 24, pages 544–552, 2005.

[8] L. Kobbelt and M. Botsch. A survey of point-based techniques in computer graphics. In *Computer and Graphics*, volume 28, pages 801–814, 2004.

[9] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira Ginzton, M. Anderson, S. Davis, J. Ginsberg, J. Shade, J, and D. Fulk. The digital michelangelo project: 3d scanning of large statues. In *SIGGRAPH '00*, pages 131–144, 2000.

[10] M. Pauly and M. Gross. Spectral processing of point-sampled geometry. In *SIGGRAPH '01*, pages 379–386, 2001.

[11] M. Pauly, M. Gross, and L. P. Kobbelt. Efficient simplification of point-sampled surfaces. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 163–170, 2002.

[12] S. Rusinkiewicz and M. Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH '00*, pages 343–352, 2000.

[13] J. M. Singh and P. J. Narayanan. Progressive decomposition of point clouds without local planes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, 2006.

[14] M. Waschbüsch, M. Gross, F. Eberhard, E. Lamboray, and S. Wurmlin. Progressive compression of point-sampled models. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 95–102, 2004.