ISSN 2250-0987
Mandakini kaushik et al, UNIASCIT, Vol 1 (3), 2011, 101-106

Achieving a Smooth Interpolation between the Points Using Modified Rendering Algorithm

Mandakini Kaushik,
M.Tech.(CSE) Scholar, Dept. of CSE,
Rungta College of Engg. & Tech.,
Bhilai – 490 024(C.G.), INDIA

Kapil Kumar Nagwanshi
Reader , Dept. of CSE,
Rungta College of Engg. & Tech.,
Bhilai – 490 024(C.G.), INDIA

Dr. Lokesh Kumar Sharma
Head, Dept. of CSE,
Rungta College of Engg. & Tech.,
Bhilai – 490 024(C.G.), INDIA

ABSTRACT— Rendering the points is a big problem due to lack of connectivity information. But lack of connectivity introduces several artifacts while in the process of rendering like aliasing and holes in the rendered model. In recent years point-based geometry has been used efficient rendering and for flexible geometry processing of highly complex models. Based on their fundamental simplicity, points have motivated a variety of research on topics such as shape modeling, object capturing, simplification, rendering and hybrid methods. In this paper we will discuss several rendering algorithms and introducing a new point rendering algorithm, a combination of ideas from different algorithms which includes advantages of different algorithms to give high quality rendering.

Our work has the basis of understanding of the various popular algorithms for point rendering like qsplat and elliptical weighted average splatting. We will discuss advantages and disadvantages of each of the approaches and we will define New Rendering System which leads to high quality rendering. Hence we propose a new Combination of modified Qsplat and Surface splatting Rendering Algorithm.

Keywords-qsplat, surface splatting, rendering.

I. INTRODUCTION

In the world of computer graphics, rendered images are represented by a collection of objects. These objects are often composite, constructed from a number of more basic primitive objects. Typically these primitives have been based on volumes (Constructive Solid Geometry) or manifold representations (splines, polygons). These methods are good for large objects of reasonable complexity but inefficient at representing objects with high levels of detail. Recently a third type of primitive has been investigated, which uses points. Using points as the rendering primitive, surfaces are constructed from a cloud of points and this is known as Point-Based Rendering (PBR).

In recent years Point rendering has received a lot of attention because of the need to store and render objects of high complexity. In this context, point rendering is potentially more efficient than other known rendering methods. Point rendering algorithms can be classified according to how they reconstruct the surface from the samples used. Point-based rendering (PBR) is a new rendering paradigm that uses sample points from surface models as display primitives. The major challenge in PBR techniques is to achieve a smooth interpolation between the points that are irregularly distributed over a surface. This surface reconstruction is view dependent and has to be repeated for every novel viewpoint. Various approaches exist for rendering point sets.

In our work we are only in concern of propose a novel. Custom, high quality point bases renderer which capture the advantages of various point based rendering approaches while, at the same time, eliminates their cons. Each algorithms has its own advantages and disadvantages. Our goal is to get a high quality rendering system. For that we need to combine advantages of existing algorithms and eliminate disadvantages of each approach. Hence we propose a new Combination of modified Qsplat and Surface splatting Rendering Algorithm to get good quality and time efficient rendering. We started off by implementing the QSplat algorithm and made several changes to the source code so that it allowed us to give input in our own format. QSplat had its own formulae, depending on input polygonal mesh model, for determining the splat radius. Since we use point models, *without any connectivity*, as our input, QSplat method of determining the splat radius does not remain valid. Hence, we used the traditional k-nearest neighbor approach to find out radius of each splat. We also looked at the theoretical and implementation details of surface splatting. During second stage we refined and optimized the already implemented QSplat and surface splatting algorithm efficiently and independently with the help of pre-existing point rendering systems like QSplat and PointShop3D. QSplat algorithm was implemented using Octree. On the other hand, with surface splatting approach, we implemented surface splatting algorithm just by using linear list of points. We also implemented differential point method. Having these three implementations of different rendering algorithms in hand, our proposed method can be efficiently implemented by combining these independently implemented algorithms.

Our approach is based on modifying Preprocessing algorithm (Surface splatting) and Octree Building procedure (QSplat) then introducing the New Point Based rendering algorithm using combination of modified QSplat and Surface splatting procedure.

II. BACKGROUNDS AND RELATED WORK

There is a large amount of work related to point based rendering. Maximum researchers have worked on the Point based rendering basis of simplifies the rendering pipeline by unifying vertex and fragment processing allowing for implementation of an efficient and flexible rendering pipelines avoiding redundant functionality. Point based rendering algorithms are output sensitive which means rendering complexity is not much dependent on the scene complexity. Point based rendering methods occupy less storage space as compared to polygonal rendering due to lack of connectivity information. Further, compression methods are also available for storing point models, which reduces the required storage space.

Point sample rendering has been done in two different approaches, point rendering and splatting. Point rendering uses zero-dimensional points as primitives, while in splatting the primitives are planar reconstruction kernels (for example, disks around points). Not all methods based on point sampling fit into these categories. The classification used here is mainly to bring out important differences between various point rendering algorithms.

Available Point Rendering Algorithms:

A. QSplat: Point Rendering Approach

In 2000 by Rusinkiewicz and Levoy [5] developed a real-time rendering system called QSplat as a part of Digital Michelangelo project to render complex models acquired from 3D scanners. QSplat is a point-based rendering system that uses the visibility culling, LOD control, and geometric compression to render very large triangular meshes. QSplat uses a precomputed bounding sphere hierarchy that performs a hierarchical clustering of points that represent the original model. The inner nodes in the hierarchy store attribute (position, normal, color) that are averages of the children that have been collapsed into the cluster.

QSplat is mainly a two step algorithm:

Preprocessing the large input model: Preprocessing algorithm begins with a triangular mesh representing the model to be encoded. This triangle mesh is used to calculate the normal information and radius of bounding sphere (such that no holes left). Next step is to build up the QSplat tree by splatting the set of vertices along the longest axis of its bounding box, recursively computing the two sub trees, and finding the bounding sphere of the two children spheres. As the tree is built up, per-vertex properties (such as normal and color) at interior nodes are set to the average of these properties in the sub trees (figure 2.1). When the recursion reaches a single vertex, we simply create a sphere whose center is the position of the vertex. A quantization scheme is used for store the tree hierarchy.

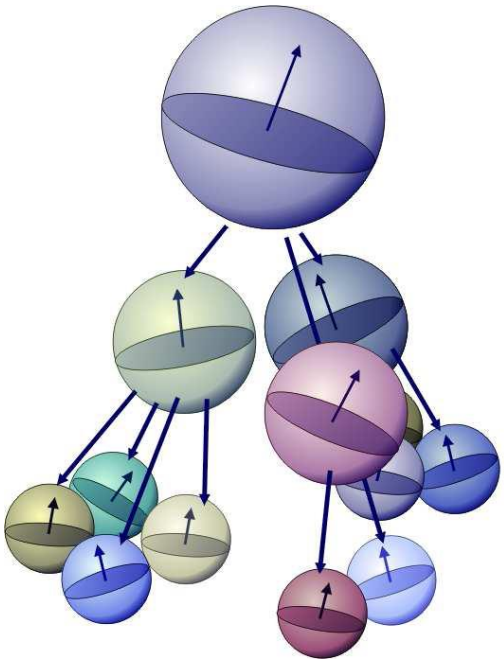

Figure 2.1: QSplat data structure

Rendering Algorithm: The rendering algorithm is a recursive method which traverses the bounding sphere hierarchy and is as shown below.

```
Procedure TraverseHierarchy (node)
1: if node is not visible then
2: skip this branch of the tree
3: else if node is a leaf node then
4: draw a splat
5: else if benefit of recursion is too low then
6: draw a splat
7: else
8: for each child in children (node) do
9: TraverseHierarchy (child)
10: end for
11: end if
```

The above algorithm can be summarized as follows. First if node is not visible (which essentially means that Hierarchical frustum / back face culling) skip that branch in the tree. Next if size of the bounding spheres (and thus the point spacing) is below a given splat sizes then fixed size splats are drawn into a z-buffer to obtain a hole free image. If the above two conditions fail then traverse its children. Figure 2.2 shows QSplat tree traversal and model that can be rendered progressively at that particular level of traversal

of the tree. QSplat provides a facility to user to render the models at interactive frame rates. QSplat also supports multi resolution by providing proper LOD(Level of Detail) control which can done by the breadth order traversal of QSplat data structure up to certain level. QSplat method guarantees high quality rendering. But if the given input model is not enough sampled then this might not give good results. QSplat doesn't handled any aliasing artifacts and even proper texture filtering.

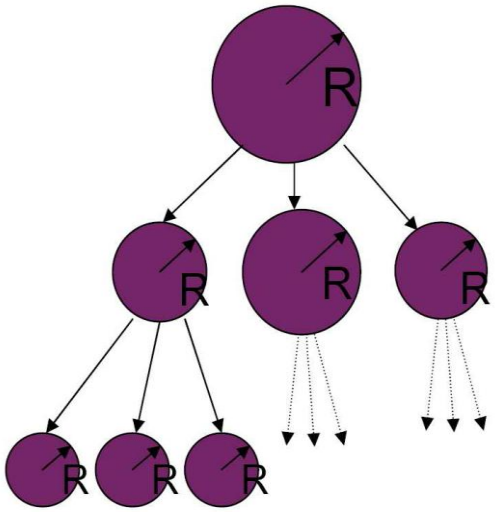

Figure 2.2.QSplat Data Structure: Quad tree representation where R is radius of the bounding sphere

B. Splatting algorithm

Splatting algorithm proceeds by iterating over all sample points: First, the point is shaded using some local illumination model. Then, the footprint or splat is computed by projecting the reconstruction kernel to image space. The splat is now raster zed and accumulated in the framebuffer. After all samples have been processed, all the values in the framebuffer have to be normalized by the accumulated weights of the warped reconstruction kernels.
1: for each sample point do
2: Shade surface sample
3: Splat = projected reconstruction kernel;
4: Rasterizing and accumulate splat;
5: end for
6: for each output pixels do
7: normalize;
8: end for
Second step of the algorithm is Filter normalization: In irregular point sets of filter normalization problem arises which means that the resampling filters don't sum-up to one. In other words if we simply choose weights of the reconstructed kernel to be the sampled color at the points these colors won't reproduce faithfully. This can be avoided by using per pixel sampling after sampling filter values in the output pixels.
Our work also implemented differential point method. Having these three implementations of different rendering algorithms in hand, our proposed method can be efficiently implemented by combining these independently implemented algorithms.

III. PROBLEM STATEMENTS

The main challenge of point based rendering (PBR) techniques is to achieve a smooth interpolation between the points that are irregularly distributed over a surface. We will discuss advantages and disadvantages of previous rendering approaches. QSplat method is an out-of-core rendering technique which is mainly developed to visualize the large densely sampled models at interactive frame rates.

QSplat method has following advantages:

• Explores a simple, hierarchical data structure for visualizing huge point clouds
• Uses efficient encoding scheme to store preprocessed hierarchy into a file format
• Uses hierarchical visibility culling (back face culling, frustum culling) which excludes unnecessary visibility computation for each point.

Thus QSplat system provides efficient data structure for hierarchical culling as well as for level of detail (LOD) control.
QSplat approach has its own limitations and assumptions, which are listed as follows.
• Input models for preprocessing is triangular not raw point cloud.
• Appropriate radius for each point is calculated so that there won't be any holes in the rendered image. However, this radius calculation uses the connected triangular mesh information. This dependency on triangular mesh is not desirable.
• Aliasing artifacts are not handled. (Occurs when more splats overlap)
• No proper texture filtering method is used.
• Input has to be sampled sufficiently.

Problem Definition: After having a look at the pros and cons of the available major point based rendering algorithm, we propose a high quality point based renderer which captures the advantages of various PBR approaches and eliminates their cons. It addresses the following issues in our proposed algorithm

1. Takes a custom point model as an input to system, sans any connectivity information, as opposed to Q-Splat.

2. Provides appropriate hierarchy (octree) for hierarchical culling and LOD control.
3. Applies high quality anti aliasing technique to handle aliasing artifacts.
4. Edges are represented correctly by orienting the surfels according to the distribution of points in the object space.

IV. PROPOSED ALGORITHM

We present a novel, hierarchical, fast and high quality point rendering algorithm. Our new renderer captures the advantages of major point based renderers available while eliminating their cons. Let us see how we modify the available point rendering algorithms so as to make them more advantageous, and finally combine them under one algorithm**.** The proposed system is based on combination of differential points, qsplat and surface splatting algorithms.

a. Modified QSplat

In general, bounding sphere hierarchies are considered as space and time efficient. They are typically stored as trees, whose inner nodes consist of spheres that bound volume of all children nodes. Their leaf nodes consist of simple primitives, disks around points. This results in a level of detail hierarchy allowing view-dependent refinement and use of several speedup techniques, especially visibility culling. The basic data structure of the modified QSplat algorithm is an Octree and its construction is as mentioned in Algorithm 1

Algorithm 1 Octree Building procedure

Procedure BUILD-OCTREE (P(p_1,……,p_n)

1: **if** number of points less than 8 **then**
2: return CombineNodes(P)
3: **end if**
4: Calculate p_{avg}
5: Divide point cloud into S1, ... , S8 octants based on p_{avg}
6: **for** each octant Si **do**
7: Ni = BuildOctree(Si)
8: **end for**
9: return CombineNodes(N1, . . . ,N8)

BUILD-OCTREE is a recursive procedure to build bounding sphere hierarchy. *CombineNodes* procedure takes Octree nodes as an input and returns a node whose attributes (like position, color, normal etc.,) are average of input nodes and makes those input nodes as children of the resultant node. PartitionAlongLongestAxis procedure fits a bounding box around the input points, selects the longest axis and partitions the input model along the midpoint of that axis.

The Speed of a point based renderer not only depend of hierarchical culling, tree traversal but also depend on the amount data transfer between CPU and main memory. A better encoding or quantization scheme for point model which results in lesser memory cost, there by leads to faster results.

We also modified QSplat source code to take only point cloud information as an input instead of triangle data which is used in finding effective radius of each surfel. We solved this problem by applying k-nearest neighbor algorithm on the given point data and effective radius assigned to a point sample is calculated as distance of k_{th} farthest point from the given point sample. The rendering speed can be further improved by adopting the better hierarchical encoding technique used .

The Figure shows low quality rendered image of the Cornell room with goddess and ganapati models. The main problem with QSplat approach is its lack of ability to do any surface reconstruction or texture filtering to avoid artifacts (normally when splats overlapping is more) in the rendered image. Hence for high quality rendering we adopted the approach of surface splatting while rendering a given scene.

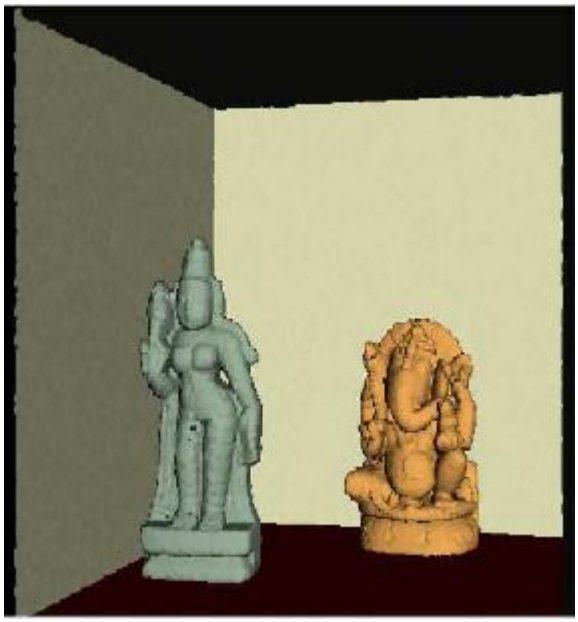

Figure : *Using QSplat*

b. Modified Splatting

Although Surface Splatting gives high quality rendered images (See Figure), it suffers from incorrect splatting at the

edges or corners of objects in the model due to the use of circular splats. To get over this issue, we opt for a better method of Modified Splatting Algorithm

Figure : *Using surface* Splatting

Surface splatting method still suffers from problems at edges and corners of the point models, because it models points as circular splats in object space and ignores the data distribution in the local neighborhood of the point to be projected.

The proposed system is based on combination of differential points approach used for modeling points in object space as elliptical surfels, modified qsplat and surface splatting algorithms. We start by evaluating local surface geometry of the input point cloud as described in Differential Points. We need to create a kd-tree with the given random point cloud and for each point apply covariance analysis to evaluate its normal, major and minor axis lengths of the elliptical surfel. We then actually start building an Octree hierarchy, encode the octree and finally store it to a file as done in Modified Q-Splat. We regard all the above steps as our pre-processing algorithm. The algorithm is as mentioned in Algorithm 2

Algorithm 2 Preprocessing algorithm

---

procedure BUILD-HIERARCHY(P($p_1, \ldots, p_n$)

1: Construct a KD-Tree with the given point cloud
2: **for** each sample point p_i **do**
3: find k-nearest neighbors N_p for each p_i
4: Apply covariance analysis on these k-points
5: major $Axis_i$ = first principle component with length d (distance of kth nearest neighbor of p_i)
6: minor $Axis_i$ = second principle component with length $d*(\lambda 2/\lambda 1)$
7: $normal_i$ = third principle component
8: **end for**
9: BUILD-OCTREE(P)
10: Apply encoding and write to file

The basic rendering data structure used in this approach is a hierarchical space partitioning data structure as in modified QSplat. Each intermediate node of the hierarchy contains a set of samples, each being a representative sample with average coordinates (average of all points in that set), as well as average normal and color. The procedure CombineNode(N1, . . . ,N8) will create a new averaging node from N1, . . . ,N8 and return that newly created node. Furthermore, for efficient view-frustum and back-face culling, each cell also includes the radius and normal cone semi-angle bounding all samples in the set. We use a point octree which partitions the space adaptively to the sample distribution rather than regularly in space like region-octrees which have been proposed more commonly. While rendering, if the view-point is far from the scene, projected area of the surfels will be less. A node in the hierarchy is rendered when its projected area is less than a threshold to add multi-resolution feature to the renderer. Multiresolution feature makes the renderer not to traverse the tree to its greatest depth in the octree to render a surfel, which helps in increasing rendering speed.

c. Combination of modified Qsplat and Surface splatting Rendering Algorithm

The rendering process is a combination of modified QSplat and Surface splatting rendering algorithms. We traverse the preprocessed octree hierarchy using breadth first search (BFS) order as explained in Algorithm3. A node in the hierarchy is rendered when its projected area is less than a threshold or if it is a leaf node. The basic surface splatting idea can be used while displaying surfels.

Algorithm 3 Rendering Algorithm

Procedure TraverseHierarchy (T)

1: Enqueue (Q, T.root)
2: **while** Q is not empty **do**
3: $node_i$ = Dequeue (Q)
4: **if** $node_i$ is not visible **then**
5: skip this branch of the tree
6: **else if** node is a leaf node or projected surfel size is too low **then**
7: Splat = projected reconstruction kernel * UnitGuissian ;

8: Rasterize and accumulate splat;
9: **else**
10: **for each** child **in** children(node) **do**
11: TraverseHierarchy(child)
12: **end for**
13: **end if**
14: **end while**
15: **for** each output pixel **do**
16: normalize;
17: **end for**
18: output the accumulation buffer content to a texture
19: Display the Texture

V. CONCLUSION

Having motivation and applications of using point as better rendering primitive, we had an overview of several popular rendering algorithms like QSplat and Surface splatting. We discussed advantages and disadvantages of both approaches. But surface splatting can't exactly represent the surfaces at the corners and edges efficiently. So we discussed about a new way of modeling points in object space known as differential points which hold local surface geometry information and are adapted to surface orientation. Each of the above methods had some disadvantages. To eliminate them and have a high quality and time efficient rendering system, we proposed and discussed a new Combination of modified Qsplat and Surface splatting Rendering Algorithm which includes benefits from all three approaches. It includes a multi resolution data structure as in modified QSplat where points are initially processed as differential points to capture local geometry and better point blending or rendering procedure as in modified splatting.

VI. REFERENCES

[1] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell,W. Richard Fright, Bruces C. McCallum, and Tim R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Siggraph '01 Proceedings*, 2001.

[2] Baoquan Chen and Minh Xuan Nguyen. POP: A hybrid point and polygon rendering system for large data. In *IEEE Visualization*, 2001.

[3] Wei Chen, Liu Ren, Matthias Zwicker, and Hanspeter Pfister. Hardware-accelerated adaptive EWA volume splatting. In *Proceedings of IEEE Visualization 2004*, October 2004.

[4] Jonathan D. Cohen, Daniel G. Aliaga, and Weiqiang Zhang. Hybrid simplification: Combining multi-resolution polygon and point rendering. In *Proceedings of IEEE Visualization*, 2001.

[5] Markus Gross, Hanspeter Pfister, Marc Alexa, Mark Pauly, Marc Stamminger, and Matthias Zwicker. Point-based computer graphics. Eurographics '02 Tutorial T6, 2002.

[6] J. P. Grossman and William J. Dally. Point sample rendering. In *Proceedings of the 9th Eurographics Workshop on Rendering*, pages 181–192, 1998.

[7] Aravind Kalaiah and Amitabh Varshney. Differential point rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering*, August 2001.

[8] Marc Levoy and Turner Whitted. The use of points as a display primitive. Technical Report TR 85-022, University of North Carolina at Chapel Hill, 1985.

[9] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *IEEE Visualization '02 Proceedings*, 2002.

[10] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. *Proceedings of SIGGRAPH 2000*, pages 335–342, July 2000.

[11] Jussi R¨as¨anen. Surface splatting: Theory, extensions and implementation. Master's thesis, Dept. of Computer Science, Helsinki University of Technology, May 2002.

[12] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 371–378. ACM Press / ACM SIGGRAPH, August 2001. ISBN 1-58113-292-1.

[13] Matthias Zwicker, Jussi Rsnen, Mario Botsch, Carsten Dachsbacher, and Mark Pauly. Perspective accurate splatting. In *Proceedings of Graphics Interface*, 2004.