POINT RENDERING SURFACE SPLATTING

Shiva Sagar B[1], Shashidhar G. Koolagudi[2]

[1]*Dept. of CS&E, National Institute of Technology Karnataka, Surathkal, (INDIA)*

[2]*Assistant Professor, Dept of CS&E, National Institute of Technology Karnataka, Surathkal, (INDIA)*

ABSTRACT

Surface splatting can be defined as a method to render point-sampled surfaces using a combination of object space reconstruction filter and screen space pre filter for each point sample. Artifact aliasing can be avoided through this and it provides a hole-free reconstruction of a point-sampled surface including for moderate sampling densities. This paper describes the implementation of Surface Splatting using webGL. Basically, this includes Rasterization of elliptical splats based on Ray casting, which is an approximation to the original Elliptical Weighted Average (EWA) Also we try to find the limitations and enhance the performance of this approach.

Keywords – Surface splatting, WebGL, Real-time, Pointcloud,

I. INTRODUCTION

Many applications uses point structured representations because they are flexible. Points have exhibited several advantages against classical polygons [1] in the various fields ranging from interactive modelling of surfaces to real time complex picture visualization. [1]

Many Complex and visually stunning models are produced by laser range and image-based scanning .But they generate huge volumes of point samples. This is the biggest drawback with these techniques. Generating triangular meshes from input values and rendering them through mesh reduction technique is the frequently used approach, but many of the scanned meshes are very big for interactive rendering, also reduction of polygon may result in loss of geometric accuracy and texture fidelity, which is not tolerated by some of the applications. Rendering point samples techniques for point samples without connectivity have become the new focus. For efficient storage and rendering of given data forward warping and hierarchical data structures are used. Scanned data must have high texture fidelity and continuous surfaces should be reconstructed from irregular point samples. Achieving both is challenging to rendering techniques. To add to this, transparency and removal of hidden surfaces need to be handled by point rendering.

Texture mapping is a method for combining different textures. By mapping functions to surfaces, visual intriacy can be increased. In the event that any of these functions improperly band limited, aliasing of texture might happen amid projections to raster images. Zwicker M [1] and his Colleagues designed Surface splatting that is generally unique in relation to traditional texture splatting [3]. Their methodology utilizes deferred shading procedure in which surface traits are remodelled before the actual shading estimations, which prompts brilliant every pixel shading impacts. Attributes of surface are reproduced in screen-space by aggregating weighted characteristics held by every splat. Amid the resampling step, artifact aliasing is evaded by utilizing filtering of EWA: High frequencies are uprooted by using screen-space low pass filter before splat rasterization.

Rendering of complex geometrical objects using point rendering has been demonstrated by many researchers. It has been found to be very efficient.

To achieve accelerated rendering temporary storage of point based structures is required. Volume splatting is a technique that is close to surface splatting and point rendering. The circular 3-D reproduction kernel is centered at every Voxel is incorporated along 1 of the dimension to 2-D footprint function. For every voxel projected in screen, 2-D footprint is accumulated into buffer or to sheet buffers which are image aligned. Previous papers addresses aliasing due to deficient rates of resample amid perspective projections. This is avoided by scaling 3-D reproduction kernels utilizing heuristic method. However, surface splatting models the reconstructing and band limiting texture function in bound together framework. In addition, as opposed to pre integrating isotropic 3-D kernel, it makes use of aligned 2-D kernel, giving anisotropic filtering to surface textures.

This paper describes about surface splatting. When compared to various other rendering approaches of points, this makes use of screen-space formulation of Edge Weighted Algorithm (EWA) filters [6], the efficient anisotropic filtering approach available for all interactive frameworks. This in turn make this splatting technique relevant to high resolution range of laser scans, high texture detailed terrain, and also point sampled geometric objects.

There has been a lot of surge towards the field of three dimensional computer graphics over the past years which maybe due to the people's fascination for viewing animated movies. Many of the techniques falls short of expectations. Surface splatting is one of the new techniques in rendering. Hence, we analyse this technique to find its deformity and try to enhance its performance by improving its data structure.

The enhancements proposed in this paper are straightforward,
common sense and exceptionally helpful to give excellent rendering
of point based geometries continuously. In the section II, the brief history of this traditional algorithm has been discussed. In the section IV the methodology that we proposed to perform the improvements has been detailed. Section V tells about the implementation techniques that we used. The results of this paper are very simple and described in section V. Conclusion and future work has been provided in section VI.

II. SURFACE SPLATTING

The essential for the surface splatting approach is the model for the representation of consistent composition works on a point surface based realistic articles. Since, 3-D focuses are irregularly situated, it utilizes a weighted whole of fundamentally symmetric basic capacities. With this model within reach, the errand of rendering point based questions as linking of filtering, sampling and warping of consistent composition capacities [5].

To proceed in detail the working of a surface splatter, It has two phases. Where in the first phase, every specimen is splatted into a buffer and after that the surface is remade and then the pixels are shaded. In the second stage also known as splatting stage which incorporates change operations and undertaking the splats into screen space, where the resampling pieces can be determined. At that point resampling pieces can be rasterized into and a buffer.

A. SPLAT RASTERIZATION

Splat Rasterization works as follows. It determines which pixel is covered by the projected splat. To ensure this, Programmable vertex and Fragment shader units are used for this task. After passing point attributes to shader in texture coordinates the point size will be determined by this programmable vertex. It is similar to the Conservative approximation of screen space size of splat. Modern GPU (Graphical processing unit) generations provide all features of hardware is essential for implementing the approach on the GPU. Today's GPUs are generally optimized for triangular based rendering and does not provide any specialized fixed function hardware for splat rasterization. The most effective way is to draw a point primitives as screen-space squares of a specific size and casting rays for every fragment to test if it is also part of given splat. Under specific conditions, Screen space extents of a splat are underestimated using this approximation, which might cause Rendering artifacts.
A easy solution to the problem lies in multiplying the estimated splat extents maintaining safety factor greater than one. However, this leads the production of a large number of not required fragments. The fragment stage is the performance bottleneck hence, this is not a particularly good solution.

B. ALGORITHM

The objective is to render graphical spaces from point clouds. This was initially stamped by Matthias Zwicker, Jeroen van Baar, Hanspeter Pfister and Markus Gross in 2000. Algorithm supports order independent transparent surfaces, anisotropic filtering, anti-aliasing and hidden surface removal. The algorithm of a simple surface splatter is given by:

- Input set to the algorithm is set of points Point[k] which represents Irregularly spaced points in the 3-D space for which we like to apply Surface splatting algorithm (e.g. Point cloud).
- A 3-D position and a normal vector, which finds out its front and back, represent every point.
- Input points may have colour attributes or colour attributes can be calculated in the Algorithm from 2-D textures for which we like to apply on the final result.

The below pseudo code shows a bare overview of a simple working surface splatter:
Algorithm one: surface splatting algorithm

```
value=0;
// considering all the points sampled in point cloud
while point[value] do
    venture point[value] to screen sample;
    discover resampling portion RHO[value];
    splat RHO[value];
```

```
end
// considering the bit values to activate in the screen
pixel=0;
while FrameBuffer[pixel] do
// highlighting the sampled bit in the frame buffer(setpixel)
        shade pixel;
end
```

We can analyse the algorithm as follows:

- It works on the fact that the colour of every pixel in the screen is a combination and blend of the points which fall into or close to the referred pixel, at a given camera view.
- Effect of the colour of point on the given pixel is calculated using the density and number of its neighbours or for every existing the points.
- Also by applying this algorithm to every point and not neglecting points that behind other points, algorithm can take into consideration the transparency of Surfaces using the alpha value\cite{4}.

III. IMPLEMENTATION

In the above analysis the pixel sampling was taken as a option where the object space was directly rasterized. Because of this during the observation of minification and top frequencies, many pixels got eliminated, which leads to aliasing artifacts. To solve this, in the original framework of EWA splatting, reconstruction Kernels are band limited by convolution with the screen space prefilter. To unravel this, in the first system of EWA splatting, remaking Kernels are band restricted by convolution with the screen space prefilter. By this we achieve a re-examining channel which will be adequately rasterized. The Convolution methodology can be viewed as just on the off chance that it wind up in coming about straightforward and logically decided.

In the meantime, by considering the aftereffects of the maximal estimation of recreation Kernel and the low pass channel, the screen space prefilter is applied. Even this approximation result in two major drawbacks. By looking at the performance, it requires the evaluation for each fragment, both the low pass filter and re-sampling Kernel which is quite expensive due to the fragments. The section stage is as a rule at the bottleneck of splatting frameworks.

The original splatting method makes use of single buffer called Array-buffer. Its main contribution is to showing accurately the individual contribution of various splats for every pixel in screen. However, array-buffer need to be modified to support the required transparency and antialiasing of edge. In our approach, instead using a single buffer we utilize layered fragment buffer. It makes sure that fragments that over are stored separately and added later stage in a periodic fashion.

III. METHODOLOGY

We make use of webGL to show the improvements of the proposed technique. WebGL is a tool based on openGL-es which needs no help of driver support.

WebGL can be dubbed itself as web version of OpenGL. It is used for rendering of 2-D and 3-D point objects.

A. POINT CLOUDS

The splatting technique makes use of point clouds to get the high quality splatting [5]. Point clouds are nothing but the set of data points in some direction framework. In a three-dimensional direction framework, these points are generally characterized by X, Y, and Z directions, and frequently are expected to speak to the outer surface of an item.

Modular approach has been used for the implementation. Point cloud which is implemented as a matrix function with the help of jQuery framework is essential for high texture splatting.

Since webGL is browser based, our implementation does not require a separate GUI. The input in the point cloud matrix will be considered then it will output the sampled object according to the varying point size.

V. RESULTS

A simple surface splatter using the accelerated techniques has been implemented in webGL. The following results are derived by executing the accelerated algorithm on a Intel i5 CPU of 2.6 GHz with 4GB RAM without a dedicated GPU. Table 1 shows the render times obtained without using any of the above discussed acceleration methods whereas Table 2 shows the render time using the acceleration method discussed above.

- Figure 1 shows the rendered image of the 3D lion with the point size of 0.2 (maximum Point size: 3.0).
- Figure 2 shows the rendered image of the 3D lion with the point size of 3.0 (maximum Pointsize: 3.0).
- Figure 3 shows the rendered image of the 3D Terrain with the point size of 3.0 (maximum Pointsize: 3.0).

Table1: Table showing page load time in webGL without speed-up technique

Sl.No.	Pageload Time (sec)	Figure Ref.
1.	1.07	1 and 2
2.	1.33	3

Table2: Table showing page load time in webGL with speed-up technique

Sl.No.	Pageload Time (sec)	Figure Ref.
1.	0.39	1 and 2
2.	0.49	3

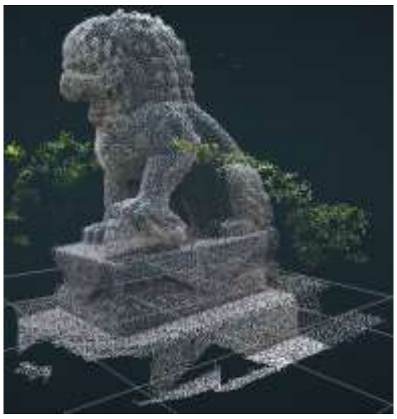
Fig. 1 Point Size: 0.2

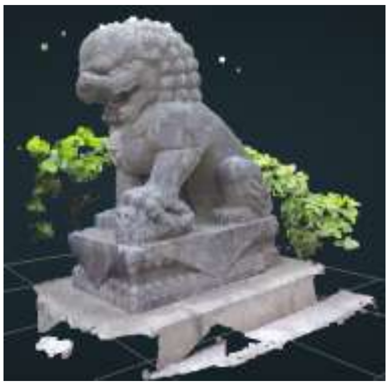
Fig. 2 Point Size: 3.0

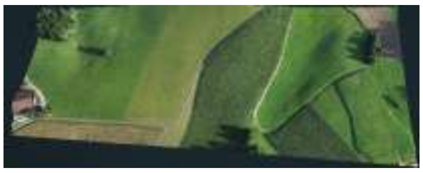
Fig. 3 Point Size: 3.0

The optimized version of the algorithm provides much sharper representation of the images. To augment the discussion, the Table 1 and Table 2 shows the page load times with and without using the improved strategies. We can also observe that more the number of points provided finer the rendered image is and the time taken also increases proportionately.

V. CONCLUSION AND FUTURE WORK

We have described the implementation of improved splat rendering pipeline using webGL. Efficiency of this is due to a fast rasterization procedure that requires a simple setup. Surface splatting technique has some drawbacks like “Circular splats in the object space are not adapted to surface orientation which may leads to artifacts at the corners and edges of the rendered model". Different variations and modifications of Surface splatting do exist. Many variations have GPU based implementation which allows Parallelized Computation.

Variations that are optimised for specific purposes E.g. facial rendering can also be found. There are other variations which are based on Phong shading techniques to provide the detailed lighting of surfaces.

Even though the webGL implementation of the above approach is proficient, further improvements to the method is possible by extending the rendering to uncoloured LIDAR (Light Detection and Ranging) data. There are many points in the implementation which are unnecessary or invisible has to be removed without affecting the normal display of the output. In addition, instead of just improving the data structure of the technique data compression can also be done for faster loading of Graphical Interface web pages.

The above representations namely Figure 3 clearly explains the limitation of the surface splatting approach. The webGL implementation of the technique fails to load the LIDAR point clouds and to render them. This can be the main drawback of this technique that has been identified in this paper.

REFERENCES

[1.] Mario Botsch, Michael Spernat, and Leif Kobbelt. Phong *splatting In Proceedings of the First Eurographics conference on Point-Based Graphics, pages 25–32. Eurographics Association, 2004*

[2.] Takashi Kanai, Yutaka Ohtake, Hiroaki Kawata, and Kiwamu Kase. *Gpubased rendering of sparse low-degree implicit surfaces.* In Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia, *pages 165–171. ACM, 2006*.

[3.] Ned Greene and Paul S Heckbert. *Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. Computer Graphics and Applications, IEEE, 6(6):21–27, 1986.*

[4.] Henrik Wann Jensen and Per H Christensen. *Efficient simulation of light transport in scences with participating media using photon maps.* In Proceedings of the 25th annual conference on Computer graphics and interactive techniques, *pages 311–320. ACM, 1998.*

[5.] Tim Weyrich, Simon Heinzle, Timo Aila, Daniel B Fasnacht, Stephan Oetiker, Mario Botsch, Cyril Flaig, Simon Mall, Kaspar Rohrer, Norbert Felber, et al. *A hardware architecture for surface splatting. In ACM Transactions on Graphics (TOG), volume 26, page 90. ACM, 2007.*