# Algebraic Splats Representation for Point Based Models

Naveen Kumar Bolla and P. J. Narayanan
Center for Visual Information Technology, IIIT Hyderabad.
{naveenb@research., pjn@} iiit.ac.in

## Abstract

*The primitives of point-based representations are independent but are rendered using surfels, which approximate the immediate neighborhood of each point linearly. A large number of surfels are needed to convey the exact shape. Higher-order approximations of the local neighborhood have the potential to represent the shape using fewer primitives,simultaneously achieving higher rendering speeds. In this paper, we propose* algebraic splats *as a basic primitive of representation for point based models. An algebraic splat based representation can be computed using a moving least squares procedure. We specifically study low order polynomial splats in this paper. Quadratic and cubic splats provide good quality and high rendering speed using far fewer primitives on a wide range of models. They can also be rendered fast using ray tracing on modern GPUs. We also present an algorithm to construct a representation of a model with a user-specified number of primitives. Our method to generates a hole-free representation parametrized by a smoothing radius. The hole-free representation reduces the number of primitives needed by a factor 20 to 30 on most models and by a factor of over 100 on dense models like David with little or no drop in visual quality. We also present a two-pass GPU algorithm that ray-traces the algebraic splats and blends them using a Gaussian weighting scheme for smooth appearance. We are able to render models like David at upwards of 200 fps on a commodity GPU using algebraic splats.*

## 1. Introduction

Despite triangles being the traditional primitives for rendering, the point based representation has become popular in the last few years. The major reason is the flexibility and topology independence that comes with isolated points. We also see triangle meshes becoming denser with triangles projecting to a small number of pixels on the screen. The screen resolution is not growing as fast as the density of the triangles. Point-based representations can exploit this situation better.

Point based models can be visualized by rendering the points directly. This requires a very large point cloud to approximate the shape well. Points need extrapolation to appear as a continuous surface as we move closer to the surface. Splats or surfels are popular to extrapolate points in a small neighborhood. They linearly approximate the surface near the point. Various algorithms for point based representation and rendering based on linear splats were discussed in the survey [14]. Surface splats are piecewise linear primitives which provide a least square approximation to the smooth surface. Differential geometry states that ellipse shaped splats provide the best linear fit. Circular splats are easier to render and provide similar quality [14].

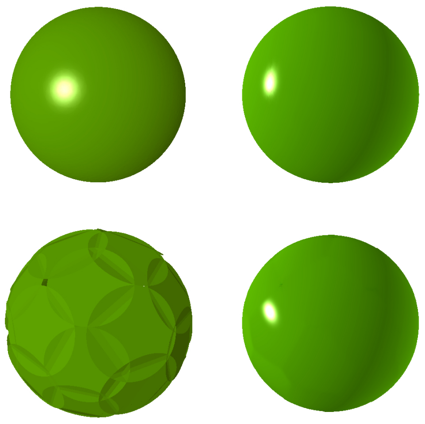
**Figure 1. Top two spheres rendered using 3200 linear (left) and quadratic (right) splats. Bottom spheres rendered using 52 linear and quadratic splats. Non-linear splats can approximate the shape better with fewer points.**

Large number of linear splats are needed to represent the shape of most smooth models. Since the number of linear primitives needed is large, the rendering is slow. Non-linear patches can approximate the shape of the model well in large neighborhoods. Thus, piecewise non-linear patches can represent the model using fewer number of primitives with same or better quality than linear primitives. Although rendering a non-linear patch is slower compared to a linear splat, overall speed of rendering could be improved. The approximation error can also be less using a small number of higher order primitives. Figure 1 shows the advantages of using non-linear splats over linear ones. Quadratic splats can approximate a sphere even with a few tens of primitives.

In this paper, we consider splats that have an algebraic form. Specifically, we use primitives defined by polynomial functions in the local neighborhood. We call these primitives *algebraic splats*. We represent a given point set with a user-specified small number of algebraic splats with optimal rendering quality. This is done by decimating the point set and jointly approximating each using a local algebraic surfaces based on the MLS procedure. Our rendering provides smooth surfaces with normals everywhere. We can render polynomials directly on today's GPUs using ray-tracing because of the semi-implicit nature of the splats in the local reference domain. Our method is efficient and can represent and render the David model using about 30K (or 0.8%) algebraic splats with little or low reduction in visual quality (Figure 2) at 220 fps. We show results on several standard models in this paper.

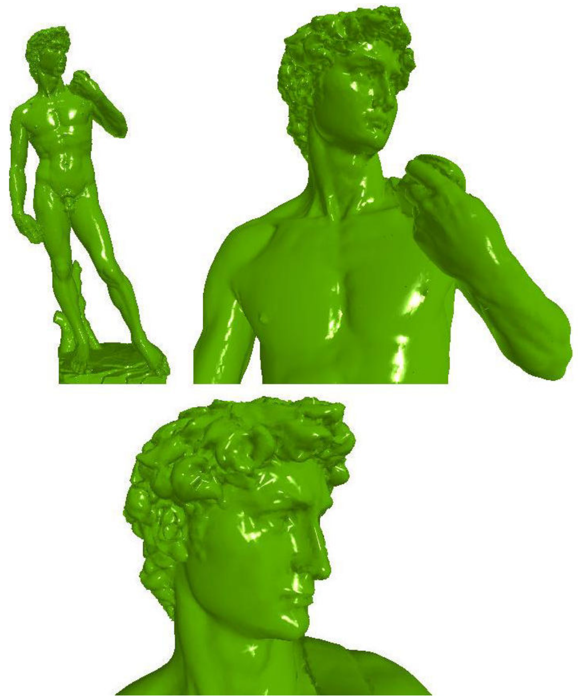
**Figure 2. David model rendered using around 30K algebraic splats at 220 fps on nVidia's GTX 280.**

### 1.1. Related Work

The important feature of point-based graphics is mesh independent surface reconstruction. A point set surface (PSS) [2, 3] is a smooth representation of a set of points constructed an MLS technique [15]. As an approximation scheme, moving least-squares is insensitive to noise and can be approximated locally [15]. Many formulations of PSS are used for surface reconstruction [4, 8, 9], with linear rendering primitives. The Algebraic Point Set Surfaces (APSS) [10] locally approximate the data using spheres instead of points, which significantly improved stability of projection under low sampling conditions. APSS gives good approximation of the shape at sharp edges. Sphere fitting mechanism uses algebraic distances between points instead of geometric distances. The planar MLS can be obtained as special case of the APSS projection.

The second class of surface approximation algorithms are based on Multi-level Partition of Unity (MPU) [20]. Their representation is defined by a blend of locally fitted implicit quadric. Sparse Low-degree IMplicits (SLIM) [19], approximate the geometry using bi-variate polynomials. Efficient rendering is done by blending the primitives in screen space. A GPU based rendering algorithm was also proposed for them [12]. However this approach still suffers from the polynomial fitting limitations and does not properly define a smooth surface due its view dependent nature. These are sensitive to noise and are variant to the rigid transformations. These surfaces are parametrized over an $\varepsilon$-ball neighborhood of points, which is not a suitable for irregularly sampled or noisy models.

Several ray tracing techniques for PSS on CPU have been proposed [1, 24], but are slow. Recently Linsen *et al* [16] proposed a splat based ray tracing of point clouds on CPU. GPU rendering is more popular these days and high-level primitives for GPUs are being proposed for ray tracing [21]. Several methods for GPU-based ray tracing of implicit surfaces have been proposed [7, 13]. Sigg et al [22] rendered molecular models directly using spheres, cylinders and ellipses using ray-casting. Stoll *et al* [23] proposed an incremental ray casting method for quadratic surfaces on the GPU. Loop and Blinn proposed a GPU based algorithm for rendering up to fourth order algebraic surfaces defined by tri-variate Bezier tetrahedra [18].

## 2. Algebraic Splats

There are many ways to approximate the surface from an unstructured point cloud. Moving Least Squares (MLS) surfaces are attractive as they can be constructed using local computations. The MLS procedure approximates the local neighborhood by defining a polynomial of second, third or fourth order in a local reference domain. The smoothness of the surface can be controlled and the approach is well suited to filter noisy input data. We create algebraic splats from the MLS polynomials by bounding each to a disc in the local parametric domain. These polynomials can be rendered directly using ray tracing.

## 2.1. Moving Least Squares

Moving Least Square approximation has two stages. First, a local reference domain, $H_i$, at a point $r_i$ is established by fitting a weighted least square plane to the points in the neighborhood. The normal $(n, D)$ is estimated by minimizing $\sum_{j=1}^{N} \left(\langle n, p_j \rangle - D\right)^2 \theta\left(||p_j - q_i||\right)$ where $q_i$ is the origin of the plane and $p_j$ is a point in neighborhood of $r_i$. The projection of $r_i$ onto the plane is used as the origin. In the second phase, a local bivariate polynomial approximation $g_i$ of surface, $S$, is computed in the local reference domain $(u_i, v_i, n_i)$ by optimizing $\sum_{j=1}^{N} \left(g_i\left(u_j, v_j\right) - f_j\right)^2 \theta\left(||p_j - r_i||\right)$, where $(u_j, v_j)$ is the projection of $p_j$ onto $H$ and $f_j = \langle n, p_j - r_i \rangle$ is the height of $p_j$ over $H$. The weighting function $\theta$ is a smooth, positive, monotone decreasing function $\theta$, such as the Gaussian given by $\theta\left(d\right) = e^{\left(\frac{-d^2}{h^2}\right)}$. In practice $k$ nearest neighbors of $r_i$ only are used for the computation.

There are many variations and extensions [6] to the original MLS surface approach [15]. Original approach sometimes exhibit undesirable behaviors for sharp features. In this paper,we have used original MLS surface approach [15]. Our method is independent of MLS surface approximation method used. Other variations and extensions which generated algebraic MLS surfaces be can be easily fit into this formulation. For example, the MLS surfaces can be generated by Robust MLS[9], which preserves the sharp features.

## 2.2. MLS to Algebraic Splats

The MLS surface fits the points in its local neighborhood defined by $h_i$. We generate an algebraic splat for point $r_i$ by bounding its MLS approximation, $g_i\left(u, v\right)$, to a disc around projection of $r_i$ on the plane $H_i$. The algebraic splat at point $r_i$ is thus defined in coordinate frame $(u, v, n)$ as

$$g_i(u, v) - n = 0, \qquad (1)$$

subject to

$$||u^2 + v^2|| \leq R_i^2, \qquad (2)$$

where $R_i = f(h_i)$. MLS surfaces are infinite surfaces in local reference domain but are correct only in the local neighborhood. Disk bounded MLS surface in local-reference domain is called an algebraic splat. Figure 3 shows the relationship between MLS surface and algebraic splat in the local domain. These splats are semi-implicit in nature and provides a descent compromise between explicit representations and implicits like MPUs and RBFs. MLS technique makes it easy to compute the intrinsic properties of the surface such as normals. A model consists of non-conforming, intersecting and overlapping, algebraic splats. Since algebraic splats are derived from MLS surface approximation, these splats obey all the properties of MLS surfaces. Algebraic splats of any order can be generated by controlling the degree of the polynomial used in the MLS approximation of the surface. We restrict our attention to the second, third and fourth orders as they are powerful enough to approximate local neighborhood.

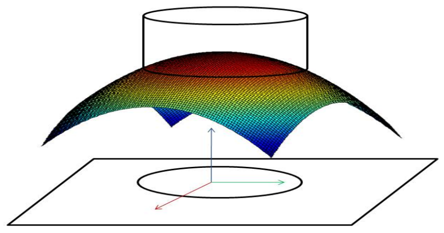

**Figure 3. Algebraic splats are the MLS surface restricted by a disc at the origin in local reference domain. The cylinder in local reference domain bounds the MLS surface.**

## 2.3. Algebraic Splat Representation

We want to approximate the shape of the model using as few higher order splats as possible. This has two aspects. First, the point set needs to be decimated or approximated using a fewer number of points. Second, a non-linear splat needs to be computed for each point in the reduced set with reference to the original point set. We can either generate a representation given an overall quality factor or a representation with user-specified number of algebraic splats.

### 2.3.1 Generation of a Single Algebraic Splat

Let $P$ be the point set. The process to generate the algebraic splat for a point $r_i \in P$ is given below.

1. Compute local feature size $h_i$ for each point $r_i \in P$.
2. Compute the MLS approximation at point $r_i$ using $P$.
3. Generate the algebraic splat $a_i$ as $(o_i, u_i, v_i, h_i, C_i)$.

An algebraic splat is represented by a tuple $(o_i, u_i, v_i, h_i, C_i)$, where $o_i$ is the origin of the local coordinate system. It is defined as the projection of $r_i$ on the local plane $H_i$ used in MLS approximation. $u_i$ and $v_i$ define the axes of the local ortho-normal coordinate system at $r_i$. The third direction can be calculated using $u_i$ and $v_i$. $C_i$ gives the coefficients of the bi-variate polynomial $g_i$. The number of coefficients depends on the order of the polynomial. We need 6, 10 and 15 coefficients for quadratic, cubic and quartic polynomials respectively. $h_i$ can be used to specify the cutoff radius for the polynomial in the local neighborhood and depends on the local feature size.

The local feature size $h_i$ plays a major role in approximating the model. A small value of $h_i$ makes the approximation more local and large values of $h_i$ smooths out the sharp features. We define $h_i$ as the average local distance between the points and calculated by:

1. Find $k$ nearest neighbors of $r_i$.
2. Project each to the plane $H_i$.
3. The $h_i$ is the mean distance in the $(u, v)$ space.

### 2.3.2 Hole-free Algebraic Splat Representation

An MLS surface can be defined at every point of the point set. It approximates the local shape at each point in its own neighborhood. Since a splat at a point is able to approximate shape in some neighborhood of radius $sh_i$, we can discard points within the distance $sh_i$ in parametric domain, where $s$ is a quality factor. At $s = 1$, the radius of splat is equal to the local feature size. If $s < 1$ then more points are included in the final hole-free representation of model. If $s > 1$ the more points gets discarded and quality decreases. The following algorithm gives a hole-free representation of $P$ , parametrized by a quality factor $s$.

1. Remove a point $p_i$ (possibly at random) from $P$, add it to the set $S$ of selected points.
2. Discard all the points from $P$ that are inside a circle of radius $sh_i$, the cut-off radius for algebraic splat $p_i$.
3. Continue doing this until all the points are exhausted from $P$.
4. Output algebraic splats for each point in $S$.

$S$ gives a set of points for a hole-free representation of the model using algebraic splats. The selection of points in step 1 can be improved to get a better or optimal hole-free representation at a higher computational effort. Number of points in the final set $S$ depends on the quality factor $s$. By changing it we can control LOD of the model. Our experiments show that the hole-free set $S$ has fewer than 10% of the points for most models at $s = 1$.

### 2.3.3 Fixed Size Algebraic Splat Generation

We describe how a model can be approximated using a user-specified number of algebraic splats. The decimation method should preserve the local structure. Lipman *et al* [17] proposed a parametrization-free projection for geometry reconstruction using the Locally optimal projection (LOP) operator. We use the LOP operator for decimation of the point set as it can adapt to the local structure well.

The LOP operator projects a set of points $X = \{x_i\} \subset \Re^3$ to an input set of points $P = \{p_j\} \subset \Re^3$. Points in $X$ move in each iteration of LOP so as to reduce the sum of weighted distances to $P$. After several iterations, the points in $X$ are regularly distributed into the input-point cloud. If we start with $M$ arbitrary points as the set $X$, an optimal approximation of $P$ using $M$ points can be obtained on convergence.

This gives an optimal representation of the original shape with $M$ points. LOP is a slow process, however the number of iterations needed for convergence will be less if the set is close to input-point cloud. We combine decimation and algebraic splat generation as follows.

1. Select $M$ random points from the input cloud.
2. Apply LOP operator between these points and the input point cloud $P$ till convergence, resulting in the decimated set $D$.
3. For each point in $D$, compute its MLS surface using the set $P \cup D$.
4. Output the algebraic splat for each $r_i \in D$. Calculate the local feature size using the decimated set $D$.

The decimated point set $D$ is regularly distributed into the shape of the model and gives an optimal approximation of model. This process is able to generate a algebraic splat representation for any user specified number of points.

## 2.4. Approximation Error

The error of approximation at point $p_i$ in the original point cloud is due the combined error of the near by overlapping splats. The error at a point $p_i$ is given by

$$\delta_i = \sum_{j \in J} \epsilon_{ij} \theta \left( ||o_j - p_i|| \right), \tag{3}$$

where $\epsilon_{ij}$ is the MLS approximation error at point $p_i$ due to splat $a_j$, $J$ a set of overlapping splats at point $p_i$ and $o_j$ the center of the splat $a_j$. The total approximation error, $\Phi$, of the algebraic splat representation is the sum of deviations of the point set from the surface defined by algebraic splats and can be given by

$$\Phi = \sum_{i=1}^{N} \delta_i \tag{4}$$

The MLS surface approximation error $\epsilon$ depends on the local feature size $h$, but is bounded [2, 3]. This ensures total weighted approximation error of algebraic splats is also bounded. If $f$ is the blended surface represented by non-conforming algebraic splats and $g$ is the surface represented by point set then upper bound on the error of approximation is defined as $||g - f|| \leq L \cdot h^{m+1}$, where $m$ is the degree of polynomial and $L$ is the constant that involves $(m+1)^{th}$ derivative of $f$, i.e., $M \in O\left(f^{m+1}\right)$. If $h$ is high, the error increases, the surface is smoothed.

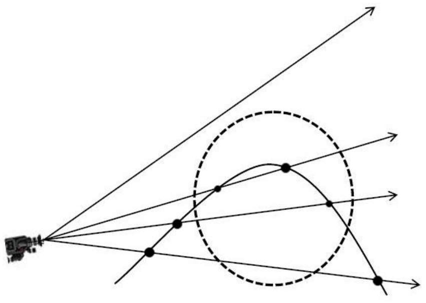

**Figure 4. Types of ray surface intersections for an algebraic splat. The smaller black dots are the desired points. Some of the bigger dots may be closet ray-surface intersection but falls outside algebraic splat radius.**

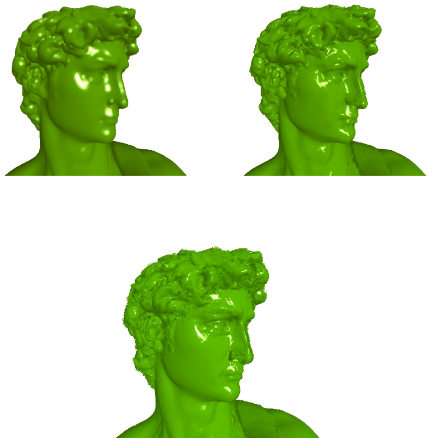

**Figure 5. Comparison of the head of David. Top Left: Linear splats; Top Right:Quadratic splats; Bottom:Cubic splats. The model used had 500K splats for linear, 36K for quadratic and 29K for cubic.**

## 3. Rendering Algebraic Splats

Algebraic surfaces can be rendered either as a 0-set surface using marching cubes or by ray tracing. Ray tracing method is a match for rendering on today's GPUs. This is because of the independence and parallelism offered by GPU. Each ray-surface intersection is independent of others and the equations can be solved in parallel.

We ray trace each splat individually and independently. Since the splat is a local approximation, we need to ray trace only a small region on the screen for each splat. The computation depends on the total area we are tracing rather than the window size. This is an output sensitive rendering algorithm. Bounding the rendering area can improve the speed.

### 3.1. Single Splat Rendering

A screen space bounding box is first computed for each splat. A splat is ray traced only inside its bounding box. This generates the fragments necessary for finding ray-surface intersection. The information required to ray-trace $i^{th}$ splat are sent from CPU through the pipeline. Since a splat has its own coordinate system, the camera center and the ray are transformed to local coordinate system for ray tracing. The equation of a ray is $R_f = c_i + t\vec{d_f}$, where $c_i$ is origin of rays for $i^{th}$ algebraic splat, $t$ is ray parameter and $\vec{d_f}$ is direction of ray for fragment $f$. Equation 3 is transformed to ray parameter space as follows:

$$F_f(t) = g_i(c_x + td_x, c_y + td_y) - (c_z + td_z) = 0, \quad (5)$$

where $F_f(t)$ is a polynomial in $t$. The smallest positive real root inside the disc gives the closest point of ray-algebraic splat intersection. $u_i, v_i$ of desired root should satisfy $\|u_i^2 + v_i^2\| \leq R_i^2$. Figure 4 shows various types of ray surface intersection possible.

If algebraic splats are at most fourth order, Equation 5 can be solved analytically. For quadratic splats, $F_f$, can be solved trivially. We use the simple and robust method proposed by Blinn to solve cubic splats [5] and the Ferrari method to solve quartic splats [11].

The normal of the surfaces is given by the surface gradient $\vec{\nabla}\{g_i(u,v) - n\}$ at the point of intersection. The normals are transformed to the camera coordinates and are used in lighting calculations.

### 3.2. Rendering Multiple Splats

The final representation consists of an algebraic splat for each point in $D$, each needing 16-25 floating point numbers depending on the degree. These numbers are packed into

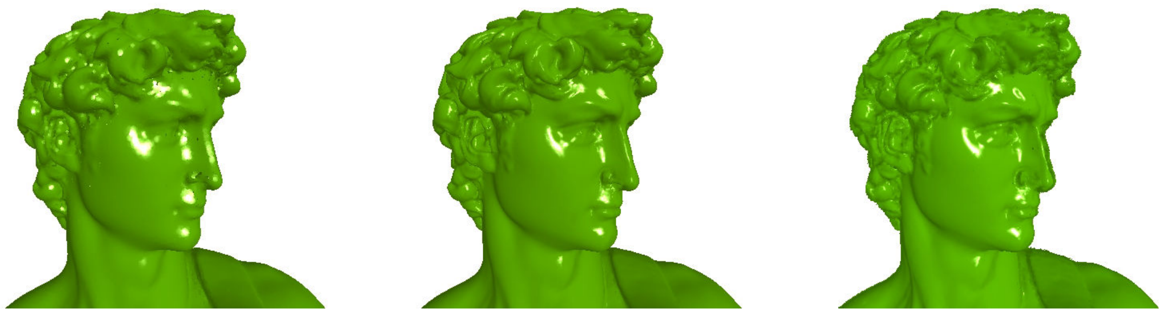

**Figure 6. Rendering of the head of David with linear, quadratic, and cubic algebraic splats. All the models used 400K splats.**

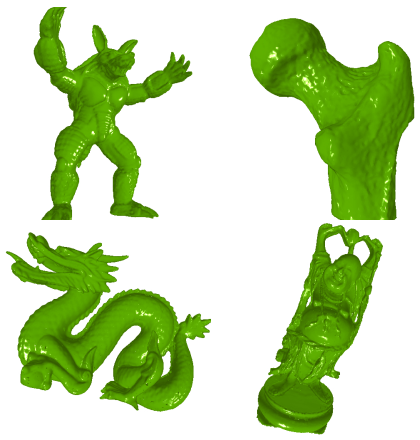

**Figure 7. Rendering of Armadillo with 12K, Ball Joint with 6K, Dragon with 42K and Happy Buddha with 24K algebraic splats using hole-free representation.**

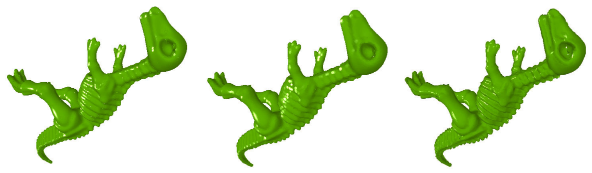

**Figure 8. Comparison of linear and quadratic for same quality. Left and Middle images are rendered with $s = 0.25$ and $s = 0.50$. Right image is rendered using quadratics with quality factor $s = 1$. These are rendered at 98, 141 and 262 fps respectively.**

a 2D texture and stored on the GPU memory. Each splat occupies a 2D section to take advantage of texture caching. Each splat can be accessed by vertex shader and rendered as described above. The rendering of these splats directly can create discontinuities at intersection. The surfaces need to be blended to have a visually smooth approximation. Each splat is ray-traced as described in 3.1. A 2-pass rendering algorithm is used to blend close by overlapping surfaces [26].

The first pass is the visibility pass. In this pass, the ray-surface intersection is calculated by using an appropriate root finding methods. The solution is discarded if the intersection falls outside the bounding circle of radius $R_i$. The depth for each intersection point is shifted away from camera by certain percentage of current depth, say $\delta$, and sent to the z-buffer. The closest $z$ value remains in z-buffer. This ensures that all surfaces within the $\delta$ distance will be blended in the second pass.

In second pass, blending is turned on. The colors and depths of the ray-surface intersections that differ by less than $\delta$ are blended along the ray. Early z-culling will ensure that the farther splats are quickly discarded. The output of the color buffer after second pass is $(\sum \alpha_i C_i, \sum \alpha_i)$, where $\alpha_i$ is the weight given to the point and $C_i = (r_i, g_i, b_i)$ is the color after shading. The weight $\alpha_i$ decreases exponentially with the distance from $o_i$ in the local reference domain. The net effect is similar to EWA splatting used with linear splats [26]. Final step normalizes the texture per-pixel and writes it to the color buffer. The color written is $(\sum \alpha_i C_i / \sum \alpha_i, 1.0)$. This ensures that the splat points within the $\delta$-distance of the closest splat are interpolated.

## 4 Results and Discussion

We tested our scheme on models with points ranging from 32K to 3.6M points. These are approximated using MLS surfaces of second, third and fourth order. As the order of algebraic splats increases, similar quality of rendering is achieved with less number of primitives. Figure 5 shows the quality of rendering with decreasing number of

| Model | Hole-free Representation | | | LOP Sampling [10%] | | | LOP Sampling [ 5%] | | | Linear at s=0.25 |
|---|---|---|---|---|---|---|---|---|---|---|
| | #splats (% of total) | FPS | | #splats | FPS | | #splats | FPS | | |
| | | Quadric | Cubic | | Quadric | Cubic | | Quadric | Cubic | FPS |
| David | 29769 (0.8) | 220 | 135 | 360K | 43 | 22 | 180K | 91 | 38 | 22 |
| Angel | 7270 (3.0) | 390 | 191 | 23K | 164 | 102 | 11K | 290 | 196 | 92 |
| Armadillo | 6567 (3.8) | 350 | 186 | 17K | 187 | 123 | 8K | 300 | 210 | 110 |
| Bone | 5324 (3.9) | 270 | 158 | 13K | 193 | 120 | 6K | 260 | 170 | 125 |
| Bunny | 1315 (3.6) | 401 | 190 | 3K | 220 | 130 | 1K | 315 | 160 | 261 |
| Dino | 2292 (4.0) | 393 | 206 | 5K | 298 | 185 | 2K | 490 | 300 | 194 |
| Dragon | 17,653 (4.0) | 198 | 119 | 43K | 158 | 92 | 1K | 215 | 141 | 40 |
| H.Buddha | 14,897 (2.7) | 240 | 159 | 54K | 167 | 75 | 27K | 200 | 129 | 45 |
| Horse | 2983 (6.1) | 460 | 220 | 4K | 350 | 245 | 2K | 560 | 311 | 260 |
| Igea | 4819 (3.6) | 221 | 148 | 13K | 170 | 110 | 6K | 240 | 310 | 135 |
| Lucy | 9123 (3.5) | 290 | 143 | 26K | 165 | 297 | 13K | 248 | 155 | 102 |
| Santa | 3922 (5.1) | 418 | 213 | 7K | 350 | 205 | 3K | 436 | 304 | 284 |
| Sphere | 112 (3.5) | 540 | 341 | 321 | 315 | 235 | 161 | 460 | 389 | 440 |

**Table 2. Comparison of rendering speed of models using algebraic splats for hole-free representation of model on nVidia's GTX280.**

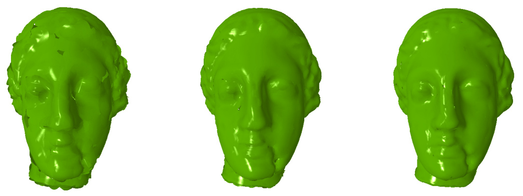

**Figure 9. Rendering of Igea model with fixed size sampling at 2, 5 and 10% of points from the complete model using algebraic splats.**

| Model | #Splats | Algebraic splats | | | SLIM[12] |
|---|---|---|---|---|---|
| | | G1 | G2 | G3 | G3 |
| Lucy | 130K | 92 | 49 | 11 | 7 |
| Armadillo | 24K | 168 | 94 | 39 | 23 |
| Dino | 6K | 308 | 180 | 87 | 44 |
| David | 932K | 18 | 10 | 4 | 3 |
| David | 126K | 108 | 69 | 14 | 7 |

**Table 1. Comparison of the FPS of algebraic splat rendering on G1 (nVidia GTX280), G2 (nVidia 8800 GTX), G3 (nVidia 7900 GTX) with SLIM surface rendering on G3 for equal number of primitives. The Hole-free representation with $s = 1$ is comparable in quality with SLIM and achieves much better rendering speeds (see Table 2).**

primitives as order of primitives increases.

For a given number of splats the shading quality increases as the order of splats increases. Figure 6 shows the rendering of linear, quadratic and cubic algebraic splats at 400K points. The third order splats brought out more detail with equal number of points on some models. The quartic splats were equivalent to the cubic ones in quality ( the $4^{th}$ order coefficients were close to 0) but were much slower to render.

The observe that shading quality increases as the order of splats increases.Figure 9 shows the rendering quality of LOP sampled models. Using very few (less than 5%) of points in LOP sampling we can see some holes on the forehead of Igea. This is because LOP distributes fewer points in the flat regions. As the number of points increases the points density in the flat regions also increases and hence the quality also increases. So, with algebraic splats we can get same or better quality than linear splats at higher fps. See figure (8).

Figure 7 shows comparison of Armadillo, Ball-joint, Dragon and Happy Buddha models using hole-free representation of point set with quality $s = 1$. Table 2 shows the rendering speed achieved with second and third order algebraic splats. The hole-free representation for linear splats at $s = 0.25$ equals hole-free representation at $s = 1$ for higher order splats for most of the models. All the readings are taken with 512x512 rendering window on nVidia GTX 280. The hole-free representation gives comparable quality with SLIM and requires less number of primitives. This may be due to inability of SLIM to approximate over large neighborhoods due to its view dependence nature. With algebraic splats, we can represent the point based models with less number of primitives with little or no drop in visual quality.

Table 1 gives comparison of rendering speeds with SLIM and algebraic splats. At a given number of splats, the algebraic splats performs better.

One of the limitation of this method is that sharp edges are smoothened out. This can be solved using other MLS formulations such as [9]. The hole-free representation may not give optimal representation or placement of splats. Getting optimal representation is computationally expensive.

## 5. Conclusions and Future Work

We introduced the algebraic splats for point based model contrary to existing linear splats. These splats were based on Moving Least Square surfaces. These piecewise splats are rendered directly by a two-pass GPU algorithm. Although hole-free representation requires fewer points it gives comparable visual quality with SLIM. We also discussed an algorithm based on LOP operator to generate the algebraic splats for a user specified number of primitives. Thus algebraic splats have potential to represent large point-based models using few primitives with little or no drop in visual quality. Large models can be rendered at interactive speeds on commodity GPUs.

We are implementing adaptive anti-aliasing for ray-tracing. The silhouette pixels can be detected using first order derivative of the function. These silhouette pixels will be sub-divided into a $nxn$ grid of sub-pixels. A grid of rays are intersected with the splats at those pixels. Anti-aliasing is achieved at the cost of rendering speed.

## 6. Acknowledgments

We gratefully acknowledge the partial financial support provided by the Naval Research Board of India. We also acknowledge Stanford University for making the data on David available for our research.

## References

[1] A. Adamson and M. Alexa. Ray tracing point set surfaces. In *SMI '03: Proceedings of the Shape Modeling International 2003*, page 272, 2003.

[2] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *Proceedings of the IEEE Conference on Visualization VIS*, pages 21–28, 2001.

[3] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9:3–15, 2003.

[4] N. Amenta and Y. J. Kil. Defining point-set surfaces. *ACM Trans. Graph.*, 23(3):264–270, 2004.

[5] J. F. Blinn. How to solve a cubic equation, part 5: Back to numerics. *IEEE Computer Graphics and Applications*, 27(3):78–89, 2007.

[6] Z.-Q. Cheng, Y.-Z. Wang, B. Li, K. Xu, G. Dang, and S.-Y. Jin. A survey of methods for moving least squares surfaces. In *Symposium on Point-Based Graphics*, 2008.

[7] E. de Groot and B. Wyvill. rayskip: faster ray tracing of implicit surface animations. *GRAPHITE*, 2005.

[8] S. Fleishman, D. Cohen-Or, M. Alexa, and C. T. Silva. Progressive point set surfaces. *ACM Transactions on Graphics*, 22(4):997–1011, 2003.

[9] S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM Trans. Graph.*, 24(3):544–552, 2005.

[10] G. Guennebaud and M. H. Gross. Algebraic point set surfaces. *ACM Trans. Graph*, 26(3):23, 2007.

[11] D. Herbison-Evans. Solving quartics and cubics for graphics. In *Graphics Gems V*, pages 3–15, 1995.

[12] T. Kanai, Y. Ohtake, H. Kawata, and K. Kase. Gpu-based rendering of sparse low-degree implicit surfaces. In *GRAPHITE*, pages 165–171, 2006.

[13] A. Knoll, Y. Hijazi, C. Hansen, I. Wald, and H. Hagen. Interactive ray tracing of arbitrary implicits with simd interval arithmetic. *Interactive Ray Tracing, 2007. RT '07*, pages 11–18, Sept. 2007.

[14] L. Kobbelt and M. Botsch. A survey of point-based techniques in computer graphics. *Computers and Graphics*, 28(6):801–814, 2004.

[15] D. Levin. Mesh independent surface interpolation. *Geometric Modelling for Scientific Visualization*, 2003.

[16] L. Linsen, K. Muller, and P. Rosenthal. Splat-based ray tracing of point clouds. In *Journal of WSCG*, volume 15, 2008.

[17] Y. Lipman, D. Cohen-Or, D. Levin, and H. Tal-Ezer. Parameterization-free projection for geometry reconstruction. *ACM Trans. Graph*, 26(3):22, 2007.

[18] C. Loop and J. Blinn. Real-time GPU rendering of piecewise algebraic surfaces. *ACM Trans. Graph.*, 2006.

[19] Y. Ohtake, A. Belyaev, and M. Alexa. Sparse low-degree implicits with applications to high quality rendering, feature extraction, and smoothing. In *EG Symposium on Geometry Processing*, 2005.

[20] Y. Ohtake, A. G. Belyaev, M. Alexa, G. Turk, and H.-P. Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, 2003.

[21] S. M. Ranta, J. M. Singh, and P. J. Narayanan. GPU objects. In *ICVGIP*, volume 4338 of *LNCS*, pages 352–363, 2006.

[22] C. Sigg, T. Weyrich, M. Botsch, and M. Gross. GPU-based ray-casting of quadratic surfaces. In *Symposium on Point-Based Graphics*, pages 59–65, 2006.

[23] C. Stoll, S. Gumhold, and H.-P. Seidel. Incremental raycasting of piecewise quadratic surfaces on the gpu. *IEEE Symposium on Interactive Ray Tracing*, pages 141–150, 2006.

[24] I. Wald and H.-P. Seidel. Interactive ray tracing of point-based models. In *Proc. of the Eurographics Symposium on Point-Based Graphics*, pages 9–16, 2005.

[25] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3D: An interactive system for point-based surface editing. In *ACM Trans. Graph.*, 2002.

[26] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 371–378, 2001.