High Quality Image Reconstruction of Point Models

Ricardo Marroquim
ricardo@lcg.ufrj.br

Antonio Oliveira
oliveira@lcg.ufrj.br

Paulo Roma Cavalcanti
roma@lcg.ufrj.br

Universidade Federal do Rio de Janeiro

Abstract

The parallel and discrete nature of many image-based techniques, such as interpolation methods, are highly suitable for GPU implementations. An important advantage is their bounded complexity by the screen resolution as opposed to the data size. Consequently, the direct surface reconstruction of points using image-based methods is an attractive solution to interactively render large data-sets. In this paper, we propose a novel approach that raises the robustness and quality standards of previous related works. The algorithm achieves quality equivalent to Surface Splatting techniques, while maintaining high performance rates compatible with previous image-based methods. A hierarchical pixel structure is employed to render the projected samples efficiently with minimum artifacts. One of the main advantages of the method is the independency from the number of samples during the surface reconstruction. Furthermore, no extra object space data structure is needed, making the approach also memory efficient.

1. Introduction

Point-based graphics is an evolving area and has gained much interest during this last decade. One of the main motivations is the increase in resolution and precision of registration devices that generate extremely large point data, e.g., 3D scanners. The major advantage of working directly with this representation is the lack of connectivity information, as opposed to triangle meshes. Points, the most basic primitive, are appealing for their simple nature, since they contain all relevant data in themselves.

Notwithstanding, interactively rendering and manipulating data in the order of millions of points is not a trivial task, even more, because the graphics hardware are specialized triangle renderers. By contrast, due to their lack of connectivity, point graphics algorithms are highly adequate for GPU implementations, being able to overcome this drawback. In addition, image-based approaches also map well to the GPU, considering that they are inherently discrete and pixel bound methods. Therefore, they are able to fully profit from the graphics processor parallelization power without resorting to complex setups and data structures.

Figure 1. The Asian Dragon model with approximately 3.6 M points rendered at 20fps with our algorithm.

Surface Splatting, a point-based rendering method, has a consolidated resampling theory background that allows it to generate quality images. On the other hand, point renderers based on image-processing techniques have attained the highest performances to this date, achieving rates of over a hundred million points per second.

In this paper, we present a new fully image-based interactive point rendering method with performance competitive with previous approaches and, at the same time, able to achieve a quality similar to *Surface Splatting*. An example of a model rendered with the proposed algorithm is depicted in Figure 1. The main points and goals proposed by this work are outlined below:

- support to large point models and minimize the cost associated with the number of samples, by using a very simple projection routine;
- use an image-based hierarchical structure for efficient and constant time surface reconstruction;

- keep the system independent of any auxiliary data structure or heavy pre-processing;
- reduce aliasing and flickering issues significantly compared to previous image-based renderers.

Briefly, our main contribution is a novel hierarchical search scheme that allows a fast coverture evaluation of the projected splats. Apart from the projection phase, the reconstruction cost is bounded only by the screen resolution, and consequently, is independent of the number of splats. The complete rendering algorithm has complexity $O(s)+O(n \log n)$, for projecting and reconstructing respectively, where s is the number of splats and n is the number of pixels.

In the next section related works are presented followed by the algorithm description in Section 3. Some results obtained with our prototype are presented in Section 4. Finally, conclusions and directions for future work are layed out in Section 5.

2. Related Work

Levoy and Whitted [9] proposed the use of points to reconstruct surfaces more than two decades ago. Nonetheless, only in the last few years has this become an appealing area. To obtain an efficient algorithm we have advocated the use of image-processing techniques. This two topics are reviewed next.

2.1. Point-Based Rendering

Surface Splatting [21] is today the most popular technique for direct rendering of point models. It relies on the projection of circular or elliptical kernels centered at each sample with radii respecting the local sampling space. The surface is reconstructed using Gaussian filters to blend the splats in screen space. Elliptical weighted average (EWA) splats[22], together with perspective correct projection [23], provide high quality results and hole-free surface reconstruction.

This rendering method was further extended to make use of graphics hardware resources [15, 3, 2, 8] achieving high frame rates, while maintaining the original quality. One of the weakness of most GPU Surface Splatting algorithms is the need for a two-pass rendering approach, where the first pass treats the visibility issue, while the second reconstructs the model by weighting the splats. Recently, a single-pass method has also been proposed by Zhang & Pajarola [19, 20]. Nonetheless, these and other similar methods tackle the surface reconstruction problem by rasterizing screen space ellipses (or Gaussian filters), and have complexity bound by the number of samples as well as their projected size.

2.2. Image-Based Techniques

With the advent of graphics hardware programming, well known image-based techniques have been rethought to solve problems many times faster. An example is the pyramid algorithm [12, 4, 6] (also known as *pull-push* or *gather-scatter* algorithm) employed to solve problems, such as the interpolation of scattered data, for example. This algorithm was recently adapted to the GPU by Strengert et al. [18] achieving interpolation times of a few milliseconds for high-resolution settings, i.e., 1024^2 and 2048^2. Another example is the Jump Flooding Algorithm [16, 17], where applications such as 2D Voronoi Diagrams and the Distance Transform are rapidly computed.

Due to their logarithmic nature, these techniques have in common their image space search complexity. Using a *gather* strategy, where each pixel "looks around" to determine its value, efficient GPU implementations are accomplished. Even though these are approximative methods, it is possible to render high quality images with little or no deviation from the exact approach.

Image-based methods appeared in the point-based rendering literature, as early as a decade ago, when Grossman and Dally [7] proposed a method to reconstruct the surface in image-space. Pfister et al. [14] introduced the concept of *Surfels*, or *Surface Elements*, and employed a pyramid algorithm to fill holes on the reconstructed surface. Howbeit, both techniques require the data to be resampled within uniform structures.

A more recent image-based surface reconstruction method for points was proposed by Marroquim et al. [10, 11]. The Pyramid Point Renderer (PPR) algorithm is based on image processing techniques, where the projected samples are interpolated using a *pull-push* approach. By integrating elliptical kernels with the pyramid hierarchy, the algorithm's complexity is lowered from $O(s \times \bar{a} + n)$ to $O(s + n)$, where $\bar{a}$ is the average splat size in pixels. All the same, the output quality still does not match the *Surface Splatting* method, and aliasing artifacts are evident, specially near the silhouettes. This is mainly caused by the averaging scheme that creates ellipses in some pyramid levels, and are not fully propagated during the push phase. Nevertheless, they reported the highest performance rates to this date, up to 130M splats per second, i.e., nearly 5M points in real-time (around 30fps).

3. Surface Reconstruction in Constant Time

Disregarding the projection phase, the reconstruction of surfaces in screen space in constant time requires every pixel to be treated indifferently, regardless of the number and size of projected samples. One possible way to accomplish this task using recent GPU techniques is to implement

a *gather* strategy, where each pixel collects the necessary information to compute its final value; as opposed to a traditional CPU splatting algorithm, where each splat *scatters* its contribution to the pixels it covers.

A naive implementation of a *gather* scheme is to project each sample's center onto a single pixel, and store in it the information necessary to render the elliptical splat, i.e. normal, radius, color, etc... Following the projection phase, each pixel searches its neighbors for splat centers and evaluates by which ones it is covered. The valid contributions are weighted averaged to determine the pixel's final value, by using, for example, elliptical distances. However, to guarantee that the pixel finds every possible splat, it must search the whole screen space. This leads to an $O(n^2)$ complexity making the algorithm inviable for interactive rendering systems.

The algorithm presented here also employs the *gather* strategy, but makes use of a multi-resolution search structure to reduce the complexity to $O(n \log n)$. This allows each pixel to gather the splats around it, without having to look at $n-1$ neighbors. In fact, we further demonstrate that for an 1024^2 screen resolution, the surface can be reconstructed with 100 fetches per pixels. This is equivalent to only fully rendering ellipses that fit in a 10×10 pixel region with the naive approach.

In a nutshell, the idea of the method is to, in a first phase, project each splat in a specific level, where it is fully covered by a kernel of size $k \times k$ pixels, as described in Section 3.1. During a second phase, as described in Section 3.2, each screen sized pixel searches for projections in all coarsest levels with a template kernel of same size. For each projection center found, if the elliptical extent covers the pixel, it is blended with the currently stored value. In a final $O(n)$ pass deferred shading is employed as described in Section 3.3.

3.1. Multi-Resolution Structure

The multi-resolution structure is created in a very similar way to the pull phase of the Pyramid Point Renderer (PPR) algorithm. Nevertheless, since there are crucial differences, we will describe the process entirely here.

Each sample is projected as a pixel-sized point onto the screen plane applying back-face culling as an early discard test. To comply with splats projecting onto the same pixel, a simple depth test is used to retain only the frontmost. The splat projection is stored in a single pixel with the following data:

- projected normal;
- projected radius;
- screen coordinates of projection;
- depth component.

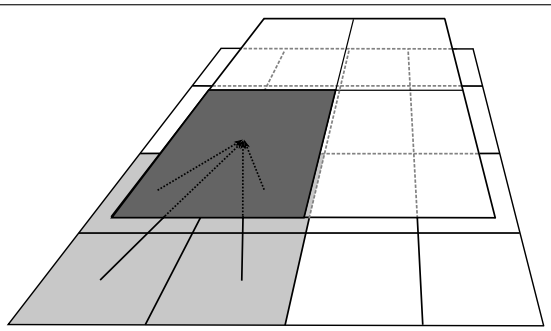

Figure 2. Each pixel in a coarsest resolution level of the pyramid structure is computed by gathering four pixels from the level below, until a one pixel level is reached.

Since the projection buffer has typically four elements per pixel, two buffers are used to store this information, without having to resort to compression techniques. In reality, during our implementation, three buffers were used to grant a fair comparison with the PPR algorithm, where the third buffer stores extra shading values such as color.

The samples are circular kernels in object space, projected as ellipses onto screen-space. As the radii size is usually very small, the splats can be projected orthogonally, instead of perspectively, without generating significant errors and leaving gaps on the surface. The major axis size is twice the pixel's radius and is perpendicular to the normal's x and y coordinates, while the minor axis is scaled by the normal's z coordinate. In this manner, ellipses near the silhouette are narrow, while samples with orthogonal normals to the screen plane are projected as circles.

The structure is based on a pyramid algorithm for image reconstruction, where each level of the pyramid halves the dimensions of the immediately higher resolution level. Each coarser pixel is computed by averaging the four pixels it covers in the level below (see Figure 2 for an illustration of this *gather* strategy).

A projected splat is propagated up the pyramid until it reaches the level that fully covers it with a $k \times k$ template. This guarantees that all pixels it extends over in level 0 (highest resolution) are able to find the projection center. Furthermore, multiple fetches of the same center in different levels have to be avoided, otherwise artifacts appear from blending the same ellipse several times.

The minimum level to fit a splat depends on the size of the search template kernel, and can be easily computed using the following equation:

$$l = \lceil log_2 \frac{D}{t_s} \rceil, \tag{1}$$

where D and t_s are respectively the projected radius and

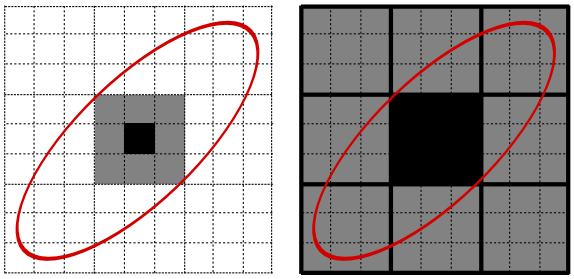

Figure 3. The projection level must be large enough for the template to cover the full splat projection. The level on the left is not large enough because only the gray pixels are able to find the projection center stored at the black pixel. The level on the right has the minimum coverage for all pixels inside the ellipse to find the black pixel.

template sizes in pixels. Without loss of generality, the algorithm will be describe and illustrated with a template size $t_s = 3$. Nevertheless, we also show results for a 5×5 kernel. The correct resolution level to store the projection is depicted in Figure 3.

Even though it is possible to project the splat directly onto the correct level without the need of propagating it from level 0 up, we have opted for this solution for a few reasons. First, multiple splats reaching a single coarser pixel must be averaged in one center before continuing. Moreover, the hierarchy construction phase cost is very low, compared to the reconstruction phase. Therefore, the decrease in performance is negligible. Not to mention that, since all levels are stored in a single texture, to actually project a pixel onto another place, corresponding to a lower resolution level, is not trivial; the projected position of the sample needs to be shifted before it passes through the vertex processor.

This simpler propagation phase, compared to the Pyramid Point Renderer algorithm, is sufficient to significantly reduce flickering and aliasing artifacts, because each ellipse is only available in one level. By leaving ellipses in multiple levels, only some pixels might access the original center, while others might access an already averaged center. This inconsistency creates discontinuities, because some ellipses are only partially rasterized.

In short, a projected splat is moved upwards the pyramid hierarchy until it finds its correct level. It is recorded as a valid projection for that level and not further propagated. When two, three or four splats meet at a single coarser pixel they are merged. This is a straightforward average of the elliptical information, such as, center, normal, radius etc... Note that, the larger the template kernel, the less ellipses merge, since they propagate less until reaching the correct level.

During this phase, a pre-depth test is also employed. From the candidates to be averaged, the frontmost is selected, and the others are only considered if they are close enough to be regarded as in the same surface. An ellipse is averaged when its depth is less than the frontmost depth minus the sum of their radii. Otherwise, the one in the back is discarded from the next phase.

3.2. Reconstructing and Blending Splats

The surface reconstruction is more computationally intense than the previous step. For each level 0 pixel, a search is carried out in all pyramid levels using the template search kernel. In other words, each pixel looks at its neighbors inside the template in each resolution.

As previously mentioned, the splat center stores the projection coordinates in the highest resolution level. Together with the ellipse's information, an inside/outside test can be easily carried out to determine if the current pixel is covered by the ellipse. If this is the case, a ternary depth test is employed using the same criterion as in the previous step. If the ellipse is in a surface in front of the current pixel, it overwrites it. If it is in a back surface, the pixel remains as it is. Otherwise, they are blended by a weighted sum of the attributes. The weight used is one minus the elliptical distance d from the pixel to the ellipse's center:

$$w = 1.0 - d. \tag{2}$$

We found that this weight creates a smoother interpolation than using the exponential weight proposed in the PPR algorithm. In our implementation, the exponential weight leaves clear silhouette traces of the elliptical splats, resulting in an artificial looking reconstruction.

The larger the template search kernel's size, the less levels need to be searched. For the simplest 3×3 kernel case, the coarsest level with a single pixel can be left out, since the template can cover the entire screen space in the level below. The propagation of the 3^2 kernel from a 16^2 screen resolution is illustrated in Figure 4.

On the other hand, the larger the kernel, the more texture fetches are needed. Even so, larger templates tend to merge less splats during the hierarchy construction phase, generating smoother reconstructions.

For a typical 1024^2 resolution, eleven levels are created, where level 0 contains the full screen size resolution and level 10 contains only one pixel. As the last level is not necessary, each pixel searches $10 \times 3^2 = 90$ positions. Each level search is carried out in a different shader pass, and for each, an extra fetch is needed to retrieve the current pixel's value, amounting to a total of 100 fetches per pixel with this configuration.

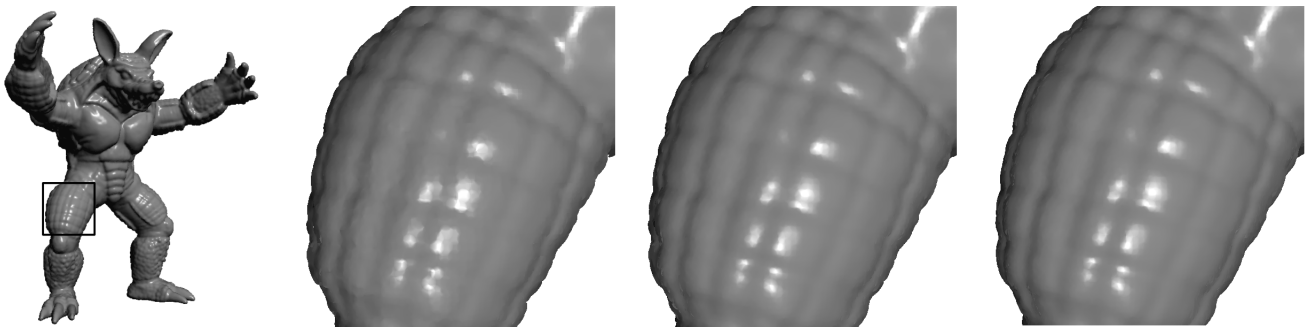

Figure 5. Same scene rendered with three different kernel sizes. From left to right, the kernel sizes are 3^2, 5^2, and 7^2. Note how there is little qualitative gain between the 5^2 and 7^2 kernels, even though the performance drops around 38%.

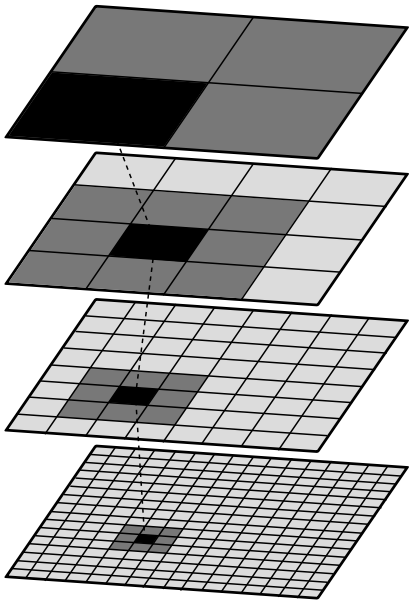

Figure 4. The search coverage for the projected pixel (highest resolution level) with a 3^2 search template. Painted black is the same pixel represented in different resolutions, while the gray pixels are its search template for each level.

The template size is a compromise between quality and performance. In fact, the 3^2 kernel achieves frame rates very close to the PPR algorithm, while a 5^2 kernel has similar quality as Surface Splatting. Nevertheless, raising the size from 3^2 to 5^2 pixels, results in an almost 40% decrease in performance. Larger templates did not amount in any significant qualitative gain. In Figure 5, a detail of the Armadillo's leg is shown for three different kernel sizes.

3.3. Shading

After reconstructing the model, each pixel stores either an interpolated normal if it belongs to a surface, or is empty if it is a background pixel. The final buffer is in fact a normal map from which deferred shading can be easily computed, either with a constant color or with extra shading attributes stored and interpolated per pixel. However, each extra attribute demands extra buffers to be interpolated during the reconstruction, hence, decreasing the performance. During our experiments we used three buffers: two to store the basic splat data, and one extra to store color information. The normal map and shaded surface for a view of the Dragon model are depicted in Figure 6.

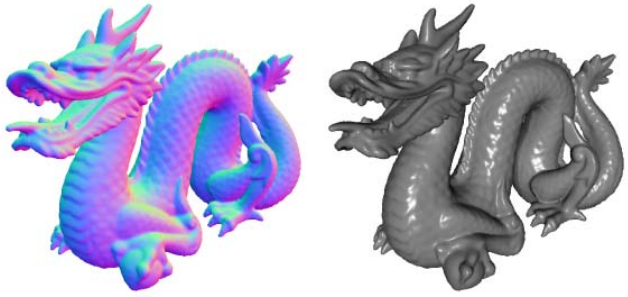

Figure 6. The normal map (left) is the result of the interpolation algorithm. On the right, the final model after the deferred shading pass.

		template size		
model	# points	3×3	5×5	PPR
Armadillo	173 K	43	24	50
Happy Buddha	544 K	40	23	49
Neptune	2.004 K	28.5	19	32.5
Asian Dragon	3.610 K	20.5	15	21.5
Statuette	5.000 K	15.5	12.5	16.5

Table 1. Rendering performance in FPS for a 768^2 screen resolution.

4. Results

The prototype was tested on a GeForce 8800 GTS with 640 MB memory and an Intel Core 2 Duo 6600 CPU (2.4 GHz) with 2 GB RAM. All models had their normals and splat radius pre-computed. The whole system is implemented using OpenGL and GLSL. We make use of vertex buffers to transfer the data to the GPU, and all texture and buffers have 16-bits floating-point elements.

In Table 1, some rendering times are layed out for comparison with the Pyramid Point Render algorithm. We provide timings for the 3^2 and 5^2 kernels.

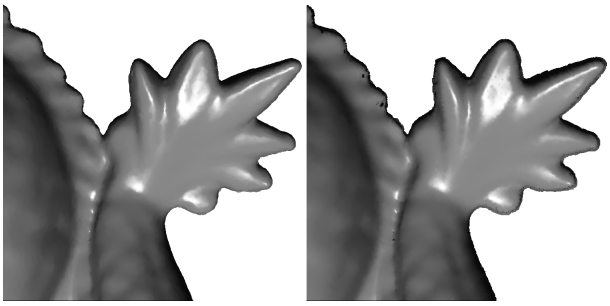

Figure 7. The Dragon tail rendered with a 5×5 kernel (left) and with the PPR algorithm (right). Note how the template strategy renders smoother silhouettes with less artifacts.

The difference from the template kernels and the compared PPR algorithm falls, as the number of points increases. This is expected, since for large models the time is dominated by the projection phase, while the reconstruction is carried out in constant time. Nonetheless, for the smaller models the difference using the 3^2 kernel is only of approximately 15%, and can drop to 5% for the larger models. The frame rates are still very high compared to splatting algorithms, taking in account that the PPR work reported an increase of up to 40%. In Figure 7, a qualitative comparison between rendering with our algorithm using a 5^2 kernel and the PPR algorithm can be observed. Other models rendered using our system are illustrated in Figures 8 and 9.

Figure 9. A magnified view of Neptune's head.

5. Conclusions & Future Work

We presented an image-based reconstruction method for point models that significantly improves the quality of previous related methods, without dropping the performance considerably. By rasterizing the elliptical projections in a similar way to *Surface Splatting*, the silhouettes are well preserved. On the other hand, by using an image-space hierarchical structure, the reconstruction remains independent of the number of splats, achieving high frame rates even for large models.

The algorithm still depends on the kernel size parameter, as a trade-off between quality and performance. Nevertheless, experiments show that a 5×5 template kernel is sufficient to render high quality images. Even so, the quality difference between the 3^2 and 5^2 templates are more noticeable under magnifications.

Figure 8. From left to right, Sitting Buddha, Neptune and the Statuette model rendered with our method.

Figure 10. A magnified view of the bottom sculptures of the Statuette model.

The inability to project more than one center to a single pixel remains a limitation of this algorithm, since this may cause aliasing artifacts for very dense models. However, we have not visually observed this effect during our experiments.

Another drawback of the algorithm, is the averaging scheme, specially for small kernel size. When ellipses are averaged they are replaced by a single one, which doesn't necessary cover their whole area. The problem with creating a larger ellipse to extend over the averaged ones, is that it will continue to propagate higher on the pyramid, and eventually will merge again, creating a cycle until it reaches the top level. Nevertheless, this solution still produces less artifacts due to the fact that each ellipse only appears once in the hierarchy. A few artifacts and flickering during animation are still perceived. However, they are expressively less than the PPR algorithm.

In all, even though image-space techniques still have some drawbacks, specially due to resampling problems, it continues to prove its value by achieving high performance rates while generating quality images. We are confident that the performance can still be increased by optimizing the hierarchical search, without compromising the quality. To reduce the reconstruction complexity from $O(n \log n)$ to $O(n)$, an independent reconstruction is necessary for each pyramid level. In other words, each level must reconstruct the splats in its own resolution, without recurring to level 0. Even though, at this moment, we are not aware of a way to achieve this using the templates, while preserving the quality, we are confident that the complexity constant can be significantly reduced. Because it is directly associated with the kernel size, and if each level is able to reconstruct their own ellipses, a search with template size 1 might be possible.

As future work, we propose to incorporate a level-of-detail structure, for example, the Sequential Point Trees pro-

posed by Dachsbacher et al. [5] or the Point Octree structure proposed by Pajarola [13]. Considering that the reconstruction algorithm is independent of the projection, these could be easily integrated allowing even higher frame rates to be achieved by reducing the projection cost. Additionally, the kernel size can also be chosen dynamically while traversing the LOD structure, as well as the maximum propagation level of the pyramid.

Another possible improvement is the combination with other shading techniques, such as shadows and more realistic lighting. The inclusion of transparency is also possible. However, since no ordering is guaranteed during the reconstruction, a method comparable to the k-buffer[1] must be employed. Likewise, the addition of anti-aliasing techniques to improve even more the silhouettes and render better animations is important. At last, it would be a valuable and natural extension to render splats with perspective correct projection as well as clipped splats for sharp features.

6. Acknowledgments

We would like to acknowledge the grant of the first author provided by Brazilian agency CNPq (National Counsel of Technological and Scientific Development).

References

[1] L. Bavoil, S. P. Callahan, A. Lefohn, J. L. D. Comba, and C. T. Silva. Multi-fragment effects on the gpu using the k-buffer. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 97–104, New York, NY, USA, 2007. ACM.

[2] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. High-quality surface splatting on today's GPUs. In *Proceedings of the Eurographics/IEEE VGTC Symposium on Point-Based Graphics '05*, pages 17–24, 2005.

[3] M. Botsch and L. Kobbelt. High-quality point-based rendering on modern GPUs. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, pages 335–343, 2003.

[4] P. J. Burt. Moment images, polynomial fit filters, and the problem of surface interpolation. In *Proceedings of Computer Vision and Pattern Recognition*, pages 144–152, 1988.

[5] C. Dachsbacher, C. Vogelgsang, and M. Stamminger. Sequential point trees. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 657–662, 2003.

[6] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 43–54, 1996.

[7] J. P. Grossman and W. J. Dally. Point sample rendering. In *9th Eurographics Workshop on Rendering '98*, pages 181–192, 1998.

[8] G. Guennebaud, L. Barthe, and M. Paulin. Splat/mesh blending, perspective rasterization and transparency for point-based rendering. In *Proceedings of the IEEE/Eurographics/ACM Symposium on Point-Based Graphics '06*, pages 49–58, 2006.

[9] M. Levoy and T. Whitted. The use of points as a display primitive. Technical Report 85-022, University of North Carolina at Chapel Hill, 1985.

[10] R. Marroquim, M. Kraus, and P. R. Cavalcanti. Efficient point-based rendering using image reconstruction. In *PBG '07: Proceedings of the Eurographics Symposium on Point-Based Graphics*, pages 101–108, September 2007.

[11] R. Marroquim, M. Kraus, and P. R. Cavalcanti. Efficient Image Reconstruction for Point-Based and Line-Based Rendering. *Computer Graphics*, 32:189–203, 2008.

[12] J. Ogden, E. Adelson, J. Bergen, and P. Burt. Pyramid based computer graphics. *RCA Engineer*, 30:4–15, 1985.

[13] R. Pajarola. Efficient level-of-details for point based rendering. In *Computer Graphics and Imaging*, pages 141–146, 2003.

[14] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 335–342, 2000.

[15] L. Ren, H. Pfister, and M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum (Eurographics 2002)*, 21(3):461–470, 2002.

[16] G. Rong and T.-S. Tan. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 109–116, New York, NY, USA, 2006. ACM.

[17] G. Rong and T.-S. Tan. Variants of jump flooding algorithm for computing discrete voronoi diagrams. In *ISVD '07: Proceedings of the 4th International Symposium on Voronoi Diagrams in Science and Engineering*, pages 176–181, Washington, DC, USA, 2007. IEEE Computer Society.

[18] M. Strengert, M. Kraus, and T. Ertl. Pyramid methods in GPU-based image processing. In *Proceedings of Vision, Modeling, and Visualization 2006*, pages 169–176, 2006.

[19] Y. Zhang and R. Pajarola. Single-pass point rendering and transparent shading. In *Proceedings of the Eurographics/IEEE VGTC Symposium on Point-Based Graphics '06*, pages 37–48, 2006.

[20] Y. Zhang and R. Pajarola. Deferred blending: Image composition for single-pass point rendering. *Computer & Graphics*, 31(2):175–189, 2007.

[21] 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.

[22] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002.

[23] M. Zwicker, J. Räsänen, M. Botsch, C. Dachsbacher, and M. Pauly. Perspective accurate splatting. In *GI '04: Proceedings of the 2004 Conference on Graphics Interface*, pages 247–254, 2004.