Eurographics Symposium on Point-Based Graphics (2007)
M. Botsch, R. Pajarola (Editors)

Efficient Point-Based Rendering Using Image Reconstruction

Ricardo Marroquim[1], Martin Kraus[2], and Paulo Roma Cavalcanti[1]

[1]Universidade Federal do Rio de Janeiro, Brazil
[2]Technische Universität München, Germany

Abstract

Image-space reconstruction of continuous surfaces from scattered one-pixel projections of points is known to potentially offer an advantageous time complexity compared to surface splatting techniques. We propose a new algorithm for hardware-accelerated image-space reconstruction using pull-push interpolation and present an efficient GPU implementation. Compared to published image-space reconstruction approaches employing the pull-push interpolation, our method offers a significantly improved image quality because of the integration of elliptic box-filters and support for deferred Phong shading. For large point-based models, our GPU implementation is capable of rendering more than 50 M points per second—including image-space reconstruction and deferred shading.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Bitmap and framebuffer operations – Display algorithms – Viewing algorithms; I.3.7 [Computer Graphics]: Hidden line/surface removal

1. Introduction

Point-based graphics have evolved from a specialized tool for particle systems [CHP*79] to one of the most powerful concepts in computer graphics. In numerous applications, points have been proven to be a suitable primitive for model acquisition, processing, animation, and rendering [GP07]. In this work, we focus on point-based surface rendering on GPUs (graphics processing units).

The performance of programmable GPUs has been successfully employed for real-time, point-based surface rendering by multipass EWA (elliptical weighted average) splatting techniques [KB04, SP04, SPL04]. Most of these techniques consist of a visibility pass to compute a depth map, an attribute pass to blend colors, normals, and other attributes, a normalization pass to finalize the interpolation of attributes, and a deferred shading pass. While this approach achieves a high rendering performance even for high-quality images [BHZK05,GBP06], the efficiency of this GPU-based method is not optimal since the two object-order passes, namely the visibility pass and the attribute pass, have to process all displayed points, i.e., the time complexity of both passes is $O(n \times \bar{a})$ for n splats, each covering on average $\bar{a}$ pixels. On the other hand, the worst-case time complexity of the normalization pass and also of the shading pass (assuming local shading operations) is $O(m)$ for a viewport consisting of m pixels, i.e., it is independent of the number of projected splats.

Many alternative point-based rendering techniques have been published in the literature on point-rendering techniques. One of the basic requirements for any surface rendering method is the reconstruction of continuous surfaces without holes. While these holes are avoided by a sufficiently high initial sampling rate [LW85], this solution is not always practical or possible. An alternative approach is the filling of holes by an image-space reconstruction of surfaces [For79, GD98, PZvBG00], i.e., an interpolation of scattered pixel data. The algorithm by Forrest produces only one-pixel holes, which "are filled most effectively by a simple averaging technique" [For79]. In contrast to this approach, the method proposed by Grossman and Dally [GD98] can handle large gaps between one-pixel projections of points. To this end, they employ a variant of the pull-push interpolation suggested by Gortler et al. [GGSC96] to reconstruct a continuous surface. Although Pfister et al. [PZvBG00] render larger splats instead of one-pixel points, they also employ the pull-push algorithm to fill holes between these splats. However, this image-space hole filling method resulted in an inferior image quality while it required additional implementation efforts. Furthermore, the method did not map well to hardware-accelerated implementations. Thus, subsequent research was focused on improving splat shapes to avoid holes between splats while minimizing rasterization overdraw, in particular for EWA splats [ZPvBG01, RPZ02, GP03, ZRB*04, BHZK05, GBP06].

Nonetheless, it is worthwhile to consider the time complexity of the image-space reconstruction by means of the pull-push algorithm. Since this algorithm is of complexity $O(m)$ for a viewport of m pixels [GGSC96, Bur88] and the projection of n points to single pixels is of complexity $O(n)$, the complexity of the whole algorithm is $O(n+m)$ [GD98]. On the other hand, the complexity of most splatting algorithms is $O(n \times \bar{a} + m)$ with the average splat size $\bar{a}$. Moreover, the visibility pass introduced by Pfister et al. [PZvBG00] results in two object-order passes in most GPU-based surface splatting techniques [RPZ02, GP03], and therefore leads to a worse constant for the dependency on the number of points n in these GPU implementations.

Strengert et al. [SKE06] have recently presented an efficient GPU implementation of the pull-push algorithm; thus, an efficient implementation of the image-space reconstruction for point-based surface rendering is also feasible. Unfortunately, the algorithm by Grossman and Dally [GD98] is not well suited for GPUs; in particular because their method for finding gaps projects points not only to a frame buffer of the target resolution but also to a coarser level of a hierarchy of depth buffers. Therefore, at least one additional object-order pass is required in a GPU implementation. Furthermore, the per-point selection of a level of a render-target hierarchy is not well supported by GPUs.

One contribution of our work is therefore the design and GPU implementation of an efficient point-based rendering technique using image-space reconstruction by means of a variant of the pull-push interpolation. The second contribution of this work is the integration of elliptical box-filters—i.e., 2D projections of 3D discs—in the pull-push interpolation, which considerably improves the rendering of silhouettes. In combination with deferred Phong shading [BSK04], this results in a significant improvement of image quality compared to published image-space reconstruction methods [GD98, PZvBG00]. As compared to surface splatting techniques that provide a similar image quality, our algorithm features a higher rendering performance due to the improved complexity and the single object-order pass.

The next section summarizes related work, while Section 3 presents our algorithm and its implementation on GPUs. Results obtained with a prototypical implementation of this algorithm are discussed in Section 4. In Section 5, potential improvements and extensions of our method as well as directions for future research are suggested.

2. Related Work

This section mentions only some selected publications related to point-rendering techniques. More comprehensive surveys of point-based graphics have been published by Kobbelt and Botsch [KB04] and Sainz et al. [SP04, SPL04]. The forthcoming book on point-based graphics edited by Gross and Pfister [GP07] includes also the latest research results in point-based graphics.

As published by Csuri et al. [CHP*79], points have been used to model and render "soft" phenomena such as smoke since the 1970s. At the same time, points have also been employed to render surfaces as published by Forrest [For79]. Catmull and Clark [CC78] have presented a subdivision scheme for B-spline patches, which is suitable for rendering surfaces by subdividing them to sub-pixel points. This concept was implemented in the form of "micropolygons" within the Reyes architecture published by Cook et al. [CCC87]. The first in-depth discussion of points as graphics primitive was provided by Levoy and Whitted [LW85].

Surface interpolation by a pyramid algorithm was suggested by Burt [Bur88] and employed by Gortler et al. [GGSC96], who named their variant "pull-push algorithm." This method was adapted by Grossman and Dally [GD98] for an image-space reconstruction of undersampled point-based surfaces, i.e., in the case of large gaps between projected points. While Grossman and Dally separated the finding and filling of gaps, Popescu et al. [PEL*00] described a hardware-architecture that implements a more elaborate separation of visibility and reconstruction for image-based rendering. For point-based rendering, Pfister et al. [PZvBG00] employed the pull-push algorithm to fill holes between splats instead of one-pixel projections of points.

Subsequent research avoided these holes by splatting as suggested by Rusinkiewicz and Levoy [RL00]. Of particular interest were EWA splats as published by Zwicker et al. [ZPvBG01]. Hardware-accelerated EWA splatting was published by Ren et al. [RPZ02] and Guennebaud and Paulin [GP03]. Improvements of EWA splatting include perspective accurate splatting by Zwicker et al. [ZRB*04], deferred splatting by Guennebaud et al. [GBP04], and deferred Phong splatting by Botsch et al. [BSK04]. Further improvements of this GPU-based approach were presented, for example, by Botsch et al. [BHZK05] and Guennebaud et al. [GBP06].

Apart from these multipass splatting methods, several alternative point-rendering techniques have been presented; in particular, a "single-pass" splatting technique [ZP06, ZP07], dedicated splatting hardware [WHA*07], and ray tracing of points [SJ00, AA03, WS05]. The "single-pass" splatting technique proposed by Zhang and Pajarola [ZP06, ZP07] requires a costly preprocess to compute groups of non-overlapping splats (in object space) and is therefore less suitable for dynamic point sets. While the dedicated hardware published by Weyrich et al. [WHA*07] achieves promising performance results, the prototypical hardware cannnot provide the rendering performance of splatting techniques implemented on today's GPUs. Ray tracing of points was first published by Schaufler and Jensen [SJ00] and led to work by Adamson and Alexa [AA03], and Wald and Seidel [WS05].

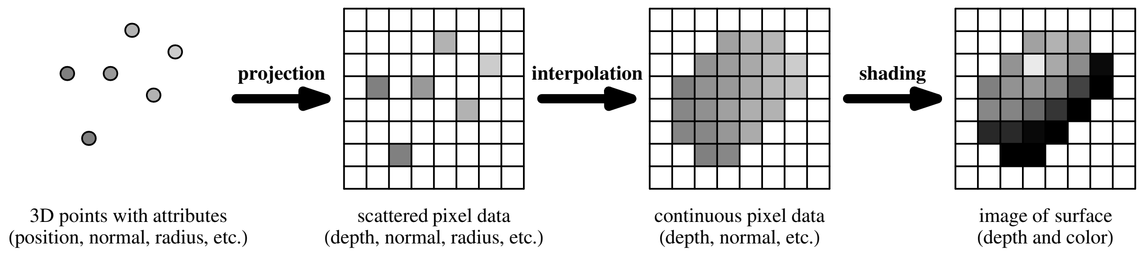

Figure 1: *Data flow in the proposed point-based surface rendering technique.*

The approach is particularly advantageous for many global illumination effects and for rendering of very large point models, which do not fit into graphics memory.

Inspite of its linear complexity, there are not many hardware-accelerated implementations of the pull-push algorithm [GGSC96] because it requires a logarithmic number of switches of the render target. These switches have been a major bottleneck in the past; therefore, Lefebvre et al. [LHN05] tried to avoid them in their approximation of the pull-push algorithm at the cost of a worse time complexity and reduced interpolation quality. However, a GPU-based implementation of the pull-push algorithm by means of the OpenGL extension for framebuffer objects performs extremely efficient as demonstrated by Strengert et al. [SKE06]. This motivated our research on applying a GPU-based pull-push interpolation to point-based rendering, which led to the method described in the next section.

3. Point Rendering Using Image Reconstruction

Figure 1 presents an overview of our point-based surface rendering technique. The input data of our algorithm consists of an unordered set of three-dimensional points with attributes, which are projected to the viewport. However, in contrast to splatting techniques, only one single pixel is rasterized for each point. Details about the projection are discussed in Section 3.1. After projecting the points, a continuous surface is reconstructed from the resulting scattered pixels by means of a pull-push interpolation as discussed in Sections 3.2 and 3.3. Based on the resulting continuous pixel data, the deferred shading of the surface is computed as described in Section 3.4.

3.1. Projection of Points to Single Pixels

Points are projected to the viewport by a standard model-view matrix. In addition to viewport clipping, we also employ backface culling based on the local surface normal vector specified for each point. Apart from this normal vector, the only additional attribute required by our algorithm is a radius that specifies the extent of the point's influence. This radius corresponds to the splat size of splatting techniques and can be computed from the local sampling spacing. Since we are only concerned with point rendering in this work, we will not discuss the acquisition of points, their normal vectors, nor sampling spacings; instead we refer the reader to the literature on these problems [KB04, GP07]. For the same reason, hierarchical point representations and dynamic level-of-detail selection are not discussed in this work while they could be integrated in the point projection of our method.

Further potential point attributes are, for example, color and texture coordinates. For each point at most one pixel is rasterized; thus, the point's attributes are written to at most one pixel of the framebuffer. Since a depth buffer is employed for depth culling, each pixel stores the attributes of at most one point. Note that the point's attributes will usually consist of more than four components; thus, a hardware-based implementation requires support for sufficiently many multiple rendering targets.

Since at most one pixel is rasterized for each point, pixel overdraw is dramatically reduced in comparison to splatting techniques. In fact, this advantage was already noted by Grossman and Dally [GD98]. Another advantage of our approach is the limitation to one object-order pass, i.e., all points are processed only once. The requirement of two object-order passes, i.e., a visibility pass in addition to the attribute pass, is a major disadvantage of most hardware-accelerated splatting approaches as noted by Zhang and Pajarola [ZP06, ZP07] and Weyrich et al. [WHA*07].

3.2. Pull-Push Interpolation: Pull Phase

The pull-push algorithm consists of a pull phase and a subsequent push phase [GGSC96]. The former phase computes coarser levels of an image pyramid of the viewport image by reducing the pixel dimensions by a factor of two in each step. The push phase of our method employs this image pyramid to fill arbitrarily large gaps, i.e., to interpolate missing pixels and also to overwrite pixels that are occluded by a surface as discussed in the next section.

In the pull phase, pyramid levels are computed in bottom-up order based on the viewport image containing projections

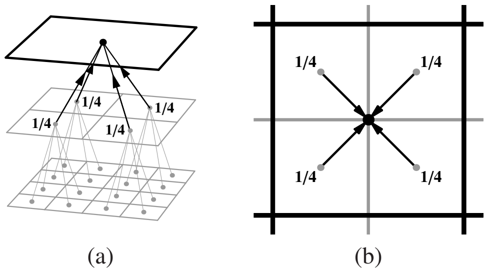

Figure 2: *Interpolation of pixel attributes in the pull phase.*

of points. The attributes of a pixel of a coarser level are determined by averaging the corresponding four pixels of the finer pyramid level as illustrated in Figure 2. However, only pixels that specify valid data are included in the average. Whether a pixel is valid or not, is indicated by a binary flag per pixel. On the finest level, only the one-pixel projections of points are marked to specify valid data while all other pixels are marked invalid. When averaging four invalid pixels to compute a pixel of a coarser pyramid level, the new pixel is also marked invalid and is left to be computed during the push phase.

After eliminating invalid pixels, a preliminary depth test is performed to also eliminate occluded pixels. To this end, each one-pixel projection is associated with a depth interval. The minimum depth of this interval is determined by the projected point's z coordinate while the maximum depth is computed by the minimum depth plus the radius of the point. A pixel is only used for averaging if its depth interval intersects the depth interval of the frontmost pixel, i.e., the pixel with the smallest minimum depth coordinate among the valid pixels, which potentially contribute to the average as depicted in Figure 2. Note that even though occluded pixels are removed from the interpolation of pixels of coarser levels, they are still present in the finer level. However, occluded pixels are recomputed in the following push phase as discussed in the next section.

In general, the attributes of a new pixel of a coarser level are determined by averaging the valid, "unoccluded" pixels; however, the minimum and maximum depth values of a new pixel are set to the smallest and largest depth values of all contributing pixels, respectively. This guarantees that the depth interval of the new coarser pixel contains all intervals of the averaged pixels from the finer level.

To accurately reconstruct the model's silhouette and boundaries, the reconstruction includes elliptical box-filters, which limit the region of influence of each projected point. Each pixel's ellipse is computed by an orthogonal projection of a circle onto the view plane. The circle's orientation in object space is determined by the pixel's normal vector while its radius is set to the radius mentioned in Section 3.1. Thus, the ellipse's major axis is aligned perpendicularly to the 2D

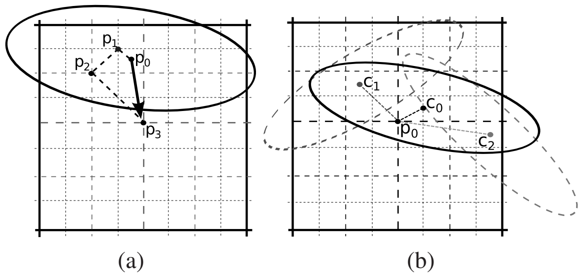

Figure 3: (a) *Displacement vector (thick arrow) for pixel p_3 computed after three steps of the the pull phase. Pixel p_0 is the original projected sample at the center of the ellipse, p_1 is the center of the pixel at level 1, etc. The dotted lines represent the difference vectors summed at each iteration.* (b) *The attributes computed for pixel p_0 define a new ellipse (continuous line), with center at c_0, by averaging two coarser level ellipses (dotted lines) with centers at c_1 and c_2.*

projection of the normal vector and its length is twice the radius; furthermore, the minor axis is parallel to the normal vector and its magnitude is the length of the major axis multiplied by the normal's z coordinate. While the center of an ellipse in the finest pyramid level is just the pixel's center, an additional 2D displacement vector is computed for each pixel to specify the ellipse's center with sub-pixel precision in coarser pyramid levels. This vector is similar to the sample offset proposed by Popescu et al. [PEL*00].

For the pull phase, the ellipses are not actually used; nonetheless, normal vectors and radii are computed for all pixels by averaging valid pixels of finer levels, as explained above, because this data is required in the push phase. Before averaging displacement vectors, however, the difference vector from the corresponding pixel center to the center of the coarser pixel is added; the arrows in Figure 2b depict these vectors. This process guarantees that the coarser pixel maintains a reference to the exact position of the ellipse. An example of the evolution of the displacement vector during the pull phase is illustrated in Figure 3a.

When a pixel of a coarser level is computed by averaging more than one valid pixel, the new normal vector, radius, and displacement vector define an ellipse that approximates two or more ellipses from the finer level (see Figure 3b). During the push phase, the ellipse is reconstructed and acts as an inside/outside test, i.e., an elliptical box-filter, to improve the interpolation of pixels as described in the next section.

3.3. Pull-Push Interpolation: Push Phase

After the image pyramid has been built in bottom-up order in the pull phase, the push phase works in top-down order, i.e., from coarser to finer levels. In this push phase, only the attributes of invalid and "occluded" pixels are (re)computed. Here, a valid pixel is considered "occluded" if its minimum

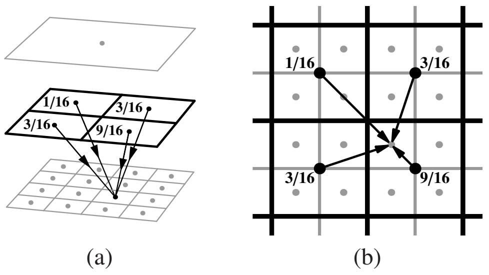

Figure 4: *Interpolation of pixel attributes in the push phase.*

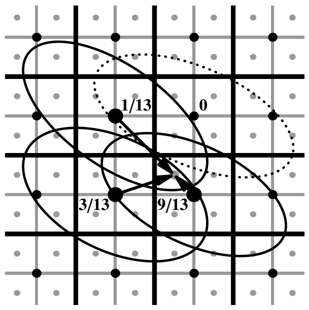

Figure 5: *Push interpolation for three contributing pixels.*

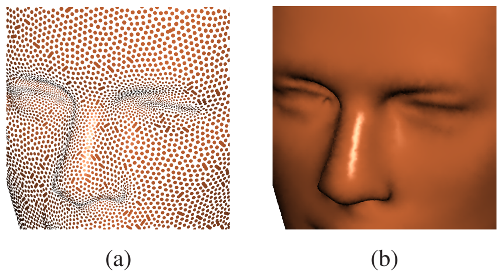

Figure 6: (a) *Illustration of the point distribution in the Head model,* (b) *rendering with our method.*

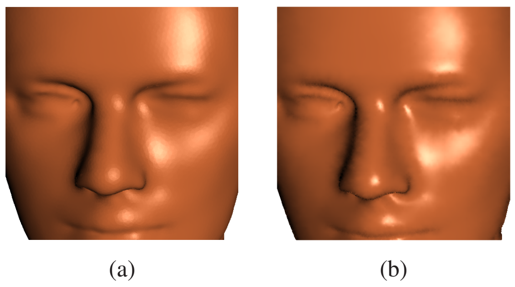

Figure 7: *Comparison between* (a) *non-deferred shading, and* (b) *deferred shading in our method.*

depth value is outside the depth interval of the corresponding pixel in the next coarser pyramid level.

The basic interpolation scheme is depicted in Figure 4. The attributes of four pixels of a coarser pyramid level are used to interpolate the attributes of a pixel of a finer level. The weights are depicted in Figure 4 and correspond to bi-quadratic B-spline subdivision, as published by Catmull and Clark [CC78]; they also correspond to Doo-Sabin subdivision of a regular quadrilateral mesh.

Analogously to the pull phase, only pixels with valid attributes are included in the interpolation; thus, the weights have to be renormalized in the case of less than four valid pixels. For example, Figure 5 illustrates the interpolation of a pixel from only three pixels of a coarser level. The weights of the contributing pixels according to Figure 4 are $\frac{1}{16}$, $\frac{3}{16}$, and $\frac{9}{16}$. After division by their sum $\frac{13}{16}$, the renormalized weights depicted in Figure 5 are $\frac{1}{13}$, $\frac{3}{13}$, and $\frac{9}{13}$.

In addition to the binary flag marking invalid pixels, elliptical box-filters, which have been computed in the pull phase for all pixels, are used to eliminate pixels whose region of influence does not include the pixel that has to be recomputed. If the center of this pixel is not within the ellipse of a coarser pixel, the latter is eliminated from the weighted average; i.e., its weight is set to 0. In Figure 5, the pixel labeled “0” does not contribute to the interpolation, since its ellipse (dotted in Figure 5) does not cover the center of the pixel to be recomputed, while the three other ellipses include this center. For clarity, all displacement vectors of ellipse centers relative to pixel centers are assumed to be $(0,0)^\top$ in this example; however, this is usually not the case.

The push phase only recomputes invalid and occluded pixels; thus, the attributes of all other pixels are just copied from the image pyramid computed in the pull phase.

3.4. Deferred Shading

Once the push phase has been performed for all levels of the image pyramid, the finest level contains interpolated attributes for all pixels that are determined to be within the silhouette of a reconstructed surface as indicated by the binary attribute flag. For these pixels, Phong shading can be computed since the normal vectors have also been interpolated. Additionally, any optional attributes such as colors or texture coordinates can be included in the shading computation.

Figures 6 demonstrates that gaps between points are interpolated with correct depth occlusions while Figure 7 compares non-deferred shading by a software splatting technique in Figure 7a with the deferred shading by our method depicted in Figure 7b, which features sharper specular highlights.

3.5. Ping-Pong Implementation on GPUs

While the hardware-accelerated implementation of the point projection and the deferred shading computation is straight-

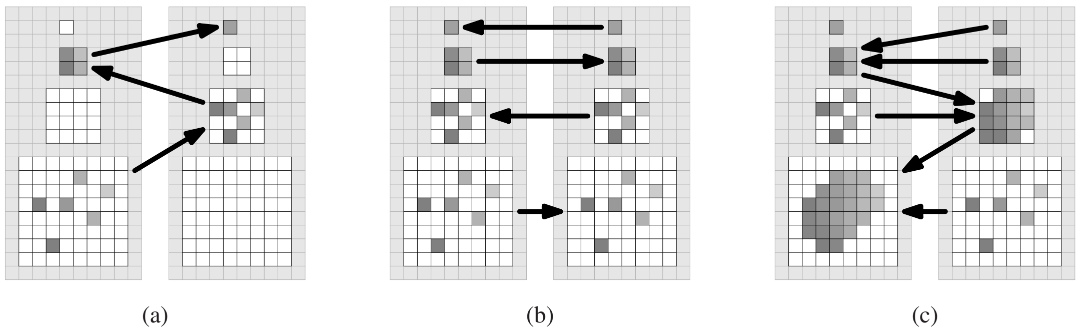

Figure 8: *Ping-pong rendering between two image buffers:* (a) *bottom-up pull phase,* (b) *copy phase,* (c) *top-down push phase.*

forward, the GPU-based implementation of the pull-push algorithm is less obvious. Strengert et al. [SKE06] present an implementation that reads and writes at the same time to one render target. While the result of this operation is unspecified by OpenGL, some GPUs offer undocumented support for it. In this section we present a GPU implementation that does not rely on unspecified hardware behavior and is therefore implementable on any OpenGL hardware supporting framebuffer objects.

Figure 8 illustrates our ping-pong implementation; in particular, the pull phase is depicted in Figure 8a. Two image buffers are required, which alternately act as renderbuffer and texture image, respectively. Each image buffer contains all levels of the pyramid image; for the purpose of illustration, Figure 8 employs a rather inefficient packing of the pyramid levels into one image buffer. As depicted by Figure 8a, the pull phase rasterizes all odd levels of the pyramid image into one image buffer and all even levels into the second image buffer. As discussed in Section 3.3, the push phase requires access to data from one odd and one adjacent, i.e., even, level of the pyramid at the same time. Thus, an additional copy phase is necessary, which simply copies each level image to the image buffer that is lacking it as illustrated in Figure 8b. After the copy phase is completed, the ping-pong implementation of the push phase rasterizes all levels of the pyramid in top-down order. As depicted in Figure 8c, it can access two adjacent levels of the pyramid image without reading and writing to the same image buffer at the same time.

The pull, copy, and push phases require several render target switches; specifically, the number of switches is of $O(\log\sqrt{m})$ for a square viewport image consisting of m pixels. Thus, these switches are not time critical for modern GPUs and viewports of reasonable size. Results of a prototypical GPU implementation are reported in the next section.

4. Results

We tested our algorithm on a GeForce 7800 GTX with 512 MB memory connected via PCI Express 16× to a Linux computer with an Intel Pentium 4 CPU (3 GHz), 2 GB RAM, and installed NVIDIA driver, version “1.0-9755.” The models were preprocessed to compute a normal vector and a radius of influence of each point. At runtime, all point attributes were transferred to the GPU by means of OpenGL vertex buffer objects. Image data was processed in one OpenGL framebuffer object with four 16-bit floating-point RGBA image buffers and a 32-bit depth buffer. All vertex and fragment shaders were implemented in the OpenGL shading language.

Rendering times for several point-based models are summarized in Table 1; exemplary renderings of these models are depicted in Figures 6b, 7b, 9, and 10. The fourth column of Table 1 presents rendering times in milliseconds for a 512×512 viewport while the sixth column presents times for a 1024×1024 viewport. Apart from the total time per frame, we have also included two times in parentheses: the time required for the projection of points as described in Section 3.1 and for the pull-push interpolation discussed in Sections 3.2 and 3.3. As expected, these two operations require most of the rendering time while other operations, e.g., the deferred shading, are almost negligible.

Our measurements demonstrate that the rendering time for small models is dominated by the pull-push interpolation; i.e., by the viewport resolution. On the other hand, the rendering time for large models on small viewports is dominated by the projection of points and therefore by the number of points. For large models on a 1024×1024 viewport, our implementation renders the equivalent to between 50 M and 60 M splats per second—including surface reconstruction and deferred shading. For comparison, Zhang and Pajarola [ZP06] reported a performance of up to 24.9 M splats per second and Guennebaud et al. [GBP06] reported 37.5 M splats per second—both for the same viewport size on the same GPU. While our algorithm does not provide the same image quality, it should be noted that the performance of our method is independent of the size of points while splatting techniques achieve their peak performance only for small splat sizes.

Table 1: *Models and rendering performance.* **Each total rendering time per frame is followed in parentheses by the time for the point projection and the time for the pull-push interpolation (all rendering times are in milliseconds).*

		512 × 512 viewport		1024 × 1024 viewport	
model	# points	fps	time per frame*	fps	time per frame*
Head	25 K	82	12 ms (0.2 ms, 12 ms)	24	42 ms (0.3 ms, 38 ms)
Armadillo	173 K	71	14 ms (1.4 ms, 13 ms)	22	46 ms (1.7 ms, 41 ms)
Happy Buddha	544 K	61	16 ms (5.1 ms, 11 ms)	21	47 ms (4.9 ms, 39 ms)
Asian Dragon	3610 K	26	39 ms (28 ms, 10 ms)	14	71 ms (29 ms, 38 ms)
Thai Statue	5000 K	20	49 ms (38 ms, 10 ms)	12	82 ms (39 ms, 38 ms)

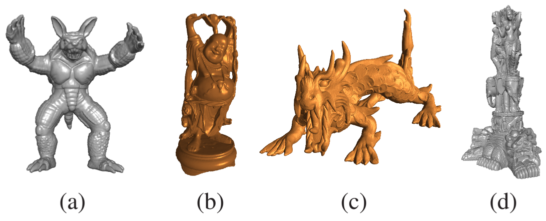

Figure 9: *Renderings with our method of the* (a) *Armadillo,* (b) *Happy Buddha,* (c) *Asian Dragon and* (d) *Thai Statue.*

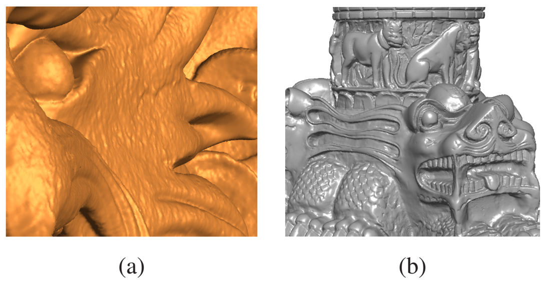

Figure 10: *Renderings of details of the* (a) *Asian Dragon and* (b) *Thai Statue.*

5. Future Work and Conclusions

We have presented a new pull-push interpolation for image-space reconstruction in point-based rendering. Compared to prior work by Grossman and Dally [GD98], our algorithm offers a significantly improved image quality by integrating elliptical box-filters and employing deferred Phong shading. Moreover, we presented an efficient GPU implementation, which performs significantly better than splatting techniques in case of high rates of rasterization overdraw, e.g., for screen-filling scenes of high depth complexity. Therefore, we consider our method to be a valuable complement to previously published splatting techniques.

There are some limitations of our algorithm, which deserve future research. For example, extremely dense projections of points in screen space result in aliasing artifacts since only one point per pixel is taken into account. Moreover, the employed elliptical box-filters result in sharp silhouettes, i.e., they also feature some aliasing. Furthermore, the surfaces computed by our variant of the pull-push interpolation are not as smooth as, for example, biquadratic B-spline subdivision surfaces [CC78].

On the other hand, there are many promising extensions and applications of our algorithm. Of particular interest is the integration of hierarchical data structures for large point-based models and scenes; e.g., the sequential point trees published by Dachsbacher et al. [DVS03].

6. Acknowledgements

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

References

[AA03] ADAMSON A., ALEXA M.: Ray tracing point set surfaces. In *SMI '03: Proceedings of Shape Modeling International 2003* (2003), pp. 272–279.

[BHZK05] BOTSCH M., HORNUNG A., ZWICKER M., KOBBELT L.: High-quality surface splatting on today's GPUs. In *Proceedings of the Eurographics/IEEE Symposium on Point-Based Graphics '05* (2005), pp. 17–24.

[BSK04] BOTSCH M., SPERNAT M., KOBBELT L.: Phong splatting. In *Proceedings of the Eurographics Symposium on Point-Based Graphics '04* (2004), pp. 25–32.

[Bur88] BURT P. J.: Moment images, polynomial fit filters, and the problem of surface interpolation. In *Proceedings of Computer Vision and Pattern Recognition* (1988), pp. 144–152.

[CC78] CATMULL E., CLARK J.: Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design 10*, 6 (1978), 350–355.

[CCC87] COOK R. L., CARPENTER L., CATMULL E.: The Reyes image rendering architecture. In *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (1987), pp. 95–102.

[CHP*79] CSURI C., HACKATHORN R., PARENT R., CARLSON W., HOWARD M.: Towards an interactive high visual complexity animation system. In *SIGGRAPH '79: Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques* (1979), pp. 289–299.

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. In *Proceedings SIGGRAPH '03* (2003), pp. 657–662.

[For79] FORREST A. R.: On the rendering of surfaces. In *SIGGRAPH '79: Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques* (1979), pp. 253–259.

[GBP04] GUENNEBAUD G., BARTHE L., PAULIN M.: Deferred splatting. *Computer Graphics Forum 23*, 3 (2004), 653–660.

[GBP06] GUENNEBAUD G., BARTHE L., PAULIN M.: Splat/mesh blending, perspective rasterization and transparency for point-based rendering. In *Proceedings of the IEEE/Eurographics/ACM Symposium on Point-Based Graphics '06* (2006), pp. 49–58.

[GD98] GROSSMAN J. P., DALLY W. J.: Point sample rendering. In *9th Eurographics Workshop on Rendering '98* (1998), pp. 181–192.

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The Lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (1996), pp. 43–54.

[GP03] GUENNEBAUD G., PAULIN M.: Efficient screen space approach for hardware accelerated surfel rendering. In *Proceedings of Vision, Modeling and Visualization 2003* (2003), pp. 485–495.

[GP07] GROSS M., PFISTER H. (Eds.): *Point-Based Graphics*. Morgan Kaufmann Publishers, 2007.

[KB04] KOBBELT L., BOTSCH M.: A survey of point-based techniques in computer graphics. *Computers & Graphics 28*, 6 (2004), 801–814.

[LHN05] LEFEBVRE S., HORNUS S., NEYRET F.: Octree textures on the GPU. In *GPU Gems 2* (2005), Pharr M., (Ed.), Addison Wesley, pp. 595–613.

[LW85] LEVOY M., WHITTED T.: The use of points as a display primitive, Technical Report TR 85-022, University of North Carolina at Chapel Hill, 1985.

[PEL*00] POPESCU V., EYLES J., LASTRA A., STEINHURST J., ENGLAND N., NYLAND L.: The WarpEngine: An architecture for the post-polygonal age. In *Proceedings SIGGRAPH '00* (2000), pp. 433–442.

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proceedings SIGGRAPH '00* (2000), pp. 335–342.

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: A multiresolution point rendering system for large meshes. In *Proceedings SIGGRAPH '00* (2000), pp. 343–352.

[RPZ02] REN L., PFISTER H., ZWICKER M.: Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum (Eurographics 2002) 21*, 3 (2002), 461–470.

[SJ00] SCHAUFLER G., JENSEN H. W.: Ray tracing point sampled geometry. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000* (2000), pp. 319–328.

[SKE06] STRENGERT M., KRAUS M., ERTL T.: Pyramid methods in GPU-based image processing. In *Proceedings of Vision, Modeling, and Visualization 2006* (2006), pp. 169–176.

[SP04] SAINZ M., PAJAROLA R.: Point-based rendering techniques. *Computer & Graphics 28*, 6 (2004), 869–879.

[SPL04] SAINZ M., PAJAROLA R., LARIO R.: Points reloaded: Point-based rendering revisited. In *Proceedings of the Eurographics Symposium on Point-Based Graphics '04* (2004), pp. 121–128.

[WHA*07] WEYRICH T., HEINZLE S., AILA T., FASNACHT D., OETIKER S., BOTSCH M., FLAIG C., MALL S., ROHRER K., FELBER N., KAESLIN H., GROSS M.: A hardware architecture for surface splatting. *ACM Transactions on Graphics (Proceedings ACM SIGGRAPH 2007) 26*, 3 (2007).

[WS05] WALD I., SEIDEL H.-P.: Interactive ray tracing of point-based models. In *Proceedings of the Eurographics/IEEE VGTC Symposium on Point-Based Graphics '05* (2005), pp. 9–16.

[ZP06] ZHANG Y., PAJAROLA R.: Single-pass point rendering and transparent shading. In *Proceedings of the Eurographics/IEEE VGTC Symposium on Point-Based Graphics '06* (2006), pp. 37–48.

[ZP07] ZHANG Y., PAJAROLA R.: Deferred blending: Image composition for single-pass point rendering. *Computer & Graphics 31*, 2 (2007), 175–189.

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings SIGGRAPH '01* (2001), pp. 371–378.

[ZRB*04] ZWICKER M., RÄSÄNEN J., BOTSCH M., DACHSBACHER C., PAULY M.: Perspective accurate splatting. In *GI '04: Proceedings of the 2004 Conference on Graphics Interface* (2004), pp. 247–254.