Image-Based Techniques for Surface Reconstruction of Adaptively Sampled Models *

Ricardo Marroquim
ricardo@lcg.ufrj.br
Universidade Federal do Rio de Janeiro

Paulo Roma Cavalcanti
roma@lcg.ufrj.br
Universidade Federal do Rio de Janeiro

Abstract

Image based methods have proved to efficiently render scenes with a higher efficiency than geometry based approaches, mainly because one of their most important advantages: the bounded complexity by the image resolution, instead of by the number of primitives. Furthermore, due to their parallel and discrete nature, they are highly suitable for GPU implementations. On the other hand, during the last few years point-based graphics has emerged as a promising complement to other representations. However, with the continuous increase of scene complexity, solutions for directly processing and rendering point clouds are in demand. In this paper, algorithms for efficiently rendering large point models using image reconstruction techniques are proposed. Except for the projection of samples onto screen space, the reconstruction time is bounded only by the screen resolution. The method is also extended to interpolate other primitives, such as lines and triangles. In addition, no extra data-structure is required, making the strategy memory efficient.

1. Introduction

The area of point based graphics has grown during the last decade, reaching a high level of acceptance among the computer graphics community. With the advent of high-resolution registration equipments, such as 3D scanners, its importance became apparent as methods that could directly deal with the provided data. Furthermore, since scene complexity increases in a much higher rate than the output devices' resolution, the usefulness of some traditional representations were put on doubt for a set of applications.

Nevertheless, point-based graphics is still an emerging area. By far its goal is not to replace other representations, such as polygonal meshes, but to complement them where it is advantageous. With a dedicated symposium during the last five years, devoted sessions in the most important computer graphics conferences, and a recently released point-based graphics book [6], the area has broadened its scope and embraced techniques on various fields, e.g., 3D acquisition, surface reconstruction, processing & modelling, rendering, and animation.

The main advantage of the point-based representation is its lack of explicit connectivity information. Each sample contains all information necessary for processing and rendering, implying in a few benefits: first, memory space is decreased, since there is no need to store connectivity; second, during processing and modelling connectivity and topological restrictions do not have to be preserved; third, there is no need for surface reconstruction from the point cloud, for example, to generate a polygonal mesh; and finally, as the geometric primitive approaches the image primitive, i.e., a point and a pixel, valuable gains can be obtained by exploiting the rendering pipeline.

This work is mainly focused on rendering techniques for point-based models. The first researches evolved from the image-based rendering field, but quickly became an independent area itself. Surface Splatting is today the most popular method for the direct rendering of unstructured point clouds. However, it has long distanced its foundation from the image-space techniques premises. It is at this point that we advocate such methods to reclaim one of its most important contributions: the decoupling of the algorithm cost from the scene complexity.

We direct our attention to the visualization of large datasets, varying from a few hundred thousand to a few million samples, where we believe point representation is more valuable. This is specially true, since for very dense models triangles are projected to less than one pixel, causing an unnecessary overhead due to the primitive assembly process. The main contributions of this work are:

- An efficient algorithm for surface reconstruction of point-based models using image-based techniques. This work was published at the Symposium on Point-Based Graphics 2007 [12].

* This paper contains extracts of the D.Sc. thesis "Interactive Point-Based Rendering" by the first author.

- The extension of the previous algorithm to deal with other primitives using the same pipeline. The possibility to render models with triangles, lines, or points further demonstrates the usefulness of the method. As a proof of concept, a scheme for rendering tree models using lines is described, where not only fewer primitives are needed, but performance is improved. This work was published in the Computer & Graphics Journal [13].
- Further investigation of the screen-based surface reconstruction has led to new insights on how to improve the rendering quality. By creating a more exact reconstruction strategy, silhouettes and details are better preserved, while the performance is not significantly degraded. This work is to be published at the Sibgrapi 2008.
- Finally, we investigate the use of Level-of-Detail structures to improve the performance of the point projection phase (the only that depends on the number of samples). A first effort using recent GPU techniques is to be presented at the Siggraph 2008 posters session.

2. Related Work

The pioneer work on point-based rendering by Levoy and Whitted [11], proposed the conversion of any representation to points. In this manner, a standardized rendering algorithm could be employed using points as the *Lingua Franca* of computer graphics. Even if it did not cause much impact at the time, in a sense, they anticipated one of the reasons point-based representation has become popular recently: the continuous increase of visual complexity of graphics scenes. More than ten years later research on point based rendering was resumed, but this time coming from a different source, image-based techniques. For this reason here we present relevant related methods on both areas, allowing for a more thorough understanding of this work's motivations and purposes.

2.1. Image-Based Rendering

The area of image-based rendering is overlapped by several others, such as image-space and image-processing techniques. Here, we classify image-based as an approach that uses the image as graphics primitives, instead of the traditional geometric primitives. This classification is important, since methods such as ray tracing are accomplished in image-space, but rely heavily on the model's geometry, that is, are physically based.

The main motivation comes from photorealism: if the goal is to render images with photo quality, then why not use it as the input primitive? However, even though a photography offers an enormous quantity of information about structure and appearance, it is still a static image. Image-based rendering is mainly about solving the following problem: given a set of static images describing the scene, how can new viewpoints be efficiently extracted to provide a dynamic high quality setting?

Two algorithms, which were developed almost simultaneously, describe appropriately the use of images as primitives: the "Light Field" by Levoy and Hanrahan [10], and the "Lumigraph" by Gortler et al. [5]. They are both based on the use of the plenoptic function to describe the rays of light traversing a scene. The full 7D plenoptic function is defined as the intensity of light rays passing through the camera center at every 3D location (V_x, V_y, V_z), at every possible angle (θ, ϕ), for every wavelength λ, at every time t, i.e., $P_7(V_x, V_y, V_z, \theta, \phi, \lambda, t)$ [9]. The authors realized that this function could be reduced to 4D by parameterizing the rays as a line intersecting two planes, one being the camera plane and the other the object plane. By using the 4D function $P_4(u, v, s, t)$, it is possible to treat the scene as a volume, where each specific viewpoint is characterized by the extraction of a 2D image from the 4D volume.

As the area evolved, it strayed away from a pure appearance based approach; new algorithms emerged adding geometric information to the image primitives (in fact, Lumigraph already used some geometric hint to accomplish better depth tests). Techniques such as *Billboards* and *Depth Sprites* use some geometric information to efficiently render dynamic scenes with few images. These strategies were specially employed in the game industry to add details to the environment, without having to resort to complex geometric models.

However, to provide a fully interactive environment, or a system able to render the scene from any viewpoint, it is necessary to turn to more complex structures. *3D Warping* uses multiple reference images from a scene and stores depth information per pixel. To render an arbitrary scene, the pixels are reprojected to 3D space and then projected to the new viewpoint.

In the same sense, Shade et al. [18] proposed the *Layered Depth Images*, or LDIs. Basically, it is the use of a single reference image with multiple color and depth information per pixel. To render from a new viewpoint, the reference image's pixels are projected to the new reference. However, this method is limited by its fixed resolution, implying that the sample density may not be convenient for every viewpoint. To overcome this problem, Chang et al. [3] proposed the LDI Trees, using an octree with an LDI stored per node. In this way, each pixel is stored in different resolutions and can be dynamically selected to compose the final image.

2.2. Point-Based Rendering

Following Levoy and Whitted's [11] initial proposal to use points as graphics primitives, Grossman and Dally [7] derived a point-based rendering system from previous image-based techniques. In fact, the idea was not yet to render point clouds, but to project the point data onto uniform structures, i.e., reference images. However, differently from 3D Warping, each sample also stores a normal, adding surface variational information. This is specially important because, with view-independent samples, it is not only possible to eliminate much of the redundancy from previous approaches, but also to compute dynamic illumination. The "pull-push" image-processing technique is employed to fill holes left by the reconstruction algorithm, that are usually caused by insufficient sampling under magnification.

The *Surfels* approach, by Pfister et al. [15], uses a similar methodology as the previous algorithm. However, instead of reference images, an LDI Tree is created. In addition, the samples are splatted onto the image buffer to perform a depth test. Nevertheless, they also make use of the "pull-push" algorithm to fill holes between the splats.

Surface Splatting [23, 24, 22] is today the most popular point-based renderer. It proposes to directly render unstructured point clouds by projecting to image space a surface patch, representing the sample's local approximation. These projections are called reconstruction kernels, and together with an anti-aliasing filter, compose the frame buffer by computing the contribution to every covered pixel. In a last phase, all contributions per pixel are weighted averaged to render the final surface reconstruction of the object.

Motivated by the high quality images provided by Surface Splatting, efficient implementations were proposed using modern GPU techniques [17, 2, 1, 8]. The main limitation of these algorithms is the impossibility of using the graphics hardware pipeline to perform the necessary ternary depth test. This test includes not only the visible/occluded traditional options, but also an extra merge configuration to allow splat composition of pixels on the same surface. In this way, GPU implementations relied on a two-pass approach: one for visibility and another for rendering. More recently, Zhang and Pajarola [20, 21] proposed a one-pass approach, but it depends on complex structures to separate the splats in non-overlapping groups.

3. Point Rendering Using Image Reconstruction

We propose to extend the use of image processing techniques used in early point-based works, to directly render unstructured point clouds. As will be shown, image reconstruction offers a complexity advantage over splatting, since it is independent of the number of samples and their sizes. Furthermore, differently from other image reconstruction approaches for point based models, no extra data structure is needed, thus, much less restrictions on sampling density or uniformity are imposed.

An overview of the proposed *Pyramid Point Renderer (PPR)* algorithm is depicted in Figure 1. The input data is an unordered set of three-dimensional points with attributes, which are projected to the viewport. Each projection is rasterized as a single pixel. Then, by means of a pull-push interpolation the continuous surface is reconstructed. In a last pass, deferred shading is applied. These steps are described in the following subsections.

3.1. Point Projection

A point is projected to the viewport, and rasterized as a single pixel. Each point contains, apart from its three dimensional coordinates, a surface normal and a radius indicating the local sampling density. These attributes are enough to reconstruct the surface in image space. However, extra attributes such as color or texture coordinates can also be interpolated. During projection, backface culling is applied, as well as a depth test to deal with multiple projections onto the same pixel.

3.2. Pull-Push interpolation: Pull Phase

The pull-push algorithm was employed by Ogden et al. [14] to interpolate scattered data in two-dimensions. Its main purpose was to reconstructed missing pixels or patches from images. It was later employed in the Lumigraph pipeline [5], and following, in early point-based methods [7, 15]. The algorithm was recently adapted to the GPU by Strengert et al. [19], achieving interpolation times of a few milliseconds for high resolution settings, i.e., 1024^2 and 2048^2. The pull phase consists of building a hierarchical pyramid of the image by reducing its dimensions by a factor of two. In the subsequent push phase, the pyramid is traversed from top to bottom synthesing missing information, i.e., empty pixels.

During the pull phase, the pyramid is constructed by creating coarser resolutions of the framebuffer containing the projected samples. In each step, a coarser pixel is computed by averaging the corresponding four pixels of the finer level. However, only valid pixels are included in the average, that is, pixels that contain sample projections. If all four pixels are invalid, the coarser pixel is also marked as so and left to be reconstructed during the next phase.

A sample's projected attributes represent an ellipse in two-dimensions. In other words, it is the projection of the circular extent of the sample in object space. Since the radii are usually very small, the circle is orthogonally projected,

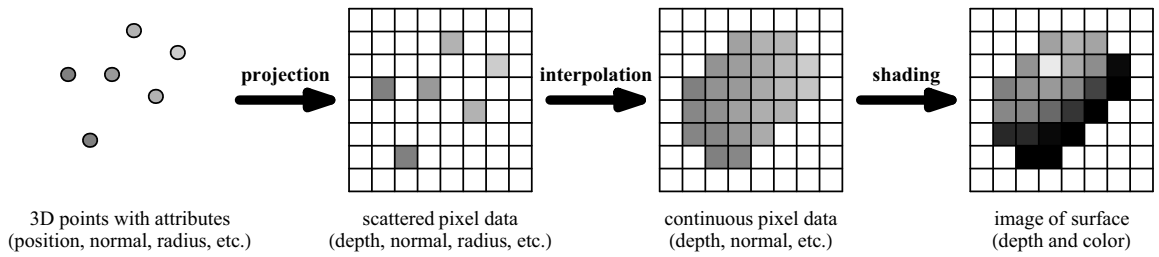
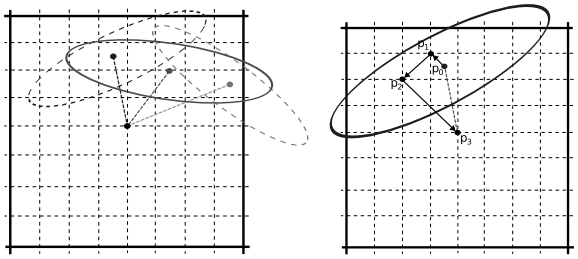


Figure 1. Data flow in the proposed point-based surface rendering technique.



(a) (b)

Figure 2. (a) A merge of two ellipses. (b) The displacement vector summed after three iterations of the pull algorithm, where p_0 is the original projected sample at the center of the ellipse, p_1 is the pixel at level 1, and so forth. The dotted line represents the displacement vector for p_3 at level 3.

instead of perspectively, without introducing significant errors. Furthermore, an early depth test is employed to discard occluded samples. This is done by comparing the depth component for each sample to be averaged with the front most sample. Occluded pixels are not included in the average, and are left to be recomputed during the push phase.

A displacement vector is the last information stored per pixel. It indicates the 2D distance from the pixel's center to the center of the stored ellipse. An example of the evolution of the displacement vector during the pull phase is illustrated in Figure 2.

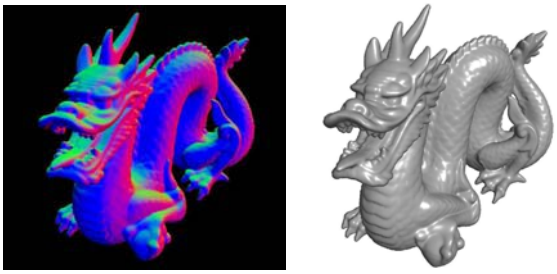
3.3. Pull-Push interpolation: Push Phase

Once the pyramid has been built from bottom to top, the push phase reconstructs the missing pixels in the opposite direction. Each invalid pixel computes its value by averaging up to four corresponding pixels in the immediately coarser level. An inside/outside test is carried out to determine if a coarser pixel will be used in the averaging scheme, i.e., if the pixel is inside its elliptical extent. The new attributes are interpolated by averaging the coarser valid pixels, weighted by the elliptical distance from the pixel to the ellipse's center. This is similar to the elliptical kernels used in Surface Splatting.

As in the pull phase, a depth test is also employed here. It is applied to the valid pixels in order to determine if they are occluded and must also be recomputed. The pixel's depth is tested against the depth interval of the pixel directly above it in the pyramid.

3.4. Deferred Shading

After the push phase has been performed for all levels, the highest resolution level contains interpolated attributes for all pixels considered to be inside the reconstructed surface. Since each pixel has a normal vector, Phong shading can be computed in a $O(n)$ pass, where n is the number of pixels. If other attributes were also interpolated, such as color or texture coordinates, they can be included in the shading computation. Figure 3 illustrates the buffer containing the interpolated normals and the final image with deferred shading; while Figure 4 depicts the head model with and without per vertex color interpolation.



(a) (b)

Figure 3. The normal map (a) generated by the pull-push algorithm, and the resulting image (b) after per-pixel shading.

		Without Color Buffer		With Color Buffer	
model	# points	fps	time per frame*	fps	time per frame*
Armadillo	173 K	89	11 ms (1.2 ms, 8.5 ms)	46	22 ms (1.5 ms, 18 ms)
Dragon	437 K	78	13 ms (2.2 ms, 8.9 ms)	44	23 ms (2.9 ms, 18 ms)
Happy Buddha	544 K	76	13 ms (2.6 ms, 8.8 ms)	42	24 ms (3.4 ms, 18 ms)
Asian Dragon	3610 K	36	28 ms (18 ms, 8.3 ms)	23	45 ms (25 ms, 17 ms)
Thai Statue	5000 K	29	35 ms (25 ms, 8.2 ms)	18	55 ms (34 ms, 18 ms)

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).

(a) (b)

Figure 4. Deferred shading with *(a)* constant material color and *(b)* per-vertex diffuse color.

3.5. Results

We tested our algorithm on a GeForce 8800 GTS with 640 MB memory with an Intel Core 2 Duo 6600 CPU (2.4 GHz) with 2 GB RAM. The models were preprocessed to compute a normal vector and a radius of influence of each point.

Rendering times for several point-based models are summarized in Table 1, while exemplary renderings of these models are depicted in Figure 5. The fourth and sixth columns of Table 1 present rendering times in milliseconds without and with interpolation of a surface color, respectively. Apart from the total time per frame, two times in parentheses are also included: 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. Note that the interpolation time is nearly constant for all models. All renderings where performed with a 1024×1024 viewport.

For large models without interpolation of surface color, our implementation renders the equivalent to about 130 M splats per second, including surface reconstruction and deferred shading. If the surface colors are included, the rendering performance is reduced to about 90 M splats per second.

Our previous implementation using a GeForce 7800 GTX achieved the equivalent to between 50 M and 60 M splats per second. For comparison, Zhang and Pajarola [20] reported a performance of up to 24.9 M splats per second, and Guennebaud et al. [8] reported 37.5 M splats per second, both for the same viewport size on the same GPU.

4. Line Rendering Using Image Reconstruction

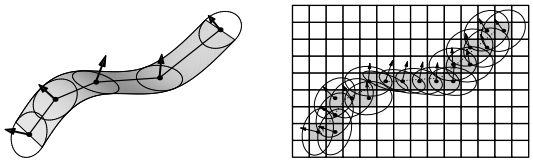
The work presented in Section 3 was extended to include line strips as rendering primitive. More specifically, two new types are included: flat line strips (ribbons) and cylindrical line strips (tubes). These two primitives are not only illuminated using different methods, but are also reconstructed differently using the pyramid algorithm. An identifier is included with the point sample to allow different types of primitives to be rendered in a single pass. A plant rendering system is described in Section 4.3 to exemplify the use of line primitives.

(a) (b)

Figure 5. Renderings with our method of the *(a)* Asian Dragon and *(b)* Thai Statue.

4.1. Ribbons

In order to use the PPR algorithm to render flat line strips, or ribbons, they must be represented as projected samples in image space. This is achieved by rendering polylines, where each point is attributed with the local normal vector of the ribbon's surface, and half its width as the radius parameter. The rasterized polyline generates a sequence of pixels representing point samples. The scheme is depicted in Figure 6. However, in contrast to the point samples described in the previous section, backface culling is not employed to ribbons. Furthermore, lightning must be computed for both sides.

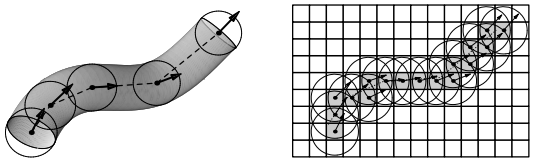


(a) (b)

Figure 6. (a) Representation of a ribbon by a polyline consisting of five vertices with normals and radii. (b) Illustration of the rasterization of a one-pixel-wide polyline corresponding to the centerline of a ribbon. Normals and radii are interpolated for each pixel covered by the polyline.

4.2. Tubes

Tubes represent bended cylinders of varying width and curvature, as illustrated in Figure 7a. The tube's centerline is approximated by a polyline, where each vertex has radius equals half the diameter, and normal with the same direction as the tangent vector. Differently from ribbons and point-based surfaces, the interpolation algorithm always considers the normalized vector to the camera center as the surface's normal vector. As shown in Figure 7, this procedure renders disks parallel to the view plane approximating the width of the projected tube.



(a) (b)

Figure 7. (a) Representation of a tube where each vertex stores a tangent vector and radius. (b) Illustration of the rasterization of a tube's polyline.

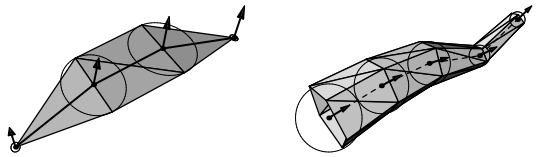
The deferred shading of tubes uses a basic approach for diffuse illumination of a line by a single light source. A surface normal n is determined by projecting the light vector l onto the plane orthogonal to the tangent vector t.

4.3. Example: Leaves and Branches

The use of lines is exemplified using a tree model, where branches are approximated by tubes and leaves by ribbons. The apple tree model from the pbrt book [16] is initially converted from polygons to lines. Leaves are converted from four triangles and six vertices to a polyline with four vertices; on the other hand, rings of the branches are heuristically determined to create the representing polylines. Figure 8 illustrates the conversion strategy.

The leaves were converted from 200K triangles and 451K vertices to 150K ribbons with 200K vertices. Likewise, 351K triangles and 371K vertices representing the branches, resulted in 63K tubes with 69K vertices. Only the 227 trunk triangles were not converted, since tubes are not appropriate for representing thick structures. The direct rendering of triangles using the pyramid algorithm is further discussed in Section 5.

The full original and line-based models are presented in Figure 9. Our implementation achieves 31fps against 28fps of the triangle model. However, as the number of primitives



(a) (b)

Figure 8. (a) Illustration of the conversion of a leaf (a) and a branch (b) to polylines.

increases so does the difference, e.g., the pyramid algorithm renders 10 trees in 15fps, while the traditional triangle renderer achieves 9fps. In addition, only about one third of the vertices are required for the tubes and ribbons representation.

5. Combining Points, Lines, and Polygons

As stated in Section 1, points complement other representations; it is unlikely one primitive is optimal for all elements of a scene, and LOD strategies may benefit from switching primitives dynamically. Fortunately, our algorithm can easily combine different primitives using the same rendering pipeline.

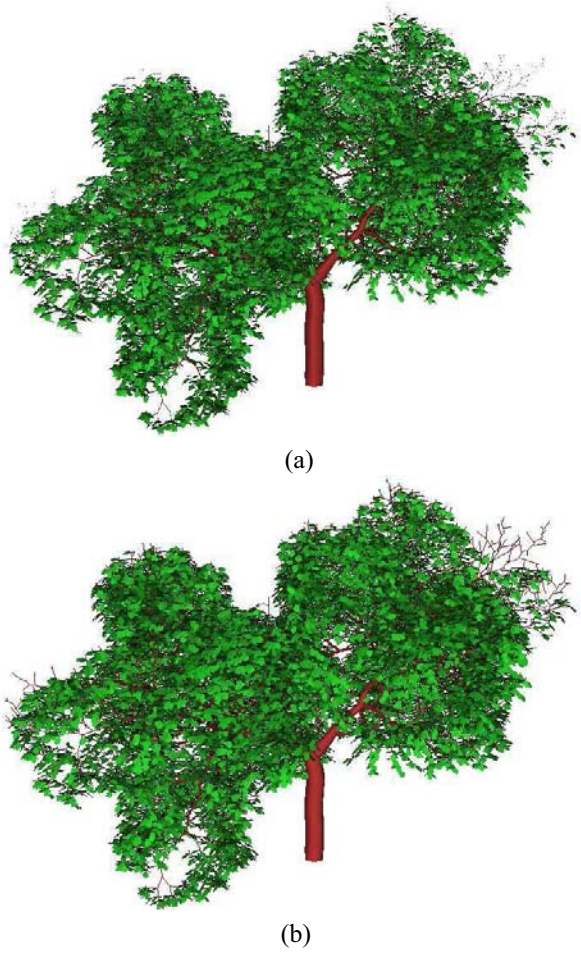
Apart from points and lines, triangles can also be rendered within the pyramid approach. The vertices are assigned normals as usual, and a radius size close to zero. When a triangle is rasterized, fragments are generated as point samples with radii limited to one pixel, that is, they will not expand during the interpolation. Since the performance of the pull-push is independent of the number of samples, this imposes no overhead. An example of models represented by different primitives is shown in Figure 10.

Even more, hybrid models containing different types of primitives can be similarly rendered. This allows, for example, for a smooth transition between primitive types when working with multiresolution hierarchies. Figure 11 illustrates an example of an hybrid model.

6. Pyramid with Templates

In this section, a new solution based on the pyramid structure is describe to render smoother and more precise silhouettes. This improves on one of the weakest points of the pyramid point renderer: the discontinuity caused by merging ellipses during reconstruction. The problem is more evident at the silhouettes, because some ellipses are only partially rasterized when they are not fully propagated to the highest resolution level.

The only difference in the pull phase, is that each sample is only available in one specific level. This is determined as



(a)

(b)

Figure 9. (a) The original triangle model and (b) the line-based model.

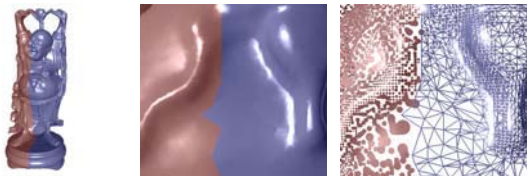
the first level that can cover the elliptical extent with a template of size $k \times k$ pixels. The correct level can be easily computed using the following equation:

$$l = \lceil log_2 \frac{D}{t_s} \rceil, \quad (1)$$

where D and t_s are, respectively, the projected radius and the template size in pixels. In the same manner as the PPR algorithm, ellipses are merged when needed. Note, however, that less merging occurs since the ellipses are not propagated to the coarsest level. Even more, the larger the kernel the less they propagate, and consequently, the less they merge.

During the reconstruction phase, analogous to the push phase, each pixel in the highest resolution level searches for projections in all coarsest levels. The search is carried out only in a template of size $k \times k$, and, for each projection found, the inside/outside test is performed. If the ellipse

Figure 10. Combined rendering of a point-based model (Dragon) and a triangle-based model (Buddha).



(a) (b) (c)

Figure 11. (a) A hybrid surface model with triangles (blue) and points (red). (b) Rendering of a detail and (c) illustration of the underlying primitives.

covers the pixel its contribution is added. The scheme is depicted in Figure 12. Note that this strategy does not create new ellipses during reconstruction, thus avoids discontinuities, i.e., even if ellipses are merged during the pull phase, the new ellipses are faithfully rasterized. In a last pass, deferred shading is also employed.

The template strategy depends on the kernel size k, which is a trade off between speed and quality. A 3^2 kernel achieves performance similar to the pyramid point renderer algorithm. However, since more ellipses merges more artifacts are noticeable. A 5^2 renders high quality results with performance dropping approximately 40%. Kernel sizes larger than 5^2 have not proven to increase the quality significantly. A detail of the Armadillo model rendered with different kernel sizes is shown in Figure 13, while a magnified view of the Neptune model can be seen in Figure 14.

Even with a higher complexity than the original algorithm, $O(n \log n)$ against $O(n)$, this new strategy proves to be a good alternative for high quality rendering. By rasterizing the ellipses in a more faithful manner, the results approach the Surface Splatting quality. In fact, if not for the merges during the pull phase, they can, in practice, be identical.

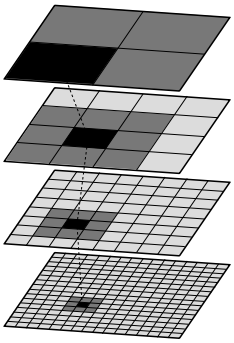


Figure 12. The search coverage (dark gray) for the projected pixel with a 3^2 search template. Painted black is the same pixel represented in different resolutions.

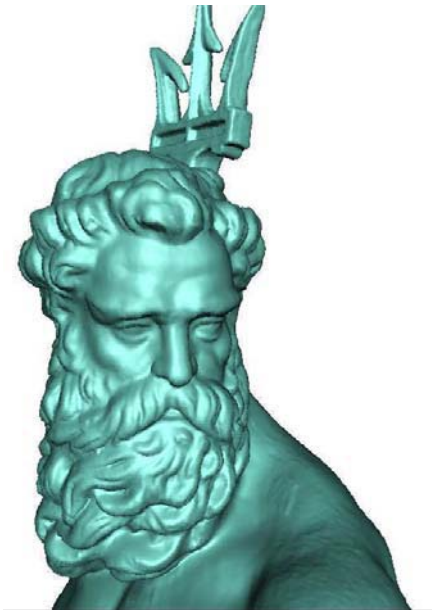


Figure 14. Magnified view of the Neptune model using a 5×5 kernel.

7. Level-Of-Detail for Point Based Models

Image reconstruction of surfaces using point representation proved to be a valuable alternative to traditional methods, mainly because its cost is independent of the number

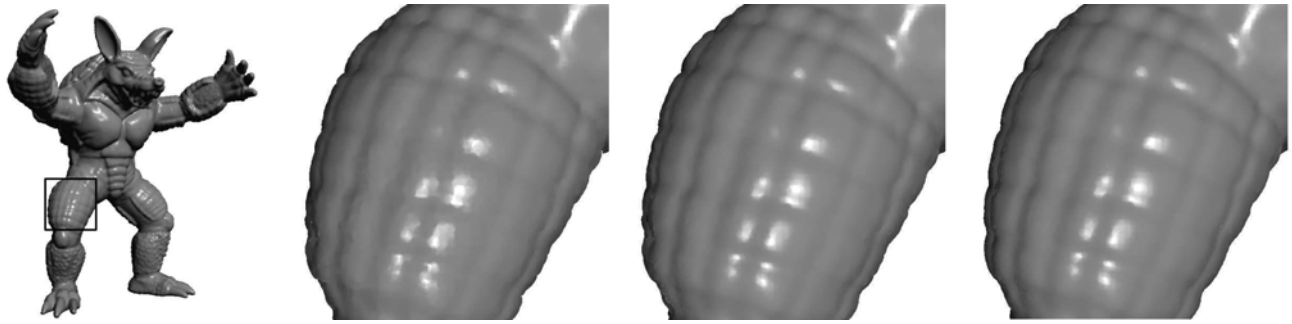


Figure 13. Same scene rendered with three different kernel sizes. From left to right, the kernel sizes are 3^2, 5^2, and 7^2. Note how there is little qualitative gain between the 5^2 and 7^2 kernels, even though the performance drops around 38%.

of projected samples. However, the projection phase is still $O(s)$, where s is the number of samples. Since the reconstruction is carried out in a few milliseconds, the projection time must be decreased in order to improve performance. One obvious way is to reduce the number of projected points. In this work, a Level-Of-Detail structure is employed to determine which points are projected given a viewpoint.

We introduce Object Texture to create our LOD structure. An Object Texture stores the object's primitives grouped in patches, where each patch is associated with a single primitive that is actually sent to the GPU. To retrieve the entire object during rendering, each patch contains information of where its first primitive is stored and how many it contains.

The different resolution levels are created in a simple manner by merging adjacent samples. Each sample in a coarsest resolution level is created by combining up to four adjacent samples from the immediate higher resolution level. The number of levels is limited to four in this application, however, this restriction can be easily extended. The three highest resolution levels are stored in an Object Texture and ordered from coarsest to finest inside each patch. The fourth level, with the coarsest resolution, contains the vertices sent to GPU and used to reference the patches, thus called *patch vertices*. These vertices are coarse resolution samples containing some extra information: the texture coordinates of the first vertex of the patch and the number of vertices in each level.

Each merged sample stores a perpendicular error precomputed as in *Sequential Trees*[4]. This error estimates the local surface smoothness prioritizing details along the silhouette, while, at the same time, taking into account the distance to the camera position. In the geometry shader, the error of the patch vertex (coarsest resolution) is projected and compared with a threshold to decide which level should be used. The chosen primitives are projected for rendering using the PPR algorithm, described in Section 3.

Since the geometry shader is still not fully optimized in its initial releases, no significant improvement in performance was noted. Yet, we expect that next generation GPUs shall improve them enhancing the contributions of our LOD structure. This deficiency is mainly attributed to a parameter that sets the maximum number of primitives to be outputed by the geometry shader. As this maximum limit increases, even if it is not reached in practice, performance drops significantly. The number of primitives sent to GPU was approximately one order of magnitude less than the model's samples.

8. Conclusions

With the growing complexity of scenes, new bearings are needed for efficiently processing, modelling and visualizing models. Point based representation is gaining an increasing attention due to the evolution of registration devices. Even though there exists methods for converting point clouds to polygonal meshes, there are several advantages on working directly with points for some applications.

This work presents an image reconstruction method for direct rendering of unstructured point clouds. In general, this approach is a good compromise between image quality and rendering performance for large models. One of its most significant advantages is that the reconstruction cost is independent of the number of projected samples. While the projection phase does not share this characteristic, the computation per sample is very small, thus interactivity is possible for millions of points. By extending the framework to include other primitives, such as lines and triangles, we also improve on the method's applicability.

Even though it is not possible to fully profit from the triangle specialized pipeline of graphics card, the simplic-

ity of points together with image based methods map extremely well to the GPU. This allows for efficient implementations and competitive performance.

The algorithm still lacks appropriate anti-aliasing techniques. However, with the quality improvement gained from the template kernels, much of the artifacts and flickering from the original implementation have been eliminated.

In all, image-space techniques still have some drawbacks, specially due to resampling problems. Nevertheless, the field continues to prove its value by achieving high performance rates while generating quality images.

9. Acknowledgments

We would like to acknowledge the grant of the first author provided by Brazilian agency CNPq (National Counsel of Technological and Scientific Development), and all published works co-authors: Martin Kraus, André Maximo, Antonio Oliveira and Claudio Esperança.

References

[1] M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. High-quality surface splatting on today's gpus. In *PBG '05: Proceedings of the Eurographics Symposium on Point-Based Graphics*, 2005.

[2] 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*, page 335, Washington, DC, USA, 2003. IEEE Computer Society.

[3] C.-F. Chang, G. Bishop, and A. Lastra. Ldi tree: a hierarchical representation for image-based rendering. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 291–298, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[4] C. Dachsbacher, C. Vogelgsang, and M. Stamminger. Sequential point trees. *ACM Trans. Graph.*, 22(3):657–662, 2003.

[5] 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, New York, NY, USA, 1996. ACM Press.

[6] M. Gross and H. Pfister, editors. *Point-Based Graphics*. Morgan Kaufmann Publishers, 2007.

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

[8] G. Guennebaud, L. Barthe, and M. Paulin. Splat/Mesh Blending, Perspective Rasterization and Transparency for Point-Based Rendering. In *IEEE/Eurographics/ACM Symposium on Point-Based Graphics, Boston, USA, 29/07/06-30/07/06*, pages 49–58, http://www.eg.org/, 2006. Eurographics.

[9] S. B. Kang, Y. Li, X. Tong, and H.-Y. Shum. Image-based rendering. *Foundations and Trends in Computer Graphics and Vision*, 2(3):173–258, 2006.

[10] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, New York, NY, USA, 1996. ACM.

[11] M. Levoy and T. Whitted. The use of points as a display primitive. Technical report, University of North Carolina at Chapel Hill, January 1985.

[12] 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.

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

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

[15] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 335–342. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[16] M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

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

[18] J. Shade, S. Gortler, L. wei He, and R. Szeliski. Layered depth images. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242, New York, NY, USA, 1998. ACM.

[19] M. Strengert, M. Kraus, and T. Ertl. Pyramid Methods in GPU-Based Image Processing. In *Workshop on Vision, Modelling, and Visualization VMV '06*, pages 169–176, 2006.

[20] 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.

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

[22] M. Zwicker. *Continuous Reconstruction, Rendering, and Editing of Point-Sampled Surfaces*. PhD thesis, Swiss Federal Institute of Technology, ETH, Zurich, 2003.

[23] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *Siggraph 2001, Computer graphics Proceedings*, pages 371–378, New York, NY, USA, 2001. ACM Press / ACM SIGGRAPH / Addison Wesley Longman.

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