Postgraduate Study Report DC-PSR-2003-03

Representing and Rendering Surfaces with Points

Jaroslav Křivánek

Supervisor: *Jiří Žára*

February 2003

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University
Karlovo nám. 13
121 35 Prague 2
Czech Republic

email: xkrivanj@fel.cvut.cz
WWW: http://www.cgg.cvut.cz/˜xkrivanj/

This report was prepared as a part of the project

Geometry Modeling with Point Sampling

This research has been supported by a grant of the Ministry of Education of the Czech Republic number 2159/2002.

. . . .
Jaroslav Křivánek
postgraduate student

. . . .
Jiří Žára
supervisor

Table of Contents

1 Introduction **1**
1.1 Report Scope . . . 2
1.2 Report Organization . . . 2

2 Chronological Overview of Point-Based Modeling and Rendering **3**

3 Point-Based Rendering **5**
3.1 Point Sample Attributes . . . 5
3.2 Point Rendering Pipeline . . . 6
3.3 Visibility and Image Reconstruction . . . 6
3.3.1 Ignoring Holes . . . 7
3.3.2 Separate Visibility and Image Reconstruction . . . 7
3.3.3 Splatting . . . 10
3.3.4 Up-sampling . . . 17
3.3.5 Discussion . . . 18
3.4 Visibility Culling . . . 20
3.4.1 View Frustum Culling . . . 20
3.4.2 Back-Face Culling . . . 20
3.4.3 Occlusion Culling . . . 20
3.4.4 Discussion . . . 21
3.5 Data Structures, Level-of-Detail Control and Compression . . . 21
3.5.1 Bounding Sphere Hierarchy . . . 21
3.5.2 LDC Tree . . . 22
3.5.3 Discussion . . . 23
3.6 Ray Tracing Point Sampled Surface . . . 23
3.6.1 Discussion . . . 24
3.7 Object Sampling . . . 24
3.7.1 Classification . . . 24
3.7.2 Overview . . . 25
3.7.3 Discussion . . . 26

4 Conclusions and Future Research **27**
4.1 Advantages of Points 27
4.1.1 . . . as Rendering Primitives . . . 27
4.1.2 . . . as Modeling Primitives . . . 27
4.2 Disadvantages of Points . . . 28
4.3 Conclusion . . . 28
4.4 Future Work . . . 28

5 Our Contribution: Depth-of-Field Rendering with EWA Surface Splatting **29**
5.1 Outline of the Algorithm . . . 30
5.2 Preliminaries . . . 30
5.2.1 Camera Model for DOF Rendering . . . 30
5.3 DOF Rendering in the EWA Surface Splatting Framework . . . 31
5.3.1 DOF rendering as a resampling kernel convolution . . . 31
5.3.2 Variance matrix of the Intensity Distribution Function . . . 32
5.3.3 DOF Rendering with Level-of-Detail . . . 33
5.4 Implementation . . . 34
5.4.1 DOF rendering without LOD . . . 34

5.4.2 DOF rendering with LOD 34
5.4.3 Normalization 36
5.4.4 Surface Reconstruction 36
5.4.5 Shading 37
5.5 Results 37
5.6 Discussion 37
5.7 Conclusions and Future Work 39

6 References 40

7 Dissertation thesis 44

8 Publications of the author 44

A List of Abbreviations 45

Representing and Rendering Surfaces with Points

Jaroslav Křivánek
`xkrivanj@fel.cvut.cz`
Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University
Karlovo nám. 13
121 35 Prague 2
Czech Republic

Abstract

This report deals with the use of points as surface rendering and modeling primitives. The main components of point-based rendering and modeling algorithms are identified, different approaches are discussed and compared. The weaknesses of current point-based techniques are pointed out and for some of them a possible solution is suggested. A new algorithm for depth-of-field rendering based on surface splatting is presented. It features rendering time independent of the amount of depth-blur and depth-of-field rendering in scenes with semi-transparent surfaces. For this algorithm a mathematical analysis, an implementation and a discussion of results are given.

Keywords

real-time rendering, point-based modeling and rendering, point sample rendering, level-of-detail, surface representation and reconstruction, surface sampling, splatting, alias, antialiasing, filtering.

1 Introduction

In this report, we discuss point-based techniques for surface modeling and rendering in computer graphics. Those techniques work with geometric objects represented by a set of point samples of objects' surface, without any connectivity or topology information.

The objective of *point-based rendering* is to display the point representation on the screen as continuous surfaces (Figure 1). The motivation for point-based rendering includes its efficiency at rendering very complex objects and environments, simplicity of rendering algorithms and emergence of 3D scanning devices that produce very dense point clouds that need to be directly visualized.

The term *point-based modeling* is used in this report to denote techniques for point-based surface representation, processing and editing. Point-based modeling techniques are motivated by the need to have a complete geometry processing pipeline from acquisition (3D scanning) to the visualization (point-based rendering). Point-based surface representations allow working with point clouds as if they were surface: we can ray-trace them, perform smoothing or any other kind of surface modification. A big advantage of points for modeling purposes is the ease of object topology modification.

A part of this report is devoted to existing point-based techniques with emphasis on point-based rendering. We identify the main components of point-based rendering algorithms and we discuss and compare different approaches. We attempt to point out the weaknesses of current point-based techniques and we propose possible improvements for some of them. The topics are not covered evenly, in particular a lot of space is reserved to EWA surface splatting, since our contribution builds on top of this technique.

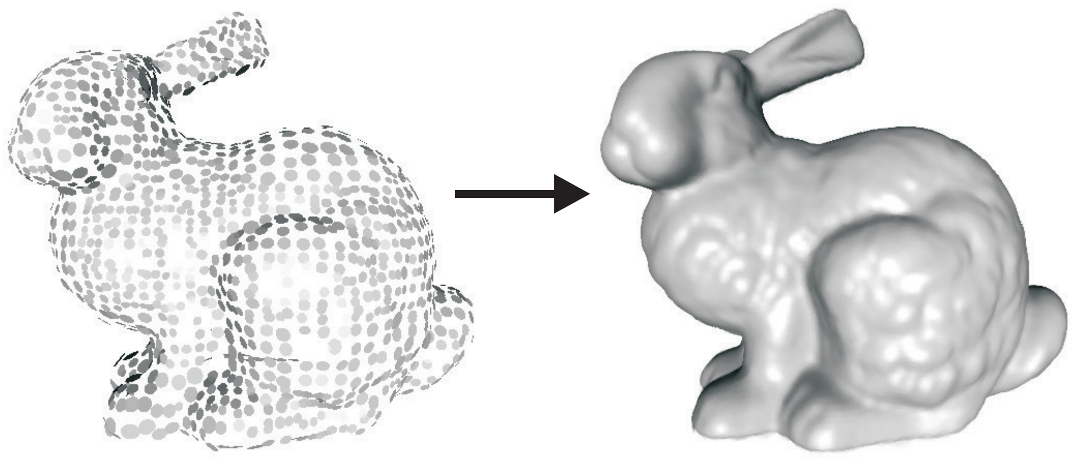

Figure 1: Point-based representation of a bunny. Left: Explicit structure of points is visible. Right: continuous surface was reconstructed with EWA surface splatting.

Another part of the report describes our contribution to the field of point-based rendering. We describe a new algorithm for depth-of-field rendering based on surface splatting. It features rendering time independent of the amount of depth-blur and depth-of-field rendering in scenes with semi-transparent surfaces. For this algorithm a mathematical analysis, an implementation and a discussion of results are given.

1.1 Report Scope

The scope of this report is mainly point-based rendering, some topics from point-based modeling are covered as well. This includes: visibility, image reconstruction, data structures for point samples, level-of-detail control, compression, object sampling.

Very related problems that we don't discuss here are *acquisition* of real-world objects [24, 36, 37], *conversion* of point-based surfaces to other representations (often denoted as surface *reconstruction* — [28, 5, 11, 52], see [3] for an overview), volume rendering [68, 66], and image-based rendering [64].

1.2 Report Organization

The report consists of three main parts. The first part is composed of Chapters 2 to 4 and gives an overview of existing approaches. Chapter 2 gives a brief chronological overview of point-based techniques. Chapter 3 discusses point-based rendering and covers also some topics from point-based modeling. Chapter 4 briefly reviews the main advantages and disadvantages of using points, it concludes the first part of the report and points out some of the unsolved problems of point-based modeling and rendering. The second part (Chapter 5) describes our contribution, a point-based algorithm for depth-of-field rendering. Chapter 6 is the list of references. The third part (Chapter 7) gives the abstract of author's intended dissertation thesis.

It is common for the surveys like this that a classification of the problems is proposed, or the problems are described in an unified framework. We did not pursue this way for more reasons of which the most important is that point-based graphics is at its beginnings and it is hardly predictable what is going to happen. It is highly likely that the classification would not survive even the year to come.

2 Chronological Overview of Point-Based Modeling and Rendering

The idea to use points as *surface* rendering primitives dates from 1985, when Levoy and Whitted [32] proposed points as universal rendering meta-primitives. They argued that for complex objects the coherence (and therefore the efficiency) of scanline rendering is lost and that points are simple yet powerful enough to model any kind of object. The conceptual idea was to have a unique rendering algorithm (that renders points) and to convert any object to point representation before rendering. This would obviate the need to write a specialized rendering algorithm for every graphics primitive. Very similar idea was used in the Reyes architecture [16] (dicing into micropolygons). Levoy and Whitted used splatting with circular splats to render points, they limited the minimum splat size to a pixel size to avoid aliasing. They used accumulated weights for coverage estimation and an a-buffer [10] for visibility and composition in the same way as Zwicker et al. [67] 16 years later.

In 1992 Szeliski and Tonneson [56] used oriented particles for surface modeling and interactive editing. The oriented particles were points with local coordinate frame, interacting with each other by long-range repulsion forces and short-range attraction forces. For visualization of oriented particles, however, they did not use point-based rendering. They visualized the primitives with ellipses and for final production they constructed a surface triangulation. They considered the oriented particles as surface elements, “surfels”.

In 1994 Witkin and Heckbert [61] used oriented particles for sampling and interactive editing of implicit surfaces. The particles were visualized in the same way as in [56].

In 1998 point rendering was revisited by Grossman and Dally [25, 26]. This time the aim was to develop an output sensitive rendering algorithm for complex objects that would support dynamic lighting. The work was mainly inspired by advances in image based rendering.

A real boom of point-based rendering started in 2000 with Surfels of Pfister et al. [43] and QSplat of Rusinkiewicz and Levoy [49]. A new motivations for developing point-based techniques were (and still are) advances in scanning technologies and rapidly growing complexity of geometric objects.

Pfister et al. [43] extended the work of Grossman and Dally with a hierarchical LOD control and hierarchical visibility culling, they developed a better technique to resolve visibility, the visibility splatting, and they paid a lot of attention to antialiasing. The resulting system was still similar to image based-rendering. A survey of techniques that led to the development of Surfels is given in [64].

Rusinkiewicz and Levoy [49], on the other hand, devised a brand new data structure for hierarchical LOD and culling and they used splatting for surface reconstruction. They receded from sampling the objects on regular lattice, therefore they left the influence of image-based rendering. Their aim was to interactively display massive meshes, with rendering quality adapting to the power of the computer used for rendering.

Tobor et al. [57] also proposed a point-based rendering system, but it neither gave good image quality neither it was shown to be fast. Schaufler and Wann Jensen [51] proposed a method for ray tracing surfaces defined by a point cloud.

Year 2001 brought the EWA surface splatting [67], dynamic (on-the-fly) object sampling during rendering [58, 55], hybrid polygon-point rendering systems [12, 15], differential points [29], spectral processing of point sampled surfaces [40], multiresolution point-based modeling [33], and the MLS-surfaces [2].

EWA surface splatting of Zwicker et al. [67] combines the ideas of Levoy and Whitted [32] with Heckbert’s resampling framework [27] to produce a high quality splatting technique that features anisotropic texture filtering, edge antialiasing and order independent transparency. Zwicker et al. also extended the EWA splatting to volume rendering [66].

Hybrid polygon-point rendering systems [12, 15] definitely leave the idea of points as an universal rendering primitive. They build on the observation that points are more efficient only if they project to a small screen space area, otherwise polygons perform better.

Methods for dynamic (on-the-fly) object sampling produce point samples of rendered geometry in a view dependent fashion as they are needed for rendering (i.e. during rendering, not as a preprocess). Wand et al. (the *randomized z-buffer*) [58] used randomized sampling of a triangle set to interactively display scenes consisting of up to 10^{14} triangles. Stamminger and Drettakis [55] used deterministic sampling pattern to dynamically sample complex and procedural objects. They also presented a randomized LOD technique which is extremely useful for hardware accelerated point rendering.

Kalaiah and Varshney's differential points [29] extend points with local differential surface properties to better model their neighborhood. The differential points are sampled from NURBS surfaces, but nothing prevents them from being sampled from any king of differentiable surface. The initial point set is then reduced by eliminating redundant points.

Pauly and Gross [40] presented a framework for applying Fourier transform and spectral techniques to point sampled surfaces. In order to get the planar supports for the geometry (to be able to apply the FT) they decompose the surface into small patches defined over planar domains. The spectral techniques (i.e. filtering) are then applied to each patch separately and in the end the patches are blended together to get the result of the filtering operation.

Linsen [33] proposed a multiresolution modeling framework for point sampled surfaces. He developed up- and down-sampling operators, surface smoothing and multiresolution decomposition. He also applied CSG operations to point sampled surfaces.

Alexa et al. [2] proposed a point-based surface definition that builds on fitting a local polynomial approximation to the point set using moving least squares (MLS). The result of the MLS-fitting is a smooth, 2-manifold surface for any point set. They used the MLS-surfaces to up- and down-sample the point set and to dynamically (on-the-fly) up-sample the point set during rendering to obtain high quality smooth surface.

Rusinkiewicz and Levoy [50] presented an extended version of QSplat capable of streaming geometry over intermediate speed networks. Luebke and Hallen [34] used perceptual metrics for LOD selection in point-based rendering. Wimmer et al. [60] used points to render far field in urban visualization.

In 2002 the focus was mainly on efficient implementation of existing point rendering algorithms (mainly EWA splatting), point-based modeling and on applications of point-based rendering.

Botsch et al. [9] proposed a space-efficient hierarchical representation of point sampled objects. They use hierarchical warping and quantized shading for very fast splatting in software. For EWA splatting, they quantize the splat parameters and precompute a number of splat shapes. Coconu and Hege [14] use hardware to accelerate EWA splatting. They render splats as point sprite primitives of DirectX 8 and map an alpha mask onto them. For correct blending between splats, they draw splats in an occlusion compatible order. Ren et al. [46] proposed a different method for hardware accelerated EWA splatting. They render splats as quads with an alpha mask and they use two-pass rendering to resolve visibility and blend splats. Wand and Straßer [59] presented a hybrid point-polygon rendering technique for keyframe animations. Deussen et al. [17] used the ideas from Stammingers's paper [55] to render large ecosystems at interactive frame rates. Räsänen [45] discusses the EWA surface splatting and proposes some improvements to the algorithm. He uses stochastic sampling for high quality EWA splatting. Dey and Hudson [18] presented another hybrid polygon-point rendering system based on Voronoi surface reconstruction techniques (see [5]). Their system differs from other hybrid polygon-point rendering systems in that it uses local geometry feature size to select appropriate primitive.

An application of point-based rendering is a 3D video system of Würmlin et al. [62]. They also proposed an efficient compression technique for point-based objects. Another application are the probabilistic surfaces of Grigoryan and Rheingans [23]. They use a random displacement of a point along surface normal to visualize uncertainty in surface definition.

Pauly et al. [41] compared different simplification methods for point sampled surfaces inspired by the methods developed for triangular meshes. They also proposed a technique for measuring error on simplified surfaces based on Alexa's MLS-surfaces. Pauly et al. used the techniques presented in [41] as a part of their multiresolution modeling framework [42]. The framework included the same features as that of Linsen, but added an intuitive editing metaphor and dynamic resampling assuring sufficiently high sampling rate during editing.

Pointshop 3D of Zwicker et al. [65] is an editing system that allows to work with point-based surface as with images. Except from all possible effects used in conventional image editors, they could perform altering of surface geometry (carving, sculpting, filtering, ...).

Fleischman et al. [20] used MLS-surfaces as a base for their multiresolution point sampled surface representation. They focused mainly on space-efficient progressive encoding of point sets than on multiresolution editing. Kalaiah and Varnshey [30] extended their paper [29] and gave more details on surface representation with differential points and on point set reduction. Zwicker et al. [68] presented an unified framework for EWA surface and volume splatting.

The Eurographics '02 tutorial on point-based graphics [24] covered these topics: point-based rendering by splatting including EWA splatting, image-based acquisition of geometry and appearance, dynamic sampling and real-time ecosystem visualization, point-based surface representations (MLS-surfaces), spectral processing of point sampled geometry, surface simplification and the Pointshop 3D system.

The following papers that are going to be published in 2003 are known to us. Alexa et al. [3] revisits and extends the paper on MLS-surfaces [2]. Adamson and Alexa [1] ray traces point sampled surfaces using the MLS-projection operation. Alexa et al. [4] establishes a direction field on the point sampled surface, which can be used for surface parametrization, texture synthesis or mapping between surfaces.

3 Point-Based Rendering

Point-based rendering can be considered as *surface reconstruction* from its point samples. That is to say, given a point set, the rendering algorithm draws a smooth surface represented by the point set. This reconstruction is view dependent and has to be repeated for every viewpoint.

Various approaches exist for rendering objects directly from point sets. Most of the point rendering algorithms proceed in *object order*: the point primitives are sent to the rendering pipeline and contribute to the final image. The exception are algorithms for ray tracing point sampled surface. They are discussed separately in Section 3.6. Here we discuss not only the rendering itself, i.e. the process of forming the image, but also the supporting techniques of point-based rendering systems: culling, data structures, level-of-detail control. A separate section is devoted to object sampling techniques (Section 3.7). Before explaining the rendering algorithms, let us have a closer look on the attributes of points.

3.1 Point Sample Attributes

The attributes are summarized in Figure 2. Each point sample has a position, surface normal and shading attributes (color). If we assign an *area* to the point sample, it becomes a surface element — *surfel* [43]. A surfel represents a piece of surface rather than being a simple point sample. We do not distinguish "surfels" from "points" and we use the two terms interchangeably. However, the term "surfel" is mostly used in point-based rendering, whereas "point" is more often used in point-based modeling. The surfel's area can be expressed e.g. by a radius, assuming that the surfel represents a circular area in object space (Figure 7). The surfel areas must fully cover the surface to assure hole-free reconstruction.

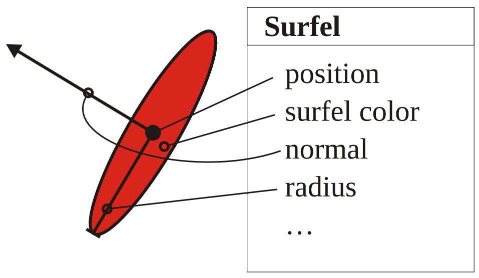

Figure 2: Surfel attributes.

The oriented particles of Szeliski et al. [56] store the local reference frame with each point (i.e. normal + orientation in tangent plane). This can be exploited for modeling purposes and for anisotropic shading.

Kalaiah and Varshney's differential points [30, 29] store additional per-point attributes that describe local differential surface properties (Figure 3): the principal directions (directions of minimum and maximum curvatures) $\hat{\mathbf{u}}_\mathbf{p}$ and $\hat{\mathbf{v}}_\mathbf{p}$, and the principal curvatures (the normal curvatures in the principal directions). The differential properties allow to model the sampled surface better using a significantly lower number of samples, however this technique is suitable only for smooth surfaces.

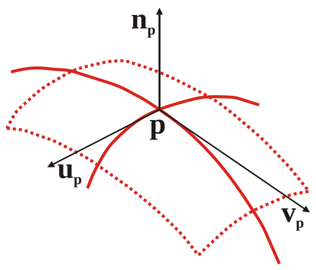

Figure 3: Differential point.

3.2 Point Rendering Pipeline

Figure 4 shows a point rendering pipeline. It accepts points on its input and produces images as output. It is the core of any point-based rendering system. The pipeline is very simple, purely forward mapping, there is high a data locality (each point carries all its information), no texture look-ups. Therefore the point rendering is very efficient and amenable to hardware implementation.

The *warping* stage projects each point to screen space using perspective projection. This can be done by homogeneous matrix-vector product or, more efficiently, by incremental [25, 26, 43] or hierarchical warping [9]. In the *shading* stage, per-point shading is performed, any local shading model is applicable. Shading is often performed after visibility (only visible points are shaded) or after image reconstruction using interpolated point attributes (per-pixel shading). The next two stages, *visibility* and *image reconstruction*, together form a view dependent *surface reconstruction* in screen space. They can be either separated or simultaneous. The surface reconstruction optionally includes filtering to prevent aliasing.

3.3 Visibility and Image Reconstruction

The ultimate goal of visibility and image reconstruction is to display a point set as a smooth surface without holes and without aliasing artifacts (i.e. reconstruct the surface in screen space). Holes are the main problem. If we rendered each point as a single pixel, some pixels wouldn't get any point (they

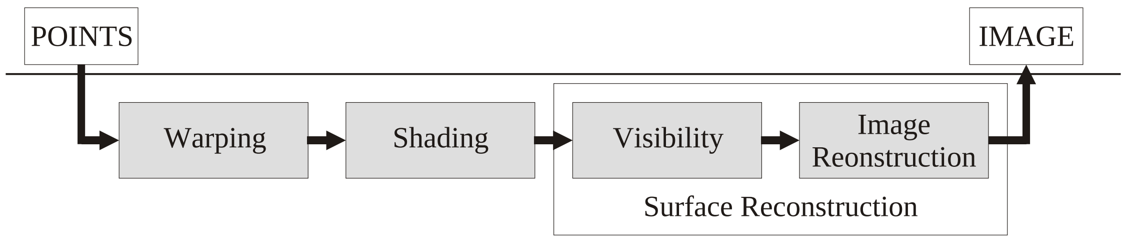

Figure 4: Point rendering pipeline.

would be "holes"), other pixels could get an occluded background point (and no point from foreground surface). The second problem (visible background points) is harder to solve: background points have to be identified even if there is no foreground point that covers them. Therefore, simple z-buffering does not always help.

Approach to reconstruction	**Comment**
Ignore	suitable for unstructured geometry
Hole detection + image reconstr.	separate visibility and image reconstruction
Splatting	simultaneous visibility and image reconstruction
Up-sampling	form of splatting

Table 1: Surface reconstruction techniques in point-based rendering.

Table 1 gives an overview of different reconstruction techniques, each of which is discussed in a separate section later. All reconstruction techniques share one property: they *need to know the density* of point samples in order to work. Table 2 gives a list of papers that deal with point-based rendering along with the reconstruction technique they propose or use.

3.3.1 Ignoring Holes

It is possible to ignore the problem of holes and background points altogether [38]. This is suitable for unstructured geometry, such as trees, where one missing leaf doesn't play a big role. However, some minimum screen-space density of points has to be assured anyway (for example by not allowing to magnify the object).

3.3.2 Separate Visibility and Image Reconstruction

Those techniques involve two steps: the first step identifies background points, deletes them from the framebuffer, and marks their pixels as holes. In the second step, the resulting framebuffer, containing only foreground pixels and holes, is subject to an image reconstruction (basically scattered data approximation). The techniques differ in the way they detect the background points and the way they do the image reconstruction. A general structure of the reconstruction algorithm that separates visibility from image reconstruction is in Figure 5.

Grossman and Dally: Point Sample Rendering

Grossman and Dally [25, 26] use hierarchical z-buffer for hole detection and Gortler's scattered data approximation algorithm [22] for image reconstruction.

Hole detection The idea of hole detection is to have a hierarchy of z-buffers at different resolutions, the highest resolution z-buffer matching the resolution of the frame buffer. Each z-buffer has half the resolution of its predecessor (Figure 6).

Paper	**Reconstruction technique**
Levoy and Whitted 1985, [32]	Gaussian splatting
Max 1995 , [38]	ignore
Grossman and Dally 1998, [25, 26]	hole detection + image reconstruction
Pfister et al. 2000, [43]	hole detection + image reconstruction
Rusinkiewicz and Levoy 2000, [49]	quad splatting, Gaussian splatting
Tobor et al. 2000, [57]	quad splatting
Schaufler and Wann Jensen, [51]	ray tracing
Alexa et al. 2001, [2]	up-sampling(+ quad splatting)
Chen and Nguyen 2001, [12]	quad splatting, surfels
Cohen et al. 2001, [15]	quad splatting
Kalaiah and Varshney 2001, [30, 29]	differential point splatting
Linsen 2001, [33]	triangle fans
Luebke 2001, [34]	quad splatting, Gaussian splatting
Stamminger and Drettakis 2001, [55]	quad splatting
Wand et al. 2001, [58]	quad splatting, surfels
Wimmer et al. 2001, [60]	quad splatting
Zwicker et al. 2001, [67]	EWA splatting
Botsch et al. 2002, [9]	quad splatting, approximate EWA splatting
Coconu and Hege 2002, [14]	quad splatting, approximate EWA splatting (HW)
Deussen et al. 2002, [17]	quad splatting
Dey and Hudson 2002, [18]	quad splatting
Grigoryan and Rheingans 2002, [23]	quad splatting
Räsänen 2002, [45]	EWA splatting
Ren et al. 2002, [46]	EWA splatting (HW)
Wand and Straßer 2002, [59]	quad splatting
Würmlin et al. 2002, [62]	quad splatting, EWA splatting
Zwicker et al. 2002 (Pointshop 3D), [65]	quad splatting, EWA splatting
Adamson 2003, [1]	ray tracing

Table 2: Use of different surface reconstruction techniques in point-based rendering.

For each point warped to screen space first a z-test is done at the highest resolution z-buffer. If it fails, the point is discarded. If it passes, *another z-test* is done on the z-buffer resolution that assures hole-free z-buffering (i.e. the more the object is magnified, the lower resolution z-buffer is used, since the point density is lower). If the second z-test passes as well, the point color is written to the color buffer. Note that information about point density is required to choose an appropriate lower resolution z-buffer.

This basic technique produces blocky artifacts, therefore a more intricate technique is actually used in the implementation. It assigns weights to the pixels expressing the confidence that they lie in foreground. Those weights are then used for better color interpolation in the image reconstruction step.

Image reconstruction Gortler's "pull-push" scattered data approximation algorithm [22] is used for image reconstruction. The aim is to fill the pixels marked as holes using averaged colors of their neighbors. The *pull* phase generates lower resolution approximations of the image and the *push* phase fills the holes by blending approximations at different resolutions (see also [40]). Note that the lower resolution images for image reconstruction have nothing in common with the lower resolution z-buffers used for hole detection, the techniques are independent.

Pfister et al.: Surfels

Pfister et al. [43] use *visibility splatting* for hole detection and Gaussian filtering for image reconstruction. Even though different at first sight, the visibility splatting is very similar to Grossman's hierarchical z-buffer.

```
Separate() {
  for each surfel {
    project surfel to screen space
    solve visibility for projected surfel's neighborhood
    if (surfel visible)
      store surfel in z-buffer
  }
  for every visible surfel {
   shade surfel
  }
  reconstruct image (=fill holes)
 }
```

Figure 5: General structure of a reconstruction algorithm that separates visibility from image reconstruction.

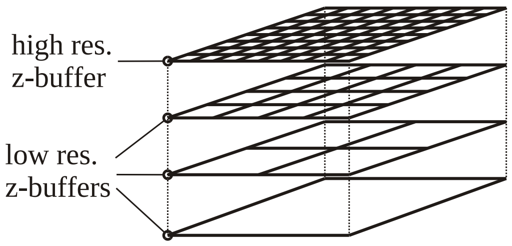

Figure 6: Hierarchical z-buffer. Each z-buffer has half the resolution of its predecessor

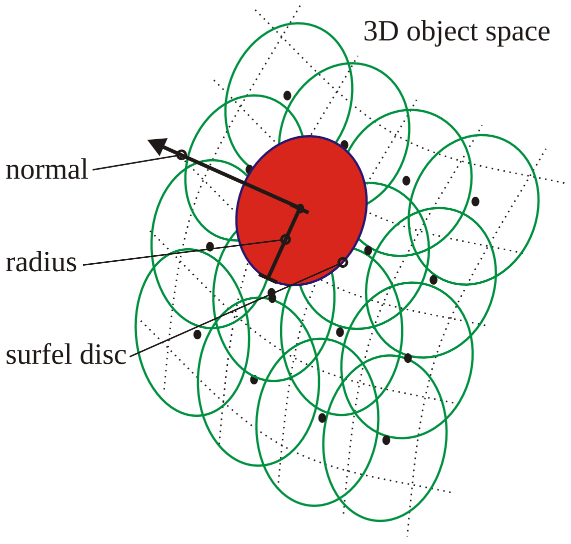

Figure 7: Surfel areas fully cover the surface.

Hole detection The visibility splatting assumes that each point represents a small circular area in object space (Figure 7). This area projects to screen space as a shape similar to an ellipse (Figure 8). Visibility splatting scan converts this ellipse to the z-buffer, with the depth computed from the tangent plane at the point. For each fragment coming from ellipse rasterization, z-test is performed. If it passes, the pixel is marked as a hole (and potential point in this pixel is deleted). If it fails nothing happens. The new point is eventually written *only to the center* of the ellipse (if the z-test at a pixel in the ellipse center passes).

This procedure yields a framebuffer without background points, because they were deleted in the course of rasterization of the ellipses from the foreground points. There are only foreground points and holes in the framebuffer. The density of the point set is needed to set up the size of the rasterized ellipses.

The comparison of visibility splatting to Grossman's hierarchical z-buffering reveals that both are similar in spirit: the points are written to the z-buffer enlarged to the size that assures no holes. Visibility splatting does it explicitly by rasterizing the projected surfel disc, Grossman uses a lower-resolution z-buffer to effectively grow the projected surfels, which is equivalent to visibility splatting constrained to squares (pixels of the lower resolution z-buffer).

Image reconstruction Image reconstruction is done by centering a Gaussian filter in the center of each hole and computing the weighted average of foreground point colors. To estimate the radius of the filter, the density of the point set is needed.

For edge-antialiasing Pfister et al. use supersampling. The z-buffer has a higher resolution than the color-buffer, each pixel of the color buffer corresponds to multiple z-buffer pixels. The reconstruction is the same: a Gaussian filter is centered at the pixel center. More surfels contribute to pixel color.

3.3.3 Splatting

Another reconstruction technique is splatting. It differs from the approaches of Grossman and Dally, and and Pfister et al. in that the visibility and image reconstruction are done simultaneously. Splatting assumes that a surface area is assigned to each surfel in object space (Figure 7) and it reconstructs the continuous surface by drawing an approximation of the projection of the surfel (which is similar to an ellipse in the case of circular surfels — see Figure 8). The *projected* surfel shape is called a splat. For

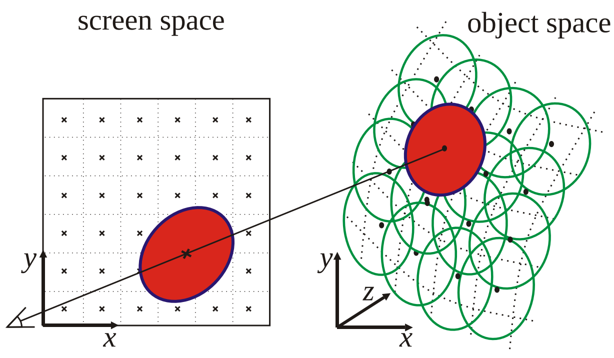

Figure 8: Circular surfels project to an ellipse-like shape in image space.

splatting algorithms the point density is often specified as a per-surfel radius r which is set to a value which assures that the surfels cover the surface completely. Pfister et al. used visibility splatting only to detect and eliminate background points and the image reconstruction was done in a different way. Here, splatting is used for solving both visibility and image reconstruction. The general structure of a splatting algorithm is given in Figure 9. Postprocessing (normalization) is done only for splatting with fuzzy splats.

Splat Shape

Flat Quads The roughest approximation to the projected surfel shape are flat quads. They are very fast to draw and are supported by hardware (un-antialiased OpenGL `GL_POINT`). The splat shape doesn't adapt to surface orientation which leads to the thickening of object silhouettes [49]. The image quality is low. Visibility is solved using a conventional z-buffer. The primitive depth is constant, equal to the z-coordinate of the projected point.

```
Splatting() {
  for each surfel {
    project surfel to screen space
    shade surfel
    compute splat parameters
    draw splat
  }
  for every pixel {
   do postprocessing (shading, normalization)
  }
}
```

Figure 9: General structure of a splatting algorithm.

Ellipses Ellipses match the real splat shape very closely, they adapt to surface orientation, there is no thickening near silhouettes. Ellipses provide a higher image quality than quads. Moreover, the depth of the primitive is not constant but inferred from the projection of the surfel's tangent plane. This leads to a better visibility resolution. As pointed out by Rusinkiewicz and Levoy [49], ellipses can leave holes near silhouettes. This can be solved by restricting the ratio between the ellipse's major and minor radius by a maximum value. Ellipses are slower to draw than quads and they are not directly supported in the hardware.

Splatting with Fuzzy Splats

The common drawback of quads, circles and ellipses as described so far is that there is *no interpolation* of color between splats. This leads to artifacts and reveals the splat shape to the observer. A better choice is to use fuzzy splats with an alpha mask (alpha falling off the center) and to blend the splat colors together in screen space according to the alpha (Figure 10). A color c of a pixel at position (x, y)

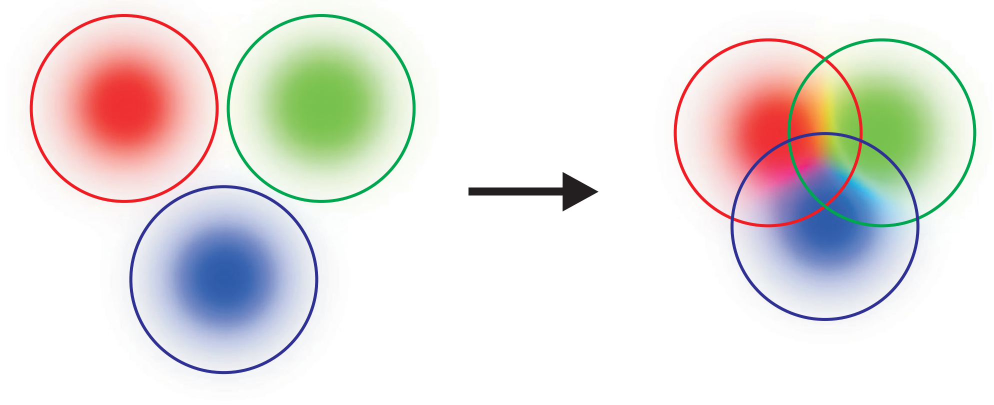

Figure 10: Fuzzy splats are blended to produce the final pixel colors.

is given by a normalized weighted sum of contributions from different splats

$$c(x,y) = \frac{\sum_i c_i w_i(x,y)}{\sum_i w_i(x,y)}, \tag{1}$$

where c_i is the color of splat i and $w_i(x, y)$ is the weight of splat i at position (x, y). The normalization has to be done, since the splat weights do not sum up to one everywhere in screen space (Figure 11). This is caused by irregular surfel positions and by the truncation of the ideal alpha masks.

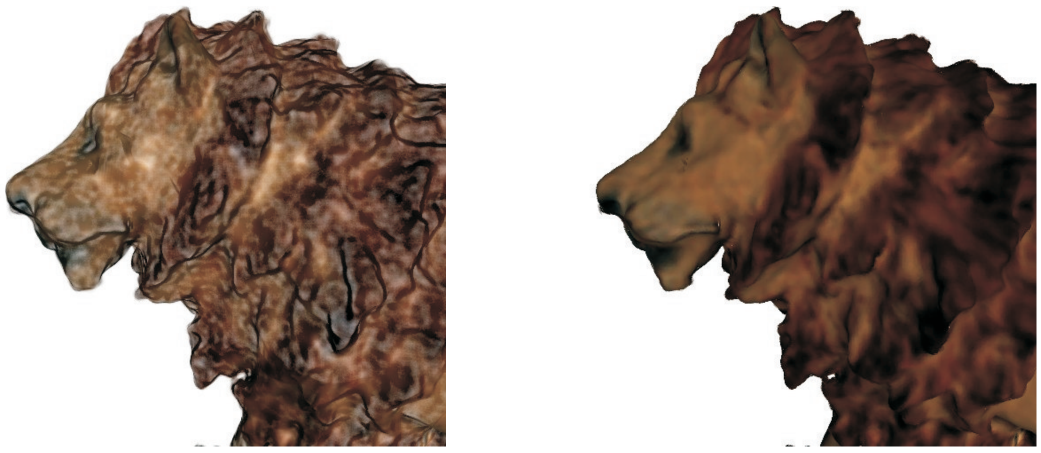

Figure 11: Left: Leaving out the normalization of splat contributions leads to changing brightness of the image. Right: The same image with per-pixel normalization.

Per-Surfel Normalization

With hardware rendering, it is not possible to perform the per-pixel normalization, or this operation is very slow and should be avoided. Ren et al. [46] addressed this problem by performing a pre-normalization of surfel weights in a way that forces the resulting surfels to sum up approximately to 1 after projecting to screen-space. They assign to each surfel a normalization factor that make the weights sum up approximately to unity in screen space. The normalization factors for each splat are computed in preprocess:

in the first pass the object is rendered without normalization and the framebuffer is saved (particularly the weights accumulated for each pixel). In the second pass the surfels are warped to screen space and for each warped surfel the accumulated weight is read from the saved framebuffer (at surfel's position) and the reciprocal of the accumulated weight is assigned to the surfel as a normalization factor. To assign the normalization factor to each surfel, the procedure is repeated from a number of viewpoints on the object's bounding sphere.

This technique is very easy to implement if one has a fuzzy splatter at hand. It gives reasonable results (in a practical implementation, the summed weights with the normalization factor fall to range 1 ± 0.1). But the problem of pre-normalization belongs rather to the field of point-set processing than point-based rendering and it would certainly be worth developing a dedicated algorithm for this purpose.

Extended z-buffering

The conventional z-buffering provides for each pixel an answer *pass* (=new fragment is in front, visible) or *fail* (new fragment is at the back, invisible). For blending fuzzy splats this binary decision has to be extended with a third answer: *blend*. This answer means that the new fragment and the fragment in the z-buffer come from the same surface and they should be blended together, since both of them will contribute to the sum in Equation (1). As the neighboring splats can be oriented rather arbitrarily, the solution to this problem always involves a bit of guessing.

Zwicker et al. [67] use a threshold (or epsilon) in the z-test: if the z-coordinate of the two fragments differ by less than the threshold, they are blended, othervise the conventional binary z-test is applied. The depth test preudocode is given in Figure 12. Main drawbacks of this approach are that it involves a user definable parameter (the threshold) and even more importantly, it fails near silhouettes. The first problem is solved by setting the threshold equal to the splat radius. The second problem can be alle-

```
DepthTestZOffset(x,y) {
 if ( fragment z < z(x,y) - epsilon ) {
   // we are in front: pass
   z(x,y)      = fragment z
   color(x,y)  = fragment color
   weight(x,y) = fragment weigh
  }
 else if ( fragment z < z(x,y) + epsilon ) {
   // we are approximately at the same depth: blend
   color(x,y)  = color(x,y)  + fragment color
   weight(x,y) = weight(x,y) + fragment weigh
  }
 // else we are at the back: fail (do nothing)
}
```

Figure 12: Extended z-buffering with z-offset.

viated by dividing the threshold by a cosine of the angle between the viewing ray and the splat normal (enlarging the threshold for silhouette splats), however the problem cannot be completely avoided.

Another approach to extended z-buffering is based on *z-ranges* [32, 45]. A z-range is computed for each splat and is assigned to every fragment that comes from the rasterization of that splat. If the z-ranges of the new fragment and the fragment in the z-buffer overlap in the depth test, blending occurs and the new range is set to union of the two ranges (Figure 13).

```
DepthTestZRange(x,y) {
 if ( fragment zmax < zmin(x,y) ) {
   // we are in front: pass
   zmin(x,y)   = fragment zmin
   zmax(x,y)   = fragment zmax
   color(x,y)  = fragment color
   weight(x,y) = fragment weigh
  }
 else if ( fragment zmin < zmax(x,y) ) {
   // the ranges overlap: blend
   zmin(x,y) = min ( zmin(x,y), fragment zmin)
   zmax(x,y) = max ( zmax(x,y), fragment zmax)
   color(x,y)  = color(x,y)  + fragment color
   weight(x,y) = weight(x,y) + fragment weigh
  }
 // else we are at the back: fail (do nothing)
}
```

Figure 13: Extended z-buffering with z-ranges.

The range for a splat is computed as a minimum and maximum z-coordinate of splat's surfel in camera space. The formulas are given in [45]. A more conservative range can be computed more quickly as $(z - r, z + r)$, where z is camera space z-coordinate of the surfel center and r is the surfel radius.

It is worth noting that the extended z-buffering has to use camera space z values, i.e. the z before the perspective division, since in post-perspective space it wouldn't be easily feasible to decide whether two fragments are closer to each other than a threshold. The advantage of z-ranges over z-offsets is that z-ranges are more robust along silhouettes and completely avoid depth interpolation — only one z-range is computed per-splat. None of the methods is supported by hardware, however the z-offsetting can be simulated by a two-pass rendering algorithm as suggested by Rusinkiewicz and Levoy [49] and more deeply elaborated by Ren et al. [46].

Aliasing in Splatting

If the surfels project to less than a pixel, it may easily happen that the projected surfel shape falls completely out of the pixel grid, i.e. no pixel center lies within the projected surfel. This can produce artifacts, for example holes.

The simplest solution is found for quad splats. Their center is rounded to the nearest pixel center, and their size is limited to minimum of one pixel. This is automatically done by OpenGL, if we set `glPointSize` to 1.

The solution that is used for quad splats can, however, not be applied to fuzzy splats. Rounding the splat positions to pixel centers would lead to distortions in texture reproduction since the relative positions of neighboring splats (that are blended together) would change.

We can use different shapes for fuzzy splats in the same way as for opaque splats. Ellipses yield very good results for magnification, but fail for minification, since they fall between pixels. The second option are circular splats whose screen space radius is not allowed to be less than 1. The result is overly blurred for magnification and acceptable for minification. Moreover, the circle does not adapt to surface orientation and the filtering is therefore not anisotropic (which is required for high quality rendering). The third option is EWA splatting (described in the next section), which combines the advantages of circles and ellipses.

The problem of splats falling between pixels is a special case of aliasing. The image is a continuous 2D signal represented by discrete samples. If a continuous signal is too fine and a grid of samples too coarse, the discrete representation fails. Not only does it not represent the fine detail of the continuous signal, but it also misinterprets it as a different, coarser signal (an *alias* of the fine detail).

In case of splatting, the continuous 2D signal (i.e. the image) is a weighted sum of splats. If the splats are large, they cannot produce fine details. If they are small, they do produce fine detail. And if they are smaller than a pixel, they produce a detail which is too fine to be represented by the pixel grid.

Two solutions to this problem are possible: we can increase the screen resolution (supersampling) or modify the splats in a way that they loose the fine detail but retain their basic shape (i.e. *band-limit* them). The second is done by EWA splatting.

An introduction to aliasing and antialiasing in computer graphics is given in [7, 6, 8]. The particular case of alias in the context of splatting is explained by Zwicker in [24].

EWA surface splatting

EWA surface splatting [67] uses fuzzy elliptical splats and band limits them to avoid aliasing. It is a formal framework for splatting that is based on Heckbert's EWA filter [27] used for texture filtering in polygonal rendering.

The definition of the texture function on the surface of a point-based object is illustrated in Figure 14. The point-based object is represented as a set of irregularly spaced points $\{\mathbf{p}_k\}$, each associated with a basis function r_k and coefficients w_k^r, w_k^g, w_k^b for color channels. Without loss of generality we proceed with the discussion using a single channel w_k. Although the domain for the basis functions r_k is the surface of the object, no global parametrization of the object surface is required. Local surface parametrization is sufficient to define the texture function since the support of functions r_k is local (usually a truncated Gaussian). Given a point $\mathbf{q}$ on the surface with local coordinates $\mathbf{u}$ (i.e. coordinates in the local parametrization), the value of the continuous texture function is expressed as

$$f_c(\mathbf{u}) = \sum_{k \in \mathbb{N}} w_k r_k(\mathbf{u} - \mathbf{u_k}), \qquad (2)$$

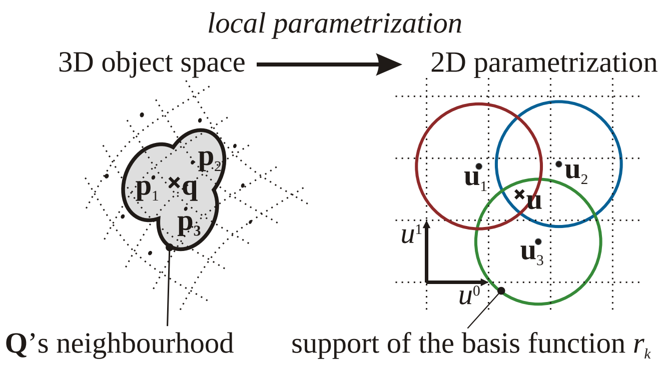

Figure 14: Texture function on the surface of a point-based object.

where $\mathbf{u_k}$ are the local coordinates of the point $\mathbf{p_k}$. The value $f_c(\mathbf{u})$ gives the color of point $\mathbf{q}$.

To render a point-based object whose texture is defined by Equation (2) the texture function f_c has to be mapped to the screen-space. Heckbert's resampling framework [27] is used for this purpose. It involves the following conceptual steps: first, the continuous texture function f_c in object-space is reconstructed from sample points using Equation (2), function f_c is then warped to screen-space using an affine approximation of the object-to-screen mapping, afterwards the warped function f_c is convolved in screen-space with the prefilter h, yielding the band-limited output function $g_c(\mathbf{x})$, and finally the band-limited function g_c is sampled to produce alias-free pixel colors. Concatenating the first three steps, the output function g_c is

$$g_c(\mathbf{x}) = \sum_{k\in\mathbb{N}} w_k \rho_k(\mathbf{x}) \tag{3}$$

where

$$\rho_k(\mathbf{x}) = (r'_k \otimes h)(\mathbf{x} - \mathbf{m}_{\mathbf{u}_k}(\mathbf{u}_k)), \tag{4}$$

r'_k is the warped basis function r_k, h is the prefilter, $\mathbf{m}_{\mathbf{u}_k}$ is the affine approximation of the object-to-screen mapping around point $\mathbf{u}_k$ and $\otimes$ denotes convolution. By $\mathbf{x}$ are denoted coordinates in screen-space. Function ρ_k is the warped filtered basis function r_k and is called the *resampling kernel*. Equation (3) states that the band-limited texture function in screen space can be rendered by first warping and band-limiting the basis functions r_k individually and then summing them up in screen space.

'Warping the basic function' in the context of the EWA splatting is what we called in previous sections 'projecting a surfel to screen-space'. The band-limiting step (i.e. the convolution with the prefilter h) makes the splat bigger, so that it never falls between pixels. The resampling kernel is basically an 'enlarged splat'.

The EWA framework uses elliptical Gaussians as the basis functions r_k and the prefilter h. With Gaussians it is possible to express the resampling kernel in a closed form as a single elliptical Gaussian. An elliptical Gaussian in 2D with the variance matrix $\mathbf{V}$ is defined as $\mathcal{G}_{\mathbf{V}}(\mathbf{x}) = \frac{1}{2\pi\sqrt{|\mathbf{V}|}} e^{-\frac{1}{2}\mathbf{x}^T\mathbf{V}^{-1}\mathbf{x}}$, where $|\mathbf{V}|$ is the determinant of $\mathbf{V}$. Matrix $\mathbf{V}^{-1}$ is the so-called conic matrix and $\mathbf{x}^T\mathbf{V}^{-1}\mathbf{x} = const.$ are the isocontours of the Gaussian $\mathcal{G}_{\mathbf{V}}$. They are ellipses if and only if $\mathbf{V}$ is positive definite [27].

The variance matrices for basis function r_k and the prefilter h are denoted $\mathbf{V}^r_k$ and $\mathbf{V}^h$ respectively. Usually $\mathbf{V}^h = \mathbf{I}$ (the identity matrix). With Gaussians, Equation (4) becomes

$$\rho_k(\mathbf{x}) = \frac{1}{|\mathbf{J}_k^{-1}|} \mathcal{G}_{\mathbf{J}_k\mathbf{V}^r_k\mathbf{J}_k^T + \mathbf{I}}(\mathbf{x} - \mathbf{m}(\mathbf{u}_k)). \tag{5}$$

In this formulation $\rho_k(\mathbf{x})$ is a Gaussian and is called the *screen space EWA resampling kernel*. $\mathbf{J}_k$ denotes the Jacobian of the object-to-screen mapping $\mathbf{m}$ at $\mathbf{u}_k$. For a circular surfel with radius r, the variance matrix $\mathbf{V}_k^r$ is

$$\mathbf{V}_k^r = \begin{pmatrix} 1/r^2 & 0 \\ 0 & 1/r^2 \end{pmatrix}.$$

For computation of $\mathbf{J}_k$, see [46, 68].

The *rendering algorithm* does not differ from the general splatting algorithm (Figure 9). Equation 5 is used to compute the splat shape.

The Gaussian resampling kernel ρ_k has an infinite support in theory. In practice, the support is truncated and the resampling kernel is evaluated only for a limited range of exponent $\beta(\mathbf{x}) = \mathbf{x}^T(\mathbf{J}_k\mathbf{V}_k^r\mathbf{J}_k^T + \mathbf{I})^{-1}\mathbf{x}$, for which $\beta(\mathbf{x}) < c$, where c is a *cutoff radius*.

EWA splatting provides a smooth transition from minification to magnification. In the case of minification, the warped basis function (i.e. the projected surfel) is very small. In this case the prefilter h is dominant and the resulting resampling kernel (i.e. splat) is nearly circular. Thus for minification EWA splatting behaves like splatting with circular splats. For magnification, the size of the prefilter h is small compared to the warped basis function and does not affect the splat shape significantly. Thus for magnification EWA splatting behaves like splatting with elliptical splats.

The EWA splatting was extended to antialiased volume rendering [66] and both the EWA volume and the EWA surface splatting were presented in an unified framework in [68].

Shading

With the splatting algorithms shading can be done per-splat or per-pixel. *Per-splat shading* leads to one shading calculation per splat, including occluded splats. The color resulting from the per-splat shading is then interpolated between splats to produce final pixel colors. *Per-pixel shading* can be done if the material properties (including normals) are interpolated between splats. This leads to one shading calculation per pixel, but excluding the occluded surfels. Per-pixel shading produces better results for specular object.

Order Independent Transparency with A-buffer

Zwicker et al. [67] proposed to use the a-buffer [10] for order independent transparency. For every pixel a linked list of fragments is maintained, sorted according to the fragments' z-depth. The conventional z-buffer can be regarded as a specialization of the a-buffer, where there is a maximum of one fragment per pixel. The fragment attributes are: depth or depth range (depending on the algorithm used for visibility), color, alpha, possibly normal and shading attributes (if the shading is to be done per-pixel).

For every new fragment coming from splat rasterization, a depth-test is performed against the fragments in the a-buffer pixel. If the fragment is found to come from the same surface as one of the fragments in the a-buffer, they are blended together. Otherwise the new fragment is inserted into the a-buffer (Figure 15). For depth testing in a-buffer both z-offsets of z-ranges can be used.

After splatting all points, the color of each pixel is computed from its a-buffer fragments using alpha blending.

Edge Anti-Aliasing

For edge antialiasing, Zwicker et al. [67] use a heuristic based on fragments weights (the weights used to blend the splats, w in Equation (1)). They assume that the weights should sum approximately to one in case of a pixel fully covered by the surface. They detect the partial coverage of a pixel by testing the summed weights against a threshold τ (they set $\tau = 0.4$). If $\sum_i w_i < \tau$, then it is assumed that the splats do not fully cover the pixel, indicating an edge or silhouette. In this case, they set $\alpha = \alpha \cdot (\sum_i w_i/\tau)$, effectively making the partially covered pixels transparent. If $\sum_i w_i >= \tau$, full coverage is assumed.

This anti-aliasing technique has one great advantage: with the a-buffer it comes at nearly no cost. However, the a-buffer is not an option but a necessity for this technique to work. This is due to the fact that the sum of weights near silhouettes is not very reliable and can change rapidly as the object moves

```
// a-buffer fragments sorted front-to-back
ABuffer(new fragment) {
 result = FAIL
 for each fragment in the a-buffer pixel {
  result = DepthTest(new fragment, current fragment)
  if (result == PASS) {
   insert new fragment before the current fragment
   break
  }
  else if (result == BLEND) {
   blend new fragment with current fragment
   break
  }
  else         // result == FAIL
   continue
 }
 if (result == FAIL)
  insert new fragment after current fragment
}
```

Figure 15: Testing new fragment against those in a-buffer.

(in an animation). The a-buffer hides the artifacts by blending most of the silhouette fragments together, but in a rather uncontrollable way. As a result this technique for edge antialiasing does not produce high quality results. Edge anti-aliasing in EWA surface splatting is discussed in more detail in [45].

Special Forms of Splatting

Kalaiah and Varshney introduced *differential points* [30, 29] storing local differential surface properties with each point (Figure 3): the principal directions (directions of minimum and maximum curvatures) $\hat{\mathbf{u}}_\mathbf{p}$ and $\hat{\mathbf{v}}_\mathbf{p}$ and the principal curvatures (the normal curvatures in the principal directions). Those differential properties define a normal variation around the point. The normal variation can be used for better shading. The differential points are rendered as rectangles on the point's tangent plane with a normal map assigned to them. Thus the differential properties are used only for shading. Even though no interpolation between splats is done (the splats are opaque) the reconstructed surfaces look very smooth.

Linsen [33] visualizes point clouds with *triangle fans*. For each point $\mathbf{p}$ he finds the k nearest neighbors, the neighbors are sorted according to the angle, thus they form a triangle fan with point $\mathbf{p}$ in the center. The fan can be rendered using standard OpenGL techniques.

3.3.4 Up-sampling

Up-sampling can be also considered as a reconstruction technique since it prevents the appearance of holes by assuring a sufficiently high density of points according to the current viewpoint. This assures that every pixel receives a foreground point. This is in fact splatting with quads of pixel size. A problem associated with this approach is that point samples have to be generated dynamically during rendering (see Section 3.7.1). This is not only tricky but often slow. The up-sampling techniques are specific rather by the way they produce the samples than the way they solve the visibility and image reconstruction. They are equally applicable to any reconstruction technique described in this chapter.

MLS-Surfaces

Alexa et al. [2] have an initial point cloud (e.g. directly from a scanning device), with each point they store a local polynomial approximation to the surface. The approximation is dynamically sampled at render-time to produce the required sampling density which assures a hole-free reconstruction. To compute the local polynomial approximations, Alexa et al. use their MLS fitting.

$\sqrt{5}$-sampling

Stamminger and Drettakis [55] first compute an initial sample set at regular intervals in a parametric domain on the surface. This initial point set is dynamically refined at render-time using the $\sqrt{5}$-sampling to assure the prescribed sample density in screen space.

Randomized z-buffer

Wand et al. [58] have a triangle set as the initial scene representation. A spatial subdivision is created in a preprocess that sorts the triangles into groups showing a similar scaling factor when projected to screen (i.e. their distance to the viewer is similar). At render-time, for each group a total number of samples is estimated that would satisfy the given screen space density. This is done using the distance of the group's bounding box from the viewer and the total area of triangles within the bounding box. Within each group, the samples are chosen randomly from the triangles with a probability density function proportional to the triangles' surface area in object space.

3.3.5 Discussion

Each of the described techniques for surface reconstruction (Table 1) includes some kind of splatting, at least to eliminate the occluded background surfels.

Quad splatting vs. EWA splatting

The most often used technique in applications is splatting with opaque quad splats (OpenGL `GL_POINTS`) mainly because of its simplicity and speed.

EWA splatting proved to be the method of choice for high quality point rendering as it provides anisotropic texture filtering and possibly edge anti-aliasing. EWA splatting is at the moment still more a research subject than of practical use in applications. Rusinkiewicz and Levoy [49] have shown that quad splatting produces a better image than fuzzy splatting in a given limited time since the splat rate of quad splatting is much higher than that of fuzzy splatting. However, they rendered objects without textures, therefore the full power of fuzzy splatting was not exploited. EWA splatting comes with several *drawbacks* that discourage its use:

Speed. Fuzzy splatting is two to four times slower than quad splatting.

Extended z-buffering. Extended z-buffering is not a standard technique, it is not supported in hardware, it can produce artifacts in the case of wrong "blend" answers, typically for intersecting surfaces.
It would be possible to extend the heuristics used for blending: once the weight at a single surface reaches some limit (>1), all subsequent fragments would be supposed to come from different surface. This simple trick could improve the quality of image, but on the other hand it is not conceptually very nice since it introduces non-geometric quantities into the depth test.

Normalization. The post-normalization of splat contributions is not standardly supported in hardware, the pre-normalization (or per-surfel normalization), as done by Ren et al. [46], doesn't always lead to satisfactory results.

Generality. Derivation of EWA splatting assumes a continuous surface. It is not clear what results it gives for an unstructured geometry as trees. In presentation of [24], Stamminger argued that quad splatting gives visually better results for trees, since the image looks sharp (with fuzzy splats it would look blurred). Matusik et al. [37] on the other hand use EWA splatting even for totally unstructured objects.

Interpolation of Attributes

It is important to interpolate surfels' attributes in order to render smooth surfaces and to hide the explicit structure of surfels.

The techniques that separate visibility from image reconstruction perform this interpolation in screen space, with filter shapes independent of the real orientation of surfaces. This is acceptable only if the density of points in screen space is high and leads to artifacts for lower densities. Moreover this kind of interpolation is reasonable only for color, not for geometric attributes such as normals or (u, v) mapping coordinates. This excludes those techniques from high quality per-pixel shading or per-pixel texture mapping (environment mapping).

Splatting with quad splats doesn't perform any kind of interpolation. It is assumed that the surfel density is high enough to hide most of the artifacts. Per-surfel texture mapping is still possible, but the quality is questionable (no one applied an environment map to quad splats).

Fuzzy splatting with elliptical splats (including EWA splatting) provides correct attribute interpolation, which takes into account surface orientation. Per-pixel shading and texture mapping is possible.

Some techniques that dynamically up-sample the point set for rendering, interpolate the attributes, others do not. In this paragraph we mean attribute interpolation when up-sampling an object, not in the rendering pipeline (those are orthogonal topics). All techniques, of course, interpolate position. Alexa et al. [2] up-sample the local polynomial approximation for object shape. This gives interpolated position and normal. It would be straightforward to build local polynomial approximations for scalar attributes (color, alpha, $\ldots$) and mapping coordinates in the same way as for position. Stamminger and Drettakis [55] ($\sqrt{5}$-sampling) up-sample surfaces in the parametric domain, thus they get mapping coordinates at no cost and they can interpolate scalar attributes in the parametric domain. Wand et al. [58] (randomized z-buffer) sample a triangle as a random linear combination of its vertices which allows them to interpolate any kind of per-vertex attributes.

Possible Extensions and Applications of EWA splatting

Kalaiah and Varshney's differential point rendering [29, 30] is excellent at rendering smooth objects. The drawback of their rendering technique is the lack of interpolation between points and consequently the inability to smoothly reconstruct textures. It would be straightforward to combine differential point rendering with fuzzy ellipse splatting. With fast hardware implementation, this would yield a high quality rendering technique that could be used for real-time rendering of NURBS models. It is, however, not clear at first sight how to combine differential points with EWA splatting, since EWA splatting modifies the projected splat shape. The question is how to map the normal variation to the prefiltered splat.

Antialiasing

We did not fully cover the problem of aliasing in point-rendering here. Briefly, point-rendering is very prone to aliasing. The point representation itself is discrete and can contain alias. This is discussed in Section 3.7.3. In the case of rendering, aliasing can occur because of discrete output image. Three kinds of aliasing can occur: texture aliasing, edge aliasing and aliasing in shading.

Texture aliasing on continuous surfaces is addressed by EWA splatting, even though the filter shape is limited to Gaussians. Other filter would merit investigation as suggested already by Levoy and Whitted [32] and later by Zwicker et al. [68].

The easiest, most robust and slowest solution to edge aliasing is supersampling (e.g. Pfister et al. [43]). Zwicker et al. [67], on the other hand, use a heuristic based on accumulated weights for edge anti-aliasing in their implementation of EWA splatting (described previously in this section). It is fast once we have got the a-buffer, it works reasonably well if the point density is high and the rendered object is sampled more or less regularly. However, it requires an a-buffer to produce satisfactory results. The a-buffer is slow in software (our implementation of EWA splatting was 2 to 3 times slower with an a-buffer than with an ordinary z-buffer) and not supported in hardware. Moreover, Zwicker et al.'s technique does not allow to control the color ramp at edge. Edge aliasing needs future research.

Aliasing in shading can occur if the normal variation is too high and highly view dependent shading is used (e.g. sharp specular highlights). This is the least objectionable form of aliasing and occurs very rarely in surface splatting. The aliasing in shading can probably be a more seriuos problem in the EWA *volume* rendering since Zwicker et al. [68] lists the shading anti-aliasing as one of the main points for future research.

There is no anti-aliasing technique for point-based rendering of unstructured objects. In our opinion this topic leads to volume rendering or rendering with volumetric tectures (see e.g. the work of Fabrice Neyret) or plenoptic techniques.

3.4 Visibility Culling

Although visibility culling is not necessarily a part of a point rendering system, it is used in nearly all of the proposed systems, since it proves to be indispensable for good performance. We recognize three kinds of culling: *view frustum culling* culls the geometry that is not inside the viewing frustum, *backface culling* culls the geometry that is oriented away from the viewer, and *occlusion culling* culls the geometry occluded by some other geometry.

Performing culling on a per-point basis wouldn't be very efficient, since we deal with a high number of points. Culling is therefore applied on a group of points or in a hierarchical fashion: e.g. Grossman and Dally [25, 26] store points in 8×8 blocks and perform culling per-block. Rusinkiewicz and Levoy [49] have a hierarchy of points, where a point at an inner node is a coarse proxy for points at its descendant nodes. By performing culling on an inner node, all descendants are effectively culled.

3.4.1 View Frustum Culling

In whatever way the points are organized, each group of points to be culled has a bounding box. View frustum culling is performed by testing whether the bounding box lies outside the view frustum. The view frustum is defined by its 'near' and 'far' planes and four 'side' planes that pass through the viewpoint and the sides of the screen.

3.4.2 Back-Face Culling

For back-face culling, we need to answer the question 'Do all normals of points in this group point away from the viewer?' If the answer is yes, then there is no need to render those points, since they are all backfacing. We need a fast way to answer this question.

Normal cones [54] is a standardly used technique for backface culling. With each group of points a *normal cone* (cone of normals) is stored that contains normals of all points in the group. The normal cone is represented by its axis (which can be viewed as an average normal of all points in the group) and width. Having the normal cone we can answer the question using a few scalar products.

3.4.3 Occlusion Culling

Occlusion culling is not addressed by current point-rendering systems. One exception is provided by Grossman and Dally's *visibility masks* [25, 26].

Visibility Masks

Grossman and Dally use visibility masks based on the normal masks by Zhang and Hoff [63]. A bitmask is assigned to a group of points, each bit corresponds to a set of directions on a sphere of directions (a triangular sphere subdivision is used to define the directions, each bit then corresponds to a triangle in the sphere triangulation). The k-th bit in a group's bitmask is 1 iff at least one point of the group is visible from a direction which lies inside the k-th triangle. Such a bitmask is called a *visibility mask*.

Visibility masks for an object are computed by rendering the whole object from a number of directions inside one direction triangle. The groups belonging to points that contributed to the final image are tagged by 1 for this direction. This is repeated for all direction triangles. This procedure takes into account not only backface culling, but also occlusion culling within the object (a part of the object occludes other part of the same object). Note that it would also be possible to compute the visibility mask purely on the basis of information about point normals in the group. However, in that case, the resulting visibility mask would take into account only backface culling and not the occlusion.

For each frame to be rendered the direction mask for the screen is computed, which sets its k-th bit to 1 iff a direction from any screen pixel to the eye lies inside the k-th direction triangle. This screen mask is then used for culling of groups of points. For a group to be culled the screen mask is ANDed with the visibility mask of that group. If the result is zero, the group is certainly not visible and it is culled.

The technique of visibility masks is fast and takes into account occlusion culling within a single object, however it cannot describe occlusion of one object by another nor it can be used for dynamic objects (objects that change their shape).

3.4.4 Discussion

Occlusion culling hasn't so far been addressed by the point-rendering systems. The main difficulty is the lack of connectivity in point-based objects: it is hard to predict whether an object is opaque and contains no gaps before actually rendering it. In such cases we cannot predict how such an object occludes other objects in a scene. Point-based representation is definitely less suitable for exact conservative occlusion culling than triangular meshes (at least it seems so at the moment...). If we allow for some error in the occlusion calculations, however, with a limited probability of its occurrence, we get to *approximate* occlusion culling and here the points can do good.

Occlusion culling is very important for point-based rendering of complex scenes, since the running time of majority of the algorithms is proportional to the projected area of the rendered object, *including occluded surfaces*.

3.5 Data Structures, Level-of-Detail Control and Compression

A number of data structures have been proposed for point-based rendering. Here we describe only two of them, namely bounding sphere hierarchy by Rusinkiewicz and Levoy [49] and LDC tree by Pfister et al. [43], since they show the main concepts: the hierarchy represents a hierarchical space subdivision of the object's bounding volume, this subdivision is used for hierarchical view frustum and backface culling. Leaves contain the most refined representation of the object, interior nodes contain coarser representation and the root contains the coarsest representation of the object. This *multiresolution representation* is used for level-of-detail control. Moreover, the hierarchy structure can be used for space-efficient encoding and quantization of point attributes by encoding deltas between the hierarchy levels.

3.5.1 Bounding Sphere Hierarchy

Rusinkiewicz and Levoy designed for their QSplat system [49] a data structure based on the hierarchy of bounding spheres. Points in the hierarchy are treated as spheres. Each node contains the sphere center and radius, a normal, the normal cone width and optionally a color. The hierarchy is constructed in preprocess and written to disk. The same layout is used for in-core representation and for the external file. The tree is written in breadth first order, thus the first part of the file contains the entire model at low resolution. This is used for the incremental uploading of the data from the disk: a file is memory mapped so the data are uploaded as they are accessed.

To *render* an object, the hierarchy is traversed and the following actions are performed for each node. 1) *view-frustum culling*, by intersecting the sphere with the viewing frustum, 2) *backface culling*, with the per-node normal cones [54] 3) *LOD decision*, if the projected size of the node is smaller than a threshold then draw splat, otherwise recurse further. The threshold for projected node-size is updated from frame to frame to maintain a user-selected frame rate. A pseudocode for the traversal algorithm is in Figure 16.

```
TraverseHierarchy(node)
{
 if (node not visible)
   skip this branch of the tree
 else if (node is a leaf node)
   draw a splat
 else if (benefit of recursing further is too low)
   draw a splat
 else
  for each child in children(node)
   TraverseHierarchy(child)
}
```

Figure 16: QSplat bounding sphere hierarchy traversal algorithm.

Preprocessing algorithm builds the bounding sphere hierarchy bottom-up. The input is a triangular mesh. Its vertices are used as the initial points. The radius is set such that no holes can occur (all points overlap). The tree is built by splitting the set of vertices along the longest axis of its bounding box, recursively computing the two subtrees and finding the bounding sphere of its children. The per-vertex properties are averaged in the interior nodes. The branching factor is approximately 4.

Compression is achieved by the quantization of point attributes. Position and radius of each sphere are encoded relatively to its parent and quantized to 13 bits. This is a form of hierarchical quantization exploiting the coherence of positions of a parent and its children. Normals are quantized to 14 bits, width of a normal cone to 2 bits and colors to 16 bits. The complete node with color takes 48 bits. Compression of point-based objects is also addressed in [9, 20, 62].

The criterion for *LOD selection* is based on the projected size of a sphere: if the projected size is smaller than the threshold, the traversal is stopped and the current sphere is splatted. Luebke and Hallen [34] proposed to base this decision on perceptual metrics. Using a model of human vision, they were able to judge every refinement that is linked to the traversal step as either perceptible or imperceptible. If it is imperceptible, the traversal is stopped. This leads to an optimized performance since coarser representation is allowed in visual periphery.

The same data structure with only minor modifications was also used in [2, 3, 12].

3.5.2 LDC Tree

Pfister et al. [43] used the LDC tree for their Surfels system. The data structure was inspired by image-based rendering. Before explaining the LDC tree structure, let us describe the sampling procedure used to produce the point set. The object is sampled from three sides of the object's bounding cube with an orthogonal camera using a ray tracer. The samples from one side form a layered depth image or LDI. All intersections along each ray are stored, not only the first one. The three orthogonal LDIs form a so-called layered depth cube (LDC).

The *LDC tree* is an octree whose nodes contain LDCs of the part of the object they represent. The union of LDCs in octree nodes at level 0 is the full object LDC as produced by the sampling process. An octree node at level $l > 0$ contains $2\times$ subsampled LDC of its four children. Thus, the number of points per octree node is constant at all levels. Although the same point can appear in more nodes of the LDC tree, the data redundancy is low, because the node contains just references to a single point instance. The LDCs at each hierarchy level are subdivided into *blocks* of user specified size (*e.g.* 8×8).

The rendering algorithm hierachically traverses the LDC tree. For each block, view frustum culling is performed using a block bounding box and backface culling is performed using the precomputed normal cones. The traversal is stopped at the level for which the estimated number of points per output pixel is equal to the desired value (1 for fast rendering, > 1 for high-quality rendering). Points at the selected level are sent to the point rendering pipeline.

3.5.3 Discussion

The disadvantage of the LDC tree is that the object must be sampled at a regular pattern. This is well suited for rendering, since it allows incremental warping, but it does not allow the structure to be used for an arbitrary point cloud without first resampling it. QSplat bounding sphere hierarchy does not share this problem.

3.6 Ray Tracing Point Sampled Surface

The motivation for ray-tracing point-based objects is the desire to apply global illumination effects to them without having to convert them to other representation. To ray-trace a point sampled surface, the fundamental problem is the ray-surface intersection test.

Schaufler and Jensen, 2000

Schaufler and Jensen [51] proposed an intersection test consisting of two parts: the detection of intersection and the computation of the actual intersection point and the interpolated attributes of the intersection.

The intersection detection algorithm looks for an intersection with an oriented disk centered at each point. The radius r of this disk is a global parameter of the algorithm. An octree is used to accelerate the search.

Once an intersection is detected, points near the intersection point are collected that lie inside a cylinder having the ray as its axis and r as its radius. The weighted average of those points' attributes (normal, texture coordinates) are used to compute the attributes of the intersection point. To compute the interpolated intersection point, tangent planes of the collected points are intersected with the ray, the average parameter t of the intersection is computed and plugged into the ray's parametric equation.

Adamson and Alexa, 2003

Adamson and Alexa [1] developed an intersection test based on Alexa's MLS-surfaces [2, 3]. The basic building blocks of MLS-surfaces are projection operators that can project any point near the surface onto the surface (while the surface is represented solely by the point samples) and computation of local polynomial approximation to the surface at any point.

The idea of the intersection test is to iteratively project a point on the ray onto the surface to converge to the intersection point. This involves the following steps (Figure 17).
1) A point r_0 on the ray near the surface is estimated.

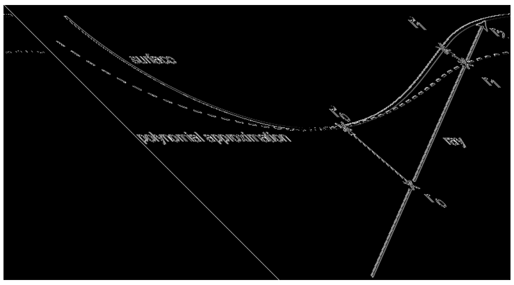

Figure 17: Iterative convergence to the intersection point by intersecting the ray with local polynomial approximation.

2) This point is projected onto the surface, resulting in the point r_0' and a polynomial approximation around r_0'.
3) The ray is intersected with the polynomial approximation, yielding the point r_1, which is supposed to lie closer to the actual intersection than the first estimate r_0.
4) If the pre-specified accuracy is met, stop. Otherwise substitute r_1 for r_0 and go to 2).
The results show that two to three iterations are sufficient to find the intersection point with the accuracy of $10^{-6}h$.

3.6.1 Discussion

Schaufler's intersection test is view dependent, its result depends on the incoming ray direction. Therefore it does not provide a consistent surface definition.

Adamson's technique has many advantages. Firstly, it is built upon a proper (view independent) surface definition, therefore primary and secondary rays hit the same surface, the resulting image is view independent allowing for animation, CSG-defined shapes are possible. Secondly, it shares the advantages of the MLS-surfaces: it is possible to define a minimum feature size and to filter out every feature smaller than this size, the surface is smooth and manifold. Those advantages are spoiled by long execution times (hours to compute a single image).

The minimum feature size property could be used for alias-free surface sampling for image-based and some point-based rendering techniques (Section 3.7.3).

The intersection test between a ray and a point sampled surface is issue that cannot be considered solved. Possible direction would be to to design a projection similar to that in MLS-surfaces, but in a way that would intrinsically contain the ray intersection.

3.7 Object Sampling

Sample sets of object surfaces can be obtained either by scanning a real object with a 3D scanning device [24, 36, 37] or by sampling a synthetic object with a sampling algorithm. Here we discuss different sampling algorithms for synthetic objects.

3.7.1 Classification

We classify the sampling techniques according to the following criteria: randomized/deterministic, in preprocess/on-the-fly (i.e. during rendering), in an input/output sensitive way (Table 3).

Randomized sampling chooses the points on the surfaces in a random way, under the constraint of being distributed according to some probability density function (most often according to surface area of sampled triangles).

Deterministic sampling includes techniques like sampling on a regular lattice, sampling from several views using a perspective camera, taking only the vertices of the input triangle mesh, etc.

Sampling in preprocess provides a view-independent set of samples that are then stored to some special purpose data structure and later used for rendering.

The *on-the-fly sampling* (or dynamic sampling) dynamically samples the object representation according to the current viewpoint to assure a pre-scribed screen space density of points. The samples are cached and used in subsequent frames.

The *output sensitive sampling* samples objects at a resolution that matches the expected resolution of the output image as in image based rendering. An output sensitive sampling algorithm must provide a certain prescribed density of samples (i.e. it must assure maximum size of gap between samples). This is crucial for hole-free point-based rendering. A common means for this sampling is ray-tracing. Grossman [25] discusses the output sensitive sampling in depth. He introduces the notion of *adequate sampling*, which is a minimum sampling rate that assures hole-free rendering of the objects at a specified resolution and unit magnification.

The *input sensitive sampling* provides point samples regardless of the expected output resolution. The aim is rather to sample at a resolution that models all the surface features.

Technique	**R**andomized / **D**eterministic	**P**reprocess / On-the-**F**ly	**I**nput / **O**utpu sensitive
Grossman and Dally [26]	D	P	O
Pfister et al. [43]	D	P	O
Tobor et al. [57]	D	P	O
Rusinkiewicz and Levoy [49]	D	P	I
Alexa et al. [2, 3]	D	F	O
Chen and Nguyen [12]	D	P	I
Cohen et al. [15]	D	P	I
Kalaiah and Varshney [29]	D	P	I
Stamminger and Drettakis [55]	D(R)	F	O
Wand et al. [58]	R	F	O
Wimmer et al. [60]	D	P	O
Deussen et al. [17]	R	P	I
Wand et al. [59]	R	P	I

Table 3: Object sampling techniques.

3.7.2 Overview

Grossman and Dally [25, 26] sample an object using orthographic cameras from 32 directions evenly distributed on the object's bounding sphere. Pfister et al. [43] samples an object with three orthographic cameras on the sides of object's bounding cube. For every ray, not only the first intersection is saved, but all intersections are taken into account, including backfaces. The result of the sampling is three orthogonal layered depth images (LDI) [53]. Tobor et al. [57] use conceptually the same technique as Pfister et al., except that only visible surfaces are saved. Technically, they sample objects by rasterizing them using graphics hardware. Rusinkiewicz and Levoy [49] expect a dense triangle mesh on the input. They simply throw away the connectivity and retain only vertex positions. Chen and Nguyen's sampling [12] is similar, but they put the points in the circumcenters of the input triangles. Alexa et al. [2, 3] have an initial point cloud (e.g. directly from a scanning device), with each point they store a local polynomial approximation to the surface. The polynomial approximation is dynamically sampled at render-time to produce required sampling density. Cohen et al. [15] convert each triangle of the input mesh into points separately, using a constant number of points per triangle. Each triangle is sampled in a way that is similar to triangle rasterization. Kalaiah and Varshney [29] sample NURBS surfaces at regular intervals in the parametric domain and then downsample the point set where it is most redundant. Stamminger and Drettakis [55] sample objects on-the-fly. First an initial sample set is created at regular intervals in the parametric domain on the surface. This initial point set is then dynamically refined using the $\sqrt{5}$-sampling to assure a prescribed sample density in screen space. Wand et al. [58] also sample the set of triangles on-the-fly, with the aim to produce constant sampling density in screen space. A spatial subdivision sorts the triangles into groups that show a similar scaling factor when projected to screen-space (i.e. their distance to the viewer is similar). The triangles within each group are sampled randomly with a probability density function proportional to the riangles' object space area. Wimmer et al. [60] sample distant geometry, so that it can be reconstructed without holes for any viewpoint within a viewing cell (a segment of a street in a city). Three perspective LDI cameras placed within the viewing cell are used. Deussen et al. [17] sample the triangular models of plants in a random way, with probability proportional to triangles' object-space area. Wand et al. [59] do the same for arbitrary triangular mesh, but provide an additional stratification step to make the sampling more even and to eliminate oversampling.

3.7.3 Discussion

Randomized vs. Deterministic

Randomized sampling is excellent when we cannot or don't want to assume anything about the geometry being sampled. It is very suitable for sampling unstructured geometry such as trees. On the downside it is hard to assure a fixed minimum sample density. For modeling trees [17] this does not matter much, no one notices one missing leaf among 100.000. But for continuous surfaces, the sampling density has to be assured, otherwise holes appear in renderings. The only way to assure the minimum sample density with randomized sampling is to significantly oversample. Wand et al. [59] explain that if the minimum number of samples needed to cover a surface with assured density is n, then with randomized sampling the required number of samples is $O(n \log n)$, thus there is $O(\log n)$ oversampling. Another problem is that the distribution of points is uneven and aliasing problems are harder to solve for unevenly sampled surfaces. The only proposed antialiasing technique for randomized sampling [59] is to supersample the surface and average the attributes.

Preprocess vs. On-the-Fly

The main argument for dynamic on-the-fly sampling was that with sampling done in preprocess, the resolution is fixed, thus the magnified rendering is limited. Dynamic sampling on the other hand can always provide object samples with a density that is high enough according to the current viewpoint. But do we need to sample objects at arbitrarily high resolution? Having a hybrid point-polygon rendering system, the answer is no. The points are used as rendering primitives to speed-up rendering, but for a triangle that would be replaced by more than two to three points, there would be no speed-up at all. Thus it is better to render magnified objects with triangles than with points. Therefore a well chosen sample set created in preprocess with a resolution in the order of triangle size seems to be a good choice. The drawbacks of the on-the-fly sampling are slower rendering due to computational resources used for sampling, lower sample set quality due to limited time (there is not enough time for e.g. prefiltering the sampled geometry).

On the other hand, dynamic sampling is still advantageous for procedurally defined objects, where the 'infinitesimal' resolution argument still holds. Other representations, such as NURBS could also benefit from dynamic sampling. There is a great chance that a point-based rendering of NURBS can eventually outperform the rendering based on tesselation of NURBS into triangles.

Input vs. Output Sensitivity

The output sensitive sampling permits maximum efficiency of rendering at the cost of lower flexibility. It risks undersampling object features and is prone to aliasing. For some objects it can even be less efficient: if the sampling density is very high and the object complexity rather low, triangle representation could be better.

Aliasing in Geometry and Texture Sampling

The output sensitive sampling algorithms are prone to introducing alias in the sampled representation if the sampling density is too low. This can be solved by prefiltering the sampled attributes (shape, normal, texture). The prefiltering can be seen in another way: the point sample is not infinitesimally small but represents a finite surface area (Section 3.1). Thus the point sample must express the average value of the attributes over this area. The prefiltering is commonly done for textures [25, 26, 43], but is not done for shape. One way to partially solve the problem of aliasing in sampling is to supersample and then average the samples [59]. Another way is smoothing surfaces before sampling them.

4 Conclusions and Future Research

In this section we list the advantages and disadvantages of point-based techniques, conclude the first part of the report and give some directions for future research on point-based rendering and modeling.

4.1 Advantages of Points . . .

4.1.1 . . . as Rendering Primitives

- Single primitive is very fast to draw (even in software).
- Efficient rendering of complex objects and environments.
- Output sensitive rendering algorithms.
- Simple and efficient rendering pipeline: purely forward with high data locality (surfel carries all its information through the pipeline), no texture look-ups, easily parallelizable.
- Simple, almost continuous LOD control.
- Many opportunities for speed-quality trade-offs.
- “No need for triangle-strips, which are complex to manage for non-static meshes.” [55]

4.1.2 . . . as Modeling Primitives

- Simple topology modification (as no topology is explicitly stored).
- “The adaptive refinement that potentially changes the surface topology is straightforward with points. For meshes the implementation of this would be complex and suffer from robustness problems.” [55] — This includes not only dynamic object sampling, but also e.g. merging of scanned datasets.
- “Point samples of an object can be generated top-down, providing coarse representations very quickly.” [55]

4.2 Disadvantages of Points

Here we list the disadvantages of points that we consider inherent to point-based techniques and serious.

- Inefficient representation and rendering of flat surfaces, no coherence is exploited.
- Inefficient representation of surface normal discontinuities (creases).
- Shading sampling rate is not independent from geometry shading rate.

4.3 Conclusion

In the first part of our report, we introduced points as rendering and modeling primitives. We discussed the state-of-the-art techniques for point-based rendering, with a focus on surface splatting as it is the basis for our work in the field of point-based rendering.

Points are very efficient at modeling and rendering complex environments, they are simple at conceptual level, therefore easy to study. One of the major goals of computer graphics is to represent and visualize reality without explicitly describing its structure, points are a big step toward this end.

4.4 Future Work

Here we summarize some possible directions for future research on point-based modeling and rendering.

- Efficient solution for hardware rendering of point primitives using current graphics APIs.
- Modeling and (antialiased) rendering of low-order surface discontinuities (creases, boundaries).
- Animating point-based objects (skeletal animation).
- New editing paradigms well suited for points.
- Fast, accurate, and robust ray-tracing of point-based surfaces.
- Occlusion culling (possibly probabilistic).
- Antialiased rendering of unstructured point sets (e.g. vegetation).
- Assessment of perceptual fidelity of different rendering techniques, statement of requirements on rendering quality for various applications.
- Better filters for texture antialiasing, better edge antialiasing.
- Standard API for point-based rendering.
- New applications for points.

5 Our Contribution: Depth-of-Field Rendering with EWA Surface Splatting

Our contribution to the field of point-based rendering consist in a new algorithm for depth-of-field (DOF) rendering [31] based on EWA surface splatting. The ability to render the depth-of-field (DOF) effect is an important feature of any image synthesis algorithm. DOF makes the image appear more natural since optical systems both in cameras and in the human eye have lens of final aperture and do not produce perfectly focused images. DOF is also an important depth cue that helps humans to perceive the spatial configuration of a scene [48].

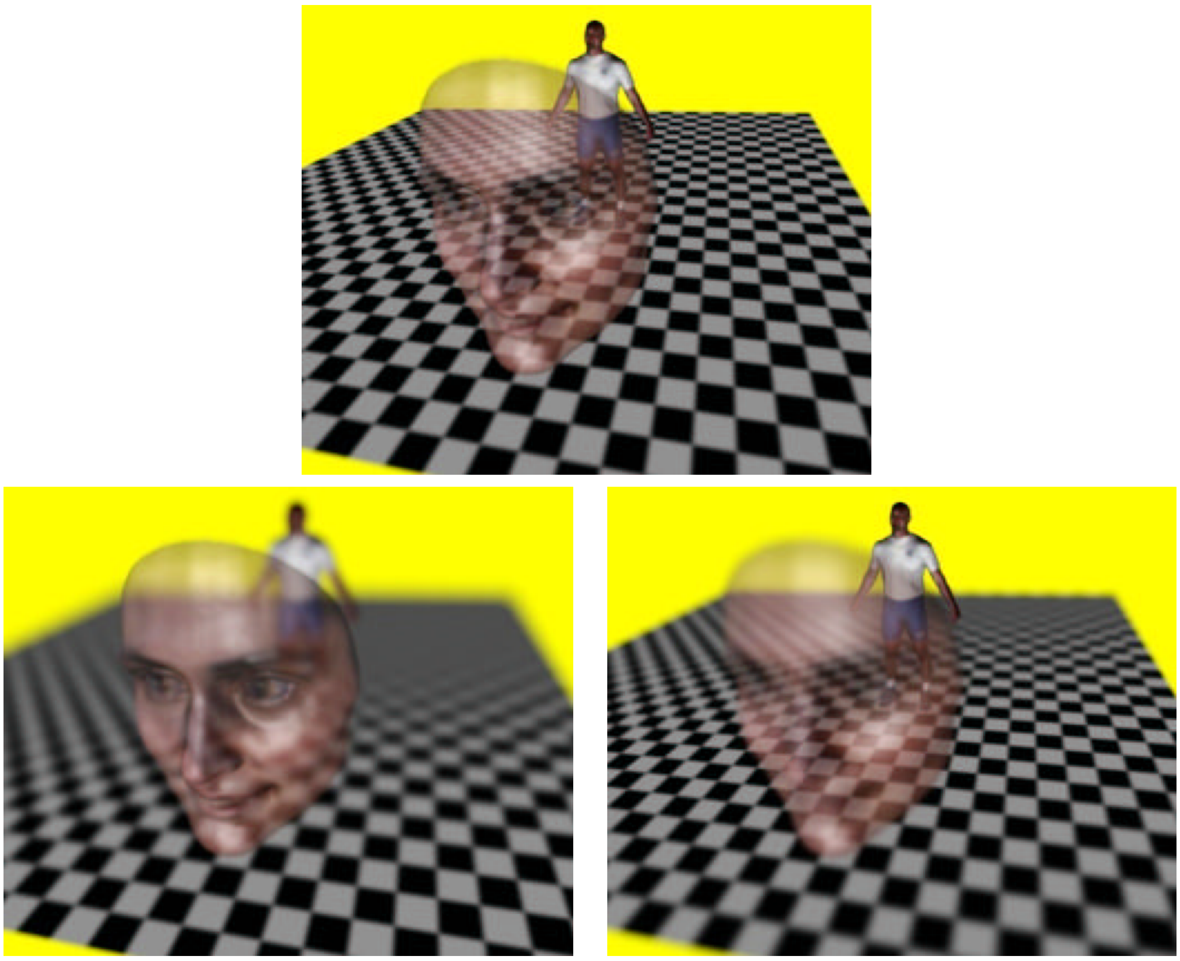

Figure 18: Example of DOF rendering with semi-transparent surface. Top: no DOF. Bottom left: DOF is on and the transparent face is in focus. Bottom right: the male body is in focus, face is out of focus.

The effect of DOF is that out-of-focus points in 3D space form circular patterns (circle of confusion, CoC) in the image plane. The problems contributing to the complexity of DOF rendering are the *partial occlusion* (visibility of objects change for different points on the lens of final aperture) and the *intensity leakage* (*e.g.* blurred background leaks into the focused object in the foreground). Algorithms that solve those problems exist but they are currently slow, especially for large amounts of depth-blur.

Our algorithm renders depth-of-field for a special case of point-based surfaces. Since the algorithm does not decouple visibility calculations from DOF rendering, it handles the partial occlusion correctly and it does not suffer from intensity leakage. The most important features of our algorithm are DOF rendering in presence of transparent surfaces (Figure 18) and a high rendering speed which is practically independent of the amount of depth-blur. The independence of the rendering speed on the amount of depth blur is achieved by exploiting the level-of-detail (LOD) to select coarser representation for highly blurred surfaces. The algorithm builds on top of the EWA surface splatting framework ([67], Section 3.3.3).

The main contributions of our work are fourforld: a mathematical analysis extending the screen space EWA surface splatting to include the DOF rendering ability, an analysis allowing the use of LOD as a means for DOF rendering, an implementation of the algorithm, and a discussion of practical issues arising from the implementation.

5.1 Outline of the Algorithm

Our DOF rendering technique belongs to the group of *post-filtering algorithms* [35]. A post-filtering algorithm for DOF rendering works as follows: first the image is computed using a pinhole camera model. The resulting image along with the depth values for each pixel are then sent to *focus processor*. The focus processor turns every pixel into a CoC whose radius is computed from the pixel depth value. The intensity of the pixel is spread onto its neighbors that fall within that pixel's CoC. Potmesil and Chakravarty [44] have given the formulas to compute the radius of the CoC and described the intensity distribution within the CoC by Lommel functions. Chen [13] proposes to simplify the intensity distribution to uniform. Other simplifying algorithms use Gaussian intensity distribution [47, 19].

Unlike these algorithms, we do not divide DOF rendering into two stages. The defocusing is done simultaneously with image generation on per-splat basis.

We think of DOF rendering as filtering (or blurring) the image with a spatially variant low-pass filter. In the case of surface splatting, the image is formed by summing the contributions from the resampling kernels (also called *splats*). DOF can thus be obtained by first low-pass filtering the individual resampling kernels and then summing the filtered kernels together. Since the supports of the resampling kernels are small, the filtering can be approximated by a convolution, which can be very easily accomplished with circular Gaussians. These statements are discussed in Section 5.3.1.

The introduced scheme for DOF rendering means that each resampling kernel is enlarged proportionally to the amount of depth-blur appertaining to its camera-space depth. Rasterization of such enlarged kernels generates a high number of fragments and thus slowing down the rendering.

We observe that the low-pass filtering (or blurring) is implicitly present in the coarser levels of LOD hierarchies for point-based objects and can be exploited for DOF rendering. The outline of the DOF rendering algorithm with LOD is as follows: while traversing the LOD hierarchy, we stop the traversal at the point whose level corresponds to the blur that is "just smaller" than the required depth-blur for that point's depth. In this way we get the screen-space blur that is approximately equal to the required depth-blur.

Since the LOD hierarchy is discrete, we cannot get arbitrary blur only by choosing the suitable hierarchy level. Therefore we perform an additional low-pass filtering in screen-space, which corrects the LOD blur to produce the desired result. DOF rendering with LOD is described in detail in Section 5.3.3.

5.2 Preliminaries

Before carrying on, we point the reader to Section 3.3.3 that describes EWA surface splatting. Another prerequisite is a camera model used for DOF rendering.

5.2.1 Camera Model for DOF Rendering

The description of the camera model we use for DOF rendering, that will follow in this section, is adopted from [21, 44, 48]. We use the thin lens model for DOF rendering. The parameters that specify the optical system are the following: the *focal length* F, the *aperture number* n, and the *focal plane* *distance* P. F and n specify the *lens diameter* $A = F/n$. Any point which is further from or closer to the lens than P appears out of focus and is displayed as a CoC. The CoC radius C for a point at distance U from the lens is

$$C = \frac{1}{2}|V_u - V_p|\frac{F}{nV_u}, \tag{6}$$

where

$$V_u = \frac{FU}{U-F},\ U > F; \quad V_p = \frac{FP}{P-F},\ P > F. \tag{7}$$

V_p is the distance from the lens to the image plane. It can be given instead of the focal length F, in this case $F = \frac{PV_p}{P+V_p}$. The CoC radius C has to be scaled to express the CoC radius in pixels. The way it is done depends on whether the simulated optical system is a *camera* or a *human eye* (*e.g.* as in virtual reality [48]).

For **camera simulation** we assume that projection and viewport transformations have been set with OpenGL call `glFrustum(L,R,B,T,N,FAR)` and `glViewport(0,0,W,H)`. We also assume undistorted image: $(R-L)/(T-B) = W/H$. Then the radius of CoC in pixels is

$$C_r = C\frac{W}{R-L}\frac{N}{V_p} = C\frac{H}{T-B}\frac{N}{V_p}. \tag{8}$$

For **eye simulation** the lens diameter A is the pupil diameter, which varies from 1 to 8 mm. The average pupil diameter $A = 4$ mm can be used. V_p is the distance from the eye's lens to the retina, which is fixed and its standard value is 24 mm. P is the distance from the observer's eye to the object the eye is focusing on. To get this distance an eye tracking device has to be used to measure the observer's direction of sight. P is then the distance to the nearest visible surface in this direction. Equation (6) gives the radius of CoC on the retina. The CoC radius on the display screen in pixels is

$$C_r = C\frac{d_s}{V_p}R, \tag{9}$$

where d_s is the distance from the eye to the display screen and R is the screen resolution in pixels per unit length.

5.3 DOF Rendering in the EWA Surface Splatting Framework

In this section we extend the screen space EWA surface splatting to include the DOF rendering ability. First, we describe how DOF can be obtained by blurring individual resampling kernels, then we extend the DOF rendering to exploit the LOD.

5.3.1 DOF rendering as a resampling kernel convolution

Neglecting the occlusion we can express the depth-blurred continuous screen space signal g_c^{dof} as

$$g_c^{\mathrm{dof}}(\mathbf{x}) = \int_{\mathbb{R}^2} I(\mathrm{coc}(\mathrm{z}(\zeta)), \mathbf{x} - \zeta)\ g_c(\zeta)\ d\zeta,$$

where g_c is the unblurred continuous screen space signal, $\mathrm{z}(\mathbf{x})$ is the depth at $\mathbf{x}$, $\mathrm{coc}(d)$ is the CoC radius for depth d and $I(r, \mathbf{x})$ is the intensity distribution function for CoC of radius r at point $\mathbf{x}$. I is circularly symmetric and is centered at origin. It is applied to g_c as a spatially variant filter.

Expanding the Equation for g_c using Equation (3) we get

$$\begin{aligned} g_c^{\mathrm{dof}}(\mathbf{x}) &= \int_{\mathbb{R}^2} \Big(I(\mathrm{coc}(\mathrm{z}(\zeta)), \mathbf{x} - \zeta) \sum_{k \in \mathbb{N}} w_k \rho_k(\zeta)\Big)\ d\zeta = \\ &= \sum_{k \in \mathbb{N}} w_k \rho_k^{\mathrm{dof}}(\zeta), \end{aligned}$$

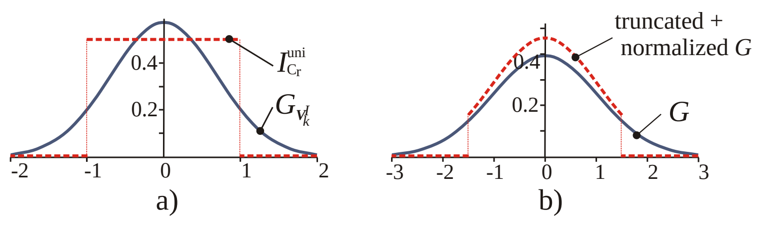

Figure 19: a) Gaussian approximation of the uniform intensity distribution. b) Normalization of a truncated Gaussian.

where

$$\rho_k^{\mathrm{dof}}(\mathbf{x}) = \int_{\mathbb{R}^2} I(\mathrm{coc}(\mathrm{z}(\zeta)), \mathbf{x} - \zeta)\rho_k(\zeta)d\zeta. \tag{10}$$

This means that we can get the depth-blurred screen space function g_c^{dof} by first depth-blurring the individual resampling kernels ρ_k and then summing up the blurred kernels.

We assume that the depth $\mathrm{z}(\mathbf{x})$ does not change within the support of ρ_k and can be replaced by a constant z_k, which is the z-coordinate of the point $\mathbf{p}_k$ in the camera-space. Therefore the function $I(\mathrm{coc}(\mathrm{z}(\zeta)), \mathbf{x} - \zeta)$ can be replaced by the spatially invariant function $I_{\mathrm{coc}(\mathrm{z}_k)}(\mathbf{x} - \zeta)$ and Equation (10) becomes the convolution

$$\rho_k^{\mathrm{dof}}(\mathbf{x}) = (I_{\mathrm{coc}(\mathrm{z}_k)} \otimes \rho_k)(\mathbf{x}). \tag{11}$$

For compatibility with the EWA framework, we choose circular Gaussians as the intensity distribution function I. If we denote the variance matrix for $I_{\mathrm{coc}(\mathrm{z}_k)}$ by $\mathbf{V}_k^I$ then $I_{\mathrm{coc}_{\mathrm{z}_k}} = \mathcal{G}_{\mathbf{V}_k^I}$. We now plug Equation (5) into (11) and we get ρ_k^{dof} in the form

$$\rho_k^{\mathrm{dof}}(\mathbf{x}) = \frac{1}{|\mathbf{J}_k^{-1}|}\mathcal{G}_{\mathbf{J}_k\mathbf{V}_k^r\mathbf{J}_k^T+\mathbf{I}+\mathbf{V}_k^I}(\mathbf{x} - \mathbf{m}(\mathbf{u}_k)). \tag{12}$$

This formulation means that we can get the depth-blurred resampling kernel easily: for each splatted point $\mathbf{p}_k$ we compute the variance matrix $\mathbf{V}_k^I$ and we add it to the variance matrix of the unblurred resampling kernel ρ_k. We show how to compute $\mathbf{V}_k^I$ in the next section. By blurring the resampling kernels individually, we get the correct DOF for whole image.

5.3.2 Variance matrix of the Intensity Distribution Function

Having the depth value z_k, we compute the CoC radius C_r using Equation (8) or (9). Now we want to find such variance matrix $\mathbf{V}_k^I$ that brings the Gaussian $\mathcal{G}_{\mathbf{V}_k^I}$ as close as possible to the uniform intensity distribution within the CoC of radius C_r (Figure 19a). We denote the uniform intensity distribution function by $I_{C_r}^{\mathrm{uni}}$. $I_{C_r}^{\mathrm{uni}}(\mathbf{x}) = 1/\pi C_r^2$ if $\|\mathbf{x}\| < C_r$ and zero otherwise. By the distance between functions we mean the distance induced by the L_2 norm. We know that $\mathcal{G}_{\mathbf{V}_k^I}$ is circular and thus $\mathbf{V}_k^I = a\mathbf{I}$, where $\mathbf{I}$ is the identity matrix and a is a scalar. Hence our problem reduces to finding a suitable a for any given C_r. We are minimizing the following functional:

$$F(a) = \|I_{C_r}^{\mathrm{uni}} - \mathcal{G}_{\mathbf{V}_k^I}\|_{L_2} = \|I_{C_r}^{\mathrm{uni}} - \frac{1}{2\pi a}e^{-\frac{1}{2}\frac{\mathbf{x}^T\mathbf{x}}{a}}\|_{L_2}.$$

We derived the solution $a = \frac{1}{2\ln 4}C_r^2$, thus the variance matrix $\mathbf{V}_k^I$ is

$$\mathbf{V}_k^I = \begin{pmatrix} \frac{1}{2\ln 4}C_r^2 & 0 \\ 0 & \frac{1}{2\ln 4}C_r^2 \end{pmatrix}.$$

One could ask why we are trying to find the best Gaussian approximation of the uniform intensity distribution, which is in turn just an approximation of what the intensity distribution really is (the Lommel distribution [44]). The reason is that the mathematical intractability of Lommel intensity distribution function did not allow us to express a in closed form.

5.3.3 DOF Rendering with Level-of-Detail

The DOF rendering algorithm as presented so far would be very slow (see timings in Section 5.5) because of the high number of fragments generated by the rasterization of the blurred resampling kernels. LOD hierarchies such as those used in [43, 49] typically low-pass filter the texture for coarser levels. We observe that this low-pass filtering can be considered as blurring — if we choose coarser hierarchy level, we get a blurred image. However, by just choosing a suitable level, we cannot steer the amount of blur precisely enough. We solve this by an additional low-pass filtering in screen-space. Another problem is that the low-pass filtering in the LOD hierarchy is done in the local coordinates on the object surface, whereas we need to perform the low-pass filtering in screen-space. Fortunately, there is a simple relation between the two spaces, given by the local affine approximation of the object-to-screen mapping.

To express those intuitions more rigorously, we slightly change the definition of the texture function for a point-based object (Equation 2) to take into account the LOD hierarchy with texture prefiltering.

Extended Surface Texture Definition

Having the multiresolution representation of a point-based surface, we assume for this discussion that there are distinct levels identified by integers 0 to M, where level 0 are leaves. The continuous texture function f_c^l at hierarchy level l is represented by a set of basis functions r_k^l. This representation is created by low-pass filtering and subsampling the representation for the texture function f_c from level 0. The basis function r_k^l is assumed to be created by convolving r_k (basis function for level 0) with a low-pass filter q_k^l: $r_k^l(\mathbf{u}) = (r_k \otimes q_k^l)(\mathbf{u})$. The continuous texture function f_c^l is then

$$f_c^l(\mathbf{u}) = \sum_{k \in \mathbb{N}} w_k r_k^l(\mathbf{u} - \mathbf{u}_k) = \sum_{k \in \mathbb{N}} w_k (r_k \otimes q_k^l)(\mathbf{u} - \mathbf{u}_k).$$

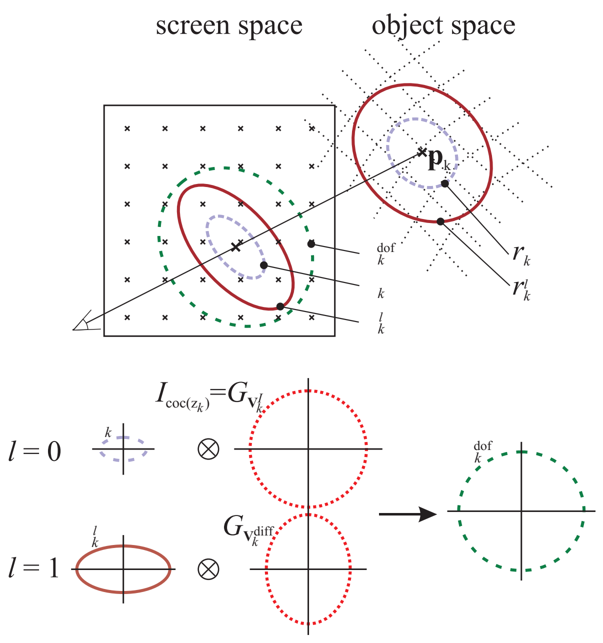

Figure 20: DOF rendering with different LODs.

Application to Depth-Blurring

After having defined the texture function on coarser LOD, we focus on transforming the filters q_k^l to the screen-space and using the coarser representation of the texture function for depth-blurring.

We assume that all the basis functions from all hierarchy levels are Gaussians and that the low-pass filters q_k^l are Gaussians as well: $r_k^l = \mathcal{G}_{\mathbf{V}_k^{r^l}}$ and $q_k^l = \mathcal{G}_{\mathbf{V}_k^{q^l}}$. Recall that $\mathbf{V}_k^r$ is the variance matrix associated with the basis function r_k from level 0. We then have $\mathbf{V}_k^{r^l} = \mathbf{V}_k^r + \mathbf{V}_k^{q^l}$ (because $r_k^l = r_k \otimes q_k^l$) and the resampling kernel ρ_k^l for the basis function r_k^l is

$$\rho_k^l(\mathbf{x}) = \frac{1}{|\mathbf{J}_k^{-1}|} \mathcal{G}_{\mathbf{J}_k(\mathbf{V}_k^r + \mathbf{V}_k^{q^l})\mathbf{J}_k^T + \mathbf{I}}(\mathbf{x} - \mathbf{m}(\mathbf{u}_k)). \tag{13}$$

The variance matrix of this Gaussian is $\mathbf{V}_k^l = \mathbf{J}_k(\mathbf{V}_k^r + \mathbf{V}_k^{q^l})\mathbf{J}_k^T + \mathbf{I} = \mathbf{J}_k\mathbf{V}_k^r\mathbf{J}_k^T + \mathbf{I} + \mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T$. Therefore we can consider the resampling kernel ρ_k^l to be the resampling kernel ρ_k *convolved in screen space* with the Gaussian $\mathcal{G}_{\mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T}$. In other words, by selecting the hierarchy level l to render the surface around point $\mathbf{p}_k$, we get the blurring in screen space by the Gaussian $\mathcal{G}_{\mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T}$.

If we now look at Equation (12), we see that to get the blurred resampling kernel ρ_k^{dof} from ρ_k, ρ_k has to be convolved with the Gaussian $\mathcal{G}_{\mathbf{V}_k^I}$. Thus, to get ρ_k^{dof} from ρ_k^l, we have to convolve ρ_k^l with the Gaussian $\mathcal{G}_{\mathbf{V}_k^{\mathrm{diff}}}$, where the variance matrix $\mathbf{V}_k^{\mathrm{diff}}$ is given by $\mathbf{V}_k^I = \mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T + \mathbf{V}_k^{\mathrm{diff}}$. Convolution with Gaussian $\mathcal{G}_{\mathbf{V}_k^{\mathrm{diff}}}$ can be regarded as an additional blurring needed to produce the required screen-space blur after we have selected the hierarchy level l.

Since $\mathbf{V}_k^{\mathrm{diff}}$ is a variance matrix of an elliptical Gaussian, it must be positive definite [27]. Such a matrix exists if Gaussian $\mathcal{G}_{\mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T}$ is "smaller" than Gaussian $\mathcal{G}_{\mathbf{V}_k^I}$ (*i.e.* iff the elliptical area $\{\mathbf{x} \mid \mathbf{x}^T(\mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T)^{-1}\mathbf{x} \leq 1\}$ is a subset of the elliptical area $\{\mathbf{x} \mid \mathbf{x}^T(\mathbf{V}_k^I)^{-1}\mathbf{x} \leq 1\}$).

The idea of using the LOD to speed-up depth-blurring is to select such a hierarchy level l that (positive definite) $\mathbf{V}_k^{\mathrm{diff}}$ exists but Gaussian $\mathcal{G}_{\mathbf{V}_k^{\mathrm{diff}}}$ is "as small as possible", *i.e.* $\mathcal{G}_{\mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T}$ is "just a bit smaller" than $\mathcal{G}_{\mathbf{V}_k^I}$. This means that the amount of blur that needs to be added by the Gaussian $\mathcal{G}_{\mathbf{V}_k^{\mathrm{diff}}}$ is very small and therefore the blurring does not significantly slow down the rendering. This concept is illustrated in Figure 20.

5.4 Implementation

In this section we describe the implementation of the DOF rendering algorithm in the screen space EWA surface splatter. Our goal is to include the ability to render DOF as smoothly as possible.

5.4.1 DOF rendering without LOD

For every point $\mathbf{p}_k$ being splatted we compute the variance matrix $\mathbf{V}_k^I$ (Section 5.3.2) and add it to the variance matrix of ρ_k (Equation 5) to get the blurred resampling kernel ρ_k^{dof} (Equation 12). It is then rasterized as in normal surface splatting.

5.4.2 DOF rendering with LOD

LOD Selection

We adopted the QSplat bounding sphere hierarchy for LOD control [49] (Section 3.5) and add one new criterion to stop the LOD hierarchy traversal. The traversal is stopped if the projected size of the node gets smaller than the CoC radius for that node. This is a sufficient condition for the existence of positive definite $\mathbf{V}_k^{\mathrm{diff}}$ since $\mathcal{G}_{\mathbf{V}_k^{r^l}}$ and $\mathcal{G}_{\mathbf{V}_k^I}$ are both circular Gaussians. Figure 21 shows an example of LOD selection for DOF rendering. The left image visualizes the points used to render the image on the right. The size of the points corresponds to the LOD.

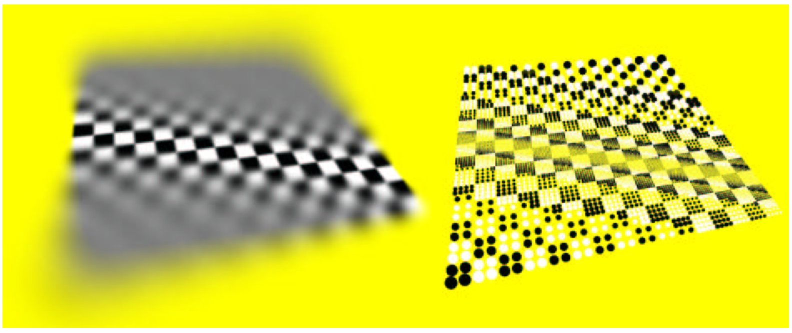
Figure 21: Example of LOD selection for DOF rendering.

Per-Splat Computation

For each point $\mathbf{p}_k^l$ being splatted we need to determine the low-pass filter q_k^l (it is given by the hierarchy level l) and we then need to compute the matrix $\mathbf{V}_k^{\text{diff}}$ for additional screen-space blurring. We use the following computations:

$$\boxed{\begin{array}{l} \mathbf{V}_k^{\text{diff}} := circumellipse(\mathbf{J}_k \mathbf{V}_k^{q^l} \mathbf{J}_k^T \ , \ \mathbf{V}_k^I) \\ \mathbf{W} := \mathbf{J}_k \mathbf{V}_k^r \mathbf{J}_k^T + \mathbf{I} + \mathbf{V}_k^{\text{diff}} \end{array}}$$

$\mathbf{W}$ is the resulting matrix of the resampling kernel. The function $circumellipse$ returns the variance matrix of the Gaussian which is “bigger” than both Gaussians $\mathcal{G}_{\mathbf{J}_k \mathbf{V}_k^{q^l} \mathbf{J}_k^T}$ and $\mathcal{G}_{\mathbf{V}_k^I}$ and its implementation is given in the next paragraph. According to how the LOD selection algorithm was designed, the most common case is that $\mathcal{G}_{\mathbf{V}_k^I}$ is “bigger” than $\mathcal{G}_{\mathbf{J}_k \mathbf{V}_k^{q^l} \mathbf{J}_k^T}$. In this case $circumellipse(\mathbf{J}_k \mathbf{V}_k^{q^l} \mathbf{J}_k^T \ , \ \mathbf{V}_k^I)$ simply returns $\mathbf{V}_k^I$. However, sometimes the relation between the “sizes” of $\mathcal{G}_{\mathbf{V}_k^I}$ and $\mathcal{G}_{\mathbf{J}_k \mathbf{V}_k^{q^l} \mathbf{J}_k^T}$ can be inverse, *e.g.* if the LOD hierarchy traversal is finished by some other criterion than the one used for depth-blurring.

Calculation of a Circumscribed Ellipse

Given two variance matrices

$$\mathbf{M} = \begin{pmatrix} A & B/2 \\ B/2 & C \end{pmatrix} \quad \text{and} \quad \mathbf{V}_k^I = \begin{pmatrix} R_k^I & 0 \\ 0 & R_k^I \end{pmatrix},$$

the task is to find the variance matrix $\mathbf{V}_k^{\text{diff}}$, whose conic matrix $(\mathbf{V}_k^{\text{diff}})^{-1}$ defines the smallest ellipse that encloses both ellipses $\mathcal{E}_{\mathbf{M}^{-1}}$ and $\mathcal{E}_{(\mathbf{V}_k^I)^{-1}}$, where $\mathcal{E}_{\mathbf{H}} = \{\mathbf{x} \mid \mathbf{x}^T \mathbf{H} \mathbf{x} = 1\}$.

Generally, if $\mathbf{H}$ is a conic matrix that defines an ellipse (*i.e.* $\mathbf{H}$ is positive definite) then $\mathbf{H}^{-1}$ is also a conic matrix defining an ellipse. The size of $\mathcal{E}_{\mathbf{H}^{-1}}$ is inversely proportional to the size of $\mathcal{E}_{\mathbf{H}}$ and their minor and major axes are swapped. This allows us to reformulate our problem as determining the largest ellipse enclosed by ellipses $\mathcal{E}_{\mathbf{M}}$ and $\mathcal{E}_{\mathbf{V}_k^I}$. Because $\mathcal{E}_{\mathbf{V}_k^I}$ is actually a circle we first rotate $\mathcal{E}_{\mathbf{M}}$ so that its minor and major axes are aligned with coordinate axes, we then solve the problem for the rotated ellipse and finally rotate the result back. The coefficients of the rotated $\mathcal{E}_{\mathbf{M}}$ are [27]:

$$A', C' = \frac{A + C \pm \sqrt{(A-C)^2 + B^2}}{2}, \quad B' = 0, \tag{14}$$

and the angle of rotation θ is given by $\tan 2\theta = \frac{B}{A-C}$. To rotate the ellipse back we use the following equations:

$$\begin{array}{l} \cos^2\theta = \frac{1}{2}\left(\frac{A-C}{\sqrt{(A-C)^2+B^2}}+1\right) \\ \sin 2\theta = \frac{B}{\sqrt{(A-C)^2+B^2}} \\ A'' = (A'-C')\cos^2\theta + C' \\ B'' = (A'-C')\sin 2\theta \\ C'' = (C'-A')\cos^2\theta + A'. \end{array} \tag{15}$$

The algorithm to find the elements A'', B'' and C'' of $\mathbf{V}_k^{\mathrm{diff}}$ is as follows:

```
**if** $A < C$ **then**   $\mathrm{swap}(A, C); \mathrm{swapped} := \mathrm{true};$
**else**   $\mathrm{swapped} := \mathrm{false};$   **end if**
$A', C' :=$ rotate with Equation (14)
$A' := \max\{A', R_k^I\}$
$C' := \max\{C', R_k^I\}$
$A'', B'', C'' :=$ rotate back with Equations (15)
**if** $\mathrm{swapped}$ **then**   $\mathrm{swap}(A'', C'')$   **end if**
```

5.4.3 Normalization

Since the resampling kernels are truncated to a finite support and the surface is sampled irregularly, the resampling kernels do not generally sum to 1 and the intensity of the rendered texture varies in screen-space. A per-pixel normalization has to be done after splatting all the points (Equation 1).

For DOF rendering we cannot do this post-normalization because we use the accumulated weights as the estimate for partial coverage in the same way as in EWA surface splatting for edge antialiasing (Section 3.3.3). In case of DOF rendering, however, this estimate has to be much more precise than for edge anti-aliasing. Motivated by Ren *et al.* [46] we perform a per-point normalization in the preprocessing step. We use the same algorithm to compute the per-splat weights (capturing the weights from rendered images) since this technique is easy to implement and works reasonably well.

Let us remind here that in EWA surface splatting the Gaussian resampling kernels are truncated to a finite support by a cutoff radius c. Unlike Ren *et al.* we do not bind the normalization to a particular choice of the cutoff radius. To compute the normalization we use a very large support of the reconstruction filters ($c = 3.5 - 4$) so that the influence of truncation becomes negligible. This allows us to use the normalized model for any value of c without having to re-normalize it. To take a smaller c into account we divide the weights during rendering by the compensation factor $1 - e^{-c}$ (Figure 19b), which makes every single truncated Gaussian always integrate to 1 and therefore keeps the sum of resampling kernels close to 1. For a visually pleasing DOF effect the value of c must be slightly higher than for surface splatting without DOF, we use $c = 2 - 3$.

5.4.4 Surface Reconstruction

To resolve visibility we use the extended z-buffering based on z-ranges as described in Section 3.3.3. The motivation to use this technique instead of Zwicker's z-offsets is that z-offsets need to compute depth for every fragment coming from splat rasterization. Since we enlarge the splats by applying the blur an extrapolation in depth could occur, which would lead to an incorrect blending of different surfaces.

5.4.5 Shading

Shading can be done per-splat, before the points are splatted, or per-pixel, after all the points have been splatted. We use per-splat shading and we interpolate colors in screen space. This is needed if view-dependent shading, such as specular highlights, is used. If we used normal interpolation and per-pixel shading, the highlights wouldn't appear blurred.

5.5 Results

We have implemented the described DOF rendering algorithm in a software EWA surface splatter. We use the A-buffer [10] for transparency and edge antialiasing. Figure 18 illustrates the DOF rendering with a semi-transparent surface. Figure 22 compares the results of our rendering algorithm (left column) with those of the multisampling algorithm [39] (right column) that is taken as a reference. The number of images averaged to produce the reference images was 200. From top to bottom, the aperture (*i.e.* the amount of blur) is increased. For flat objects such as the plane the difference is hardly perceptible. However, for complex objects like the lion, the algorithm produces some artifacts. They are mainly due to the incorrect pass/blend/fail decisions in the extended z-buffer (Section 3.3.3). Another reason is that the *shape* (unlike the texture) is not low-pass filtered for coarser levels of the LOD hierarchy and the surface sample positions are highly irregular but the algorithm is quite sensitive to the regularity of surface sample positions.

Data	Aperture	LOD	#FRAG	#PTS	time
Plane	0	-	5 685 $\times 10^3$	262 144	0.76 s
	0.5	YES	8 521 $\times 10^3$	178 696	0.97 s
	2	YES	7 246 $\times 10^3$	54 385	0.75 s
	0.5	NO	17 630 $\times 10^3$	262 144	1.79 s
	2	NO	196 752 $\times 10^3$	262 144	20.2 s
Lion	0	-	2 266 $\times 10^3$	81 458	0.43 s
	0.01	YES	4 036 $\times 10^3$	53 629	0.56 s
	0.04	YES	5 318 $\times 10^3$	17 271	0.56 s
	0.01	NO	7 771 $\times 10^3$	81 458	0.91 s
	0.04	NO	90 219 $\times 10^3$	81 458	8.93 s

Table 4: Rendering performance

The rendering performance is summarized in Table 4. It was measured for 512×512 frames, cutoff radius c was set to 2.5. The system configuration was a 1.4 GHz Pentium 4 with 512 MB RAM, GCC 3.1 compiler with optimization set to Pentium 4 architecture. The table shows the number of generated fragments (#FRAG), the number of points used for rendering (#PTS) and the rendering time (time) for objects in Figure 22 with varying apertures. The table also compares the DOF rendering speed with and without LOD. The rendering time is directly proportional to the number of fragments generated by rasterizing the resampling kernels, since the rendering pipeline is fill-limited. This is due to the fact that we use an A-buffer with linked lists of fragments that degrade cache performance significantly. The rendering times in the table also show that thanks to LOD the rendering speed is practically independent of the amount of depth-blur. The time for computing the reference images was 147 sec. (plane, 200 images) and 83 sec. (lion, 200 images).

5.6 Discussion

Let us now summarize the simplifying assumptions we have made in the algorithm.

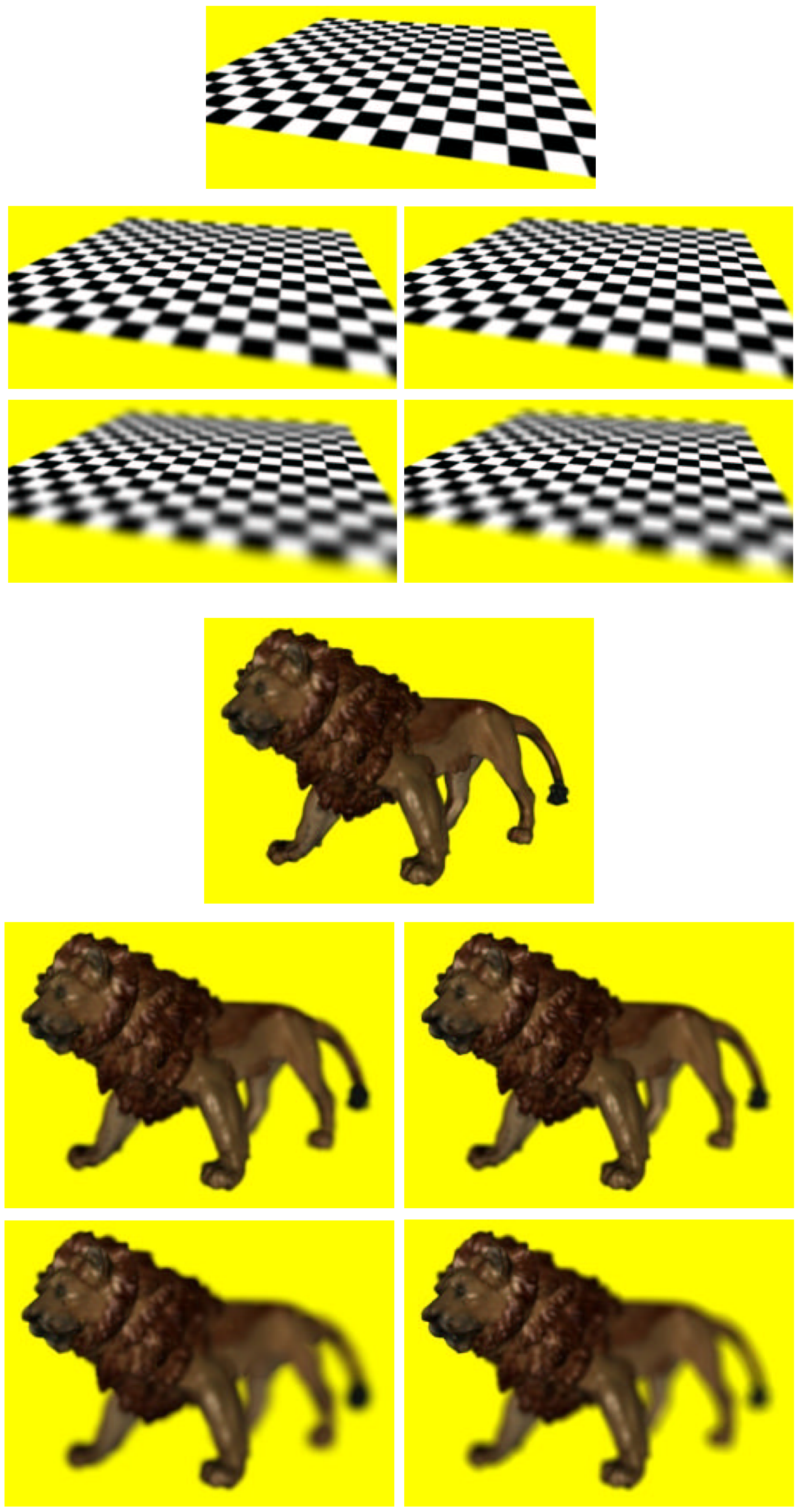

Figure 22: Comparison of our algorithm (left) with reference images created with multisampling algorithm (right). Centered images are without DOF.

- *Depth of splats is constant.* We use a constant depth to compute a single CoC radius for the whole splat. This is a mild assumption and does not cause any kind of artifacts.
- *Intensity distribution within CoC is a Gaussian.* This is a commonly made assumption in DOF rendering algorithms that aim at real-time. We derived an optimum Gaussian variance matrix that makes the Gaussian approximation as close as possible to the uniform intensity distribution. For an inexperienced viewer the effect is hardly noticeable. However, for an artist having experience with photography this might be disturbing. Since high-end image generation is not our aim, this is not a problem.
- *Splat contributions sum up to unity.* We use this assumption to estimate partial coverage. As the correctly estimated partial coverage is crucial for our algorithm, we perform a per-point pre-normalization and we normalize truncated Gaussians. As a result, the splats sum up to 1 ± 0.1, which is acceptable for our purposes. However, the pre-normalization works well only for models that are sampled without irregularities.
- *Extended z-buffer correctly solves visibility and reconstructs surfaces.* This is the most restricting assumption, which often fails and leads to severe artifacts. The reason lies in overly big z-ranges, which we are forced to assign to the blurred splats. They in effect lead to incorrect blending between separate surfaces. This artifact is most pronounced on the intersection of two surfaces.

5.7 Conclusions and Future Work

We have presented an efficient algorithm for DOF rendering for point-based objects which is a modification of the EWA surface splatting and requires minimal implementation efforts once the EWA splatter is ready. It renders DOF in presence of semi-transparent surfaces, handles the partial occlusion and does not suffer from intensity leakage. It is to our knowledge the first algorithm that uses LOD for DOF rendering and whose speed is independent of the amount of depth-blur.

The drawbacks of the algorithm are mainly high sensitivity to the regularity of sample positions on the surface of point-based objects and artifacts due to the incorrect surface reconstruction. In spite of the artifact the algorithm can be used as a preview tool to set camera parameters for high quality image generation. Since this is often done in a trial-and-error fashion, a fast preview tool is highly desirable.

In the future we would like to implement the DOF rendering algorithm for EWA volume rendering. DOF in volume rendering could be used to attract the viewers' attention by blurring unimportant features. We would also like to develop a specialized tool for the normalization of point-based objects.

6 References

[1] Anders Adamson and Marc Alexa. Ray tracing point set surfaces. In *Proceedings of Shape Modeling International 2003*, 2003. 5, 8, 23

[2] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. In *Proceedings of IEEE Visualization*, pages 21–28, 2001. 3, 4, 5, 8, 18, 19, 22, 23, 25

[3] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003. 2, 5, 22, 23, 25

[4] Marc Alexa, Tobias Klug, and Carsten Stoll. Direction fields over point-sampled geometry. *Journal of WSCG (WSCG '03 Proceedings)*, 11(1), February 2003. 5

[5] Nina Amenta, Marsahll Bern, and Manolis Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *Siggraph '98 Proceedings*, pages 415–421, 1998. 2, 4

[6] Jim Blinn. Return of the jaggy. *IEEE Computer Graphics and Applications*, 9(2), March 1989. Also published in [8]. 14

[7] Jim Blinn. What we need around here is more aliasing. *IEEE Computer Graphics and Applications*, 9(1), January 1989. Also published in [8]. 14

[8] Jim Blinn. *Jim Blinn's Corner: Dirty Pixels*. Morgan Kaufmann, August 1998. 14, 40

[9] Mario Botsch, Andreas Wiratanaya, and Leif Kobbelt. Efficient high quality rendering of point sampled geometry. In *Proceedings of the 13th Eurographics Workshop on Rendering*, 2002. 4, 6, 8, 22

[10] L. Carpenter. The A-buffer, an antialiased hidden surface method. In *Siggraph '84 Proceedings*, 1984. 3, 16, 37

[11] Jonathan C. Carr, Richard K. Beatson, Jon B. Cherrie, Tim J. Mitchell, W. Richard Fright, Bruces C. McCallum, and Tim R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Siggraph '01 Proceedings*, 2001. 2

[12] Baoquan Chen and Minh Xuan Nguyen. POP: A hybrid point and polygon rendering system for large data. In *IEEE Visualization*, 2001. 3, 4, 8, 22, 25

[13] Yong C. Chen. Lens effect on synthetic image generation based on light particle theory. *The Visual Computer*, 3(3), 1987. 30

[14] Liviu Coconu and Hans-Christian Hege. Hardware-accelerated point-based rendering of complex scenes. In *Proceedings of the 13th Eurographics Workshop on Rendering*, 2002. 4, 8

[15] Jonathan D. Cohen, Daniel G. Aliaga, and Weiqiang Zhang. Hybrid simplification: Combining multi-resolution polygon and point rendering. In *Proceedings of IEEE Visualization*, 2001. 3, 4, 8, 25

[16] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The Reyes image rendering architecture. *Computer Graphics*, 21(4):95–102, 1987. 3

[17] Oliver Deussen, Carsten Colditz, Marc Stamminger, and George Drettakis. Interactive visualization of complex plant ecosystems. In *IEEE Visualization '02 Proceedings*, 2002. 4, 8, 25, 26

[18] Tamal K. Dey and James Hudson. PMR: Point to mesh rendering, a feature-based approach. In *IEEE Visualization '02 Proceedings*, pages 155–162, 2002. 4, 8

[19] Krysztof Dudkiewicz. Real-time depth of field algorithm. In *Image Processing for Broadcast and Video Production*, 1994. 30

[20] Shachar Fleishman, Daniel Cohen-Or, Marc Alexa, and Claudion T. Silva. Progressive point set surfaces. *ACM Transactions on Graphics, accepted for publication*, 2002. 5, 22

[21] Andrew Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufman, 1995. 30

[22] Stephen J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Siggraph '96 Proceedings*, 1996. 7, 8

[23] Gevorg Grigoryan and Penny Rheingans. Probabilistic surfaces: Point based primitives to show surface uncertainty. In *IEEE Visualization '02 Proceedings*, 2002. 4, 8

[24] Markus Gross, Hanspeter Pfister, Marc Alexa, Mark Pauly, Marc Stamminger, and Matthias Zwicker. Point-based computer graphics. Eurographics '02 Tutorial T6, 2002. 2, 5, 14, 19, 24

[25] J. P. Grossman. Point sample rendering. Master's thesis, Department of Electrical Engineering and Computer Science, MIT, August 1998. 3, 6, 7, 8, 20, 25, 27

[26] J. P. Grossman and William J. Dally. Point sample rendering. In *Proceedings of the 9th Eurographics Workshop on Rendering*, pages 181–192, 1998. 3, 6, 7, 8, 20, 25, 27

[27] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, June 1989. 3, 14, 15, 34, 35

[28] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Siggraph '92 Proceedings*, pages 71–78, 1992. 2

[29] Aravind Kalaiah and Amitabh Varshney. Differential point rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering*, August 2001. 3, 4, 5, 6, 8, 17, 19, 25

[30] Aravind Kalaiah and Amitabh Varshney. Modeling and rendering of points with local geometry. *IEEE Transactions on Visualization and Computer Graphics (to appear)*, 2002. 5, 6, 8, 17, 19

[31] Jaroslav Křivánek, Jiří Žára, and Kadi Bouatouch. Fast depth of field rendering with surface splatting. In *Proceedings of Computer Graphics International 2003*, 2003. 29

[32] Marc Levoy and Turner Whitted. The use of points as a display primitive. Technical Report TR 85-022, University of North Carolina at Chapel Hill, 1985. 3, 8, 13, 19

[33] Lars Linsen. Point cloud representation. Technical Report No. 2001-3, Fakultät für Informatik, Universität Karlsruhe, 2001. 3, 4, 8, 17

[34] David Luebke and Benjamin Hallen. Perceptually driven interactive rendering. Technical Report #CS-2001-01, University of Virginia, 2001. 4, 8, 22

[35] S. D. Matthews. Analyzing and improving depth-of-field simulation in digital image synthesis. Master's thesis, University of California, Santa Cruz, December 1998. 30

[36] Wojciech Matusik, Hanspeter Pfister, Addy Ngan, Paul Beardsley, Remo Ziegler, and Leonard McMillan. Image-based 3D photography using opacity hulls. In *Siggraph '02 Proceedings*, 2002. 2, 24

[37] Wojciech Matusik, Hanspeter Pfister, Remo Ziegler, Addy Ngan, and Leonard McMillan. Acquisition and rendering of transparent and refractive objects. In *Eurographics '02 Proceedings*, 2002. 2, 19, 24

[38] Nelson Max and Keiichi Ohsaki. Rendering trees from precomputed z-buffer views. In *Proceedings of the 6th Eurographics Workshop on Rendering*, pages 45–54, June 1995. 7, 8

[39] Jackie Neider, Tom Davis, and M. Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison-Wesley, Reading Mass., first edition, 1993. 37

[40] Mark Pauly and Markus Gross. Spectral processing of point sampled geometry. In *Siggraph '01 Proceedings*, 2001. 3, 4, 8

[41] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *IEEE Visualization '02 Proceedings*, 2002. 5

[42] Mark Pauly, Leif P. Kobbelt, and Markus Gross. Multiresolution modeling of point-sampled geometry. Technical Report #378, Computer Science Department, ETH Zurich, September 2002. 5

[43] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Siggraph 2000 Proceedings*, pages 335–342, 2000. 3, 5, 6, 8, 20, 21, 22, 25, 27, 33

[44] Michael Potmesil and Indranil Chakravarty. A lens and aperture camera model for synthetic image generation. In *Siggraph '81 Proceedings*, 1981. 30, 32

[45] Jussi Räsänen. Surface splatting: Theory, extensions and implementation. Master's thesis, Dept. of Computer Science, Helsinki University of Technology, May 2002. 4, 8, 13, 17

[46] Liu Ren, Hanspeter Pfister, and Matthias Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *Eurographics '02 Proceedings*, 2002. 4, 8, 12, 14, 16, 18, 36

[47] Przemyslaw Rokita. Fast generation of depth of field effects in computer graphics. *Computers & Graphics*, 17(5), 1993. 30

[48] Przemyslaw Rokita. Generating depth-of-field effects in virtual reality applications. *IEEE Computer Graphics and Applications*, 16(2):18–21, 1996. 29, 30, 31

[49] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Siggraph 2000 Proceedings*, pages 343–352, 2000. 3, 8, 10, 11, 14, 18, 20, 21, 25, 33, 34

[50] Szymon Rusinkiewicz and Marc Levoy. Streaming QSplat: A viewer for networked visualization of large, dense models. In *Proceedings 2001 Symposium for Interactive 3D Graphics*, 2001. 4

[51] Gernot Schaufler and Henrik Wann Jensen. Ray tracing point sampled geometry. In *Proceedings of the 11th Eurographics Workshop on Rendering*, pages 319–328, 2000. 3, 8, 23

[52] Vincent Scheib, Jörg Haber, Ming C. Lin, and Hans-Peter Seidel. Efficient fitting and rendering of large scattered data sets using subdivision surfaces. In *Eurographics '02 Proceedings*, 2002. 2

[53] Jonathan W. Shade, Steven J. Gortler, Li-Wei He, and Richard Szeliski. Layered depth images. In *Siggraph '98 Proceedings*, pages 231–242, 1998. 25

[54] L. Shirman and S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. In *Eurographics '93 Proceedings*, 1993. 20, 22

[55] Marc Stamminger and George Drettakis. Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the 12th Eurographics Workshop on Rendering*, 2001. 3, 4, 8, 18, 19, 25, 26, 27

[56] Richard Szeliski and David Tonnesen. Surface modeling with oriented particle systems. In *Siggraph '92 Proceedings*, pages 185–194, 1992. 3, 6

[57] Ireneusz Tobor, Christophe Schlick, and Laurent Grisoni. Rendering by surfels. In *Proceedings of Graphicon*, 2000. 3, 8, 25

[58] Michael Wand, Matthias Fischer, Ingmar Peter, Friedhelm Meyer auf der Heide, and Wolfgang Straßer. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *Siggraph '01 Proceedings*, 2001. 3, 4, 8, 18, 19, 25, 26

[59] Michael Wand and Wolfgang Straßer. Multi-resolution rendering of complex animated scenes. In *Eurographics 2002 Proceedings*, 2002. 4, 8, 25, 26, 27

[60] Michael Wimmer, Peter Wonka, and François Sillion. Point-based impostors for real-time visualization. In *Proceedings of the 12th Eurographics Workshop on Rendering*, 2001. 4, 8, 25, 26

[61] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In *Siggraph '94 Proceedings*, pages 269–277, 1994. 3

[62] Stephan Würmlin, Edouard Lamboray, Oliver G. Staadt, and Markus H. Gross. 3D video recorder. In *Proceedings of Pacific Graphics 2002*, 2002. 4, 8, 22

[63] Hansong Zhang and Kenneth E. Hoff. Back-bace culling using normal masks. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, 1997. 21

[64] Matthias Zwicker, Markus Gross, and Hanspeter Pfister. A survey and classification of real time rendering methods. Technical Report 332, Computer Science Department, ETH Zurich, 1999. 2, 3

[65] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3D: An interactive system for point-based surface editing. In *Siggraph '02 Proceedings*, pages 322–329, 2002. 5, 8

[66] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. EWA volume splatting. In *Proceedings of IEEE Visualization*, 2001. 2, 3, 16

[67] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *Siggraph '01 Proceedings*, 2001. 3, 8, 12, 14, 16, 20, 30

[68] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. EWA splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002. 2, 5, 16, 19, 20

7 Dissertation thesis

Title: Points as Surface Modeling and Rendering Primitives

Abstract

In the field of point-based rendering we are going to study the problem of edge and texture aliasing. We will focus on using higher quality filters than those currently being used and on incorporation of proper treatment of colors in filtering algorithms. We want to asses rendering and anti-aliasing techniques with respect to human visual system in order to identify their suitability to various scenarios (high quality rendering, real-time rendering). Efficient hardware implementation for high quality point rendering will be devised. In point-based modeling we will focus on oriented particles and consider their usability to various modeling tasks such as multiresolution modeling or ray tracing.

Keywords

Point-based rendering and modeling, antialiasing, filtering, visual perception, graphics hardware, oriented particles, multiresolution modeling, ray tracing.

8 Publications of the author

[1] Jaroslav Křivánek, Jiří Žára, and Kadi Bouatouch. Fast Depth of Field Rendering with Surface Splatting. To appear in *Proceedings of Computer Graphics International*, 2003.

[2] Jaroslav Křivánek. Depth-of-Field Rendering for Point-Based Object. In *Proceedings of Workshop 2003*. Prague : CTU, 2003.

[3] Jaroslav Křivánek. Perceptually Driven Point Sample Rendering. In *Proceedings of Workshop 2002*. Prague : CTU, 2002, vol. A, p. 194-195. ISBN 80-01-02511-X.

[4] Jaroslav Křivánek and Pavel Slavík. Virtual Reconstruction of Celtic Oppidum Zavist Near Prague. In *Spring Conference on Computer Graphics SCCG 2002 - Conference Materials, Posters*. Bratislava : Comenius University, 2002, p. 56. ISSN 1335-5694.

[5] Jaroslav Křivánek. New Technologies as Presented by ATI. In Chip. 2001, vol. 11, no. 11, p. 54-56. ISSN 1210-0684. (in Czech).

[6] Jaroslav Křivánek. GeForce3: Hopefully not a Flaming Expectation. In Chip. 2001, vol. 11, no. 10, p. 62-65. ISSN 1210-0684. (in Czech).

[7] Jiří Žára and Jaroslav Křivánek. Graphics Performance Benchmarking Based on VRML Browsers. In *VRIC 2001 Proceedings*. Laval : ISTIA Innovation, 2001, p. 111-120. ISBN 2-9515730-0-6.

[8] Jaroslav Křivánek. Using Photon Maps and Irradiance Cache in Photorealistic Image Synthesis. In *Poster 2001*. Prague : CTU, Faculty of Electrical Engineering, 2001, p. IC22.

[9] Jaroslav Křivánek, Vojtěch Bubník. Ray Tracing with BSP and Rope Trees. In *Proceedings of the CESCG '00*, 2000.

A List of Abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
DOF	Depth of Field
BRDF	Bidirectional Reflectance Distribution Function
EWA	Elliptical Weighted Average
GPU	Graphics Processing Unit
LDI	Layered Depth Image
LDC	Layered Depth Cube
LOD	Level of Detail
MLS	Moving Least Squares
NURBS	Non-Uniform Rational B-Splines