Fast Depth of Field Rendering with Surface Splatting

Jaroslav Křivánek[1,2]
e-mail: xkrivanj@fel.cvut.cz

Jiří Žára[1]
e-mail: zara@fel.cvut.cz

Kadi Bouatouch[2]
e-mail: kadi@irisa.fr

[1]Department of Computer Science and Engineering,
Czech Technical University in Prague,
Karlovo náměstí 13, 121 35 Praha 2, Czech Republic

[2]IRISA - INRIA Rennes,
Campus de Beaulieu,
35042 Rennes Cedex, France

Abstract

We present a new fast algorithm for rendering the depth-of-field effect for point-based surfaces. The algorithm handles partial occlusion correctly, it does not suffer from intensity leakage and it renders depth-of-field in presence of transparent surfaces. The algorithm is new in that it exploits the level-of-detail to select the surface detail according to the amount of depth-blur applied. This makes the speed of the algorithm practically independent of the amount of depth-blur. The proposed algorithm is an extension of the Elliptical Weighted Average (EWA) surface splatting. We present a mathematical analysis that extends the screen space EWA surface splatting to handle depth-of-field rendering with level-of-detail, and we demonstrate the algorithm on example renderings.

Keywords point-based rendering, EWA surface splatting, depth-of-field, lens effect, level-of-detail, LOD.

1. Introduction

The ability to render the depth-of-field (DOF) effect is an important feature of any image synthesis algorithm. DOF makes the image look more natural and it is also an important depth cue that helps humans to perceive the spatial configuration of a scene [17]. The effect of DOF is that out-of-focus points in 3D space form circular patterns (circle of confusion, CoC) in the image plane. Algorithms for DOF rendering can be divided into two groups: post-filtering and multi-pass algorithms. The *post-filtering algorithms* work as follows. First the image is computed using a pinhole camera model, then the image is sent to the *focus processor* which turns every pixel into a CoC according to its depth. Potmesil and Chakravarty [13] have given formulas for radius of the CoC and described the light intensity distribution within the CoC by Lommel functions. Chen [2] simplify the distribution to uniform. The post-filtering algorithms suffer from *intensity leakage* (blurred background leaks into a focused object in the foreground) and they do not take into account the *partial occlusion* (visibility of objects change for different points on the lens). Some approaches to partially solve those problems are given in [9, 19]. Other post-filtering algorithms are presented in [16, 4, 17, 5, 10]. *Multi-pass algorithms* [11, 3] are more general, can handle the partial occlusion, but are slower.

We present a new fast algorithm which renders DOF for point-based surfaces. Our algorithm is similar to post-filtering algorithms, but unlike them it does not involve a separate focus processor. Instead, the individual points are blurred *before* they form the final image. This allows to handle the partial occlusion and avoids intensity leakage. The algorithm can also render DOF in presence of transparent surfaces (Figure 1). The presented algorithm builds on top of the Elliptical Weighted Average (EWA) surface splatting proposed by Zwicker *et al.* [20, 21]. The related work on splatting include [8, 15, 14]. Namely Räsänen [14] propose a point rendering pipeline that handles DOF rendering. His method is based on stochastic sampling, it requires high number of samples to produce noise-free images and thus it is somewhat slow.

The basic idea of our algorithm is to blur the individual splats by convolving them with a Gaussian low-pass filter instead of blurring the image itself. It essentially means that each splat is enlarged proportionally to the amount of depth-blur appertaining to its depth. To accelerate DOF rendering, we use coarser level-of-detail (LOD) for blurred surfaces. This makes the speed of the algorithm independent of the amount of depth-blur.

The contributions of this paper are consist in a mathematical analysis extending the EWA surface splatting with DOF rendering, an analysis allowing the use of the LOD as a means for DOF rendering, an implementation of the algorithm, and a discussion of practical implementation issues.

The paper is organized as follows: Section 2 reviews the screen-space EWA surface splatting algorithm, Section 3 contains the mathematical analysis of our DOF rendering algorithm, Section 4 presents its implementation, Section 5 gives the results, and Section 6 concludes the work.

Figure 1. Example of DOF rendering with semitransparent surface. Left: no DOF. Middle: DOF is on and the transparent mask is in focus. Right: the male body is in focus, the mask is out of focus.

2. Screen Space EWA Surface Splatting

This section briefly reviews the EWA surface splatting framework as described by Zwicker *et al.* [20].

The definition of the texture function on the surface of a point-based object is illustrated in Figure 2. 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 (we proceed with the discussion using a single channel w_k). Local surface parametrization is sufficient to define the texture function since the support of functions r_k is local. Given a point $\mathbf{Q}$ on the surface with local coordinates $\mathbf{u}$, the value of the continuous texture function is expressed as

$$f_c(\mathbf{u}) = \sum_{k\in\mathbb{N}} w_k r_k(\mathbf{u} - \mathbf{u}_k), \tag{1}$$

where $\mathbf{u}_k$ are the local coordinates of the point $\mathbf{P}_k$. The value $f_c(\mathbf{u})$ gives the color of the point $\mathbf{Q}$.

To **render a point-based object**, the texture function f_c is mapped to the screen-space with Heckbert's resampling framework [6]. It involves the following conceptual steps: first, the continuous texture function f_c is reconstructed from sample points using Equation (1), second, f_c is warped to screen-space using the affine approximation of the object-to-screen mapping $\mathbf{m}$, third, the warped f_c is convolved in screen-space with the prefilter h, yielding the band-limited output function $g_c(\mathbf{x})$, lastly g_c is sampled to produce alias-free pixel colors. Concatenating the first three steps, the output function g_c is

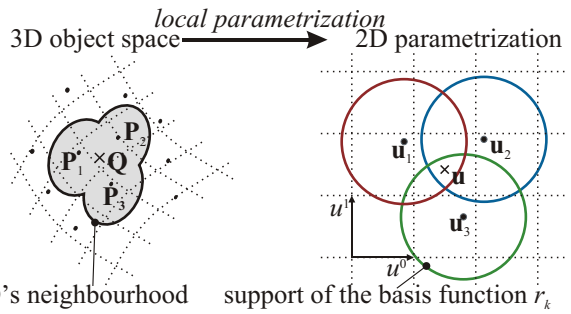

Figure 2. Texture function on the surface of a point-based object (after Zwicker et al. [20]).

$$g_c(\mathbf{x}) = \sum_{k\in\mathbb{N}} w_k \rho_k(\mathbf{x}), \tag{2}$$

$$\text{where} \quad \rho_k(\mathbf{x}) = (r'_k \otimes h)(\mathbf{x} - \mathbf{m}_{\mathbf{u}_k}(\mathbf{u}_k)), \tag{3}$$

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 $\mathbf{u}_k$. Function ρ_k is the warped filtered basis function r_k and is called the *resampling kernel*.

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 $\mathcal{G}_{\mathbf{V}}(\mathbf{x}) = \frac{1}{2\pi\sqrt{|\mathbf{V}|}} e^{-\frac{1}{2}\mathbf{x}^T\mathbf{V}^{-1}\mathbf{x}}$. Matrix $\mathbf{V}^{-1}$ is so called conic matrix and $\mathbf{x}^T\mathbf{V}^{-1}\mathbf{x} = const$ are the isocontours of the Gaussian $\mathcal{G}_{\mathbf{V}}$, that are ellipses iff $\mathbf{V}$ is positive definite [6]. The variance matrices for basis function r_k and the prefilter h are denoted $\mathbf{V}_k^r$ and $\mathbf{V}^h$ respectively. Usually $\mathbf{V}^h = \mathbf{I}$ (the identity matrix). With Gaussians, Equation (3) becomes

$$\rho_k(\mathbf{x}) = \frac{1}{|\mathbf{J}_k^{-1}|} \mathcal{G}_{\mathbf{J}_k\mathbf{V}_k^r\mathbf{J}_k^T + \mathbf{I}}(\mathbf{x} - \mathbf{m}(\mathbf{u}_k)), \tag{4}$$

where $\mathbf{J}_k$ is the Jacobian of the object-to-screen mapping $\mathbf{m}$ evaluated at $\mathbf{u}_k$. In this formulation ρ_k is a Gaussian and is called the *screen space EWA resampling kernel*. It has an infinite support in theory. In practice it is truncated and is evaluated only for 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*.

The surface splatting algorithm takes the points $\{\mathbf{P}_k\}$, for each point it computes the resampling kernel ρ_k, rasterizes it and accumulates the fragments in the accumulation buffer.

3. DOF in the EWA Splatting Framework

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

3.1. DOF 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 this equation using (2) we get

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

$$\text{where} \quad \rho_k^{\mathrm{dof}}(\mathbf{x}) = \int_{\mathbb{R}^2} I(\mathrm{coc}(\mathrm{z}(\zeta)), \mathbf{x}-\zeta)\rho_k(\zeta) d\zeta. \quad (5)$$

This means that we can get the depth-blurred screen space function g_c^{dof} by first depth-blurring the individual 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 (5) becomes the convolution

$$\rho_k^{\mathrm{dof}}(\mathbf{x}) = (I_{\mathrm{coc}(\mathrm{z}_k)} \otimes \rho_k)(\mathbf{x}). \quad (6)$$

For compatibility with the EWA framework we choose circular Gaussians for I. If we denote the variance matrix for $I_{\mathrm{coc}(\mathrm{z}_k)}$ by $\mathbf{V}_k^I$, then $I_{\mathrm{coc}_{z_k}} = \mathcal{G}_{\mathbf{V}_k^I}$. We now plug (4) into (6) and we get ρ_k^{dof} in the form

$$\rho_k^{\mathrm{dof}}(\mathbf{x}) = \frac{1}{|\mathbf{J}_k^{-1}|} \mathcal{G}_{\mathbf{J}_k \mathbf{V}_k^r \mathbf{J}_k^T + \mathbf{I} + \mathbf{V}_k^I}(\mathbf{x} - \mathbf{m}(\mathbf{u}_k)). \quad (7)$$

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 the whole image.

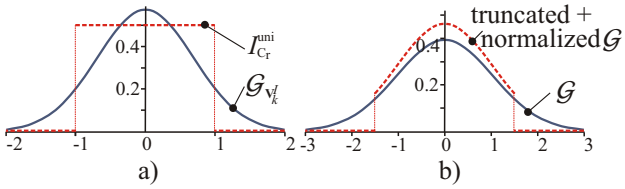

Figure 3. a) Gaussian approximation of the uniform intensity distribution. b) Normalization of a truncated Gaussian.

3.2. Variance matrix of the Intensity Distribution Function

Having the depth value z_k we compute the CoC radius C_r [7, 13]. Now we want to find such a variance matrix $\mathbf{V}_k^I$ that brings the Gaussian $\mathcal{G}_{\mathbf{V}_k^I}$ as close as possible (by the L_2 norm) to the uniform intensity distribution within the CoC of radius C_r (Figure 3a). The uniform distribution is $I_{C_r}^{\mathrm{uni}}(\mathbf{x}) = 1/\pi C_r^2$ if $\|\mathbf{x}\| < C_r$ and zero otherwise. Gaussian $\mathcal{G}_{\mathbf{V}_k^I}$ 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 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 $a = \frac{1}{2\ln 4} C_r^2$, yielding the variance matrix $\mathbf{V}_k^I = \begin{pmatrix} \frac{1}{2\ln 4}C_r^2 & 0 \\ 0 & \frac{1}{2\ln 4}C_r^2 \end{pmatrix}$. Why are we 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 function [13])? The reason is that the mathematical intractability of the Lommel intensity distribution function did not allow us to express a in a closed form.

3.3. DOF Rendering with Level-of-Detail

The DOF rendering without LOD is slow because of the high number of fragments generated by the rasterization of the blurred resampling kernels. To accelerate rendering we use coarser LOD to render blurred parts of the objects. This yields some blur in the image, since the texture in the LOD hierarchy is typically low-pass filtered on coarser levels [12, 18]. To steer precisely the amount of blur we need to quantify the blurring produced by the coarser LOD and eventually add an additional filtering in screen-space. To express those intuitions more rigorously, we slightly change

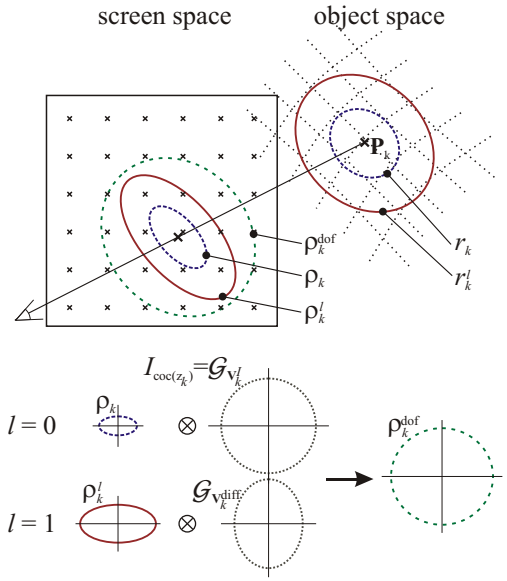

Figure 4. DOF rendering with different LODs.

the definition of the texture function (1) to take into account the LOD hierarchy.

Extended Surface Texture Definition. We assume for this discussion that there are distinct levels in the LOD hierarchy identified by integers, where level 0 are leaves. The continuous texture function f_c^l at level l is represented by basis functions r_k^l. This representation is created by low-pass filtering and subsampling 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, *i.e.* $r_k^l(\mathbf{u}) = (r_k \otimes q_k^l)(\mathbf{u})$. The continuous texture function f_c^l is then

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

Application to Depth-Blurring. Now we focus on transforming the filters q_k^l to the screen-space and using them for depth-blurring. We assume that the basis functions r_k^l and the low-pass filters q_k^l are Gaussians, $r_k^l = \mathcal{G}_{\mathbf{V}_k^{r^l}}$ and $q_k^l = \mathcal{G}_{\mathbf{V}_k^{q^l}}$. Recall that $\mathbf{V}_k^r$ is the variance matrix of 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

$$\rho_k^l(\mathbf{x}) = \frac{1}{|\mathbf{J}_k^{-1}|}\mathcal{G}_{\mathbf{J}_k(\mathbf{V}_k^r+\mathbf{V}_k^{q^l})\mathbf{J}_k^T+\mathbf{I}}(\mathbf{x}-\mathbf{m}(\mathbf{u}_k)). \qquad (8)$$

The variance matrix of this Gaussian is $\mathbf{V}_k^l = \mathbf{J}_k(\mathbf{V}_k^r + \mathbf{V}_k^{q^l})\mathbf{J}_k^T + \mathbf{I} = \mathbf{J}_k\mathbf{V}_k^r\mathbf{J}_k^T + \mathbf{I} + \mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T$.

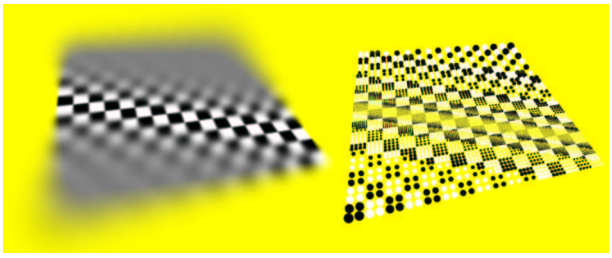

Figure 5. Example of LOD selection for DOF rendering.

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 (7), 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 $\mathcal{G}_{\mathbf{V}_k^{\text{diff}}}$, where the variance matrix $\mathbf{V}_k^{\text{diff}}$ is given by $\mathbf{V}_k^I = \mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T + \mathbf{V}_k^{\text{diff}}$. Convolution with Gaussian $\mathcal{G}_{\mathbf{V}_k^{\text{diff}}}$ can be regarded as an additional blurring needed to produce the required screen-space blur after we have selected the hierarchy level l.

The idea of using the LOD to speed-up depth-blurring is to select such a hierarchy level l that Gaussian $\mathcal{G}_{\mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T}$ is "smaller" than Gaussian $\mathcal{G}_{\mathbf{V}_k^I}$ but Gaussian $\mathcal{G}_{\mathbf{V}_k^{\text{diff}}}$ 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 $\mathcal{G}_{\mathbf{V}_k^{\text{diff}}}$ is very small and therefore the blurring does not significantly slow down the rendering. This concept is illustrated in Figure 4.

4. Implementation

In this section we describe the implementation of the DOF rendering in the screen space EWA surface splatter.

DOF rendering without LOD. For every splat we compute the matrix $\mathbf{V}_k^I$ (Section 3.2) and add it to the variance matrix of ρ_k (Equation 4) to get the blurred resampling kernel ρ_k^{dof} (Equation 7). It is then rasterized as in normal surface splatting.

LOD selection for DOF rendering. We adopted the QSplat hierarchy [18] and add one new criterion to stop the traversal. We stop if the projected size of the node is smaller than the CoC radius for that node. Figure 5 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.

LOD augmented DOF rendering. For each splatted point $\mathbf{P}_k^l$ we determine the low-pass filter q_k^l (it is given by the hierarchy level l) and we then compute the matrix $\mathbf{V}_k^{\text{diff}}$ for additional screen-space blurring with following steps:

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

$\mathbf{W}$ is the resulting matrix of the resampling kernel. The function $circumellipse(\mathbf{A}, \mathbf{B})$ returns the variance matrix for an ellipse that circumscribes ellipses defined by conic matrices $\mathbf{A}^{-1}$ and $\mathbf{B}^{-1}$ (implementation is given in [7]). 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 $circumellipse()$ 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.* example if the LOD hierarchy traversal is finished by some other criterion than the one used for depth-blurring.

Surface Reconstruction. We use A-buffer [1] to accumulate splats as described by Zwicker *et al.* [20]. They use an extended depth test based on *z-threshold* — if the depth of two fragments is smaller than a threshold, they are assumed to come from a single surface and they are merged, otherwise they are kept separated in the A-buffer. Z-thresholding is prone to errors since the depth values gets extrapolated if splat support is enlarged by some screen space filtering (*e.g.* prefiltering, depth of field filtering). We use a depth test based on *z-ranges* [8, 14] that is more robust especially near silhouettes.

Normalization. The splat weights generally do not sum to 1 everywhere in screen-space which leads to varying texture intensity in the resulting image. Zwicker *et al.* [20] solves this by a per-pixel normalization after splatting all points. We cannot use this post-normalization, since we use the weights as an estimate for partial coverage which needs to be precise in case of DOF rendering. We perform a per-point normalization in preprocess as described by Ren *et al.* [15]. Unlike Ren *et al.* we do not bind the normalization to a particular choice of the cutoff radius c. To compute the normalization we use a 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 during rendering we divide the weights by the compensation factor $1 - e^{-c}$ (Figure 3b) which makes every truncated splat integrate to 1 and keeps the sum of splats 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 $c = 2 - 3$.

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

Table 1. Rendering performance

Shading. Shading can be done per-splat or per-pixel [14]. We use per-splat shading. This is needed if view-dependent shading, such as specular highlights, is used. If we used per-pixel shading, the highlights wouldn't appear blurred.

5. Results

We have implemented the DOF rendering algorithm in a software EWA surface splatter. Figure 1 illustrates the DOF rendering with a semitransparent surfaces. Figure 6 compares the results of our rendering algorithm (left column) with those of the multisampling algorithm [11] (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 is increased. For flat objects like the plane the difference is hardly perceptible, however, for complex objects like the lion our algorithm produces some artifacts. They are mainly due to the incorrect merge/separate decisions in the A-buffer. Another reason is the irregularity of points on coarser LOD levels. Since we blur the splats individually and the surface reconstruction is applied *after* blurring, we avoid intensity leakage and we can handle partial occlusion. The A-buffer moreover allows for transparent surfaces. For surfaces that are close to each other or for intersecting surfaces, artifacts cannot be avoided, because of incorrect merge/separate decisions.

Rendering performance is summarized in Table 1. It was measured for 512×512 frames with a cutoff radius $c = 2.5$ on a 1.4 GHz Pentium 4, 512 MB RAM, GCC 3.1 compiler with optimization set to Pentium 4 architecture. The table shows the number of generated fragments (#FRAG - in thousands), number of points (#PTS) and rendering time for objects in Figure 6. The table also compares the DOF rendering speed with and without LOD. The rendering time is determined by the number of fragments, since the rendering pipeline is fill-limited. The timings also show that thanks to LOD the 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).

6. 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 correctly in presence of semitransparent 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 main drawbacks of the algorithm are high sensitivity to the regularity of sample positions on the surface of point-based objects and occasional artifacts due to the incorrect surface reconstruction in the A-buffer.

In the future we would like to implement the DOF rendering algorithm for EWA volume rendering. We would also like to develop a tool for the normalization of point-based objects.

Acknowledgments

This work has been supported by the grant number 2159/2002 from the Grant Agency of the Ministry of Education of the Czech Republic. Thanks to Jiří Bittner and Vlastimil Havran for proofreading the paper and providing many hints on how to improve it.

References

[1] L. Carpenter. The A-buffer, an antialiased hidden surface method. In *Siggraph 1984 Proceedings*, 1984.

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

[3] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *Siggraph 1984 Proceedings*, 1984.

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

[5] P. Fearing. Importance ordering for real-time depth of field. In *Proceedings of the Third International Conference on Computer Science*, pages 372–380, 1996.

[6] P. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, University of California, 1989.

[7] J. Křivánek and J. Žára. Rendering depth-of-field with surface splatting. Technical Report DC-2003-02, Dept. of Computer Science, CTU Prague, 2003.

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

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

[10] J. D. Mulder and R. van Liere. Fast perception-based depth of field rendering. In *VRST 2000*, pages 129–133, 2000.

[11] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison-Wesley, Reading Mass., first edition, 1993.

[12] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Siggraph 2000 Proceedings*, pages 335–342, 2000.

[13] M. Potmesil and I. Chakravarty. A lens and aperture camera model for synthetic image generation. In *Siggraph '81 Proceedings*, 1981.

[14] J. Räsänen. Surface splatting: Theory, extensions and implementation. Master's thesis, Helsinki University of Technology, May 2002.

[15] L. Ren, H. Pfister, and M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *EUROGRAPHICS 2002 Proceedings*, 2002.

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

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

[18] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *Siggraph 2000 Proceedings*, pages 343–352, 2000.

[19] M. Shinya. Post-filtering for depth of field simulation with ray distribution buffer. In *Graphics Interface*, 1994.

[20] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *Siggraph 2001 Proceedings*, 2001.

[21] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. EWA splatting. *IEEE TVCG*, 8(3):223–238, 2002.

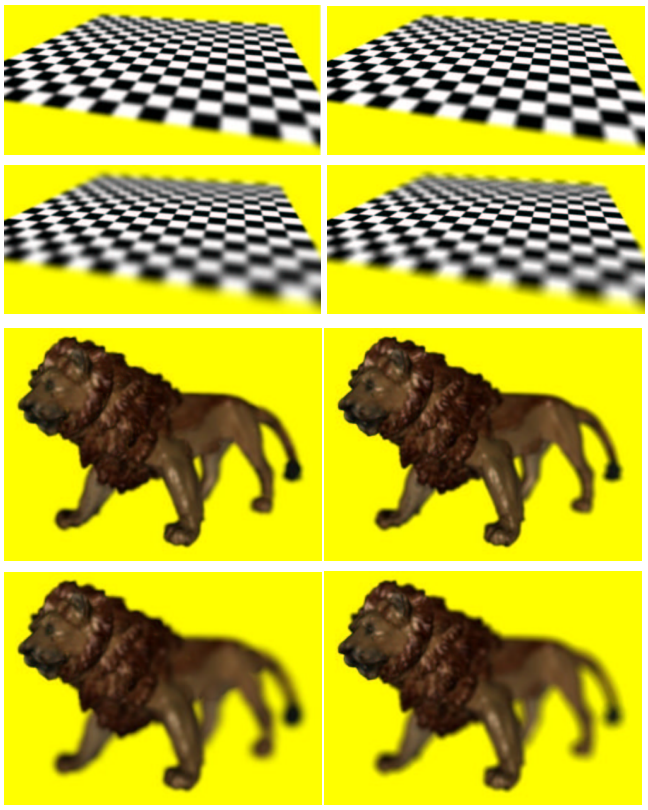

Figure 6. Comparison of our algorithm (left) with reference solution (right).