# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## Rendering Depth-of-Field with Surface Splatting

Jaroslav Křivánek, JiříŽára

Research Report DC-2003-02


**Czech Technical University**
**Prague**
**Czech Republic**

# CZECH TECHNICAL UNIVERSITY

## Faculty of Electrical Engineering

### Department of Computer Science and Engineering

Karlovo nám. 13
121 35 PRAHA 2
CZECH REPUBLIC

Tel : (+420) 224357470, 224357366
Fax : (+420) 224923325
E-mail : OFFICE@CS.FELK.CVUT.CZ


## LIST OF RECENTLY PUBLISHED REPORTS

(limited number of copies is available at departmental library,
where a complete list of published reports is also available )

**Holub, J., Šimánek, M. (eds.):** *Proceedings of the Prague Stringology Club Workshop '98*,
Collaborative Report DC–98–06, July 1998.

**Šoch, M., Tvrdík, P.:** *Time-Optimal Gossip in Noncombining 2-D Tori with Constant Buffers*,
Research Report DC–98–07, October 1998 (electronic version only).

**Hlavička, J., Novák, O.:** *Methods of Pseudoexhaustive Test Pattern Generation*,
Research Report DC–98–08, October 1998.

**Jelínek, I., Náplava, P.:** *Annual Report 1998*,
Annual Report DC–99–00, January 1999.

**Janoušek, J.:** *One–Pass Translations Generated by Unambiguous Translation Grammars with LR Input Grammars*,
Research Report DC–99–01, January 1999.

**Bělaška, P., Müller, K.:** *Attribute-Directed Bottom-Up Parsing*,
Research Report DC–99–02, January 1999.

**Šoch, M., Tvrdík, P.:** *Bufferless gossip of large packtes in noncombining 2-D tori*,
Research Report DC–99–03, April 1999 (electronic version only).

. **Šoch, M., Tvrdík, P.:** *Time–optimal gossip in noncombining 3-D tori with constant buffers*,
Research Report DC–99–04, April 1999 (electronic version only).

**Holub, J., Šimánek, M. (eds.):** *Proceedings of the Prague Stringology Club Workshop '99*,
Collaborative Report DC–99–05, June 1999.

**Trdlička, J., Tvrdík, P. (eds.):** *Embedding of Complete k-ary Trees into Wormhole 2-D Meshes*,
Research Report DC–99–06, December 1999 (electronic version only).

**Trdlička, J., Tvrdík, P. (eds.):** *Embedding of Complete k-ary Trees into Wormhole 3-D Meshes*,
Research Report DC–99–07, December 1999 (electronic version only).

**Trdlička, J., Tvrdík, P. (eds.):** *Optimal k-ary Divide&Conquer Computations on Wormhole 2-D and 3-D Meshes*,
Research Report DC–99–08, December 1999.

**Hlavička, J., Racek, S., Herout, P.:** *C-Sim v.4.1*,
Research Report DC–99–09, December 1999.

**Salinger, P., Tvrdík, P.:** *Optimal broadcasting and gossiping in one–port meshes of trees with distance–insensitive routing*,
Research Report DC–2000–01, January 2000 (electronic version only).

**Náplava, P., Hlavička, J., Jelínek, I., (eds.):** *Annual Report 1999*,
Annual Report DC–2000–02, January 2000.

**Balík, M., Šimánek, M., (eds.):** *Proceedings of the Prague Stringology Club Workshop '2000*,
Annual Report DC–2000–03, September 2000.

**Hlavička, J.:** *Built-in Self-Test in Circuits with more than 10 Million Transistors*,
Research Report DC–2000–04, November 2000.

**Náplava, P., Hlavička, J., Jelínek, I., (eds.):** *Annual Report 2000*,
Annual Report DC–2001–01, January 2001.

**Salinger, P., Tvrdík, P.:** *Broadcasting in all–output–port meshes of trees with distance–insensitive routing*,
Research Report DC–2001–02, January 2001 (electronic version only).

**Salinger, P., Tvrdík, P.:** *All–to–all Scatter in de Brujin and Kautz Networks*,
Research Report DC–2001–03, March 2001 (electronic version only).

**Valenta, M.:** *DATESO '01, Proceedings of Workshop on Databases, Texts, Specifications, and Objects*,
Research Report DC–2001–04, June 2001.

**Fišer, P., Hlavička, J.:** *BOOM – a Boolean Minimizer*,
Research Report DC–2001–05, June 2001.

**Balík, M., Šimánek, M., (eds.):** *Proceedings of the Prague Stringology Conference '01*,
Annual Report DC–2001–06, September 2001.

**Salinger, P., Tvrdík, P.:** *Broadcasting in all–output–port cube–connected–cycles*,
Research Report DC–2001–07, October 2001 (electronic version only).

**Havran, V.:** *Průvodce doktorského studia na katedře počítačů (K336)*,
Report DC–2001–08, December 2001.

**Bečvář, M., Daněk, M., Elshafey, Khaled A.M., Hlavička, J., Schmidt, J.:** *Architecture Acceleration using FPGAs*,
Research Report DC–2001–09, December 2001.

**Náplava, P., Hlavička, J., Jelínek, I., (eds.):** *Annual Report 2001*,
Annual Report DC–2002–01, January 2002.

**Troníček, Z.:** *Common Subsequence Automaton*,
Research Report DC–2002–02, April 2002. (electronic version only).

**Šimánek, M., (ed.):** *The Prague Stringology Club Conference 2002*,
Annual Report DC–2002–03, September 2002.

**Náplava, P., Jelínek, I., (eds.):** *Annual Report 2002*,
Annual Report DC–2003–01, January 2003.

# Rendering Depth-of-Field with Surface Splatting

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

Jiří Žára[1]
e-mail: zara@fel.cvut.cz

[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 is able to handle partial occlusion correctly, it does not suffer from intensity leakage and it is also capable of depth-of-field rendering in presence of transparent surfaces. The algorithm is new in that it exploits the level-of-detail paradigm 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 the depth-of-field rendering, we modify the definition of surface texture to take the level-of-detail into account, allowing us to use the level-of-detail for depth-of-field rendering, and we demonstrate the algorithm on example renderings of point-based objects.*

**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 appear more natural since the 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 [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. The problems contributing to the complexity of DOF rendering are the *partial occlusion* (visibility of objects changes 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.

We present a new fast algorithm which renders DOF for point-based surfaces. Since our algorithm does not decouple visibility calculations from DOF rendering, it handles the partial occlusion correctly and it does not suffer from intensity leakage. The algorithm can also render DOF in presence of transparent surfaces (Figure 1). The speed of the algorithm is practically independent of the amount of depth-blur, since it profits from the level-of-detail (LOD) to select coarser representation for highly blurred surfaces. The presented algorithm builds on top of the Elliptical Weighted Average (EWA) surface splatting framework proposed by Zwicker *et al.* [22, 23] and as such it aims at rendering point-based surfaces that are establishing as an important auxiliary primitive in rendering systems.

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

The rest of the paper is organized as follows: Section 2 draws the main ideas of the DOF rendering algorithm, Section 3 presents related work, Section 4 describes the camera model used for DOF rendering and reviews the screen-space EWA surface splatting algorithm. From Section 5 on, our algorithm is discussed: Section 5 contains the mathematical analysis of our DOF rendering algorithm, Section 6 presents its implementation, Section 7 gives the results, Section 8 discusses the simplifying assumptions we have made in the algorithm and their consequences, and Section 9 concludes the work.

## 2. Outline of the Algorithm

The basic idea is to blur the individual splats *before* they make up the image instead of blurring the image itself. We think of DOF rendering as filtering (or blurring) the image

**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.**

with a spatially variant low-pass filter. In the case of surface splatting, the image is formed by summing the contributions from the splats (called resampling kernels in the context of EWA surface splatting). 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 5.1.

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 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. Related Work

**Depth-of-Field Rendering** Thorough discussion of the DOF rendering algorithms can be found in [9]. We divide the algorithms for DOF rendering into two groups: *post-filtering* algorithms and *multi-pass* algorithms.

**Post-filtering algorithms** work 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* which 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 [13] were the first to present a DOF rendering algorithm. They have given the formulas to compute the radius of the CoC and described the intensity distribution within the CoC by Lommel functions. Chen [2] propose to simplify the intensity distribution to uniform. Rokita [16, 17] and Dudkiewicz [4] use multiple passes of hardware convolution filters to create the DOF effects. The post-filtering algorithms listed so far suffer from intensity leakage and they do not take the partial occlusion into account. They also produce undervalued pixel intensities near the boundaries of two differently focused objects. Those effects are described by Matthews [9], who also proposes his approach to partially solve them. Another approach that solves the partial occlusion is Shinya's ray-distribution buffer [20] which has unfortunately very high computational and memory overhead. Scofield [19] sorts objects to be rendered according to their depth, renders them independently to separate images, filters the images and combines them into a single final image. Fearing [5] use frame-to-frame coherence to avoid the DOF computation for some pixels. Mulder and van Liere [10] combine two hardware based algorithms to render DOF in virtual reality applications. The accurate and slower algorithm is used in areas near the gaze direction, whereas less accurate but fast algorithm is used in periphery.

**Multi-pass algorithms** do not decouple visibility from DOF rendering, therefore they can handle the partial occlusion correctly, however at a higher computational cost. The method described in [11] renders DOF by taking multiple pinhole camera renderings of the same scene, with viewpoints evenly distributed on the lens surface, while preserv-

ing a common plane in focus. In distributed ray tracing [3], multiple primary rays are cast for each pixel that originate from different locations of the lens.

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.

**Point-Based Rendering** Different approaches exist to render points as a continuous surface. One of them, which is particularly important for us, is *splatting*. A single point is rendered as multiple pixels and the colors of the pixels are weighted averages of contributions from different points. This approach was used by Levoy and Whitted in [8]. Zwicker *et al.* [22] extended it to handle the anisotropic texture filtering and call their method the surface splatting. The surface splatting is the basis for the work we present here. The same authors then extended the surface splatting to volume rendering [21] and presented both algorithms in an unified framework in [23]. Ren *et al.* [15] implemented the surface splatting in hardware. The surface splatting algorithm is thoroughly discussed by Räsänen [14], who also proposes a point rendering pipeline that handles DOF rendering. His algorithm is based on stochastic sampling of the resampling kernel. This method requires high number of samples to produce noise-free images and thus it is somewhat slow.

## 4. Preliminaries

### 4.1. 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 [6, 13, 17]. 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

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

where

$$V_u = \frac{FU}{U-F},\ U > F; \quad V_p = \frac{FP}{P-F},\ P > F. \tag{2}$$

$V_p$ 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 [17]).

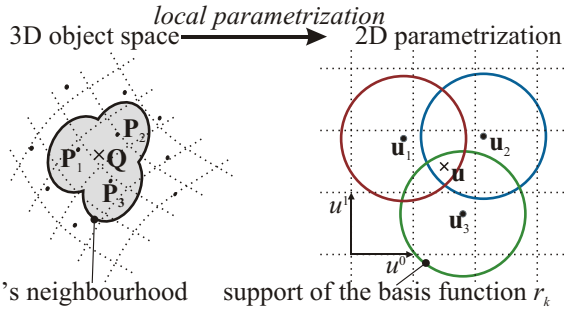


**Figure 2. Texture function on the surface of a point-based object (after Zwicker et al. [22]).**

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

$$C_r = C\frac{W}{R-L}\frac{N}{V_p} = C\frac{H}{T-B}\frac{N}{V_p}. \tag{3}$$

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 (1) gives the radius of CoC on the retina. The CoC radius on the display screen in pixels is

$$C_r = C\frac{d_s}{V_p}R, \tag{4}$$

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

### 4.2. Screen Space EWA Surface Splatting

This section briefly reviews the screen space EWA surface splatting algorithm as described by Zwicker *et al.* [22].

**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. 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. Given a point $\mathbf{Q}$ on the surface with local coordinates $\mathbf{u}$, the value of the continuous texture function is expressed as

$$f_c(\mathbf{u}) = \sum_{k \in \mathbb{N}} w_k r_k(\mathbf{u} - \mathbf{u}_k), \tag{5}$$

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** whose texture is defined by Equation (5), the texture function $f_c$ has to be mapped to the screen-space. Heckbert's resampling framework [7] 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 (5), 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

$$g_c(\mathbf{x}) = \sum_{k \in \mathbb{N}} w_k \rho_k(\mathbf{x}), \tag{6}$$

where

$$\rho_k(\mathbf{x}) = (r'_k \otimes h)(\mathbf{x} - \mathbf{m}_{\mathbf{u}_k}(\mathbf{u}_k)), \tag{7}$$

$r'_k$ 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 $\rho_k$ is the warped filtered basis function $r_k$ and is called the *resampling kernel*. Equation (6) 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 in screen space.

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 $\mathcal{G}_{\mathbf{V}}(\mathbf{x}) = \frac{1}{2\pi\sqrt{|\mathbf{V}|}} e^{-\frac{1}{2}\mathbf{x}^T\mathbf{V}^{-1}\mathbf{x}}$, where $|\mathbf{V}|$ is the determinant of $\mathbf{V}$. Matrix $\mathbf{V}^{-1}$ is so called conic matrix and $\mathbf{x}^T\mathbf{V}^{-1}\mathbf{x} = c$ are the isocontours of the Gaussian $\mathcal{G}_{\mathbf{V}}$, that are ellipses iff $\mathbf{V}$ is positive definite [7].

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 (7) becomes

$$\rho_k(\mathbf{x}) = \frac{1}{|\mathbf{J}_k^{-1}|} \mathcal{G}_{\mathbf{J}_k\mathbf{V}^r_k\mathbf{J}^T_k + \mathbf{I}}(\mathbf{x} - \mathbf{m}(\mathbf{u}_k)), \tag{8}$$

where $\mathbf{J}_k$ is the Jacobian of the object-to-screen mapping $\mathbf{m}$ evaluated at $\mathbf{u}_k$. In this formulation $\rho_k(\mathbf{x})$ is a Gaussian and is called the *screen space EWA resampling kernel*. Informally, this formula gives the shape and "alpha mask" of a splat for point $\mathbf{P}_k$.

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

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

# 5. 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.

## 5.1. Depth-of-Field rendering as a resampling kernel convolution

Neglecting the occlusion we can express the depth-blurred continuous screen space signal $g_c^{\mathrm{dof}}$ 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 (6) we get

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

where

$$\rho_k^{\mathrm{dof}}(\mathbf{x}) = \int_{\mathbb{R}^2} I(\mathrm{coc}(\mathrm{z}(\zeta)), \mathbf{x} - \zeta)\rho_k(\zeta) d\zeta. \tag{9}$$

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

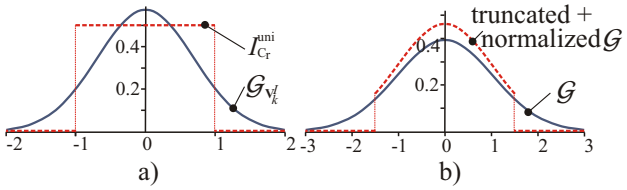


**Figure 3. a) Gaussian approximation of the uniform intensity distribution. b) Normalization of a truncated Gaussian.**

We assume that the depth $z(\mathbf{x})$ does not change within the support of $\rho_k$ 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 $I(\mathrm{coc}(z(\zeta)), \mathbf{x}-\zeta)$ can be replaced by the spatially invariant function $I_{\mathrm{coc}(z_k)}(\mathbf{x}-\zeta)$ and Equation (9) becomes the convolution

$$\rho_k^{\mathrm{dof}}(\mathbf{x}) = (I_{\mathrm{coc}(z_k)} \otimes \rho_k)(\mathbf{x}). \qquad (10)$$

For compatibility with the EWA framework, we choose circular Gaussians as the intensity distribution function $I$. If we denote the variance matrix for $I_{\mathrm{coc}(z_k)}$ by $\mathbf{V}_k^I$, then $I_{\mathrm{coc}_{z_k}} = \mathcal{G}_{\mathbf{V}_k^I}$. We now plug (8) into (10) and we get $\rho_k^{\mathrm{dof}}$ in the form

$$\rho_k^{\mathrm{dof}}(\mathbf{x}) = \frac{1}{|\mathbf{J}_k^{-1}|}\mathcal{G}_{\mathbf{J}_k\mathbf{V}_k^r\mathbf{J}_k^T+\mathbf{I}+\mathbf{V}_k^I}(\mathbf{x}-\mathbf{m}(\mathbf{u}_k)). \qquad (11)$$

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 $\rho_k$. 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.

## 5.2 Variance matrix of the Intensity Distribution Function

Having the depth value $z_k$, we compute the CoC radius $C_r$ using Equation (3) or (4). 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 3a). 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 [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.

## 5.3. DOF Rendering with Level-of-Detail

The DOF rendering algorithm as presented so far would be very slow (see timings in Section 7) because of the high number of fragments generated by the rasterization of the blurred resampling kernels. The LOD hierarchies such as those used in [12, 18] typically low-pass filter the texture for coarser levels. We observe that this low-pass filtering can be considered as blurring — if we choose a 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 (5) 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

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

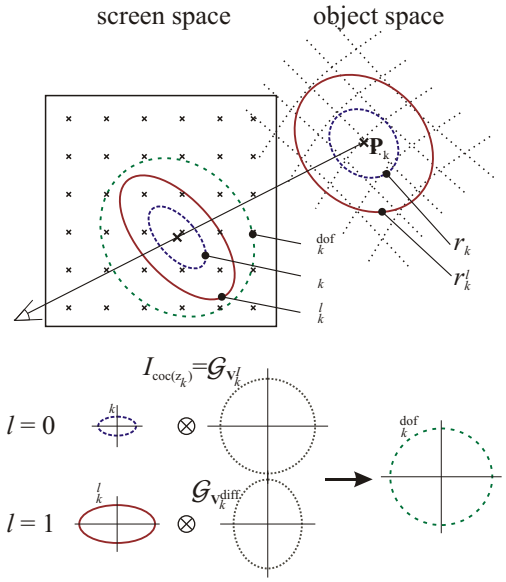


**Figure 4. DOF rendering with different LODs.**

**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 $\rho_k^l$ for the basis function $r_k^l$ is

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

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 $\rho_k^l$ to be the resampling kernel $\rho_k$ *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 (11), we see that to get the blurred resampling kernel $\rho_k^{\mathrm{dof}}$ from $\rho_k$, $\rho_k$ has to be convolved with the Gaussian $\mathcal{G}_{\mathbf{V}_k^I}$. Thus, to get $\rho_k^{\mathrm{dof}}$ from $\rho_k^l$, we have to convolve $\rho_k^l$ 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 $l$.

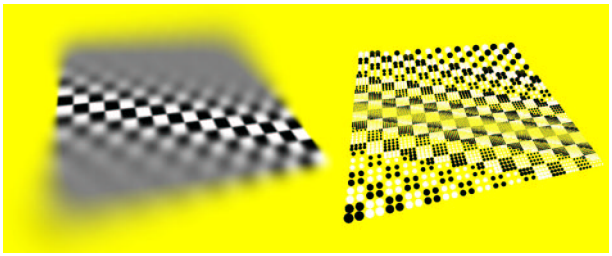

**Figure 5. Example of LOD selection for DOF rendering.**

Since $\mathbf{V}_k^{\mathrm{diff}}$ is a variance matrix of an elliptical Gaussian, it must be positive definite [7]. 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 $\{\mathbf{x} \mid \mathbf{x}^T(\mathbf{J}_k\mathbf{V}_k^{q^l}\mathbf{J}_k^T)^{-1}\mathbf{x} \le 1\}$ is a subset of the elliptical area $\{\mathbf{x} \mid \mathbf{x}^T(\mathbf{V}_k^I)^{-1}\mathbf{x} \le 1\}$).

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 $\mathcal{G}_{\mathbf{V}_k^{\mathrm{diff}}}$ is very small and therefore the blurring does not significantly slow down the rendering. This concept is illustrated in Figure 4.

## 6. Implementation

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

### 6.1. DOF rendering without LOD

For every point $\mathbf{P}_k$ being splatted we compute the variance matrix $\mathbf{V}_k^I$ (Section 5.2) and add it to the variance matrix of $\rho_k$ (Equation 8) to get the blurred resampling kernel $\rho_k^{\mathrm{dof}}$ (Equation 11). It is then rasterized as in normal surface splatting.

### 6.2. DOF rendering with LOD

**LOD Selection** We adopted the QSplat multiresolution hierarchy [18] 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 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.

**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 $\mathbf{V}_k^{\mathrm{diff}}$ for additional screen-space blurring. We use the following computations:

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

$\mathbf{W}$ 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 $\mathbf{B}^{-1}$. In our case $circumellipse$ 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 $\mathcal{G}_{\mathbf{V}_k^I}$. Its implementation is given in Appendix A. 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.

### 6.3. Normalization

Since the resampling kernels are truncated to a finite support and the surface is sampled irregularly, the resampling kernels do not sum to 1 and the intensity of the rendered texture varies in screen-space which is an unpleasant artifact. Zwicker *et al.* [22] perform an additional per-pixel normalization in screen-space after the points had been splatted to rectify this problem.

In DOF rendering we cannot do this post-normalization because we use the accumulated weights as the estimate for partial coverage. In case of DOF rendering, this estimate has to be much more precise than in the case of edge antialiasing. Motivated by Ren *et al.* [15] 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. However, a specialized tool for this purpose would be useful.

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 very large support of the reconstruction filters ($c = 3.5 - 4$) such that the influence of truncation becomes negligible. This allows us to use the normalized model for any value of $c$ without having to renormalize it. To take a smaller $c$ into account we divide the weights during rendering by the compensation factor $1 - e^{-c}$ (Figure 3b) 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 $c = 2 - 3$.

### 6.4. Surface Reconstruction

To resolve visibility a means for deciding whether two fragments come from a single surface (and should be merged) or from different surfaces (and should be kept separated) must be introduced in the EWA surface splatting algorithm. Zwicker *et al.* [22] use the depth threshold — if the depths of two fragments differ by less than the threshold, they are supposed to lie on the same surface. This technique gets into troubles if some kind of filtering is applied that enlarges the resampling kernel support in screen space (*e.g.* prefiltering, depth of field filtering). The resampling kernel depth values then get extrapolated and reasonable surface reconstruction is not possible.

We use the surface reconstruction algorithm based on *z-ranges* as described by Räsänen [14]. A minimum and maximum depth is computed for each basis function $r_k$ in camera space and is assigned to each fragment emerging from rasterizing the resampling kernel $\rho_k$. When a new fragment is inserted into the A-buffer [1], its z-range is checked for intersection with the z-ranges of the fragments already present in the A-buffer. If the z-ranges overlap, the fragments are merged and their z-range is the union of respective z-ranges of new and existing fragments.

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. However, for surfaces that are close to each other or for intersecting surfaces, artifacts cannot be avoided, because of incorrect merge/separate decisions.

### 6.5. Shading

Shading can be done per-splat, before the points are splatted, or per-pixel, after all the points have been splatted [14]. We use per-splat shading. 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.

| 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**

## 7. Results

We have implemented the DOF rendering algorithm in a software EWA surface splatter. We use A-buffer [1] for transparency and edge antialiasing. Figure 1 illustrates the DOF rendering with semitransparent surface. 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 (*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 our algorithm produces some artifacts. They are mainly due to the incorrect merge/separate decisions in the surface reconstruction process (Section 6.4). 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.

Rendering performance is summarized in Table 1. It was measured for $512 \times 512$ 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 - in thousands), the number of points used for rendering (#PTS) and the rendering time (time) for objects in Figure 6 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).

## 8 Discussion

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

- *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 [16, 4, 17, 10]. 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 $1 \pm 0.1$ which is acceptable for our purposes. However, the pre-normalization works well only for models that are sampled without irregularities.

- *A-buffer correctly solves visibility and reconstructs surfaces.* This is the most restricting assumption which often fails and can lead to severe artifacts. The reason lies in overly big z-ranges that 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.

## 9. 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 drawbacks of the algorithm are mainly high sensitivity to the regularity of sample positions on the surface of point-based object 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 where it can be used to focus observer's attention to important features. We would also like to develop a specialized tool for 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] A. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufman, 1995.

[7] P. 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.

[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, Dept. of Computer Science, 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] C. Scofield. $2\frac{1}{2}$-d depth-of-field simulation for computer animation. In *Graphics Gems III*, pages 36–38. AP Professional, 1992.

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

[21] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. EWA volume splatting. In *Proceedings of IEEE Visualization*, 2001.

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

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

## Appendix A: 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 $\mathcal{E}_{\mathrm{M}}$ are [7]:

$$A', C' = \frac{A + C \pm \sqrt{(A-C)^2 + B^2}}{2}, \quad B' = 0, \quad (13)$$

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

$$\begin{aligned} \cos^2\theta &= \tfrac{1}{2}\left(\frac{A-C}{\sqrt{(A-C)^2+B^2}} + 1\right) \\ \sin 2\theta &= \frac{B}{\sqrt{(A-C)^2+B^2}} \\ A'' &= (A' - C')\cos^2\theta + C' \\ B'' &= (A' - C')\sin 2\theta \\ C'' &= (C' - A')\cos^2\theta + A'. \end{aligned} \quad (14)$$

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 (13)
$A' := \max\{A', R_k^I\}$
$C' := \max\{C', R_k^I\}$
$A'', B'', C'' :=$ rotate back with Equations (14)
**if** swapped **then**
  $\mathrm{swap}(A'', C'')$
**end if**

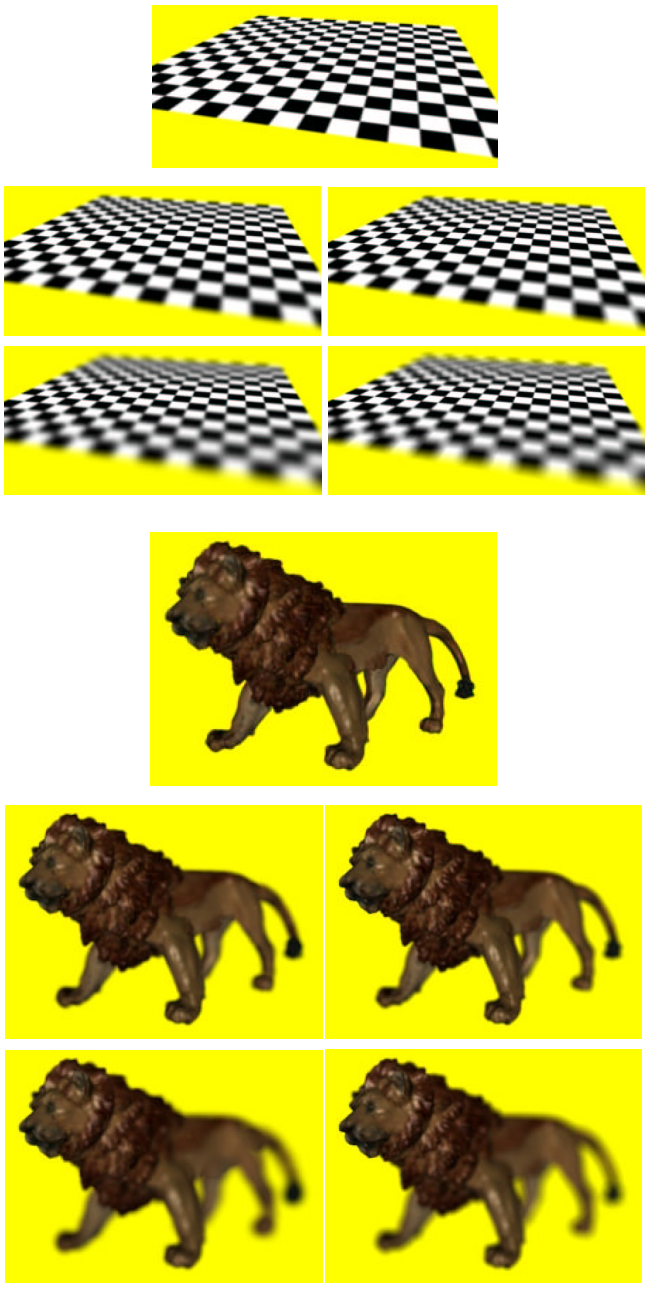

**Figure 6. Comparison of our algorithm (left) with reference images created with multisampling algorithm (right). Centered images are without DOF.**