EUROGRAPHICS 2010 / T. Akenine-Möller and M. Zwicker
(Guest Editors)

Volume 29 (2010), Number 2

Motion Blur for EWA Surface Splatting

Simon Heinzle[1] Johanna Wolf[1] Yoshihiro Kanamori[2] Tim Weyrich[3] Tomoyuki Nishita[4] Markus Gross[1]

[1]ETH Zurich [2]University of Tsukuba [3]University College London [4]University of Tokyo

Abstract

This paper presents a novel framework for elliptical weighted average (EWA) surface splatting with time-varying scenes. We extend the theoretical basis of the original framework by replacing the 2D surface reconstruction filters by 3D kernels which unify the spatial and temporal component of moving objects. Based on the newly derived mathematical framework we introduce a rendering algorithm that supports the generation of high-quality motion blur for point-based objects using a piecewise linear approximation of the motion. The rendering algorithm applies ellipsoids as rendering primitives which are constructed by extending planar EWA surface splats into the temporal dimension along the instantaneous motion vector. Finally, we present an implementation of the proposed rendering algorithm with approximated occlusion handling using advanced features of modern GPUs and show its capability of producing motion-blurred result images at interactive frame rates.

1. Introduction

Point-based geometries have evolved into a valuable alternative to polygonal meshes because of their conceptual simplicity and superior flexibility [GP07]. Point rendering is particularly interesting for highly complex models whose triangle representation requires millions of tiny primitives. The projected area of those triangles is often less than a few pixels, resulting in inefficient rendering because of the overhead for triangle setup and, more importantly, may lead to aliasing artifacts in cases where triangles get discarded during rasterization. Well-established among point rendering methods is the technique of EWA surface splatting [ZPBG02] which allows for rendering high-quality anti-aliased images of geometric objects that are given by a sufficiently dense set of sample points. The idea is to approximate local regions of the surface by planar elliptical Gaussian reconstruction kernels – so-called surface splats – in object space and then render the surface by blending these splats in screen space. The projected kernels are combined with a low-pass filter such that sampling artifacts like holes or aliasing can be effectively avoided.

On the other hand, EWA surface splatting generates still frames, which depict a perfect instant in time, lacking realism and the sensation of dynamics due to the absence of motion blur. The term motion blur denotes the visual effect that appears in still images or film sequences when objects moving with rapid velocity are captured. The image is perceived as smeared or blurred along the direction of relative motion to the camera. The reason for this is that the image taken by a camera in fact is an integration of the incoming light over the period of exposure. This appears natural because the human eye behaves in a similar way. To achieve this effect in computer graphics the most correct approaches are either temporal supersampling, that is, producing frames as a composite of many time instants sampled above the Nyquist frequency, or - which is theoretically more justified - bandlimiting the incoming signal before sampling to guarantee that its Nyquist frequency is met.

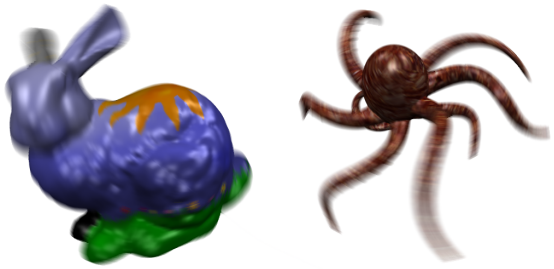

Figure 1: *EWA motion blur examples rendered with our GPU implementation.*

In this paper we propose a new method for applying motion blur to EWA surface splatting. We present a consistent extension of the theoretical basis of the EWA splatting framework in the time dimension to mathematically repre-

sent motion-blurred images with point-sampled geometry. The conceptual elegance of our approach lies in replacing the 2D Gaussian kernels which continuously reconstruct the point-sampled surface by 3D Gaussian kernels which unify a spatial and temporal component. By use of these kernels the scene can be reconstructed continuously in space as well as time and the incoming signal can be bandlimited before sampling to guarantee that its Nyquist frequency is met. The derived result naturally fits into the EWA splatting algorithm such that the final image can be computed as a weighted sum of warped and bandlimited kernels.

Based on the developed mathematical framework, we introduce a rendering algorithm with strong parallels to the original EWA surface splatting. This algorithm applies ellipsoids with spatial and temporal dimensionality as new rendering primitives. Specifically, the surface splats are extended by a temporal dimension along the instantaneous velocity vector. The emerging ellipsoids automatically adapt to the local length of the piecewise linearized motion trajectory. Finally, we provide an approximation of the rendering algorithm by the description of an entire point rendering pipeline using vertex, geometry and fragment program capability of current GPUs.

2. Related Work

Point-based Rendering: The possibility of using points as rendering primitives was first suggested by Levoy and Whitted [LW85] and has since been explored extensively [KB04, SP04]. While early point rendering techniques [GD98, PZvBG00] proposed simple forward mapping of points to the frame buffer, more advanced techniques focused on anti-aliasing methods for points. The EWA surface splatting algorithm [ZPBG02] is based on Heckbert's texture filtering approach [Hec89] and is a high-quality rendering approach for point-sampled geometry. EWA splatting has been implemented on conventional GPUs either by employing textured quads to approximate the points [RPZ02] or by making use of advanced features of GPUs [ZRB*04, BHZK05, ZP06, GBP06]. Recently, a dedicated hardware architecture for EWA surface splatting has been presented [WHA*07, HSA*08]. Reconstruction methods for rendering point sets [ABCO*01, GG07] have been investigated as well, either through raytracing [AKP*05] or a dynamic upsampling approach which directly splats the generated samples [GGG08].

Motion Blur: A well-argued discussion on motion blur is provided by the work of Sung et al. [SPW02]. Here the authors define the problem of motion blur based on the rendering equation and categorize previous work according to the respective approach to approximate this equation. Monte Carlo integration methods such as Distributed Raytracing [CPC84, Coo86] try to approximate the integral directly by stochastic supersampling. However, a large number of samples is usually required to avoid excessive noise. In a similar context the frameless rendering approach [BFMZ94] simulates motion blur directly via immediate randomized pixel updates directed by the rate of change. Haeberli and Akeley [HA90] achieved motion blur by supersampling rasterized scenes at different time instants. Nevertheless, to avoid banding artifacts the complete scene must be rendered at a frequency higher than the Nyquist frequency of the fastest motion. Recent work by Egan et al. [ETH*09] observed that motion blur is caused by a shear in the space-time signal as well as in the frequency domain. The resulting adaptive sampling scheme combined with a sheared reconstruction filter then produces high-quality results with lower sampling rates as previous supersampling methods.

Other work reduces the complexity of the problem by making assumptions on the scene behavior and/or employing further simplifications. The work of Korein and Badler [KB83] computes the exact per-pixel visible intervals for each geometry element. Grant [Gra85] proposes a 4D representation for 3D polyhedra in linear motion to compute temporally continuous visible polyhedra in 3D. Both approaches rely on a temporal box filter and assume constant shading over time.

Further methods are based on a geometric morphing of the objects or the introduction of new geometry. In the case of particle systems, [Ree83] suggests rendering particle points as line segments. [Cat84] uses a circular filter at every pixel and shows that motion blur can be achieved by morphing the filter or, as he proposes, by morphing the objects. [WZ96] approximate motion blur by constructing new semi-transparent geometry without solving the visibility function and inter-object relations.

Another field is constituted of various post-processing techniques which operate on the synthesized images and disregard the actual geometry. Potmesil and Charavarty [PC83] produce motion blur by a convolution of the rendered still image with a point spread function derived from the motion of the objects. The two-and-a-half-D motion blur algorithm [ML85] handles multiple scene objects by convolving them individually, followed by a composition in a back-to-front manner. In comparison, similar to other post-processing techniques [Max90, Shi93, CW93], these approaches cannot adapt to local properties of the geometry and cannot address the situation where moving objects cannot be separated into non-overlapping layers in depth.

The work of [MMS*98] proposes a splatting approach for volumetric rendering and shows how motion blur can be achieved by constructing new footprint functions based on circular splats moving in the image plane. The work of [GM04] extends this approach for EWA surface splatting by focusing on strong motion hints instead of photo-realistic motion blur. Their method combines a static image of the scene with blurred motion hint ellipses constructed from the original object using the motion vector.

3. Extended EWA Surface Splatting

To formulate the problem of motion blur we interpret an image as a 2D signal in screen space. For an instantaneous image at time t, the intensity at screen-space position $\mathbf{x}$ is given by the continuous spatio-temporal screen-space signal $g(\mathbf{x},t)$. A motion-blurred image which captures the scene

over the exposure period T is represented by $G_T(\mathbf{x})$. The intensity value at position $\mathbf{x}$ is generated by a weighted integration of incoming intensities over the exposure time:

$$G_T(\mathbf{x}) = \int_T a(t)g(\mathbf{x},t)dt \;, \tag{1}$$

where $a(t)$ denotes a time-dependent weighting function used to model the influence of the camera shutter and the medium which captures the scene. The process of generating $G_T(\mathbf{x})$ can be considered as a resampling problem of $g(\mathbf{x},t)$.

In the following subsections we extend the original EWA framework [Hec89, ZPBG02] by a time dimensionality and introduce a temporal visibility function to determine occluded surface parts. We then introduce three-dimensional reconstruction kernels representing a local, linear approximation of the points' motion trajectories, very much like the two-dimensional reconstruction kernels of EWA splatting do in the spatial domain. The algorithm presented in Section 4 finally renders these kernels to the screen.

3.1. The Continuous Spatio-Temporal Screen-Space Signal

Within the EWA framework, scene objects are represented by a continuous surface function $f(\mathbf{u},t)$ which is defined over a local surface parametrization $\mathbf{u}$, also called the source space. The projective mapping

$$\mathbf{m}(\mathbf{u},t)\colon \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2 \times \mathbb{R} \tag{2}$$

then maps this surface function from source space to screen space. It is locally invertible for a fixed time instant t and assigns a surface point $\mathbf{u}$ to its corresponding screen position $\mathbf{x}$. Using this mapping the spatio-temporal screen-space signal can be formulated as

$$g(\mathbf{x},t) = f(\mathbf{m}^{-1}(\mathbf{x},t),t) \;. \tag{3}$$

The continuous surface function $f(\mathbf{u},t)$ itself is represented by the set $\{P_k(t)\}$ of time-dependent, irregularly spaced point samples. Each point $P_k(t)$ is associated with a position $\mathbf{u}_k(t)$ at time t and an ellipsoidal reconstruction kernel $r_k(\mathbf{u}-\mathbf{u}_k(t),t)$. The continuous surface function $f(\mathbf{u},t)$ at time t is reconstructed by the weighted sum

$$f(\mathbf{u},t) = \sum_{k\in\mathbb{N}} w_k(t) r_k(\mathbf{u}-\mathbf{u}_k(t),t) \;, \tag{4}$$

where $w_k(t)$ denotes the attribute value of the k-th point sample at time t.

We define the projective mapping $\mathbf{m}(\mathbf{u},t)$ individually for each reconstruction kernel as $\mathbf{m}_k(\mathbf{u},t)$ to simplify the following derivations.

3.2. Time-Varying EWA Surface Splatting

We extend the EWA surface splatting framework by a time dimension t and introduce a visibility function $v_k(\mathbf{x},t)$ which defines the visibility of any surface point $\mathbf{u}(t) = \mathbf{m}_k^{-1}(\mathbf{x},t)$ in the local parametrization plane of point sample $P_k(t)$ at time t from the camera viewpoint. By combining Equations 3 and 4 the spatio-temporal screen-space signal reformulates to

$$g(\mathbf{x},t) = \sum_{k\in\mathbb{N}} w_k(t) v_k(\mathbf{x},t) r'_k(\mathbf{x},t) \;, \tag{5}$$

where $r'_k(\mathbf{x},t) = r_k(\mathbf{m}_k^{-1}(\mathbf{x},t) - \mathbf{u}_k(t),t)$ represents a reconstruction kernel projected to screen space. Similar to Zwicker et al. [ZPBG02], we bandlimit the spatio-temporal screen-space signal with a spatio-temporal anti-aliasing filter $h(\mathbf{x},t)$:

$$\begin{aligned} g'(\mathbf{x},t) &= g(\mathbf{x},t) * h(\mathbf{x},t) \\ &= \int_T \int_{\mathbb{R}^2} g(\xi,\tau) h(x-\xi,t-\tau) d\xi d\tau \\ &= \sum_{k\in\mathbb{N}} w_k(t) \rho_k(\mathbf{x},t) \end{aligned} \tag{6}$$

where the filtered resampling kernels $\rho_k(\mathbf{x},t)$ are given as

$$\rho_k(\mathbf{x},t) = \int_T \int_{\mathbb{R}^2} v_k(\xi,\tau) r'_k(\xi,\tau) h(\mathbf{x}-\xi,t-\tau) d\xi d\tau \;. \tag{7}$$

The visibility is dependent on the reconstruction kernels, and $\rho_k(\mathbf{x},t)$ can be evaluated as follows. In the first step, all reconstruction kernels are filtered using $h(\mathbf{x},t)$. The visibility is then determined based on the filtered reconstruction kernels leading to the filtered resampling kernels.

3.3. Temporal Reconstruction

In analogy to a spatial reconstruction of $f(\mathbf{u},t)$ we sample the motion trajectories of $P_k(t)$ in time and subsequently build a reconstruction which is also continuous over time. To achieve this we employ ellipsoidal 3D reconstruction kernels $R_{kt_k}(\mathbf{u})$ which define linearized trajectory patches of the moving point samples, similarly to the 2D reconstruction kernels that define linearized patches of the surface in the original EWA splatting algorithm. These new reconstruction kernels are centered at the point-sample positions $\mathbf{u}_{kt_k} = \mathbf{u}_k(t_k)$ and are constructed by convolving the elliptical 2D Gaussian kernels $r_k(\mathbf{x},t_k)$ with a 1D Gaussian along the instantaneous velocity vector. The surface function is now continuously reconstructed as follows:

$$f(\mathbf{u},t) = \sum_{k\in\mathbb{N}} \sum_{t_k} w_k(t) R_{kt_k}(\mathbf{u}-\mathbf{u}_{kt_k}) \;. \tag{8}$$

Combining this result with Equation 3 leads to the following equation for the continuous spatio-temporal screen-space signal $g(\mathbf{x},t)$:

$$g(\mathbf{x},t) = \sum_{k\in\mathbb{N}} \sum_{t_k} w_k(t) v_{kt_k}(\mathbf{x},t) R'_{kt_k}(\mathbf{x}) \;, \tag{9}$$

where $R'_{kt_k}(\mathbf{x}) = R_{kt_k}(\mathbf{m}_k^{-1}(\mathbf{x},t_k) - \mathbf{u}_{kt_k})$ are the reconstruction kernels projected to screen space. The time index t_k reflects that the sampling times are chosen adaptively for the respective point-sample trajectories depending on the motion and shading changes over time, see Figure 2 for an illustration. The actual sampling we used in our implementation is described in Section 5.5.

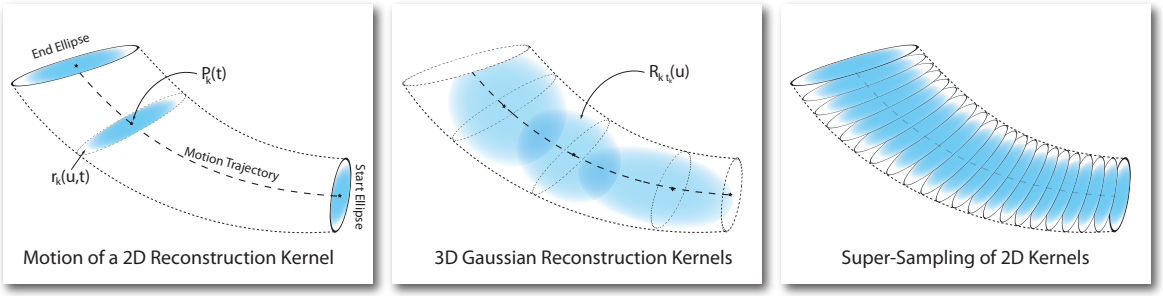

Figure 2: *A single 2D splat moving along its motion trajectory (left). Along the motion trajectory we place volumetric kernels that comprise the spatial and temporal domain of the moving 2D splat (center). In comparison, an approach consisting of a pure accumulation of temporal supersamples would require a high number of sampled 2D splats (right).*

An explicit expression for the bandlimited spatio-temporal screen-space signal of Equation 6 can then be expressed as:

$$\begin{aligned} g'(\mathbf{x},t) &= g(\mathbf{x},t) * h(\mathbf{x},t) \\ &= \sum_{k\in\mathbb{N}} \sum_{t_k} w_k(t)\hat{\rho}_{kt_k}(\mathbf{x},t) \,, \end{aligned} \tag{10}$$

with the filtered resampling kernels $\hat{\rho}_{kt_k}(\mathbf{x},t)$:

$$\hat{\rho}_{kt_k}(\mathbf{x},t) = \int_T \int_{\mathbb{R}^2} v_{kt_k}(\xi,\tau) R'_{kt_k}(\xi,\tau) h(\mathbf{x}-\xi, t_k-\tau) d\xi d\tau \,. \tag{11}$$

The resampling kernels $\hat{\rho}_{kt_k}(\mathbf{x},t)$ are again evaluated by first filtering the reconstruction kernels and then determining the visibility based on the filtered reconstruction kernels. We use the A-Buffer approach presented in Section 4.4 to resolve the visiblity in practice: in a first step, the visibility function is computed by determining which filtered reconstruction kernels contribute to a single pixel. The visibility is then used as temporal opacity contribution for each kernel when evaluating the integral in Equation 1.

4. Rendering

This section briefly gives an overview of the rendering algorithm before going into detail. First, the reconstruction filters $R'_{kt_k}(\mathbf{x})$ are constructed in a similar way to EWA surface splatting. The two axes $\mathbf{a}_1, \mathbf{a}_2$ span the plane of the original 2D surface splat and a third temporal axis $\mathbf{a}_3$ is constructed based on the motion vector $\mathbf{m}$. The latter is determined as the difference vector between the start and end positions of a point with respect to the time window of the temporal supersample and the temporal axis is constructed as $\mathbf{a}_3 = \frac{\alpha}{2}\mathbf{m}$, where α controls the variance in the time domain.

The reconstruction filter is projected to screen space, sampled at the pixel locations by rasterization and accumulated to the current image. In contrast to Zwicker et al. [ZPBG02] we do not use an affine approximation to map the filter to screen space but integrate the filter in a perspectively correct manner along the direction of the viewing ray [BSK04]. To reduce the computational complexity we limit the extent of the filter to a fixed cut-off value and can therefore also limit the extent of the filter to a bounding polygon in screen space.

The correct visibility is computed using an A-Buffer approach to separate distinct portions of the surface, regarding to the time interval the surface has been visible. After all visible filter integration values have been accumulated the result needs to be normalized since, in general, the EWA filters do not form a partition of unity in space.

The following subsections will provide details on the construction of the resampling filter, the integration of the resampling filter along the viewing ray, the bounding volume and the visibility A-Buffer.

4.1. The 3D Spatio-Temporal Reconstruction Filter

In analogy to EWA surface splatting we choose ellipsoidal Gaussians $\mathcal{G}^3_{\mathbf{Q}}(\mathbf{x})$ as 3D reconstruction kernels $R_{kt_k}(\mathbf{x}) = \mathcal{G}^3_{\mathbf{Q}_{kt_k}}(\mathbf{x})$ and as low-pass filters. We define a 3D ellipsoidal Gaussian $\mathcal{G}^3_{\mathbf{Q}}(\mathbf{x})$ with the 4x4 quadric matrix $\mathbf{Q}$ using homogeneous coordinates $\mathbf{x} = [x\ y\ z\ 1]^{\mathrm{T}}$ similar to [ZPBG02] as:

$$\mathcal{G}^3_{\mathbf{Q}}(\mathbf{x}) = \sqrt{\frac{\delta^3|\mathbf{Q}|}{\pi^3}}\, e^{-\delta\, \mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x}} \,, \tag{12}$$

where the Gaussian is normalized to the unit volume integral. The scaling factor δ controls the variance of the Gaussian. The quadric matrix $\mathbf{Q}$ can be decomposed into $\mathbf{Q} = \mathbf{T}^{-\mathrm{T}}\mathbf{D}\mathbf{T}^{-1}$, where the 4x4 transformation matrix $\mathbf{T}$ is constructed out of the three arbitrary, independent axis vectors $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ spanning the Gaussian centered at point $\mathbf{c}$:

$$\mathbf{T} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 & \mathbf{u} \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{13}$$

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} . \tag{14}$$

From a geometric viewpoint the three axis vectors span a skewed coordinate system, that is to say, the system of the Gaussian filter, see Figure 3. A point in space is transformed into the coordinate system of the Gaussian filter by $\mathbf{T}^{-1}\mathbf{x}$ and the weight of the filter is evaluated based on the distance of this point to the origin.

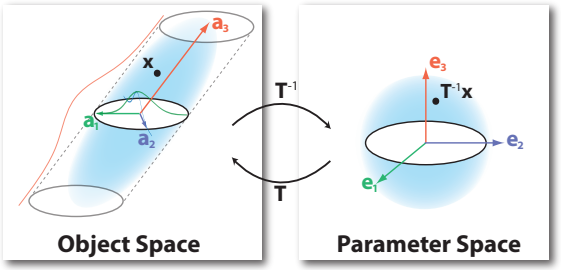

Figure 3: *We construct the 3D reconstruction kernels based on the two original splat axes $\mathbf{a}_1, \mathbf{a}_2$ and the instantaneous motion vector $\mathbf{m}$. The matrix $\mathbf{T}$ is used to transform points and lines from object space to the respective parameter space. In parameter space an iso-level of the reconstruction kernel can be interpreted as the unit sphere.*

The correct screen-space bandlimiting filter is approximated by evaluating it in object space similar to [GBP06]. We first estimate the spacing of the pixel grid in object space. If the length of the axes, projected onto this estimation, does not match the filter width, we enlarge the axes to the size of the filter radius.

The next section will show how to integrate a reconstruction filter along the viewing ray.

4.2. Sampling of the Reconstruction Filter

To evaluate the contribution of a single 3D Gaussian to a pixel, the Gaussian is integrated along the path of the viewing ray within the integration time which equals the time of exposure. The viewing ray for a pixel can be put in parametrized form as $\mathbf{r}(s) = \mathbf{p} + s\,\mathbf{d}$, where $\mathbf{p} = [x_w\ y_w\ 0\ 1]^{\mathrm{T}}$ denotes the pixel position in window coordinates and vector $\mathbf{d} = [0\ 0\ 1\ 0]^{\mathrm{T}}$ represents the viewing direction of the ray. In a first step the viewing ray is transformed into the parameter system of the ellipsoid:

$$\tilde{\mathbf{r}}(s) = (\mathbf{V}\cdot\mathbf{P}\cdot\mathbf{M}\cdot\mathbf{T})^{-1}\mathbf{r}(s) = \tilde{\mathbf{p}} + s\,\tilde{\mathbf{d}}\,, \tag{15}$$

where $\mathbf{V}$, $\mathbf{P}$ and $\mathbf{M}$ denote the viewport, projection and modelview matrix, respectively.

In a next step $\tilde{\mathbf{r}}(s)$ is transformed to the integration line $\mathbf{l}(t)$. The integration line parametrizes the position on the ray at integration time t. Let $\tilde{\mathbf{p}}' = \tilde{\mathbf{p}}_{xyz}/\tilde{\mathbf{p}}_w$ and $\tilde{\mathbf{d}}' = \tilde{\mathbf{d}}_{xyz}$ be the dehomogenized vectors. The integration line is then defined as

$$\mathbf{l}(t) = \mathbf{b} + \mathbf{f}t\,, \tag{16}$$

with the transformed support point and direction

$$\mathbf{b} = \tilde{\mathbf{p}}' - \frac{\tilde{\mathbf{p}}'_z}{\tilde{\mathbf{d}}'_z}\tilde{\mathbf{d}}',\ \ \mathbf{f} = -\frac{\tilde{\mathbf{d}}'}{\tilde{\mathbf{d}}'_z}\,, \tag{17}$$

where $\mathbf{b}$ is projected along $\tilde{\mathbf{r}}(s)$ to time $z = 0$ and $\mathbf{f}$ is the direction from point a time $z = 0$ to the point at time $z = 1$. Recall that the z-axis in parameter space represents the temporal dimensionality. Figure 4 illustrates this conversion. The

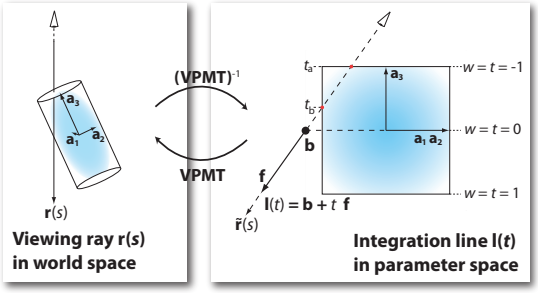

Figure 4: *To evaluate the line integral along the viewing ray $\mathbf{r}(s)$ it is first transformed to the parameter space viewing ray $\tilde{\mathbf{r}}(s)$ and finally normalized to the integration line $\mathbf{l}(t)$. The integration line, again, is normalized to the time interval $[-1,1]$ which represents the total integration period. The integral is evaluated in the time interval $[t_a,t_b]$ in which the kernel is visible.*

integral along the 3D Gaussian visible in the time interval $[t_a,t_b]$ becomes

$$\begin{aligned}\int_{t_a}^{t_b}\mathcal{G}^3(\mathbf{l}(t)dt &= \int_{t_a}^{t_b} e^{-\delta\,\mathbf{l}^{\mathrm{T}}(t)\mathbf{l}(t)}dt \\ &= \int_{t_a}^{t_b} e^{-\delta\,(\mathbf{b}^{\mathrm{T}}\mathbf{b}+2\mathbf{b}^{\mathrm{T}}\mathbf{f}\,t+\mathbf{f}^{\mathrm{T}}\mathbf{f}\,t^2)} \\ &= \int_{t_a}^{t_b} \underbrace{e^{-\delta\,\mathbf{l}_{xy}^{\mathrm{T}}(t)\mathbf{l}_{xy}(t)}}_{\text{Spatial}}\cdot\underbrace{e^{-\delta\,t^2}}_{\text{Temporal}}\,dt\,,\end{aligned} \tag{18}$$

where the normalization of the Gaussian is performed implicitly by the transformation from the viewing ray to the integration ray. The solution for the finite integral is given as

$$\begin{aligned}\int_{t_a}^{t_b}\mathcal{G}^3(\mathbf{l}(t))dt = &\frac{\sqrt{\pi}e^{-\frac{\delta(\mathbf{b}^{\mathrm{T}}\mathbf{b}\mathbf{f}^{\mathrm{T}}\mathbf{f}-(\mathbf{b}^{\mathrm{T}}\mathbf{f})^2)}{\mathbf{f}^{\mathrm{T}}\mathbf{f}}}}{2\sqrt{\delta\mathbf{f}^{\mathrm{T}}\mathbf{f}}} \\ &\cdot(\mathrm{erf}(f(t_b)) - \mathrm{erf}(f(t_a)))\,,\end{aligned} \tag{19}$$

where

$$f(x) = \sqrt{\frac{\delta}{\mathbf{f}^{\mathrm{T}}\mathbf{f}}}\left(\mathbf{f}^{\mathrm{T}}\mathbf{f}\cdot x + \mathbf{b}^{\mathrm{T}}\mathbf{f}\right),\ \ \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt\,. \tag{20}$$

The Gauss error function $\mathrm{erf}(x)$ cannot be computed analytically and will be stored as a look-up table for performance reasons.

4.3. Bounding Volume Restriction

Theoretically the Gaussian reconstruction filters need to be evaluated on the whole image plane. Nevertheless, the Gaussian decays very fast and rays through pixel locations farther than a certain cut-off distance from the projected kernel center have negligible contributions to the image. To simplify the computations for the visibility we therefore bound the evaluation of the line integral by bounding the Gaussian filter with a 3D tube.

The tube is defined as $\mathbf{x}^{\mathrm{T}}\mathbf{T}^{-\mathrm{T}}\mathbf{D}_{xy}\mathbf{T}^{-1}\mathbf{x}-1=0$ in object space, where $\mathbf{D}_{xy}=\mathrm{diag}(1,1,0,0)$ is a diagonal 4x4 matrix and T is identical to the variance matrix of the ellipsoid in Equation 14. Intersection points between a ray and the tube are then determined by inserting Equation 15 into this relation and solving the quadratic equation

$$\mathbf{r}(s)^{\mathrm{T}}(\mathbf{VPMT})^{-\mathrm{T}}\mathbf{D}_{xy}(\mathbf{VPMT})^{-1}\mathbf{r}(s)= \tilde{\mathbf{r}}_{xy}(s)^{\mathrm{T}}\tilde{\mathbf{r}}_{xy}(s)=1\,. \qquad (21)$$

In addition to the quadratic equation we bound the tube by the two ellipses lying on the cut-off planes $\tilde{\mathbf{r}}_z(s)=\pm 1$ and arrive at the bounding cylinder.

In the same way as the ellipsoid volume is constrained in object space, its extent in screen space can be bounded by the projection of the cylinder. The bounding box computation of the latter can be performed as described by Sigg et al. [SWBG06]: First, the two axis-aligned bounding boxes of the bounding ellipses are computed. Sigg et al. construct a single, axis-aligned bounding box out of these two boxes which leads to a non-optimal bounding box.

We propose to use the convex hull polygon instead. It can be determined efficiently by comparing each of the four corresponding corner pairs separately. For each pair the following simple relation holds: If one of the two vertices lies in the inside quadrant the inside vertex is discarded. The inside quadrant is defined by the two lines of the bounding box adjacent to the vertex. In case the vertex is not in the inside quadrant both vertices are connected. This simple algorithm can be organized in a way that it yields a sorted list of points defining the convex hull and can be directly used as an output for a triangle strip by interleaving the vertex order. See Figure 5 for an illustration.

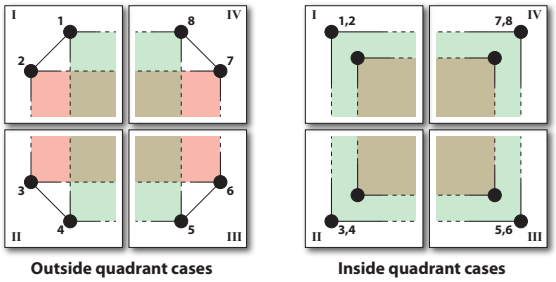

Figure 5: *Convex hull of two axis-aligned screen-space bounding boxes. The algorithm compares the corresponding four corners of the boxes in counter-clockwise order. If none of the two corresponding corners lie within the inside quadrant of the other corner, both corners belong to the convex hull (left). Otherwise the corner located in the inside quadrant is discarded (right).*

4.4. Visibility

The visibility of a surface point is a discontinuous, non-bandlimited function. The optimal strategy to solve the visibility problem for motion-blurred EWA surface splatting in terms of visual quality is similar to the approach proposed by [KB83]. As a result of the integration of all kernels along the viewing rays, the time intervals in which each single kernel is visible at a pixel are known. These integration intervals $[t_a^n, t_b^n]$ are furthermore associated with a depth interval $[d_a^n, d_b^n]$, resembling the sorted lists of the A-Buffer method [Car84].

Our adopted A-Buffer algorithm iterates over the intervals starting with the earliest time interval. All subsequent time intervals are compared for temporal and depth overlap using a ternary depth test commonly used in EWA surface splatting. In case both a temporal and depth overlap is detected, the intervals belong to reconstruction kernels associated with the same surface. Therefore, they are weighted and accumulated to the current pixel value and finally normalized. If there is only a temporal overlap present, the depth interval of the later time interval is adjusted accordingly by removing the occluded interval from it. After all intervals have been processed, the visibility functions $v_k(\mathbf{x},t)$ of all reconstruction kernels contributing to the pixel are known. The final pixel value is calculated based on Equation 1 by summing up all contributions weighted with the time-dependent shutter function $a(t)$.

The visibility algorithm as described above could be implemented efficiently on a ray-tracer which naturally traverses the objects in a roughly sorted depth order. Section 5 presents an approximative strategy to solve the visibility problem on modern GPUs.

4.5. Discussion

This rendering framework is able to produce realistic motion blur within the EWA surface splatting framework as shown in Figures 8a and 9a. Our method determines the temporal visibility of the ellipsoidal reconstruction kernels and is able to render motion-blurred scenes using accurate surface contributions dependent on their visibility.

Similar to our work, the method of [GM04] also constructs three-dimensional reconstruction kernels by convolving the two-dimensional object space ellipses with a low-pass filter along the motion direction. In a subsequent step, the reconstruction kernels are projected to screen space ellipses and used for rendering.

However, our framework and its rendering algorithm differ substantially from [GM04]. Their work combines a static image of the scene combined with blurred motion hint ellipses. The static scene is generated using the original EWA surface splatting algorithm, whereas the blurred motion hint ellipses are rendered using a modified technique. Furthermore, their work neglects the temporal visibility and cannot separate the spatial contributions of the motion ellipses and consequently their opacity is not accurate. The results of this limitation are the striping artifacts and disturbing overblending in areas with high overlap of motion ellipses. Therefore it cannot reproduce photo-realistic motion blur as our approach.

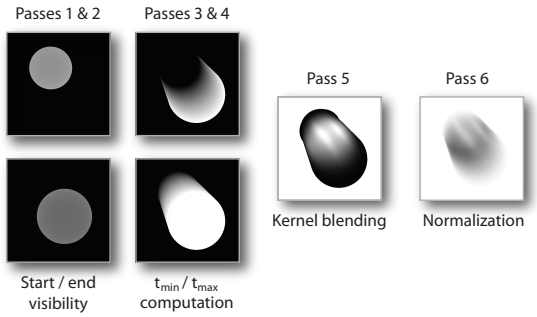

Figure 6: *The GPU implementation uses six passes to synthesize motion-blurred images. The first two passes approximate the visibility of the surface function at the start and end of the exposure time (left). The subsequent passes three and four approximate the visibility of the object in relation to the background by computing the per-pixel time intervals during which geometry is visible (middle left). After all reconstruction kernels have been blended (middle right), the accumulated values are normalized and the rendered object is composed with the background (right).*

5. GPU Implementation

The GPU implementation approximates the rendering framework using multiple render passes:

1. The first pass renders elliptical splats at the start time of the integration period into the depth buffer only. The result is used as an approximation for the subsequent depth tests.
2. The second pass renders elliptical splats at the end time of the integration period, similar to step 1.
3. The third pass approximates the visibility against static objects and the background by determining the earliest instant of time in which geometry is visible for every pixel.
4. The fourth pass determines the latest instant of time in which geometry is visible for every pixel, similar to step 3.
5. The fifth pass blends all visible reconstruction kernels into the accumulation buffer.
6. The sixth pass performs the normalization of the blended kernels.

The following sections provide details on these steps.

5.1. Visibility Passes 1 and 2

The first two passes render the scene to the depth buffer only by making use of elliptical discs as rendering primitives. The first pass renders the scene at the start of the integration period, whereas the second pass renders the same scene transformed to the end of the integration period. Similar to [BHZK05], the elliptical discs are rendered without blending to determine the visible surface at a single time instant for each pixel.

The resulting two depth buffers are then applied during the subsequent steps to approximate the kernel visibilities. The depth test works as follows. For every ellipsoid its minimum depth at the current fragment is determined and compared to the corresponding entries of both depth buffers. If the ellipsoid is visible in either one of the depth buffers the processing is continued, otherwise the rest of the computations for this fragment can be discarded.

5.2. Background Visibility Passes 3 and 4

In a next step we approximate the solution for the visibility problem with respect to the background or static objects by estimating the total fraction of time in which any of the kernels is contributing to a pixel. The fifth pass reuses the same vertex and geometry shader as described here.

The vertex shader constructs the velocity vector based on the transformation matrices of the start and end frame. The normals of the 2D splats at the start and end frame are computed to perform backface culling in case both normals are facing away from the camera. In a following step the screen-space bandlimiting filter is approximated by means of ensuring that the projection of the axes equals at least the size of the filter radius. Based on the outcome of the filter the three axis vectors $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ are constructed. To avoid numerical issues during the transformation of the viewing ray we enforce a minimum length of the velocity axis $\mathbf{a}_3$ as well as a minimum angle between the $\mathbf{a}_1, \mathbf{a}_2$ plane and $\mathbf{a}_3$. Then, based on the axis vectors $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ and the kernel center $\mathbf{c}$, the transformation matrix $(\mathbf{PMT})^{-1}$ is constructed. Similar to [SWBG06] we compute the two axis-aligned bounding boxes for the start and end frame ellipses. Instead of simply taking the combined axis-aligned bounding box, we send both bounding boxes to the geometry shader to construct a tighter bounding primitive.

The geometry shader combines the two axis-aligned bounding boxes to a convex octagon with the method described earlier in Section 4.3.

The fragment shader estimates the time intervals in which the ellipsoids are visible. The idea is to determine the earliest time instance t_{min} and the latest time instance t_{max} at which any reconstruction kernel is covering the pixel. The viewing ray is first transformed from camera space to parameter space and intersected with the cylinder. The z-component of the intersection points can directly be used as the integration interval $[t'_{min}, t'_{max}]$ of a single kernel. The depth test ensures that only the smaller or greater values, respectively, are written to the depth buffer. At the end of the rendering cycle the two depth buffers contain the earliest and the latest times per pixel when any of the kernels becomes visible.

Pass 6 later uses $[t_{min}, t_{max}]$ as an approximation of the correct time intervals.

5.3. Blending Pass 5

In this pass the vertex and geometry program of the passes 3 and 4 are reused. The fragment shader transforms the viewing ray to the integration ray and performs the integration. To evaluate $\exp(\mathbf{x})$ and $\mathrm{erf}(x)$ we utilize look-up tables which are stored in textures. As an approximation, all visible reconstruction kernels are blended using the integration weight

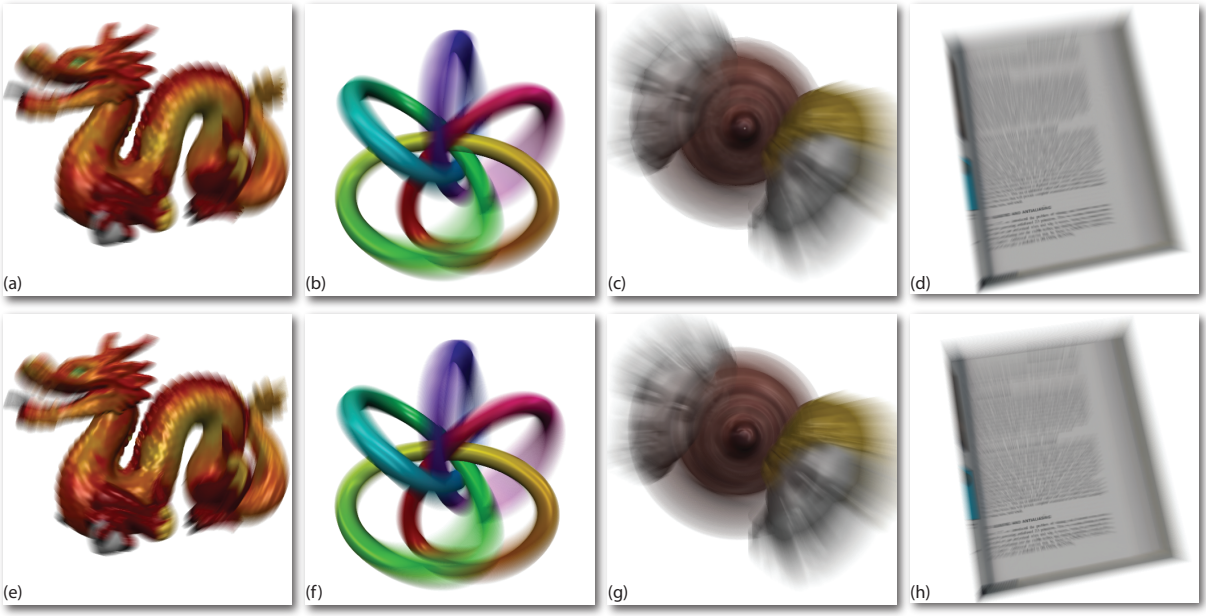

Figure 7: *Comparison of the result images rendered with our GPU implementation (a-d) and images rendered with supersampling of conventional EWA surface splatting framework (e-h), performance figures are given in Table 1. Figures (a,b): dragon is rotating along the depth axis. Figures (b,f): colored knot rotating around the up and depth axis. Figures (c,g): fast rotating face and igea heads, the heads moving along depth axes and overlap the face. Figures (d,h): book moving towards the camera.*

of their whole integration period. To perform an accurate visibility test partial occlusions during the kernel integration period would have to be resolved as will be discussed later.

5.4. Normalization Pass 6

In a final step the rendered image is normalized by the sum of the kernel weights. The resulting image is combined with the background where the blending weight is given by the integral $\int_{t_{\min}}^{t_{\max}} e^{-t^2} dt$ reusing the time interval information acquired in the fourth and fifth pass.

5.5. Sampling of the Motion Path

To support fast, non-linear movements we resample the motion path in the case of such movements. The motion path is split into equidistant time frames and all 6 passes are performed on the sampled sub-sets individually. The results of each iteration are accumulated into an accumulation texture, and each sample is weighted using the integral $\int_{t_{\text{subframe}_i}}^{t_{\text{subframe}_i+1}} e^{-t^2} dt$. Furthermore, a higher sampling of the motion path helps to reduce visible artefacts due to the approximation of the visibility function. This approach allows us therefore to trade rendering speed for correctness of the result. Note that this is similar to super-sampling, however, a much lower amount of samples is generally needed.

6. Results and Limitations

The GPU implementation is able to achieve high-quality pictures at competitive frame rates compared to the pure temporal supersampling approach. Figures 1 and 7 show some sample renderings and comparisons. Table 1 lists the corresponding temporal supersampling rates and performance figures for all examples in this paper. All results have been computed on an NVIDIA GeForce 280 GPU. To fully experience the visual quality of our approach refer to the accompanying video.

The crucial benefit of our approach is that the volumetric kernels adapt automatically to the speed of the motion. In cases where an object moves at varying speeds a pure temporal supersampling would usually sample the complete animation at a constant rate nevertheless, whereas the volumetric kernels adapt implicitly.

As a limitation inter-object relationships cannot be handled correctly on the GPU because the visibility cannot be computed exactly, see Figure 8. The proposed approximation using the depth images is only able to exhibit a binary visibility function based on the visibility of the ellipsoids in the start and end frame. Additionally, artifacts may occur due to the approximated backface culling where kernels may get falsely discarded or accepted.

The visibility passes for determining $t_{\min}$ and $t_{\max}$ try to approximate the total time during which any geometry is visible at a fragment. This information is used to blend the blurred image with the background. As only the earliest and latest time instants are determined, problems arise if, for example, geometry is only visible shortly at the beginning and the end of the time interval, see Figure 9. However, our

Table 1: *Performance comparisons for the depicted examples, measured using an NVIDIA GTX 280. The supersampling factors 'x' have been chosen for similar visual quality.*

Figure	3D Kernels		Super sampling		Point
	x	Points/s	x	Points/s	count
1.a	4	1.80M	30	1.44M	350k
1.b	2	4.63M	25	2.91M	466k
7.a/7.e	4	1.91M	30	1.78M	554k
7.b/7.f	12	0.42M	40	0.46M	303k
7.c/7.g	12	0.63M	80	0.65M	310k
7.d/7.h	1	1.65M	60	0.50M	825k

approximation would classify the whole interval as being covered by kernels.

While objects moving at similar speed are blended in a plausible way, fast moving objects cannot be blended plausibly with slow moving objects due to the above mentioned approximations. Although our framework has been designed for generality, the visibility approximation limits the applicability of the GPU algorithm to scenes where different objects do not exhibit simultaneous overlaps in time and depth intervals. However, the artifacts which arise in this case can be alleviated by increasing the temporal supersampling rate along the motion trajectory which sacrifices rendering performance in favor of quality.

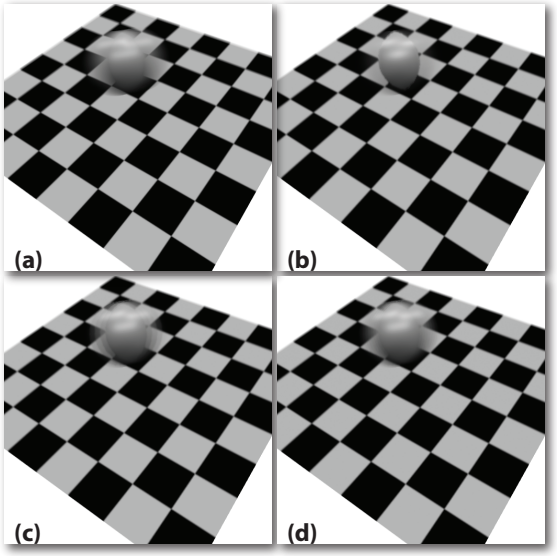

Figure 8: *Inter-object relations can only be handled using a higher sampling due to the visibility approximation on the GPU. Figure (a) A-Buffer software implementation. GPU implementation: (b) 1 sample, (c) 4 samples, (d) 8 samples.*

7. Conclusion

In this paper we have introduced a new method for rendering dynamic point-based geometry with motion blur effects based on EWA surface splatting. We have extended the theoretical basis of the EWA framework in a conceptually elegant and consistent way and consequently provided a mathematical representation of motion-blurred images with point-sampled geometry. Our method replaces the 2D surface splats of the original EWA surface splatting framework by 3D kernels which unify a spatial and temporal component. This allows for a continuous reconstruction of the scene in space as well as time and the motion-blurred image is computed as a weighted sum of warped and bandlimited kernels. Accordingly, we have described an approximation of the theoretical algorithm by means of a six-pass GPU rendering algorithm. Our point rendering pipeline applies ellipsoids with spatial and temporal dimensionality as new rendering primitives and exploits vertex, geometry and fragment program capability of current GPUs. Our framework for point splatting maintains generality by avoiding any assumptions or constraints on the nature of the captured scene such as motion or lighting.

As for future work, the focus has to be put on a more accurate handling of the visibility function. One of the possibilities to address this issue would be an implementation which exploits the GPU for general purpose programming. Another approach could compute the visibility function much more accurately by using specialized hardware extensions to current GPUs which support a reduced A-Buffer or by employing an extended dedicated hardware architecture such

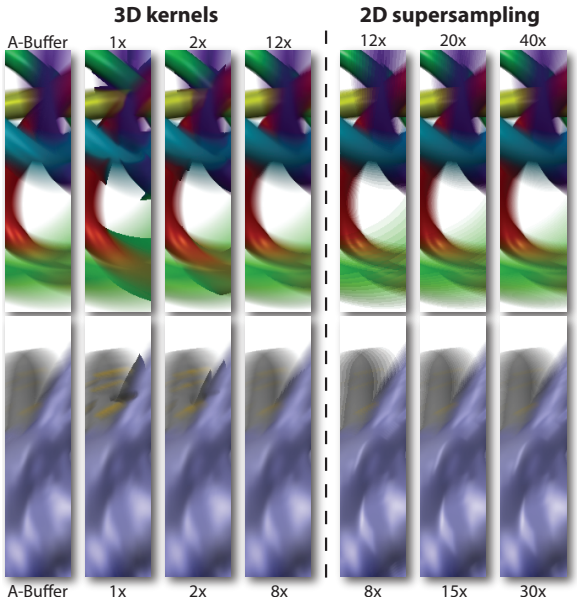

Figure 9: *Left: 3D kernels rendered with the A-Buffer software implementation and with different sampling rates for the GPU implementation. Right: 2D kernel supersampling with varying sampling rate. The visibility approximation for the GPU produces artifacts when geometry has been visible only shortly once at the beginning and once at the end of the integration interval. As a solution, the number of temporal samples can be increased. The A-Buffer implementation does not suffer from those artifacts.*

as [WHA*07]. As our current implementation uses a constant number of samples per frame, the sampling could be varied adaptively for each point based on its motion and shading changes over time in a future work.

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *IEEE Visualization* (2001), pp. 21–28. 2

[AKP*05] ADAMS B., KEISER R., PAULY M., GUIBAS L., GROSS M., DUTRÉ P.: Efficient raytracing of deforming point-sampled surfaces. *Computer Graphics Forum 24*, 3 (2005). 2

[BFMZ94] BISHOP G., FUCHS H., MCMILLAN L., ZAGIER E. J. S.: Frameless rendering: double buffering considered harmful. In *SIGGRAPH* (1994), ACM, pp. 175–176. 2

[BHZK05] BOTSCH M., HORNUNG A., ZWICKER M., KOBBELT L.: High-quality surface splatting on today's GPUs. In *Point-Based Graphics* (2005), Eurographics, pp. 17–24. 2, 7

[BSK04] BOTSCH M., SPERNAT M., KOBBELT L.: Phong splatting. In *Point-Based Graphics* (2004), Eurographics, pp. 25–32. 4

[Car84] CARPENTER L.: The A-buffer, an antialiased hidden surface method. In *SIGGRAPH* (1984), ACM, pp. 103–108. 6

[Cat84] CATMULL E.: An analytic visible surface algorithm for independent pixel processing. In *SIGGRAPH* (1984), ACM, pp. 109–115. 2

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. In *SIGGRAPH* (1986), ACM, pp. 51–72. 2

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. In *SIGGRAPH* (1984), ACM, pp. 137–145. 2

[CW93] CHEN S. E., WILLIAMS L.: View interpolation for image synthesis. In *SIGGRAPH* (1993), ACM, pp. 279–288. 2

[ETH*09] EGAN K., TSENG Y.-T., HOLZSCHUCH N., DURAND F., RAMAMOORTHI R.: Frequency analysis and sheared reconstruction for rendering motion blur. *ACM Transactions on Graphics (SIGGRAPH) 28*, 3 (2009), 1–13. 2

[GBP06] GUENNEBAUD G., BARTHE L., PAULIN M.: Splat/Mesh Blending, Perspective Rasterization and Transparency for Point-Based Rendering. In *Point-Based Graphics* (2006), Eurographics, pp. 49–58. 2, 5

[GD98] GROSSMAN J. P., DALLY W.: Point sample rendering. In *Rendering Techniques* (1998), Springer, pp. 181–192. 2

[GG07] GUENNEBAUD G., GROSS M.: Algebraic point set surfaces. *ACM Transactions on Graphics (SIGGRAPH) 26*, 3 (2007), 23:1–23:9. 2

[GGG08] GUENNEBAUD G., GERMANN M., GROSS M.: Dynamic sampling and rendering of algebraic point set surfaces. *Computer Graphics Forum 27*, 2 (2008), 653–662. 2

[GM04] GUAN X., MUELLER K.: Point-based surface rendering with motion blur. In *Point-Based Graphics* (2004), Eurographics. 2, 6

[GP07] GROSS M., PFISTER H.: *Point-Based Graphics*. Morgan Kaufmann, 2007. 1

[Gra85] GRANT C. W.: Integrated analytic spatial and temporal anti-aliasing for polyhedra in 4-space. In *SIGGRAPH* (1985), ACM, pp. 79–84. 2

[HA90] HAEBERLI P., AKELEY K.: The accumulation buffer: hardware support for high-quality rendering. In *SIGGRAPH* (1990), ACM, pp. 309–318. 2

[Hec89] HECKBERT P.: *Fundamentals of Texture Mapping and Image Warping*. Master's thesis, University of California at Berkeley, 1989. 2, 3

[HSA*08] HEINZLE S., SAURER O., AXMANN S., BROWARNIK D., SCHMIDT A., CARBOGNANI F., LUETHI P., FELBER N., GROSS M.: A transform, lighting and setup ASIC for surface splatting. In *International Symposium on Circuits and Systems* (2008), IEEE, pp. 2813–2816. 2

[KB83] KOREIN J., BADLER N.: Temporal anti-aliasing in computer generated animation. In *SIGGRAPH* (1983), ACM, pp. 377–388. 2, 6

[KB04] KOBBELT L., BOTSCH M.: A survey of point-based techniques in computer graphics. *Computers & Graphics 28* (2004), 801–814. 2

[LW85] LEVOY M., WHITTED T.: *The Use of Points as Display Primitives*. Tech. Rep. 85-022, UNC Chapel Hill, 1985. 2

[Max90] MAX N.: Polygon-based post-process motion blur. *The Visual Computer 6* (1990), 308–314. 2

[ML85] MAX N. L., LERNER D. M.: A two-and-a-half-D motion-blur algorithm. *SIGGRAPH* (1985), 85–93. 2

[MMS*98] MUELLER K., MÖLLER T., SWAN J. E., CRAWFIS R., SHAREEF N., YAGEL R.: Splatting errors and antialiasing. *IEEE TVCG 4*, 2 (1998), 178–191. 2

[PC83] POTMESIL M., CHAKRAVARTY I.: Modeling motion blur in computer-generated images. In *SIGGRAPH* (1983), ACM, pp. 389–399. 2

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *SIGGRAPH* (2000), ACM, pp. 335–342. 2

[Ree83] REEVES W. T.: Particle systems – a technique for modeling a class of fuzzy objects. In *SIGGRAPH* (1983), ACM, pp. 359–376. 2

[RPZ02] REN L., PFISTER H., ZWICKER M.: Object-space EWA surface splatting: A hardware accelerated approach to high quality point rendering. In *Computer Graphics Forum* (2002), vol. 21, pp. 461–470. 2

[Shi93] SHINYA M.: Spatial anti-aliasing for animation sequences with spatio-temporal filtering. In *SIGGRAPH* (1993), ACM, pp. 289–296. 2

[SP04] SAINZ M., PAJAROLA R.: Point-based rendering techniques. *Computers & Graphics 28*, 6 (2004), 869–879. 2

[SPW02] SUNG K., PEARCE A., WANG C.: Spatial-temporal antialiasing. *IEEE TVCG 8*, 2 (2002), 144–153. 2

[SWBG06] SIGG C., WEYRICH T., BOTSCH M., GROSS M.: GPU-based ray-casting of quadratic surfaces. In *Point-Based Graphics* (2006), Eurographics, pp. 59–65. 6, 7

[WHA*07] WEYRICH T., HEINZLE S., AILA T., FASNACHT D., OETIKER S., BOTSCH M., FLAIG C., MALL S., ROHRER K., FELBER N., KAESLIN H., GROSS M.: A hardware architecture for surface splatting. *ACM Transactions on Graphics (SIGGRAPH) 26*, 3 (2007), 90:1–90:11. 2, 10

[WZ96] WLOKA M. M., ZELEZNIK R. C.: Interactive real-time motion blur. *The Visual Computer 12*, 6 (1996), 283–295. 2

[ZP06] ZHANG Y., PAJAROLA R.: Single-pass point rendering and transparent shading. In *Point-Based Graphics* (2006), Eurographics, pp. 37–48. 2

[ZPBG02] ZWICKER M., PFISTER H., BAAR J. V., GROSS M.: EWA splatting. *IEEE TVCG 8*, 3 (2002), 223–238. 1, 2, 3, 4

[ZRB*04] ZWICKER M., RÄSÄNEN J., BOTSCH M., DACHSBACHER C., PAULY M.: Perspective accurate splatting. In *Graphics Interface* (2004). 2