Eurographics Symposium on Point-Based Graphics (2004)
M. Alexa, S. Rusinkiewicz, (Editors)

Point-based Surface Rendering with Motion Blur

Xin Guan and Klaus Mueller

Department of Computer Science, State University of New York at Stony Brook

Abstract

In this paper we show how to extend point-based surface rendering to illustrate object motion. We do this by first extruding the circular points into ellipsoids, which fill the space traced out by the points in motion. Using ellipsoids instead of cylinders achieves a low-passing effect of the motion trail. We then find the screen-space projection of each ellipsoid, which is an ellipse. These can be rendered conveniently using hardware acceleration. Our technique thus facilitates the rendering of complex objects with real-time motion blur. It gives the viewer a sharply rendered object together with the hint of the direction of motion. The construction of the motion blur trails can be based on different rendering primitives, as is also discussed in the paper. Various trail textures are presented to achieve artistic rendering results.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Display algorithms, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism: Color, shading, shadowing, and texture.

1. Introduction

Point-based rendering [RL00][PZvG00][ZPvBG02] has gained much popularity in recent years. One reason for this is the simplicity of rendering point-based primitives. Another reason is that, given the high level of detail of many geometric objects, it is more efficient to represent this detail by an easy-to-render point than a tiny polygon, which has significantly more rendering overhead. In this paper we show how the convenient point representation can be extended to illustrate the motion of point-based objects. In fact, there are two ways to illustrate motion: one is to show an animation sequence of the moving object, the other is to illustrate the motion in a single image. The latter can be most accurately achieved by averaging the images of the moving object over time. However, rendering each image and averaging them can be a time-consuming affair. Getting around this and still maintaining accuracy is difficult, however, since for each object instance along the motion path both shading and visibility change. A well-argued discussion on this is provided in [SPW02]. In this work, it is our goal to achieve an interactive rendering, using the convenience of the point primitive that models our objects. At the same time we seek to suggest the motion without encountering a blurring of the moving object. In that respect, we may want to replace the notion of motion blur with that of motion hint. In our work, we assume the motion to be a low-passing of the object location over time. Since the object is a collection of points, one could hence low-pass each point by itself, yielding an ellipsoid, and then render the resulting collection of these low-passed points. This yields a very efficient rendering framework to illustrate motion, but it is understood that it is not entirely physically accurate, due to the disregard to the changing shading effects across the trail. What is achieved is more of a non-photorealistic motion-bur effect, where, however, we have much control over the visual effects that can be accomplished. We demonstrate our results using mainly objects from volume graphics, which have been converted to surface point objects. But our approach will readily apply to any point-based object. Our paper is structured as follows. First, Section 2 will discuss previous work on related issues, and Section 3 and 4 will describe how the motion ellipsoid are constructed and finally rendered. Section 5 and 6 will then present results and give conclusions.

2. Previous Work

Anti-aliasing has been a well-studied research field in computer graphics. Aliasing occurs whenever the sampling rate falls below the Nyquist rate. In rendering, aliasing effects can be prevented by using smooth filters for interpolation. Specifically, point-based rendering reduces aliasing by representing the point samples as extended kernel functions,

such as Gaussians, tents, or squares (see, for example, [RL00][PZvG00][ZPvBG02].

Our paper extends anti-aliasing by ways of extending kernels into the temporal domain. This gives rise to a variety of motion-blur effects, which are able to give suggestive hints of the object in motion.

Motion blur is an intuitive way to create temporally anti-aliased renderings of graphics objects. With respect to the different underlying geometric models, motion blur techniques can be applied to polygonal models, volumetric objects, particle systems, or images. Polygon-based techniques focus on creating motion-blurred renderings of polygonal surface models. Volume-based techniques use 3D volumetric models for motion blur generation and the final rendering. Particle systems are often used to model fuzzy objects, and due to the similarity with polygon-based techniques, this method is discussed in Section 2.1 along with polygonal methods. Finally, the fourth category is to post-process a rendered image and simulate the motion blur effects in the synthesized image.

2.1. Polygon-based Techniques

The rendering equation for polygon models under motion is (see [SPW02] for further detail):

$$i(x,y,t) = i(w,t) = \sum_l \int_\Omega \int_T r(w,t) g_l(w,t) L_l(w,t) dt dw. \quad (1)$$

Here, Ω is the solid angle of the viewing cone emerging from the pixel at (x,y), T is the time at which the camera shutter is open, and l counts the number of objects in the scene. The function r is a camera-dependent reconstruction filter, which describes its shutter geometry as a function of time, g is a function describing the observed geometry (usually a value 1 when visible, and 0 otherwise), and L is the object's luminance over time. Following [SPW02], we can categorize existing motion blur techniques on polygonal models based on the different terms in Equation (1).

Monte-Carlo based methods [CPC84][CCC87][Coo86] [DW85][LRU85] use statistical super-sampling to approximate the entire integral of Equation (1). [HA90] used hardware to accelerate this method with multi-pass z-buffer scan conversions. Super-sampling is inevitably slow since it needs to process more samples, and also, the number of samples required is often adaptive and thus hard to predict. For this reason, with Monte-Carlo based methods it is difficult to achieve real-time constant rendering rates.

Assuming constant shading in Equation (1), some methods solve for visibility, that is, the geometry term. [KB83][Gra85] search for visible geometric objects and display them. On the other hand, geometric morphing methods strategically deform or introduce new geometry from available geometry based on the actual motion. In their particle systems, [Ree83] renders the particle points as line segments along their motion trail to model fuzzy objects. [Cat84] solves the visible surface problem at each screen pixel independently, while [WZ95] creates a transparent motion volume to approximate the motion blur of constant-colored opaque objects.

[NRS82] addresses the shading problem by proposing an object space method that interpolates between samples and provides a continuous transition from a sampled signal to another signal. Their method can be naturally extended for motion blur generation. This work does not consider the visibility problem.

Sung et al. [SPW02] observe from [Gra85][KB83] that, in general, it is not possible to solve the shading and visibility problem separately in temporal and spatial domain, thus they separate the shading problem from the visibility problem. This method generates high quality images, however, since it requires adaptive super sampling, the rendering speed is limited and cannot be guaranteed.

2.2. Volumetric Techniques

Mueller et al. [MMSI*98] describe a framework for motion blur generation in the context of splatting-based volume rendering. For volume rendering, the geometric visibility problem persists when semitransparent or opaque compositing is used. However, for X-ray-like or emission volume models, Max [Max95] and Crawfis et al. [CMB94] have shown that the compositing order is immaterial. Mueller et al.'s method simplified the problem and did not address the visibility problem, but only the integrated energy across the time domain. The elongated Gaussian splat is constructed as a rectangle that spans the motion vector of a voxel, with two half-spherical Gaussian splats at both ends. This idea is similar to [WZ95] by creating a motion volume for each geometric elements.

2.3. Post-processing Techniques

Post-processing techniques [Max90][ML85] [PC83][Shi93] [SSC03] operate on the rendered images, and can be applied to any geometric model. Post-processing techniques only address the 2D problem (extended by [ML85][Shi93] for 2.5D problems) and are capable of creating high quality motion blurred images. [CW93] requires a per-pixel correlation and requires significant computation time. [PC83] proposes an approach of producing motion blur images by time convolution of the normal image with the motion function. However, in general, solutions to post-processing approaches cannot adapt to local properties of the images, and cannot address the situation where motion objects cannot be separated into non-overlapping layers in depth.

3. Construction of Ellipsoids for Motion Blur

3.1. Motivation

One of our goals was to extend the work by Mueller et al. [MMSI*98], which presented an algorithm for the generation of motion blur of volumetric objects. However, we felt that rendering all object voxels motion-blurred, even the interior ones, does not give the viewer more information on either the motion, or the shape of the volume. In addition, it would also increase the rendering time substantially. Thus, instead of rendering motion blur for the entire volume, we decided to only consider a specific iso-surface of the volume as a time. This iso-surface can be selected by the user at run-time. Once the iso-surface has been selected, we can use point-based surface rendering instead of full volume splatting. We used the method described in [BC03] for creating a point-based surface model from a volumetric iso-surface.

Different from most of the previous work, our emphasis is to generate images that provide strong motion hints for the viewer quickly, instead of trying to create theoretically correct motion blurs. The only mathematically strict method for generating realistic motion blur is to render the scene multiple times and then composite the rendering results. All existing techniques attempt to simulate this effect by making assumptions and approximations. However, when we consider the nature of motion blurred images, they do not offer a clear view of the geometric shape of the objects, or even a hint of the direction of the motion. We feel that this aspect, along with efficient rendering capability, is very important for visualization.

To render motion blurred images that at the same time offer hints of object property and motion direction requires some study of conventional photography art. To accomplish this goal, photographers would clearly make use of the flash and the shutter. In Figure 1, one football and one basket ball drop from the hands above. To capture this motion in a single 2-dimensional picture, the photographer would first shoot the picture in a room with only a small amount of light. After the shutter has been open for a long enough time to capture the motion trail, the photographer gives the scene a strong flash for a very short time interval and captures the geometric and material information of the object, as if the object stays still in that interval. Thus the overall picture retains both the object and the motion information.

Our approach can accomplish these effects by sending two primitives down the rendering pipeline: one for realistic rendering of the surface object, and the other to simulate the motion trail.

3.2. Point-based Surface Rendering

The extraction of iso-surface voxels from the volume generates a surface representation of the object. Many techniques exist for point-based rendering of surface models. We

Figure 1: *Photographer's techniques to generate motion blur images while maintaining the clear shape of objects and the hint of motion direction.*

choose to use a simplified version of EWA surface splatting [ZPvBG02]. Instead of going through the expensive computation of the elliptical Gaussian texture mapping, we use one 2-dimensional round Gaussian splat that is perpendicular to the normal of this point. We render this round splat as a texture-mapped square in the space, and thus resort to OpenGL to do the projective transformation. An even more simplified approach is to consider each surface point as a Gaussian sphere, whose projection onto the screen is a disk. This can be combined with a billboard technique to achieve fast rendering rates. However, the latter technique does not offer the same image quality as the former one, as shown in Figure 2, and we choose to use the normal-oriented splats.

Finally, in contrast to the 2-pass approaches of [PZvG00][ZPvBG02][ZPvG01] which use a z-buffer method in conjunction with blending to determine occlusion, we first bucket-sort the points with respect to the viewpoint and then render them in front-to-back order.

3.3. Motion Trail from Gaussian Disks

Now that we have the rendering method to visualize the sharp point-based object extracted from the volumetric object, we need to create the motion trail from these points. Here we perceive the motion object to be generated from low-passing the object along the fourth, that is, the time, dimension. This makes some assumptions on the linearity of the motion path. But since we consider each atomic object

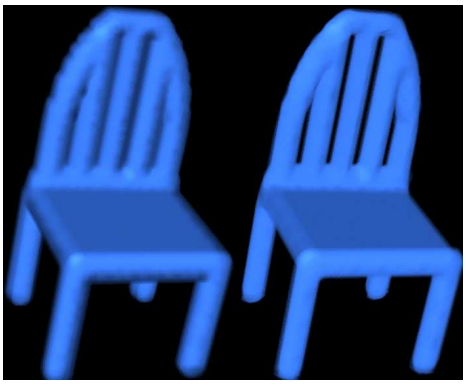

Figure 2: *Different point-based rendering primitives: Gaussian sphere (left) and normal-oriented Gaussian disks (right). Bucket-sort is used to ensure correct ordering.*

point separately, this is only a mild assumption. Thus, rotations of an object can be easily accomplished by computing each point's trajectory as the local path in the global rotation. Translations and rotations can be combined by adding the vectors at each point.

The Gaussian filter is a good anti-aliasing filter, and we shall use it here to accomplish the temporal blurring. Figure 3 illustrates a circular (Gaussian) point moving from P_1 to P_2, passing through P_0. Low-passing this sequence with a temporal Gaussian centered at P_0 produces a Gaussian ellipse. This would generate motion blur. To obtain a motion trail, giving the desired motion hint, one would place its center slightly behind P_0. Rendering both the point at P_2 and the ellipse would then generate a picture similar to Figure 1.

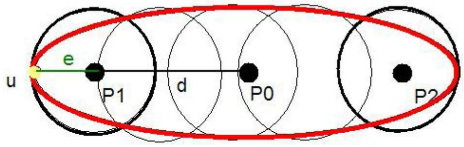

Figure 3: *During a given time interval, a point moves from P_1 to P_2, where P_0 is the middle point along the line segment between P_1 and P_2.*

The just described method can be readily extended into 3D, where a moving spherical point would produce an ellipsoid. Although this train of thought is valid for volume rendering with splatting, where objects are indeed made up of spherical Gaussians, this is not true for surface point-based objects that are constructed from Gaussian disks. We need to be more specific when creating the motion ellipsoid with normal oriented Gaussian disks as point primitives. Consider Figure 4, where we show the settings of the geometry. The motion vector means that in the given time interval, the surface point moves from its current position to the other end of the motion vector (or, P_1 to P_2 in Figure 3).

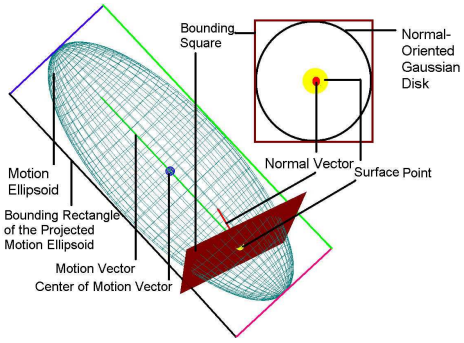

Figure 4: *Ellipsoid Generation and Rendering*

Now given the disk's normal and the motion vector, we need to create the motion ellipsoid that simulates the low-passed motion volume created by the swiping of the normal-oriented Gaussian disk. One can construct this ellipsoid in 3D using the screen-space transformed normal and motion vectors, followed by its projection into a screen-space ellipse. This requires two cross-products and one dot product to form the matrix T that transforms an axis-aligned unit ellipsoid I_3 into the motion ellipsoid, a 3×3 variance-covariance (VC) matrix:

$$M_{ellipsoid} = T \times I_3 \times T'.$$

The screen-space ellipse can be obtained by dropping the ellipsoid's last row and column [ZPvBG02]. The eigenvalues and eigenvectors of this 2D matrix determine the orientation and stretch of the polygon onto which the Gaussian footprint is mapped for projection as follows. Given a general ellipse in a plane with a variance matrix

$$V_2 = \begin{pmatrix} A & D \\ D & B \end{pmatrix},$$

we can compute the two half vectors that define its bounding rectangle (represented by the polygon). The two eigenvalues of matrix V_2 are

$$\lambda_1 = \left(A+B-\sqrt{(A-B)^2+4D^2}\right)/2$$
$$\lambda_2 = \left(A+B+\sqrt{(A-B)^2+4D^2}\right)/2$$

Let

$$w = \frac{1}{\sqrt{(\lambda_1 - B)^2 + D^2}},$$

we have $\sin\alpha = wD, \cos\alpha = w(\lambda_1 - B)$. The length of the major and minor axes are $a = 1/\sqrt{\lambda_2}$, $b = 1/\sqrt{\lambda_1}$. Let **right**

be the long axis and **up** be the short axis, we have

$$\mathbf{right} = [-a\sin\alpha, a\cos\alpha]$$
$$\mathbf{up} = [b\cos\alpha, b\sin\alpha]$$

We need 4 square roots, 2 divisions and 20 multiplications.

An alternative and more efficient method skips the 3D construction of the motion ellipsoid entirely and instead constructs the 2D motion ellipse from a convolution of the 2D Gaussian functions of the projected point and the projected motion vector. This convolution can be conveniently performed by adding their VC matrices [ZPvBG02]. We obtain the point's 2D VC matrix by first transforming its object space 3D VC matrix into screen space, taking into account the transformation due to motion, and then dropping the last row and column of the resulting matrix. Note that the 3D VC matrix has the shape of an ellipsoid with zero width in the disk's normal direction, giving rise to 2 non-zero eigenvalues. The world-space motion vector is projected to the screen as well, and its VC matrix (bounded by an ellipse of zero width and having one non-zero eigenvalue) is built using the orientation and length of the 2D vector. Adding these two matrices results in the projected motion ellipsoid, from which we extract the eigenvectors and values for orientation and scale of the polygon that maps the texture of the Gaussian footprint using the equations outlined above.

To obtain different rendering effects, we can vary the length of the motion ellipse, move it closer or farther away from the object's point primitive, or generate a number of motion ellipses of identical orientation, but shorter along the motion direction and spaced apart at a fixed interval on the motion axis. In the latter variation, we can render the point's motion trail in a sequence of different colors, ranging from, say, red to white. This is just to emphasize that the approach is quite general and allows for a lot of "playing around", once we have a framework for rendering and projecting ellipsoids. This will be described next.

4. Point-Based Rendering with Motion Trails

To create real-time motion blur when the user moves the object, we need the motion vector for each point in each time step. Assume that the current and previous **ModelView** matrices are M_{curr} and M_{prev}, respectively, and the position of a point in the object space is p, the motion vector in the world coordinate system is:

$$motion_{world} = M_{curr}p - M_{prev}p = (M_{curr} - M_{prev})p.$$

The motion vector in the object space is

$$\begin{aligned} motion &= M_{curr}^{-1} motion_{world} \\ &= M_{curr}^{-1}(M_{curr} - M_{prev})p \\ &= (I - M_{curr}^{-1}M_{prev})p. \end{aligned}$$

Thus we only need to compute this matrix once for each frame.

We can easily extend our texture-based motion trail generation to other textures beyond Gaussian texture or other colors, as shown in Figure 7, to create interesting non-photorealistic (*NPR*) rendering effects (Figure 8).

The overall rendering algorithm works as follows. First, a surface representation of a volume is extracted based on a user-specified iso-value. This yields a collection of circular Gaussian splats, rendered as polygons. When viewed under motion, each point also gives rise to an elliptical Gaussian splat. We maintain a bucket list which orders the object and motion splats front to back to resolve occlusions.

Alternatively, one may also maintain two bucket lists, one for the motion splats and one for the object splats. Using these separation of points, motion and object, we would render two images: one for the motion trail, modelling a long-open photographic lens, and one for the object captured by the final flash photography. Adding these two images together is synonymous to what happens on filmed motion trails. One can de-emphasize motion trails falling within the extent of the object by assigning a high opacity to the object points and alpha-blending the object and the motion image in that order. Alternatively, one can also displace the motion ellipse far from the object point, which means that the camera was closed for some time until the flash picture was taken, as shown in Figure 6 (c). Merging the two bucket lists will provide images in which motion trails of distant object parts are occluded by closer object portions.

Alpha-compositing is always used to render the depth-sorted object points, but either adding or alpha-blending can be used to render the motion splats. The former is more true to the photographic model we are striving to simulate, while the latter may provide better blending. The influence of a point on the motion trail image is strongly related to its reflected light. For uni-colored objects, we can model this by modulating the motion splat's intensity by the dot product of the normal and the light vectors (Figure 6 (b)). A more general solution would modulate the motion splat's intensity by the result of the complete shading equation. This would generate motion trails that show a mix of the object colors (see Figure 1 and Figure 6 (c)).

We do not generate motion splats for points with back-facing normals. We also provide the option to modulate the intensity of the motion splat by the dot product of point normal and viewing vector. This ensures that points with small screen footprints will not contribute much to the motion image.

5. Results

We implemented the algorithm on a computer with Pentium 1.2G CPU, 512M memory, and ATI Radeon 9700 Graphics

Card. The rendering speed for all volume datasets shown below has a frame rate of above 10 fps, thus we consider the algorithm as interactive. Due to the smoothness of the splats we have not observed strobing in animated viewing.

Figure 5 shows a simple example: a rotating rod and a translating rod, rendered with a white motion trail to suggest its motion. Figure 6 shows some motion trails when a solid sphere moves. Figure 7 shows a dataset of a cerebral vessel with an aneurism, undergoing various motions, rendered with different motion trail effects. This, for example, could be used to suggest the trajectories of vessels in a beating heart. Finally, Figure 8 shows some engine parts undergoing different kinds of motion and Figure 9 shows a dropping engine block.

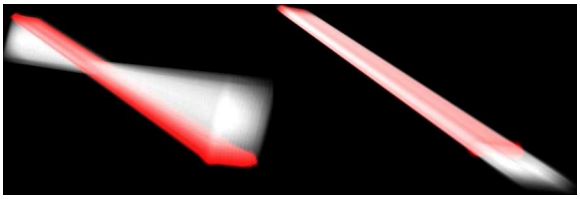

Figure 5: *Rotation and translation of a rod.*

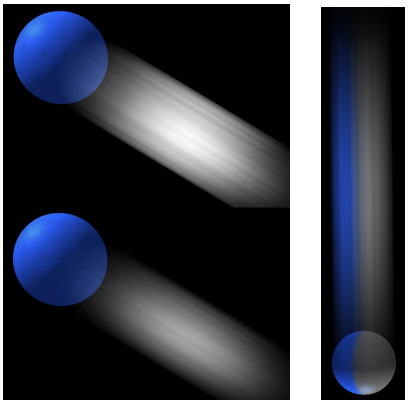

Figure 6: *Sphere. Upper Left: (a). without shading or dot product modulation. Lower left: (b). without shading but with dot product modulation. Right: (c). with shading but without dot product modulation. The motion trail is displaced from the* 128^3 *object by* 50.

6. Conclusions

This paper presents an original technique to simulate motion blur and motion hints for point-based objects. By illustrating to the viewer a sharply rendered object and the direction of its motion, it provides strong motion hints without diminishing the visual appearance of the moved object. Since the object itself is not blurred, we do not encounter the problems associated with a temporally changing shading function.

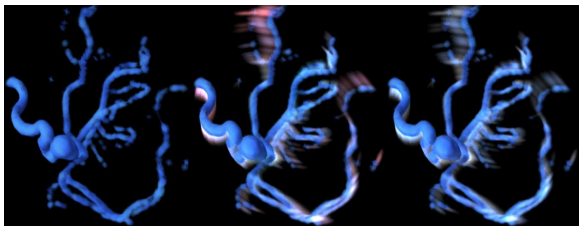

Figure 7: *Point-based rendered vessel dataset. Left: Original dataset. Right: with rotational motion blur. Middle: with color-coded trails to disambiguate if the illustrated length of motion trails in the image was due to the motion or the viewing angle. Redder trails have larger motion.*

A current downside of our approach is that the visibility function of the moving objects is not properly handled. Although we perform depth sorting of all points, surface points and motion ellipsoids, it is absolutely possible that there are motion ellipsoids that cross each other in temporal space, which is particularly true for rotations. This can be handled by using an image-aligned slice-based rendering technique, such as the one described in [MC98]. It would slice the 3D motion ellipsoids and not use the 2D elliptical projections. With this functionality, the visibility part of Equation (1) would also be correctly solved. It would enable correct semi-transparent viewing of object parts hidden in the motion blur of more front-facing objects, using alpha-compositing. A slice-based rendering technique would also enable better post-shading rendering effects. Finally, in order to also model the change of shading, which is dependent on the orientation of the normal vector, we plan to associate the point normals with the motion ellipses. Shading would then be performed as a post-process on these blurred normals. Future work is directed towards these directions.

Our approach works well for the translational motion of the Gaussian disks, which is easily modelled by ellipsoids and ellipses. On the other hand, the trajectory of points under rotation follows a curve – giving rise to a bent motion ellipsoid for the Gaussian disks. We currently only handle small rotations, where motion vectors can be approximated by a straight line. One way to deal with larger rotations could be to bend the screen space motion ellipse, centered at an average orientation, by the projected rotation trajectory. In that case, translations and rotations would be combined by updating the 3D rotation curve by the translation vector. The rendering would use a polygon strip approximating the shape of the bent ellipse, with the Gaussian texture mapped onto it.

At the current time we do not check during motion splat generation if the point is actually visible from the view, at least partially. This means that front-facing points hidden in concavities will still give rise to motion splats, yet they are invisible in the object image. This, of course, will not be the case in motion trail photography. Visibility splatting

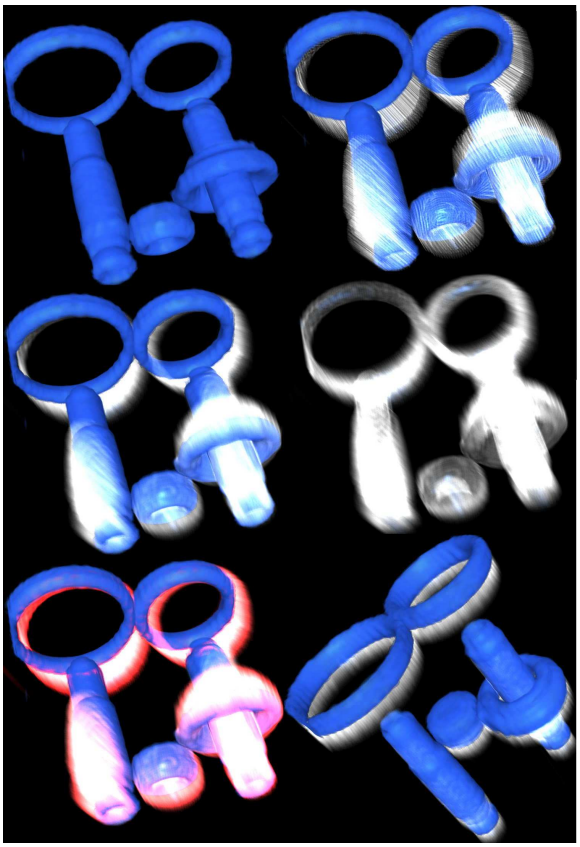

Figure 8: *Point-based rendered engine dataset with* 37520 *surface points rendered. First row: Left: Original dataset. Right: rotational motion displayed as streak lines. Second row: Left: with rotational motion. Right: motion blur trail only. Third row: Left: rotational motion with color-coded trails. Right: translational motion.*

[PZvG00] could provide help in this effect. Here, the intensity of the motion splat, and even its width, could be determined by the z-buffer pixel ID's. Hidden splats would not produce a motion splat at all.

We would also like to experiment with other textures to model motion blur with NPR effects, similar to those used in cartoons. Furthermore, it may be interesting to use dynamic multi-resolution representations of volume or surface models, so that instead of low-passing the trajectory of each point, we could low-pass each part in the desired layer of resolution. Finally, we would like to extend the approach presented in this paper to time-varying volumes, where we could estimate the local motion trail by computing the voxel motion vectors using optical flow or mpeg-style motion prediction.

Figure 9: *A dropping engine.*

Acknowledgments

This work was supported by NSF Career Grant ACI-0093157. We would like to thank the paper reviewers, in particular reviewer 3, for their valuable comments.

References

[BC03] BRENTZEN J. A., CHRISTENSEN N. J.: Hardware accelerated point rendering of isosurfaces. In *The* 11*th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision* (2003). 3

[Cat84] CATMULL E.: An analytic visible surface algorithm for independent pixel processing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (1984), ACM Press, pp. 109–115. 2

[CCC87] COOK R. L., CARPENTER L., CATMULL E.: The reyes image rendering architecture. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), ACM Press, pp. 95–102. 2

[CMB94] CRAWFIS R., MAX N., BECKER B.: Vector field visualization. *IEEE Computer Graphics and Applications 14*, 5 (Sept. 1994), 50–56. 2

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Trans. Graph. 5*, 1 (1986), 51–72. 2

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (1984), ACM Press, pp. 137–145. 2

[CW93] CHEN S. E., WILLIAMS L.: View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM Press, pp. 279–288. 2

[DW85] DIPPÉ M. A. Z., WOLD E. H.: Antialiasing through stochastic sampling. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (1985), ACM Press, pp. 69–78. 2

[Gra85] GRANT C. W.: Integrated analytic spatial and temporal anti-aliasing for polyhedra in 4-space. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (1985), ACM Press, pp. 79–84. 2

[HA90] HAEBERLI P., AKELEY K.: The accumulation buffer: hardware support for high-quality rendering. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (1990), ACM Press, pp. 309–318. 2

[KB83] KOREIN J., BADLER N.: Temporal anti-aliasing in computer generated animation. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques* (1983), ACM Press, pp. 377–388. 2

[LRU85] LEE M. E., REDNER R. A., USELTON S. P.: Statistically optimized sampling for distributed ray tracing. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (1985), ACM Press, pp. 61–68. 2

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

[Max95] MAX N.: Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (June 1995), 99–108. 2

[MC98] MUELLER K., CRAWFIS R.: Eliminating popping artifacts in sheet buffer-based splatting. In *Proceedings of the conference on Visualization '98* (1998), IEEE Computer Society Press, pp. 239–245. 6

[ML85] MAX N. L., LERNER D. M.: A two-and-a-half-d motion-blur algorithm. *SIGGRAPH Comput. Graph. 19*, 3 (1985), 85–93. 2

[MMSI*98] MUELLER K., MOLLER T., SWAN II J., CRAWFIS R., SHAREEF N., YAGEL R.: Splatting errors and antialiasing. *IEEE Transactions on Visualization and Computer Graphics 4*, 2 (apr – jun 1998), 178–191. 2, 3

[NRS82] NORTON A., ROCKWOOD A. P., SKOLMOSKI P. T.: Clamping: A method of antialiasing textured surfaces by bandwidth limiting in object space. In *Proceedings of the 9th annual conference on Computer graphics and interactive techniques* (1982), ACM Press, pp. 1–8. 2

[PC83] POTMESIL M., CHAKRAVARTY I.: Modeling motion blur in computer-generated images. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques* (1983), ACM Press, pp. 389–399. 2

[PZvG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 335–342. 1, 2, 3, 7

[Ree83] REEVES W. T.: Particle systemsa technique for modeling a class of fuzzy objects. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques* (1983), ACM Press, pp. 359–375. 2

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: a multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 343–352. 1, 2

[Shi93] SHINYA M.: Spatial anti-aliasing for animation sequences with spatio-temporal filtering. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM Press, pp. 289–296. 2

[SPW02] SUNG K., PEARCE A., WANG C.: Spatial-temporal antialiasing. *IEEE Transactions on Visualization and Computer Graphics 8*, 2 (April – June 2002). 1, 2

[SSC03] SHIMIZU C., SHESH A., CHEN B.: Hardware accelerated motion blur generation. *EUROGRAPHICS 22*, 3 (2003). 2

[WZ95] WLOKA M. M., ZELEZNIK R. C.: interactive real-time motion blur. *The Visual Computer*, 12 (1995), 283–195. 2

[ZPvBG02] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: EWA splatting. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (jul – sep 2002), 223–238. 1, 2, 3, 4, 5

[ZPvG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM Press, pp. 371–378. 3