Fast Motion Blur through Sample Reprojection

Micah T. Taylor
taylormt@cs.unc.edu

Abstract

The human eye and physical cameras capture visual information both spatially and temporally. The temporal aspect leads to a blurring effect seen on moving objects. However, computer image synthesis is limited to creating samples at discrete time points. To enhance images with motion blur effects, temporal sampling must be added to the image rendering process. In this paper, I describe a simple, novel motion blur algorithm using sample reprojection. The algorithm has high performance compared to typical distributed motion blur ray tracing while maintaining comparable image quality for simple transforms.

1 Introduction

Since all physical image capture devices capture events over some time scale, the resulting image is subject to blurring if an object moves during the exposure. Computer generated images are sampled at discrete times and are not subject to this effect. Thus, adding motion blur effects to computer generated images is important if realistic images are to be created.

In this paper, I describe a simple technique for generating motion blur effects in rendered images. This is done by reprojecting image space samples between two consecutive frames. Since the method only uses samples that have already been rendered , the algorithm cost is dependent on the resolution of the image. Since the cost to compute reprojections is small, this method can generate images that approximate time sampled images with low cost.

2 Previous Work

There has been much prior work on generating realistic motion blur in images. These techniques can be broadly divided based on whether they operate in world space or as a post-process in image space.

Techniques operating in world space generally give high quality results. However, since they rely on greater sampling density, the time cost is often high. Distributed ray tracing in the time dimension [4] produces excellent blurring at very high costs. Another common method is to accumulate successive render frames to estimate the blur. Given a large number of samples, this method has the same quality as distributed sampling, but unpleasant visual artifacts arise when using few samples. Such methods are being accelerated by recent advances in interactive ray tracing [1].

To avoid the high cost of calculating new samples, many post-process image space techniques have been proposed. All such methods rely on some combining algorithm to produce blurred images from little temporal data. Line integral convolution has been used [2] to produce motion blurred images. Techniques using graphics hardware [6] can operate in real-time, but use only a single color frame in the blurring process and do not capture the full range of motion of the object due to blending artifacts.

My work is most similar to the method presented by Chen et al. [3]. They propose using depth and color samples to create new frames. This is done by precomputing optimized morphing maps between frames. The claim that simple rasterization of the point samples is too computationally intensive is no longer the case, given modern hardware, as shown in my work.

The described point rasterization is nearly identical to sample reprojection. Sample reprojection is the process of reusing a previously rendered sample in a new image frame. This is done using the 3d world space location of the sample. If such data is available, the color value of the sample can be easily reprojected with a new camera position. Reprojection has been used in many ray tracing engines to decrease rendering time while retaining high quality output. By reprojecting and post-processing the data, old sample data can be enhanced to provide nearly complete image frames [7]. Advanced sampling and reprojection algorithms have been used to create very responsive rendering engines [5].

The method I propose is a pairing of image space blurring with sample reprojection. Reprojection is used to generate the high number of samples that accurate motion blur requires. However, since reprojection takes place in image space, the actual scene geometry need not be sampled during this process. The result is motion blur effects that approximate high temporal sampling, but have low cost.

3 Algorithm

The motion blur algorithm works by interpolating between camera positions and reprojecting many samples to new positions in the frame buffer. When many samples are reprojected from different cameras, an approximation of the motion integral can be formed.

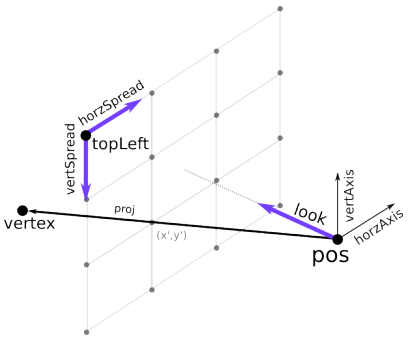


Figure 1: The vectors used in the reprojection process.

Sample Reprojection: For sample reprojection, I will assume a simple camera and image plane described by several vectors. Figure 1 provides a visual guide to the reprojection process. When creating a ray from the camera, I assume a precomputed top left corner of the image plane, $\mathbf{topLeft}$, in world space and samples at intervals in world space of $\mathbf{horzSpread}$ horizontally and $\mathbf{vertSpread}$ vertically to create the rays. Creating a ray given an x and y position in the image plane is then:

$$\mathbf{ray} = \mathbf{topLeft} - \mathbf{horzSpread} \cdot x - \mathbf{vertSpread} \cdot y$$

To reproject a sample $\mathbf{vertex}$, we first compute the projection vector from the camera position, then scale the projection to bring it into the image plane. The projection from the top left corner is then used to find the horizontal and vertical offsets from the corner.

$$\mathbf{proj} = \mathbf{vertex} - \mathbf{pos}$$

$$lookScale = \mathbf{proj} \cdot \mathbf{look}$$
$$\mathbf{scaledProj} = \frac{\mathbf{proj}}{lookScale}$$
$$\mathbf{planeProj} = \mathbf{topLeft} - \mathbf{scaledProj}$$

$$horzScale = \mathbf{horzAxis} \cdot \mathbf{planeProj}$$
$$vertScale = \mathbf{vertAxis} \cdot \mathbf{planeProj}$$

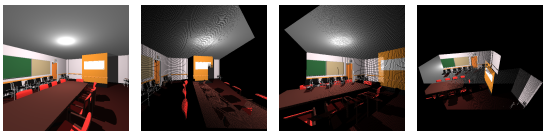
$$x = \frac{horzScale}{horzSpread}$$
$$y = \frac{vertScale}{vertSpread}$$

Where the values are in world space as follows: $\mathbf{pos}$ is the camera position, $\mathbf{look}$ is the normalized look direction, $\mathbf{topLeft}$ is the position of the top left corner of the image plane, $\mathbf{horzAxis}$ and $\mathbf{vertAxis}$ are the basis axis of the camera, and $horzSpread$ and $vertSpread$ correspond to the respective lengths of the image plane in world space by half of the image space lengths.

This results in image space coordinates x' and y' to which the vertex may be projected, but the location must be verified to lie in the image plane before writing.

Frame Projection: Given an image buffer, corresponding vertex data, and a new camera position, a new image frame can be generated by reprojection. Reprojecting each sample from the original image results in a new image space sample location; performing this on all samples results in a new motion frame from the given camera position. Figure 2 shows an example of a scene and several reprojected views.

When reprojecting, each sample coordinates must be rounded and verified to lie in the image plane. If desired, additional validation can also be performed by testing



(a) Initial (b) New view (c) New view (d) New view

Figure 2: The image space samples are reprojected according to their world space coordinates.

that sample normals face the camera and that samples pass the depth test for the new frame. When reprojecting for motion blur, these tests are not performed due to their cost.

Motion Blur: Motion blur effects are achieved by projecting many new frames into a image buffer. This is done by tracking camera positions, color data, and vertex positions for frames $n-1$ and n. Using this data, two reprojection cycles take place: the image buffer from frame $n-1$ is reprojected towards camera n and the image buffer from frame n is reprojected towards camera $n-1$.

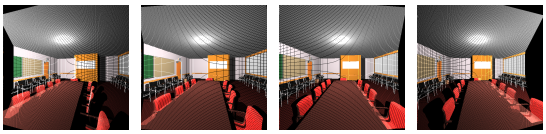


Figure 3: Sequence of reprojected views at interpolated camera positions.

Each reprojection cycle is completed by interpolating from the starting camera data to the final camera data. The number of interpolation samples is user controlled. For each interpolated camera position, a new color frame is generated by reprojection.

All new frames are accumulated in an image buffer. As samples are written to positions in the buffer, a reprojection count is kept for each location. When all cycles are complete, the color at each position is scaled based on the number of reprojections. This is done by accumulating the counts in a buffer, then looking up the scaling value in a precomputed table (see Figure 4).

When all cycles are complete, all frame data is shifted to prepare for the next frame. Since the the data from the current frame is combined with the current frame and the previously rendered frame, the blur effect is between frame $n-1$ and n.

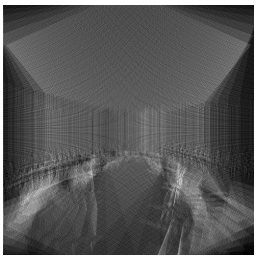


Figure 4: The accumulated reprojection count. The color intensity must be scaled according to this map.

4 Results

Since this is an image space operation, it is independent of scene complexity and shading cost, rather, it is limited by image resolution. At the end of this report, I show examples of motion blur by distributed ray tracing and frame blending. For each scene, I compare the blur effects given a time budget of 2 seconds and 10 seconds. The reference image is motion blur by distributed ray tracing with 1000 samples per pixel. All images are generated on a CPU using 2 threads at 2Ghz.

While the quality of the other methods increases with larger time budgets, reprojection blurring quickly reaches a quality limit. Nonetheless, for small time budgets, reprojection motion blur provides much smoother images. For both translation and scaling transforms, reprojection motion blur quality is comparable to more expensive methods.

Limitations: Since no new samples are generated, view dependent effects, or scene changes that occur between sample points cannot be captured. Also, since the new samples are accumulated in a single buffer, proper depth testing cannot be performed. This can result in the projection of samples that should be occluded given a camera position.

My implementation is used with a ray tracing renderer. As such, I compare it to typical motion blur effects com-

puted with ray tracing. It would be interesting to compare the reprojection method to other image space motion blur effects. However, useful comparison would require either a CPU implementation of the other methods, or an implementation of reprojection blurring on graphics hardware.

5 Conclusion

The presented motion blur algorithm is simple to implement and provides blur effects that correspond directly to motion in the scene. Since the algorithm performs sampling in image space, it is easy to adjust the quality/speed tradeoff by changing the number of motion interpolation samples used.

There are many avenues for future work. Since reprojection is very similar to rasterization, there are likely great performance benefits to be had by implementing the algorithm on graphics hardware. Also, instead of projecting point samples to screen space, using line integrals or line drawing would increase visual quality. An extension of the method to allow for object motion would be necessary before use in general rendering environments.

References

[1] S. Boulos, D. Edwards, J. D. Lacewell, J. Kniss, J. Kautz, P. Shirley, and I. Wald. Packet-based Whitted and Distribution Ray Tracing. In *Proc. Graphics Interface*, May 2007.

[2] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 263–270, New York, NY, USA, 1993. ACM.

[3] S. E. Chen and L. Williams. View interpolation for image synthesis. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 279–288, New York, NY, USA, 1993. ACM.

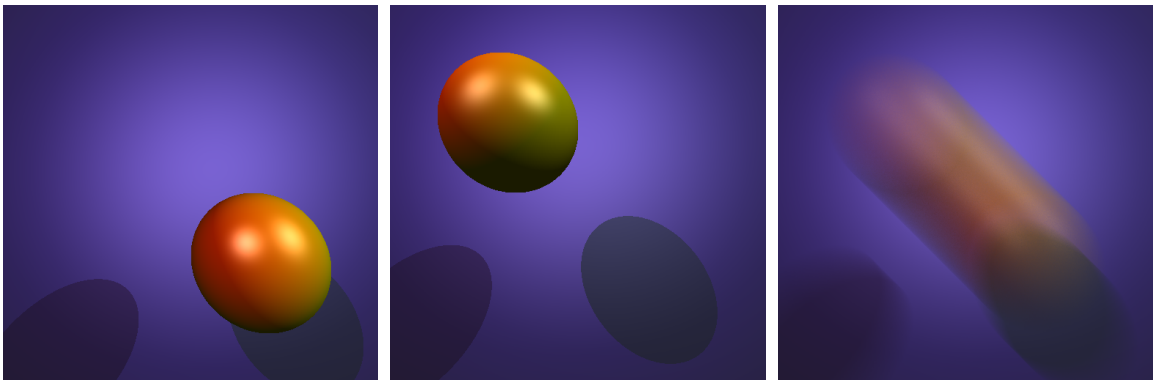
[4] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, 1984.

[5] A. Dayal, C. Woolley, B. Watson, and D. Luebke. Adaptive frameless rendering. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 24, New York, NY, USA, 2005. ACM.

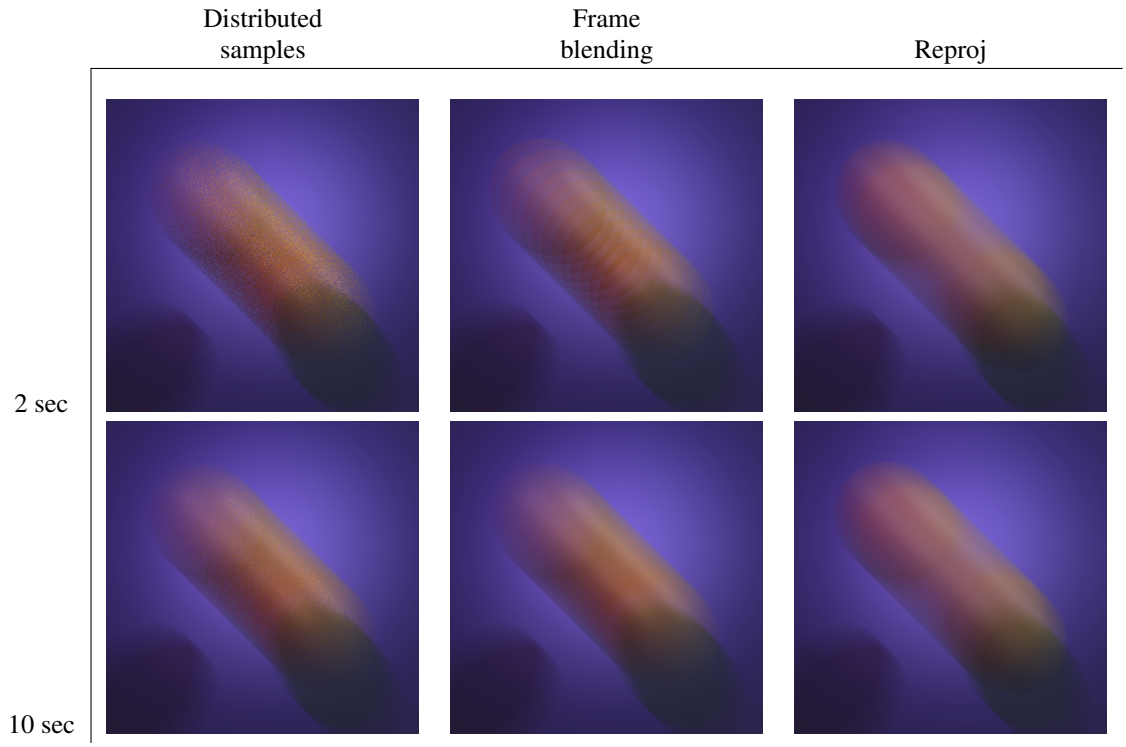
[6] C. Shimizu, A. Shesh, and B. Chen. Hardware-accelerated-motion-blur-generation. In *University of Minnesota Computer Science Department Technical Report 2003-01*, 2003.

[7] B. Walter, G. Drettakis, and S. Parker. Interactive rendering using the render cache. In D. Lischinski and G. Larson, editors, *Rendering Techniques (Proceedings of the Eurographics Workshop on Rendering)*, volume 10, pages 235–246, New York, NY, Jun 1999. Springer-Verlag/Wien.

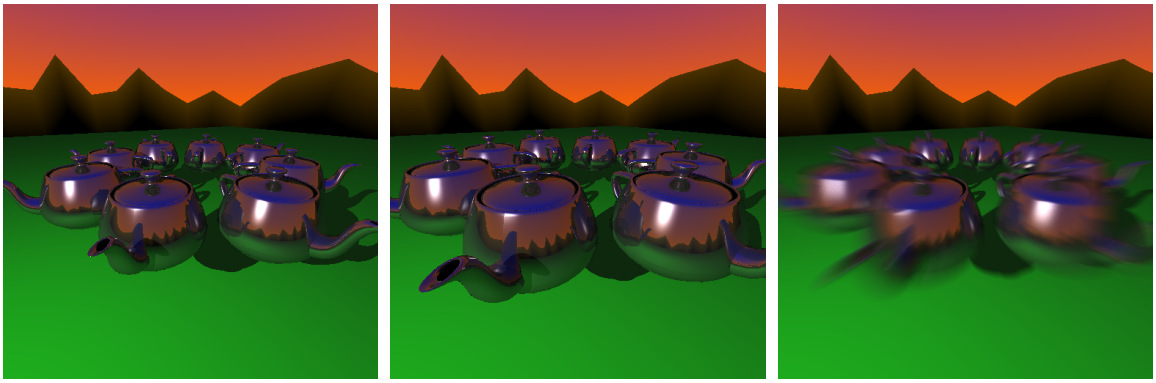
Translation:



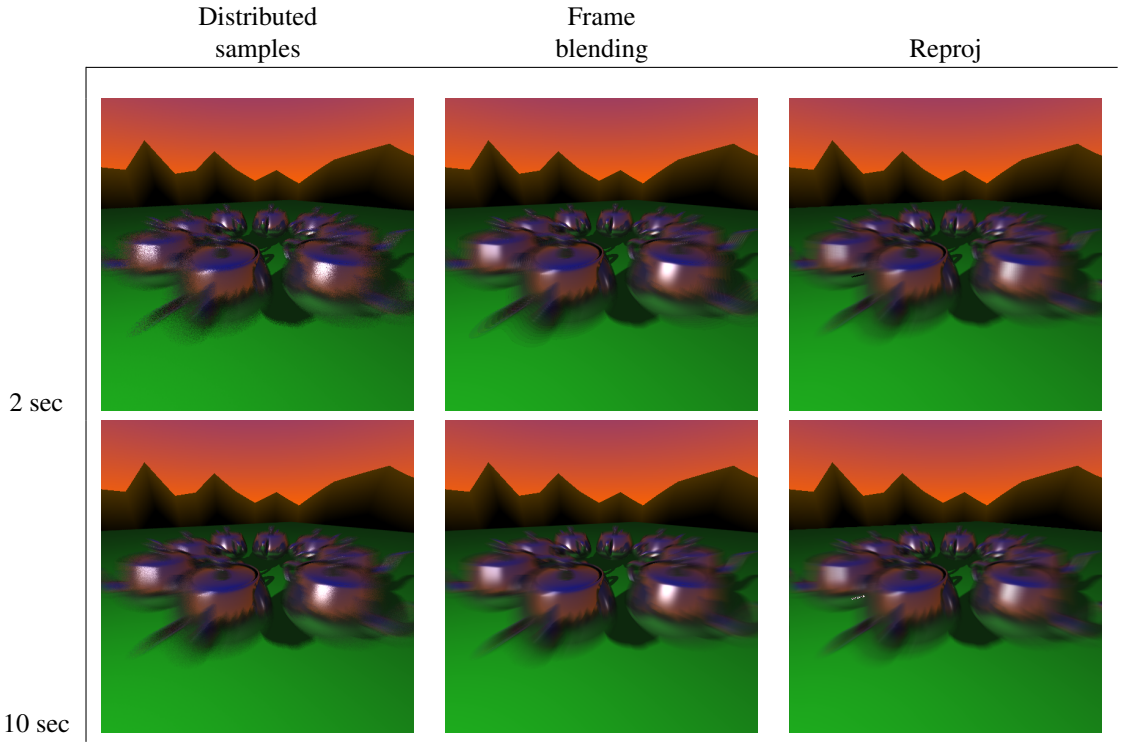
(a) Frame 0 (b) Frame 1 (c) Reference blur



Scaling:



(a) Frame 0 (b) Frame 1 (c) Reference blur



Rotation:

(a) Frame 0 (b) Frame 1 (c) Reference blur