Eurographics Symposium on Rendering - Experimental Ideas & Implementations (2015)
J. Lehtinen and D. Nowrouzezahrai (Editors)

High Performance Non-linear Motion Blur

J.-P. Guertin & D. Nowrouzezahrai

Université de Montréal

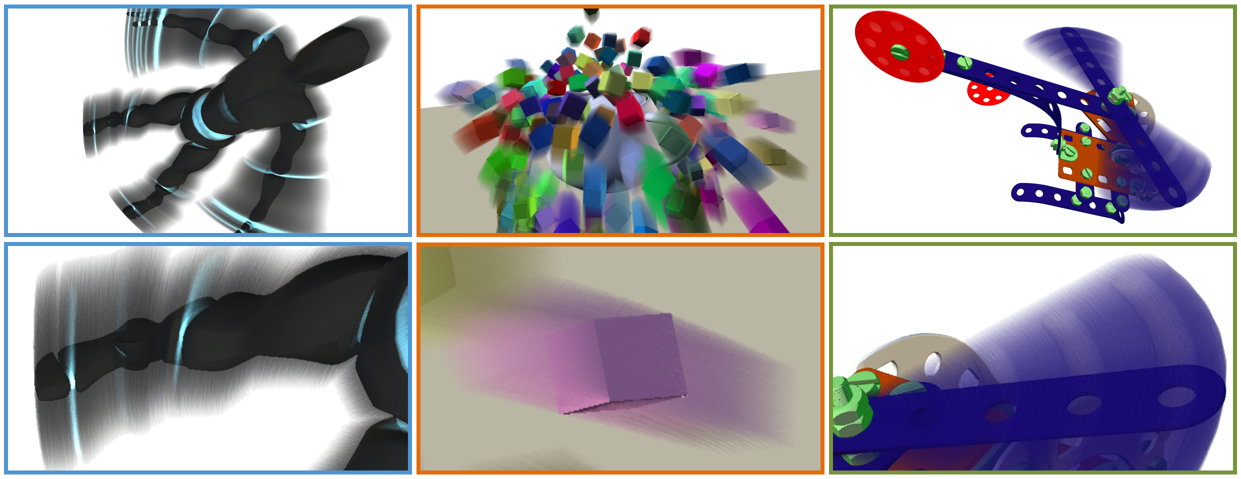

Figure 1: *Smooth motion blur on a variety of complex, potentially non-linear motions, computed in* 3.5 *to* 6.5 *ms at* 1920 × 1080.

Abstract
Motion blur is becoming more common in interactive applications such as games and previsualization tools. Here, a common strategy is to approximate motion blur with an image-space post-process, and many recent approaches demonstrate very efficient and high-quality results [Sou13,GMN14]. Unfortunately, all such approaches assume underlying linear motion, and so they cannot approximate non-linear motion blur effects without significant visual artifacts. We present a new motion blur post-process that correctly treats the case of non-linear motion (in addition to linear motion) using an efficient curve-sampling scatter approach. We simulate plausible non-linear motion blur in 4ms at 1920×1080 and our approach has many desirable properties: its cost is independent of geometric complexity, it robustly estimates blurring extents to avoid typical over- and under-blurring artifacts, it supports unlimited motion magnitudes, and it is less noisy than existing techniques.

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

1. Introduction

Motion blur effects give important visual cues about the dynamics of a scene, and as such they have played an almost essential role in realistic image synthesis for visual effects. More recently, the development of high-performance post-processing techniques for approximating motion blur have led to their almost ubiquitous integration in interactive graphics applications, such as video games.

Despite these recent advances in more efficient and realistic motion blur simulation, almost every existing motion blur solution (including the majority of offline, high-fidelity solutions) assume that the underlying motion of an object is *strictly linear*. This assumption dramatically reduces the complexity of simulating motion blur effects, and is particularly important for high-performance approximations that rely on image-space post-processing.

While, in practice, this limitation can sometimes be disguised by either cleverly crafting an animation sequence, or limiting the virtual exposure to short bursts, it can still lead to very distracting visual artifacts. Avoiding these visual artifacts becomes even harder with the state of the art in interactive motion blur approximations that rely primarily on image-space post-processing. Here, camera and object motion can both very easily combine to cause very jarring visual artifacts, even in scenes with simple motions (e.g., Figure 1).

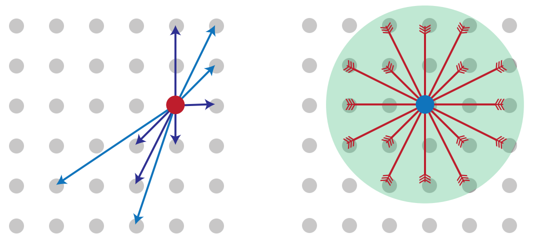

Figure 2: *Visualizing scatter and gather operations. In a scatter-based algorithms, each data point (i.e., pixel) deposits data onto neighboring points; for gather-based algorithms, each point queries its neighbors to compute its final value. Scatter-as-gather emulates the former using the latter.*

We present a high-performance motion blur approximation that gracefully handles linear and non-linear motion, supports long exposure times, does not introduce temporal artifacts under camera motion, and maintains the same advantages of existing techniques: it scales independently with the underlying scene/motion complexity, and it uses a simple post-process that integrates easily into existing engines.

2. Previous Work

We present recent work most related to our approach below, and we forward readers to the comprehensive survey on motion blur [NSG11] for a more complete view of the area.

Offline Sampling. Traditionally, motion blur (and other distribution effects) can be estimated via numerical integration, as presented by Cook in his seminal work on the topic [Coo86]. Recent work on these offline solutions design more elaborate sampling and filtering schemes capable of leveraging the structure of object motion, including multi-dimensional sampling schemes [HJW*08], or adaptive sampling schemes based on wavelet-space [ODR09] or frequency-based [ETH*09] formulations of the motion blur problem. Adaptive sampling can also be combined with anisotropic spatial-temporal filtering [LAC*11].

We are motivated by Reeves' [Ree83] approach, where motion points are advected according to a (world-space) particle system to form motion segments, and a world-space blur is applied to the segments in order to approximate motion blur effects. We instead sample points on a screen-aligned grid and analytically fit motion curves, leveraging the entire programmable rasterization pipeline to efficiently implement a true motion blurring scatter operation.

We target interactive applications, where object-space sampling is not an option and motion blur effects must be computed on the order of milliseconds, not seconds/minutes.

Interactive Approximations. Apart from heuristic object- or texture-space extrusion and sorting approaches [ML85, TBI03, RMM10], many interactive solutions aim to approximate motion blur. Our work falls in this category, and we are most related to image-space post-processing techniques: such approaches are sample, manipulate (e.g., dilated), and blur frame-buffer colors according to screen-space color and velocity information [Sou11, KS11]. Recent tile-based variants segment image-space to more accurately determine blurring directions and neighborhoods, approximating the motion blur scattering operation as a localized gather. Blurring along a single, "dominant" velocity direction [Len10, MHBO12, ZG12, Sou13] is most efficient, however we base our comparisons on the most recent "multi-direction" tile-based post-process approach of Guertin et al. [GMN14], which is capable of resolving many of the tile- and image-space artifacts of previous "single-direction" techniques, but still maintains a very high-performance profile.

We will show that even the most robust high-performance post-process motion blur technique can fail in common scenarios, specifically when non-linear motion exists and/or large motion magnitudes (and/or large exposure times) are used. We are able to generate more accurate and more spatially/temporally coherent motion blur in these scenarios, with only a modest performance overhead: instead of requiring on the order of 1 to 3ms (as in [GMN14]), our (unoptimized) approach requires 3.5 to 6.5ms.

Alternative Rendering Architectures. Recent work on GPU micropolygon rendering [AMMH07] and stochastic rasterization techniques [AMMH07, MESL10] provide a middle-ground between accuracy and performance: object visibility and shading are decoupled, which allows a more accurate motion blur effect compared to interactive post-processes, however at increased cost.

3. Method

Modern interactive motion blur approaches rely heavily on the principle of *scatter-as-gather*, since algorithms design in this manner can readily benefit from accelerated processing on massively parallel modern GPU architectures. Specifically, a scatter operation (such as motion blur), where each pixel $p_{x,y}$ influences the value at one or more pixels $p_{x',y'}$, is implemented as a gather operation, where each pixel $p_{x,y}$ queries pixels in its neighborhood $p_{x',y'}$ to determine their potential contributions (see Figure 2). In the general case, the gather solution would require a neighborhood size equal to the image resolution in order to perfectly simulate the scatter, but a common acceleration strategy reduces this neighborhood size heuristically in exchange for introducing some approximation error. In the motion blur setting, this restriction constrains both the form (i.e., linear vs. non-linear) and the length of motion blur features.

Our method instead directly implements the scatter solution to motion blur, but in a manner that completely avoids

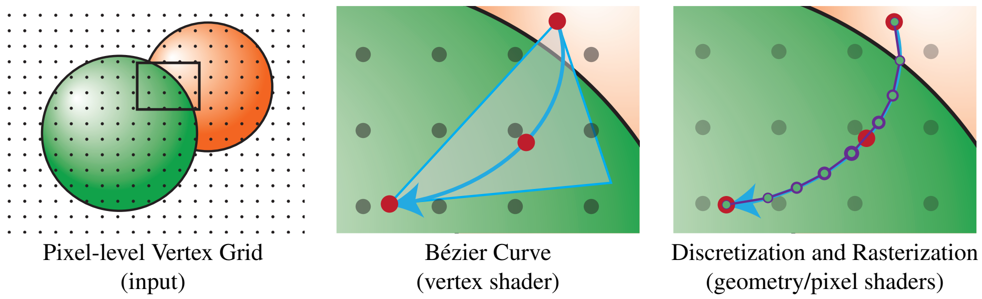

Figure 3: *Grid scatter pass: we fit Bézier curves to a grid of pixel-aligned vertices with a vertex shader, querying an object's previous and next positions. We discretize the curves into line segments in a geometry shader and then rasterize the segments. We compute the spatially-varying line color based on the originating pixel's color and distance-based weight in a pixel shader.*

its principal disadvantage: scattering on a GPU is inherently inefficient since it reduces thread coherence by performing unordered buffer writes. Moreover, unordered buffer writing operations are typically unoptimized at the driver- and hardware-levels since they break the SIMD processing model of GPUs, further increasing their cost in practice. In contrast, *primitive rasterization* is perhaps the most optimized set of routines on a GPU; we exploit the fact that primitive rasterization reduces to a series of unordered buffer writes, and build our optimized GPU solution atop it.

We discuss our rendering approach below. It is divided into three stages: a motion pre-pass, a *grid scatter* pass, and a normalization pass.

3.1. Motion Pre-Pass

Similarly to most post-processed interactive motion blur algorithms, we first use a set of pre-passes to output the necessary motion data. Specifically, we store two buffers with the location of each pixel's geometry in screen space at two different time steps. Each pixel has an associated 3D position and 1D time coordinate, $p(t) = [x_t, y_t, z_t, t]$, with the current frame's pixel $p(0)$ at $t = 0$, and the "previous" and "next" buffers storing $p(-1)$ at $t = -1$ and $p(1)$ at $t = 1$. At render-time, given the current pixel's screen coordinates (x, y), we can retrieve its full screen space position at these three different points in time.

3.2. Grid Scatter

Given the motion data, the first pass of our algorithm requires a set of pixel-aligned vertices, at the same $M \times N$ resolution of the final image, with positions $p_{x,y}$ where $1 \leq x \leq M, 1 \leq y \leq N$. We generate (and render) this data as a pre-generated point list on the GPU.

Bézier Curve Computation. We first fit a Bézier curve $B(s)$ at each pixel from the three positions we have at times $t = \{-1, 0, 1\}$. To do so, we perform vertex position texture fetches in the vertex shader and set the necessary fitting constraints for a Bézier curve B as

$$B(0) = p(-1) \qquad B(1/2) = p(0) \qquad B(1) = p(1)$$

to solve for the curve control points

$$B_0 = p(-1) \qquad B_2 = p(1) \qquad B_1 = p(0) - \frac{p(-1) - p(1)}{2}\,.$$

We then output these three points, as well as the vertex's (u, v) coordinates, which will be used later on. We chose $s = \{0, 1/2, 1\}$ since we uniformly sampled the time steps, but different sample locations points could also be used.

Discretization and Generation. We leverage a geometry shader during the second step of our algorithm. Using the three points outputted from the vertex shader above, we generate geometry to represent the Bézier curve as faithfully as possible. Given our performance targets, we have found that discretizing the curves into line lists balances accuracy and performance, especially since lines are efficient to compute and rasterize across a limited number of pixels (scaling with $O(N)$, versus $O(N^2)$ for a polygonal approximation, for N curves). The geometry shader generates a list of fixed length Δ lines as follows:

$$\left\{(1-s^2)B_0 + 2(1-s)sB_1 + s^2 B_2 \middle| s \in S\right\} \tag{1}$$

with S defined as

$$S = \frac{1}{\Delta - 1}\{0, 1, 2, \ldots, \Delta - 2, \Delta - 1\}. \tag{2}$$

We discard motionless pixels prior to segment generation.

Rasterization. The final step in our first pass rasterizes the line segments into an accumulation buffer. Given many line segments, the potential for significant overdraw is high and so our shading routine must remain as simple as possible. To do so, we can simply use the (u, v) coordinates stored in the first step to query the color of the original pixel and output this constant color for the line, effectively implementing the scattering operation. While this works, we can improve the visual quality of the blur by additionally weighting the sampled color according the pixel's interpolated s coordinate value. We use a simple 1D Gaussian blur kernel with $(\mu = 1/2, \sigma)$, and we also output this weight to the alpha channel.

We render every pixel with fully additive alpha blending so that the final buffer stores the sum of all scatter operations. We additionally enable depth tests but disable depth writes: this means that each line also in rendered using proper z-order tests with the other objects in the scene, correctly accounting for objects moving behind other objects, even through heavily non-linear motion. This strategy also has the benefit of reducing the number of processed pixels processed with early depth testing. Due to the high variability of the values written in the accumulation buffer, we recommend using a 32-bit floating-point buffer.

3.3. Normalization

The second pass of our algorithm is a "traditional" fullscreen post-processing pass, with the rendered scene and accumulation buffer as input. For each pixel, we wish to compute the weighted average of every line rasterized onto the pixel. Concretely, we wish to compute the color $c'_{x,y}$ of the pixel at (x,y) according to the contribution of all of the other pixels on the screen:

$$c'_{x,y} = \frac{\sum_{a=1}^{M}\sum_{b=1}^{N} w_{a,b,x,y}\; c_{a,b} + w_b\; c_{x,y}}{\sum_{a=1}^{M}\sum_{b=1}^{N} w_{a,b,x,y}\; + w_b} \tag{3}$$

where $c_{x,y}$ is the original color of the pixel at (x,y) before any blurring, $w_{a,b,x,y}$ is the weight of the contribution of pixel (a,b) to pixel (x,y), and w_b is the (constant) background weight.

The accumulated values in buffer A, generated during the previous pass, effectively stores the first term of the numerator in Equation 3 and its alpha channel α stores the first term of the denominator and so we can trivial compute Equation 3 in a pixel shader as

$$c'_{x,y} = (A_{x,y} + w_b\; c_{x,y}) / (\alpha_{x,y} + w_b)\ . \tag{4}$$

We additionally output $\alpha_{x,y}$ in the alpha channel to support transparency. Note that the explicit background contribution is required, since we discard pixels without motion: without it, all motionless pixels would render as black.

4. Results

All results were computed on a Core i7-3770K with 16GB of RAM and a GTX780. Unless noted otherwise, we render at 1920×1080 and with $\{w_b, \Delta, \sigma\} = \{5, 11, 2\}$. We compare against an optimized implementation of Guertin et al.'s efficient tile-based motion blur post-process [GMN14] using the parameters listed in the paper, as well as comparing against ground truth computed using brute-force accumulation (with temporal samples distributed according to a Halton sequence, to avoid banding). We adjusted motion magnitudes in each scene in order to produce similar blurring effects for each of the three algorithms, and we chose a linear blur sample count N for each scene that reduces noise, and eliminates it where possible. We do not apply any antialiasing.

Helicopter Scene. The first scene is the simplest, but highlights an important feature of non-linear blur: the ability to correctly motion blur spinning objects. While linear algorithms can approximate very low velocity rotations, they quickly fall apart as soon as faster motions (and/or longer exposures) are used. The helicopter's spinning blades are a simple representative example. As illustrated in Figure 4, linear algorithms are unable to represent the arcing motion of the blades and tail rotor, and instead approximate it as patches of discrete linear velocity blurs (in wildly different directions). For thin objects such as the blades, the effect is incorrect but relatively acceptable. For round objects such as rotor, a distracting "pinwheel artifact" is glaringly obvious. Compared to the reference image, we note that apart from the additional presence of the unblurred image, the shape and appearance of the blur surrounding the blade is very accurate (see Section 5), closely matching ground truth.

Teapot and Cubes Scene. The teapot scene represents a more chaotic animation, with many objects moving along different (often curved) trajectories. This scene highlights the stability of our approach, which more accurately follows all motion vectors for every object; approximating blurs per-tile, on the other hand, can lead to directionless blurs and blur effects that are difficult to visually parse. Specifically, previous approaches have a tendency to over-blur the top of the teapot, where chaotic motion is highest and thus where it is extremely likely for a few highly mobile pixels to cause an entire tile neighborhood's blur estimate to deviate. These approaches also have difficulty rendering the motion blur of the movement at the bottom, where motion is largely linear and parallel, but of a higher magnitude due to gravity. Our non-linear algorithm manages to accurately represent both scenarios, once again achieving a result that is very close to the reference, aside from the overlaid presence of the unblurred objects (see once again Section 5).

Jumping Jack Scene. The last scene illustrates the algorithm's behavior with rigged characters. Character animations are an excellent example of non-linear motion, since limbs generally perform rotational movements rather than purely rectilinear ones. As with previous scenes, the blur's magnitude is more accurate with our approach, and it varies smoothly depending on the actual velocity of the limb at any given point. Details are better preserved and shading is closer to the reference, ground truth accumulated image.

4.1. Performance

Due to our algorithm's design, computation time tends to be higher on average versus linear techniques, but not significantly so (see Table 1). The pre-pass cost is easy to quantify: it is roughly double the cost of the linear algorithm's pre-pass, as it requires two buffers instead of one. The post-process cost is more complex, since it depends on the scene, including the area of the screen which is blurred as well as

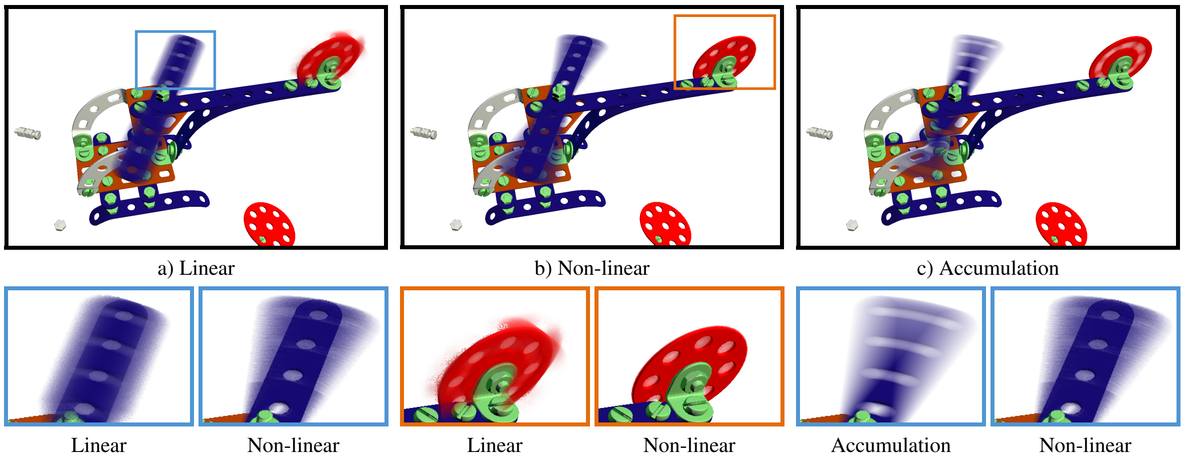

Figure 4: *The helicopter scene has significant rotational motion, a common failure case for linear motion blur: linear algorithms fail to properly convey the scale of motion, they cause over- and under-blurring, and they either give the impression of pure linear motion (as seen on the blades) or cause a pinwheel artifact (as seen on the tail rotor). This is due to clamping to dominant linear velocity directions at different angles, across tiles.*

the magnitude of the blur. Even so, it appears that a good experimental estimate is roughly double the cost of the linear algorithm's post-process. As such, it is fair to say that our non-linear approach is roughly twice as expensive as the state of the art linear motion blur post-process. This may seem significant, but in practice many modifications could be applied to limit the impact of our post-process (see Section 5.1); even then, it is important to note that we generate higher quality results in only **4 to 6.5ms** of compute time.

5. Discussion and Limitations

While our approach generates good results in only a handful of milliseconds, especially in scenes where the state of the art fails, it still has some drawbacks which we discuss below and will address in future work (see Section 5.1).

Performance. Due to our very straightforward blurring approach treatment, we also impose some additional constraints on our input. These two properties lead to variance in the rendering cost: a scene with little to no blur can easily be cheaper than existing, linear algorithms, since our shader code is comparatively simpler; however, a scene with large amounts of blur can cause significant overdraw and reduce performance. On average our approach has modest performance characteristics requiring between 4 and 6.5ms of compute time, but this is still 50 to 100% slower than the state of the art. More complex scenes are penalized more by the requirement of our second motion pre-pass.

Magnitude Constraints. Unlike existing techniques, our algorithm does not impose any constraint on the magnitude of the motion (or the exposure time), however extremely large and high-frequency motions are unlikely to be captured accurately: any lack of motion information between our sampled time steps, as well as the the inherent limitations of simple quadratic Bézier curve fits, allows our approach to handle motions roughly a few times the magnitude of current algorithms, which is still a significant improvement but is not completely general.

Masked Information. The post-processing nature of our approach adopts the same limitations as existing techniques: due to the limited scene information available per-pixel, moving objects occluded by other objects (whether stationary or not) will not produce any motion blur, which can cause vanishing motion trails when objects move in of out of view.

Depth ordering. Our algorithm processes all pixels simultaneously and without any constraint as to depth ordering. Since we use purely additive blending, this does not affect the final result, but a more accurate blending would require proper depth ordering. It is possible that improvements such as rasterizer ordered views [**?**] may allow fast and accurate sorting, therefore presenting more opportunities for accurate blending.

	Linear			Non-linear	
	Pre-pass	Post-process	Samples	Pre-pass	Post-process
Heli.	0.45	1.4	35	0.88	3.1
Teapot	0.25	5.2	61	0.51	5.8
J. Jack	0.076	2.0	35	0.15	3.8

Table 1: *Performance comparisons (in milliseconds) at* 1920 × 1080 *for the scenes in Figures 4, 5, and 7. Sample counts are provided for the linear algorithm. As expected, our pre-pass is almost exactly twice as costly since it requires two motion vector passes. Our approach is clearly slower, however the difference is not overwhelming and we have not yet made any optimization attempts.*

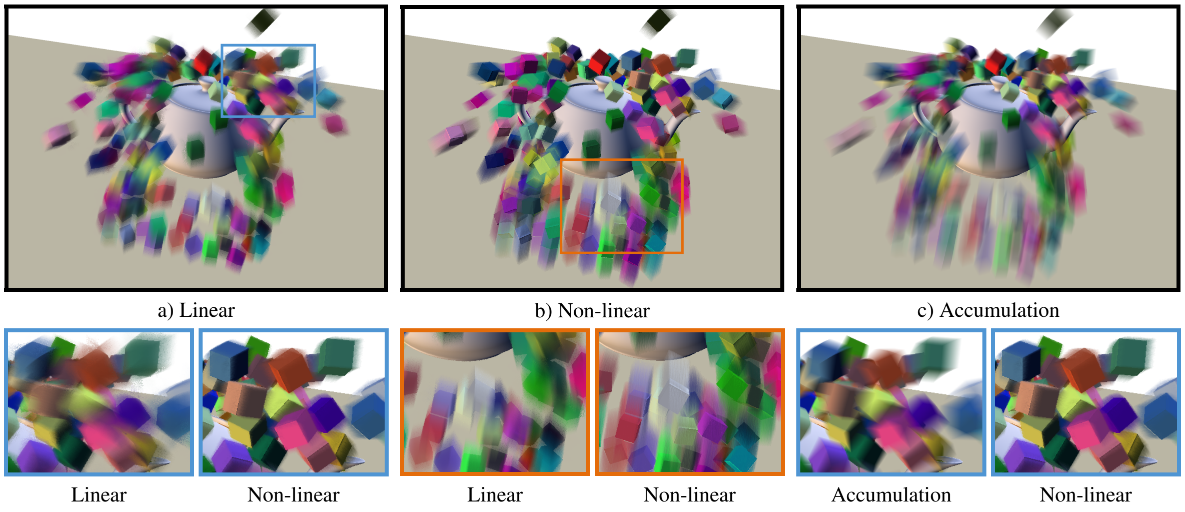

Figure 5: *This teapot scene showcases chaotic movement with many superimposed velocities on each pixel of the screen. We note that most pixel motion is slightly curved, which is more faithfully represented with our non-linear approach. Moreover, there is a high variability in the motion vectors, which can is only partially captured by linear tile-based algorithms, while our algorithm properly accounts for all directions and all magnitudes of motion.*

5.1. Extensions

We will outline the avenues of future work we will pursue in order to address some of the limitations listed above.

Cubic Bézier Curves and Circular Arcs. We can alleviate some of the limitations caused by the use of quadratic Bézier curves by moving to higher-order Bézier curves: a cubic curve would require a third motion pre-pass (i.e., at either $t = -2$ or $t = -1/2$; see Section 3), but would produce a much more accurate blur, especially for larger movements. Neither quadratic nor cubic curves would support conic motions however, and so a third option would be to instead fit circular arcs to the motions. Transitioning between these different representations is an interesting idea we are exploring as well. This is of particular interest in scenes with fast spinning objects.

Multi-pass Rendering. In order to address artifacts that result from a lack of background information (which is a limitation inherent to post-process approaches), we require some important changes to the rendering pipeline. One possible trade-off would be to segment and render the scene twice, storing two color and depth buffers: one for objects in motion, and one for static objects. In this case, camera motion would have to be handled separately. Now, we can apply our approach to the moving objects, whereby they exclusively access information from "moving" color/depth buffers *except* when computing the background color contribution, where they use the "static" buffer's colors. This significantly improves the appearance of the blur with only a moderate additional overhead cost, effectively removing the most prevalent visual artifact of our approach (see Figure 6). An open problem in this extension is how to correctly handle very low velocities, where the background weight must tend toward zero.

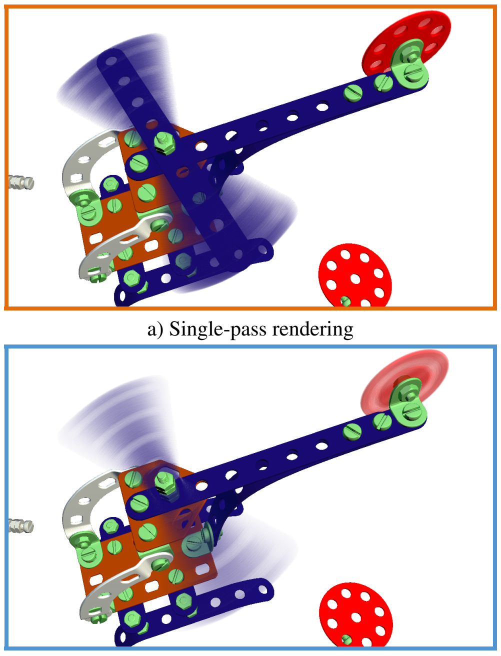

Figure 6: *Comparison between non-linear blur using the basic, single-pass algorithm (a) and using our modified two-pass rendering (b). Segmenting static objects allows us access the background color behind the blurred objects, resulting in a more accurate relative weighting between the foreground and background,* ***completely eliminating*** *the most distracting visual artifact of our current approach.*

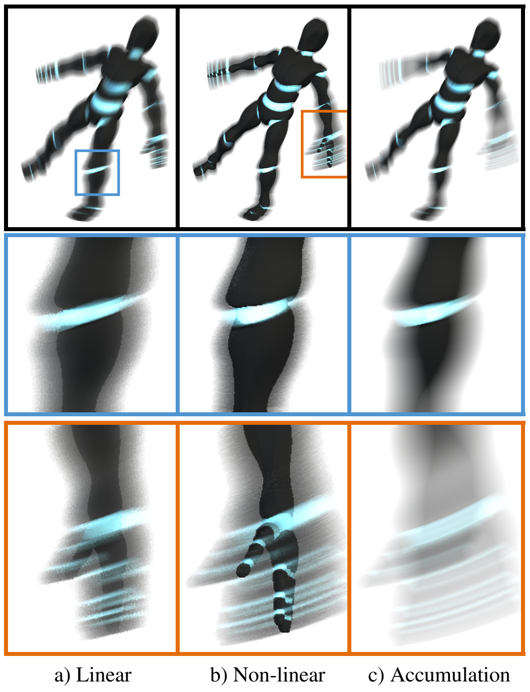

Figure 7: *The jumping jack scene illustrates the algorithms' behavior with animated characters. our approach better conserves the blur on the character's knee, whereas linear algorithms can almost completely miss this effect.*

Optimizations. Our current algorithm is designed with clarity in mind, however several avenues for optimization are available: firstly, it should be possible to determine the magnitude of the movement and clamp generated curves' lengths according to the maximum; secondly, small motions can be handled using a cheaper approximation, with our approach toggled on a *per-pixel* basis to handle complex motion; lastly, rendering our blur at a reduced resolution and using e.g., a bilateral upsampling technique [SGNS07] is an obvious direction to explore.

6. Conclusion

We propose a new approximate motion blur post-processing approach capable of more accurately capturing blur effects caused by non-linear motion. Our approach reduces spatial and temporal noise (see the supplemental videos) and is designed to leverage the shader pipeline in order to implement a true scatter operation. As such, we are able to handle longer motion trails, our algorithm is simple to implement atop existing rendering engines, it scales independently of the underlying geometric complexity, and it generates spatially- and temporally-smooth results in scenes where the state of the art fails. We discuss the limitations imposed by our approach, and propose several avenues for future work. We have begun exploring these directions, and a preliminary result (Figure 6) shows significant promise, eliminating the most significant visual artifact of our approach.

References

[AMMH07] AKENINE-MÖLLER T., MUNKBERG J., HASSELGREN J.: Stochastic rasterization using time-continuous triangles. In *Graphics Hardware* (2007), Eurographics, pp. 7–16. 2

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

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

[GMN14] GUERTIN J.-P., MCGUIRE M., NOWROUZEZAHRAI D.: A fast and stable feature-aware motion blur filter. In *High Performance Graphics* (June 2014), ACM/Eurographics. 2, 4

[HJW*08] HACHISUKA T., JAROSZ W., WEISTROFFER R. P., DALE K., HUMPHREYS G., ZWICKER M., JENSEN H. W.: Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27*, 3 (2008). 2

[KS11] KASYAN N., SCHULZ N.: Secrets of cryengine 3 graphics technology. In *SIGGRAPH Talks*. ACM, 2011. 2

[LAC*11] LEHTINEN J., AILA T., CHEN J., LAINE S., DURAND F.: Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30*, 4 (2011), 55. 2

[Len10] LENGYEL E.: Motion blur and the velocity-depth-gradient buffer. In *Game Engine Gems*, Lengyel E., (Ed.). Jones & Bartlett Publishers, March 2010. 2

[McM] MCMULLEN M.: Direct3D New Rendering Features. 5

[MESL10] MCGUIRE M., ENDERTON E., SHIRLEY P., LUEBKE D. P.: Real-time stochastic rasterization on conventional GPU architectures. In *High Performance Graphics* (2010). 2

[MHBO12] MCGUIRE M., HENNESSY P., BUKOWSKI M., OSMAN B.: A reconstruction filter for plausible motion blur. In *I3D* (2012), pp. 135–142. 2

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

[NSG11] NAVARRO F., SERÓN F. J., GUTIERREZ D.: Motion blur rendering: State of the art. *Computer Graphics Forum 30*, 1 (2011), 3–26. 2

[ODR09] OVERBECK R. S., DONNER C., RAMAMOORTHI R.: Adaptive wavelet rendering. *ACM Trans. Graph. 28*, 5 (2009). 2

[Ree83] REEVES W. T.: Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graph. 2*, 2 (Apr. 1983), 91–108. 2

[RMM10] RITCHIE M., MODERN G., MITCHELL K.: Split second motion blur. In *SIGGRAPH Talks* (NY, 2010), ACM. 2

[SGNS07] SLOAN P.-P., GOVINDARAJU N. K., NOWROUZEZAHRAI D., SNYDER J.: Image-based proxy accumulation for real-time soft global illumination. In *Proceedings of Pacific Graphics* (USA, 2007), IEEE, pp. 97–105. 7

[Sou11] SOUSA T.: Cryengine 3 rendering techniques. In *Microsoft Game Technology Conference*. August 2011. 2

[Sou13] SOUSA T.: Graphics gems from cryengine 3. In *ACM SIGGRAPH Course Notes* (2013). 2

[TBI03] TATARCHUK N., BRENNAN C., ISIDORO J. R.: Motion blur using geometry and shading distortion. In *ShaderX2: Shader Prog. Tips & Tricks with DirectX 9.0*, Engel W., (Ed.). 2003. 2

[ZG12] ZIOMA R., GREEN S.: Mastering DirectX 11 with Unity, March 2012. Presentation at GDC 2012. 2