Image-Based Motion Blur for Stop Motion Animation

Gabriel J. Brostow   Irfan Essa

GVU Center / College of Computing
Georgia Institute of Technology
http://www.cc.gatech.edu/cpl/projects/blur/

Abstract

Stop motion animation is a well-established technique where still pictures of static scenes are taken and then played at film speeds to show motion. A major limitation of this method appears when fast motions are desired; most motion appears to have sharp edges and there is no visible motion blur. Appearance of motion blur is a strong perceptual cue, which is automatically present in live-action films, and synthetically generated in animated sequences. In this paper, we present an approach for automatically simulating motion blur. Ours is wholly a post-process, and uses image sequences, both stop motion or raw video, as input. First we track the frame-to-frame motion of the objects within the image plane. We then integrate the scene's appearance as it changed over a period of time. This period of time corresponds to shutter speed in live-action filming, and gives us interactive control over the extent of the induced blur. We demonstrate a simple implementation of our approach as it applies to footage of different motions and to scenes of varying complexity. Our photorealistic renderings of these input sequences approximate the effect of capturing moving objects on film that is exposed for finite periods of time.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.4.9 [Image Processing and Computer Vision]: Scene Analysis—Time-varying Images

Keywords: Animation, computer vision, image-based rendering, motion blur, stop motion animation, temporal antialiasing, video post-processing.

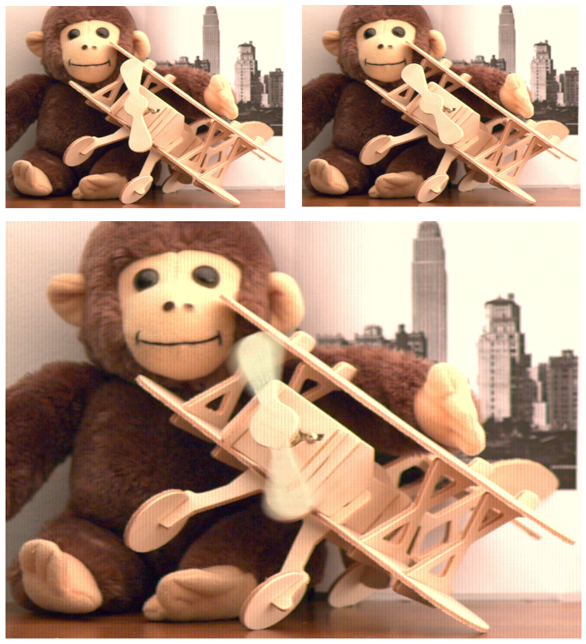

Figure 1: The miniKingKong stop motion sequence was shot by manually rotating the propeller. The top two images are original frames and the bottom image shows a blurred propeller as rendered automatically from that input.

1 Introduction

Stop motion animation is a widely respected and traditional process for generating movies. *Chicken Run*, *Wallace and Gromit*, and *Nightmare Before Christmas* are some of the recent feature works following from earlier productions like *King Kong*. Stop motion is also widely used to generate dynamic (and controlled) effects in regular productions [17]. The process of stop-motion animation is laborious, requiring that each scene be photographed after being moved and modified incrementally [12]. Talented animators show very elaborate and detailed motions of characters and objects with varying speeds and trajectories. A tool unavailable to these animators is one that would aid in the generation of motion blur between static frames to show fast motions. We address this problem with a technique that functions as a post-process, requiring no extra work on the part of the animator.

The presence of motion blur is very important perceptually. Like many artifacts of photo imagery, one usually only notices its absence; it's presence gives an air of realism [9, 13]. In addition, without motion blur, animated image sequences are more susceptible to strobing effects and aliasing in time. Motion blur is a well-studied topic in computer graphics, with commercially available software and hardware solutions that aid in the synthesis of realistic motion.

Stop-motion animators toil to prevent audiences from being distracted by the lack of motion blur by changing the pose of their characters between frames only slightly. In addition, several tricks that involve deforming objects and repeated film exposures are also employed. Further, a mechanical technique called "go-motion" was developed at Industrial Light and Magic and was first used in the 1981 film *Dragonslayer* [17]. This technique uses computer-controlled armatures to effectively enhance the motion of stop motion animated creatures.

1.1 Related Work

Film exposed to live action implicitly contains motion blur, which guarantees that fast objects will be registered differently than slow moving ones. At present, only research in computer animation has addressed the problem of creating photorealistic motion blurred images. Antialiasing of time-sampled 3D motion is now a standard part of most rendering pipelines.

The seminal work in motion blur was first introduced in 1983 by Korein and Badler [11], and Potmesil and Chakravarty [14]. Korein and Badler introduced two approaches, both modeled after traditional cartoonists' work. The first implementation parameterized the motion over time of 2D primitives such as discs. The second relied on supersampling the moving image and then filtering the resulting intensity function to generate images with superimposed multiple renderings of the action. The resulting images look like multiply exposed film.

Potmesil and Chakravarty proposed a different method for blurring motion continuously. A general-purpose camera model accumulates (in time) the image-plane sample points of a moving object to form a path. Each sample's color values are then convolved with that path to generate finite-time exposures that are photorealistic.

The most significant subsequent development in antialiasing in the time domain came with the distributed raytracing work of Cook *et al.* [6]. This work successfully combines image-space antialiasing with a modified supersampling algorithm that retrieves pixel values from randomly sampled points in time.

These algorithms cannot be applied directly to raster images of stop motion animation because the transformation information about the scene is absent. Both convolution with a path and supersampling that path in time require that the motion be fully specified while the "shutter" is open. Raster images of stop motion are effectively snapshots, which in terms of motion, were shot with an infinitely fast shutter. While only the animator knows the true paths of the objects, visual inspection of the scene allows us to infer approximations of the intended motion paths. Cucka [7] managed to estimate localized motions when developing a system to add motion blur to hand-drawn animations. Much like the commercial blur-generation and video-retiming packages currently available, only small frame-to-frame motions are tracked successfully, and pre-segmentation must be done either by hand or as part of the animation process.

Our approach is entirely image-based. Animators can adjust the extent of the generated blur to their liking after the images are acquired. We take advantage of well-established computer vision techniques to segment and track large motions throughout a sequence. Our proposed techniques rely on combinations of background subtraction and template matching in addition to the previously explored capabilities of optical flow and visual motion computations. Though existing techniques for encoding motions are becoming quite refined (*e.g.,* MPEG-4 [10]), such methods are not sufficient for our needs.

1.2 Overview

We are interested in changing the regions of an image sequence which correspond to stop motion activity to show motion blur. Our goal is to give the user control over the amount of blur, and to perform the rest of the process as automatically as possible. One major task is the detection and extraction of regions that are affected by movement in consecutive frames.

To help determine the correspondence between pixels in two consecutive frames, we initially group neighboring pixels within each image into blobs, and these are assumed to move independently (Section 2). The affine transformations of these blobs are tracked and recorded as pixel-specific motion vectors. These vectors may subsequently be refined individually to model other types of motion. Once the motion vectors adequately explain how the pixels from a previous image move to their new positions in the current image, each pixel is blurred (Section 3). The pixel color values are redistributed according to the distance traveled, so that pixels with null vectors are unchanged while pixels with long motion vectors are convolved (or smeared) in accordance with that path. Finally, we demonstrate the various results rendered by our experimental implementation (Section 4). Figure 2 illustrates our pipeline graphically.

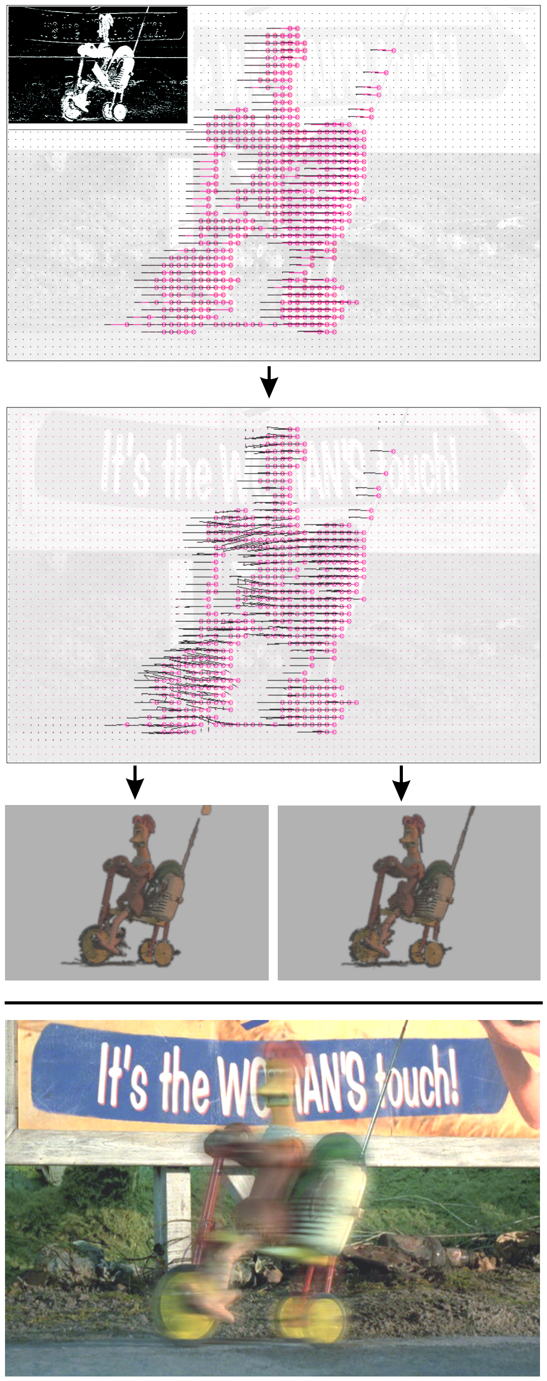

Figure 2: Using frames from the film Chicken Run, and starting with the inset of the top image, each level represents a stage of our technique: segmentation, rigid blob tracking, flow correction, and accumulation of smeared samples. Combined, they yield the blurred image of Rocky rolling backwards on the tricycle.

2 Finding Pixel Transformations

Ideally, we desire the same 3D transformation data as is available to mesh-based animation renderers. Some structure-from-motion and scene reconstruction techniques, studied by computer vision researchers, are capable of extracting such pose information from certain classes of image sequences [8]. They usually seek optimal scene geometry assuming that a single motion, like that of the camera, caused all the changes in the scene. Footage containing articulated characters is troublesome because multiple motions contribute to the changes. Ours is consequently a rather brute-force but general purpose 2D motion estimation method. The resulting system is but a variant of the possible motion estimators that could satisfy the needs of our approach. The following description explains how to handle objects that are traveling large distances between subsequent frames.

2.1 Scene Segmentation

The task of segmenting and grouping pixels that are tracked is simplified by the high quality of the footage captured for most stop motion animations. Additionally, scenes shot with a moving camera tend to be the exception, so background-subtraction is a natural choice for segmenting the action.

If a clean background plate is not available, median filtering in the time domain can usually generate one. We observe a pixel location over the entire sequence, sorting the intensity values (as many as there are frames). By choosing the median, the background can be reconstituted one pixel at a time. This highly parallelizable process results in choosing the colors which were most frequently sampled by a given pixel, or at least colors that were closest to doing so. Given a reasonably good image of the background (I_b), the pixels that are different in a given frame (I_f) are isolated. An image (I_m) containing only pixels that are moving is obtained according to this criterion:

$$I_m(x,y) = \begin{cases} I_f(x,y) & |I_f(x,y) - I_b(x,y)| > \text{threshold} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

A good threshold value, that worked on all sequences we processed, was 7.8% of the intensity scale's range. Note that we use grayscale versions of our color difference images when comparing against this threshold. Each contiguous region of pixels within I_m is grouped as a single blob (b), with the intent of locating it again in the subsequent frame. Naturally, blobs representing separate objects can merge and separate over time, depending on their proximity. While this has not been a problem for us, it can be dealt with by using color-similarity and contour-completion as supplementary means of chopping the scene into manageable blobs. These future extensions would also help in dealing with footage shot with a moving camera.

2.2 Blob Tracking

Objects, and therefore the blobs that represent them, can move significantly between frame i and frame $i+1$. To determine the paths along which blur will eventually take place, we must first find a correspondence which maps each blob to its appearance in successive frames. This motion estimation is a long-standing vision research problem. Large regions of solid color and non-uniform scene-lighting compound the violation of the optical flow constraint equation. Hierarchical techniques, some of which include rigidity constraints, have been developed to make this problem more tractable. Bergen *et al.* [1] developed a hierarchical framework which unifies several parameter-based optical flow methods. These help to constrain the motion of the trouble spots, though a more recent work [16] shows more promise for the large motions we must handle. Tweed and Calway perform block-based translational motion estimation followed by a partial-correlation pass which refines the segmentation blocks to meaningful shapes. While rotations and translations greater than half a block-size are not yet handled, occlusion information is handled explicitly.

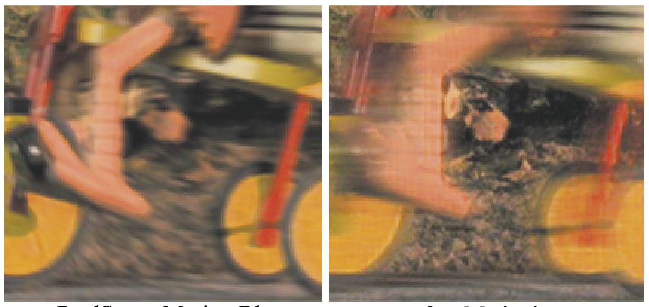

Figure 3: The same sequence was processed using both a commercial motion blur package and our technique. The close-up on the left shows that the interframe motion was too large because the hierarchical flow vectors have affected the parts of the background visible between the moving wheels. Our results on the right give a stronger indication that movement is occurring from left to right.

To deal with footage containing both substantial translation and rotation, our two-pass approach starts with a non-local search before verifying with proximity dependent techniques. For this purpose, we perform a rather exhaustive search by evaluating the similarity between each blob b in $I_m(i)$ with the blobs in $I_m(i+1)$, referred to here as the sets $\mathbf{B}(i)$ and $\mathbf{B}(i+1)$ respectively. A fully exhaustive search would entail rotating various scaled versions of a blob b by small angular increments, and testing it's correlation with $\mathbf{B}(i+1)$, centered on each possible (x,y) location. Eventually, one would expect to find the rotation, translation, and scale parameters which would best transform all the blobs in $\mathbf{B}(i)$ to look like blobs in $\mathbf{B}(i+1)$.

Here, we make two assumptions about our footage that do not compromise our goals, have minimal impact on our results, and allow for a more efficient parameter search. These assumptions are: (a) scale does not significantly affect the appearance of an object in consecutive frames, and (b) the extent of rotation can be determined by evaluating absolute orientations of all blobs. The orientation (θ) is found for each blob using its first and second normalized central moments $u_{(x,y)}$, which essentially measure the asymmetry about the x and y axes.

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2u_{(1,1)}}{u_{(2,0)} - u_{(0,2)}}\right), \tag{2}$$

where each blob's different moments $u_{(m,n)}$, are calculated as:

$$u_{(m,n)} = \frac{\sum_{j=1}^{b_h}\sum_{k=1}^{b_w}(x_k - x_0)^m (y_j - y_0)^n I_{m(k,j)}}{{b_a}^2}, \tag{3}$$

where b_w, b_h are the width and height of the blob and b_a is its area measured in pixels.

Taking each $b(i)$ in turn, we generate as many rotated versions of it as there are elements in $b(i+1)$, plus an additional one representing the possibility that no rotation is occurring. Each version of $b(i)$ is template-matched against $I_m(i+1)$. This way, each rendition of $b(i)$ is found to have a translational offset which maximizes the normalized cross correlation (NCC). The normalized correlation between a blob's pixel values, $b(x, y)$, and an image, $I(x, y)$, is calculated at each coordinate as:

$$\mathrm{NCC} = \frac{\sum_{j=1}^{b_h}\sum_{k=1}^{b_w} b(k,j)\, I(x+k, y+j)}{\sqrt{\sum_{j=1}^{b_h}\sum_{k=1}^{b_w} b(k,j)^2 \left(\sum_{j=1}^{b_h}\sum_{k=1}^{b_w} I(x+k, y+j)^2\right)}}. \quad (4)$$

The version of $b(i)$ which achieves the highest correlation score (usually above 0.9) is recorded as the best estimate of a blob's motion between frames i and $i + 1$. It is interesting to note that with objects entering and leaving the scene, the best translation and rotation parameters often push a blob beyond the image boundaries.

2.3 Flow Correction

The current blob tracking results represent the best estimate of the rotations and translations performed by the objects in the scene. By applying the respective 2D transformations to each blob's constituent pixel coordinates, a vector map $\mathbf{V}_i$, representing the movement of each pixel from its original location is obtained. A current estimate of $I_f(i+1)$, called $I_r(i+1)$, is regenerated by applying the $\mathbf{V}_i$ vectors to $I_f(i)$. Ideally, $I_r(i + 1)$ would look like $I_f(i + 1)$ if all the motions were purely due to translation and rotation. In practice, other transformations and lighting changes combine with the effects of perspective projection to reveal that we must locally refine the motion estimates. Since there is a large selection of algorithms for computing visual motion, we tried several different ones and are now using a slightly modified version of the method proposed by Black and Anandan [2]. This method computes hierarchical optical flow to find the motion vectors which best warped $I_r(i + 1)$ to look like $I_f(i + 1)$. The benefits of other local-motion estimation algorithms can be significant, but each has a breaking point that can be reached when the frame-to-frame deformation or lighting changes are too large.

Estimating optical flow allows for the creation of a map of corrective motion vectors. The map is combined with the initial motion vectors in $\mathbf{V}_i$. $I_r(i+1)$ images regenerated using the corrected $\mathbf{V}_i$'s now have fewer visible discrepancies when compared against their intended appearance (see Figure 2).

3 Rendering Blur

The frame-to-frame transformations intended by the animator have now been approximated. We proceed to interpolate paths along which pixels will be blurred.

A $\mathbf{V}_i$ map tells us where to move the pixels sampled at time t_i (timestamp of frame i) to make them look like $I_f(t_i + 1)$, the image sampled at frame $i + 1$. To render the simulated object motion, which is supposed to be occurring between these discrete time samples, the intermediate pixel motion must be interpolated. For simplicity, we started with linear interpolation, though B-splines are more appropriate for arching motions. We model the path followed by a pixel over time, or its locus, as a function $L_i(x, y, t)$. The function reports the interpolated offsets of a pixel that was at (x, y) at time t_i (see Figure 4). The locus is a parametric function of time with units in pixels, valid over the period from $t = t_i$ to $t = t_{i+1}$. We assume that a pixel's locus most faithfully represents the true motion of a pixel at the times immediately surrounding t_i. Therefore, motion blur is generated based on the events immediately before and after each picture was shot, rather than sampling $L(t)$ at some other phase or frequency.

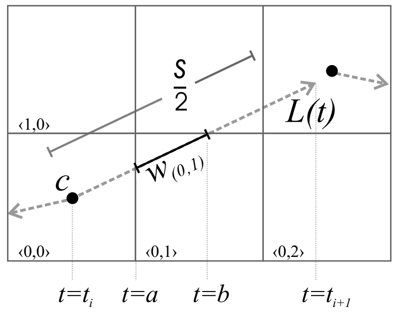

Figure 4: Color pixel c moved according to the dotted path L. After choosing a desired shutter speed s, c's RGB values are redistributed according to the time it spent moving through each pixel.

3.1 Smearing Pixels

We now tie together the extracted information to actually simulate film that was exposed to our scene's motions for finite periods of time. The animator who posed the scene, shot it at a period of τ seconds between samples, corresponding to the inverse of the intended playback frame-rate. The real camera's actual shutter speed only affected intensity. Thus, we refer here to shutter speed s as the time during which our virtual camera is exposed to the interpolated pixel motions. For example, for a playback τ of 1/24 sec., a typical s might be 1/58 sec., but will rarely exceed τ.

To render a simulated exposure time, non-zero motion dictated by each $L_i(x, y, t)$ is integrated for a total of s seconds: half before and half after t_i. Let us first discuss the blurring of motion occurring after time t_i, noting that we process one pixel at a time, independently of the rest. This will result in a blurred image we call $I_{\mathrm{Aft}}(t_i)$. It is worth noting that Cabral and Leedom [5] demonstrated a variant of their Line Integral Convolution which used the vector field magnitude to vary the length of the line integral to render blur. However, the approach below is simpler and does not suffer from singularities when adjacent streamline cells point at each other.

First, the valid interval of $L_i(x, y, t)$ is plotted on a blank grid of pixels, approximated for efficiency as square shaped (see Figure 4). The relevant interval is from t_i to $t_i + s/2$, and the pixel in question can be represented by color c =R, G, B. We found it impractical to actually create a box filter in the shape of L and convolve c with it. To preserve the original intensity of each image, we distribute c along the interval of L, proportionately with the fraction of time that pixel spent occupying each grid-square. This fraction of the path ($w_{(x,y)}$) which fell within a given destination pixel's boundaries $[a, b]$ for a time period of $a \le t \le b$, is calculated as follows:

$$w_{(x,y)} = \frac{\int_a^b \left(\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}\right) dt}{s/2}, \quad (5)$$

where both x and y can be expressed as parametric functions of t. A form of stylized motion blur is also possible. Instead of weighting all the points on the locus equally, one can bias the weighting to emphasize the beginning or end of a motion, making the individual streaks look more like comets.

The color values of each destination pixel at a given (x, y) location are incremented by $cw_{(x,y)}$. This way, each destination pixel accumulates a smeared (dimmer but same color) version of all the moving pixels that passed through it. Also, an accumulator-image keeps count of the total $w_{(x,y)}$ deposited at each location.

The same procedure just followed in rendering $I_{\mathrm{Aft}}(t_i)$, the motion of $I_f(t_i)$ from t_i to $t_i + s/2$, can now be repeated to generate

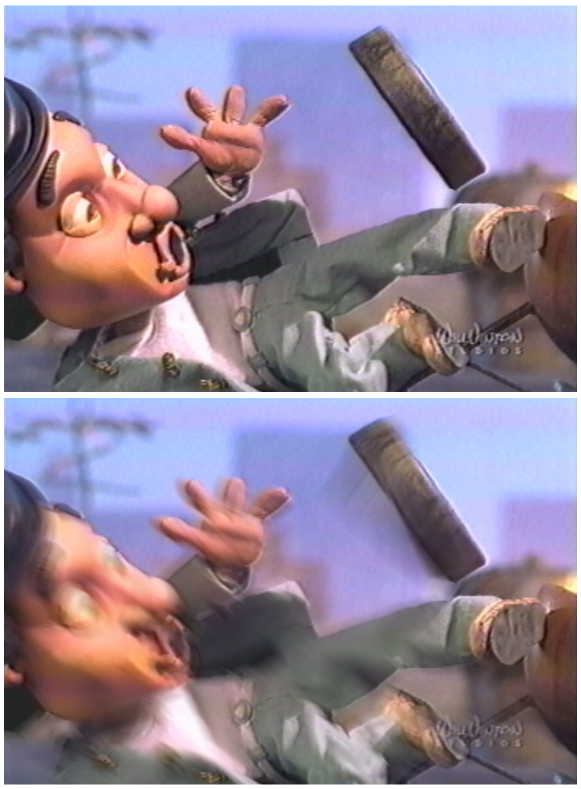

Figure 5: Jackie Chan in an episode of the PJs is shown jump-kicking. The bottom image was rendered with a simulated shutter speed of 0.025 seconds.

$I_{\text{Bef}}(t_i)$: the motion of that image from t_i to $t_i - s/2$. In contrast to the I_{Aft} case, the L used here comes from traveling along the $L_{i-1}(t)$ obtained from estimating the motion of the image sequence in reverse.

3.2 Compositing Pixel Motion

The $I_{\text{Bef}}(t_i)$ and $I_{\text{Aft}}(t_i)$ can be regarded as two pieces of exposed film, each having witnessed half of the motion occurring during s. Since each image was normalized with respect to c, merging the two means performing a pixel-wise average. Adding the two occupancy maps together reveals that many pixels have been exposed for less than s seconds.

Pixels that exhibited no motion have zero occupancy, and other pixels may have been visited by motion only briefly. The remaining time must be accounted for by blending in a proportional amount, $(s - w_{(x,y)})/s$, of a static pixel located at that (x, y). For pixels where $I_m(x, y)$ is 0, that pixel is taken from I_b, and it is taken from I_f otherwise.

4 Discussion and Results

To test our algorithm, we shot stop motion footage of our own and digitized sequences made by professional studios. We tested clips from *Chicken Run*, The *PJs*, and *The Life & Adventures of Santa Claus*. Figure 5 shows a representative input and output pair. Note that the previous and subsequent frames of the input sequence were also used to make each blurred image.

After the motion has been estimated once, the rendered motion blurred image sequence looks as if the objects were moving while the shutter was open. The user-controlled virtual shutter speed s allows for the same motion to be rendered with more or less blur, as seen in Figure 6.

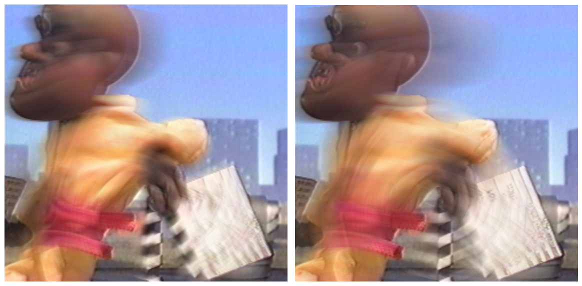

Figure 6: The PJs character ”The Clapper” is twirling his weapons while running forward. Two shutter speeds, 0.025 and 0.058 seconds, were chosen to render the left and right images respectively.

Our own footage proved to be more challenging because of the overall lower quality, including bad lighting and large inter-frame motions. One sequence included clay and a stick rotating like a pendulum. The large motions are detected and the pendulum appears blurred in Figure 7, in accordance with the linear interpolation of its motion. The results break down if s is long enough to integrate the pixels' locus functions when they poorly model the object's actual motion. The pendulum contains motion that would be better served by interpolating the path to fit a B-spline, though this too is not a general solution because other large motions might be better modeled currently, with the linear interpolation.

The matter of shadows cast by moving objects deserves special attention, and is still not fully resolved. Shadows incorrectly appear to our system as moving objects, and get blurred according to the same rules (see Figure 8). The texture under a moving shadow is not moving, and should therefore be left unfiltered. Shadows of fast-moving objects should be broader and brighter because they are purportedly the result of light being obscured during only part of the exposure. Unfortunately, generating an accurate shadow with an appropriately larger footprint requires knowing both the 3D geometry of the shadowed region, and the locations of the light sources.

In contrast to CG-blurring, occlusions are not as critical in this task because we are dealing with only the visible surfaces of our scene. Nevertheless, we must be cautious not to smear the moving pixels onto foreground objects that occlude the action. Such a foreground mask can be painted by hand for each scene, but we have successfully used a simplified implementation of [4] to accomplish the same task automatically.

5 Summary and Future Work

The particular benefit of post-processing stop motion footage into motion blurred renderings is the smoothness of fast motions, allowing for better integration with live-action footage. Without resort-

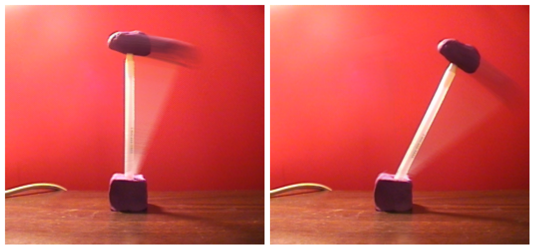

Figure 7: The extended shutter speed of these blurred images reveals that this pendulum motion is being interpolated linearly.

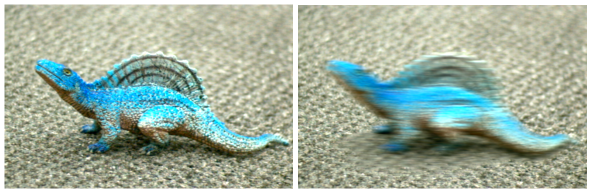

Figure 8: Carpet appears to move when shadows pass over it.

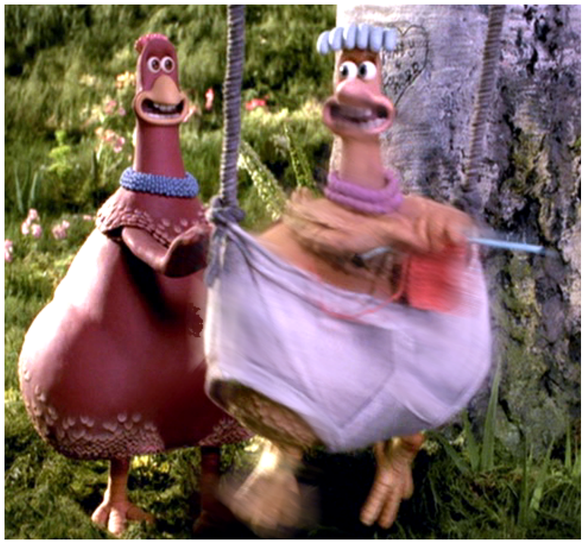

Figure 9: The shutter speed of 0.025 seconds reveals in one still that Bunty is pushing Babs on a swing in Chicken Run.

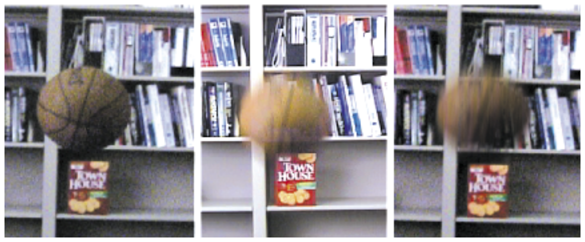

Figure 10: A dropping basketball was filmed with two cameras. The left image was shot with a shutter speed of 0.001 seconds, while the center image is from a camera with a 0.017 seconds shutter. We processed the crisper sequence to render the image on the right, simulating the the longer exposure. As expected, the blur has been mimicked and the original lighting is retained.

ing to multiple-exposures or additional posing of the scene, even individual frames can convey the motion that the animator intended (see Figure 9). This approach can actually reduce the labor involved in animating certain scenes. Potentially, fast motions, which have normally been posed at higher frequencies to maintain visual continuity, will require no more posing than slowly moving objects.

While our approach emulates real motion blur successfully (see Figure 10), certain areas deserve further attention so motion blurring can become a general-purpose post-processing tool. It will be interesting to incorporate other motion estimation algorithms, possibly stereo and 3D, to help with accurate frame-to-frame tracking of pixels. The added benefit of range data will be the increased precision in separating out and regenerating shadows.

Other curve-based interpolation algorithms could be evaluated as models for pixel locus. We expect that at least in most cases, even linear motions will yield reasonable results when modeled as curves. A good motion blurring system might do well to have user control of both shutter speed and interpolation type.

Finally, as motion estimation improves, a user-interface could be added to allow animators to specify the elapsed time between snapshots. These would correspond to keyframes that could subsequently be rendered out at the desired frame-rate and shutter-speed.

6 Acknowledgments

The full effect of our algorithm is only visible when tested on real stop motion footage. We are very grateful to Lyndon Barrois, Will Vinton Studios, Aardman Animations, and Dreamworks SKG for providing permissions to use their movie clips. The copyrights of these sequences are held exclusively by the respective companies: PJs Episode "The Last Affirmative Action Hero!"©2000 Will Vinton Studios; Chicken Run©2000 DreamWorks, Pathe and Aardman. We thank RE:Vision Effects for their demo version of Motion Blur v1.8.2. We also wish to thank Jerome Solomon, Christina de Juan, and Elizabeth Greene for their help at different stages.

References

[1] J. R. Bergen, P. .J. Burt, R. Hingorani, and S. Peleg, Computing Two Motions from Three Frames. *In Proceedings of International Conference on Computer Vision 1990*, pages 27–32, 1990.

[2] M. J. Black, P. Anandan, The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, CVIU, 63(1), pp. 75-104, Jan. 1996.

[3] G. Bradksi and V. Pisarevsky. Intel's computer vision library: Applications in calibration, stereo, segmentation, tracking, gesture, face, and object recognition. In *In Proc. of IEEE Computer Vision and Pattern Recognition Conference 2000*, volume II, pages II:796–797, 2000. Demonstration Paper.

[4] G. J. Brostow, I. Essa. Motion Based Decompositing of Video. In *Proc. of International Conference on Computer Vision*, pages 8-13, September 1999.

[5] B. Cabral and L. Leedom. Imaging Vector Fields Using Line Integral Convolution, *Proc. of ACM SIGGRAPH 1993*, pages 263–270, 1993.

[6] R. L. Cook, T. Porter, L. Carpenter. Distributed Ray Tracing. In *Proc. of ACM SIGGRAPH'84*, pages 137-145, July 1984.

[7] P. Cucka. Postprocess 2D Motion Blur for Cel Animation. In ACM SIGGRAPH 1999 Conference Abstracts and Applications, Technical Sketch, page 265, August 1999.

[8] O. Faugeras. *Three-Dimensional Computer Vision : A Geometric Viewpoint* MIT Press, November 1993.

[9] T. Grimm, J. Burchfield, M. Grimm. *The Basic Darkroom Book*. Plume, 3rd Edition, August 1999.

[10] H. Kalva. *Delivering MPEG-4 Based Audio-Visual Services*. Kluwer Academic 2000.

[11] J. D. Korein, N. I. Badler. Temporal anti-aliasing in computer generated animation. In *Proc. of ACM SIGGRAPH'83*, pages 377-388, July 1983.

[12] P. Lord, B. Sibley. *Creating 3-D Animation : The Aardman Book of Filmmaking* Harry N. Abrams Inc., Publishers. October 1998.

[13] D. Morley. *The Focal Guide to Action Photography*. Focal Press, London, 1978.

[14] M. Potmesil, I. Chakravarty. Modeling motion blur in computer-generated images. In *Proc. of SIGGRAPH 1983*, pages 389-399, July 1983.

[15] T. Smith *Industrial Light and Magic: The Art of Special Effects*. New York: Ballantine Books, 1986.

[16] D. Tweed and A. Calway. Motion Segmentation Based on Integrated Region Layering and Motion Assignment. *Proc. of Asian Conference on Computer Vision*, pages 1002–1007, January 2000.

[17] M. C. Vaz, P. R. Duigan, *Industrial Light and Magic: Into the Digital Realm*. New York: Ballantine Books, 1996.