sap_0034: Pixmotor: A *Pix*el *Mo*tion Integra*tor*

Ivan Neulander
Rhythm & Hues Studios

Pixmotor is an image-based technique developed by Rhythm & Hues for adding realistic motion blur to a statically rendered image, using a *pixel motion* and a *depth* image, which are inexpensively generated by the renderer. The resulting motion blur is of sufficiently high quality to have seen extensive use in recent motion pictures, including *Garfield: A Tale of Two Kitties* and *Night at the Museum*. Unique to our approach is a rich set of heuristics, which virtually eliminate many of the artifacts that commonly plague 2D motion blur solutions.

Figure 1: **top:** A 2k frame motion-blurred with Pixmotor was generated in under a minute on a 2 GHz dual Opteron. **bottom:** Pixmotor's motion, depth, color inputs, and output.

Motivation

Our primary motivation in creating Pixmotor was to reduce rendering time and memory, but we found two additional advantages over true 3D motion blur (by which we mean physically accurate temporal integration computed within the renderer):

1) Pixmotor enables unique art direction of the motion blur. For example, we can use it to seamlessly reduce the motion blur on the face of a fast-moving character by rotosplining and darkening the corresponding part of the motion image. Also, the effective camera shutter can be trivially adjusted at any frame without rerendering.

2) Pixmotor lets us properly motion-blur images that are shaded outside the renderer, typically in our compositing software (into which the Pixmotor engine is integrated). For example, we frequently shade reflection-mapped surfaces in the compositor, using position and normal images generated by the renderer. Such "data" images cannot be motion-blurred by the renderer without combining neighboring pixel values, which leads to incorrect shading.

Implementation Overview

Pixmotor requires a statically rendered image with some additional data channels for each pixel: pixel motion and depth. The former consists of X, Y, Z components, with X and Y in NDC space and Z in camera space. The latter contains Z-values in camera space. Our renderer generates the motion image efficiently by subtracting each primitive's positions at the shutter endpoints.

Pixmotor discretizes the shutter timespan into a set of evenly spaced moments, whose number depends on the speed of the fastest moving pixel. The time integration loop follows: At each moment, pixmotor motion-interpolates each input pixel into a double-resolution *slice buffer*. The Z-components of the interpolated pixels are used to resolve visibility in the slice buffer using Z-buffering. At this point, the heuristics discussed below are applied to remove any artifacts that emerge due to pixel divergence or movement into the view frustum. When all the input pixels have been interpolated into the slice buffer, it is added to a persistent *accumulation buffer*, which ultimately contains the output image. We repeat the process for all remaining moments and apply a final heuristic.

Our multithreaded C++ implementation interpolates multiple moments in parallel, with each thread having a dedicated slice buffer. This yields a near-linear overall speedup, at the expense of increased memory use.

Limitations and Artifacts

Because Pixmotor only considers a single vector of color, motion, and depth per pixel, it cannot reliably deal with transparency. Specifically, information about any surface behind the one nearest the camera simply does not exist in Pixmotor's input, so it is ignored. Removing this limitation would require an algorithm that processes a more complex dataset, such as a per-pixel fragment list (clearly not an image-based solution). Nevertheless, we have achieved satisfactory results using Pixmotor on the majority of materials we commonly render, including those with slight transparency, such as fur.

Transparency aside, the main problem with any 2D motion blur algorithm, including Pixmotor's, is the absence of information about relevant hidden surfaces in the input images—ones that are either occluded or outside the camera's field of view. This leads to *holes*, which we define as areas of erroneously low pixel coverage (hence, reduced color and matte values) in the final output image.

The typical workflow at Rhythm & Hues is to render various elements of a scene separately and combine them in the compositor. This favors Pixmotor by eliminating occlusion between different elements, though self-occlusion and holdouts must still be dealt with.

Hole-Filling Heuristics

Before integration, the motion image can be blurred by a user-controlled amount to reduce divergent motion among pixels. To preserve accuracy, we must ignore out-of-matte pixels during the blur convolution. Using a small blur radius affects motion trajectories slightly but is effective in eliminating certain holes.

Next, a motion gradient image is computed to detect which pixels are susceptible to holes. The gradient computation assumes that pixels on the edges of the frame neighbor stationary pixels, which yields high edge gradients when objects move into frame, allowing the resulting holes to be effectively filled by the following process.

During the motion integration loop, holes are filled in the slice buffer by searching along their corresponding motion vectors for a pixel that has a solid matte (the precise value is a user setting that defaults to just under 1.0). The distance searched depends on the speed of motion. If a solid pixel is found within a suitable "leash" distance, which depends on the amount of divergence near the hole, the solid pixel is replicated into the hole.

Once the motion integration is complete, a final heuristic is optionally applied to remove any remaining fractional holes from the accumulation buffer. Essentially, non-solid pixels sufficiently distant from silhouette edges are matte-unpremultiplied to become solid. The distance threshold depends on the magnitude of pixel motion near the holes, multiplied by a user-specified scalar.