

# Spark Motion Integrator

*Compositor node*

by Olaf Razzoli

Petru Ciobanu

based on a paper of Rithm&Hues (2d vector-motion blur: Pixmotor)

Note: here you can find what we'd like to realize. What's in bracket is still in the "ought to be there" list

Input:

1. Image: RGB

2. Motion vectors image :  $[(X_t, Y_t) - (X_{t-1}, Y_{t-1}), (X_{t+1}, Y_{t+1}) - (X_t, Y_t)]$

3. Depth map : Z (0..1)

Parameters:

1.  $N_{\text{slice}}$  Number of slices: integer

2.  $(N_{\text{disp}}$  maximum number of pixels displacement among slices)

Output:

1. Motion blurred image: RGB

## Node purpose

Motion blur is the process to simulate the blurring generated by a moving object or camera during shutter span. The result is an image with a blurred trail along the object path.

The aim of 2D simulation is to drastically reduce the 3D motion blur generation time, which usually requires several full image or object rendering, using simple linear motion interpolation and heuristics to reduce visual artifacts.

## Implementation

To accomplish this the node requires a motion vector and the Z depth for each pixel.

Motion vectors are generated by the render subtracting the previous pixel position from the current pixel position coordinates.

### 1 Initialization

1.1 (Search for the maximum motion vector)

1.2 (basing on  $N_{disp}$  compute  $N_{slice}$ )

1.3 Create  $N_{slice}$  slices, twice the original image resolution

1.4 (To reduce motion divergences, the motion vector image can be blurred. To preserve correctness a small radius should be used. The kernel must ignore out of matte pixels)

1.5 (Holes in the resulting image are more frequent where the speed gradient is higher. A gradient image is computed assuming 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. )

### 2 Interpolation

2.1 For each slice

2.1.1 scan the input image, copy each pixel moving it to the new position along the motion vector. Occlusion is resolved using the Depth map. No blending is done at this stage.

2.1.2 Apply heuristics to remove artifacts

2.1.2.1 (heuristics: 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.)

2.2 when a slice is complete, copy onto the accumulation buffer. The accumulation can be modulated over a function (gaussian, box, triangular, etc.)

### 3 Final step

3.1 apply final heuristics: 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.