

Real-Time Geometric Motion Blur for a Deforming Polygonal Mesh

N. Jones and J. Keyser[†]

Department of Computer Science, Texas A&M University, USA
<http://research.cs.tamu.edu/keyser/graphics>

Abstract

We present a method for adding a geometric motion blur to a deforming polygonal mesh. We keep track of a model's motion silhouette, and use it to create a polygonal mesh that, when inserted into the scene, gives the appearance of a motion blur. The method is generic enough to work on nearly any moving polygonal model.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Motion blur is important to real-time applications, as a way to portray an object in motion and draw the viewer into the scene. The prior methods are either not real-time [KB83, DK00, BE01], visually unacceptable [nVi01], or unable to deal with highly deforming meshes [WZ96].

Using a method similar to that of swept volume construction, we determine the set of edges that lie along its *silhouette of motion*. By connecting silhouettes captured at regular time intervals with polygonal quadrilaterals, and decreasing the opacity of the polygons the farther back in time they were created, we create a visually appealing motion blur. This generated set of polygons is referred to as the *blur shell*.

Primary distinguishing features of our method are that it:

- Produces a motion blur even for meshes that are animating and deforming,
- Can be used to enhance realism, add an effect to interactive scenes, or show motion in a still image,
- Has low computational overhead,
- Easily integrates with other graphics applications,
- Can be easily implemented with today's graphics hardware, and should be able to improve with future hardware advances.

[†] Supported in part by NSF grant CCR-0220047

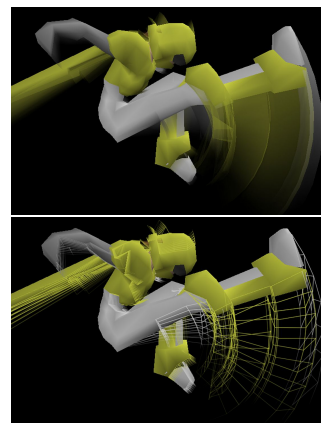


Figure 1: Top: Our geometric motion blur effect. Bottom: The blur shell is rendered in wireframe.

2. Method

Our scheme creates a *blur shell*, a set of polygons that when added to the scene gives the impression that there is a motion blur associated with the object. We assume the mesh can deform, but that we can easily obtain world position and normal information. Smoothly varying normals give the best results. We take as input the length of time the blur should represent (from one frame to several frames) and the number of subdivisions in the blur shell (the more subdivisions, the smoother the look, but slower the computation). On each update (may be more or less than a frame) we find the Silhouette of Motion for the mesh. In each frame displayed, we construct polygonal strips representing the blur shell.

2.1. Finding the Silhouette of Motion

We define the silhouette of motion, or SoM, as the set of edges on a polygonal mesh that represents the local silhouette as seen when looking at an object along its axis of motion. We use a per-polygon silhouette detection technique. We examine each polygon—if one of its edges lies on the SoM, that edge is stored in a list associated with the polygon. The collection of all edges found in this manner will be an approximation to the actual silhouette.

We first determine which side of motion each vertex is on. The facing of any vertex is determined by the sign of the dot product between the world normal of the vertex and the vertex's direction vector. Next we find the set of edges that lie on the SoM. Each polygon with one front facing vertex (positive dot product) and two back facing vertices (negative dot product) records the edge for two back facing vertices, storing both positions and normals. This is an edge of the SoM. Polygons without such an edge, but which have recently had an edge on the SoM, store the positions and normals of that recent edge.

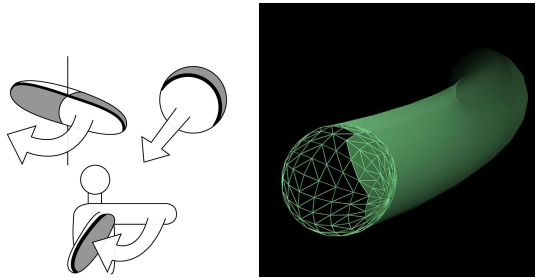


Figure 2: Left: Silhouettes of Motion (dark line); the trailing region is shaded. Right: the blur shell for a sphere.

2.2. Rendering the Blur Shell

To display the blur, we use quadrilateral strips. Similar to how the method generates the SoM, each polygon independently displays any blur associated with the edges that belong to it. Polygons that have not had an edge on the SoM within some fixed time frame do not contribute to the blur.

Each polygon adds pairs of vertices to a quad strip in order to build up the blur. To create the quad strip, we first add the mesh vertices that belonged to the edge the polygon last had on the SoM. Then we cycle through each of the polygon's list elements, starting with the one corresponding to the update index. The two vertices stored in each list element, with their normals, are added to the quad strip. The normals are used to help retain the shading the mesh had at the point the vertices were stored. At most, there will be a number of vertex pairs equal to the number of subdivisions plus one, (the extra pair connects the blur to the base mesh). The number of additional polygons is smaller than for other geometric blur approaches.

To create the blur effect, the vertices that make up the polygon strips are given decreasing alpha values, relying on the hardware to blend the blur shell's polygons. If the strip is displayed with the full number of subdivisions, the last set of vertices added will be completely transparent.

2.3. Analysis and Limitations

Our basic method has some drawbacks, but we can easily address most of these. Holes can appear when an edge continually moves on and off of the SoM; hysteresis removes this problem. Rather than removing a blur strip immediately when the edge is not on the SoM, we fade it out over time, eliminating any popping effect. Highly concave objects can create a blur shell that pierces the rest of the object; fixing this requires tagging (eliminating the blur calculation) of the concave parts. Our method, like all other geometric motion blur techniques, is not geared toward texture-mapped objects, but there are ways to approximate the blurred texture. Finally, backfacing polygons in the blur shell cannot be culled, though there are methods that require significantly more work that could render these with better visual effect.

3. Implementation

The performance of our method will depend on a number of factors: the complexity of the moving objects, the performance of the graphics hardware, implementation optimizations, whether the application is CPU-bound or rendering-bound, etc. For models we tested, blur-shell rendering (GPU and bandwidth bound) was the dominant additional cost for low (less than 1000) polygon models, while SoM detection (CPU bound) became more important in high (several thousand) polygon models. For all test cases, the geometric blur was only a minor part of the overall animation system runtime (usually <1%).

References

- [BE01] BROSTOW G., ESSA I.: Image-based motion blur for stop motion animation. In *Proceedings of Siggraph* (2001), pp. 561–566. 1
- [DK00] DACHILLE F., KAUFMAN A.: High-degree temporal antialiasing. In *Proceedings of Computer Animation* (2000), pp. 49–54. 1
- [KB83] KOREIN J., BADLER N.: Temporal anti-aliasing in computer generated animation. In *Proceedings of Siggraph* (1983), pp. 377–388. 1
- [nVi01] NVIDIA: *Developer Relations Notes*. Tech. rep., 2001. http://developer.nvidia.com/object/motion_blur.html. 1
- [WZ96] WLOKA M., ZELEZNIK R.: Interactive real-time motion blur. *The Visual Computer* 12, 6 (1996), 283–295. 1