Motion Blur Algorithm for Clipped Triangle Rendering

Jerry Tessendorf

December 6, 2004

1 Clipped Triangle Framework

The rendering process reduces the 3D geometry to a set of 2D triangles in the image plane. The projection from 3D is standard. The projected triangles are clipped and sorted so that the final collection of triangles in a pixel do not overlap each other. So at frame f, the image plane holds a collection of pixels $P_k, k = 1, \ldots M$, each of which contains a set of 2D nonoverlapping triangles $\{T_q(f), q = 1 \ldots N_k(f)\}$. We give the triangles a frame dependence because the motion blur process means the geometry moves around in the image plane. Also, the number of triangles in the pixel varies with frame number.

The set of all of the triangles from all of the pixels in the image plane will be labelled $\mathcal{T}(f)$:

$$\mathcal{T}(f) = \{T_q(f),\ q = 1 \ldots N_k(f),\ k = 1, \ldots M\} \tag{1}$$

which, for our purposes we simply relabel the triangles without any particular ordering:

$$\begin{aligned} \mathcal{T}(f) &= \{T_q(f),\ q = 1 \ldots R(f)\} \\ R(f) &= \sum_{k=1}^{M} N_k \end{aligned}$$

and again, the number of triangles in the image plane, $R(f)$ will in general vary with time.

For our purposes here, each triangle consists of three image plane vectors and three range values (distance from camera to the 3D vertex):

$$T = (\vec{u}_0, \vec{u}_1, \vec{u}_2, R_0, R_1, R_2) \tag{2}$$

Suppose we have a set of triangles at frame $f = 0$ and we label them $\mathcal{T}_0(0)$, i.e. the triangles generated at frame 0, positioned as in frame 0. In principle, as the camera and/or geometry move from frame 0 to frame 1, we could track the motion of these triangles to any intermediate time f, and denote that collection of triangles at $\mathcal{T}_0(f)$. Similarly, at frame $f = 1$ the clipping process generates a different set of triangles, denotes by $\mathcal{T}_1(1)$, which in general we can track *backward* in time to times intermediate between 0 and 1. At anytime in this range these triangles are denoted $\mathcal{T}_1(f)$. In general, these two sets are not equal for several reasons: (1) because of both the motion and hiding/reveal behavior, the clipped sets in different, and (2) the lighting of the two sets is in general different. But although $\mathcal{T}_1(f) \neq \mathcal{T}_0(f)$, some triangles in the two sets will overlap. This overlap is very important because individually either set can introduce gaps in the geometry due to relative separation of the geometry during the frame time. These gaps are artificial because additional geometry is being revealed by them, but that new geometry is not reflected in the motion of the original triangles. The overlapping triangles in the combined set take into account this geometry reveal/hide and should provide a complete and accurate set.

The key to motion blur in this algorithm is that at any given subframe time f we can build a new but suboptimal set of triangles out of the union of the tracked triangles $\{\mathcal{T}_0(f),\ \mathcal{T}_1(f)\}$ from the two bounding frames. The set is suboptimal in that there are many triangles that overlap. This set can be made optimal by subsequentally reclipping them to generate an optimal set labelled $\mathcal{T}^{opt}(f)$, and using that to compose the subframe at time f.

In the next several sections we explore the following topics:

Section 2: We develop the equations for the dynamics of a vertex that moves across the camera plane due to motion of the camera and geometry intraframe. For this analysis we will take the motion to be simple. However even for simple camera and geometry motion, the motion of a vertex in the image plane is not so simple. By capturing accurately the vertex motion a more realistic blur can be built.

Section 3: We combine the vertex dynamics of section 2 and triangle clipping process described above into a complete algorithm for motion blurred rendering.

2 Vertex Dynamics for Camera and Geometry Motion

In this section we lay out the equations of motion for a vertex across the image plane due to the motion of the camera and geometry. We begin by building the equation for the projection of points in 3D space to points on the image plane. Then we look at the dynamical equations of motion that are induced when camera/geometry motion take place.

2.1 Projection

The first step in setting up the dynamics is laying out the projection equations defining the important variables and dependencies. Here we describe the projection equations to be used to build the vertex dynamics model for the next subsection.

A point (or vertex) at the 3D spatial position $\mathbf{r}$ projects to a position on the image plane $\mathbf{p}$ using the straightforward projection equation

$$\mathbf{p} = \frac{\mathbf{r} - \mathbf{r}_c}{\hat{\mathbf{n}}_c \cdot (\mathbf{r} - \mathbf{r}_c)} - \hat{\mathbf{n}}_c \tag{3}$$

In this expression, $\mathbf{r}_c$ is the 3D position of the image plane and $\hat{\mathbf{n}}_c$ is the camera viewing direction. With this notation, the image plane position $\mathbf{p}$ is actually a position in 3D space. However, it is located on a plane perpendicular to the view direction, as can be seen explicitly by taking the inner product of the right hand side with $\hat{\mathbf{n}}_c$. Being in a plane, we can define two unit vectors $\{\hat{\mathbf{e}}_i, i = 1, 2\}$ that are orthonormal to themselves and to the view direction:

$$\hat{\mathbf{e}}_i \cdot \hat{\mathbf{e}}_j = \delta_{ij} \tag{4}$$

$$\hat{\mathbf{e}}_i \cdot \hat{\mathbf{n}}_c = 0 \tag{5}$$

Typically one of these directions corresponds to the up direction of the image. Using these two unit vectors, the image plane point $\mathbf{p}$ can be written in terms of coordinates (p_1, p_2):

$$\mathbf{p} = p_1 \hat{\mathbf{e}}_1 + p_2 \hat{\mathbf{e}}_2 \tag{6}$$

2.2 Component Dynamics

For our purposes, dynamics shall consist of the motion of the camera and/or geometry. Camera motion is described as a change in position with velocity $\dot{\mathbf{r}}_c$, a change of viewing direction with angular velocity $\dot{\hat{\mathbf{n}}}_c$, and a spin of the camera about its viewing axis, $\dot{\hat{\mathbf{e}}}_1$.

For geometry motion, we need focus on the motion of individual vertices. A vertex at position $\mathbf{r}$ moves with velocity $\dot{\mathbf{r}}$.

Putting all of this motion together, the vertex at $\mathbf{r}$ which maps to the image plane position $\mathbf{p}$ moves with image plane velocity

$$\begin{aligned} \dot{\mathbf{p}} &= \sqrt{1 + |\mathbf{p}|^2}\,\{1 - (\mathbf{p} + \hat{\mathbf{n}}_c)\hat{\mathbf{n}}_c\} \cdot \frac{\dot{\mathbf{r}} - \dot{\mathbf{r}}_c}{|\mathbf{r} - \mathbf{r}_c|} \\ &- \{1 + (\mathbf{p} + \hat{\mathbf{n}}_c)\mathbf{p}\} \cdot \dot{\hat{\mathbf{n}}}_c \end{aligned} \tag{7}$$

so that even for the simplest camera and geometry moves, the path a vertex takes across the image plane is not a straight line. In addition to this motion, there is additional motion of the components of the image plane location because rotation of the camera generates motion of the image plane. Using the component expansion in equation 6, the rate of change of the components p_i is obtained from

$$\dot{\mathbf{p}} = \dot{p}_1 \hat{\mathbf{e}}_1 + \dot{p}_2 \hat{\mathbf{e}}_2 + p_1 \dot{\hat{\mathbf{e}}}_1 + p_2 \dot{\hat{\mathbf{e}}}_2 \tag{8}$$

Keep in mind that the two basis vectors $\hat{\mathbf{e}}_1$ and $\hat{\mathbf{e}}_2$ are orthonormal as in equation 4. Because of this we can write the most general form:

$$\hat{\mathbf{e}}_i \cdot \dot{\hat{\mathbf{e}}}_j = \epsilon_{ij}\, \omega \tag{9}$$

The quantity ω is the angular velocity of the changing coordinate system as the camera undergoes rotation, and derives from the camera motion data. Combining equations 8 and 9 we get equations for the rate of change of the coordinates as

$$\dot{p}_i = -\sum_j \epsilon_{ij}\, p_j\, \omega \;\; + \;\; \hat{\mathbf{e}}_i \cdot \dot{\mathbf{p}} \tag{10}$$

or explicitly

$$\begin{aligned}\dot{p}_1 &= -p_2\,\omega + \hat{\mathbf{e}}_1\cdot\dot{\mathbf{p}}\\ \dot{p}_2 &= p_1\,\omega + \hat{\mathbf{e}}_2\cdot\dot{\mathbf{p}}\end{aligned}$$

If we now combine these last two equations with equation 7, we get the coupled equations

$$\dot{p}_1 = -p_2\,\omega + \sqrt{1+|\mathbf{p}|^2}\,\{\hat{\mathbf{e}}_1 - p_1\,\hat{\mathbf{n}}_c\}\cdot\frac{\dot{\mathbf{r}}-\dot{\mathbf{r}_c}}{|\mathbf{r}-\mathbf{r}_c|} - \{\hat{\mathbf{e}}_1 + p_1\,\mathbf{p}\}\cdot\dot{\hat{\mathbf{n}}}_c \tag{11}$$

$$\dot{p}_2 = p_1\,\omega + \sqrt{1+|\mathbf{p}|^2}\,\{\hat{\mathbf{e}}_2 - p_2\,\hat{\mathbf{n}}_c\}\cdot\frac{\dot{\mathbf{r}}-\dot{\mathbf{r}_c}}{|\mathbf{r}-\mathbf{r}_c|} - \{\hat{\mathbf{e}}_2 + p_2\,\mathbf{p}\}\cdot\dot{\hat{\mathbf{n}}}_c \tag{12}$$

To solve for the path a vertex takes across the camera plane, we will have to integrate this equation numerically. First, we need to understand the form of the input data for the camera and geometry motion.

2.3 Motion Parameters

Lets keep the problem as simple as possible. We suppose that all we know about the motion of the camera and geometry is that it is uniform between frames, and we have the initial values and final values (i.e. the camera pose and vertex positions at the precise frame times). Then we have the following observations:

1. The velocities $\dot{\mathbf{r}}$ and $\dot{\mathbf{r}}_c$ are constant, and the relative position $\mathbf{r}-\mathbf{r}_c$ changes linearly in time between the position at $f=0$ and $f=1$.
2. The angular rate of change of the camera coordinate system is constant, but $\hat{\mathbf{e}}_i$, $\hat{\mathbf{n}}_c$, ω, and $\dot{\hat{\mathbf{n}}}_c$ are not constant during the interval.

So how do we represent the rotated quantities in the second observation? Here we have the benefit of using rotation matrices. Since these are 3D rotations, we need to use a notation that helps us do these calculations. In particular, we need to define the unit vector $\hat{\mathbf{e}}_3 \equiv \hat{\mathbf{n}}_c$. Then the set $\{\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \hat{\mathbf{e}}_3\}$ are an orthonormal basis for R^3.

Now suppose that at $f=0$ we know the basis is $\{\hat{\mathbf{e}}_1(0), \hat{\mathbf{e}}_2(0), \hat{\mathbf{e}}_3(0)\}$ and at time $f=1$ it is $\{\hat{\mathbf{e}}_1(1), \hat{\mathbf{e}}_2(1), \hat{\mathbf{e}}_3(1)\}$. The relationship between the two is that of a rotation, expressed generally as

$$\hat{\mathbf{e}}_i(1) = \mathbf{R}\cdot\hat{\mathbf{e}}_i(0) \tag{13}$$

The elements of the matrix $\mathbf{R}$ can be obtained directly from inner products of the basis at the two times:

$$(\mathbf{R})_{ij} = R_{ij} = \hat{\mathbf{e}}_i(0)\cdot\hat{\mathbf{e}}_j(1) \tag{14}$$

What we need in order to do the dynamics calculations is a rotation matrix for intraframe times, $\mathbf{R}(f)$, with the boundary conditions $\mathbf{R}(0)=1$ and $\mathbf{R}(1)=\mathbf{R}$. For this purpose, it is useful to represent $\mathbf{R}$ in the SO(3) representation:

$$\mathbf{R} = \exp(\vec{\alpha}\cdot\vec{\tau}) = 1\cdot\cos(\alpha) + \hat{\alpha}\hat{\alpha}\,(1-\cos(\alpha)) + \hat{\alpha}\cdot\vec{\tau}\,\sin(\alpha) \tag{15}$$

Here the vector $\vec{\tau}$ is a triplet of 3X3 matrices that are traceless and have the explicit form

$$(\tau_k)_{ij} = \epsilon_{ijk} \tag{16}$$

There is a very direct relationship between the $\mathbf{R}$ representation of equation 14 and equation 15. First, α can be constructed from the trace:

$$trace(\mathbf{R}) = \sum_i \hat{\mathbf{e}}_i(0)\cdot\hat{\mathbf{e}}_i(1) = 1 + 2\cos(\alpha) \tag{17}$$

$$trace(\tau_k\cdot\mathbf{R}) = \sum_{ij}\epsilon_{ijk}\hat{\mathbf{e}}_i(0)\cdot\hat{\mathbf{e}}_j(1) = -2(\hat{\alpha})_k\,\sin(\alpha) \tag{18}$$

The purpose of this approach is to allow us to create an intraframe representation of the rotation, $\mathbf{R}(f)$. We can easily accomplish this from the SO(3) representation as:

$$\mathbf{R}(f) = \exp(\vec{\alpha}\cdot\vec{\tau}f) = 1\cdot\cos(\alpha f) + \hat{\alpha}\hat{\alpha}\,(1-\cos(\alpha f)) + \hat{\alpha}\cdot\vec{\tau}\,\sin(\alpha f) \tag{19}$$

Case	Initial Position	Initial View Direction	Initial Up Direction	Final Position	Final View Direction	Final Up Direction
2 Spin	x, y, 10	0, 0, 1	0, 1, 0	x, y, 10	0, 0, 1	0.1, 1, 0
3 Pan	x, y, 10	0, 0, 1	0, 1, 0	x, y, 10	0, 0.1, 1	0, 1, 0
4 Spin/Pan	x, y, 10	0, 0, 1	0, 1, 0	x, y, 10	0, 0.1, 1	0.1, 1, 0
5 Recede	x, y, 10	0, 0, 1	0, 1, 0	x, y, 20	0, 0, 1	0, 1, 0
6 Recede/Spin/Pan	x, y, 10	0, 0, 1	0, 1, 0	x, y, 20	0, 0.1, 1	0.1, 1, 0

Table 1: Initial and final camera and object positions for computed blur paths. x and y are any of the selected positions on a grid in the image plane.

Knowing the rotation matrix, we can construct all of the angular variables needed:

$$\hat{\mathbf{e}}_i(f) = \hat{\mathbf{e}}_i(0)\ \cos(\alpha f) + \hat{\alpha}(\hat{\alpha}\cdot\hat{\mathbf{e}}_i(0))\ (1-\cos(\alpha f)) + \hat{\alpha}\times\hat{\mathbf{e}}_i(0)\ \sin(\alpha f) \tag{20}$$

$$\hat{\mathbf{n}}_c(f) = \hat{\mathbf{n}}_c(0)\ \cos(\alpha f) + \hat{\alpha}(\hat{\alpha}\cdot\hat{\mathbf{n}}_c(0))\ (1-\cos(\alpha f)) + \hat{\alpha}\times\hat{\mathbf{n}}_c(0)\ \sin(\alpha f) \tag{21}$$

$$\dot{\hat{\mathbf{n}}}_c(f) = -\hat{\mathbf{n}}_c(0)\ \alpha\ \sin(\alpha f) + \hat{\alpha}(\hat{\alpha}\cdot\hat{\mathbf{n}}_c(0))\alpha\ \sin(\alpha f) + \alpha\ \hat{\alpha}\times\hat{\mathbf{n}}_c(0)\ \cos(\alpha f) \tag{22}$$

$$\omega(f) = \alpha\ \hat{\mathbf{e}}_1(0)\cdot(\hat{\alpha}\times\hat{\mathbf{e}}_2(0)) \ +\ \alpha\ (\hat{\alpha}\cdot\hat{\mathbf{e}}_1(0))\ (\hat{\alpha}\cdot\hat{\mathbf{e}}_2(0))\ \sin(\alpha f)\cos(\alpha f) \tag{23}$$

2.4 Numerical Evaluation

We can begin to get some intuitive feeling for what these vertex paths look like in the image plane by numerically evaluating the dynamical equations 11 and 12 above. The numerical solver implemented for this is a straight forward explicit scheme contained in the class `BlurDynamics`. The test code that uses this class is listed in Listing 1. The vectors `start_p` and `end_p` are the initial and final 3D positions of the vertex relative to the camera. Similarly, `start_view`, `end_view`, `start_up`, and `end_up` are the initial and final vectors for the camera orientation. The values of these vectors for each of the five demonstration cases are shown in Table 1. The orientation vectors are used by `BlurDynamics` to create the orthonormal 3D basis $\hat{\mathbf{e}}_1,\ \hat{\mathbf{e}}_2, \hat{\mathbf{n}}_c$, with $\hat{\mathbf{e}}_1$ corresponding to the up direction.

Examining the various demonstration cases, one thing that is clear is that when the camera is rotating, this blur path approach produces blur streaks along curved paths. This should provide realistic blur effects in many situations.

3 Blur Algorithm

So now we know how to assemble a collection of nonoverlapping triangles at the beginning of a frame, and propagate the vertices of those triangles to their positions at any later time in that frame. We could also go backwards in time from the end of a frame.

Now we just have to assemble a scheme for generating a good blurred image from this information.

3.1 Subframe Selection

The blur algorithm requires that we propagate triangles from the end frames 0 and 1 to a set of common subframes between them. So a primary issue is selecting the number and/or time spacing of the subframes. Ultimately this is a user decision, but we need to provide the user with some conservative criterion that indicates the most rigorous amount of blurring that might be needed.

A useful guide for subframe spacing is that we want the typical amount of distance a vertex propagates between subframes to be around or less than 1 pixel. This choice guarantees that the blur does not have gaps in its structure. Such a choice leads to the subframe spacing estimate of

$$df = \max\left\{\frac{\sqrt{a_p}}{\sqrt{\dot{p}_1^2+\dot{p}_2^2}}\right\} \tag{24}$$

where a_p is the area of a pixel. This quantity cannot be taken literally because, among other things, its values varies across the image plane. However, it may prove a useful guideline.

```
//-----------------------------------------
//
//  Little test program to evaluate blur dynamics
//
//  Jerry Tessendorf  December 18, 2003
//
//-----------------------------------------
#include <iostream>
#include "BlurDynamics.h"
using namespace hog;
using namespace std;
int main(int argc, char **argv )
{
   BlurDynamics dyn;
   for( double x=-20;x<=20;x += 4 )
   {
   for( double y=-20;y<=20;y+= 4 )
   {
      Vector start_p( x,y,10 );
      Vector end_p( x,y,20 );
      Vector start_view( 0,0,1 );
      Vector end_view( 0,0.1,1 );
      Vector start_up( 0,1,0 );
      Vector end_up( 0.1, 1,0 );
      dyn.SetMotion( start_p, start_view, start_up,
                     end_p,   end_view,   end_up    );
      while( dyn.GetTime() <= 1.0 )
      {
         double p1, p2;
         dyn.Update( p1, p2 );
         cout << dyn.GetTime() << "  " << p2 << "  " << p1 << endl;
      }
      cout << endl << endl;
   }
   }
};
```

Figure 1: The test code.

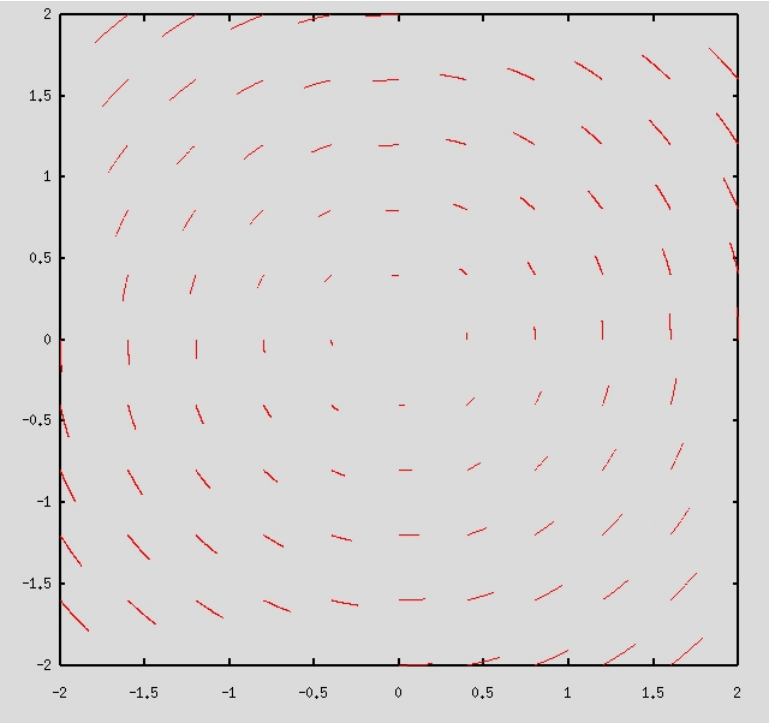

Figure 2: Blur paths due to a six degree spin of the camera.

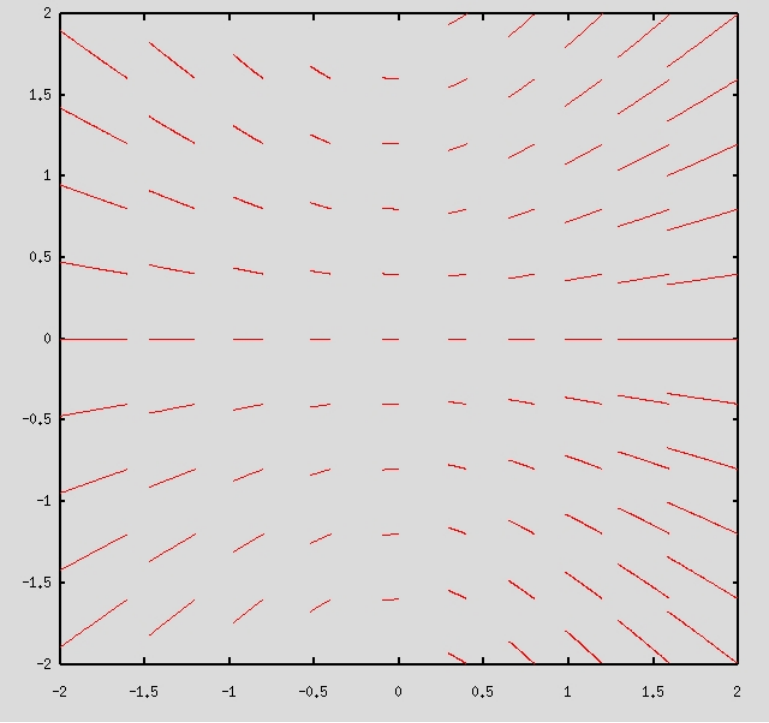

Figure 3: Blur paths due to a six degree pan of the camera.

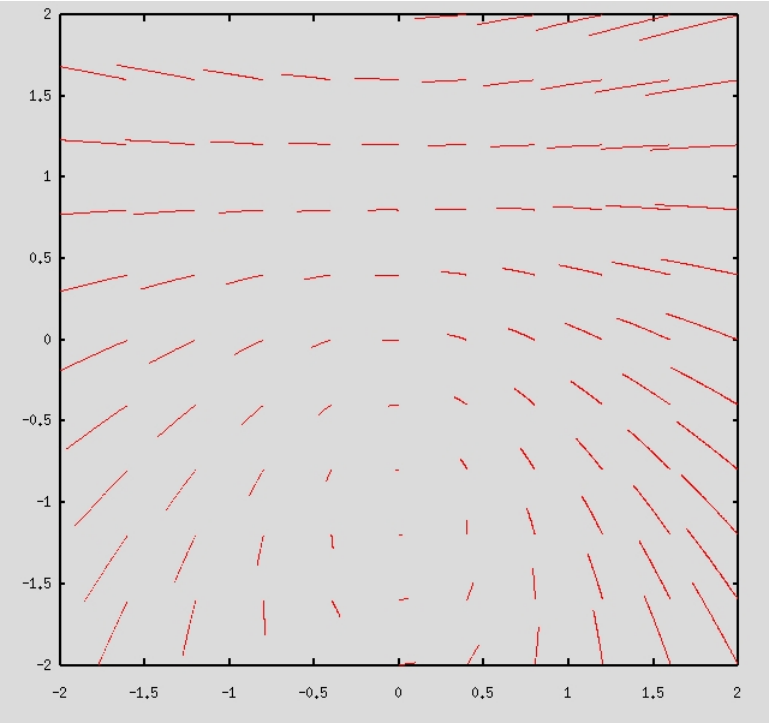

Figure 4: Blur paths due to a six degree spin and a six degree pan of the camera.

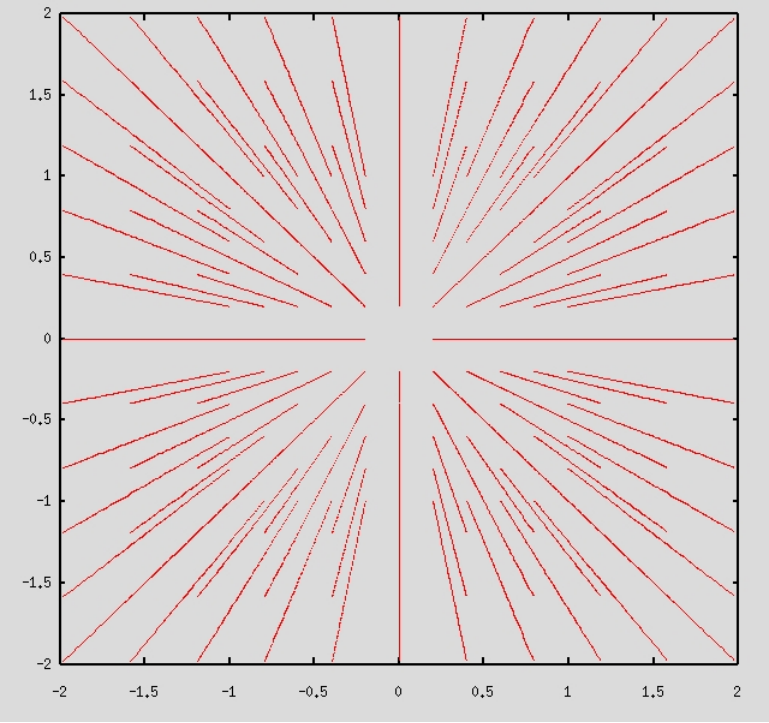

Figure 5: Blur paths due movement of vertex away from camera.

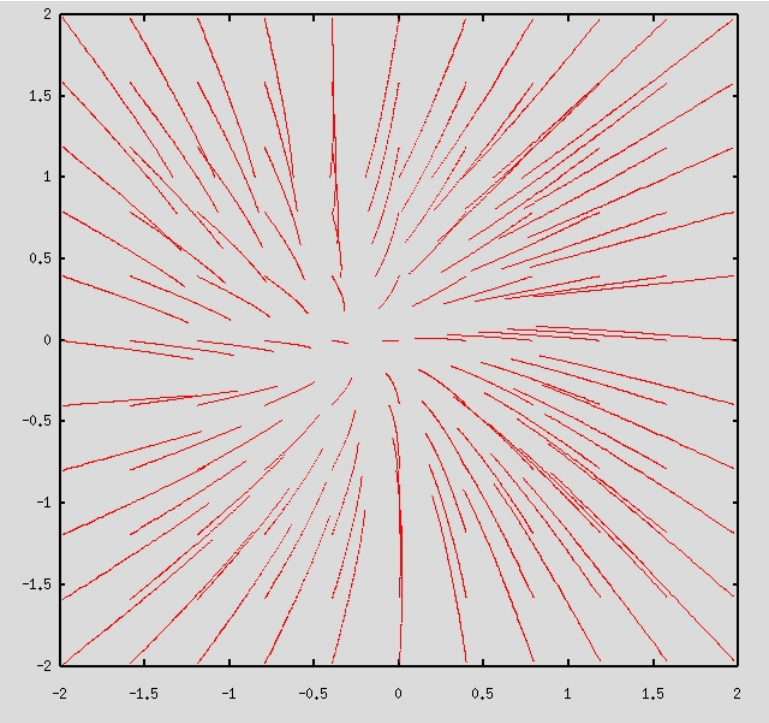

Figure 6: Blur paths due to movement of vertex away from camera, a six degree spin, and a six degree pan of the camera.

3.2 Pre and Post Projection

For subframe f, the triangles $\mathcal{T}_0(0)$ from frame 0 are propagated forward in time to the set $\mathcal{T}_0(f)$ using the dynamics described above. Simultaneously, the triangles $\mathcal{T}_1(0)$ from frame 1 are propagated backward in time to the set $\mathcal{T}_1(f)$ at the same subframe as the set $\mathcal{T}_0(f)$. The propagation of both sets of triangles to a common subframe also requires that the range values of the projected triangles are also updated.

The two sets contain triangles that overlap each other and have different lighting/shading values. However, the combination of the two sets of triangles forms an overlapping set of triangles covering all of the geometry and screen space that is needed. Also, the triangles in both sets no longer reside wholly within pixels, but cross pixel boundaries in general. The fact that the triangles overlap and cross pixel boundaries means that additional processing is necessary before rendering the color and other data for each pixel. The additional processing is clipping.

3.3 Clipping

With the combined set of overlapping triangles $\{\mathcal{T}_0(f),\ \mathcal{T}_1(f)\}$, the next step is to apply clipping to reduce them to a final set of nonoverlapping triangles within each pixel. This is very much like the primary clipping/rastering of the hog renderer, except that some stages of that process need not be applied here.

During the clipping it is possible that some overlapping triangles have the same projected range, so that range sorting does not completely eliminate overlapping. This can occur when triangles from each of the two sources cover a common area. In this situation triangles from one or the other set must be chosen. One possible scheme for that is to chose one of them, but give the final triangle fragments a color that is a weighted mix of the colors from the two original triangles.

The outcome of this stage is an ”optimized” set of triangles $\mathcal{T}^{opt}(f)$ which are nonoverlapping, contained within pixels, and ready to complete the rendering of this subframe.

3.4 Averaging

The triangles that were used to construct the optimal set $\mathcal{T}^{opt}(f)$ have already been colored. The final stage is to update the motion blur averaged image frame by (1) reducing the triangles in each pixel to a final pixel color, and (2) adding this color to the running blurred color of the pixel, using an appropriate weight proportional to the iverse of the number of subframes.