Cheap Field Rendering

Daniel Maas, Maas Digital LLC*

Introduction

Occasionally we are asked to create CGI for an interlaced video format. Instead of 24 or 30 progressive frames per second, we need to create 60 interlaced fields. At first it seems like we have to render 60 full-resolution frames per second and throw out half the scanlines. Compared to progressive video, this doubles the amount of time to render a shot! (and rendering each field at half vertical resolution causes unacceptable blurriness on stills).

Ideally we would like to find a method that does not require any more shading effort than a progressive-frame render. It turns out that this is indeed possible. The difference between progressive and field rendering is mainly that a different shutter interval is used on alternate scanlines. If we render a frame progressively, but store all the shaded, motion-blurred micropolygons, we can then go back and "re-sample" the motion with a different shutter interval. By re-sampling the micropolygons at two different intervals, we can create the two fields of an interlaced frame with only a little more effort than a single progressive render.

Pass 1: Dump Micropolygons

Thanks to the new geometry display driver feature in PRMan version 13, we can read out micropolygon grids cleanly and easily. We simply write a display driver that turns each grid into a (motion-blurred) RIB PointsPolygons statement. The RIB driver that ships with PRMan is almost adequate, but it doesn't handle motion blur, so we have to modify it slightly.

The standard display driver already has access to the surface point positions, `P`, at shutter open. We also need to pass in the positions at shutter close, which are computed by the shader using the motion blur vector `dPdtime`. And finally, we also store the micropolygon's shaded color and opacity.

Pass 2: Rasterize Fields

Once we have a RIB file containing all of the shaded, moving micropolygons, we can simply read it back into the renderer with a different shutter interval. This grabs the correct portion of the motion blur for each field. We render a full-resolution image at each half frame interval, and then interlace the successive fields together. These re-renders are very fast since we are just rasterizing already-shaded micropolygons.

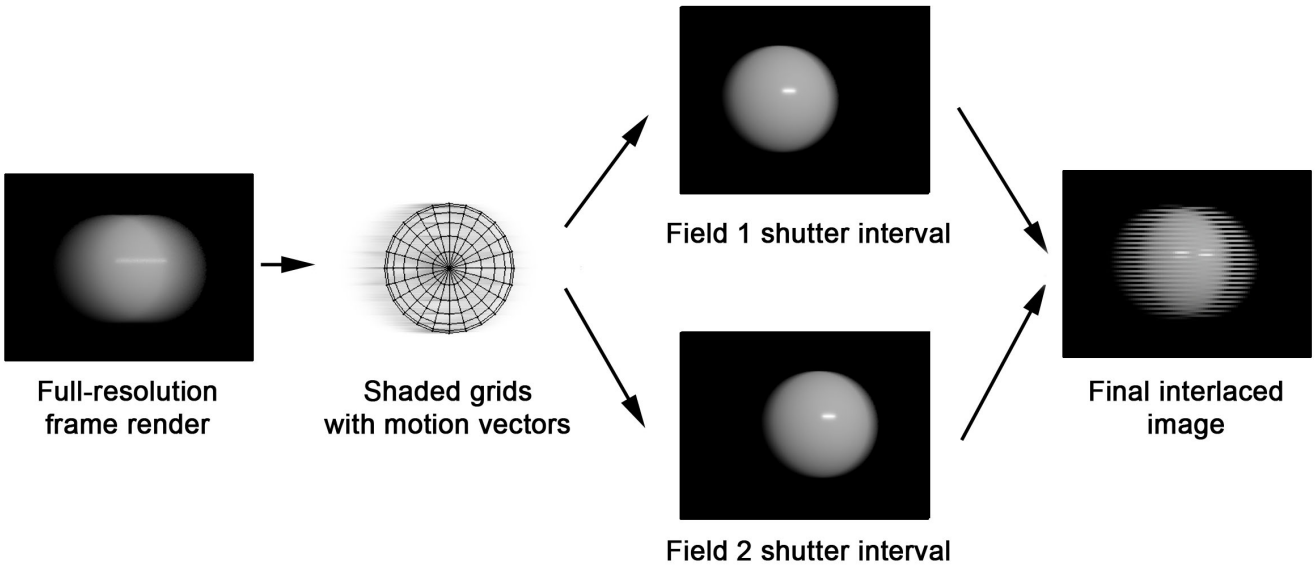
Notes

More than two motion samples per frame may be necessary to capture rapid motion accurately. Furthermore, we are only computing the shading once for every pair of fields, so highlights and shadows won't quite match a true full-resolution field rendering. We can always fall back to full-resolution field rendering on a particular shot if the artifacts become apparent. RIB dumping is rather slow, and in simple scenes it can cause the two-pass approach to render more slowly than two ordinary progressive frames.

The RIB files containing micropolygon dumps tend to be quite large (hundreds of megabytes). There are several ways we can deal with this problem. Encoding into the gzipped binary format will save disk space at the expense of some CPU time. To save memory during the second rendering pass, we can bucketize the RIB, breaking it into pieces loaded on demand via DelayedReadArchive.

The ultimate solution would be if RenderMan allowed the use of a different shutter interval on alternate scanlines within a single render. This would enable efficient field rendering in a single pass.

*dmaas@maasdigital.com



Example Code

Pass 1: Dump Micropolygons

We do our RIB dumping with an atmosphere shader which can easily be applied to all objects in the scene. The only catch is that we must message-pass the surface normal from the surface shader, since atmosphere shaders cannot ordinarily access the normal. We dump the shaded color and opacity as Cs and Os so that the rasterization pass can use a "constant" shader.

Sample RIB:

```
# Cs = shaded color, Os = shaded opacity
DisplayChannel "varying color" "Cs"
DisplayChannel "varying color" "Os"

# P at shutter close (end of frame interval)
DisplayChannel "varying point" "blurP"

Shutter 0 1 # capture the entire frame interval

# enable accurate motion interpolation
Option "shutter" "clampmotion" [0]
...
WorldBegin

# use 1/4th of your usual motionfactor value
GeometricApproximation "motionfactor" 0.25

# run the dumprib shader on all geometry
Atmosphere "dumprib" "string filename" "myrib.rib"
...
```

```
WorldEnd
```

The atmosphere shader:

```
atmosphere dumprib(string filename = "";)
{
  normal surfN = normal(0,0,0);
  if(surface("surfN", surfN) == 0) {
    printf("surface did not pass surfN\n");
  }

  // P at shutter close
  point blurP = P + dtime*dPdtime;

  bake3d(filename, "Cs,Os,blurP",
         P, surfN,
         "coordsystem", "camera",
         "driver", "dumprib",
         "Cs", Ci, "Os", Oi,
         "blurP", blurP);

  // note: our "dumprib" display driver is identical to the one
  // that ships with PRMan, except that if it sees a primitive
  // attribute called "blurP", it will output motion-blurred
  // geometry where "blurP" is used in place of "P" for the
  // second motion sample.
}
```

Pass 2: Rasterize Fields

During the second pass, it is vital that we use the exact same camera parameters, `PixelFilter`, and `PixelSamples` as the first pass. This is necessary because PRMan will cull any micropolygons that don't cross a visible point sample. If the sample locations used for the second pass do not exactly match those of the first pass, we will see "pinprick" holes in surfaces where micropolygons were culled. (We could avoid this problem by disabling occlusion culling, but it is usually too good an optimization to pass up).

```
# use same camera parameters as the first pass

# shutter interval for first field
Shutter 0.00 0.25
#  (use 0.50 0.75 for second field)

# enable accurate motion interpolation
Option "shutter" "clampmotion" [0]
...
WorldBegin

# emit geometry in camera space
```

```
CoordSysTransform "camera"

Surface "constant_premul"

# no need to subdivide the micropolygons
ShadingRate 9999999

# Sides 2 is necessary for all geometry to show up
Sides 2

ReadArchive "myrib.rib"

WorldEnd
```

We need a variation of the "constant" shader that doesn't multiply `Cs` by `Os`, since that was done already in the first pass. The shader also has an option to knock out geometry that faces away from the camera, in order to avoid z-fighting on grids that came from doubleshaded primitives. This should not be enabled by default, since it may cause silhouette edge artifacts on single-sided geometry.

```
surface constant_premul(float knockout = 0;)
{
  Oi = Os;
  Ci = Cs;

  if(knockout) {
      // get rid of non-camera-facing geometry
      if((N.I) > 0) {
          Oi = 0;
          Ci = 0;
      }
  }
}
```