Lecture 6:
3D and 5D Rasterization

Image Synthesis
Stanford CS348b, Spring 2013
Higher-Dimensional Rasterization

- “Spending lots more processing time to render blurry things”

- Motion blur: avoid strobing with fast-moving objects
  - Three dimensions: two spatial (on image), time

- Defocus blur: simulate camera depth of field
  - Four dimensions: two spatial (image), two spatial (lens)

- Motion + defocus blur
  - Five dimensions...
3D Rasterization

- Triangle vertex positions are now a function of time
- Pixel values now come from an integral over both area & time

\[ \int \int f(x, y, t) \, dx \, dy \, dt \approx \frac{1}{N} \sum_{i} f(x_i, y_i, t_i) \]

- Image samples are now \((x_i, y_i, t_i)\)
- Simplifications:
  - Assume linear vertex motion
  - In Reyes, we only shade at a single point in time
Blurry Big Guy
Rendering at Four Distinct Times
Sixteen Times
64 Times
Errors From Shading Once, Linear Motion

- No Motion
- 30° rotation
Errors From Shading Once, Linear Motion

No Motion

30° rotation
Errors From Shading Once, Linear Motion

No Motion

30° rotation

120° rotation

All three should be the same!
Four Samples Per Pixel

Low-Discrepancy Time Samples

Fixed Time Samples
Sixteen Samples Per Pixel

Low-Discrepancy Time Samples

Fixed Time Samples
Sixty-Four Samples Per Pixel

Low-Discrepancy Time Samples

Fixed Time Samples
2D Rasterization Review

- For 2D rasterization, we found raster coordinates \((x_r, y_r)\) using the projection matrix \(M_p\), the raster matrix \(M_r\), and a homogeneous divide:

\[
(x', y', z', w')^T = M_r M_p (x, y, z, 1)^T
\]
\[
(x_r, y_r) = (x'/w', y'/w')
\]

- We then rasterized the triangle formed by these coordinates

- ...this approach doesn’t work in the presence of motion blur
Interpolating Triangle Positions

- In 3D, consider a vertex with linear motion moving from $v_0$ to $v_1$
  $$v(t) = (1-t) v_0 + t v_1$$
- The standard form of the projection matrix is
  $$M_p = \begin{pmatrix}
  a & 0 & 0 & 0 \\
  0 & b & 0 & 0 \\
  0 & 0 & c & d \\
  0 & 0 & e & 0 \\
\end{pmatrix}$$

What happens when we project $v(t)$?
Interpolating Triangle Positions

- Projecting \( v(t) = (1-t) v_0 + t v_1 \):

\[
M_p \ v(t)^T = (x', y', z', w') = (a ((1-t) x_0 + t x_1), ..., e ((1-t) z_0 + t z_1))
\]

- Raster coordinates as a function of time:

\[
x_r(t) = x'/w' = \frac{a ((1-t) x_0 + t x_1)}{e ((1-t) z_0 + t z_1)}
\]

- ...this is not in general equal to first projecting \( v_0 \) and \( v_1 \) to raster coordinates and interpolating:

\[
(1-t) \ \frac{a x_0}{e z_0} + t \ \frac{a x_1}{e z_1}
\]

→ Must interpolate \( x(t), y(t), w(t) \) individually, then divide by \( w(t) \)
Bounding Vertex Positions

- Projected vertex position is a function of time, e.g.

\[ x_r(t) = \frac{x'(t)}{w'(t)} = \frac{(1 - t)x_0 + tx_1}{(1 - t)w_0 + tw_1} \]

- Need lower and upper bounds over time range \([t_0, t_1]\) to bound pixel range of the vertex. How to do this?
Bounding Vertex Positions

- Projected vertex position is a function of time, e.g.
  \[
  x_r(t) = \frac{x'(t)}{w'(t)} = \frac{(1 - t)x_0 + tx_1}{(1 - t)w_0 + tw_1}
  \]

- Need lower and upper bounds over time range \([t_0, t_1]\) to bound pixel range of the vertex. How to do this?

- Option 1: calculus
  - Min/max are either at \(t_0\), \(t_1\), or where derivative is 0

- Option 2: for min, compute min of \(x(t)\), max of \(w(t)\), divide...
Point in Moving Triangle Tests

- Time-continuous edge functions
  - $e(x,y,w,t)$
- Interpolate vertex positions and compute barycentric coordinates
- Both of these are much more expensive than 2D rasterization
  - Best-case 2D rasterization tested a sample with 3 ADDs of incremental cost
  - Here: 18 ADDs, 21 MULs per sample :-(

Stanford CS348b, Spring 2013
Another Challenge of 3D Rasterization

- Bounds of a moving triangle may cover many pixels
- Yet: the number of samples that hit the triangle is proportional to the triangle’s area
  - Think about this carefully
## Interval Algorithm

- Decompose time range into a set of $N$ intervals
  - $[0, 1/N), [1/N, 2/N), ...$
- For each interval:
  - Compute bounding box for triangle motion over interval
  - Find samples inside the bound
  - Only test samples that have a time inside the interval
- Well-designed sample representation eliminates search for sample to test
## How Many Intervals?

- It depends on how much things are moving...

<table>
<thead>
<tr>
<th># intervals</th>
<th>16</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1046ms</td>
<td>1361ms</td>
<td>2076ms</td>
</tr>
<tr>
<td></td>
<td>293ms</td>
<td>592ms</td>
<td>1457ms</td>
</tr>
<tr>
<td></td>
<td>117ms</td>
<td>620ms</td>
<td>3098ms</td>
</tr>
</tbody>
</table>

Time to render moving bigguy at 1080p, 16 samples per pixel
Interleaved Sampling

- Recall averaging N images together for motion blur
  - Equivalent to using same N time sample values in each pixel

Interleaved: have a multiple of N fixed time samples
- Interleave assignment of samples to pixel areas
Interleaved Sampling Comparisons

Fixed

Interleaved 2x2

Low-Discrepancy
Culling for Motion Blur

- Backface culling is tricky
  - Triangle may be back-facing at the start and end time, yet front-facing in the middle!
Culling for Motion Blur

- Backface culling is tricky
  - Triangle may be back-facing at the start and end time, yet front-facing in the middle!
- Can also apply tile culling (tricky!)
  - As before, entire tiles can be culled
Culling for Motion Blur

- Backface culling is tricky
  - Triangle may be back-facing at the start and end time, yet front-facing in the middle!

- Can also apply tile culling (tricky!)
  - As before, entire tiles can be culled
  - Can also compute the time interval the triangle overlaps a tile, only test samples with times within that interval
  - Unfortunately, effective sample patterns are designed to have very different time values at spatially-nearby locations...
Depth of Field

less depth of field

more depth of field

wider aperture

smaller aperture

London and Upton,
Thin Lens Model

Focal Length, $F$, is the distance behind the lens where parallel rays focus.

Lens diameter is $F/n$, where $n$ is the lens's f-number.
Gaussian Lens Formula

Points at distance $z$ focus at distance $z'$ behind the lens given by

$$\frac{1}{F} = \frac{1}{z} + \frac{1}{z'}$$

Gaussian Lens Formula

Points at distance \( z \) focus at distance \( z' \) behind the lens given by

\[
\frac{1}{F} = \frac{1}{z} + \frac{1}{z'}
\]

Lenses are focused by moving them closer to/farther from the image plane.

Thin Lens Demonstration

http://graphics.stanford.edu/courses/cs178-10/applets/gaussian.html
Defocus Blur

Lens is focused at depth $z_f$; points at other depths image to an area on the image plane: the circle of confusion
Defocus Blur

Lens is focused at depth $z_f$; points at other depths image to an area on the image plane: the circle of confusion
Finding the Size of the Circle of Confusion

What is $d_c$, the diameter of the circle of confusion of a point at depth $z$?

We know $z'_i$, the image plane position, and the lens diameter $d$.

We can compute $z'$ from $F$ and $z$:

$$\frac{1}{F} = \frac{1}{z} + \frac{1}{z'}$$

Similar triangles:

$$\frac{d_c}{(z'_i - z')} = \frac{d}{z'}$$

Solve:

$$d_c = d \mid z'_i - z' \mid (1/F - 1/z)$$
Finding the Size of the Circle of Confusion

What is \(d_c\), the diameter of the circle of confusion of a point at depth \(z\)?

We know \(z'_i\), the image plane position, and the lens diameter \(d\).

We can compute \(z'\) from \(F\) and \(z\):

\[
\frac{1}{F} = \frac{1}{z} + \frac{1}{z'}
\]

Similar triangles:

\[
d_c/(z'_i-z') = d/z'
\]

Solve:

\[
d_c = d \mid z'_i - z' \mid (1/F - 1/z)
\]
Defocus Rasterization

- Pixel values now come from an integral over both pixel area and lens area

\[
\int \int f(x, y, u, v) \, dx \, dy \, du \, dv \approx \frac{1}{N} \sum_i f(x_i, y_i, u_i, v_i)
\]

- Image samples are now \((x_i, y_i, u_i, v_i)\)

- Given a point on the image \((x_i, y_i)\) and a point on the lens \((u_i, v_i)\), how do we determine if the sample is in the triangle?
DOF Rasterization

- We can also bound the circle of confusion at a depth $z, d_z$
- Parameterize lens over $[-1, 1]$

Looking through lens at point $u$, objects at depth $z$ are shifted in $x$ by $-u d_z$

For a 2D lens and a 3D scene, equivalent shift in $y$ from $v$
DOF Rasterization

- For each triangle
  - Compute raster-space bounds, accounting for defocus blur expansion of bounds
  - For each sample \((x_i, y_i, u_i, v_i)\) in raster-space bounds
    - Offset triangle vertices in \((x, y)\) based on vertex z depth, \((u_i, v_i)\) sample values, and lens settings
    - Test sample point \((x_i, y_i)\) against offset triangle

- Not super speedy: 15 ADDs, 16 MULs per sample test
Culling for DOF Rasterization

- Backface culling is also not straightforward
- Interval and interleaved algorithms can be applied
  - If we consider a subset of the lens, then an out-of-focus point covers fewer pixels...
- Also, fairly complex tile culling tests have been derived
Motion Blur + Defocus Rasterization

- Now 5D: integrate over pixel area, lens area, and time

\[
\int \int f(x, y, t, u, v) \, dx \, dy \, dt \, du \, dv \approx \frac{1}{N} \sum_i f(x_i, y_i, t_i, u_i, v_i)
\]

- As before, complex backface cull check
- And can apply interval, interleaved, and tile culling algorithms...

- 24 ADDs, 25 MULs per sample test
Aperture and Shutter

f/16 1/8s

f/4 1/128s

f/2 1/500s

London and Upton
Moving Defocus Insanity

7.8M micropolygons, 720p resolution, 256 samples per pixel
## Look On My Hit Rate, and Despair

<table>
<thead>
<tr>
<th>interval $n_u \times n_v \times n_{\text{time}}$</th>
<th>Render Time</th>
<th>Sample Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x1x1</td>
<td>40.39s</td>
<td>2.34%</td>
</tr>
<tr>
<td>1x1x2</td>
<td>46.62s</td>
<td>2.40%</td>
</tr>
<tr>
<td>1x1x4</td>
<td>66.23s</td>
<td>1.75%</td>
</tr>
<tr>
<td>2x2x1</td>
<td>81.01s</td>
<td>1.38%</td>
</tr>
<tr>
<td>2x2x2</td>
<td>77.82s</td>
<td>2.64%</td>
</tr>
<tr>
<td>4x4x1</td>
<td>228.71s</td>
<td>2.65%</td>
</tr>
</tbody>
</table>

720p, 64 spp