Stochastic Rasterization

Eric Enderton, Morgan McGuire
NVIDIA Research
9 fps on Geforce GT 280
>60 fps on Geforce GT 280
Distribution Ray Tracing

- Spatial AA
  - pixel \((x, y)\)
- Motion blur
  - time \((t)\)
- Defocus blur / DOF
  - lens \((u, v)\)
- Monte Carlo sampling
- Same rays sample all dimensions at once
Stochastic Rasterization

- The same thing, but leveraging hardware rasterization
- No rays
- Interactive speeds
- Transparency
MSAA

- **Multi-Sample Anti-Aliasing**
  - Hardware supports up to 8 samples per pixel
  - Spatial anti-aliasing only (triangle edges)
  - Not stochastic

- **Shade once per pixel, not once per sample**
  - Actually once / fragment
  - A major performance criterion for us
(Kayvon Fatahalian, “Beyond Programmable Shading” course, SIGGRAPH 2010)
Familiar stuff, sub-pixel

Pixel patterns for GeForce FX (left) and GeForce 6 Series (right) architectures showing horizontal and vertical values.
Motion Blur
Conventional: no blur

Accumulation: ghosts

Stochastic: noisy

1 sample per pixel

256x MSAA...or 16x + animation?
2D Rasterization

Geometry Shader
1. Bound the triangle’s screen-space extent due to its shape (xy) and motion (t)

2D Rasterizer
2. Iterate over the samples in that bound

Pixel Shader
3. Perform some $xyt$ inside-outside test per sample
4. Shade the samples that pass

$(2+1)D$
Extending an Existing Renderer

C++ Host

Vertex Shader

Geometry Shader

Pixel Shader
Extending an Existing Renderer

C++ Host

+ velocity attribute
+ camera velocity uniform

Vertex Shader

+ duplicate transform code for prev. vtx

Geometry Shader

+ convex hull of prev & current

Pixel Shader

+ set MSAA mask from ray-triangle tests

Shading code is unmodified!

(source code at nvidia.com/research)
Geometry Shader

“Normal”: All $z < 0$: Projected Hull

“$z=0$ crossing”: $z_{\text{min}} < z_{\text{near}}$ and $z_{\text{max}} > 0$: Clip and Box

All $z > z_{\text{near}}$: Cull
Extreme Example of $z = 0$ Case
Extreme Example of $z = 0$ Case
Extreme Example of $z = 0$ Case
Correct result for a moving camera
Integration with MSAA

- Perform ray-triangle test per sample
- Override the pixel’s coverage mask
- Shade at most once per pixel

- Use ray differentials (like screen-space derivatives) to set anisotropic MIP-map levels
// Sample stochastic visibility
for (int i = 0; i < MSAA_SAMPLES; ++i) {
    getRay(origin, dir, i, gl_FragCoord.xy);
    if (intersectRay(origin, dir, time[i], u, v)) {
        gl_SampleMask[0] |= (1 << i);
    }
}

if (gl_SampleMask[0] == 0) discard;

// Barycentric interpolation
vertex = v0 * u + v1 * v + v2 * (1 - u - v);
normal = n0 * u + n1 * v + n2 * (1 - u - v);
texCoord = t0 * u + t1 * v + t2 * (1 - u - v);

// Shade as usual...
Multisample Rate: 1x
Multisample Rate: 8x
Multisample Rate: 16x
Multisample Rate: 64x
Multisample Rate: 256x
Bridge

Motion blur
1.8M Triangles
8 vis, 4 tex, 1 shade / pix
1280 × 720 @ 19 fps
GeForce GTX 480
Sample Test Efficiency

Scene

AABB

Convex Hull
Depth of Field
New Bounding Geometry
Multisample Rate: 1x
Multisample Rate: 4x
Multisample Rate: 8x
Multisample Rate: 16x
Multisample Rate: 64x
Fairy

Defocus blur
174k Triangles
8 vis, 4 tex, 1 shade / pix
1280 x 720 @ 10 fps
GeForce GTX 480
Stochastic Transparency
Transparency

- hair
- foliage
- particles
- windows

- shadows thereof
Order dependent

\[ c = \alpha_1 c_1 + (1 - \alpha_1)(\alpha_2 c_2 + (1 - \alpha_2)\alpha_3 c_3) \]
But sorting is painful

- Sort primitives
  - Fails for overlaps
  - Disrupts engine code

- Sort per pixel
  - A-Buffer
  - Unbounded memory

- Want: Order Independent Transparency (OIT)
Correct on average

\[ c = \alpha_1 c_1 + (1 - \alpha_1)(\alpha_2 c_2 + (1 - \alpha_2)\alpha_3 c_3) \]

- ... *if* the masks are uncorrelated.
- Uses the z buffer to be order independent.
Stochastic Transparency

Screen-door + multi-sampling + random masks.

- Correct on average
- All cases unified in a single algorithm
  - Foliage, Smoke, Hair, Glass, mixtures
- One order-independent pass over the geometry
- Small fixed space (one MSAA frame buffer)
- But, noise
MSAA helps

- Get $S = 8^P$ samples per pixel, in $P$ passes
- Pixel Shader sets 8-bit mask as pseudo-random function of alpha, $x$, $y$, prim ID.
- Spatial AA with the same samples
- Shading rate stays $1$/fragment
8 spp
16 spp
32 spp
64 spp
512 spp
Stochastic Shadow Map

- Run Stochastic Transparency, rendering only $z$
- Optional: Render with MSAA hardware
  - Each map pixel contains $S$ depth values
- Models the deep shadow function $\text{vis}(z) = \ldots$
  - How much light gets from camera to depth $z$
Stochastic Shadow Map

\[ \text{vis}(z) \approx \frac{\text{count}(z \leq z_i)}{S} \]
Stochastic Shadow Map

- Every pixel looks the same
  - $S$ z-values
  - $z$’s not sorted

- Look-up is just PCF
  - $S$ comparisons per shadow-map pixel

- Color, better reconstruction filter in [CSSM paper]
Colored Stochastic Shadow Maps

Reference Photograph

Williams78 Shadow Map

New Colored Stochastic Shadow Map
Depth-based Stochastic Transparency

- With an extra pass, you can do a lot better
- Shadow map from camera POV
- Final color is (parallel) sum:
  \[ c = \sum \text{vis}(z_i) \alpha_i c_i \]
  - \text{vis}(z) estimated using shadow map
- Very convenient on GPU:
  - For sum, compare fragment \( z \) to shadow map \( z \)'s
  - MSAA z-buffer comparison can do S compares at once
Stochastic Transparency

- Randomized, sub-pixel screen-door transparency
- Distribute samples over an invisible dimension of Swiss cheese
- Correct on average but noisy
- Streaming, fixed space, parallel, MSAA
- Turns transparent stuff into opaque stuff
- ... Which can then be handled by opaque-only algorithms (including stochastic MB + DOF)
Advanced Topics

- Stratified sampling
- Sample test efficiency
- Reconstruction
Bibliography

- “Hardware-Accelerated Stochastic Rasterization on Conventional GPU Architectures,” McGuire, Enderton, Shirley & Luebke, HPG 2010
- “Stochastic Transparency,” Enderton, Sintorn, Shirley & Luebke, TVCG 2011
- “Stratified Sampling for Stochastic Transparency,” Laine & Karras, EGSR 2011
- “Clipless Dual-Space Bounds for Faster Stochastic Rasterization,” Laine, Aila, Karras & Lehtinen, Siggraph 2011
Backup
Deep Shadow Maps

How much light gets from camera to depth $z = \cdots$

"Deep Shadow Maps", Lokovic and Leach (Siggraph 2000)

- How much light gets from camera to depth $z = \cdots$

$$
\text{vis}(z) = \prod_{z_i < z} (1 - \alpha_i)
$$

$= \text{How much light gets from depth } z \text{ to camera}$
Transparency without sorting

Porter-Duff:

\[ c = \alpha_1 c_1 + (1 - \alpha_1)(\alpha_2 c_2 + (1 - \alpha_2)\alpha_3 c_3) \]

Rearrange to find the contribution of each fragment:

\[ c = \sum \text{vis}(z_i)\alpha_i c_i \]

using our friend

\[ \text{vis}(z) = \prod_{z_i < z} (1 - \alpha_i) \]

The sum is order independent, given \( \text{vis}(z) \).

Approximate \( \text{vis}(z) \) \( \to \) approximate sum.