Stochastic Rasterization using Time-Continuous Triangles

Tomas Akenine-Möller    Jacob Munkberg    Jon Hasselgren

Department of Computer Science
Lund University
Sweden
Motivation

• We want:
  • Motion blur
  • Depth of field
  • Glossy reflections

• Stochastic Sampling!

• Seldom or never used for Real-Time rasterization

• We present:
  A new framework for Stochastic Rasterization (SR)
Current HW methods for Motion Blur

- Accumulation Buffering Techniques (ABT)
  - Rendering \( n \) buffers at different points in time
    [Deering et al. 88, Haeberli et al. 90]
- Motion vectors [Shimizu et al. 03]
- Texture space blur only [Loviscash 05]
- Silhouette-based methods
  [Jones 01, Wloka 96]
- Too slow or too inaccurate
Our approach

- Stochastic rasterization of “moving triangles”
- We call them “time-continuous triangles” (TCT)
Interpolation of TCTs

• For simplicity, we use linear interpolation
  • Simple to extend to, e.g., quadratic Bézier curves

• Interpolation is done in homogeneous coordinates
  • After application of projection matrix, but before division by $w$

• Important: same result as interpolating in world space!
High-level overview (1)

• For each TCT:
  1. Find tight bounding volume (BV) around TCT
High-level overview (2)

- For each TCT:
  2. Compute time-dependent edge functions
High Level Overview (3)

3. For each 2x2 pixel quad that overlaps the BV, fetch a set of sample times, $t_i$.

- For each $t_i$:
  - Check whether quad overlaps interpolated triangle.
  - If overlap, interpolate vertex attributes w.r.t $t_i$, and execute pixel shader for current quad.
Example - Chain Link

- Accumulation Buffer Techniques (ABT) using $N$ images, render a complete scene $N$ times
- Our approach renders $N$ samples in a single pass, saving geometry processing and memory bandwidth
Sampling strategy

• **Target:**
  - few samples (4-8)
  - piggyback on much of already existing HW
  - comply with *quad requirement* (for derivatives)
  - Evenly distributed samples in space and time

• **We describe our strategy using RGSS**
  - Used in most GPU:s
  - However, any spatial sampling pattern can be used
Sampling strategy

- Each sample time, \( t_i \), must exist once per pixel in each quad
- Each pixel has \( n \) samples
  \[ s_i = \{x_i, y_i, t_i\} \]
- Jittering
Results

- Bad pixelation due to stamping out "pixels in time" with size of 2x2, instead of optimal 1x1
Improved sampling (1)

• Solution: offset the quads depending on which time-interval, $T_i$, they belong to
Improved sampling (2)

- Increase size of filter kernel
- 4 more samples from immediate pixel neighbors
Comparison

- Sampling quality (4 samples per pixel)
Rasterization of TCTs

• Bad options:
  • Rasterize in screen space
    • TCT: quad surfaces are bilinear patches (not planar)
    • Clipping $\rightarrow$ headache
    • TCTs can move through the near plane
  • 2D BBox in screen space may be too large [Wexler05]
• We propose a two-level algorithm…
Two-level rasterization of TCT

• Compute tight-fit oriented bounding box (OBB) around TCT
Two-level rasterization of TCT

- Rasterize backfaces of OBB using z-fail (similar to robust shadow volume rendering)
Two-level rasterization of TCT

- For fragments inside OBB, check whether samples are inside using time-dependent edge functions.
Time-dependent edge functions

- Simple to derive:
  - \( e(x_i, y_i, t_i) = a(t_i) * x_i + b(t_i) * y_i + c(t_i) \)
  - where, for example, \( a(t_i) = f * t_i^2 + g * t_i + h \)
- \( f, g, h \) only depends on TCT vertices
- Can be computed during triangle setup
- The **standard** edge functions of a triangle for a particular time, \( t_i \), are obtained from the time-dependent edge functions
Example - Textured Wheel

ABT, 4 samples

SR, 4 samples

Jittered, 64 samples (reference)
No blur

Our (4)

ABT (4)
Time-dependent textures

- Motivation: motion blurred geometry without motion blurred shadows.... looks bad!
- Deep shadow maps [Lokovic and Veach 00]
  - Correct only for static shadow receivers, as seen from the light source
- Our approach: let each shadow map pixel have \( n \) time-samples
- Support time-dependent reads...
Time-dependent texture reads

- Strategy: Pick sample from same interval in time
Time-dependent texture reads

- The more time-samples per pixel, the more accurate the result
- We use it for motion blurred shadows and reflections

\[ T_0, T_1, ..., T_{n-1}, T_n \]

Screen space

Texture space
This clip shows an example of time-dependent shadow maps. An alternative between PBT with down-sampled images and our stock D lecturer.

Notice the stroboscopic effect.
Depth of Field

- A highly desired photorealistic effect
- Great for directing the focus of the viewer
- Usually expensive, or poorly approximated
Depth of field

- Standard technique: Many point samples over the lens

- New idea: Use stochastic rasterization in one direction at a time
  - We get “line samples”
Render the scene in \( n \) passes

- Best strategy: long lines, uniform coverage

- We correct for oversampling in the center
Result using one line (4 samples)
Result using 8 lines (only 8x4 samples)
Bandwidth analysis

- Random sampling could potentially reduce performance in a modern GPU
  - Texturing, depth compression, ...
- Texture bandwidth (6kB cache):

Implementation aspects

• We have a partial implementation of the "inner loop" of our algorithm in fragment prog:
  • nvshaderperf: 11 clock cycles on GeForce 7800 with expected fillrate: 873 Mpixels/s
• Too slow for practical use (e.g. Bump, Tex, ...)
• Conclusion: need hardware support for time-dependent edge functions and interpolation
Summary

• New algorithm for pseudo-random sampling of dynamic triangles
  • Need minor hardware modifications

• Enables motion blur, depth-of-field, and planar glossy reflections
  • Substantial geometry bandwidth savings compared to Accumulation Buffering Techniques
  • Efficient alternative compared to ray tracing
Thanks for listening!

http://graphics.cs.lth.se