High-Quality Spatio-Temporal Rendering using Semi-Analytical Visibility

Carl Johan Gribel
Lund University

Rasmus Barringer
Lund University

Tomas Akenine-Möller
Lund University
Intel Corporation
Overview

• Previous work
• Line sampling, approximation of LTD-space
• Visibility Engine
• Ambient Occlusion
• Results
• Future Work
Previous work

- **Stochastic sampling**
  - [Cook et al. 85], [Akenine-Möller et al. 2007], [Fatahalian et al. 2009], [McGuire et al. 2010]

- **Decoupled shading and reuse**
  - [Ragan-Kelley et al. 2011]
  - [Burns et al. 2010]
Previous work

- **2D visibility**
  - [Catmull 1978]
  - [Grant 1985]

- **1D visibility**
  - [Jones and Perry 2000]
Previous work

- Single sample visibility, continuous time
  - [Korein and Badler 1983]
  - [Sung et al. 2002]
  - [Gribel et al. 2010]
Our algorithm

- Treat time and one spatial dimension as continua
Our algorithm

- Treat time and one spatial dimension as continua
Line sample patterns

1H1V
2H2V
RGLS
RGSS

[Jones & Perry 2000]
lyd ⇒ ltd
$lydt \rightarrow ltd$
Approximation

Feature points
Approximation
Approximation
Approximation
Approximation
Approximation
Approximation
Approximation refinement

\[ \frac{\partial l}{\partial t}(t_a) \]

\[ \frac{\partial l}{\partial t}(t_b) \]
Approximation refinement
2 iterations
3 iterations
Visibility Engine

- Resolves visibility between triangles in ltd-space
- Calculates the color contributed to each pixel overlapping the line sample
- Stages:
  1. Binning
  2. Depth sorting
  3. Pixel integration
Stage 1: Binning

- Split $lt$-space into a uniform grid ($m$ by $n$)
- Conservatively rasterize triangles to $lt$-cells
- Following stages: Process a single cell at a time!
Stage 2: Depth sorting

- Clip binned triangles to cell boundaries
- Split polygons against each other (if necessary)
  - Using a BSP-tree
- Sort according to depth
Stage 3: Pixel integration

- Hidden surface removal [Catmull 1978]
- Shading
  - Sample each visible polygon at its barycenter
- Integration
  - Sum area weighted colors
Chess (n = 1)
Chess (12 cores using 24 threads)

Our: 3.6 s

49 samples: 3.8 s
Sponza ($n = 16$)
Sponza ($n = 16$)
Performance with increasing motion

\[ n = 1 \quad \text{(subdivisions along } t) \quad n = 16 \]
Our algorithm can be used to render ambient occlusion as well.

Common approximation of global illumination.
Ambient occlusion

- Line samples instead of rays
- Both static and motion blurred
Ambient occlusion

Static AO

Dynamic AO
Ambient occlusion

Static AO

Dynamic AO
Future work

- Line sampling patterns
- Adaptive splitting of \(lt\)-space
- Depth of field
- Higher dimensional problems
- Rigorous ambient occlusion evaluation
Thanks for listening!
Hidden surface removal [Catmull78]

- Insert polygons into tree in order
- Split against first edge
  - Left part is added to the sub tree of that edge
  - Right part is split against the next edge (or thrown away at last edge)
Performance with increasing motion

![Graph showing performance with increasing motion](image)
Near-z clipping

- Establish coordinate basis for the line sample in view space
- Transform moving triangles into this basis and generate \( ltd \)-triangles
- Clip against \( z = z_{\text{near}} \)
- Project into viewport space and map along line
Depth approximation error

GT

0 iterations

3 iterations
Refinement iterations

0
1
2
3

GT