A Hybrid Approach

One more shadow algorithm which deserves mention is McCool's clever idea shadow volume reconstruction from depth maps [McCool 2000].

This algorithm is a hybrid of the shadow map and shadow volume algorithms and does not require a polygonal representation of the shadow volumes.

Not Another Talk on Shadows?!

Main ideas:
- combination of shadow maps + shadow volumes
- computation masks

Fillrate Problem

Lots and lots of fillrate!
- rasterization
- stencil updates

Why?
- polygons have large screen area
- polygons overlap
Fillrate Problem

Lots and lots of fillrate!
- rasterization
- stencil updates

Why?
- polygons have large screen area
- polygons overlap

But is this really a problem?

But Is This Really A Problem?

Case study: Doom 3 engine (id Software)
- bump mapping
- per-pixel surface shading
- dynamic and projected lights
- atmospheric effects
- particle effects
- shadow volumes

But Is This Really A Problem?

Case study: Doom 3 engine (id Software)
- bump mapping
- per-pixel surface shading
- dynamic and projected lights
- atmospheric effects
- particle effects
- shadow volumes

“Shadowing accounts for about half of the game’s rendering time.”
— John Carmack

Two Observations

Two Observations (shadow maps)

Shadow-map aliasing is ugly
But — only noticeable at shadow silhouettes

Two Observations (shadow volumes)

Shadow volumes are accurate everywhere
But — accuracy is only needed at silhouettes
Hybrid Approach

Decompose the problem:
- use shadow volumes at silhouettes
- use shadow maps everywhere else

Algorithm

1. create a shadow map
2. find silhouette pixels
3. apply shadow volumes only at silhouette pixels
4. apply shadow maps everywhere else
Algorithm Details

Questions:
- how to find silhouette pixels?
- how to rasterize only silhouette pixels?

Find Silhouette Pixels

Silhouette pixels

Look for depth discontinuities

Use nearest 2x2 depth samples of the shadow map

Find Silhouette Pixels (example)

shadow map query point

Check results:
- 2 in shadow
- 2 visible

Disagreement!
- silhouette pixel

Restricted Rasterization

Use a mask to limit rasterization:
- tag silhouette pixels in framebuffer
- mask off all other pixels

Computation Mask

We need a computation mask
- user-specified mask
- hardware early pixel rejection
- reduces rasterization, shading, memory bandwidth

Hardware Support

Current hardware doesn’t have computation mask
- but — hardware already has early z culling!
- minimal changes needed for native mask support
- our implementation uses a simulated mask
Results

- 2.6 GHz Pentium 4
- NVIDIA GeForce 6 (NV40) + crazy blue power supply

Hybrid Algorithm Example

- Aliased shadow of a ball
- Standard shadow map

- Blue and red regions handled by shadow maps
- Visualization

- Blue and red regions handled by shadow maps
  - Black and green regions handled by shadow volumes
  - Visualization

Hybrid Algorithm Example

- Standard shadow map
- Hybrid algorithm

Test Scenes
Artifacts

Low-resolution shadow map → discretization errors
Misclassified silhouette pixels → missing features
Difficult cases: fine geometry

Example of Missing Features

<table>
<thead>
<tr>
<th>Result</th>
<th>Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>256x256</td>
<td><img src="image" alt="Result" /></td>
</tr>
<tr>
<td>1024x1024</td>
<td><img src="image" alt="Visualization" /></td>
</tr>
</tbody>
</table>

Discussion

Algorithm designed to help fillrate-bound applications:
- requires an extra rendering pass
- 30% to 100% speedup in our test scenes
- performance depends a lot on culling hardware

More details in the paper and web page …
- tradeoff analysis
- comparison to related work
- implementation details
- more performance and image comparisons

Summary

Hybrid shadow algorithm

Screen-space decomposition:
- most pixels use fast (but inexact) algorithm
- a few pixels use accurate (but expensive) algorithm
Computation Masks

Why?
- pixels are not created equal
- programmer marks “interesting” pixels
- fast reject all other pixels
- not just for shadows!
- useful in general for multipass algorithms
- hardware is (mostly) already there

Acknowledgments

Nick Triantos and Mark Kilgard (NVIDIA)
Jan Kautz and Addy Ngan (MIT)
Timo Aila
ASEE NDSEG Fellowship