SIGGRAPH 2011

Road to Real-Time Order-Independent Transparency

Marco Salvi

Talk Outline

- Motivation
- Compositing Equation
  - Recursive Solvers
  - Visibility Based Solvers
- State of the Art and Future Work
- Q&A

Why Compositing?

“Reliance on a single program for rendering an entire scene is a poor strategy..”

“Separating the image into elements which can be independently rendered saves enormous time..”

“Compositing Digital Images”, @SIGGRAPH 1984
Thomas Porter & Tom Duff

Motivation

*"Order-**dependent** transparency has always been a big limitation for content creators & developers*

- *Restrictive art pipeline: no glass houses*
- *Even windows on cars & buildings can be painful*
- *Restrictive interaction between objects"*

*"Order-**independent** transparency is must going forward*

- *Big challenge! Gradual process"*

"Five Major Challenges in Interactive Rendering", @SIGGRAPH 2010
Johan Andersson – DICE/EA

Motivation

Alpha-Blending

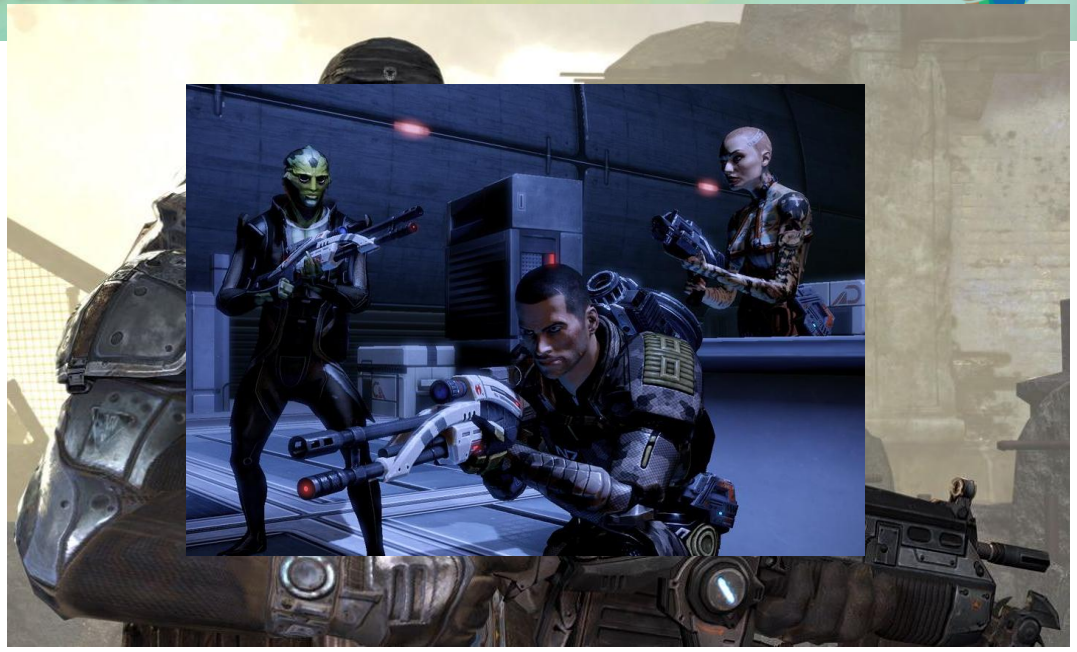
Motivation

Alpha-Blending

Order-Independent Transparency

Motivation

Motivation



Motivati

Motivation

Alpha-Test

Order-Independent Transparency

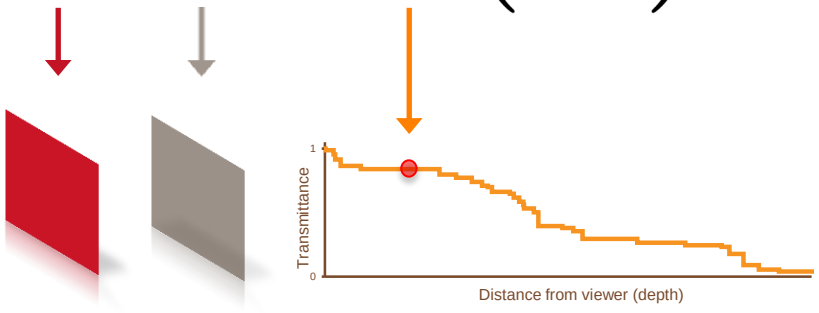
Motivation

Alpha-Test

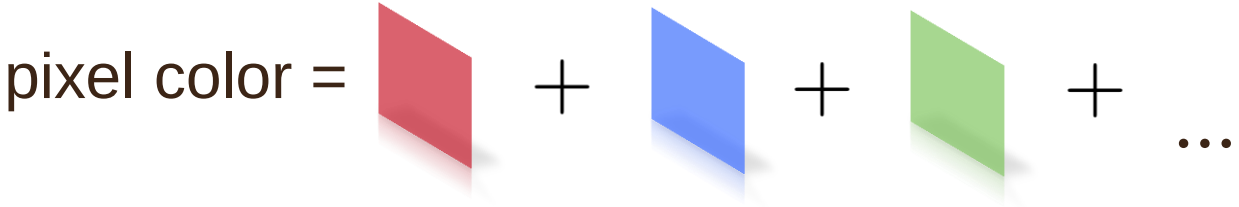
Order-Independent Transparency

Compositing Equation

$$\text{pixel color} = \sum_{i=0}^{n} c_i \, \alpha_i \, vis(z_i)$$



Compositing Equation



Ideal Real-Time OIT Method

- High image quality
  - Accurately evaluate compositing equation
  - No major spatial and temporal artifacts
- High and stable performance
  - Low variability. Performance mostly independent from fragments ordering
- Bounded memory usage
  - No variable length data structures

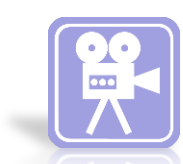
OIT Algorithms Classification

- Solve compositing equation recursively
  - Composite fragment with result of previous composite operation
- Solve compositing equation by computing and evaluating (compressed) visibility functions

Recursive Solvers

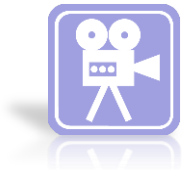
- Alpha Blending / Compositing
- Depth Peeling
- A-buffer
- Z^3 algorithm

Alpha Compositing [Porter and Duff 1984]

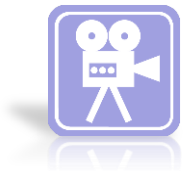


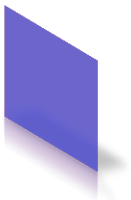
Alpha Compositing [Porter and Duff 1984]

Alpha Compositing [Porter and Duff 1984]

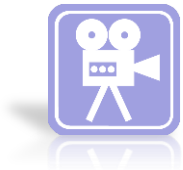


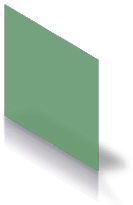
Alpha Compositing [Porter and Duff 1984]





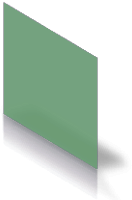
Alpha Compositing [Porter and Duff 1984]





Alpha Compositing [Porter and Duff 1984]

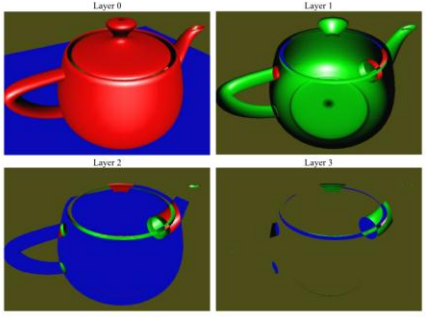
- Fast and stable/predictable performance
- No additional storage required
- But order-dependent..



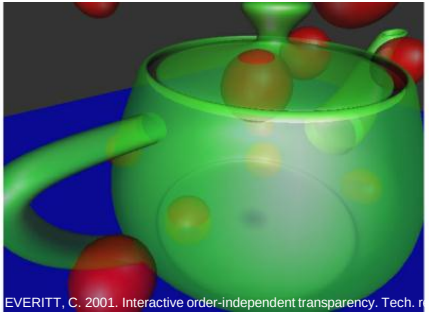
$$C_0 = \alpha_0 c_0$$
$$C_n = \alpha_n c_n + (1 - \alpha_n) C_{n-1}$$

Depth Peeling [Everitt 2001][Liu et al. 2009]

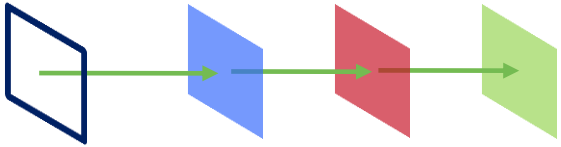
- Peel layers and composite in depth-sorted order



- Number of passes proportional to max image depth complexity

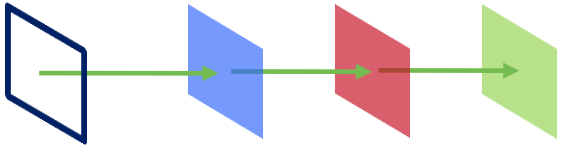


1) Render fragments color and depth in per-pixel lists

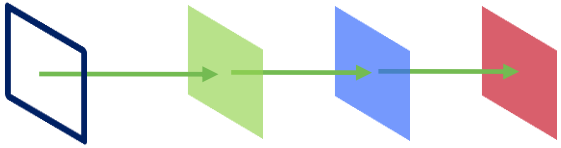


A-buffer [Carpenter 1984] [Yang et al. 2009]

1) Render fragments color and depth in per-pixel lists

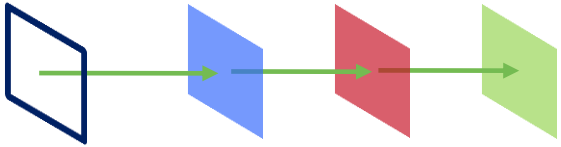


2) Per-pixel sort and composite fragments



A-buffer [Carpenter 1984] [Yang et al. 2009]

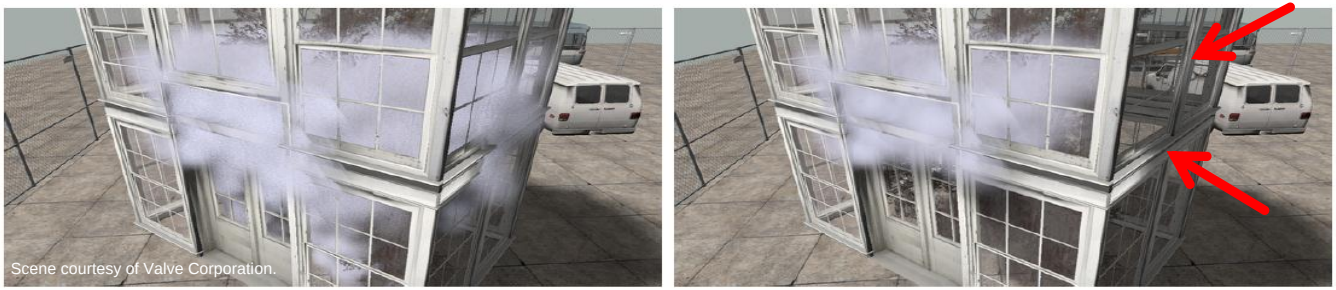
1) Render fragments color and depth in per-pixel lists



2) Per-pixel sort and composite fragments

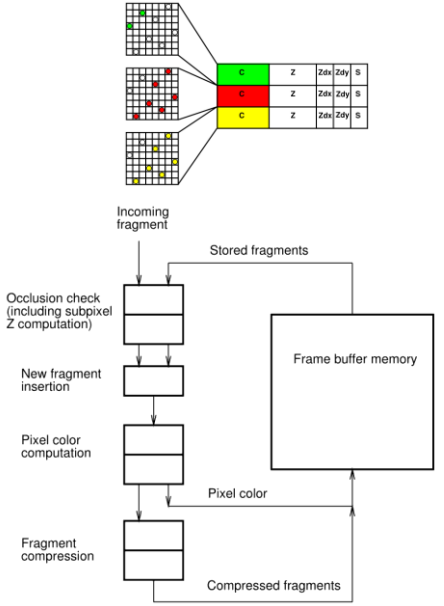
A-buffer Limitations

- Poor & unstable performance, tipically memory bandwidth limited
- Unbounded memory requirements



The Z³ algorithm [Jouppi and Chang 1999]

- Bounded A-buffer
  - Up to N fragments per pixel (sorted)
- Merge fragments to keep pixel memory footprint constant
  - Distance & coverage based compression metric



JOUPPI, N. P., AND CHANG, C.-F. 1999.
Z3: an economical hard- ware technique for high-quality antialiasing and transparency.

Visibility Function

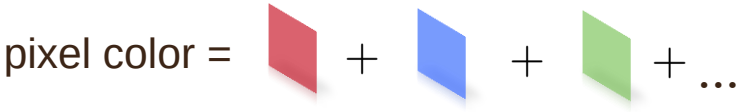
- Models light absorption
- Product of step functions (thin blockers)

Visibility Based Solvers

- Fragment Parallel Compositing
- Occupancy Maps
- Opacity Shadow Methods
- Stochastic Transparency
- Adaptive Transparency

Frag. Parallel Compositing [Patney et al. 2010]

- Compute visibility via parallel segmented scan
  - Load-balance across irregular number of fragments per pixel
- Evaluate compositing equation via parallel reduction



Occupancy Maps [Sintorn et al. 2009]

- Assume all transp. geom. has the same alpha
  - Not applicable to many objects (smoke, foliage, etc.)
- Bit-field represents step functions of same height and spaced at regular depth
  - A slab map allows to re-modulate steps height on a given depth region
- Use visibility representation for OIT & shadows

Opacity Mapping Methods

- Replace visibility with opacity: $-\frac{d}{dz_i}\ln\left(vis(z_i)\right)$
  - Less well-behaved function (no monotonicity, no [0,1] bounds)
- Basis function
  - Piece-wise constant intervals: Opacity Shadow Maps [Kim et al. 2001]
    - Warp OSMs first layer: Deep Opacity Shadow Maps [Yuksel et al. 2008]
  - Trigonometric: Fourier Opacity Mapping [Jansen et al. 2010]

Stochastic Transparency [Enderton et al. 2010]

- Fixed length visibility representation
  - Regular steps at irregular locations
  - A2C & z-test on MSAA samples
    - Compression by removing random step
    - Efficient use GPU fixed function HW
- Fast but “noisy” with complex objs
  - Can require many samples per-pixel

ENDERTON, E., SINTORN, E., SHIRLEY, P., AND LUEBKE, D. 2010. Stochastic transparency

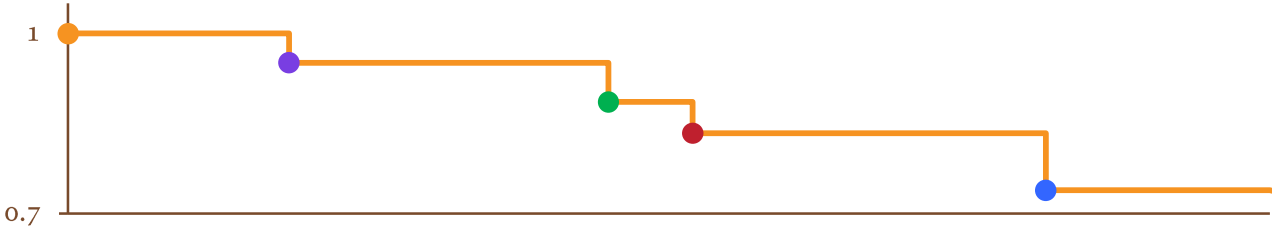
Adaptive Transparency [Salvi et al. 2011]

- Derived from volumetric shadow method [Salvi et al. 2010]
  - Optimized for thin objs and OIT
- Store up to N steps per pixel
  - Arbitrary location and height
  - Area based compression metric

Salvi M., Montgomery J., Lefohn A. 2001. Adaptive Transparency

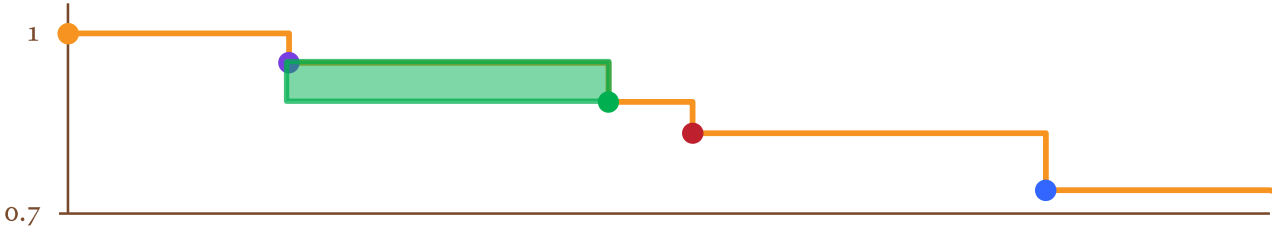
Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



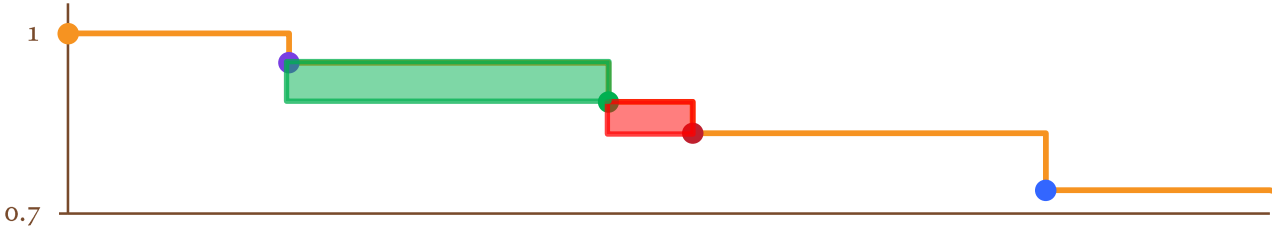
Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



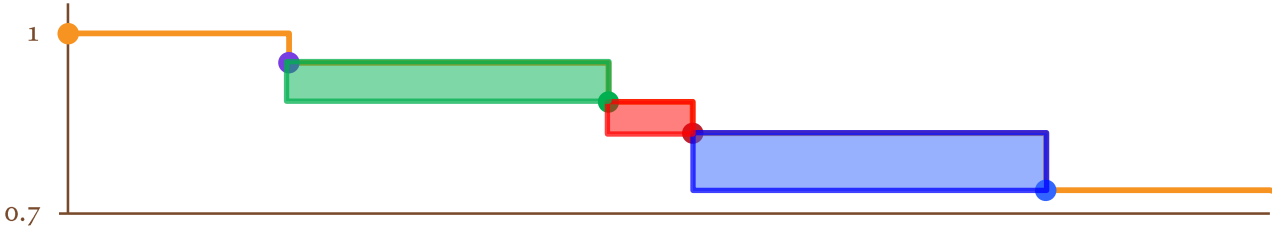
Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



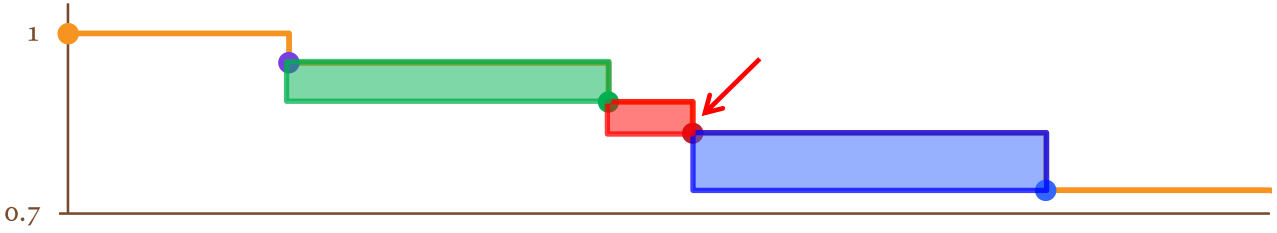
Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



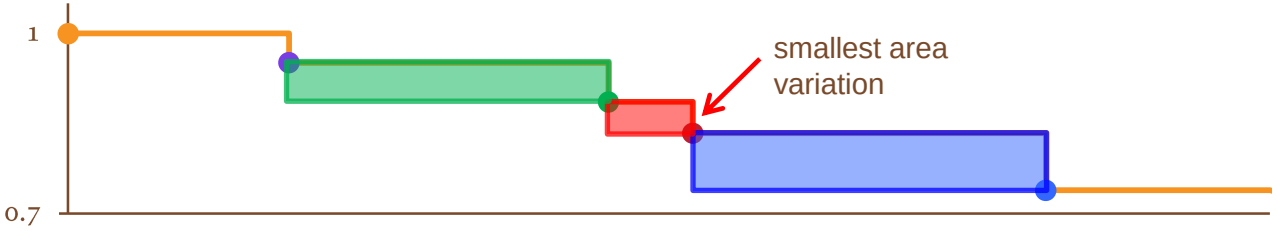
Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



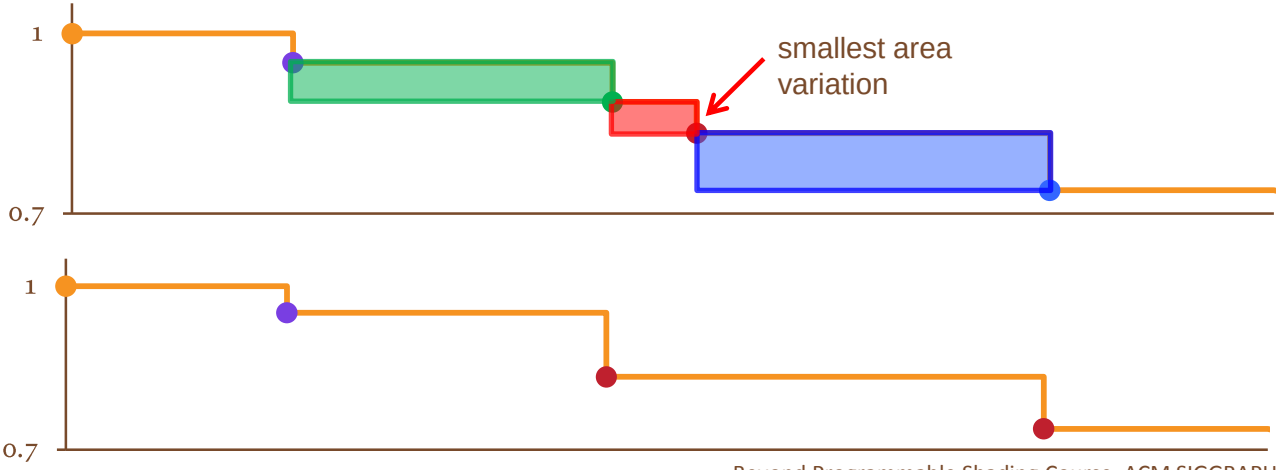
Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



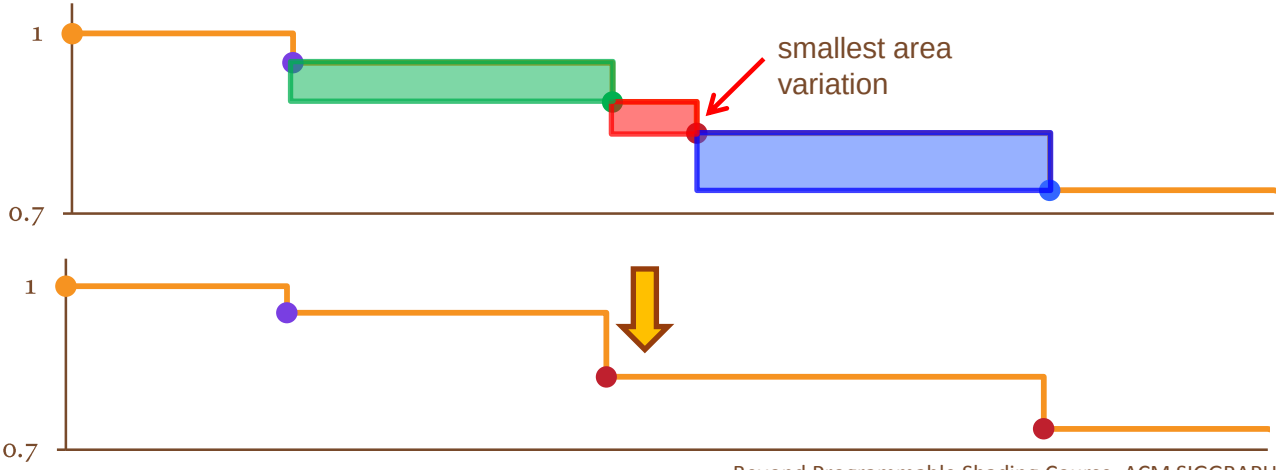
Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



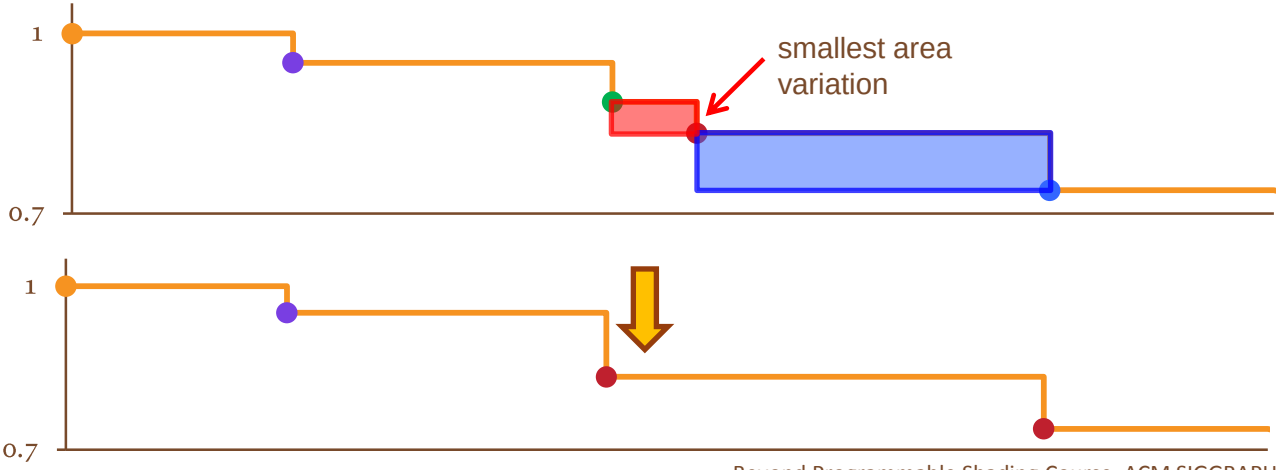
Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



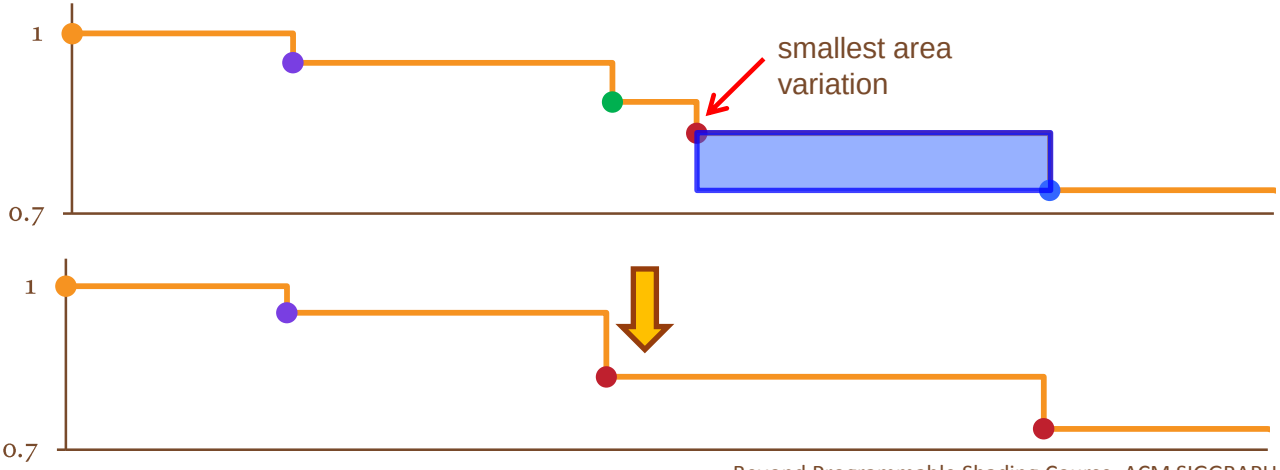
Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



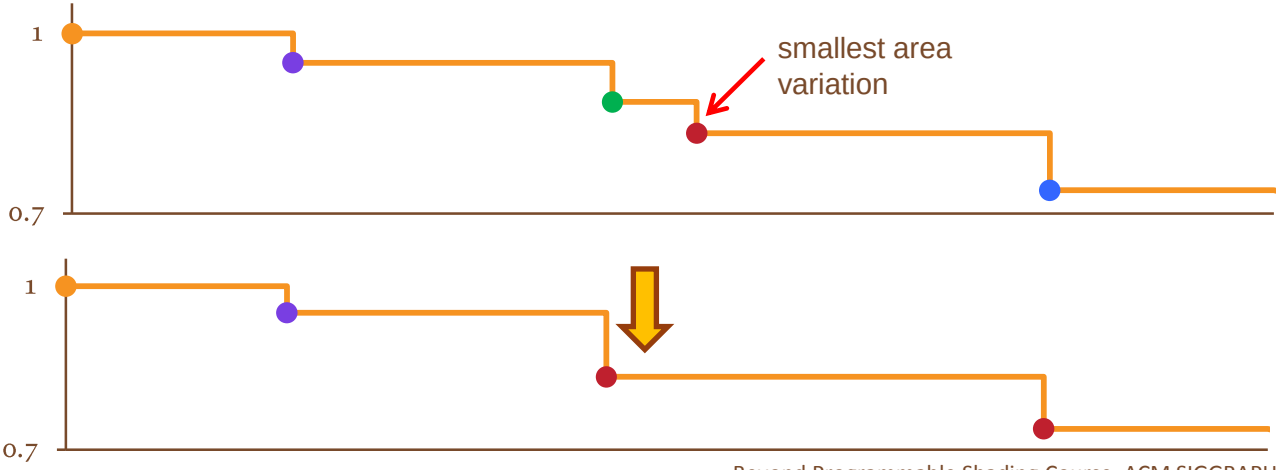
Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



Adaptive Transparency [Salvi et al. 2011]

- To compress visibility AT removes the node that generates the smallest area variation



Adaptive Transparency [Salvi et al. 2011]

SMOKE scene
21 ms - 10.6 MFragment
Max fragment per pixel: 312
30x faster than A-buffer
2.5x faster than Stoc. Transp.

HAIR scene
48 ms - 15.0 MFragment
Max fragment per pixel: 663
40x faster than A-buffer
2x faster than Stoc. Transp.

FOREST scene
8 ms - 6.0 MFragment
Max fragment per pixel: 45
7x faster than A-buffer
2x faster than Stoc. Transp.

Adaptive Transparency [Salvi et al. 2011]

- Higher quality than other “lossy” OIT methods
  - Works on any transparent object type (foliage, smoke, glass, etc.)
- Designed to run in fixed memory
  - Prototype uses variable memory data struct due to 3D APIs limitations
- High and scalable performance
  - Easy to trade-off IQ for performance and storage by tuning per-pixel step/node count

What's Next?

- AT & ST are good examples of new algorithms close to ideal OIT method for real-time apps
  - ST main limitation is noise / lower quality per vis. rep. storage
  - AT main limitation is current variable memory implementation
- We are rapidly converging towards entirely practical and robust OIT methods for real-time rendering!

Open Problems

- Deferred shading and OIT methods
  - Many methods are trivial to extend to DSers..
  - ..but awfully inefficient..
- Some interesting work left to do in this area

Acknowledgements

- Special thanks to Jefferson Montgomery and Aaron Lefohn

- Craig Kolb, Matt Pharr, Charles Lingle and Elliot Garbus for supporting this work.

- Jason Mitchell and Wade Schinn at Valve Software for the assets from Left-for-Dead-2

- Cem Yuksel (Cornell University) for the hair model

Q&A

"Adaptive Transparency" pre-print: http://intel.ly/at_hpg11

"Adaptive Transparency" source code: http://intel.ly/aoit_gdc

twitter: @marcosalvi

e-mail: marco.salvi@intel.com

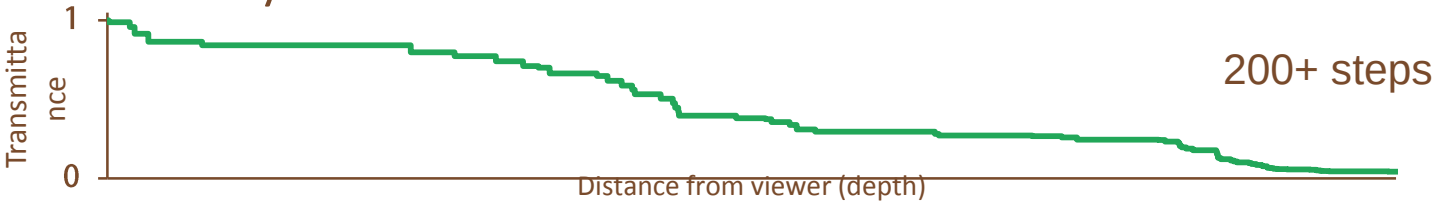
blog: http://pixelstoomany.wordpress.com

Backup

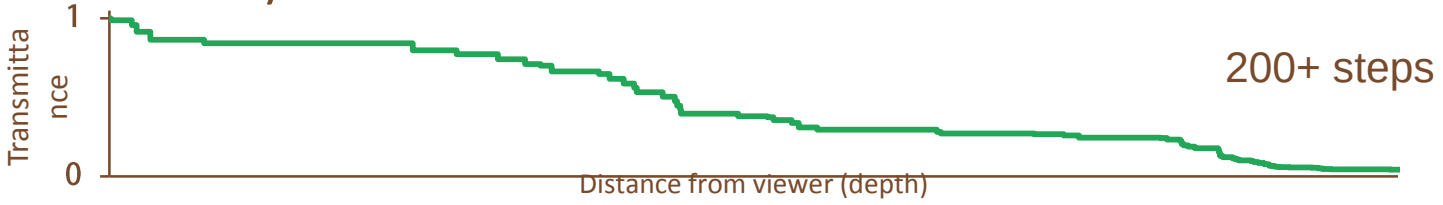
Bibliography

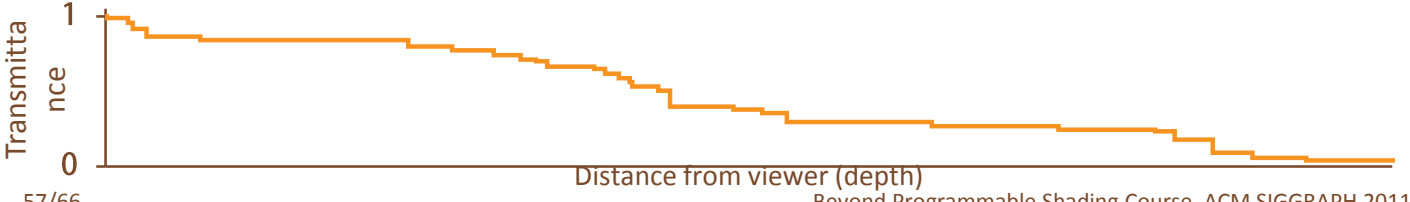
- PORTER, T., AND DUFF, T. 1984. *"Compositing digital images."* SIGGRAPH Comput. Graph. 18, 3, 253–259.
- CARPENTER, L. 1984. "*The A -buffer, an antialiased hidden surface method*." SIGGRAPH Comput. Graph. 18, 3, 103–108.
- SINTORN, E., AND ASSARSON, U. 2009. "*Hair self shadowing and transparency depth ordering using occupancy maps.*" In I3D '09: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, 67–74
- PATNEY, A., TZENG, S., AND OWENS, J. D. 2010. *"Fragment parallel composite and filter."* Computer Graphics Forum (Proceedings of EGSR 2010) 29, 4 (June), 1251–1258.
- YANG, J., HENSLEY, J., GR¨U 547 N, H., AND THIBIEROZ, N. 2010. *"Real-time concurrent linked list construction on the gpu."* Computer Graphics Forum (Proceedings of EGSR 2010) 29, 4 (June),
- SALVI, M., VIDIMCE, K., LAURITZEN, A., AND LEFOHN, A. 2010. *"Adaptive volumetric shadow maps."* Computer Graphics Forum (Proceedings of EGSR 2010) 29, 4 (June), 1289–1296.

- **Idea:** Save bandwidth by working with an approximate visibility function

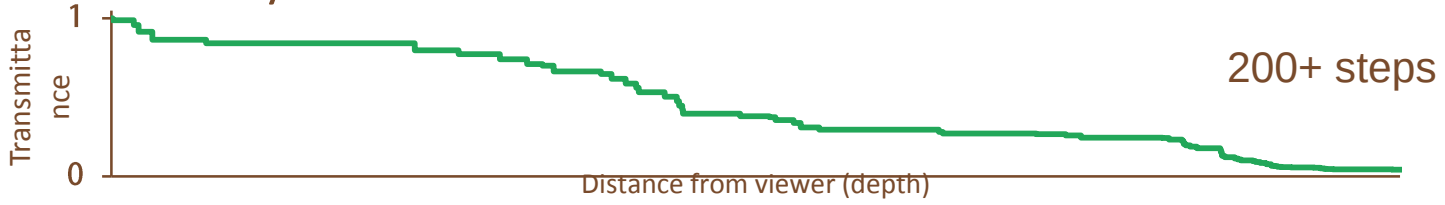


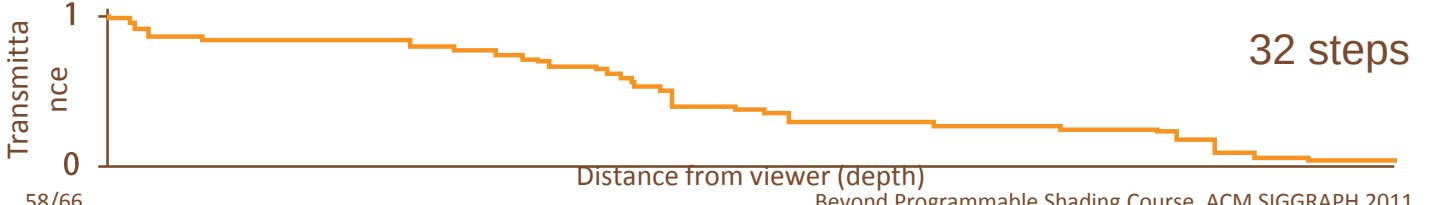
- **Idea:** Save bandwidth by working with an approximate visibility function



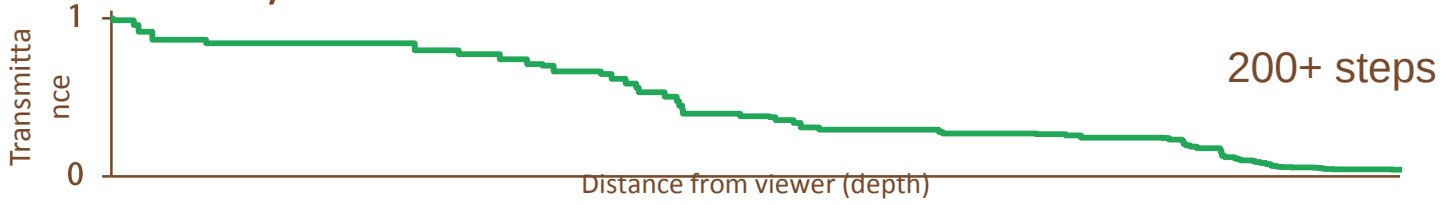


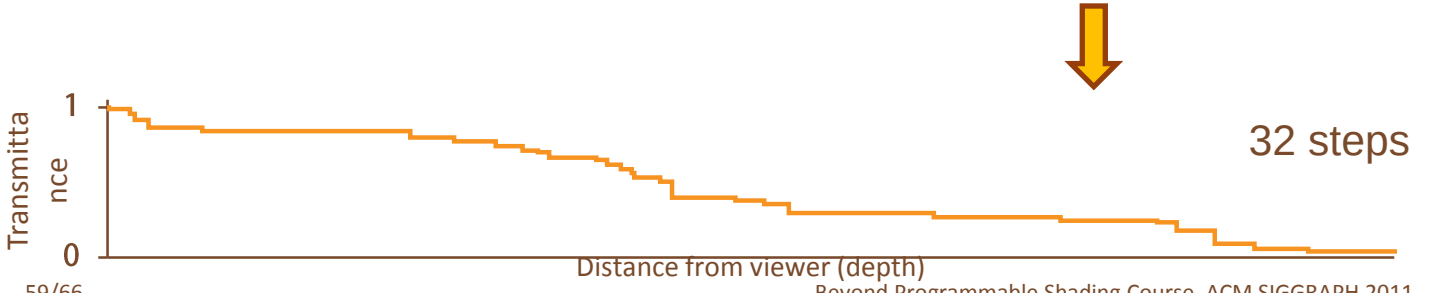
- **Idea:** Save bandwidth by working with an approximate visibility function



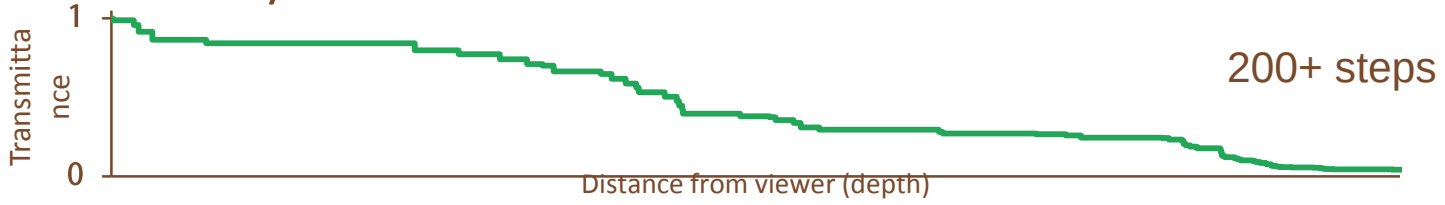


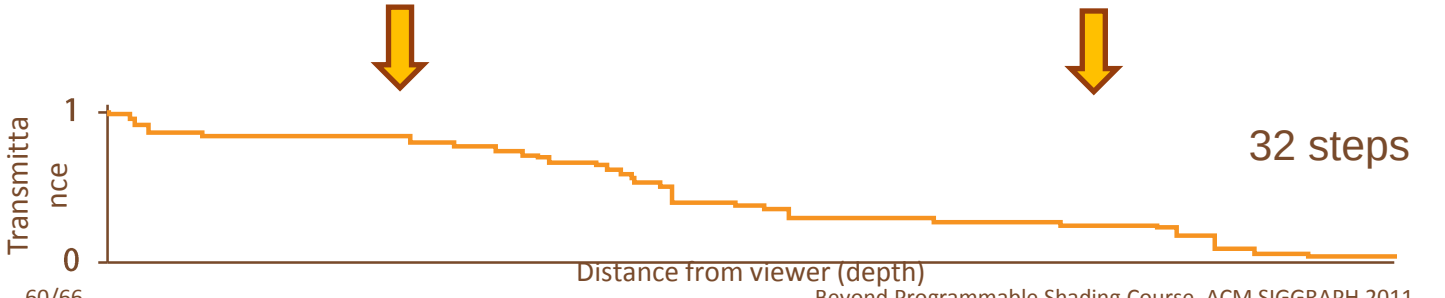
- **Idea:** Save bandwidth by working with an approximate visibility function





- **Idea:** Save bandwidth by working with an approximate visibility function





AT GPU Implementation

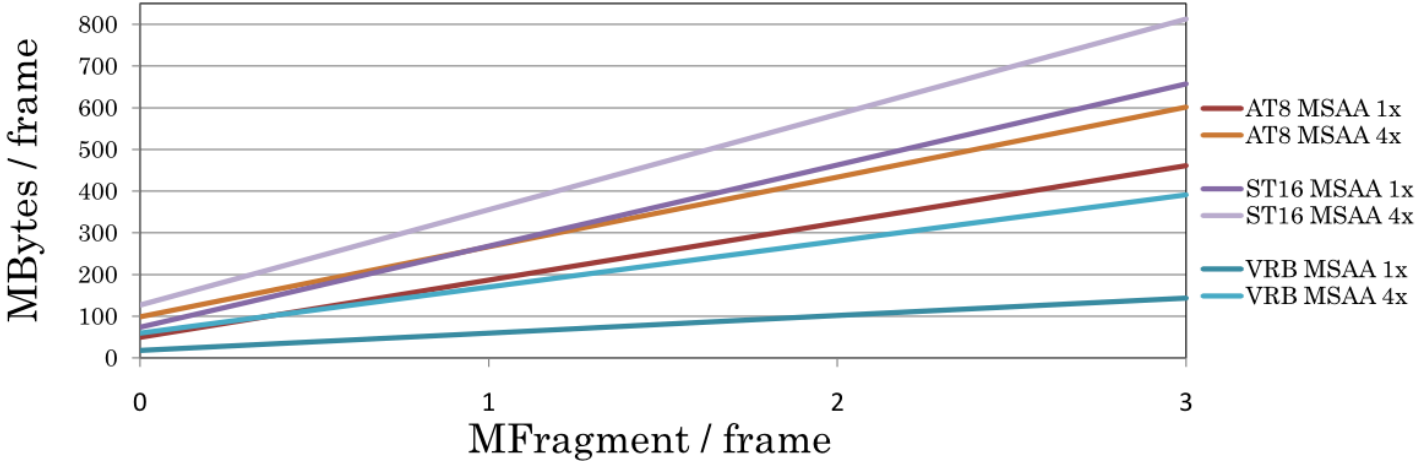
- Store visibility in the frame buffer?
  - Data update cannot be mapped to DX11 blend modes
  - No RMW operations on the frame buffer
- Store visibility in a Read/Write buffer (UAV)?
  - Cause data races

AT Proof-of-Concept Implementation

1) Render transparent fragments to per-pixel lists
   - Same as A-buffer implementation

2) For each pixel: build an approximate visibility function and use it to composite all transparent fragments
   - Full-screen pass guarantees atomicity

Bandwidth Requirements



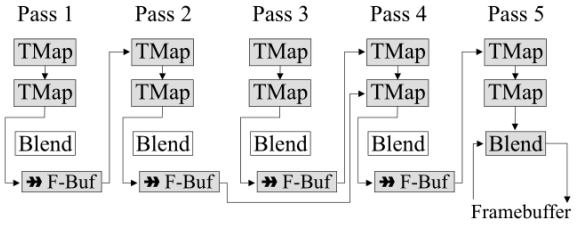
AT Future Work

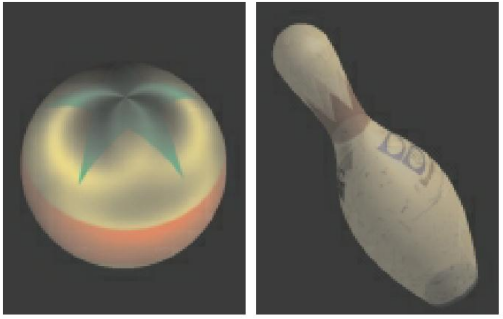
- Investigate bounded memory implementations
  - Per-pixel locks? New frame-buffer format?
- Better visibility data compression
  - Reduce MSAA impact on memory requirements

Scene courtesy of Valve Corporation.

F-Buffer [Mark and Proudfoot 2001]

- Capture fragments in FIFO
  - Enable efficient multi-pass shading
- Require global sort
  - A-buffer requires local sort
- FIFO can overflow
  - Similar to A-buffer





MARK, W. R., AND PROUDFOOT, K. 2001.
The F-buffer: A rasterization-order FIFO buffer for multi-pass rendering