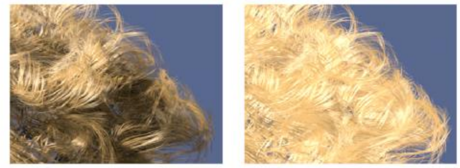
Adaptive Volumetric Shadow Maps

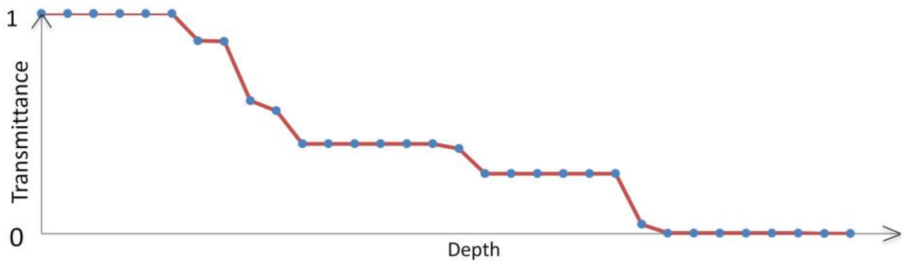
Marco Salvi, Kiril Vidimce, Andrew Lauritzen, Aaron Lefohn
Intel Corporation

Problem Background

- Realistic lighting of volumetric media
  - Hair, smoke, fog, etc..
- Compute visibility curve
  - **Transmittance**: Fraction of light that passes through a material



LOKOVIC T., VEACH E. "Deep shadow maps", SIGGRAPH 2000

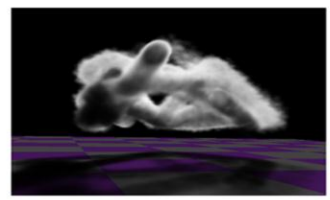


-Transmittance maps to the [0,1] interval.
- In this work we don't take in consideration light scattering, transmittance can only change (decrease) via absorption. Emissive materials are technically possible as our work doesn't make any assumption on the monotonicity of the visibility curve.

Previous Methods

LOKOVIC T., VEACH E. "Deep shadow maps", SIGGRAPH 2000

- Deep Shadow Maps [Lokovic et al. 2000]
  - Capture visibility curve & compress
  - Used defined error threshold
    - Variable number of nodes
  - Designed for off-line rendering, easy to implement on DX11 but slow
- Opacity shadow Maps [Kim et al. 2001]
  - Sample visibility at regular intervals
  - Numerous variants optimized to handle special case (i.e. hair)
  - Depth range dependent
- Fourier Opacity Mapping [Jansen et al. 2010]
  - Visibility function expansion via trigonometric series
  - Converge slowly, especially around sharp features
  - Ringing
  - Depth range dependent

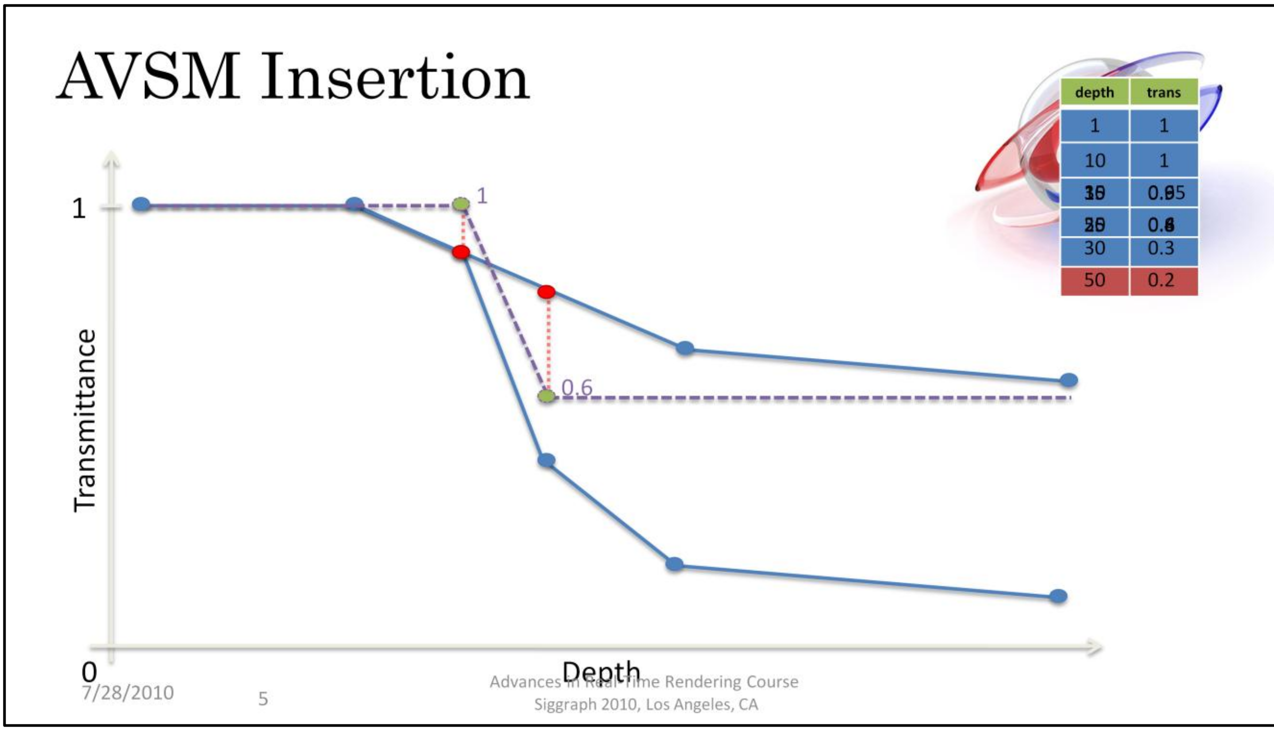


JANSEN J., BAVOIL L. Fourier opacity mapping. I3D 2010

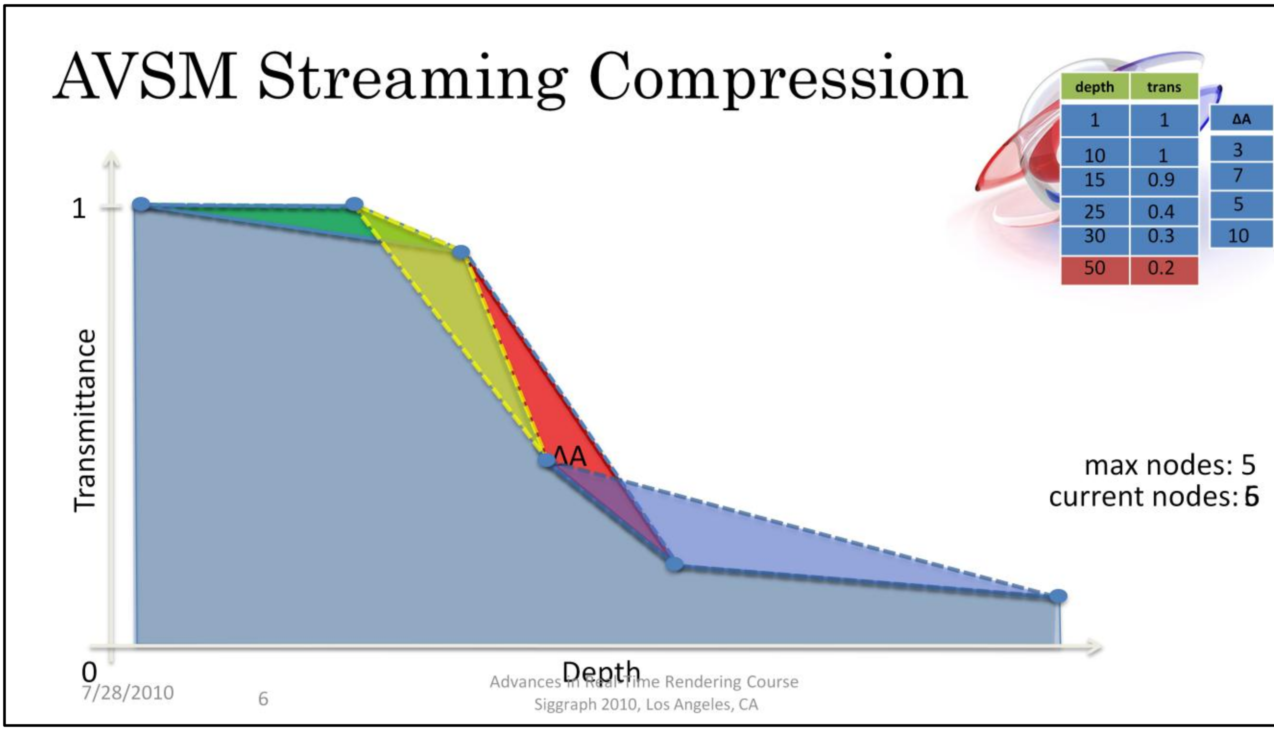
AVSM

- Streaming simplification algorithm
- Generates an adaptive volumetric light attenuation function using a small fixed memory footprint

- Fixed number of nodes. Variable and unbounded error
- Easy to use method that does not make any assumption about light blockers type and/or their spatial distribution



- An individual AVSM texel encodes N nodes, each node is represented by a depth and a transmittance value. Nodes are always stored as sorted (front-to-back) sequence.
- The AVSM is cleared by initializing all nodes within a texel to the same value. We set depth to the far plane and transmittance to 1 (no occlusion)
- Incoming light blockers are represented by light-view vector aligned segments. A segment is defined by two points (entry and exit points) and transmittance at the exit point (transmittance at the entry point is implicitly set to 1).
- We assume that the space between the entry and the exit points is filled by an uniformly dense media. This would typically generate transmittance curves shaped as piece-wise exponential curves, we use lines instead to simplify the problem (not much visual difference in most cases)



- The first and last node are never compressed/removed as the provide very important visual cues. The last node is extremely important as it encodes information on the shadow that is cast on any receiver which is located behind the volumetric blockers. For instance the shadow cast by some cigarette smoke over a table will always be correct (no compression artifacts)
- After a node is removed we don’t update the remaining nodes location in order to better fit the original curve (ala deep shadow maps). In fact updating nodes location over dozen of insertion-compression iterations can generate some unpredictable results as the nodes perform random walks over the compression plane.

Implementation Details (DX11)

- Algorithm designed for streaming simplification but..
  - In-flight fragments that map to the same pixel cause data races
    - Atomic RMW operations on structures not currently available from pixel shaders
- A tale of two implementations:
  - Compute shader based, slower but fixed memory
    - Software pipeline prototype for particles has received little optimization work
    - ~2x slower than variable memory implementation
  - Pixel shader based, faster but variable memory

- The variable memory (pixel shader based) implementation avoid data races by reading back per pixel linked lists and building an AVSM via a full screen pass. A full screen pass avoids data races by guaranteeing that only one fragment that maps to a specific pixels in shaded in flight at any given time (no overlapping primitives)

Variable Memory Implementation

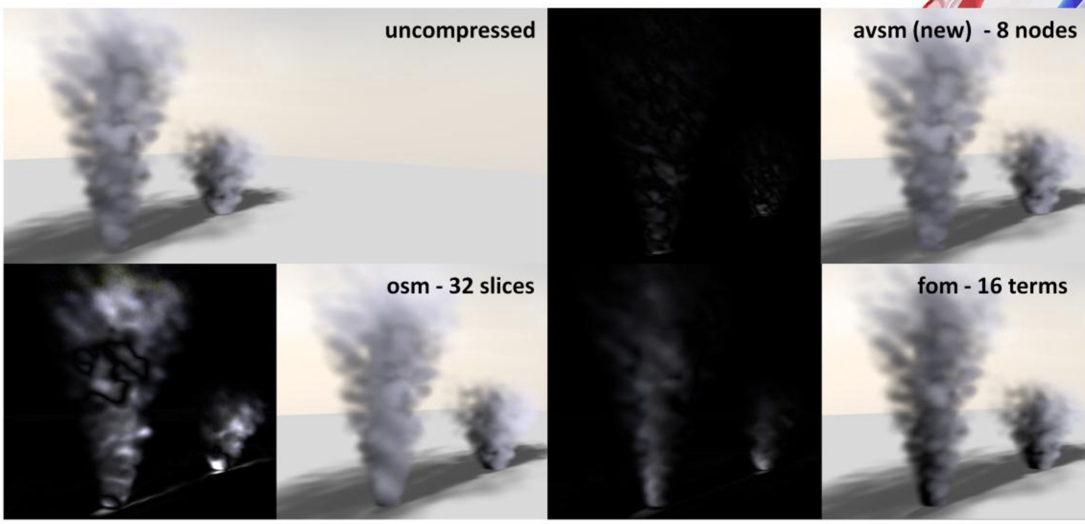
- Light blockers AVSM insertion in two steps
  1. Render blockers in light space and capture them in a per pixel linked list [Yang et al. 2010]
  2. Traverse per pixel lists and build AVSM entirely on-chip
     - Optionally sort blockers to remove temporal artifacts due to out of order fragments shading

- AVSM sampling and filtering
  - Evaluate transmittance at receiver depth via linear (or exponential) interpolation
  - Filtering implemented in software (bi-linear, tri-linear, Gaussian, etc..)

-AVSM sampling is implemented via a 2-level search performed over a sorted (front-to-back) array of nodes. The first step is a linear search performed with a 4-node stride, while the second level search within 4 nodes. Since we always work with a pre-determined number of nodes it is possible to generate some very efficient search code that doesn't employ any dynamic control flow statement or dynamic access to arrays of temporary values.
- It is possible to generate mip-maps for an AVSM texture, which are mostly useful to improve data locality and to improve IQ for volumetric shadows generated by sharp and thin light blockers.

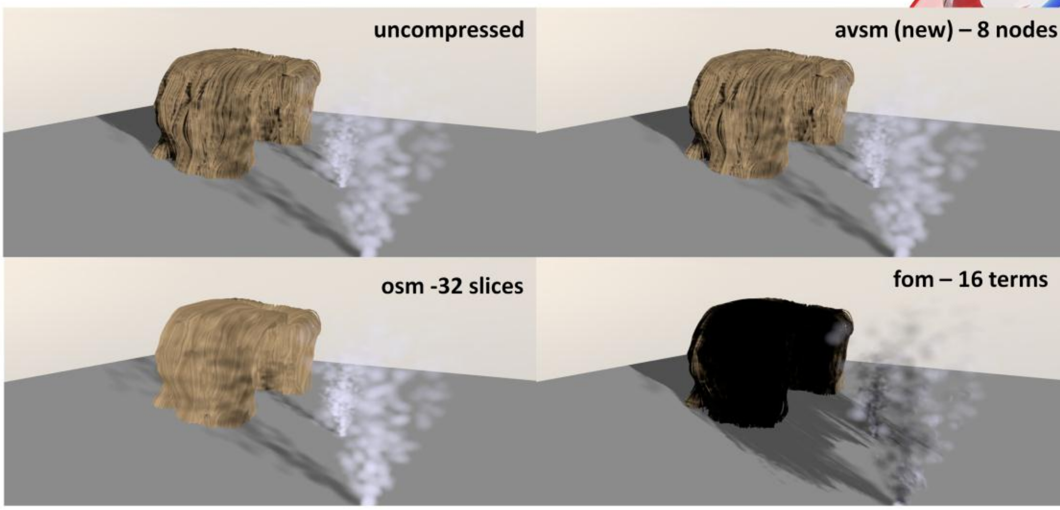
Results

(1/3)

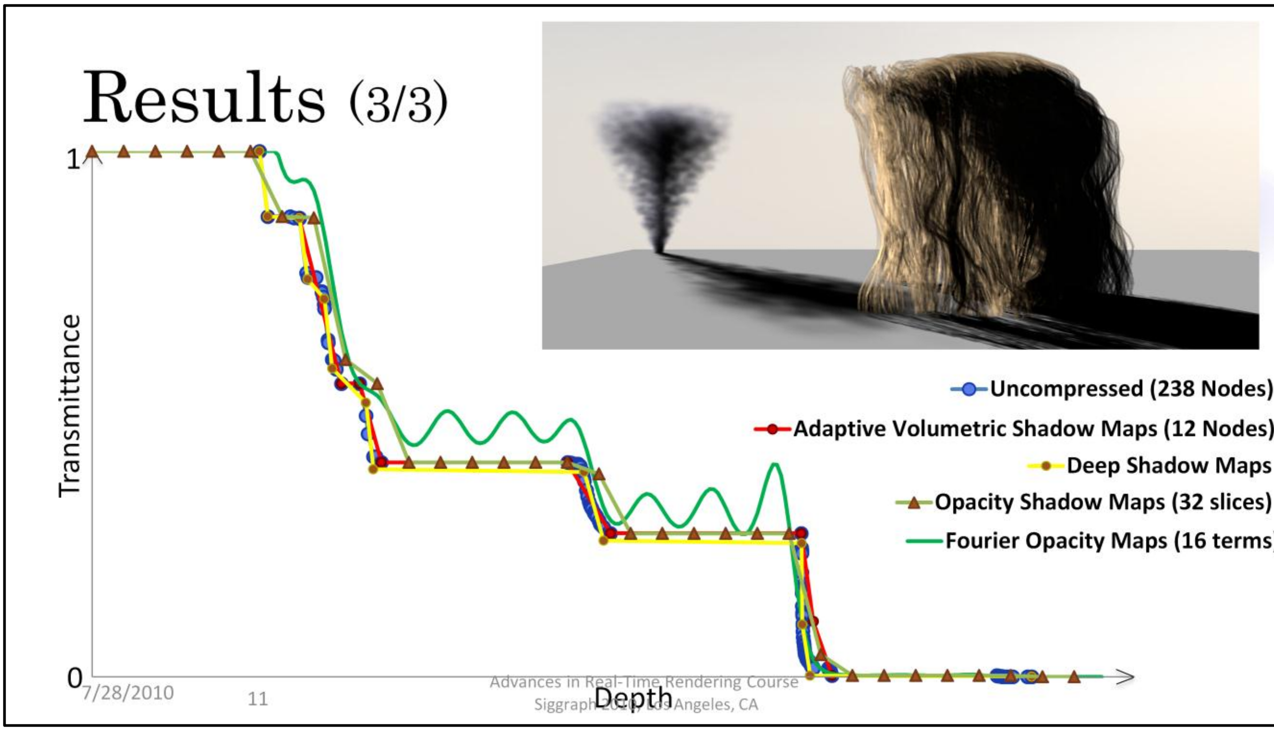


- Diff images have been enhanced by 4X

Results (2/3)



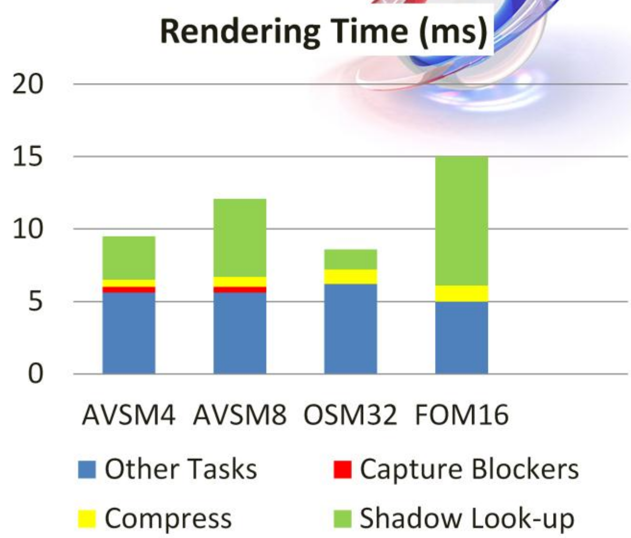
- FOM has significant issues with very sharp transmittance function transitions generated by hair-like geometry



- In this particular case AVSM generates slightly better results than Deep Shadow Maps. The latter performs a local analysis of the visibility curve. AVSM, while working on an incomplete data set, always try solve a global (within a texel) optimization problem.

AVSM Performance

- Competitive performance
- Higher image quality
- Shadow look-up dominates
  - Often < 30% of AVSM related rendering time is spent in the insertion code
- DSM is 20x-40x slower than AVSM



- The per-pixel-list and AVSM rendering (compress) time is often negligible compared to the AVSM sampling/filtering time.

Conclusions

- The Good:
  - Higher image quality via adaptive sampling
    - Avoid common pitfalls of methods based on regular sampling or series expansion of the visibility function
  - Robust and easy to use
    - Doesn't require any a priori knowledge of light blockers type and spatial distribution
    - Easy to trade-off image quality for speed and storage
- The Bad:
  - A fast fixed-memory implementation requires graphics hardware to add support for read-modify-write operations on the frame-buffer

- Pixel shader based implementation is fast but uses unbounded memory

What's Next?

- Improve AVSM filtering performance
  - Find bottleneck(s)
    - Not an external memory bandwidth issue
  - Re-encode AVSM data?
- Fixed memory implementation with pixel shaders
  - Avoid RMW hazards (per pixel mutex?)
- Lossy Order Independent Transparency via AVSM streaming compression

Acknowledgements

- Jason Mitchell and Wade Schin (Valve)
- Natasha Tatarchuk and Hao Chen (Bungie)
- Johan Andersson (DICE)
- Matt Pharr, Craig Kolb and the rest of the Advanced Rendering Technology team at Intel
- Nico Galoppo, Greg Johnson, Doug McNabb, Jeffery Williams and Mike Burrows from Intel
- Hair model courtesy of Cem Yuksel

Questions?

- Paper*:
  - Salvi M., Vidimce K., Lauritzen A., Lefohn A.,
    Adaptive Volumetric Shadow Maps
    Computer Graphics Forum - Volume 29, Number 4, pp. 1289-1296
    http://www.eg.org/EG/DL/CGF/volume29/issue4
- Source code and binaries:
  - http://visual-computing.intel-research.net/art/publications/avsm/
- To contact the authors:
  - firstname.lastname@intel.com

**contact us to get a copy of the paper*

References

- AGARWAL P. K., VARADARAJAN K. R.: Efficient algorithms for approximating polygonal chains. *Discrete and Computational Geometry 23*, 2 (2000), 273–291
- BOSEA P., CABELLOB S., CHEONGC O., GUDMUNDSSOND J., VAN KREVELDE M., SPECKMANN B.: Areapreserving approximations of polygonal paths. *Journal of Discrete Algorithms 4*, 4 (2006), 554–566.
- JANSEN J., BAVOIL L.: Fourier opacity mapping. In *I3D '10: Proceedings of the 2010 Symposium on Interactive 3D Graphics and Games* (Feb. 2010), pp. 165–172.
- KIM T.-Y., NEUMANN U.: Opacity shadow maps. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering* (June 2001),
- LOKOVIC T., VEACH E.: Deep shadow maps. In *Proceedings of ACM SIGGRAPH 2000 (July 2000), Computer Graphics Proceedings*, ACS, pp. 385–392.
- SINTORN E., ASSARSON U.: Hair self shadowing and transparency depth ordering using occupancy maps. In I3D '09*: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (Feb./Mar. 2009), pp. 67–74
- YANG J., HENSLEY J., GRÜN H., THIBIEROZ N.: Real-time concurrent linked list construction on the gpu. In *Rendering Techniques 2010: Eurographics Symposium on Rendering (2010),* vol. 29, Eurographics

Backup

Fixed Memory Implementation

- ComputeShader threadgroups mapped to screen tiles
- First step: parallelize over particles
  - Each threadgroup builds on chip a list of particles that overlap their tile ordered by primitive ID
- Second step: parallelize over pixels
  - Run AVSM insertion code for each pixel inside a particle
  - Enforce the correct frame buffer ordering update by mapping each pixel to a single ComputeShader thread (i.e., SIMD lane)
- Loop until all particles have been processed