EXPLORING AND EXPANDING THE CONTINUUM OF OIT ALGORITHMS

Chris Wyman
WHAT’S THIS PAPER NOT ABOUT?
WHAT’S THIS PAPER NOT ABOUT?

- Not a “survey paper,” at least in the traditional sense
  - You will not identify “the right” OIT algorithm for you
WHAT’S THIS PAPER ABOUT?

- Not a “survey paper,” at least in the traditional sense
  - You will not identify “the right” OIT algorithm for you

- Not an “algorithms paper,” at least in the traditional sense
  - Do present two new algorithms
  - Do not intend to claim these algorithms are right for you
WHAT’S THIS PAPER ABOUT?
WHAT’S THIS PAPER ABOUT?

- Story following my thoughts on order-independent transparency
  - Spurred by discussions w/developers
    - E.g., Johan Andersson’s SIGGRAPH 2015 talk
WHAT’S THIS PAPER ABOUT?

- Story following my thoughts on order-independent transparency
  - Spurred by discussions w/developers
  - Started with re-exploration of space
  - Placed on multi-dimensional continuum
WHAT’S THIS PAPER ABOUT?

- Story following my thoughts on order-independent transparency
  - Spurred by discussions w/developers
  - Started with re-exploration of space
  - Placed on multi-dimensional continuum
  - Develop algorithms exploring new spaces
    - Will talk about one today: Stochastic Layered Alpha Blending
    - Provides continuous transition between stochastic transparency & k-buffering
Why is OIT hard?
WHY BOTHER AT ALL?

- [Porter and Duff 84] outlined numerous common compositing operations.
  - The “over” operator, using multiplicative blending, describes most real interactions:
    \[
    c_{result} = \alpha_0 c_0 + (1 - \alpha_0) \alpha_1 c_1
    \]
  - For streaming compute, you need to sort geometry **or** keep all \( \alpha_i \) and \( c_i \) around.

*Merge two fragments then later try to insert one in between?*
WHY BOTHER AT ALL?

- Sorting geometry in advance can fail
  - May be no “correct” order for triangles

- Keep a list of fragments per pixel (i.e., A-Buffers [Carpenter 84])
  - Virtually unbounded** GPU memory
  - *Still* need to sort fragments to apply over operator in correctly

- Not just a raster problem; affects ray tracing, too
  - Unless it guarantees ray hits returned perfectly ordered

** You can define a very conservative upper bound, but it’s quite unhelpful.
Building an OIT continuum
HOW DO OIT ALGORITHMS WORK?

new triangle fragment contributes to pixel?
HOW DO OIT ALGORITHMS WORK?

- Different answers, including:
  - Only if closest fragment: [Depth peeling]
  - Closest & passes α-threshold: [Alpha testing]
  - Randomly decide: [Stochastic transparency]
  - Always use new fragments: [Most algorithms]
HOW DO OIT ALGORITHMS WORK?

new triangle fragment

contributes to pixel?

no

yes

per-pixel storage format?
HOW DO OIT ALGORITHMS WORK?

Different answers, including:

- Store 1 layer per pass [Depth peeling]
- Store k layers [K-buffer, alpha blending (k=1), many other algorithms]
- Store k samples [Stochastic transparency]
- Store k nodes [Deep shadow maps]
- Store k coefficients [Fourier opacity maps]
HOW DO OIT ALGORITHMS WORK?

1. new triangle fragment
2. contributes to pixel? no
3. per-pixel storage format? yes
4. combine k+1 items into storage for only k?
HOW DO OIT ALGORITHMS WORK?

- Different answers, including:
  - Discard furthest
  - Merge frags w / closest depth
  - Merge 2 most distant frags
  - Merge 2 most near frags
  - Sum coefs in Fourier space

  [Depth peeling, hybrid transparency]
  [Z^3]
  [Multi-layer alpha blend]
  [Original k-buffer]
  [Fourier opacity maps]
HOW DO OIT ALGORITHMS WORK?

- Discarding introduces bias or noise
HOW DO OIT ALGORITHMS WORK?

- Discarding introduces bias or noise
  - That’s OK; discard [Depth peeling, screen-door transparency]
  - Sum $\alpha$-weighted contribs of discarded frags [Stochastic transparency, hybrid transparency, phenomenological models]
## CONTINUUM SUMMARY

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory Limit</th>
<th>Insertion Heuristic</th>
<th>Merge Heuristic</th>
<th>Normalize?</th>
<th>Use Alpha or Coverage?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-buffer [Car84]</td>
<td>none</td>
<td>always</td>
<td>no merging</td>
<td>no</td>
<td>either†</td>
</tr>
<tr>
<td>Alpha Testing</td>
<td>1 layer</td>
<td>if $\alpha &gt;$ thresh</td>
<td>discard furthest</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Alpha Compositing [PD84]</td>
<td>1 layer</td>
<td>always</td>
<td>over operator</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Screen-Door Transparency [FGH*85]</td>
<td>k z-samples</td>
<td>always</td>
<td>z-test, discard occluded</td>
<td>no</td>
<td>coverage</td>
</tr>
<tr>
<td>$Z^3$ [JC99]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/closest depths</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Deep Shadow Maps [LV00]</td>
<td>k line segments</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Depth Peeling [Eve01]</td>
<td>1 layer</td>
<td>if closest</td>
<td>discard furthest</td>
<td>no</td>
<td>either†</td>
</tr>
<tr>
<td>Opacity Shadow Maps [KN01]</td>
<td>k bins</td>
<td>always</td>
<td>$\alpha$-weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Density Clustering [MKBVR04]</td>
<td>k bins</td>
<td>always</td>
<td>k-means clustering</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>k-Buffers [BCL*07]</td>
<td>k layers</td>
<td>always</td>
<td>merge closest to camera</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Sort-Independent Alpha Blending [Mes07]</td>
<td>1 layer</td>
<td>always</td>
<td>weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Deep Opacity Maps [YK08]</td>
<td>k bins</td>
<td>always</td>
<td>$\alpha$-weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Multi-Layer Depth Peeling [LHLW09]</td>
<td>k layers</td>
<td>if in k closest</td>
<td>discard furthest</td>
<td>no</td>
<td>either†</td>
</tr>
<tr>
<td>Occupancy Maps [SA09]</td>
<td>k bins</td>
<td>always</td>
<td>discard if bin occupied</td>
<td>renormalize alpha</td>
<td>alpha</td>
</tr>
<tr>
<td>Stochastic Transparency [ESSL10]</td>
<td>k samples</td>
<td>stochastic</td>
<td>z-test, discard occluded</td>
<td>$\alpha$-weighted average</td>
<td>coverage</td>
</tr>
<tr>
<td>Fourier Opacity Maps [JB10]</td>
<td>k Fourier coefs</td>
<td>stochastic</td>
<td>sum in Fourier domain</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Adaptive Volumetric Shadow Maps [SVLL10]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Transmittance Function Maps [DGMF11]</td>
<td>k DCT coefs</td>
<td>always</td>
<td>sum in DCT basis</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Adaptive Transparency [SML11]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Hybrid Transparency [MCTB13]</td>
<td>k layers</td>
<td>always</td>
<td>discard furthest</td>
<td>$\alpha$-weighted average</td>
<td>alpha</td>
</tr>
<tr>
<td>Weighted Blended OIT [MB13]</td>
<td>empirical func</td>
<td>never</td>
<td>discard all</td>
<td>$\alpha$-weighted average</td>
<td>alpha</td>
</tr>
<tr>
<td>Multi-Layer Alpha Blending [SV14]</td>
<td>k layers</td>
<td>always</td>
<td>merge furthest</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Phenomenological OIT [MM16]</td>
<td>empirical func</td>
<td>never</td>
<td>discard all</td>
<td>$\alpha$-weighted average</td>
<td>alpha</td>
</tr>
</tbody>
</table>
## CONTINUUM SUMMARY

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory Limit</th>
<th>Insertion Heuristic</th>
<th>Merge Heuristic</th>
<th>Normalize?</th>
<th>Use Alpha or Coverage?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-buffer [Car84]</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
<td>either†</td>
</tr>
<tr>
<td>Alpha Testing</td>
<td>1 layer</td>
<td></td>
<td></td>
<td></td>
<td>alpha</td>
</tr>
<tr>
<td>Alpha Compositing [PD84]</td>
<td>1 layer</td>
<td></td>
<td></td>
<td></td>
<td>alpha</td>
</tr>
<tr>
<td>Screen-Door Transparency [FGH*85]</td>
<td>k z-samples</td>
<td>always</td>
<td>z-test, discard occluded</td>
<td>no</td>
<td>coverage</td>
</tr>
<tr>
<td>Z³ [JC99]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/closest depths</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Deep Shadow Maps [LV00]</td>
<td>k line segments</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Depth Peeling [Eve01]</td>
<td>1 layer</td>
<td>if closest</td>
<td>discard furthest</td>
<td>no</td>
<td>either†</td>
</tr>
<tr>
<td>Opacity Shadow Maps [KN01]</td>
<td>k bins</td>
<td>always</td>
<td>α-weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Density Clustering [MKBVR04]</td>
<td>k bins</td>
<td>always</td>
<td>k-means clustering</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>k-Buffers [BCL*07]</td>
<td>k layers</td>
<td>always</td>
<td>merge closest to camera</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Sort-Independent Alpha Blending [Mes07]</td>
<td>1 layer</td>
<td>always</td>
<td>weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Deep Opacity Maps [YK08]</td>
<td>k bins</td>
<td>always</td>
<td>α-weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Multi-Layer Depth Peeling [LHLW09]</td>
<td>k layers</td>
<td>if in k closest</td>
<td>discard furthest</td>
<td>renormalize alpha</td>
<td>either†</td>
</tr>
<tr>
<td>Occupancy Maps [SA09]</td>
<td>k bins</td>
<td>always</td>
<td>discard if bin occupied</td>
<td>α-weighted average</td>
<td>alpha</td>
</tr>
<tr>
<td>Stochastic Transparency [ESSL10]</td>
<td>k samples</td>
<td>stochastic</td>
<td>z-test, discard occluded</td>
<td>no</td>
<td>coverage</td>
</tr>
<tr>
<td>Fourier Opacity Maps [JB10]</td>
<td>k Fourier coef</td>
<td>always</td>
<td>sum in Fourier domain</td>
<td></td>
<td>alpha</td>
</tr>
<tr>
<td>Adaptive Volumetric Shadow Maps [SVLL10]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Transmittance Function Maps [DGMF11]</td>
<td>k DCT coef</td>
<td>always</td>
<td>sum in DCT basis</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Adaptive Transparency [SML11]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Hybrid Transparency [MCTB13]</td>
<td>k layers</td>
<td>always</td>
<td>discard furthest</td>
<td>α-weighted average</td>
<td>alpha</td>
</tr>
<tr>
<td>Weighted Blended OIT [MB13]</td>
<td>empirical func</td>
<td>never</td>
<td>discard all</td>
<td>α-weighted average</td>
<td>alpha</td>
</tr>
<tr>
<td>Multi-Layer Alpha Blending [SV14]</td>
<td>k layers</td>
<td>always</td>
<td>merge furthest</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Phenomenological OIT [MM16]</td>
<td>empirical func</td>
<td>never</td>
<td>discard all</td>
<td>α-weighted average</td>
<td>alpha</td>
</tr>
</tbody>
</table>

**Notice:** Normalization only occurs when algorithms “discard” fragments.

*†: either alpha or coverage depends on specific implementation details.*
## CONTINUUM SUMMARY

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory Limit</th>
<th>Insertion Heuristic</th>
<th>Merge Heuristic</th>
<th>Normalize?</th>
<th>Use Alpha or Coverage?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-buffer [Car84]</td>
<td>none</td>
<td>always</td>
<td>z-test, discard occluded</td>
<td>no</td>
<td>either†</td>
</tr>
<tr>
<td>Alpha Testing</td>
<td>1 layer</td>
<td>always</td>
<td>merge w/closest depths</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Alpha Compositing [PD84]</td>
<td>1 layer</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Screen-Door Transparency [FGH+85]</td>
<td>k z-samples</td>
<td>always</td>
<td>z-test, discard occluded</td>
<td>no</td>
<td>coverage</td>
</tr>
<tr>
<td>Z³ [JC99]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/closest depths</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Deep Shadow Maps [LV00]</td>
<td>k line segments</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Depth Peeling [Eve01]</td>
<td>1 layer</td>
<td>if closest</td>
<td>discard furthest</td>
<td>no</td>
<td>either†</td>
</tr>
<tr>
<td>Opacity Shadow Maps [KN01]</td>
<td>k bins</td>
<td>always</td>
<td>discard if bin occupied</td>
<td>renormalize alpha</td>
<td>alpha</td>
</tr>
<tr>
<td>Density Clustering [MKBVR04]</td>
<td>k bins</td>
<td>always</td>
<td>stochastic z-test, discard occluded</td>
<td>α-weighted average</td>
<td>coverage</td>
</tr>
<tr>
<td>k-Buffers [BCL*07]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Sort-Independent Alpha Blending [Mes07]</td>
<td>1 layer</td>
<td>always</td>
<td>weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Deep Opacity Maps [YK08]</td>
<td>k bins</td>
<td>always</td>
<td>α-weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Multi-Layer Depth Peeling [LHLW09]</td>
<td>k layers</td>
<td>if in k closest</td>
<td>discard furthest</td>
<td>no</td>
<td>either†</td>
</tr>
<tr>
<td>Occupancy Maps [SA09]</td>
<td>k bins</td>
<td>always</td>
<td>discard if bin occupied</td>
<td>renormalize alpha</td>
<td>alpha</td>
</tr>
<tr>
<td>Stochastic Transparency [ESSL10]</td>
<td>k samples</td>
<td>stochastic</td>
<td>z-test, discard occluded</td>
<td>α-weighted average</td>
<td>coverage</td>
</tr>
<tr>
<td>Fourier Opacity Maps [JB10]</td>
<td>k Fourier coefs</td>
<td>always</td>
<td>sum in Fourier domain</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Adaptive Volumetric Shadow Maps [SVLL10]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Transmittance Function Maps [DGMF11]</td>
<td>k DCT coefs</td>
<td>always</td>
<td>sum in DCT basis</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Adaptive Transparency [SML11]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Hybrid Transparency [MCTB13]</td>
<td>k layers</td>
<td>always</td>
<td>discard furthest</td>
<td>α-weighted average</td>
<td>alpha</td>
</tr>
<tr>
<td>Weighted Blended OIT [MB13]</td>
<td>empirical func</td>
<td>never</td>
<td>discard all</td>
<td>α-weighted average</td>
<td>alpha</td>
</tr>
<tr>
<td>Multi-Layer Alpha Blending [SV14]</td>
<td>k layers</td>
<td>always</td>
<td>merge furthest</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Phenomenological OIT [MM16]</td>
<td>empirical func</td>
<td>never</td>
<td>discard all</td>
<td>α-weighted average</td>
<td>alpha</td>
</tr>
</tbody>
</table>

Notice: Normalization only occurs when algorithms “discard” fragments

Normalization can be viewed as “storing k+1 layers,” using α-weighted merge on the furthest layer.
So what?

(Or: Let’s look at an example of how this is useful)
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory Limit</th>
<th>Insertion Heuristic</th>
<th>Merge Heuristic</th>
<th>Normalize?</th>
<th>Use Alpha or Coverage?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-buffer [Car84]</td>
<td>none</td>
<td>always</td>
<td>no merging</td>
<td>no</td>
<td>either†</td>
</tr>
<tr>
<td>Alpha Testing</td>
<td>1 layer</td>
<td>if $\alpha &gt; \text{thresh}$</td>
<td>discard furthest</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Alpha Compositing [PD84]</td>
<td>1 layer</td>
<td>always</td>
<td>over operator</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Screen-Door Transparency [FGH*85]</td>
<td>k z-samples</td>
<td>always</td>
<td>$z$-test, discard occluded</td>
<td>no</td>
<td>coverage</td>
</tr>
<tr>
<td>$Z^3$ [JC99]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/closest depths</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Deep Shadow Maps [LV00]</td>
<td>k line segments</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Depth Peeling [Eve01]</td>
<td>1 layer</td>
<td>if closest</td>
<td>discard furthest</td>
<td>no</td>
<td>either†</td>
</tr>
<tr>
<td>Opacity Shadow Maps [KN01]</td>
<td>k bins</td>
<td>always</td>
<td>$\alpha$-weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Density Clustering [MKBVR04]</td>
<td>k bins</td>
<td>always</td>
<td>$k$-means clustering</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>$k$-Buffers [BCL*07]</td>
<td>k layers</td>
<td>always</td>
<td>merge closest to camera</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Sort-Independent Alpha Blending [Mes07]</td>
<td>1 layer</td>
<td>always</td>
<td>weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Deep Opacity Maps [YK08]</td>
<td>k bins</td>
<td>always</td>
<td>$\alpha$-weighted sum</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Multi-Layer Depth Peeling [LHLW09]</td>
<td>k layers</td>
<td>if in k closest</td>
<td>discard furthest</td>
<td>no</td>
<td>either†</td>
</tr>
<tr>
<td>Occupancy Maps [SA09]</td>
<td>k bins</td>
<td>always</td>
<td>discard if bin occupied</td>
<td>renormalize alpha</td>
<td>alpha</td>
</tr>
<tr>
<td>Stochastic Transparency [ESSL10]</td>
<td>k samples</td>
<td>always</td>
<td>$z$-test, discard occluded</td>
<td>$\alpha$-weighted average</td>
<td>coverage</td>
</tr>
<tr>
<td>Fourier Opacity Maps [JB10]</td>
<td>k Fourier coef $s$</td>
<td>always</td>
<td>sum in Fourier domain</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Adaptive Volumetric Shadow Maps [SVLL10]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Transmittance Function Maps [DGMF11]</td>
<td>k DCT coef $s$</td>
<td>always</td>
<td>sum in DCT basis</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Adaptive Transparency [SML11]</td>
<td>k layers</td>
<td>always</td>
<td>merge w/smallest error</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Hybrid Transparency [MCTB13]</td>
<td>k layers</td>
<td>always</td>
<td>discard furthest</td>
<td>$\alpha$-weighted average</td>
<td>alpha</td>
</tr>
<tr>
<td>Weighted Blended OIT [MB13]</td>
<td>empirical func</td>
<td>never</td>
<td>discard all</td>
<td>$\alpha$-weighted average</td>
<td>alpha</td>
</tr>
<tr>
<td>Multi-Layer Alpha Blending [SV14]</td>
<td>k layers</td>
<td>always</td>
<td>merge furthest</td>
<td>no</td>
<td>alpha</td>
</tr>
<tr>
<td>Phenomenological OIT [MM16]</td>
<td>empirical func</td>
<td>never</td>
<td>discard all</td>
<td>$\alpha$-weighted average</td>
<td>alpha</td>
</tr>
</tbody>
</table>

Interesting note: For the Stochastic Transparency algorithm, the decision to renormalize alpha is indicated as "alpha," suggesting that normalization is performed in the context of stochastic calculations.
WHAT IS STOCHASTIC TRANSPARENCY?

- When rasterizing frag into k-sample buffer:
  - Stochastically cover $\alpha \cdot k$ samples
WHAT IS STOCHASTIC TRANSPARENCY?

- When rasterizing frag into k-sample buffer:
  - Stochastically cover $\alpha \cdot k$ samples
  - Let’s look at an example pixel with 16x MSAA
    - *(MSAA pattern simplified for display)*

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Values represent current depth sample.
WHAT IS STOCHASTIC TRANSPARENCY?

- When rasterizing frag into k-sample buffer:
  - Stochastically cover $\alpha \cdot k$ samples
  - Let’s look at an example pixel with 16x MSAA
    - (MSAA pattern simplified for display)
  - First: draw red fragment, $z = 0.5$, $\alpha = 0.5$

Values represent current depth sample

Set 8 samples to red; depth test each
WHAT IS STOCHASTIC TRANSPARENCY?

- When rasterizing frag into k-sample buffer:
  - Stochastically cover $\alpha \cdot k$ samples
  - Let’s look at an example pixel with 16x MSAA
    - (MSAA pattern simplified for display)
  - First: draw red fragment, $z = 0.5$, $\alpha = 0.5$
  - Second: draw blue fragment, $z = 0.7$, $\alpha = 0.5$

Values represent current depth sample

Set 8 samples to blue; depth test each
WHAT IS STOCHASTIC TRANSPARENCY?

- When rasterizing frag into $k$-sample buffer:
  - Stochastically cover $\alpha \cdot k$ samples
  - Let’s look at an example pixel with 16x MSAA
    - (MSAA pattern simplified for display)
    - First: draw red fragment, $z = 0.5$, $\alpha = 0.5$
    - Second: draw blue fragment, $z = 0.7$, $\alpha = 0.5$
    - Third: draw green fragment, $z = 0.3$, $\alpha = 0.5$

<table>
<thead>
<tr>
<th>Values represent current depth sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
</tr>
<tr>
<td>0.7</td>
</tr>
<tr>
<td>0.5</td>
</tr>
<tr>
<td>0.3</td>
</tr>
</tbody>
</table>

Set 8 samples to green; depth test each
WHAT IS STOCHASTIC TRANSPARENCY?

» When rasterizing frag into k-sample buffer:
  » Stochastically cover $\alpha \cdot k$ samples
  » Let’s look at an example pixel with 16x MSAA
    » (MSAA pattern simplified for display)
    » First: draw red fragment, $z = 0.5$, $\alpha = 0.5$
    » Second: draw blue fragment, $z = 0.7$, $\alpha = 0.5$
    » Third: draw green fragment, $z = 0.3$, $\alpha = 0.5$
    » Fourth: draw yellow fragment, $z = 0.9$, $\alpha = 1.0$

Values represent current depth sample

Set 16 samples to yellow; depth test each
WHAT IS STOCHASTIC TRANSPARENCY?

- When rasterizing frag into k-sample buffer:
  - Stochastically cover $\alpha \cdot k$ samples
  - Let’s look at an example pixel with 16x MSAA
    - (MSAA pattern simplified for display)
    - First: draw red fragment, $z = 0.5$, $\alpha = 0.5$
    - Second: draw blue fragment, $z = 0.7$, $\alpha = 0.5$
    - Third: draw green fragment, $z = 0.3$, $\alpha = 0.5$
    - Fourth: draw yellow fragment, $z = 0.9$, $\alpha = 1.0$
  - 2nd pass accum. color using this as depth oracle
## OBSERVATIONS

- Can lose surfaces (like yellow one)
  - But it still converges; surface loss is *stochastic*
OBSERVATIONS

- Can lose surfaces (like yellow one)
  - But it still converges; surface loss is *stochastic*
- Loss worse if nearby surfaces almost opaque
  - Could easily lose blue surface
Can lose surfaces (like yellow one)

- But it still converges; surface loss is \textit{stochastic}

Loss worse if nearby surfaces almost opaque

- Could easily lose blue surface
- Also noticed in my experiments
  - Dashboard and seat noisier with high alpha than low!

\( \alpha = 0.4, \ 8 \ \text{spp} \)

\( \alpha = 0.98, \ 8 \ \text{spp} \)

Note: Even uses stratified sampling!
OBSERVATIONS

- Can lose surfaces (like yellow one)
  - But it still converges; surface loss is *stochastic*
- Loss worse if nearby surfaces almost opaque
  - Could easily lose blue surface
  - Also noticed in my experiments
    - Dashboard and seat noisier with high alpha than low!
- Seems wasteful to store 8 copies of $z = 0.3$ **
  - Why not store one copy of $z = 0.3$ and a coverage mask?

** Glossing over some details here; feel free to ask later.**
OBSERVATIONS

- Can lose surfaces (like yellow one)
  - But it still converges; surface loss is *stochastic*

- Loss worse if nearby surfaces almost opaque
  - Could easily lose blue surface
  - Also noticed in my experiments
    - Dashboard and seat noisier with high alpha than low!

- Seems wasteful to store 8 copies of $z = 0.3$ **
  - Why not store one copy of $z = 0.3$ and a coverage mask?

- *Implicitly* layered – stores (up to) 16 surfaces per pixel (for 16x MSAA)
  - Also wasteful to store just 3 layers in a structure that can hold 16
Stochastic Layered Alpha Blending (SLAB)
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- An explicit $k$-layered algorithm with stoc. transparency’s characteristics
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- An explicit $k$-layered algorithm with stoc. transparency’s characteristics
  - Memory: store $k$ layers, each with depth and b-bit coverage mask
  - Insertion: probabilistically insert fragments into per-pixel lists
  - Merging: if $> k$ layers, simply discard the furthest
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- An explicit $k$-layered algorithm with stoc. transparency’s characteristics
  - Memory: store $k$ layers, each with depth and $b$-bit coverage mask
  - Insertion: probabilistically insert fragments into per-pixel lists
  - Merging: if $> k$ layers, simply discard the furthest

- Identical results to $k$ spp stoc. transparency, if $k \geq b$
  - But can independently change values of $k$ and $b$
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- An **explicit** $k$-layered algorithm with stoc. transparency’s characteristics
  - **Memory:** store $k$ layers, each with depth and $b$-bit coverage mask
  - **Insertion:** probabilistically insert fragments into per-pixel lists
  - **Merging:** if $> k$ layers, simply discard the furthest

- Identical results to $k$ spp stoc. transparency, if $k \geq b$
  - **But** can independently change values of $k$ and $b$
    - Useful since stoc. transp. rarely stores $k$ surfaces in a $k$-sample buffer
    - Also can explicitly increase $b$ much further $\rightarrow$ reduce noise on existing layers
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- Our same example from before:
  - First: draw red fragment, \( z = 0.5 \), \( \alpha = 0.5 \)
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- Our same example from before:
  - First: draw red fragment, \( z = 0.5, \alpha = 0.5 \)
  - Second: draw blue fragment, \( z = 0.7, \alpha = 0.5 \)
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

Our same example from before:

- First: draw red fragment, $z = 0.5$, $\alpha = 0.5$
- Second: draw blue fragment, $z = 0.7$, $\alpha = 0.5$
- Third: draw green fragment, $z = 0.3$, $\alpha = 0.5$
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- Our same example from before:
  - First: draw red fragment, $z = 0.5$, $\alpha = 0.5$
  - Second: draw blue fragment, $z = 0.7$, $\alpha = 0.5$
  - Third: draw green fragment, $z = 0.3$, $\alpha = 0.5$
  - Fourth: draw yellow fragment, $z = 0.9$, $\alpha = 1.0$
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- Our same example from before:
  - First: draw red fragment, $z = 0.5, \alpha = 0.5$
  - Second: draw blue fragment, $z = 0.7, \alpha = 0.5$
  - Third: draw green fragment, $z = 0.3, \alpha = 0.5$
  - Fourth: draw yellow fragment, $z = 0.9, \alpha = 1.0$

- Layers get inserted only if not occluded
  - Adds stochasm, if masks randomly chosen
  - Different random masks might keep this layer

Coverage Mask

<table>
<thead>
<tr>
<th>Coverage Mask</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
</tr>
</tbody>
</table>

Layers

occluded by closer surfaces
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- Our same example from before:
  - First: draw red fragment, \( z = 0.5, \alpha = 0.5 \)
  - Second: draw blue fragment, \( z = 0.7, \alpha = 0.5 \)
  - Third: draw green fragment, \( z = 0.3, \alpha = 0.5 \)
  - Fourth: draw yellow fragment, \( z = 0.9, \alpha = 1.0 \)
- Layers get inserted only if not occluded
- Adds stochasm, if masks randomly chosen
- Different random masks might keep this layer
- If \( k = 2 \), layers beyond 2\(^\text{nd}\) get discarded

<table>
<thead>
<tr>
<th>Coverage Mask</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td></td>
</tr>
</tbody>
</table>

<diagram with coverage masks and depth values>

not in 2 closest layers
ADJUSTING PARAMETERS

- Aim to reduce noise
  - One way: avoid discarding layers that impact color

Coverage Mask

<table>
<thead>
<tr>
<th>Layers</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td></td>
</tr>
</tbody>
</table>
ADJUSTING PARAMETERS

- Aim to reduce noise
  - One way: avoid discarding layers that impact color
- How to increase chance to store yellow frag?
ADJUSTING PARAMETERS

- Aim to reduce noise
  - One way: avoid discarding layers that impact color
- How to increase chance to store yellow frag?
  - Increase number of bits in coverage mask
ADJUSTING PARAMETERS

- Aim to reduce noise
  - One way: avoid discarding layers that impact color
- How to increase chance to store yellow frag?
  - Increase number of bits in coverage mask
- Larger coverage masks → lower noise
- What happens as # coverage bits increases?

<table>
<thead>
<tr>
<th>Coverage Mask</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>0.9</td>
<td>0.9</td>
</tr>
</tbody>
</table>
ADJUSTING PARAMETERS

- Aim to reduce noise
  - One way: avoid discarding layers that impact color

- How to increase chance to store yellow frag?
  - Increase number of bits in coverage mask

- Larger coverage masks → lower noise

- What happens as # coverage bits increases?
  - Starts to behave as alpha

- Interesting to ask:
  - Can we stochastically insert fragments using alpha?
Let’s compute an insertion probability

Q: What’s the chance random bitmask B is visible behind random bitmask A?
SLAB USING IMPLICIT COVERAGE

- Let’s compute an insertion probability
  - Q: What’s the chance random bitmask B is visible behind random bitmask A?

![Diagram](image_url)

Hidden if *none* of these get covered by bits in bitmask B
SLAB USING IMPLICIT COVERAGE

- Let’s compute an insertion probability
  - Q: What’s the chance random bitmask B is visible behind random bitmask A?

Naïve random sampling:
Covered with probability $\alpha_B$
Uncovered with prob $(1 - \alpha_B)$
Let’s compute an insertion probability

Q: What’s the chance random bitmask B is visible behind random bitmask A?

Naïve random sampling:
Covered with probability $\alpha_B$
Uncovered with prob $(1 - \alpha_B)$

All uncovered with prob: $(1-\alpha_B)^6$
Bitmask B visible with prob: $1-(1-\alpha_B)^6$
Let’s compute an insertion probability

Q: What’s the chance random bitmask $B$ is visible behind random bitmask $A$?

\[
P_b(\beta_A, \beta_B) = 1 - \left(1 - \frac{\beta_B}{b}\right)^{(b-\beta_A)}
\]

Or

\[
P_b(\beta_A, \alpha_B) = 1 - (1 - \alpha_B)^{(b-\beta_A)}
\]

$\beta_A \equiv \# \text{ bits covered}$

$\beta_A = \lfloor \alpha_A b \rfloor$ or $\lceil \alpha_A b \rceil$

for $b$ bits in bitmask
SLAB USING IMPLICIT COVERAGE

Let’s compute an insertion probability

- Q: What’s the chance random bitmask B is visible behind random bitmask A?

\[
P_b(\beta_A, \gamma_B) = 1 - \left(1 - \frac{\beta_B}{b}\right)^{(b-\beta_A)}
\]

Or

\[
P_b(\beta_A, \alpha_B) = 1 - (1 - \alpha_B)^{(b-\beta_A)}
\]

\(\beta_A \equiv \# \text{ bits covered}\)
\(\beta_A = [\alpha_A b] \text{ or } [\alpha_A b]\)

for b bits in bitmask

\(\gamma_B \equiv \# \text{ bits that must be uncovered}\)
SLAB USING IMPLICIT COVERAGE

- Let’s compute an insertion probability
  - Q: How about for random masks using stratified samples?

\[
P_b(\beta_A, \beta_B) = \begin{cases} 
1 - \frac{\beta_A!(b-\beta_B)!}{b!(\beta_A-\beta_B)!} & \text{if } \beta_B \leq \beta_A \\
1 & \text{if } \beta_B > \beta_A 
\end{cases}
\]

- Based on combinatorics
  - Choosing dependent probabilities so all mask bits in B are covered by A

\( \beta_A \equiv \# \text{ bits covered} \)
WAIT! NOT USING INFINITE # BITS?

Both equations require a number of bits $b$ in the coverage mask

$$P_b(\beta_A, \beta_B) = \begin{cases} 1 - \frac{\beta_A(1-\beta_B)^b}{b!(\beta_A-\beta_B)!} & \text{if } \beta_B \leq \beta_A \\ 1 & \text{if } \beta_B > \beta_A \end{cases}$$

using stratified random samples

$$P_b(\beta_A, \beta_B) = 1 - \left(1 - \frac{\beta_B}{b}\right)^{(b-\beta_A)}$$

using naïve random samples
WAIT! NOT USING INFINITE # BITS?

- Both equations require a number of bits $b$ in the coverage mask
  - Can ask what happens to $P_b$ as $b \to \infty$
  - Turns out as $b \to \infty$, $P_b \to 1$
  - Instead of stochastic insertion of fragments, they’re always inserted

\[
P_b(\beta_A, \beta_B) = \begin{cases} 
1 - \frac{\beta_A!(b-\beta_B)!}{b!(\beta_A-\beta_B)!} & \text{if } \beta_B \leq \beta_A \\
1 & \text{if } \beta_B > \beta_A
\end{cases} \quad \text{using stratified random samples}
\]

\[
P_b(\beta_A, \beta_B) = 1 - \left(1 - \frac{\beta_B}{b}\right)^{(b-\beta_A)} \quad \text{using naïve random samples}
\]
WAIT! NOT USING INFINITE # BITS?

- Both equations require a number of bits \( b \) in the coverage mask
  - Can ask what happens to \( P_b \) as \( b \to \infty \)
  - Turns out as \( b \to \infty \), \( P_b \to 1 \)
  - Instead of stochastic insertion of fragments, they’re always inserted

- Going back to our continuum
  - When \( b = k \), SLAB is equivalent to stochastic transparency
  - When \( b \to \infty \), SLAB is equivalent to hybrid transparency (a variant of k-buffer)
WAIT! NOT USING INFINITE # BITS?

- To get something between k-buffers and stoc. transp.
  - Need to use $k \leq b < \infty$
WAIT! NOT USING INFINITE # BITS?

- To get something between k-buffers and stoc. transp.
  - Need to use $k \leq b < \infty$
    - Can do this with an explicit coverage mask with b random bits
      - Using deterministic insertion based on random coverage masks
WAIT! NOT USING INFINITE # BITS?

- To get something between k-buffers and stoc. transp.
  - Need to use $k \leq b < \infty$
    - Can do this with an explicit coverage mask with b random bits
      - Using deterministic insertion based on random coverage masks
    - Can do this with an implicit coverage (i.e., alpha) using b virtual bits
      - Using stochastic insertion using probability functions
      - $b$ only controls distance along the k-buffer $\leftrightarrow$ stoc transp continuum
Let’s demonstrate
FOLIAGE MAP
(From Epic’s Unreal SDK)

All surfaces $\alpha = 0.5$
FOLIAGE MAP
(From Epic’s Unreal SDK)

All surfaces $\alpha = 0.5$

Stoc transp, 8 spp
SLAB, $k = b = 8$
SLAB, $k = 8, b = 32$
SLAB, $k = 8, b = 128$
SLAB, $k = 8, b = 32$ using alpha
Hybrid Transparency
FOLIAGE MAP
(From Epic’s Unreal SDK)

All surfaces $\alpha = 0.5$
FOLIAGE MAP
(From Epic’s Unreal SDK)

All surfaces $\alpha = 0.5$
STOCHASTIC TRANSPARENCY TO K-BUFFERS

Stochastic Layered Alpha Blending, $k=b=4$

Stochastic Transparency, 4 spp
STOCHASTIC TRANSPARENCY TO K-BUFFERS

Stochastic Layered Alpha Blending, k=4, b=32

Stochastic Transparency, 4 spp
STOCHASTIC TRANSPARENCY TO K-BUFFERS

Stochastic Layered Alpha Blending, k=4, b=8 (using alpha rather than coverage)

Stochastic Transparency, 4 spp
STOCHASTIC TRANSPARENCY TO K-BUFFERS

Stochastic Layered Alpha Blending, k=4, b=32 (using alpha rather than coverage)
Summary
SUMMARY

- Introduced an OIT continuum
  - More detailed discussion in the paper
  - My key takeaways
    - All OIT algorithms limit memory by using k “layers” (k=0,1,4,8,32 common)
    - Biggest difference is merge heuristic
    - Some algorithm do renormalization; just a fancy merge heuristic
    - Insertion via stochastic processes is underexplored
    - Algorithms using coverage masks are underexplored
SUMMARY

- Proposed two new algorithms
  - Stochastic layered alpha blending (SLAB)
  - Multi-layer coverage blending (MLCB)
    - Not discussed today, see paper for details
    - Explored combining OIT + MSAA sampling
SUMMARY

- Proposed two new algorithms
  - Stochastic layered alpha blending (SLAB)
  - Key takeaways:
    - K-buffers need not be deterministic
    - Stochastic transparency and k-buffering are similar; transition via bit count
    - “Stochastic” need not mean random bitmask generation
    - Algorithms connecting others useful; here, allow trading noise for bias
    - SLAB with alpha values can stratify samples in z (between layers)
      - (Not really discussed in this talk)
Blacksmith building, from Unity’s “The Blacksmith” demo

- Stochastic transparency
  - 4 spp
- SLAB
  - $k = 4$, $b = 4$
- SLAB
  - $k = 4$, $b = 16$
  - using alpha
- Hybrid transparency
  - 4 layers
- Multi-layer alpha blending
  - 4 layers
- Ground truth
  - (A-buffer)
- 8x MSAA, alpha-to-coverage