TRANSPARENCY IS HARD

- Work fits in the context of “order independent transparency”
  - In real time, transparency is hard
TRANSPARENCY IS HARD

- Work fits in the context of “order independent transparency”
  - In real time, transparency is hard
- Why? Existing algorithms:
  - Not in same rendering pass as opaque
  - Interacts in complex ways with other effects (e.g., AA)
  - (Some) greedily use memory
  - Often use complex locking and atomics
TRANSAPRENCY IS HARD

- Work fits in the context of “order independent transparency”
  - In real time, transparency is hard
- Why? Existing algorithms:
  - Not in same rendering pass as opaque
  - Interacts in complex ways with other effects (e.g., AA)
  - (Some) greedily use memory
  - Often use complex locking and atomics
- Takeaway:
  - Current solutions not ideal; many minimize use of transparency
WHAT’S THE PROBLEM?

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

\[ \text{Incorrect Order} \quad \text{Correct Order} \]

*Merge two fragments then later try to insert one in between?*
WHAT’S THE PROBLEM?

- 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.
<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>(\overline{\text{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>\text{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 coeffs</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 coeffs</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>

* See my High Performance Graphics 2016 paper
# RECENT WORK: OIT CONTINUUM

*See my High Performance Graphics 2016 paper*

<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>1 layer</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>$\alpha$-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 coeffs</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 coeffs</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>
So what is Stochastic Layered Alpha Blending?
WHAT IS STOCHASTIC LAYERED ALPHA BLEND

- Shows how to use stochasm in a $k$-buffer algorithm
  - I.e., allows stochastic insertion of fragments
WHAT IS STOCHASTIC LAYERED ALPHA BLEND

- Shows how to use *stochasm* in a *k*-buffer algorithm
  - i.e., allows stochastic insertion of fragments

- Shows stochastic transparency \( \equiv k \)-buffering
WHAT IS STOCHASTIC LAYERED ALPHA BLEND

- Shows how to use *stochasm* in a $k$-buffer algorithm
  - I.e., allows stochastic insertion of fragments
- Shows stochastic transparency $\equiv k$-buffering
- How?
  - By providing an explicit parameter that transitions
    - Stochastic transparency \textbf{[Enderton 10]} $\leftrightarrow$ hybrid transparency \textbf{[Maule 13]}
To Understand: Start With Stochastic Transparency
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)

Values represent current depth sample

<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>
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

<table>
<thead>
<tr>
<th></th>
<th>0.5</th>
<th>1.0</th>
<th>1.0</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.5</td>
<td>0.5</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>1.0</td>
<td>0.5</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>0.5</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

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, $z = 0.3$, $\alpha = 0.5$
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, $z = 0.9$, $\alpha = 1.0$

2\textsuperscript{nd} 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

<table>
<thead>
<tr>
<th></th>
<th>0.5</th>
<th>0.3</th>
<th>0.7</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>0.5</td>
<td>0.5</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>0.7</td>
<td>0.3</td>
<td></td>
</tr>
</tbody>
</table>
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!

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 stochastic 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

Layers get inserted only if not occluded
- Adds stochasm, if masks randomly chosen
- Different random masks might keep this layer
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 2nd get discarded
ADJUSTING PARAMETERS

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

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

![Coverage Mask and Depth Diagram](image)
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></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>

![Diagram of Coverage Masks and Depth]
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?
Let’s compute an insertion probability

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

Hidden if *none* of these get covered by bits in bitmask 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 $(1 - \alpha_B)$
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}
\]

\(\beta_A = \lfloor \alpha_A b \rfloor \text{ or } \lceil \alpha_A b \rceil
\)

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?

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

Or

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

\(\beta_A \equiv \# \text{bits covered}\)

\(\beta_A = [\alpha_A b] \text{ or } [\alpha_A b]\)

for \(b\) bits in bitmask

\(\text{prob of leaving } 1 \text{ bit uncovered}\)

\(\text{number of bits that must be uncovered}\)
SLAB USING IMPPLICIT 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
WAIT! NOT USING INFINITE # BITS?

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

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

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

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

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

- 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)

Hybrid Transparency, 4 layers
Summary
SUMMARY

- Proposed new algorithm
  - Stochastic layered alpha blending (SLAB)
SUMMARY

- Proposed new algorithm
  - 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)