EXPLORING AND EXPANDING THE CONTINUUM OF OIT ALGORITHMS

Chris Wyman

NVIDIA

High Performance Graphics 2016; June 20, 2016; Dublin, Ireland

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 ^NOT 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?

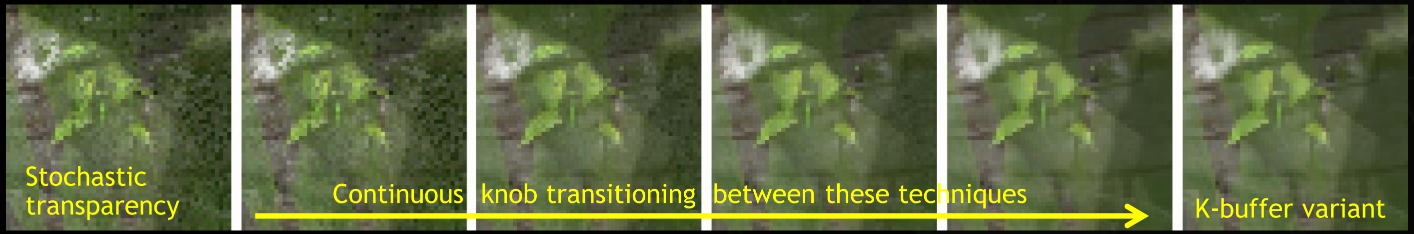
- Story following my thoughts on order-independent transparency
  - Spurred by discussions w/developers
    - E.g., Johan Andersson's SIGGRAPH 2015 talk

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?

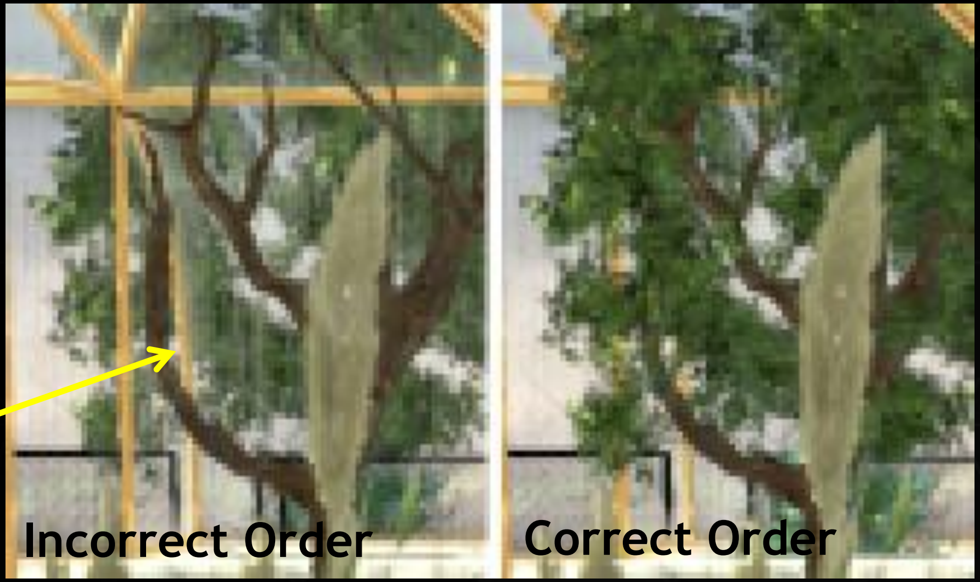
WHY BOTHER AT ALL?

- [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 α_i and c_i around

Merge two fragments then later try to insert one in between?



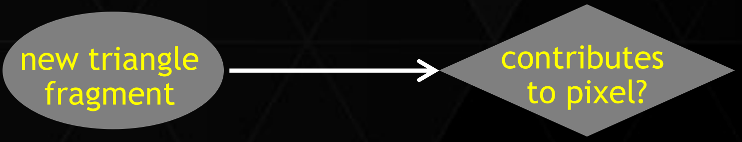
WHY BOTHER AT ALL?

- 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

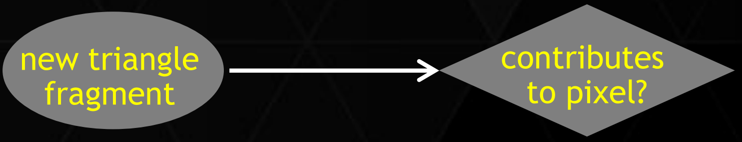
*** You can define a **very** conservative upper bound, but it’s quite unhelpful.*

Building an OIT continuum

HOW DO OIT ALGORITHMS WORK?

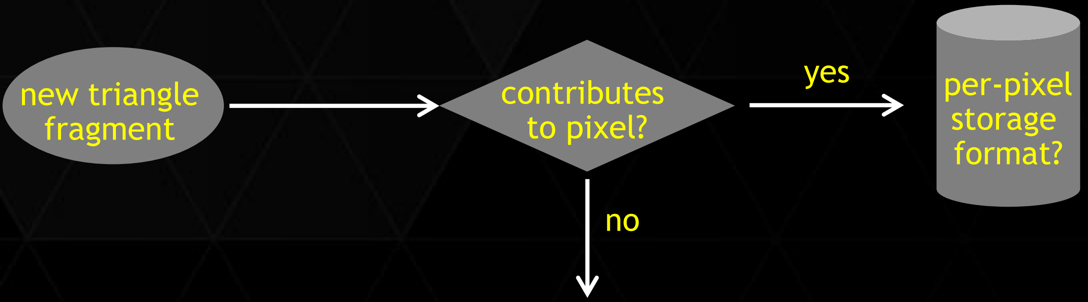


HOW DO OIT ALGORITHMS WORK?

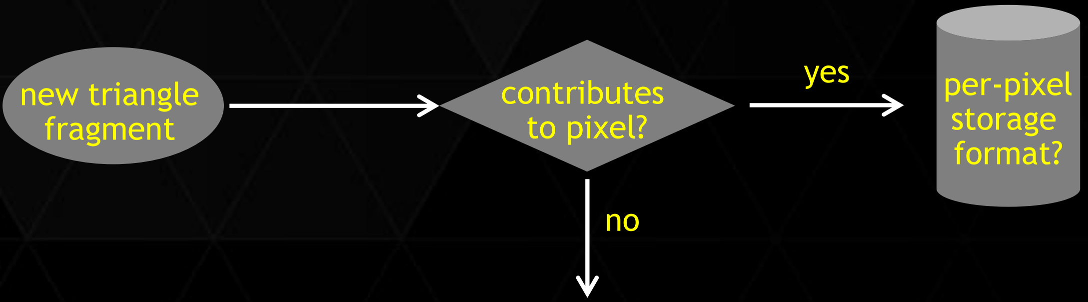


- Different answers, including:
  - Only if closest fragment [Depth peeling]
  - Closest & passes α-threshold [Alpha testing]
  - Randomly decide [Stochastic transparency]
  - Always use new fragments [Most algorithms]

HOW DO OIT ALGORITHMS WORK?

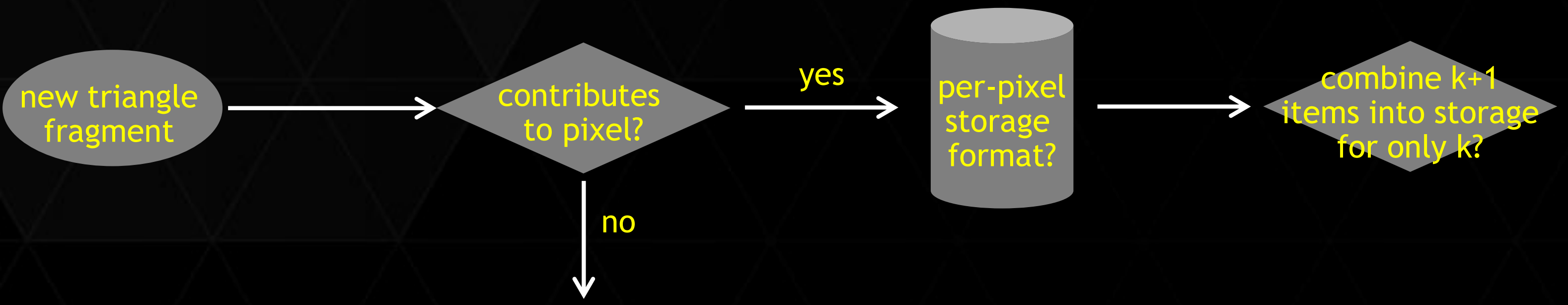


HOW DO OIT ALGORITHMS WORK?

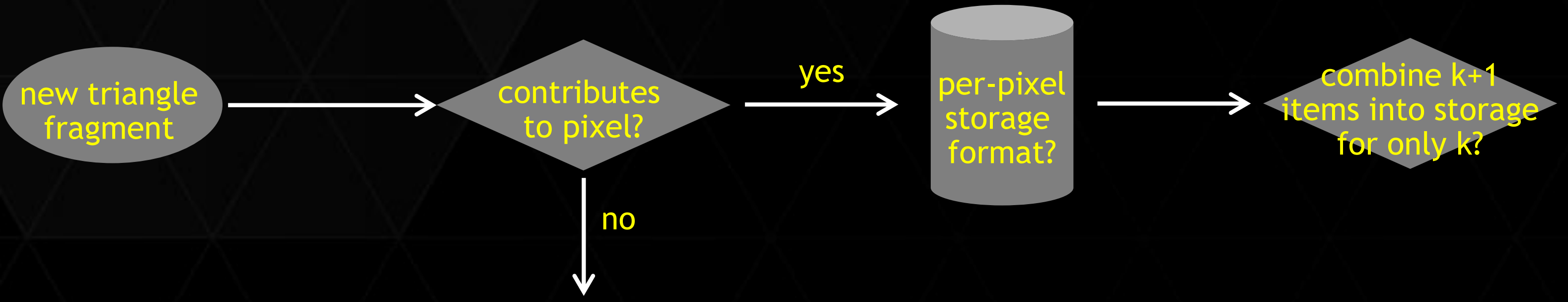


- Different answers, including:
  - Store 1 layer per pass [Depth peeling]
  - Store k layers [K-buffer, alpha blending (k=1), many other algorithms]
  - Store k samples [Stochastic transparency]
  - Store k nodes [Deep shadow maps]
  - Store k coefficients [Fourier opacity maps]

HOW DO OIT ALGORITHMS WORK?

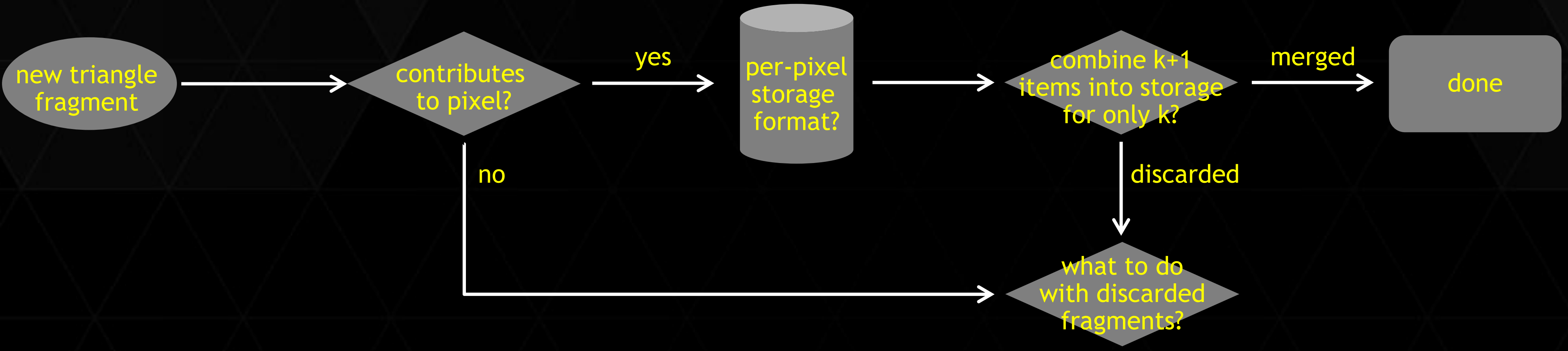


HOW DO OIT ALGORITHMS WORK?



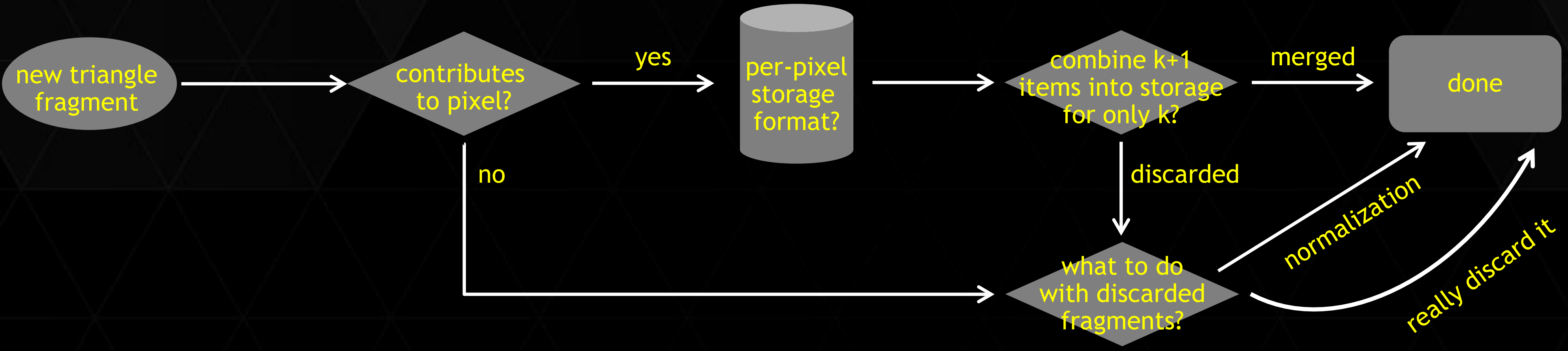
- Different answers, including:
  - Discard furthest [Depth peeling, hybrid transparency]
  - Merge frags w / closest depth [Z^3]
  - Merge 2 most distant frags [Multi-layer alpha blend]
  - Merge 2 most near frags [Original k-buffer]
  - Sum coefs in Fourier space [Fourier opacity maps]

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 α-weighted contribs of discarded frags [Stochastic transparency, hybrid transparency, phenomenological models]

CONTINUUM SUMMARY

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

CONTINUUM SUMMARY

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

Notice: Normalization only occurs when algorithms “discard” fragments

CONTINUUM SUMMARY

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

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)

CONTINUUM SUMMARY

Interesting note

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

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 α • k samples
  - Let's look at an example pixel with 16x MSAA
    - *(MSAA pattern simplified for display)*

Values represent current depth sample

1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0

WHAT IS STOCHASTIC TRANSPARENCY?

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

Values represent current depth sample

0.5	1.0	1.0	0.5
1.0	0.5	0.5	1.0
0.5	1.0	0.5	0.5
1.0	0.5	1.0	1.0

Set 8 samples to red; depth test each

WHAT IS STOCHASTIC TRANSPARENCY?

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

Values represent current depth sample

0.5	1.0	0.7	0.5
0.7	0.5	0.5	0.7
0.5	0.7	0.5	0.5
1.0	0.5	0.7	1.0

Set 8 samples to blue; depth test each

WHAT IS STOCHASTIC TRANSPARENCY?

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

Values represent current depth sample

Set 8 samples to green; depth test each

WHAT IS STOCHASTIC TRANSPARENCY?

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

Values represent current depth sample

0.5	0.3	0.7	0.3
0.7	0.5	0.5	0.3
0.5	0.3	0.3	0.5
0.3	0.3	0.7	0.3

Set 16 samples to yellow; depth test each

WHAT IS STOCHASTIC TRANSPARENCY?

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

Values represent current depth sample

0.5	0.3	0.7	0.3
0.7	0.5	0.5	0.3
0.5	0.3	0.3	0.5
0.3	0.3	0.7	0.3

OBSERVATIONS

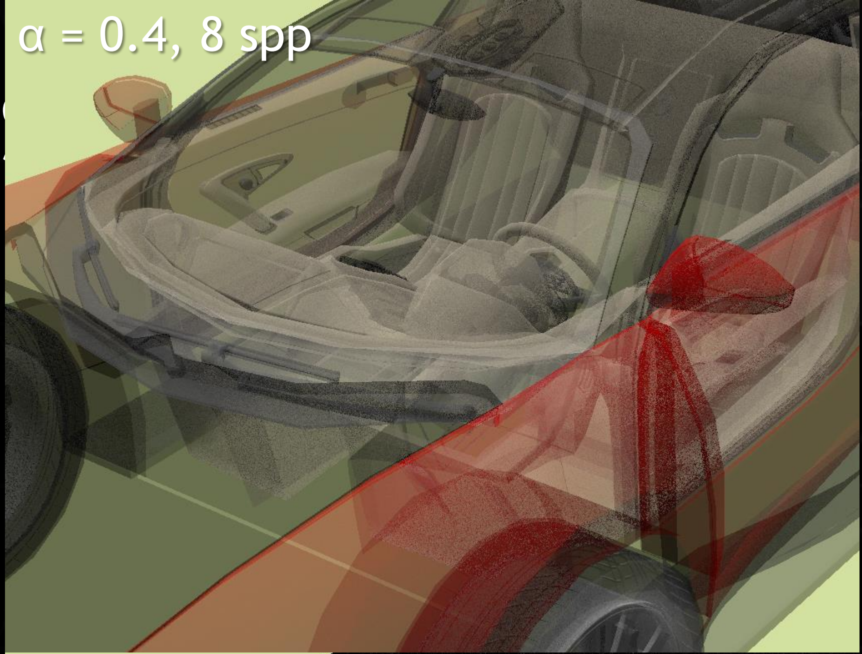
- Can lose surfaces (like yellow one)
  - But it still converges; surface loss is *stochastic*

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

OBSERVATION

- 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!



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 (<u>up to</u>) 16 surfaces per pixel (for 16x MSAA)
  - Also wasteful to store just 3 layers in a structure that can hold 16

0.5	0.3	0.7	0.3
0.7	0.5	0.5	0.3
0.5	0.3	0.3	0.5
0.3	0.3	0.7	0.3

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 ≥ b
  - *But* can independently change values of k and b

WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

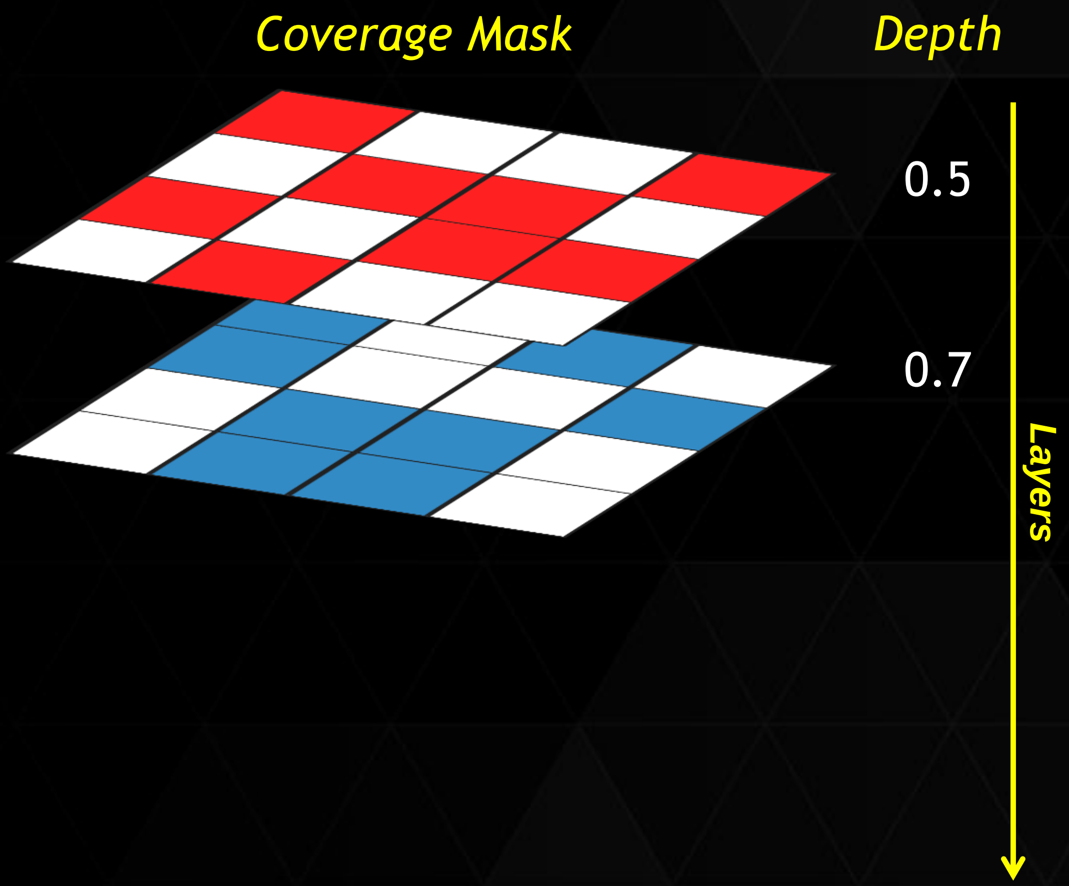
- An explicit *k*-layered algorithm with stoc. transparency's characteristics
  - Memory: store k layers, each with depth and b-bit coverage mask
  - Insertion: probabilistically insert fragments into per-pixel lists
  - Merging: if > k layers, simply discard the furthest
- Identical results to k spp stoc. transparency, if k ≥ 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 → reduce noise on existing layers

WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- Our same example from before:
  - First: draw red fragment, z = 0.5, α = 0.5

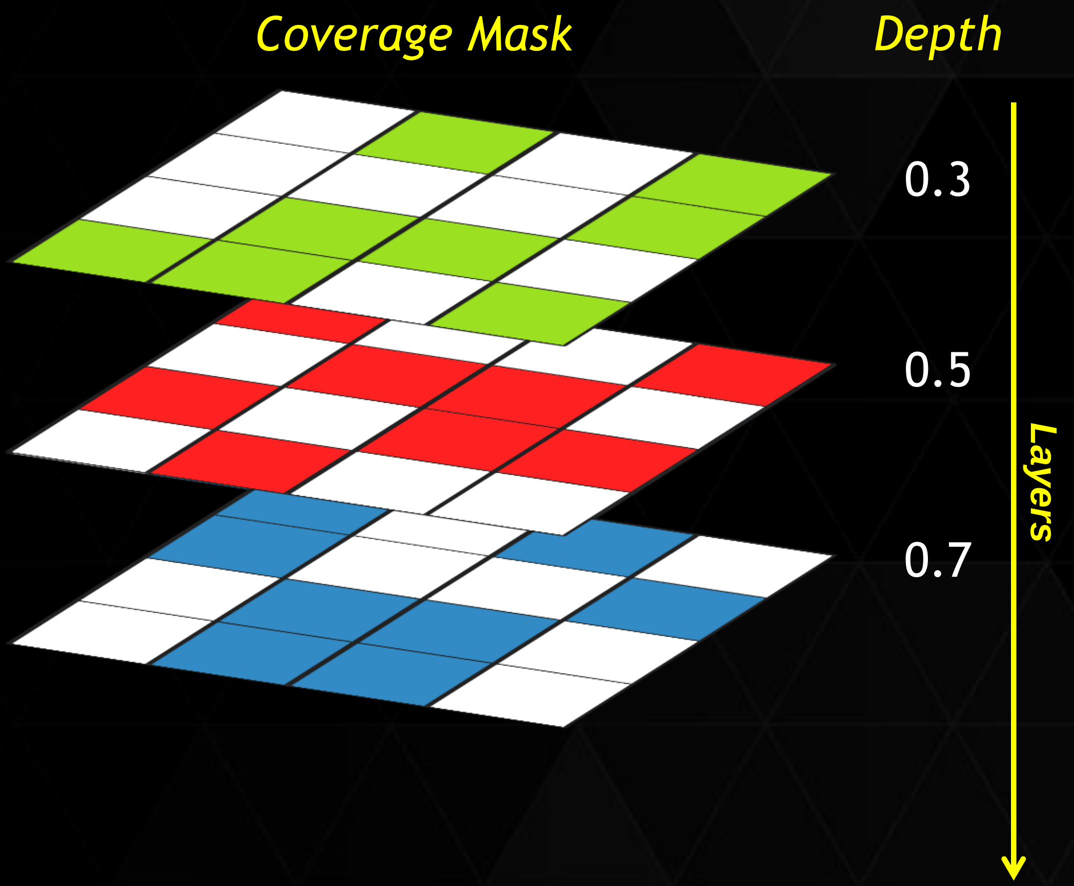
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- Our same example from before:
  - First: draw red fragment, z = 0.5, α = 0.5
  - Second: draw blue fragment, z = 0.7, α = 0.5



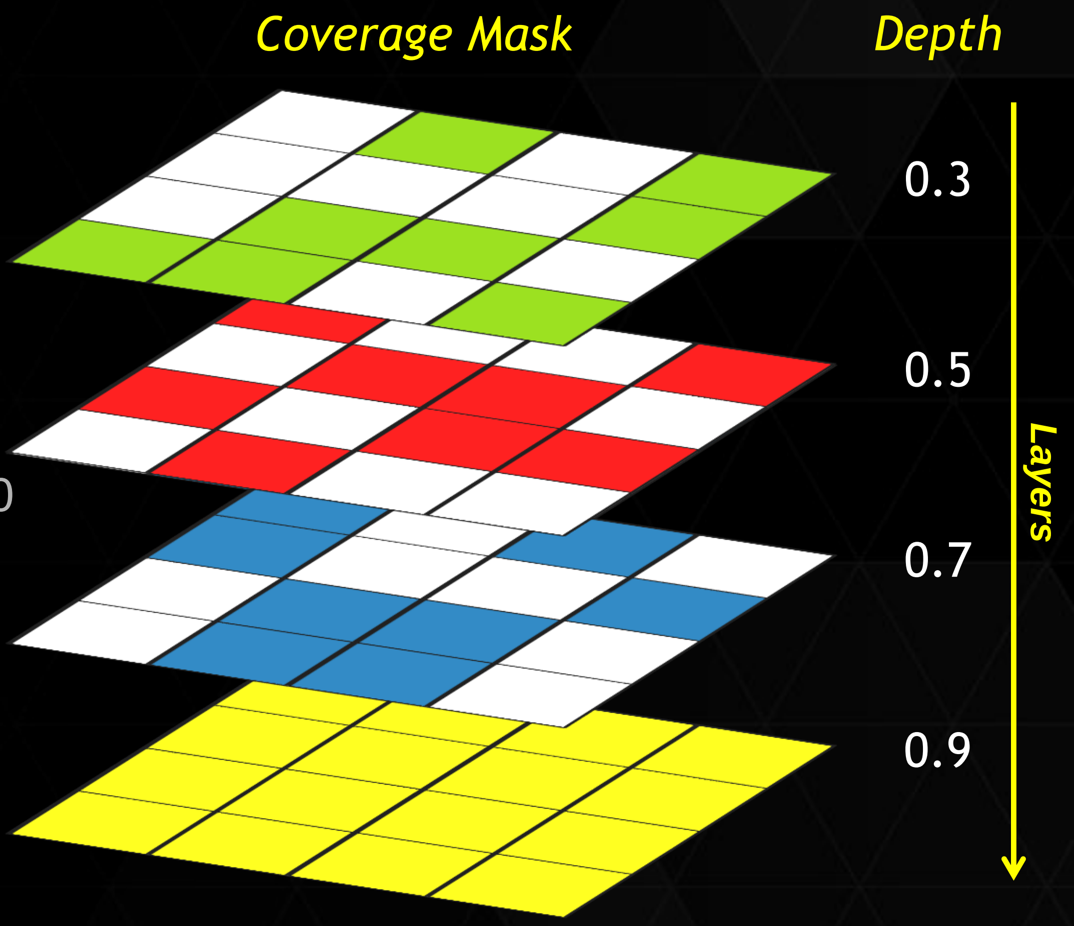
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- Our same example from before:
  - First: draw red fragment, z = 0.5, α = 0.5
  - Second: draw blue fragment, z = 0.7, α = 0.5
  - Third: draw green fragment, z = 0.3, α = 0.5



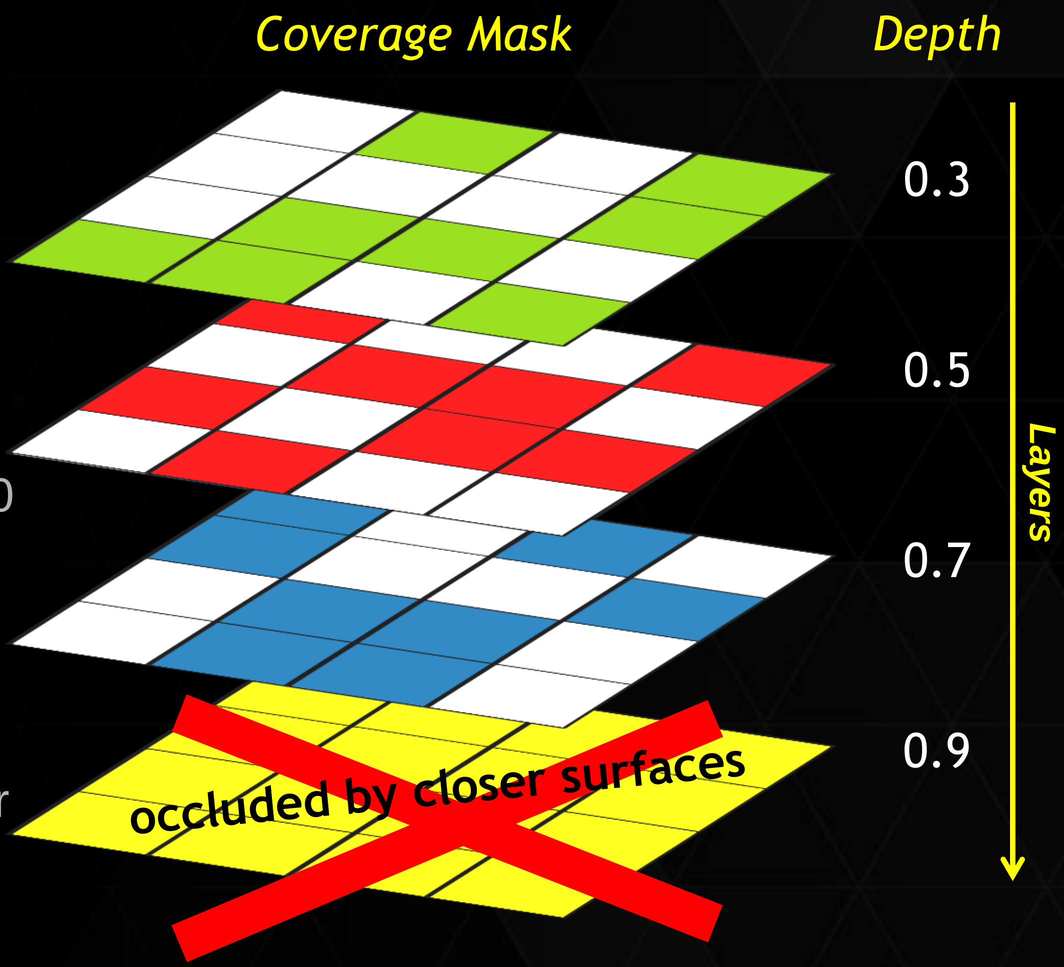
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

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



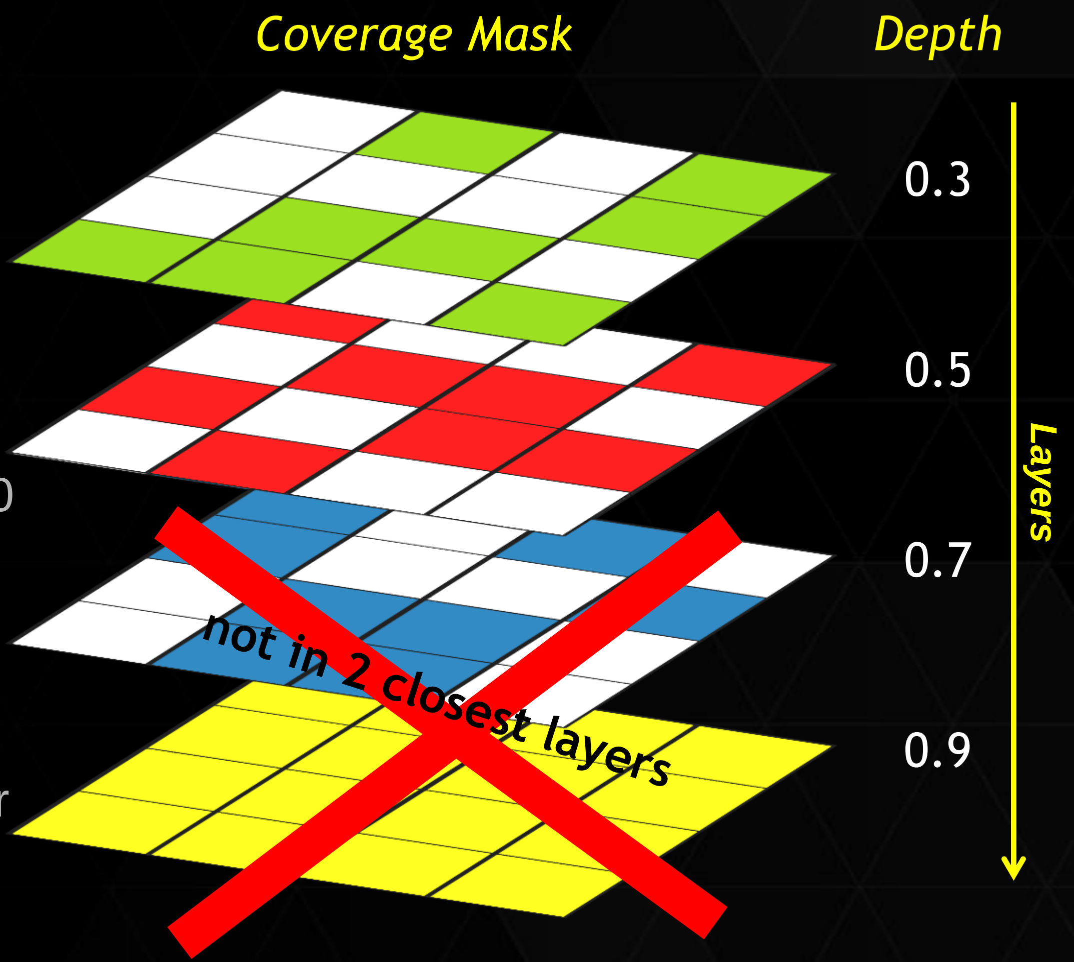
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

- Our same example from before:
  - First: draw red fragment, z = 0.5, α = 0.5
  - Second: draw blue fragment, z = 0.7, α = 0.5
  - Third: draw green fragment, z = 0.3, α = 0.5
  - Fourth: draw yellow fragment, z = 0.9, α = 1.0
- Layers get inserted only if not occluded
  - Adds stochasm, if masks randomly chosen
  - Different random masks might keep this layer



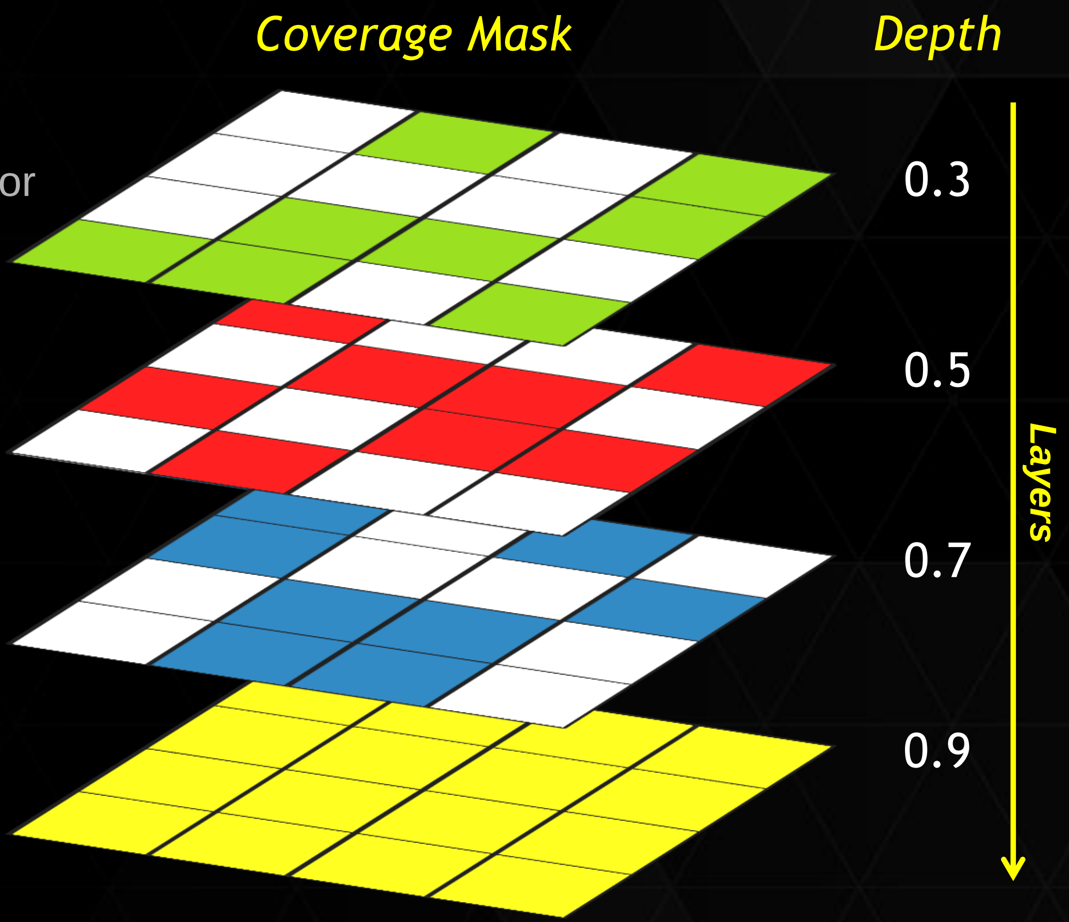
WHAT IS STOCHASTIC LAYERED ALPHA BLEND?

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



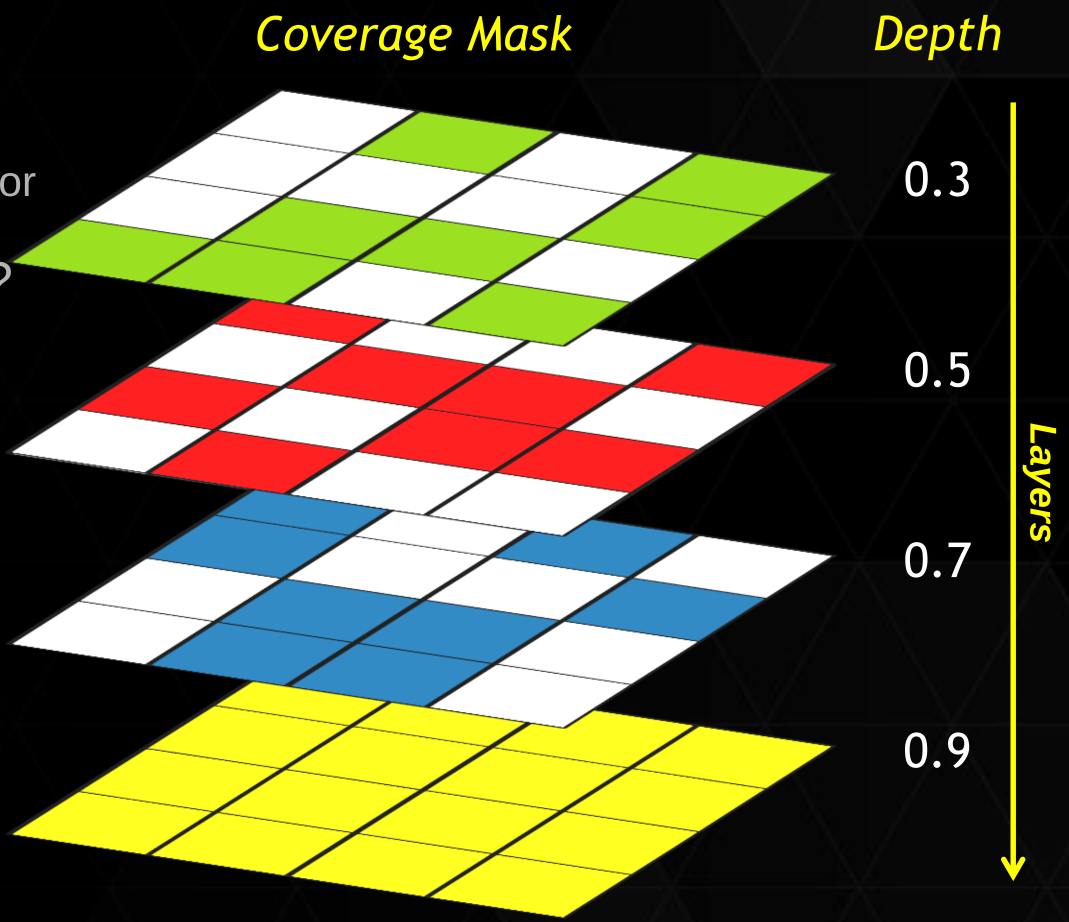
ADJUSTING PARAMETERS

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



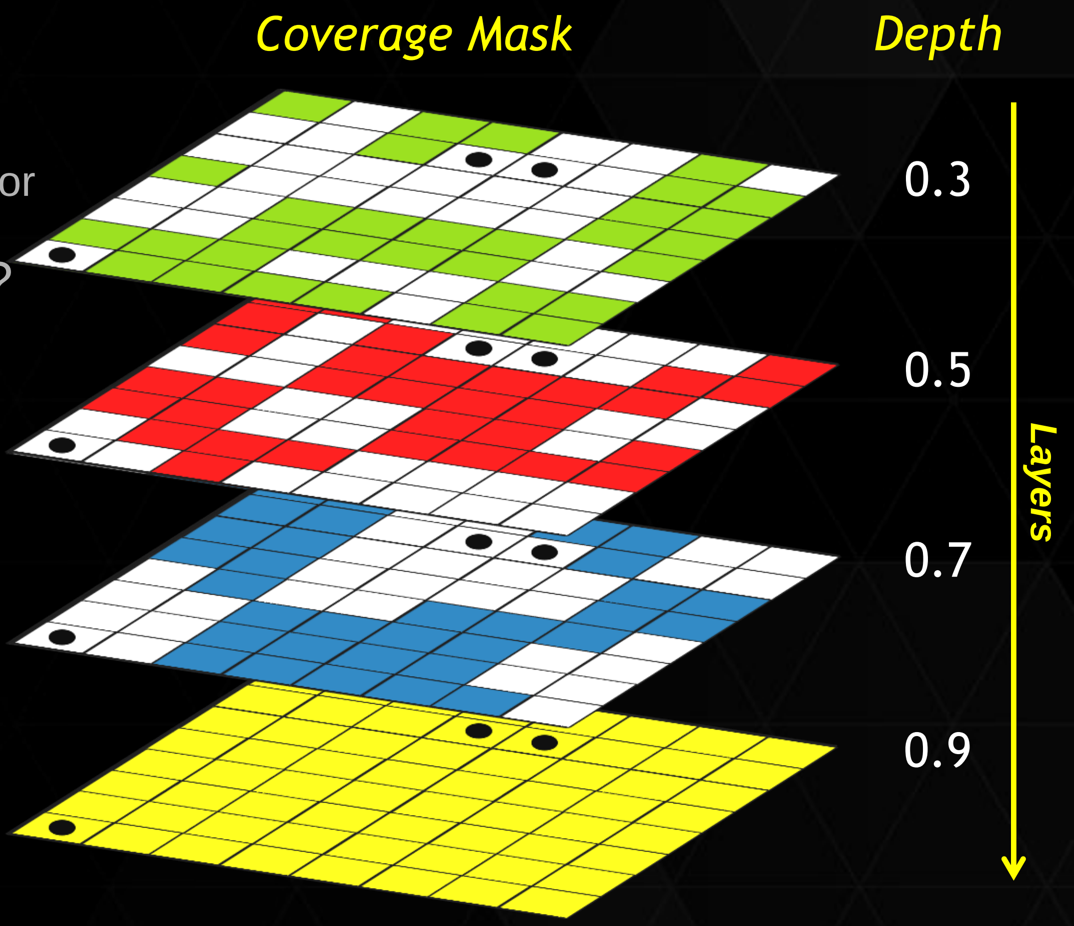
ADJUSTING PARAMETERS

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



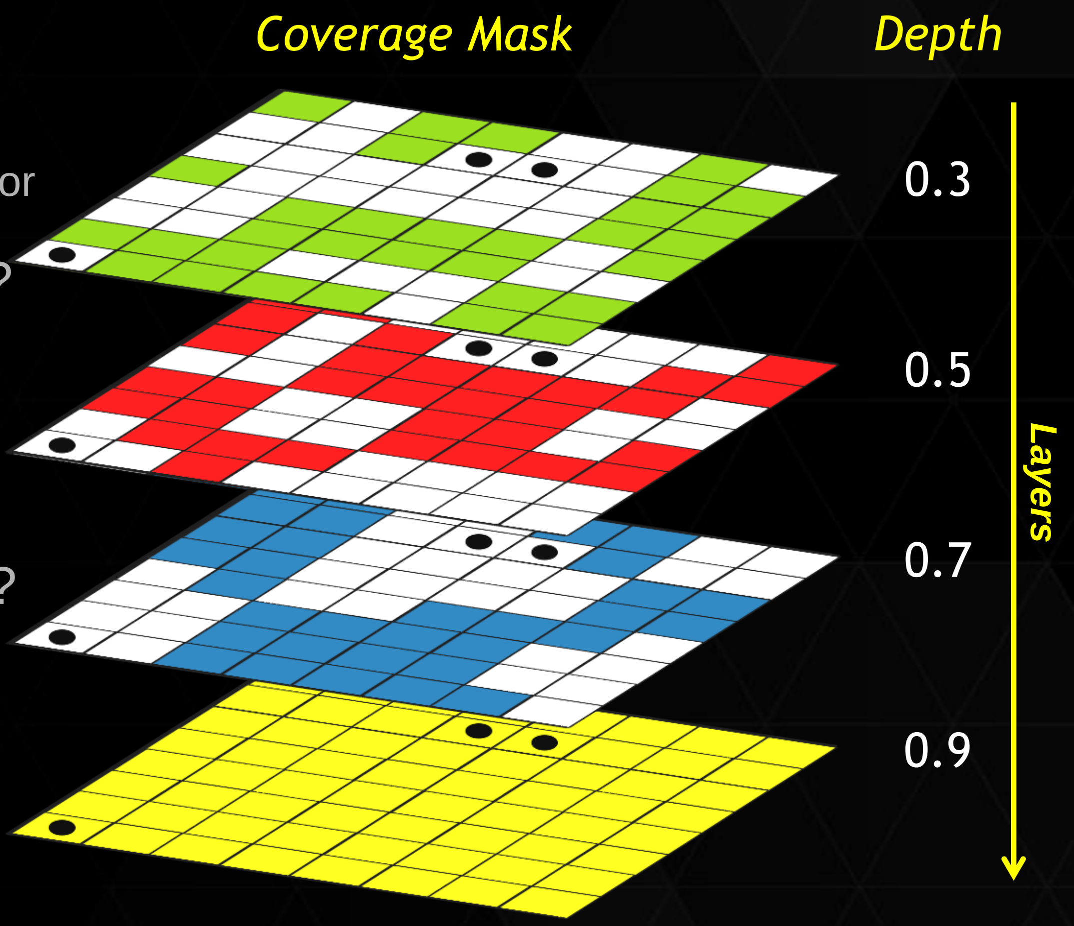
ADJUSTING PARAMETERS

- 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



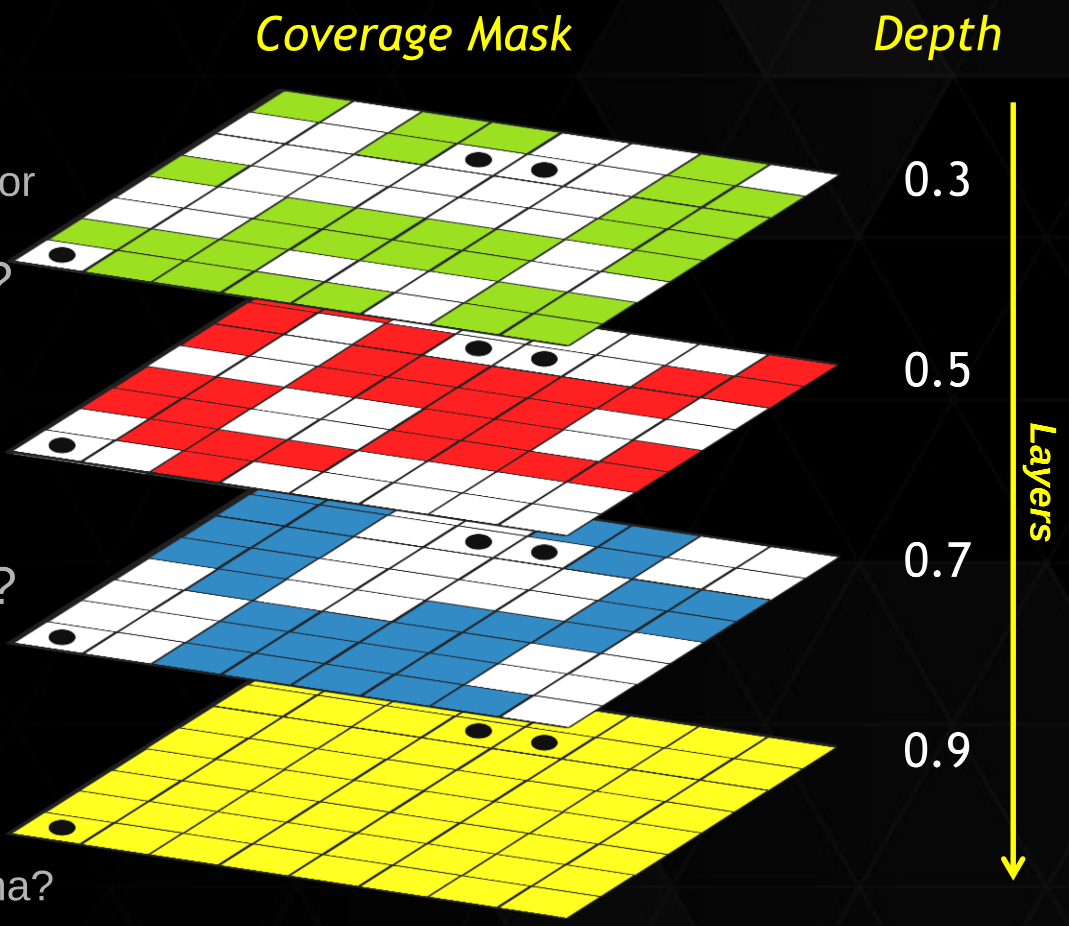
ADJUSTING PARAMETERS

- 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?



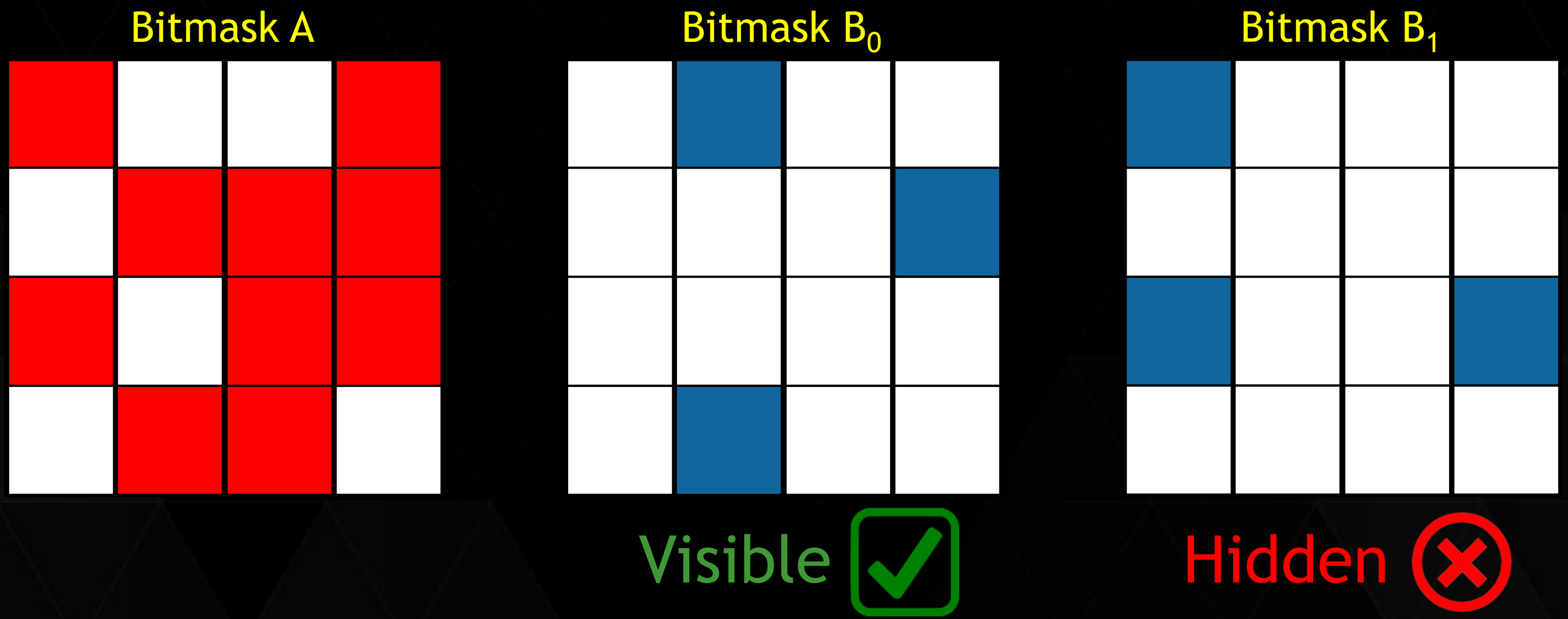
ADJUSTING PARAMETERS

- 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?



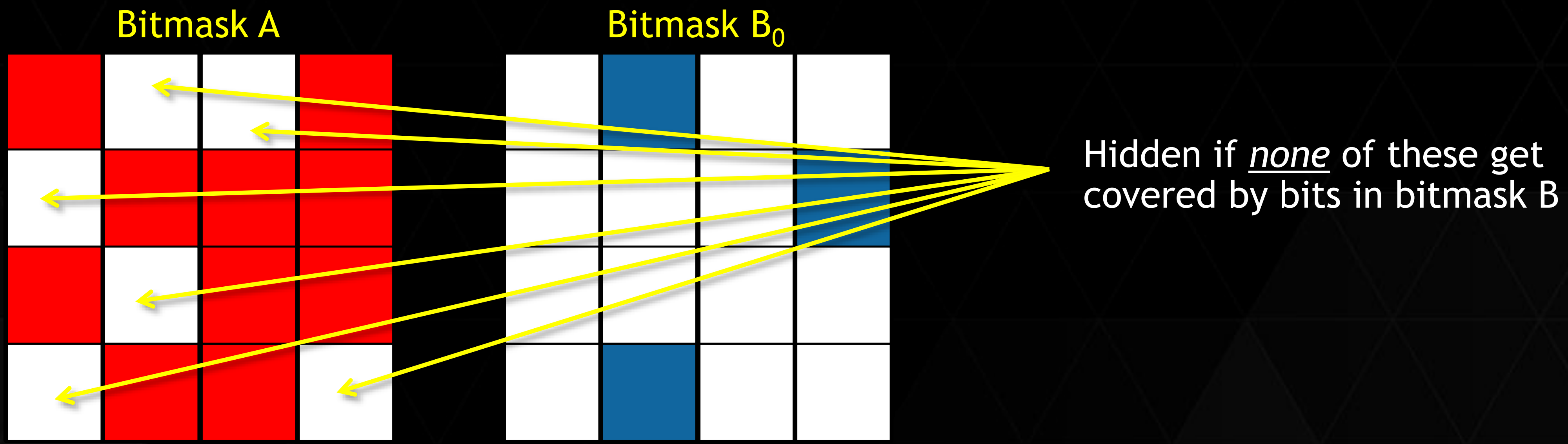
SLAB USING IMPLICIT COVERAGE

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



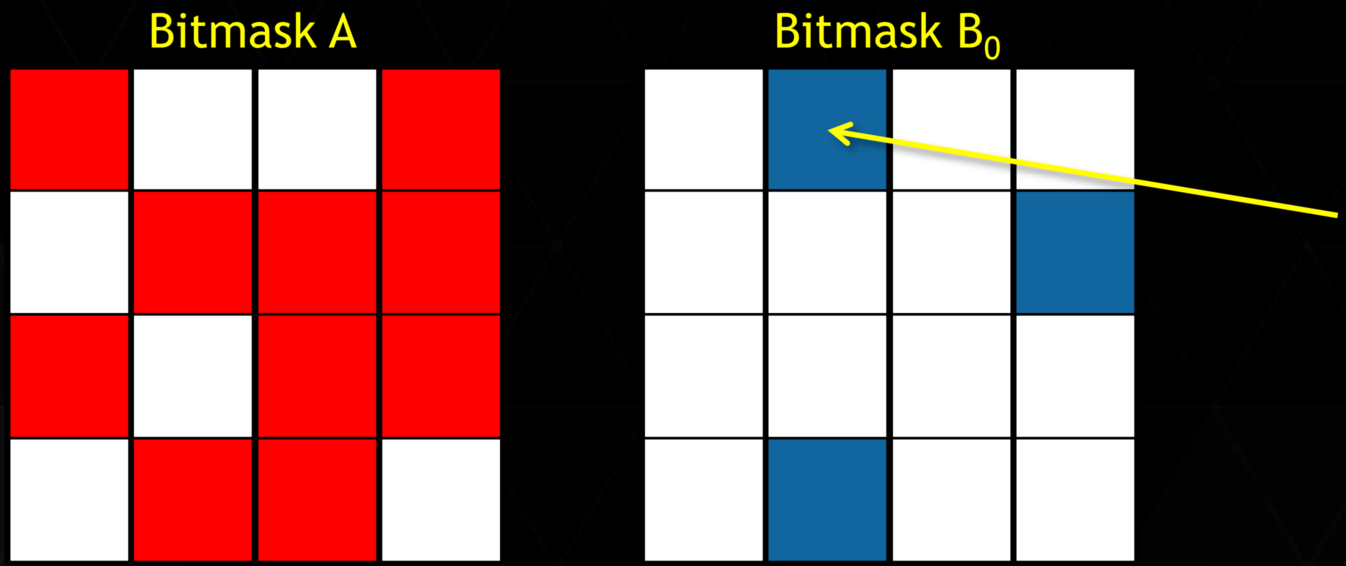
SLAB USING IMPLICIT COVERAGE

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



SLAB USING IMPLICIT COVERAGE

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



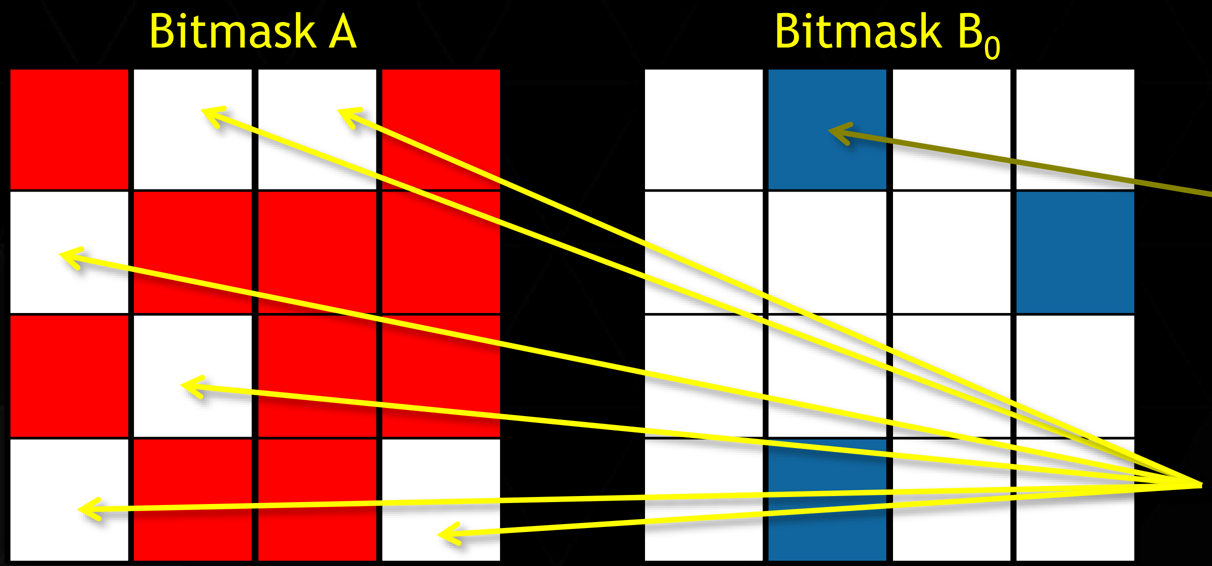
Naïve random sampling:

Covered with probability α_B

Uncovered with prob $(1 - \alpha_B)$

SLAB USING IMPLICIT COVERAGE

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



<u>Naïve random sampling:</u>

Covered with probability α_B

Uncovered with prob $(1 - \alpha_B)$

<u>*All*</u> uncovered with prob: $(1-\alpha_B)^6$

Bitmask B visible with prob: $1-(1-\alpha_B)^6$

SLAB USING IMPLICIT COVERAGE

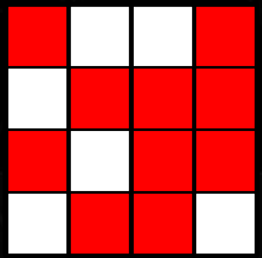
- Let's compute an insertion probability
  - Q: What's the chance random bitmask B is visible behind random bitmask A?

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

Or

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

Bitmask A



$\beta_A \equiv$ # 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?

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

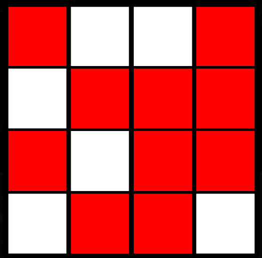
Or

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

$(1 - \alpha_B)$: prob of leaving 1 bit uncovered

$(b-\beta_A)$: number of bits that must be uncovered

Bitmask A



$\beta_A \equiv$ # bits covered

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

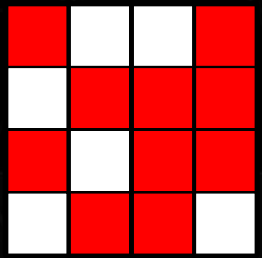
for b bits in bitmask

SLAB USING IMPLICIT COVERAGE

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

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

Bitmask A



$\beta_A \equiv$ # bits covered

- 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)

Stochastic Transparency [ESSL10]	k samples	stochastic	z-test, discard occluded	α-weighted average	coverage
Hybrid Transparency [MCTB13]	k layers	always	discard furthest	α-weighted average	alpha
(NEW) Stochastic Layered Alpha Blending	k layers	stochastic	discard furthest	α-weighted average	either‡

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 ↔ stoc transp continuum

Let’s demonstrate

FOLIAGE MAP

(From Epic's Unreal SDK)

FOLIAGE MAP

(From Epic's Unreal SDK)

FOLIAGE MAP

(From Epic's Unreal SDK)

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

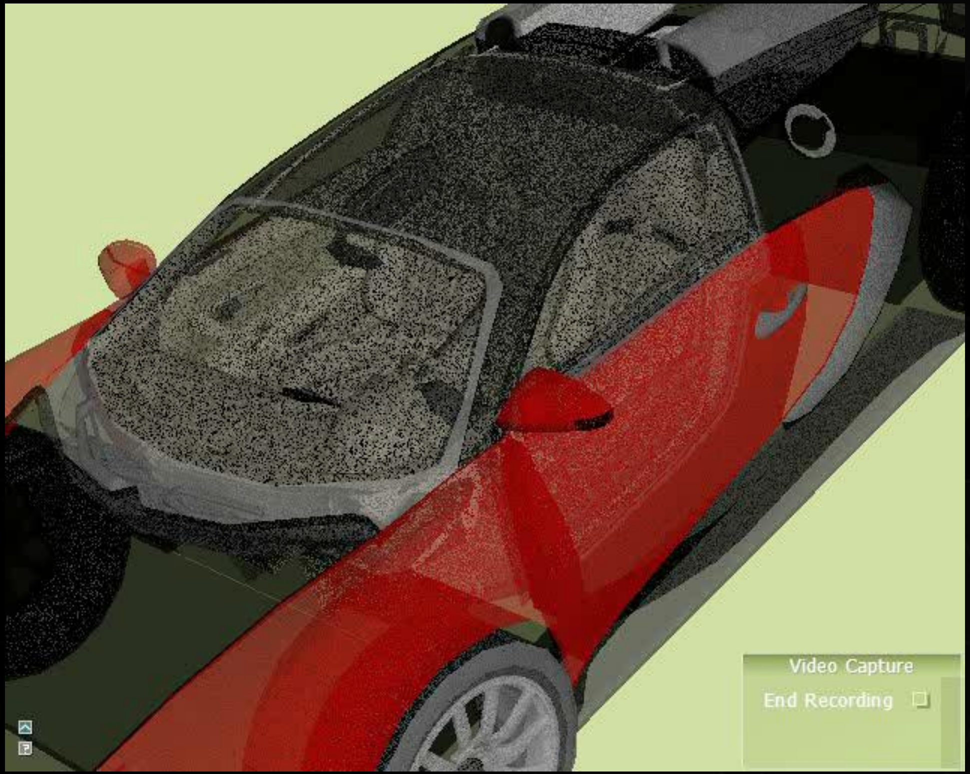
FOLIAGE MAP

(From Epic's Unreal SDK)

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

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

- 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

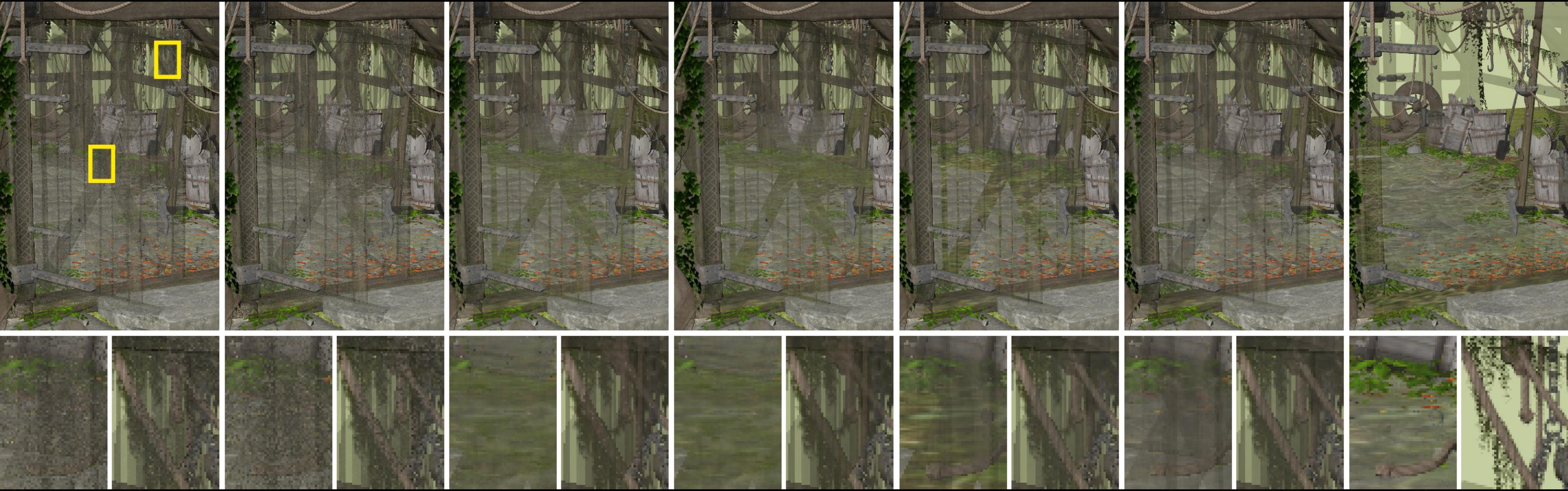
SUMMARY

- 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)

QUESTIONS?

E-mail: cwyman@nvidia.com, Twitter: _cwyman_

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