Lighting Research at Bungie

Hao Chen

Natalya Tatarchuk

Advances in Real-Time Rendering in 3D Graphics and Games,

Siggraph 2009, New Orleans, LA

Talk Outline

- Introduction
- Real-time Lighting
- Pre-computed Lighting

Pre-computed Global Illumination

Real-time Lighting in Games

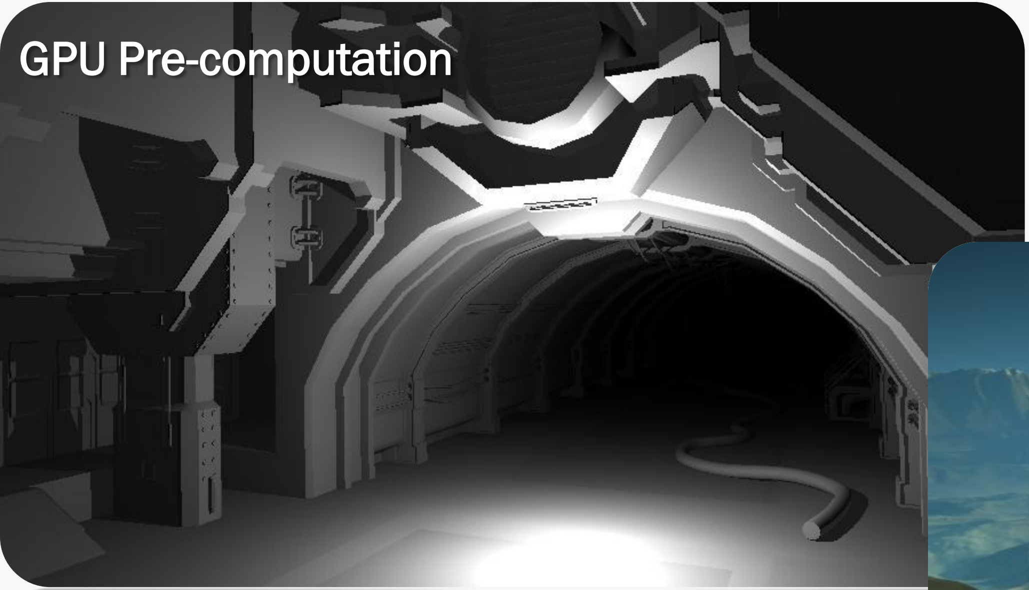
Trends

- Pipeline quality == graphics quality
- Artistic style over photo-realism
- Real time lighting is getting more GI
- GPGPU is tangible and real

R&D Focus

- Content Pipeline
- Artistic Vision And Style
- End-user Experience
- Scalable Technology

Two Research Directions



BUNGIE

Real-Time Lighting

Sky and Atmosphere

Previous Model

HALO 3

- [PSS99][PreethamHoffman03]
- Offline pre-computed sky texture
- Real-time scattering
- Single scattering only
- Viewable from ground only

Current Model

- [BrunetonNeyret2008]
- Single and multiple scattering
- Pre-computation on the GPU
- Viewable from space
- Light shafts

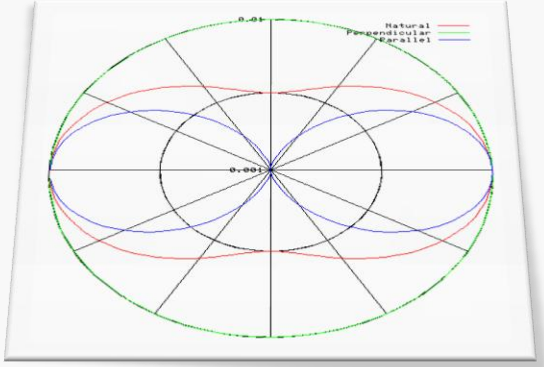
Raleigh Scattering

Raleigh Scattering

- Small particles scattering (air): $x = \frac{2\pi r}{\lambda}$ where $x << 1$
- Chromatic dependency:

$$\beta_R^S(h,\lambda) = \frac{8\pi^3(n^2-1)^2}{3N\lambda^4} e^{-\frac{h}{H_R}}$$

$$P_R(\mu) = \frac{3}{16\pi}(1+\mu)^2 \quad \text{where} \quad \mu = \cos\Theta$$



[Elek08]

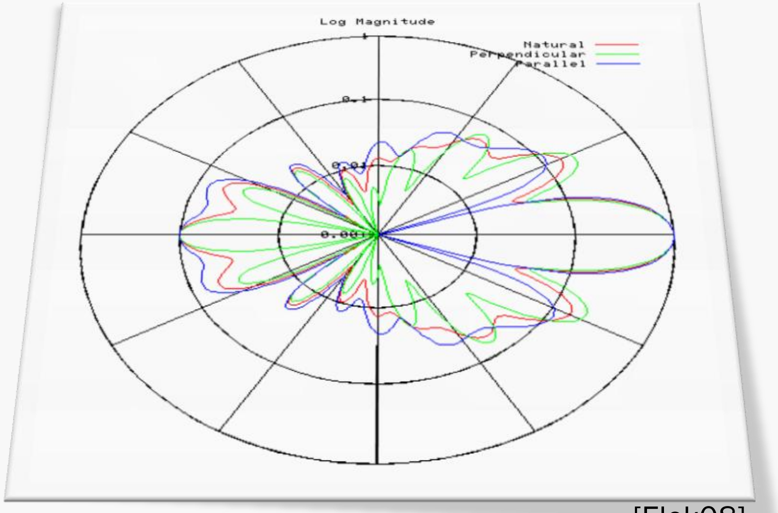
- Depends on altitude, wavelength, molecular density at sea level, and atmospheric density

Mie Scattering

Mie Scattering

- Light scattering on larger particles
  - Achromatic – λ-independence
- Phase function is strongly anisotropic
- Analytical approximation by Cornette-Shanks:

$$x \geq 1$$



[Elek08]

$$\beta_M^S(h,\lambda) = \beta_M^S(0,\lambda)e^{-\frac{h}{H_M}}$$

$$P_M(\mu) = \frac{3}{8\pi}\frac{(1-g^2)(1+\mu^2)}{(2+g^2)(1+g^2-2g\mu)^{3/2}}$$

Rendering Equation for the Atmosphere

$$L(\mathbf{x},\mathbf{v},\mathbf{s}) = \left(L_0 + R[L] + S[L]\right)(\mathbf{x},\mathbf{v},\mathbf{s})$$

- *x* – viewer, *v* – view direction, *s* – sun direction
- Account for:
  - Direct sun light L_0
  - Reflected light at point being shaded (x_0) $R[L]$
  - Inscattered light $S[L]$ (toward the viewer)
- Accurate solution is non-trivial to compute in real-time still

Direct Sun Light Computation

$$L_0(\mathbf{x},\mathbf{v},\mathbf{s}) = T(\mathbf{x},\mathbf{x}_0)L_{sun}, \text{ or } 0$$

- Direct sunlight is attenuated by transmittance function before reaching the viewer
- Accounts for occlusions

Reflected Light

$$R[L](\mathbf{x},\mathbf{v},\mathbf{s}) = T(\mathbf{x},\mathbf{x}_0)I[L](\mathbf{x}_0,\mathbf{s})$$

- Reflected light is attenuated by the transmittance
- Depends on the light $I[L]$ reflected at x_0
- Reflected light is null on the top atmosphere boundary

Inscattered Light

$$S[L](\mathbf{x},\mathbf{v},\mathbf{s}) = \int_{\mathbf{x}}^{\mathbf{x}_0} T(\mathbf{x},\mathbf{y}) J[L](\mathbf{y},\mathbf{v},\mathbf{s}) dy$$

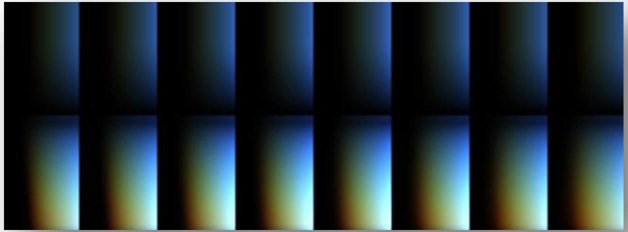
- Light scattered towards the viewer between the point being shaded and the viewer
- Depends on the transmittance *T* and the radiance *J* of light scattered toward the viewer

Pre-computation

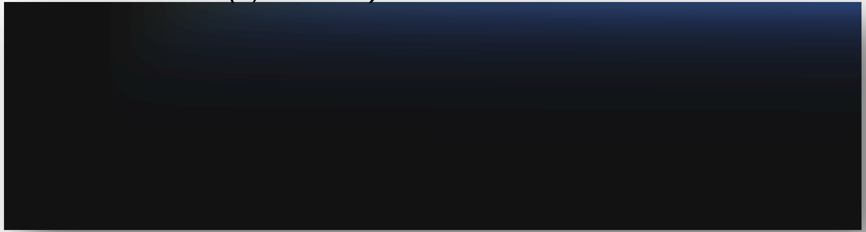
- Store pre-computed look-up tables as textures
- Use GPU to generate the textures

Transmittance (r, mu)

Inscatter (r, mu, muS, nu)

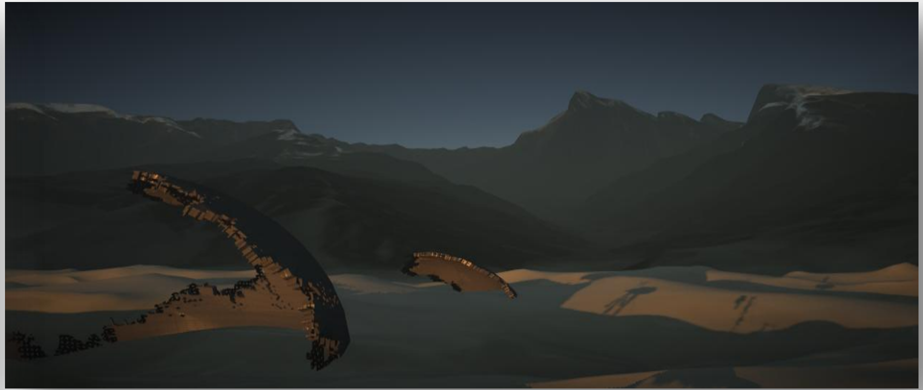


Irradiance (r, muS)



Different Atmospheres

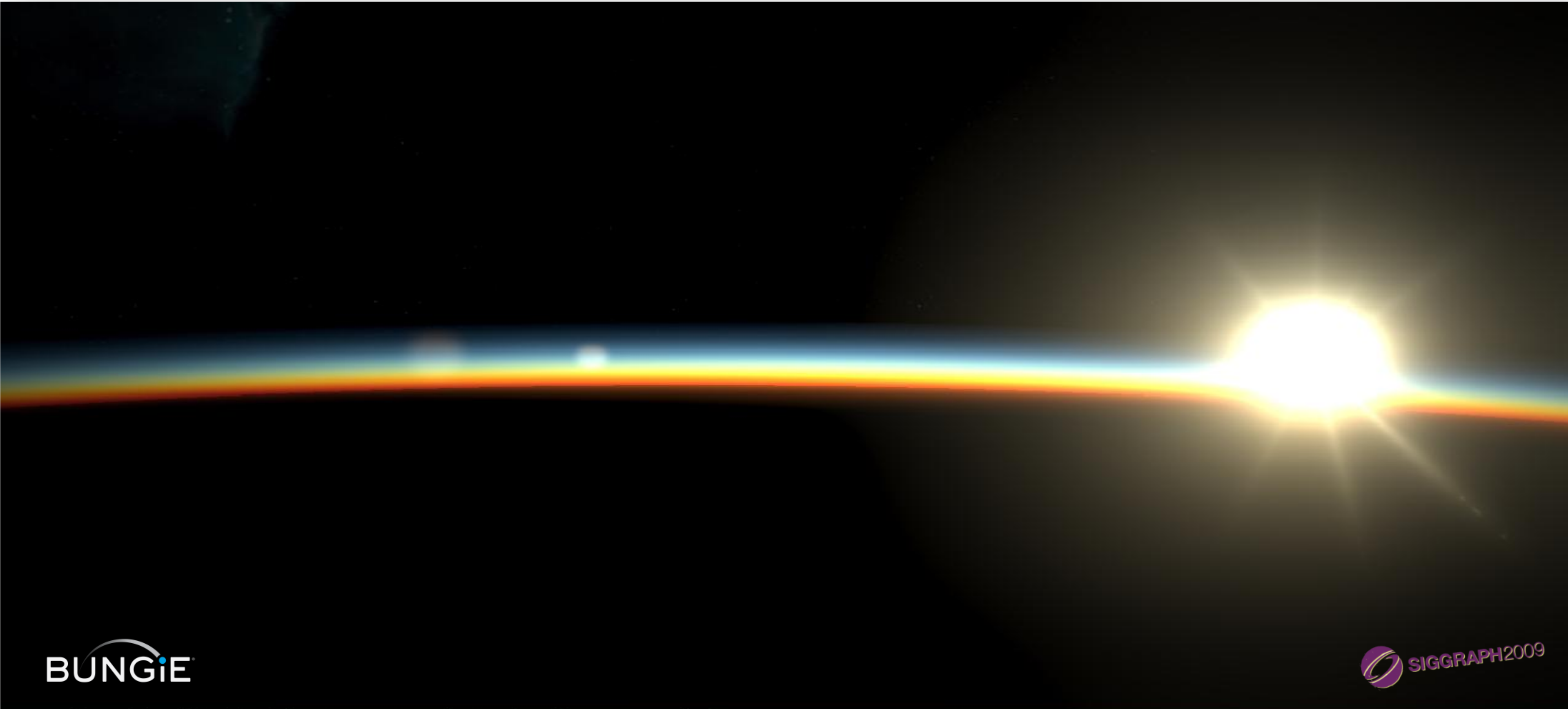
Time Of Day



BUNGIE

SIGGRAPH2009

Atmosphere Seen From Space



Sky Light

- [BrunetonNeyret2008] used a single color for sky irradiance
  - For distant mountains / objects, just use that
- Better approximation for close-up geometry:
  - Use CIE sky luminance distribution
  - Scale by the pre-computed irradiance
  - Project to SH per azimuth angle
  - Fit the coefficients with a polynomial
  - Render with PRT for GI look

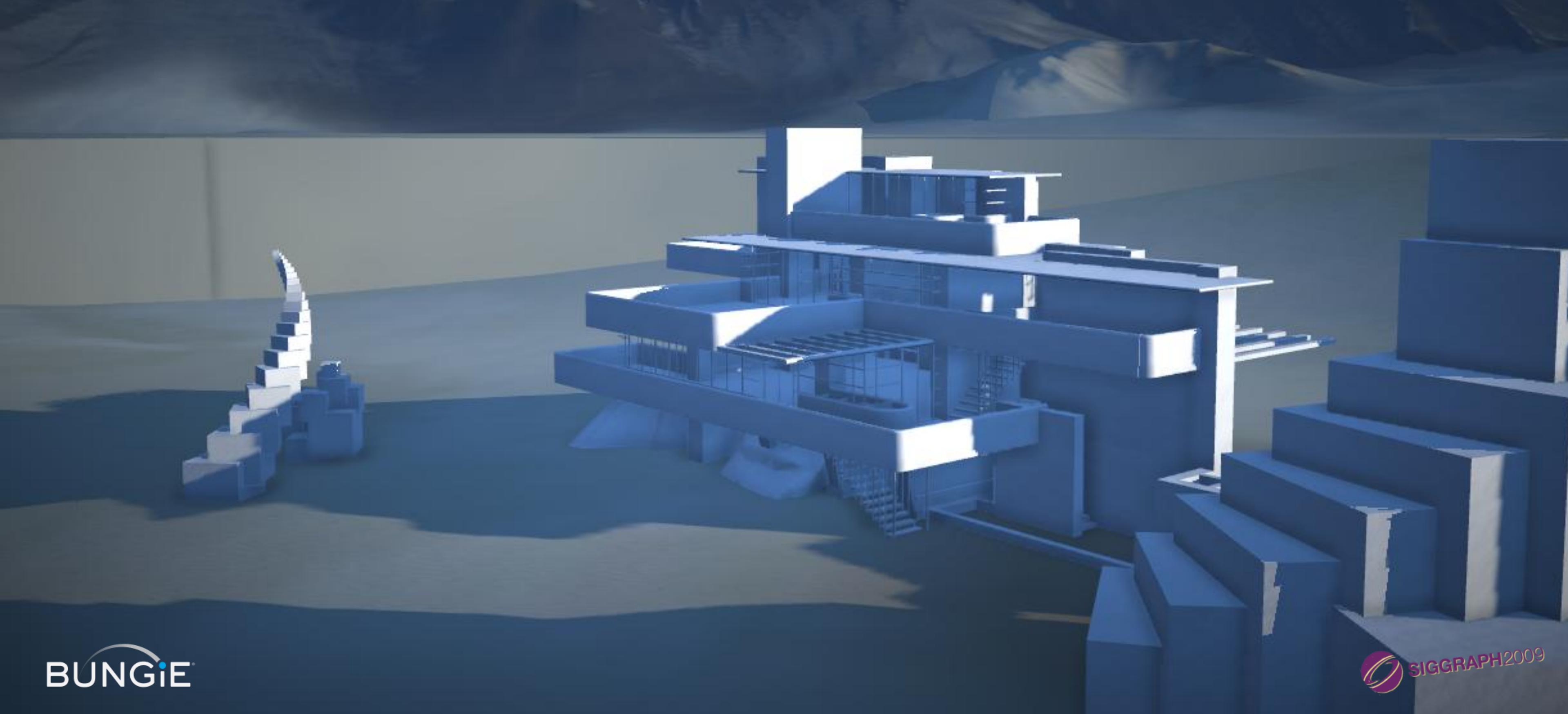
CIE Standard Luminance Distribution

Table 1. Standard parameters

Type	Gradation	Indikatrix	*a*	*b*	*c*	*d*	*e*	Description of luminance distribution
1	I	1	4.0	-0.70	0	-1.0	0.00	**CIE Standard Overcast Sky, alternative form** Steep luminance gradation towards zenith, azimuthal uniformity
2	I	2	4.0	-0.70	2	-1.5	0.15	Overcast, with steep luminance gradation and slight brightening towards the sun
3	II	1	1.1	-0.8	0	-1.0	0.00	Overcast, moderately graded with azimuthal uniformity
4	II	2	1.1	-0.8	2	-1.5	0.15	Overcast, moderately graded and slight brightening towards the sun
5	III	1	0.0	-1.0	0	-1.0	0.00	**Sky of uniform luminance**
6	III	2	0.0	-1.0	2	-1.5	0.15	Partly cloudy sky, no gradation towards zenith, slight brightening towards the sun
7	III	3	0.0	-1.0	5	-2.5	0.30	Partly cloudy sky, no gradation towards zenith, brighter circumsolar region
8	III	4	0.0	-1.0	10	-3.0	0.45	Partly cloudy sky, no gradation towards zenith, distinct solar corona
9	IV	2	-1.0	-0.55	2	-1.5	0.15	Partly cloudy, with the obscured sun
10	IV	3	-1.0	-0.55	5	-2.5	0.30	Partly cloudy, with brighter circumsolar region
11	IV	4	-1.0	-0.55	10	-3.0	0.45	White-blue sky with distinct solar corona
12	V	4	-1.0	-0.32	10	-3.0	0.45	**CIE Standard Clear Sky, low illuminance turbidity**
13	V	5	-1.0	-0.32	16	-3.0	0.30	**CIE Standard Clear Sky, polluted atmosphere**
14	VI	5	-1.0	-0.15	16	-3.0	0.30	Cloudless turbid sky with broad solar corona
15	VI	6	-1.0	-0.15	24	-2.8	0.15	White-blue turbid sky with broad solar corona

Direct Illumination Only

CIE Sky Illumination in SH



Sky Light with PRT

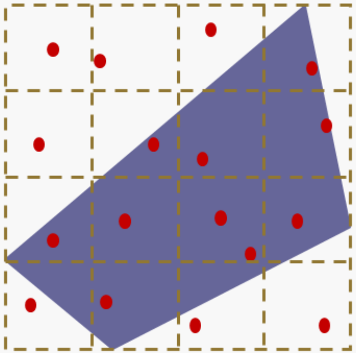
Shadows

Shadow Mapping in Games

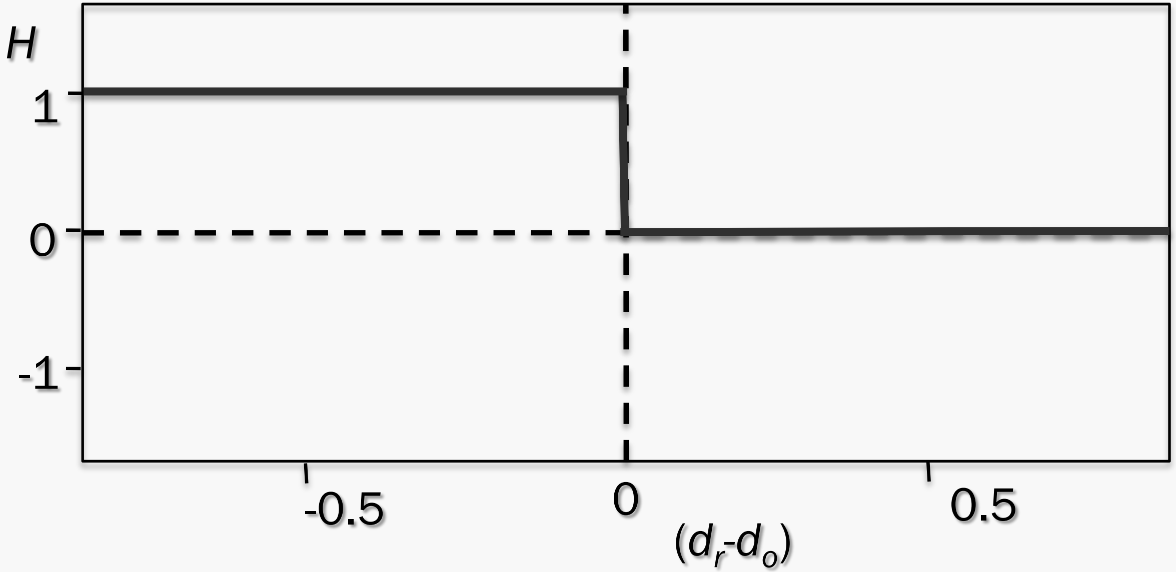
- Shadow mapping is now fairly common in latest video games
- A number of practical production issues remain for high quality stable shadows:
  - Managing aliasing due to resolution and projection
- Open-world scenarios now frequently resort to a variant of cascade shadow mapping
  - Used for resolution management
  - Unfortunately, cascading doesn't solve projection, or sampling, aliasing artifacts

Sampling Aliasing

- Currently, sampling approached are typically resolved via PCF [Reeves et al. 1987] for soft shadows results
  - Filter shadow test results
  - Often combined with a rotated Poisson disk filter
- Expensive at run-time
  - Requires a lot of samples to hide visible structure patterns
  - Linear in cost in terms of # of samples



Shadow Mapping [RSC87]



- Heaviside step function: $H(d_r - d_o)$ where d_r is the receiver depth, and d_o is the occluder depth.
- 1 means no shadows (fully lit) and 0 means completely in shadow.

Shadow Prefiltering

- **Linearly filterable shadow test**
  - Reformulate shadow filtering test to support pre-filtering
- **A number of recent techniques designed to address this:**
  - Variance Shadow Maps [Donnelly / Lauritzen 06]
  - Convolution Shadow Maps [Annen et al 2007]
  - Exponential Shadow Maps [Annen et al 2008] [Salvi 2008]

BUNGiE

SIGGRAPH2009

Shadow Test Reformulation

- Separate the terms for occluder and receiver
  - Thus we can pre-filter occluder terms with hardware mipmapping and with image-space blurs for soft shadows
- Depth bias no longer necessary to alleviate ‘shadow acne’
  - Due to the changed shadow test

Probabilistic Shadow Test

- Inspired by the Deep Shadow Maps [LocovicVeach2000]
- Probability that a given sample is in shadow, given current receiver & occluder depths

$$f(d_r) = \Pr(d_o \geq d_r)$$

- d_o becomes a random variable
    - Represents the *occluder depth distribution* function
- d_r is the current receiver depth

[LocovicVeach00]

Variance-Based Shadow Test

- Binary test becomes a probability distribution function
  - Probability current fragment is in shadow
- $\Pr(d_o \geq d_r)$ is derived from two moments:

  $\mu = E(d_o)$ and $\sigma^2 = E(d_o^2) - E(d_o)^2$

Variance-Based Shadow Test

- Use Chebyshev's inequality as upper bound for the test:

$$\Pr(d_o \geq d_r) \leq p_{\max}\left(d_r\right) \equiv \frac{\sigma^2}{\sigma^2 + \left(\mu - d_r\right)^2}$$

BUNGIE

SIGGRAPH2009

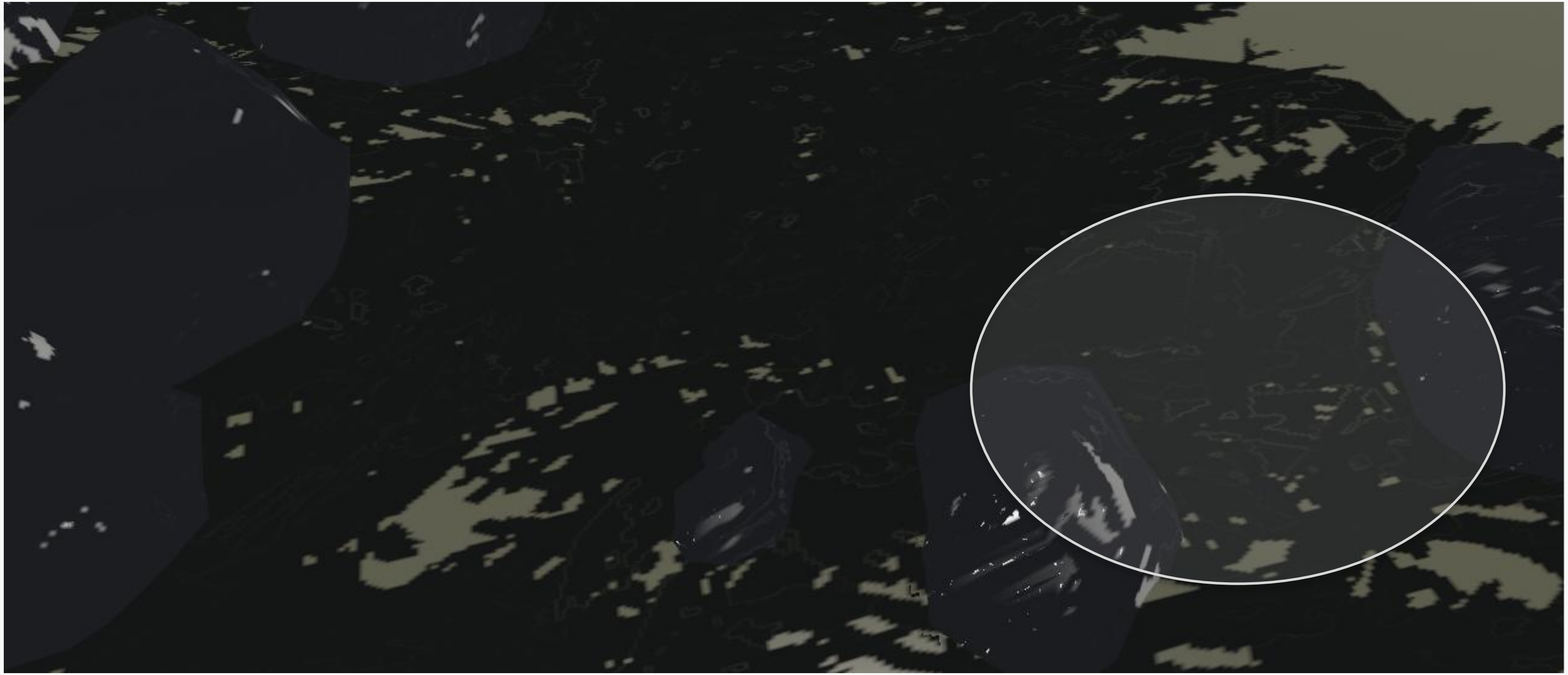
Variance Shadow Map Approach Pros

- Image-space & hardware filtering for soft shadows
- Alleviates depth bias artifacts for polygons that span depth ranges
  - Especially when filtering

Variance Shadow Map Approach Cons

- Twice the memory of the regular shadow map
- Light bleeding in areas of high depth complexity
- Exacerbated by filtering with large kernels
  - Variance is increased with large blurs

Variance Shadow Maps: Light Bleeding



BUNGIE

SIGGRAPH2009

Light Bleeding Fix-up

- All shadow test results below some minimum variance p_{min} get clamped to 0
- The rest of the range rescaled to [0..1]
- Removes light bleeding
  - But similarly to dilation, this ‘fattens’ up shadows
  - Especially when applying large blurs

Can We Do Better?

- Two moments simply do not provide enough information to fully reconstruct the shadow test
  - We don't know the distribution function a priori
- Recall that n^{th} moment can be expressed as

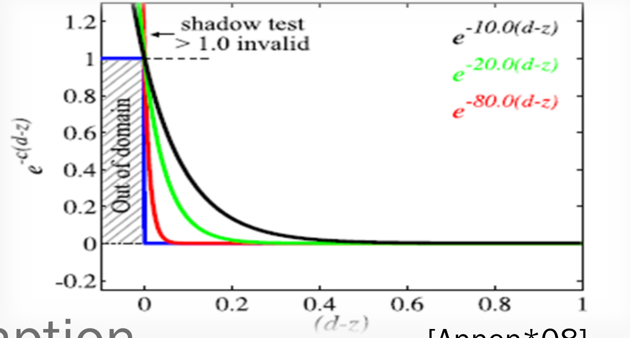
$$\mu_n = E\left[x^n\right] = \frac{1}{N}\sum_{i=1}^{N} x_i^n$$

- However, we don't want to just render n moments
  - 2 channels of 16F or 32F textures is hurtful enough

Exponential Shadow Map Test

- Assume $d_r \geq d_o$
- Shadow test becomes $f(d_o, d_r) = \lim_{\alpha \to \infty} e^{-\alpha(d_o, d_r)}$
- Approximate by using a large positive constant *c:*

$$f(d_o, d_r) = e^{-c(d_o - d_r)}$$



[Annen*08]

- Clamp result to [0..1] range to ensure correct results
  - Fixes up some regions where the assumption does not hold

BUNGIE SIGGRAPH2009

Exponential Shadow Map Prefiltering

- Separate terms which depend on occluder and receiver depths: $f(d_o, d_r) = e^{-c(d_o - d_r)} = e^{-cd_o} e^{cd_r}$

$$f(d_o, d_r) = f(d_o) f(d_r)$$

- Convolving $f(d_o, d_r)$ with a filter kernel w:

$$w \cdot f(d_o, d_r) = w \cdot \left(e^{-c(d_o - d_r)}\right) = \boxed{w \cdot e^{-cd_o}} \cdot e^{cd_r}$$

- Allows filtering of only the occluder terms == prefiltering

Exponential Shadow Map Benefits

1. Extremely easy to implement:
   a) Render the exponential of occluder depth
   b) Prefilter
   c) Using mip maps and/or applying separable Gaussian blurs
   d) Reconstruct ESM test at run-time

ESM Shadow Test Computation

```
float ComputeESM( float2 vShadowMapUVs, float fReceiverDepth,
                  float fCascadeIndex )
{
    // Filtered look up using mip mapping
    float fOccluderExponential = tCascadeShadowMaps.Sample(
                                 sShadowLinearClamp,
                                 float3(vShadowMapUVs,fCascadeIndex)).r;
   float fReceiverExponential = exp( -fESMExponentialMultiplier *
                                     fReceiverDepth );
   float fESMShadowTest = fOccluderExponential * fReceiverExponential;
   return saturate(fESMShadowTest);
}
```

Exponential Shadow Map Benefits

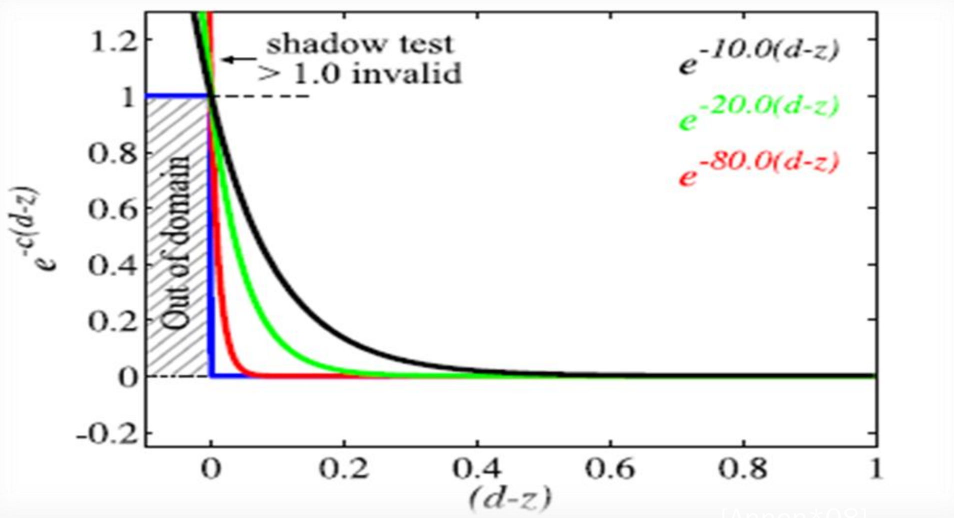
2. Solves biasing problems ('shadow acne') that exist with regular shadow maps
3. Excellent soft shadows visual results with even small filters
   a) For example, a 5x5 separable Gaussian

Exponential Shadow Map Benefits

4. Only uses a single channel texture
5. Deals well with scene depth complexity
   - Not based on variance
   - Thus light bleeding due to depth variance doesn't show up
   - Doesn't get exacerbated with wider filter kernels

Thought We're Done?

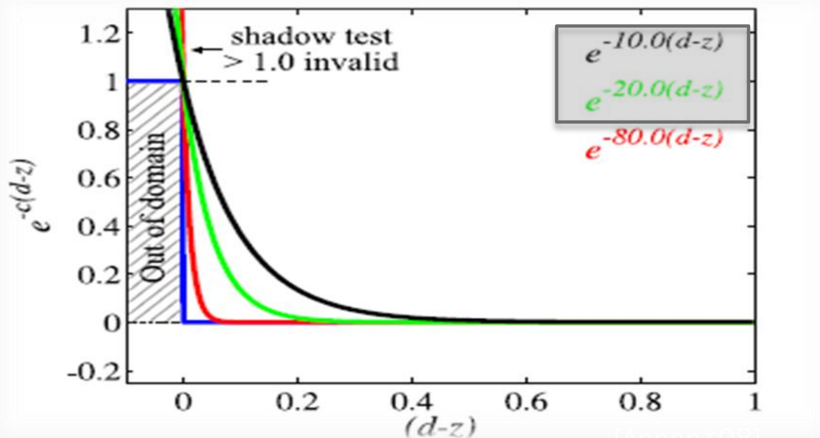
- Not yet, unfortunately.
- Let's look at the shadow test again:



[Annen*08]

Thought We're Done?

- Small values for *c* only work in scenes with low depth complexity
- Otherwise we see a lot of light leaking artifacts

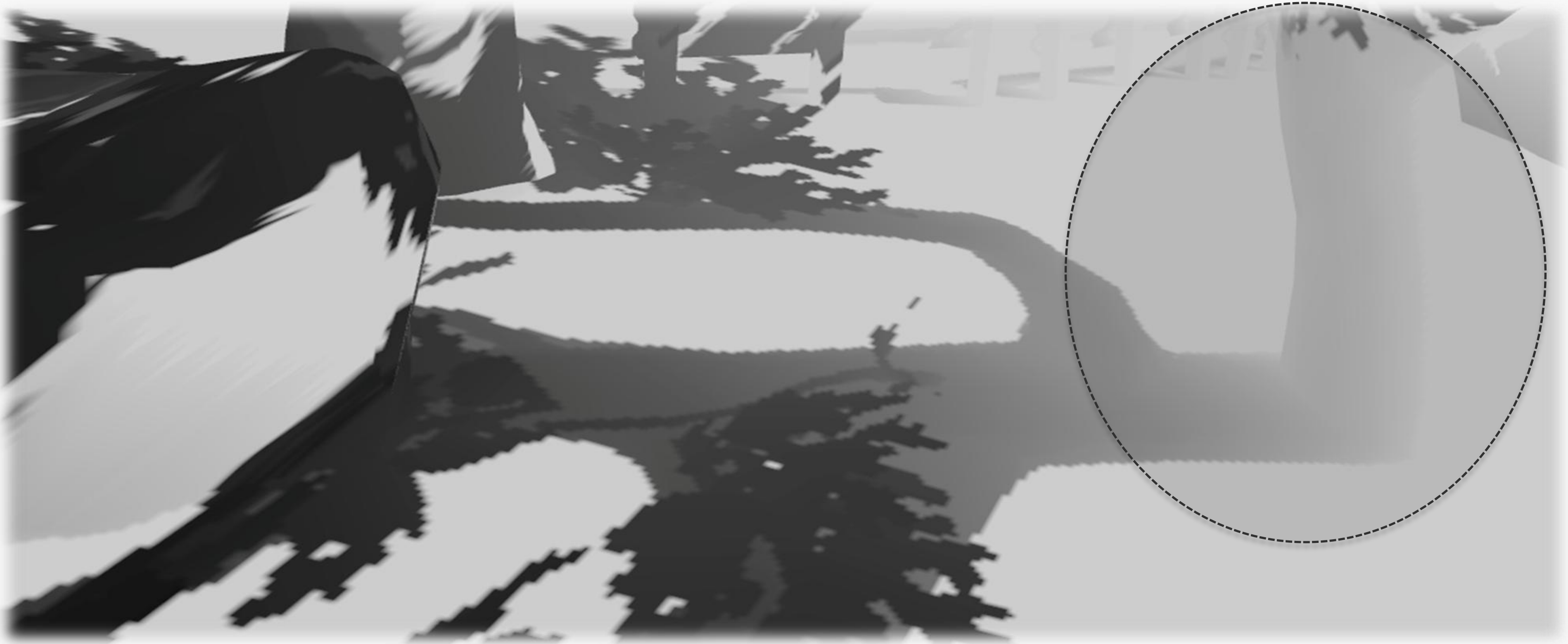


[Annen*08]

Thought We’re Done?

- However, larger values of *c* such as *c* = 80 demand high precision floating-point buffers
  - c ~= 88 is the maximum value for 32F; otherwise overflow

ESM Light Leaking Example



ESM Logarithmic Space Filtering

- Render linear depth instead of the exponential
- Filter in log space
- Let's expand the filtering operation on occluder depths:

$$w \cdot f(d_o) = \sum_{i=0}^{N} w_i e^{cd_{o_i}} = w_0 e^{cd_{o_0}} + w_1 e^{cd_{o_1}} + \ldots + w_N e^{cd_{o_N}}$$

ESM Logarithmic Space Filtering

- For 3 samples, we have:

$$w_0 e^{cd_{o_0}} + w_1 e^{cd_{o_1}} + w_2 e^{cd_{o_2}} =$$

$$e^{cd_{o_0}}\left(w_0 + w_1 e^{c\left(d_{o_1} - d_{o_0}\right)} + w_2 e^{c\left(d_{o_2} - d_{o_0}\right)}\right)$$

- Since $e^{\ln p} = p$ we can write:

$$w \cdot f\left(d_o\right) = e^{cd_{o_0}} e^{\ln\left(w_0 + w_1 e^{c\left(d_{o_1} - d_{o_0}\right)} + w_2 e^{c\left(d_{o_2} - d_{o_0}\right)}\right)}$$

BUNGIE

SIGGRAPH2009

ESM Logarithmic Space Filtering

- Generalizing to N samples:

$$w \cdot f(d_o) = e^{c d_{o_0}} e^{\ln\left(w_0 + \sum_{i=1}^{N} w_i e^{c\left(d_{o_i} - d_{o_0}\right)}\right)}$$

- This replaces the standard Gaussian or box filter summation
  - Weights are from the Gaussian filter kernel
  - Instead of regular summation, compute the result above, summing over the samples

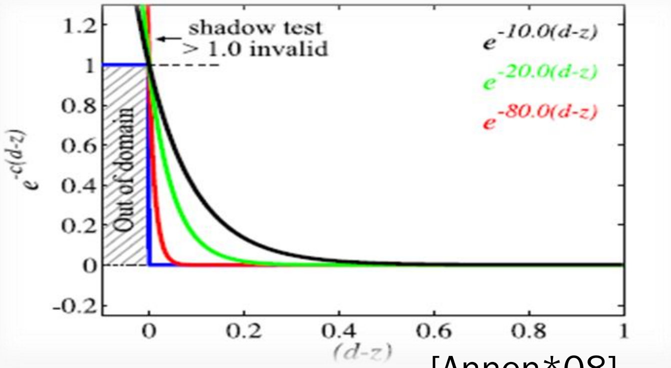
ESM Logarithmic Space Filtering

- Allows us to use 16F texture format with high values for *c*
  - During the actual filtering operation we have at least 24 bit precision (on consoles) and 32 bit on most recent PC hardware
- Every little bit helps
  - Pun intended!

Thought We're Done?™

- Furthermore, ESM shadow test has the following limitation:

$$f(d_o, d_r) = \lim_{\alpha \to \infty} e^{-\alpha(d_o, d_r)}$$



[Annen*08]

- As $d_o \to d_r$, $f(d_o, d_r) \to 1$
- Thus we see *contact light leaking* with ESM
  - In places where the occluder is near the receiver
  - Turns out this is a fairly frequent occurrence

Contact Leaking Reduction

- A brute-force solution is to over-darken the results of shadow test based on occluder-receiver proximity

```
// Filtered look up using bilinear filtering
float fOccluderExponential =
     tCascadeShadowMaps.Sample( sShadowLinearClamp,  float3( vShadowMapUVs, fCascadeIndex ) ).r;

float fUnfilteredOccluderDepth =
   tCascadeDepthBuffers.SampleLevel( sShadowPointClamp, float3( vShadowMapUVs, fCascadeIndex ), 0 ).r;

float fReceiverExponential = exp( -fESMExponentialMultiplier * fReceiverDepth );
float fESMShadowTest       = saturate( fOccluderExponential * fReceiverExponential );

if (fUnfilteredOccluderDepth < fReceiverDepth )
{
   const float fDarkeningAmount = 0.05;
   fESMShadowTest *= fDarkeningAmount;
}
```

ESM Over Darkening

- That works fine – so long as we do not prefilter shadows

BUNGIE

SIGGRAPH2009

ESM Over Darkening with Filtering

- Results in “fat & stylized shadows”

Cascade Shadow Maps & Prefiltered Shadow Formulations

- At first glance, cascade shadow maps are orthogonal to prefiltered shadow maps
  - One manages shadow map resolution, the other – filtering / sampling
- However, in practice we encounter the need for additional fix-ups for using VSM / ESM with cascades
  - Specifically with regards to selection of cascade frustum

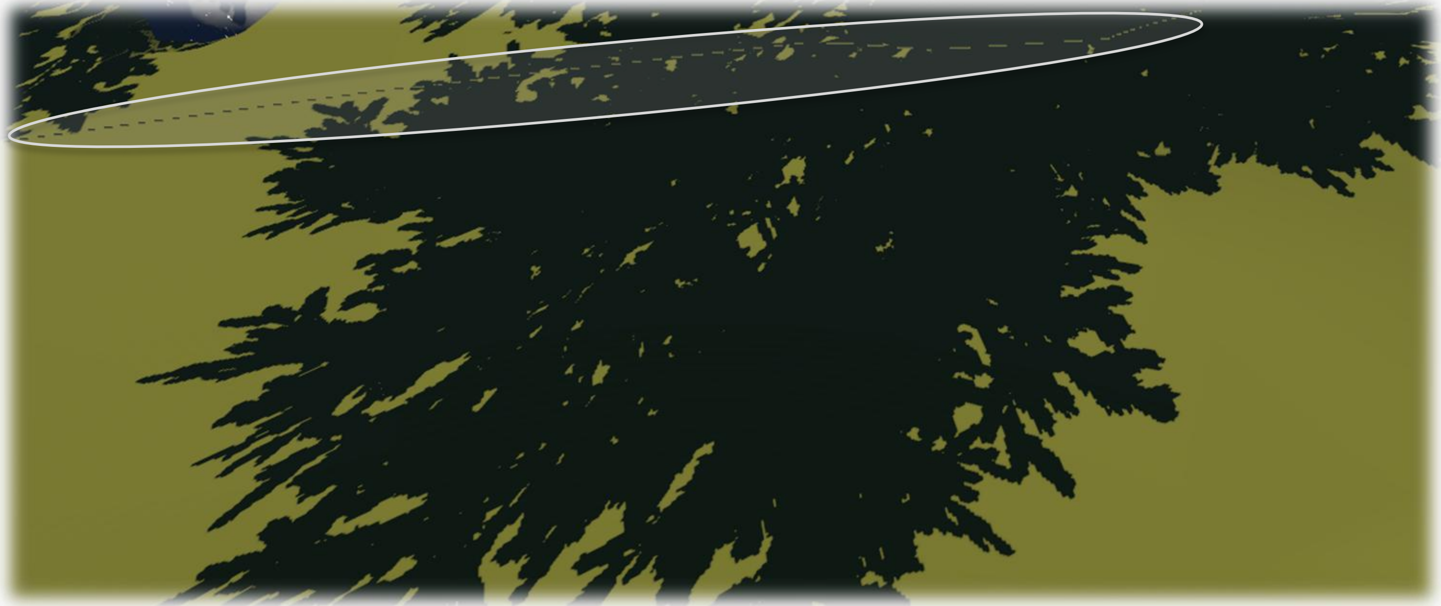
Typical Cascade Frustum Selection

```
int GetInitialFrustumIndex( float3 vPositionWS )
{
   float fPosZ = -mul( mCascadeViewMatrix, float4(vPositionWS,1.0f)).z;
   int nFrustumIndex= 0;
   if ( fPosZ <= vFarBounds[0] )
   {
      nFrustumIndex = 0;
   }
   else if ( fPosZ <= vFarBounds[1] )
   {
      nFrustumIndex = 1;
   }
   else if ( fPosZ <= vFarBounds[2] )
   {
      nFrustumIndex = 2;
   }
   else
   {
      nFrustumIndex = 3;
   }
   nFrustumIndex = min( nFrustumIndex, NUM_CASCADES );
   return nFrustumIndex;
}
```

Prefiltered Shadow Cascade Selection

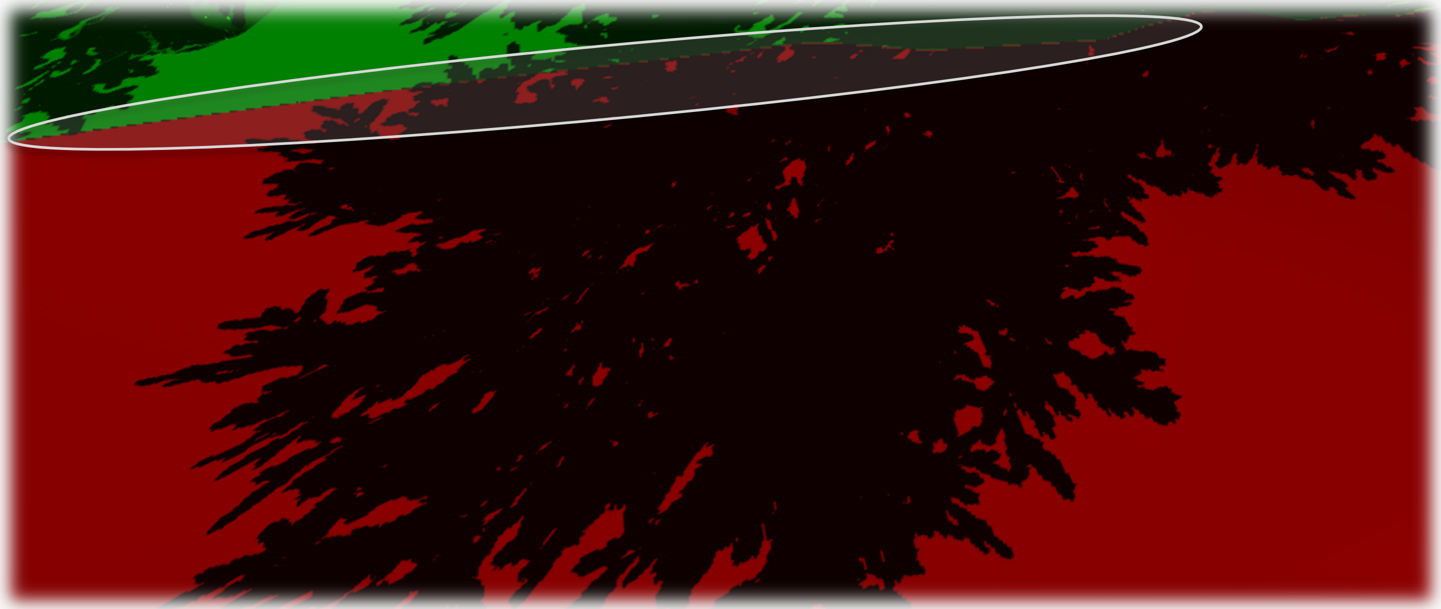
- Need to make sure that every fragment in a pixel quad chooses the same cascade frustum
- This is required so that derivatives are meaningful and mip selection is correct
  - Necessary for ESM / VSM whenever we use mip mapping
- Want to select the same frustum index for all fragments in the same quad

Artifacts Due to Incorrect Cascade Selection with Prefiltered Shadows



A ”traveling” line of ‘flipped’ shadow test result along the boundary of cascade frustums

Artifacts Due to Incorrect Cascade Selection with Prefiltered Shadows



A ”traveling” line of ‘flipped’ shadow test result along the boundary of cascade frustums

Prefiltered Shadow Cascade Selection

```
float4 ComputePrefilteredCascadesShadowPositionAndFrustumIndex ( float3 vPosWS )
{
   int nFrustumIndex = GetInitialFrustumIndex( vPositionWS );
   const int aLog2LUT[8] = { 0, 1, 1, 2, 2, 2, 2, 3 };
   int n2PowFrustumIndex  = 1 << nFrustumIndex;
   // Now determine the difference across pixels in the quad:
   int nFrustumIndexDX     = abs( ddx( n2PowFrustumIndex ));
   int nFrustumIndexDY     = abs( ddy( n2PowFrustumIndex ));
   int nFrustumIndexDXDY   = abs( ddx( nFrustumIndexDY   ));
   // This quantity will be _the same_ for all pixels across the quad,
   // which is what allows us to consistently select frustum index for
   // all pixels in the quad:
   int nMaxDifference  = max( nFrustumIndexDXDY, max( nFrustumIndexDX,
                                                      nFrustumIndexDY ) );
   // If the derivatives are zero across the quad, we can simply use the original
   // frustum index. If there are differences, we will recover the desired
   // frustum index by looking up into the log table:
   nFrustumIndex = nMaxDifference > 0 ? aLog2LUT[nMaxDifference-1]:nFrustumIndex;
   return ComputeCascadeSamplingParameters( vPositionWS, nFrustumIndex )
}
```

Let's Fix Contact Leaking – Round 2

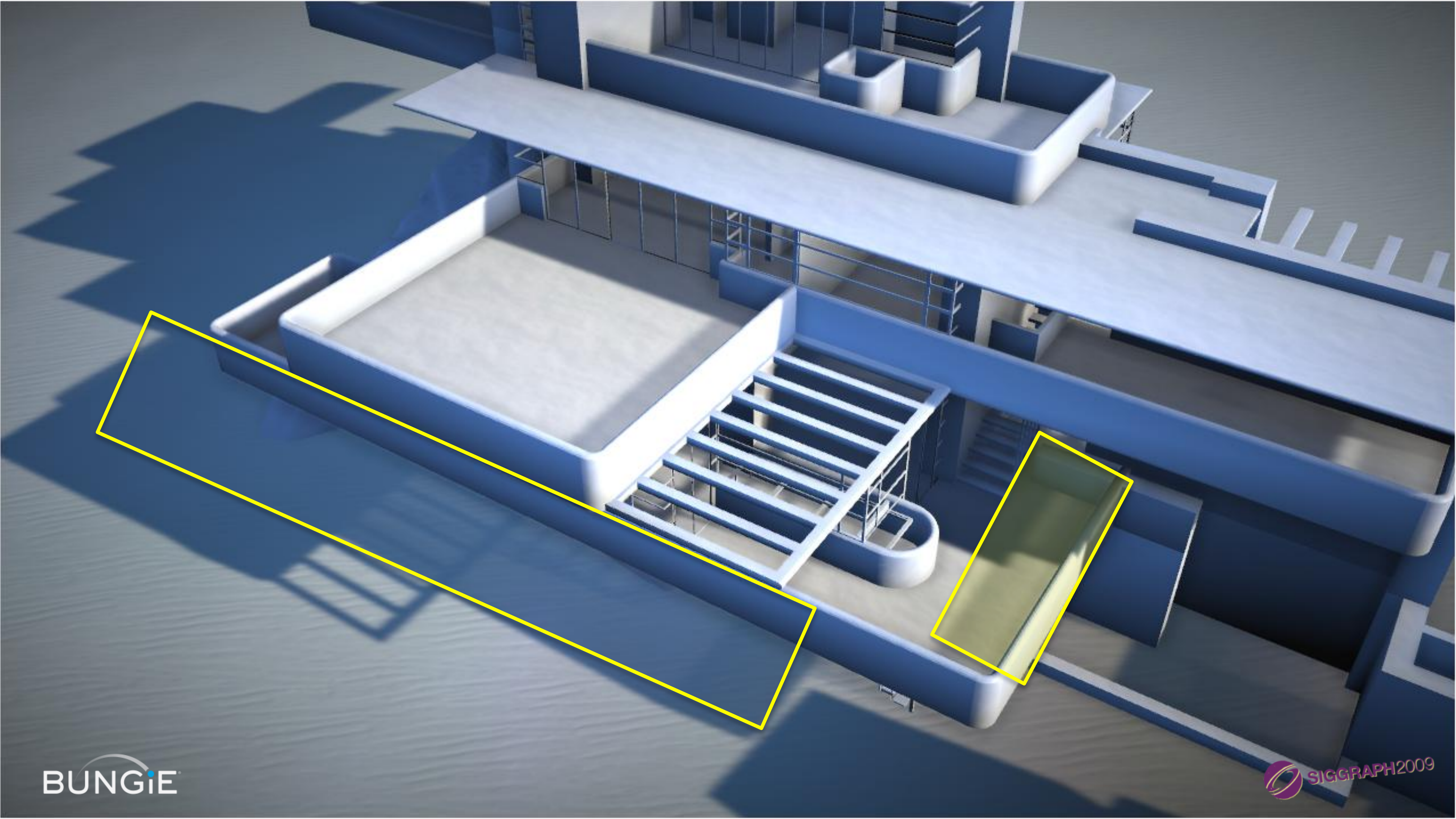
- **Another thing we can try is to have tighter depth range for each cascade**
  - Clamp the depth / z range to the bounding volume of the cascade frustum in light space
  - What happen to occluders outside the bounds?

Let’s Make Pancakes –
Shadow Pancakes, Of Course!

- As we clamp, the occluders *outside* of the bounding volume are flattened onto the near / far plane of the frustum bounding box
  - Aka the ‘shadow pancakes’

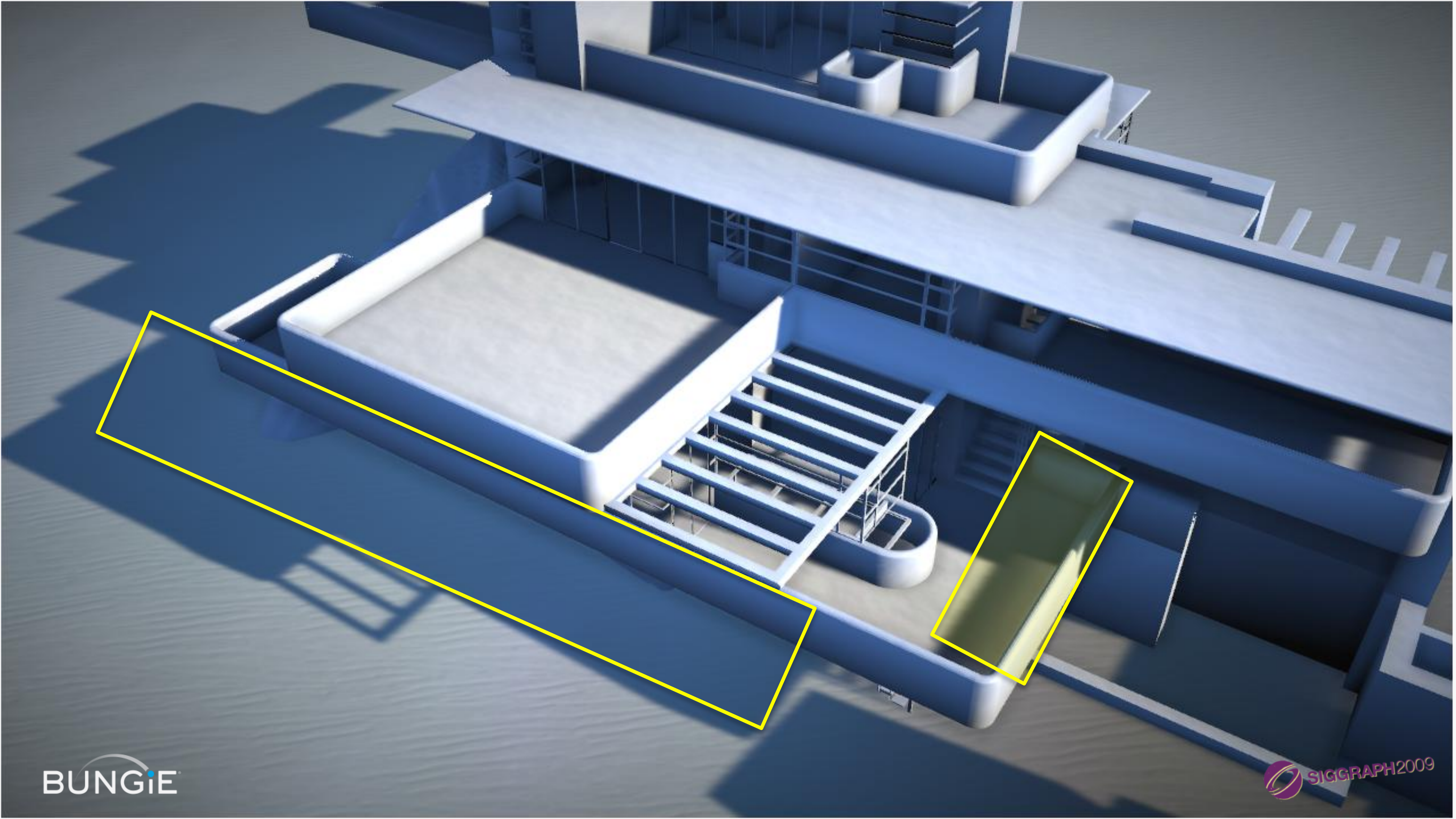
Let’s Make Pancakes –
Shadow Pancakes, Of Course!

- When the occluder object is outside the viewing frustum we don’t care about the actual depth of the occluder
- Just need to know its effect on the rest of the scene
  - Is it going to shadow the objects within the cascade frustum?
  - Can’t see these occluders any way



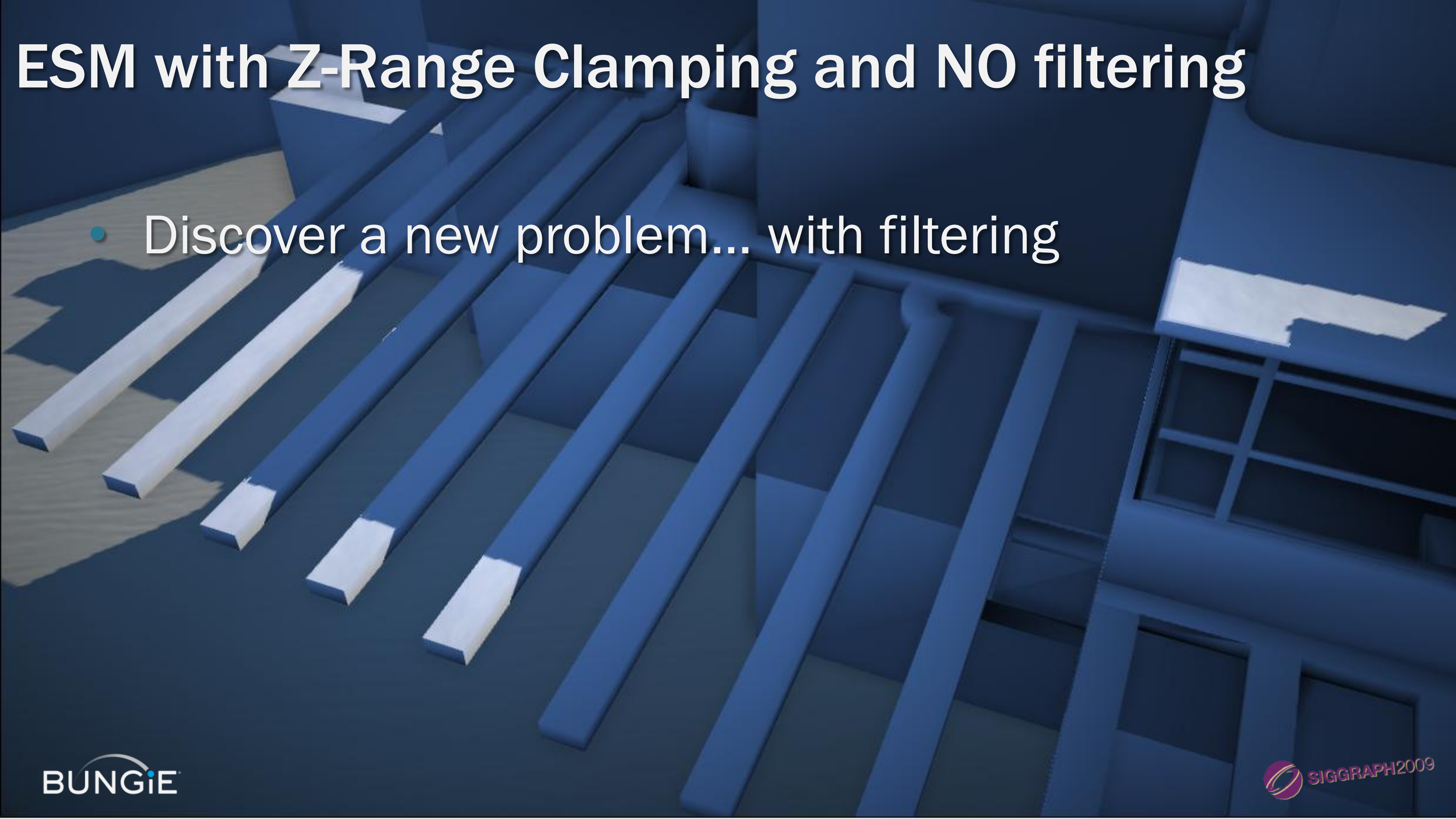
BUNGIE

SIGGRAPH2009

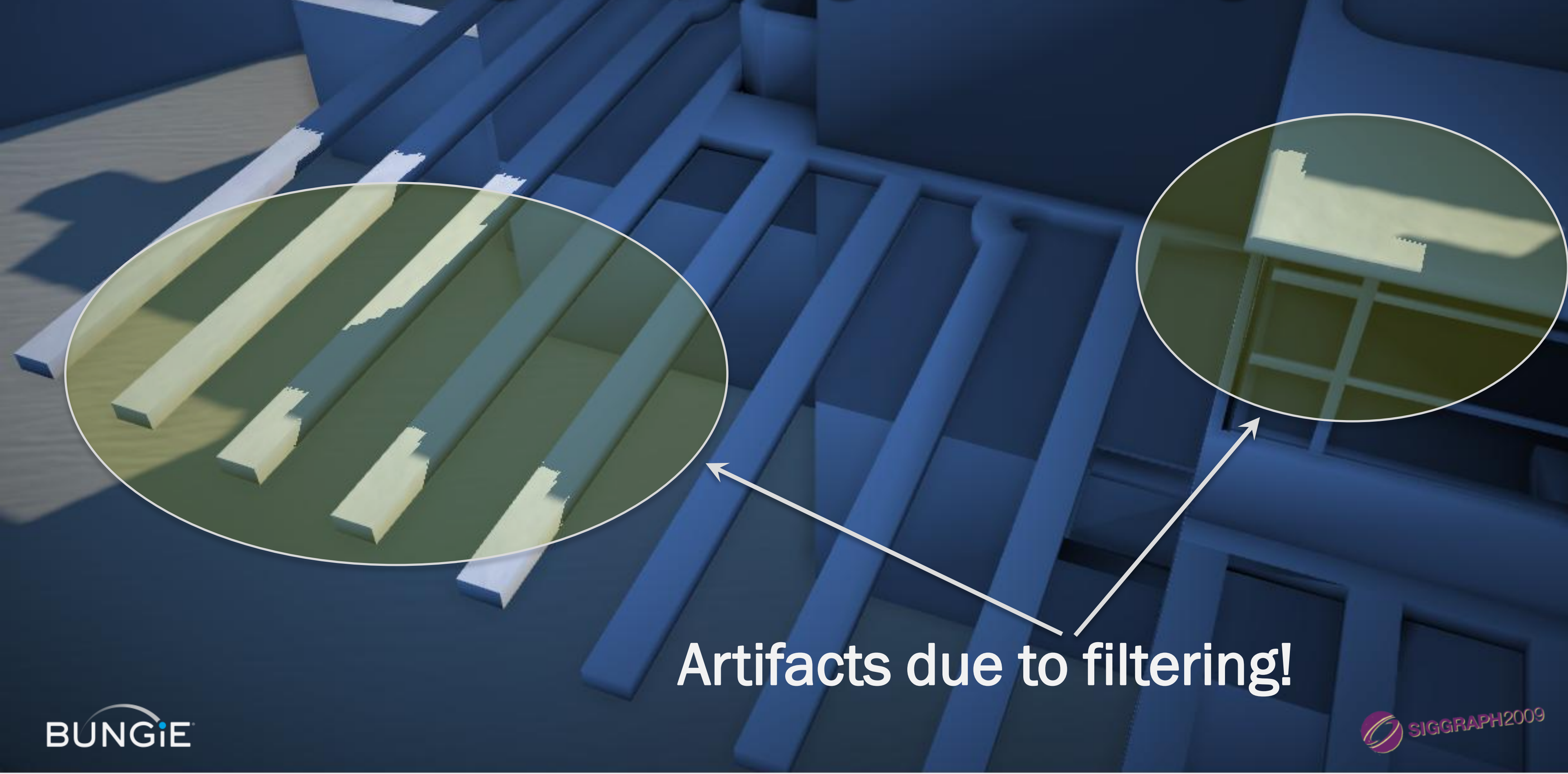

BUNGIE
SIGGRAPH2009

ESM with Z-Range Clamping and NO filtering

- Discover a new problem… with filtering



ESM with Z-Range Clamping and Filtering



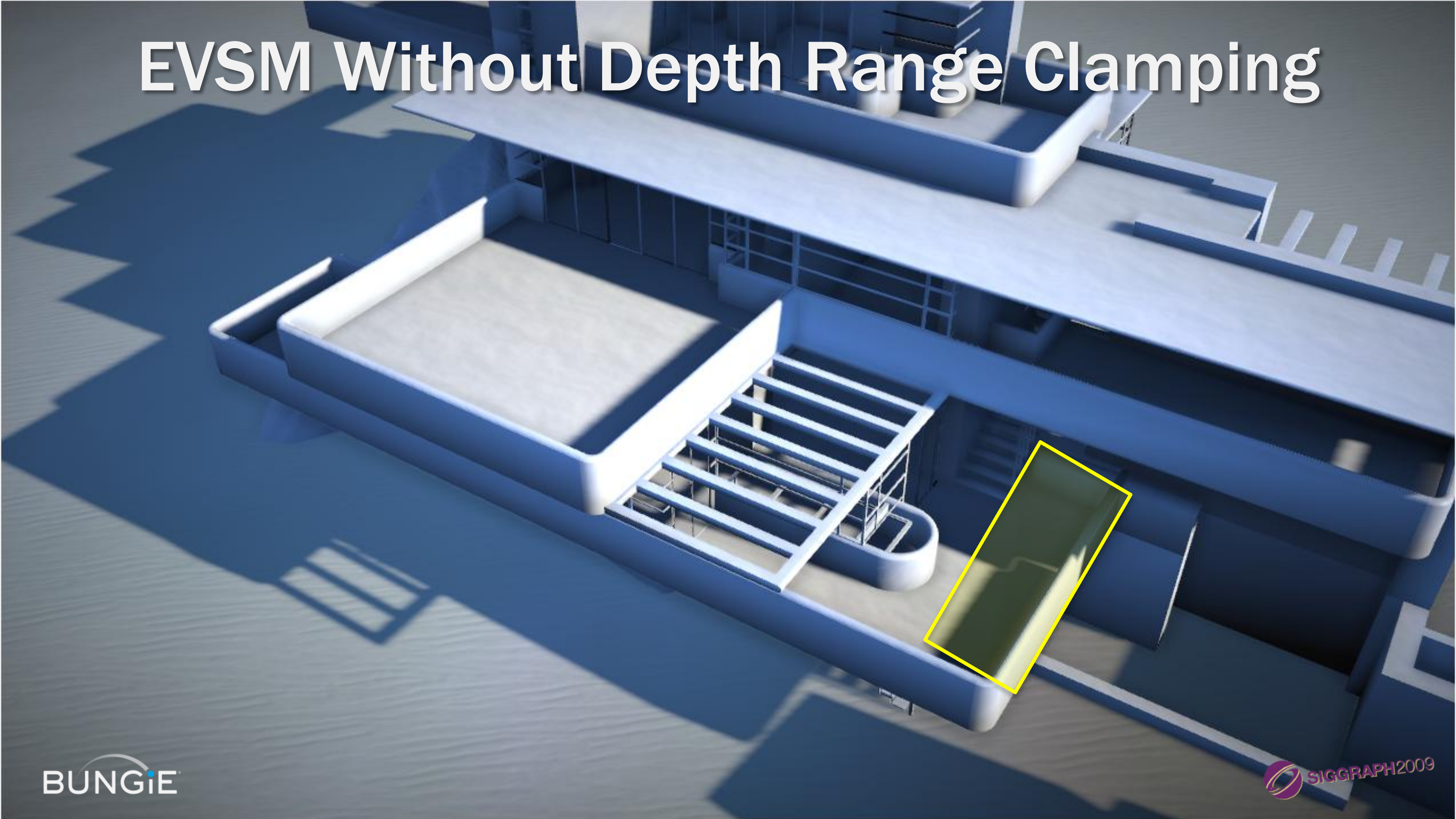
EVSM with Depth Warps

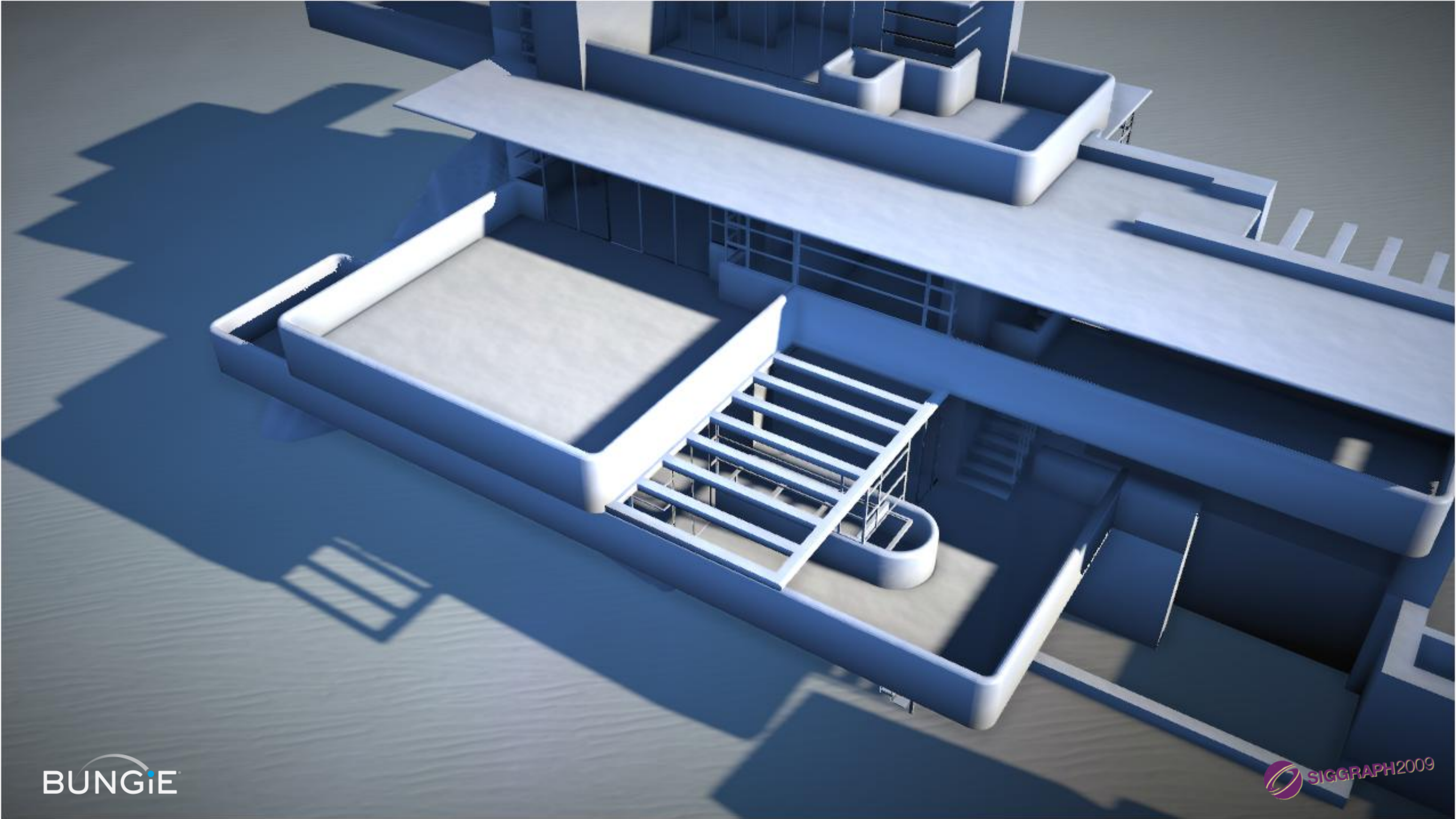
- Can we do better? Yes, we can – using Exponential Variance Shadow Maps (EVSM)
  - Combines the benefits of ESM and VSM
- Significantly alleviates contact leaking artifacts
  - At increased memory cost (4X!)
- Light bleeding at high variance areas re-appears
  - However, this can be easily reduced (especially as compared to VSMs)
- No need to clamp the depth range

EVSM

```
float ComputeEVSM( float2 vShadowMapUVs, float fReceiverDepth, float fCascadeIndex ) {
   //depth should be 0 to 1 range.
   float2 warpedDepth = WarpDepth(fReceiverDepth);
   float  posDepth = warpedDepth.x;
   float  negDepth = warpedDepth.y;

   float4 occluder = tCascadeShadowMaps.Sample( sShadowLinearClamp,
                            float3( vShadowMapUVs, fCascadeIndex ));
   float2 posMoments = occluder.xz;
   float2 negMoments = occluder.yw;

   // compute derivative of the warping function at depth of pixel and use it to scale min
   // variance
   float posDepthScale  = fESMExponentialMultiplier * posDepth;
   float posMinVariance = VSM_MIN_VARIANCE * posDepthScale * posDepthScale;
   float negDepthScale  = fESMExponentialMultiplier2 * negDepth;
   float negMinVariance = VSM_MIN_VARIANCE * negDepthScale * negDepthScale;

   //compute two Chebyshev bounds, one for positive and one for negative, and takes the
   // minimum
   float shadowContrib1= ComputeChebyshevBound(posMoments.x, posMoments.y, posDepth,
                                                posMinVariance);
   float shadowContrib2= ComputeChebyshevBound(negMoments.x, negMoments.y, negDepth,
                                               negMinVariance);
   return min(shadowContrib1, shadowContrib2);
   }
```

EVSM Without Depth Range Clamping




BUNGIE
SIGGRAPH2009

Conclusions on Shadows

- No perfect *and* inexpensive solution exists at the moment (at least not yet)
- Presented a grab-bags of techniques – pick and choose to suit the needs of your game
- Tried to provide the intuition behind the solutions and hacks

GPU Pre-computed Lighting

Motivation

- Exploit massive parallelism of GPU architecture
- Take advantage of GPGPU advances
- Integrated workflow
- High quality global illumination
- Possible path to the future

Goals/Requirements

- Handle large scenes (5 to 7 million triangles)
- Support all kinds of light sources
- Fast performance
- Real time preview
- User controlled quality-time tradeoff
- General purpose

CPU Photon Mapping Farm

Initialization → Direct Illumination → Photon Cast

↓

Final Gather ← Exit Illumination ← Radiance Estimate

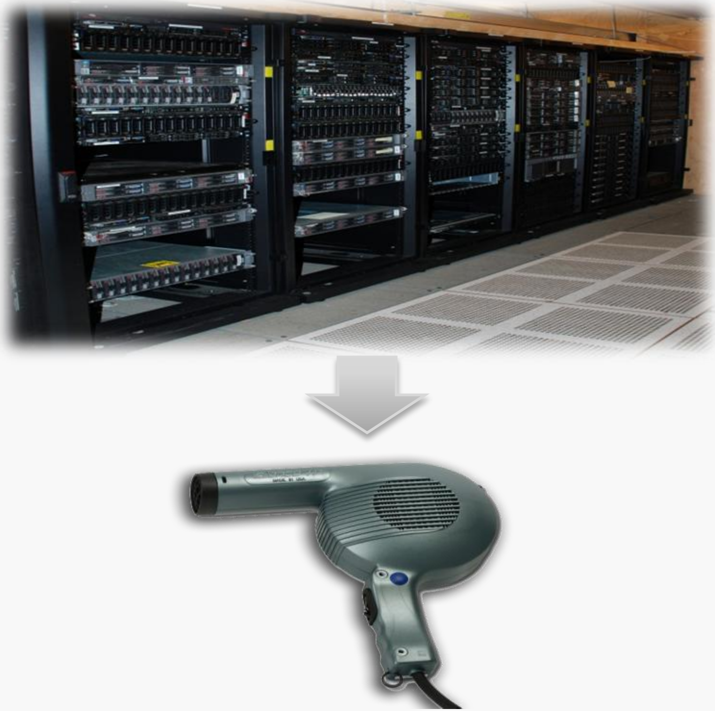
↓

Signal Compression → DXT Compression

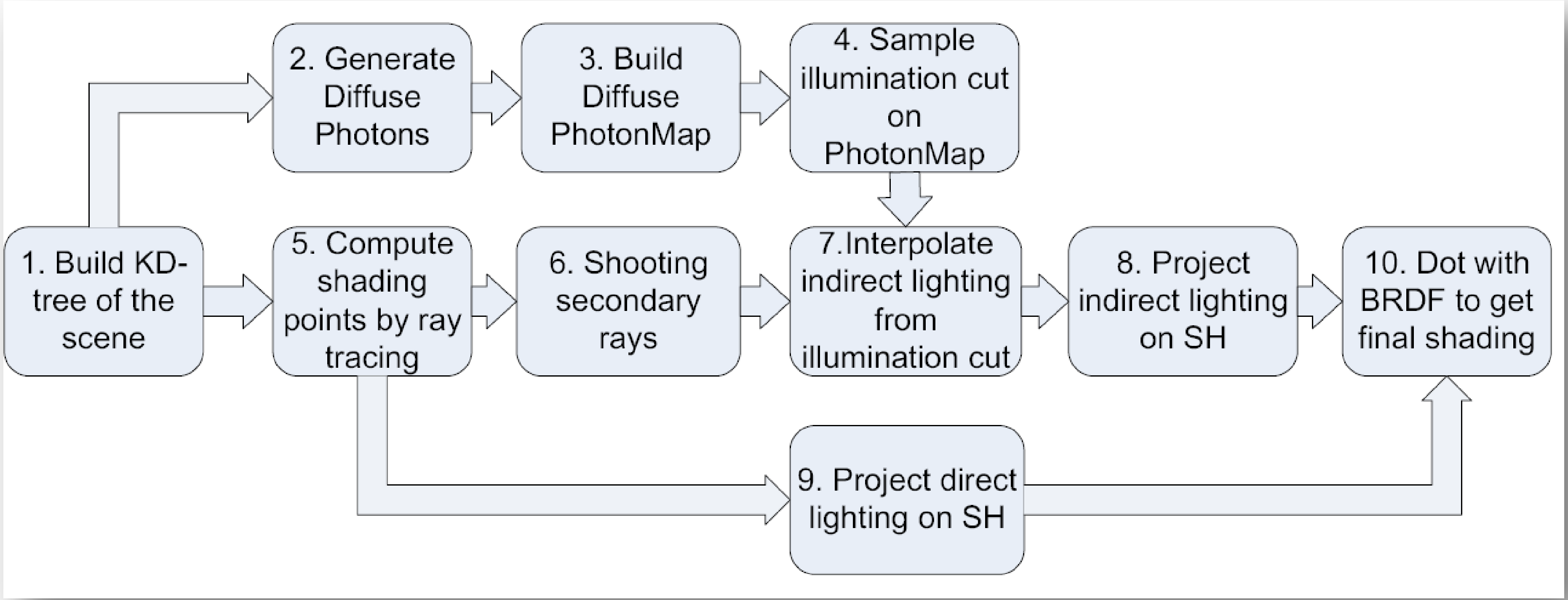
Slow!

Speeding up the slow parts

- Direct Illumination
  - Fast ray-cast using GPU KD tree
- Final Gather
  - Fast ray-cast using GPU KD tree
  - Photon Illumination Cut
  - Cluster sample points for indirect illumination



Core Algorithm

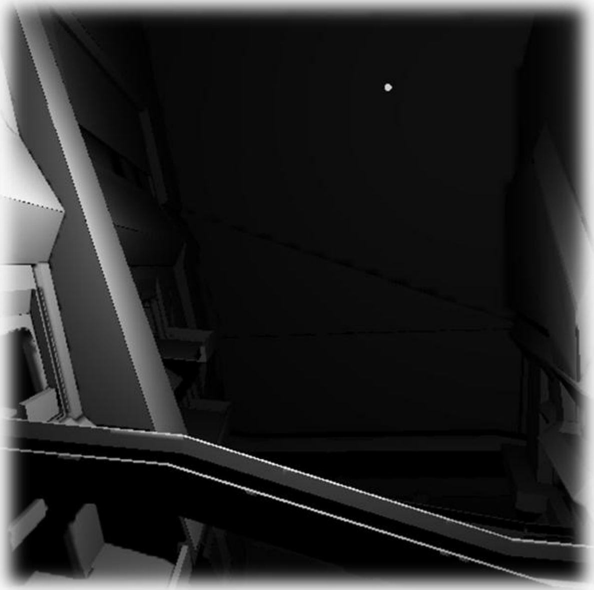


GPU K-D Tree Construction

- [Zhou2008]: General purpose KD - tree in GPU
  - Fast
  - High quality
  - High Peak Memory
- [Zhou2009]: Memory scalable KD-Tree
  - Bounded memory usage

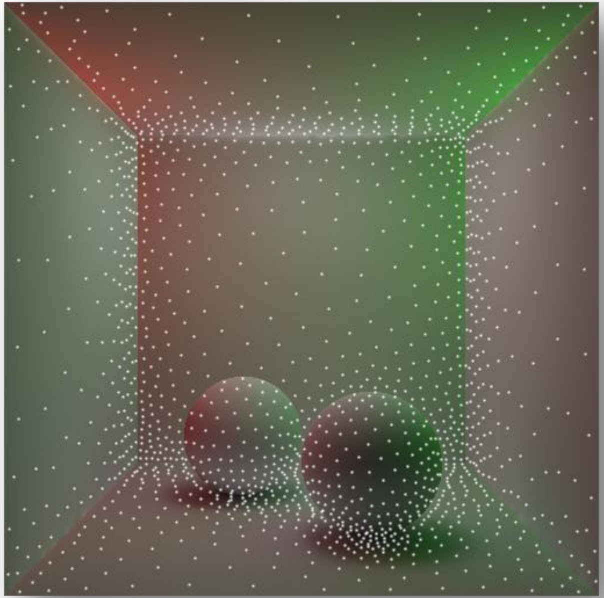
Direct Illumination

- Generate shading points
  - For preview, ray trace
  - For light map, use texels
- Cast shadow rays towards light source
  - Area light source
  - Multiple rays per light



Indirect Illumination Sampling

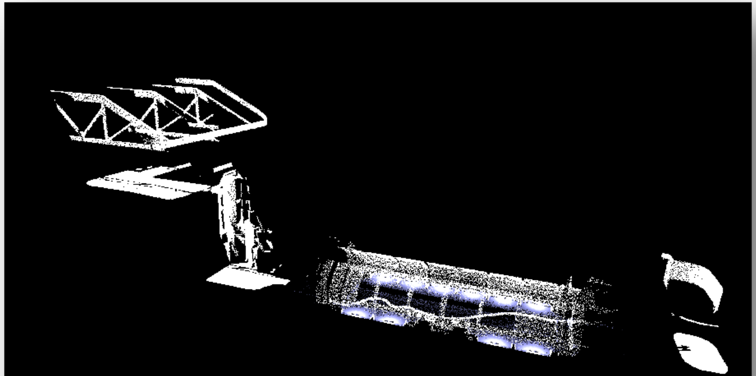
- Indirect Illumination is low frequency
  - Don’t need to sample at every shading point
- Cluster samples using geometry and normal variation
- Sample at cluster center
- Coarse to fine interpolation



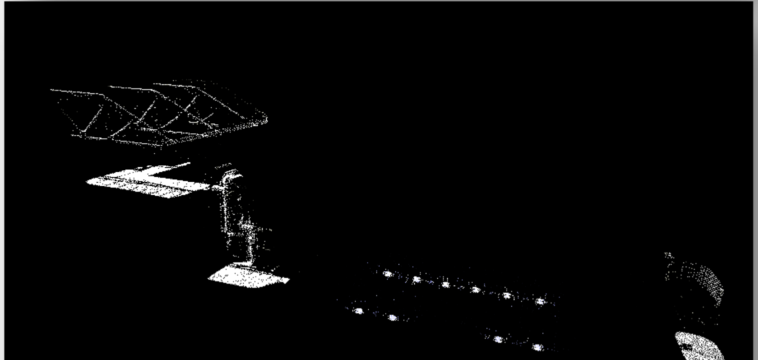
[WZPB2009]

Photon Illumination Cuts

- Similar to light cuts
- Estimate irradiance at each node of photon tree
- Compute "cut" through the tree
- Interpolate using RBF basis



Photon Distribution

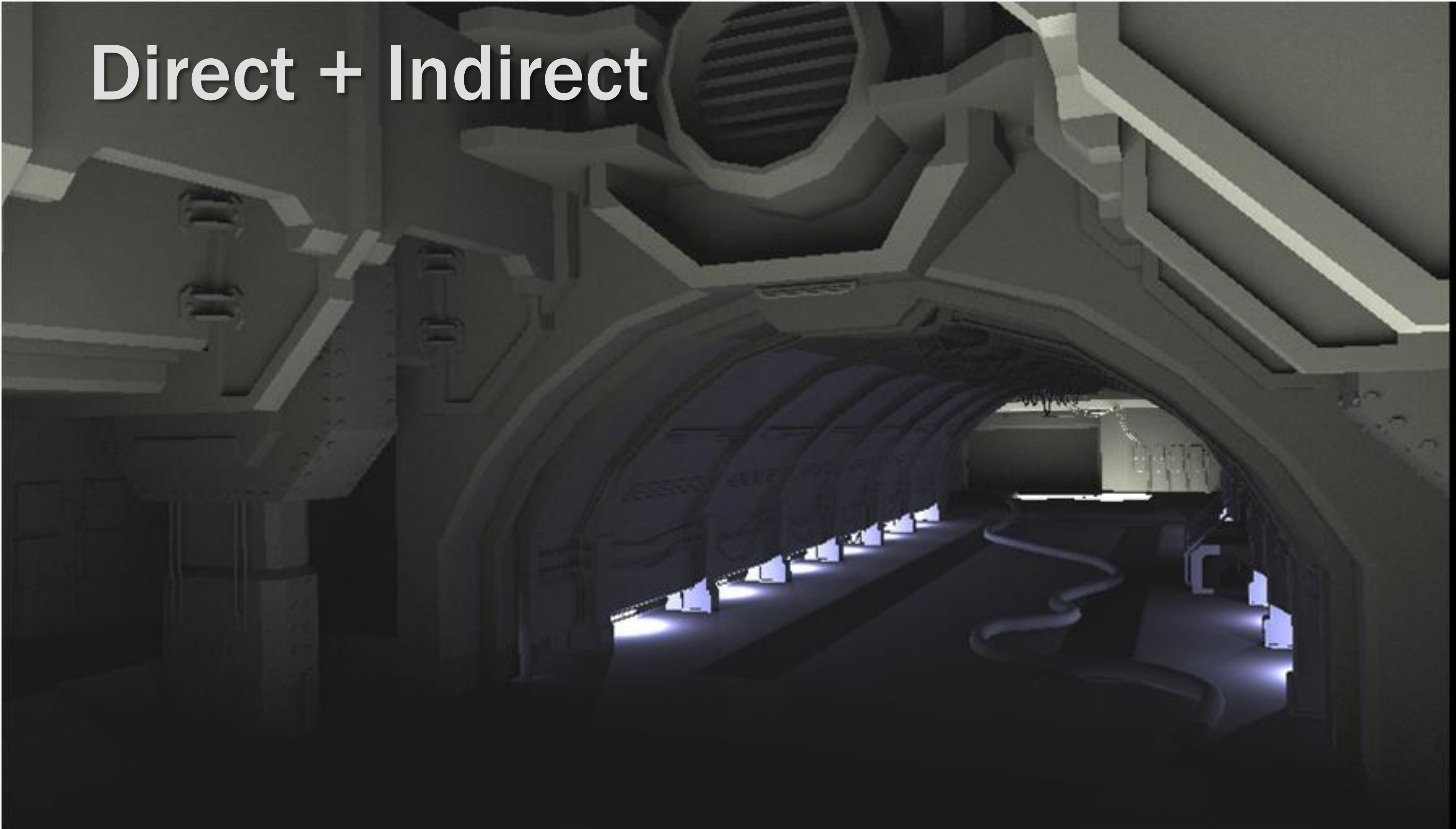


Cut Nodes

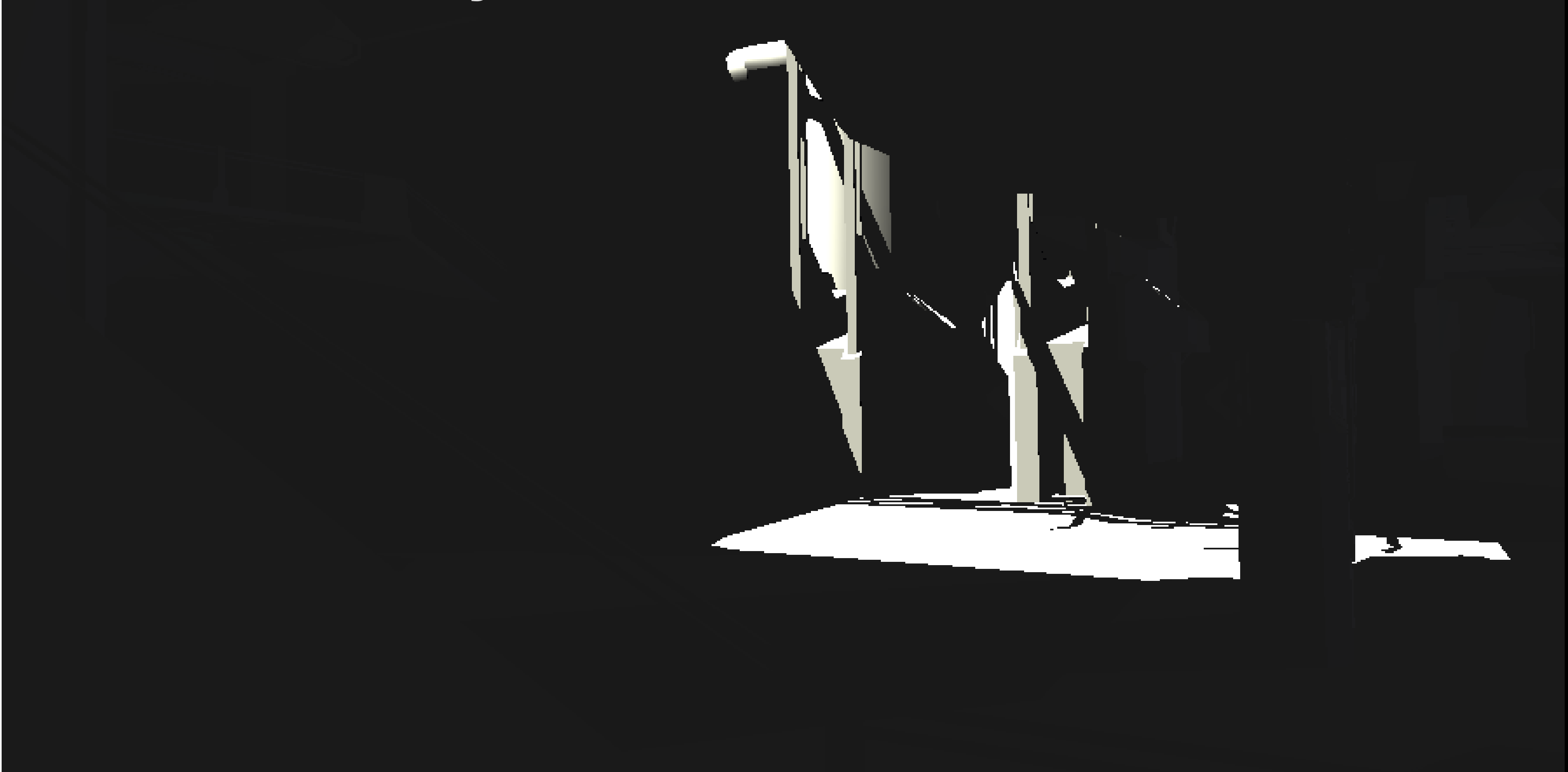
Direct Only

Indirect Only

Direct + Indirect



Direct Only




Indirect Only

Direct + Indirect

Result

Light number: 12

Photon number: 700k

Triangles: 253911

Vertices: 761733

Time:

Scene KD-tree	Photon Tracing	Compute direct	Illumination cut	Irradiance cache	Interpolation
175.11ms	293.12ms	144.80ms	3177.03ms	117.8ms	680.97ms

Conclusions

- Direct illumination is still not a “solved” problem
- Gap closing up on interactive global illumination
  - Different methods converging towards that goal
- Choose right technique for the right job

Acknowledgements and Thanks

- Adrian Perez, Shi Kai Wang, Chris Barrett, Ryan Ellis, Mark Goldsworthy and Paul Vosper at Bungie for their awesome work on the demos shown
- Marco Salvi, Andrew Lauritzen, Aaron Lefohn, Nicolas Thibieroz & Holger Grun for many discussions on the imperfect nature of shadows, their ideas and existing work (especially Marco and Andrew!)
- Kun Zhou and his group at Zhezhiang University for GPU LightMapper collaboration

BUNGIE®

Is Hiring!

www.bungie.net/jobs

Selected References: Atmosphere

- Bruneton, E. and Neyret, F. 2008. Precomputed Atmospheric Scattering. EGSR 2008. Computer Graphics Forum, 27(4), June 2008, pp. 1079-1086.
- Habel, R., Mustata, B., Wimmer, M. Efficient *Spherical Harmonics* Lighting with the *Preetham* Skylight. *Model*. In *Eurographics 2008 - Short Papers*, pages 119-122. April 2008.
- Preetham, A. J., Shirley, P., and Smits., B. E.: A Practical Analytic Model for Daylight. In Siggraph 1999, Computer Graphics Proceedings (Los Angeles, 1999), Rockwood A., (Ed.), Addison Wesley Longman, pp. 91–100.
- [HP03] Hoffman, N., and Preetham, A. J.: Real-time Light-Atmosphere Interactions for outdoor scenes. *Graphics Programming Methods* (2003), pp. 337–352.

Selected References: Shadows

- Reeves, W. T., Salesin, D. H., and Cook, R. L. 1987. Rendering antialiased shadows with depth maps. In *Proceedings of the 14th Annual Conference on Computer Graphics and interactive Techniques* SIGGRAPH '87. ACM, New York, NY, 283-291.
- Donnelly, W. and Lauritzen, A. 2006. Variance shadow maps. In *Proceedings of the 2006 Symposium on interactive 3D Graphics and Games* (Redwood City, California, March 14 - 17, 2006). I3D '06. ACM, New York, NY, 161-165
- Salvi, M. 2008. Rendering Filtered Shadows with Exponential Shadow Maps, ShaderX^{6}, Charles River Media
- Annen, T., Mertens, T., Seidel, H., Flerackers, E., and Kautz, J. 2008. Exponential shadow maps. In *Proceedings of Graphics interface 2008* (Windsor, Ontario, Canada, May 28 - 30, 2008). GI, vol. 322. Canadian Information Processing Society, Toronto, Ont., Canada, 155-161.
- Lauritzen, A. and McCool, M. 2008. Layered variance shadow maps. In *Proceedings of Graphics interface 2008* (Windsor, Ontario, Canada, May 28 - 30, 2008). GI, vol. 322. Canadian Information Processing Society, Toronto, Ont., Canada, 139-146.

Selected References: GPU LightMapping

- Wang, R., Wang, R., Zhou, K., Pan, M., and Bao, H. 2009. An efficient GPU-based approach for interactive global illumination. *ACM Trans. Graph.* 28, 3 (Jul. 2009), 1-8
- Zhou, K., Hou, Q., Wang, R., and Guo, B. 2008. Real-time KD-tree construction on graphics hardware. *ACM Trans. Graph.* 27, 5 (Dec. 2008), 1-11.
- Hou, Q., Zhou, K., and Guo, B. 2008. BSGP: bulk-synchronous GPU programming. In *ACM SIGGRAPH 2008 Papers* (Los Angeles, California, August 11 - 15, 2008). SIGGRAPH '08. ACM, New York, NY, 1-12.

Thank you!

- These slides and course notes will be available online

http://www.bungie.net/publications