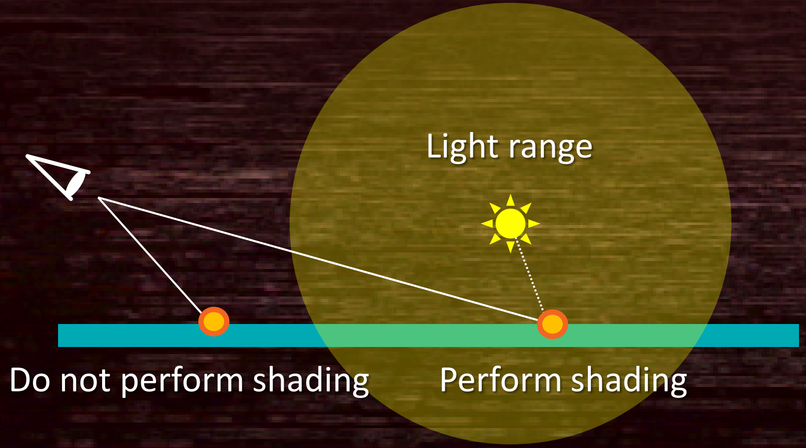
# STOCHASTIC LIGHT CULLING

YUSUKE TOKUYOSHI (SQUARE ENIX CO., LTD.)
TAKAHIRO HARADA (ADVANCED MICRO DEVICES, INC.)

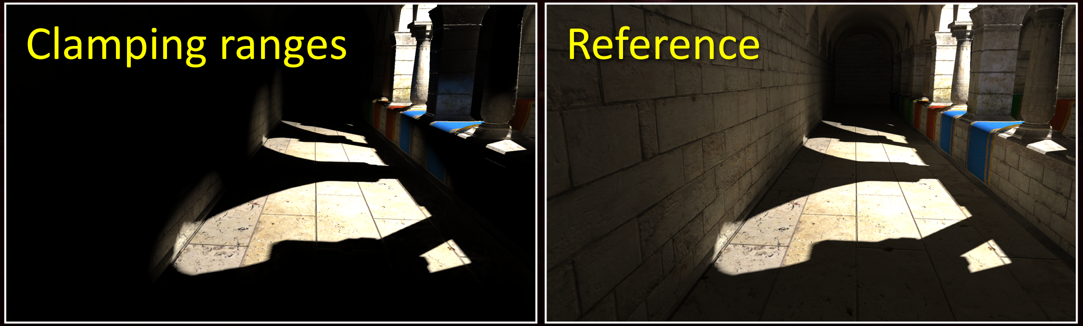
# LIGHT CULLING

## PREVIOUS WORK

- Restrict the influence range of light
- Perform shading only inside that range
  - Splatting [Dachsbacher06]
  - Tile-based culling [Olsson11; Harada12]
  - Clustered shading [Olsson12]




Darkening bias accumulates as the number of lights increases




Indirect illumination using 65536 virtual point lights (VPLs) [Keller97]

# STOCHASTIC LIGHT CULLING

# INDIRECT ILLUMINATION (65536 VIRTUAL POINT LIGHTS)

VIDEO


Resolution: 1920×1152, GPU: AMD Radeon$^{TM}$ R9 290X

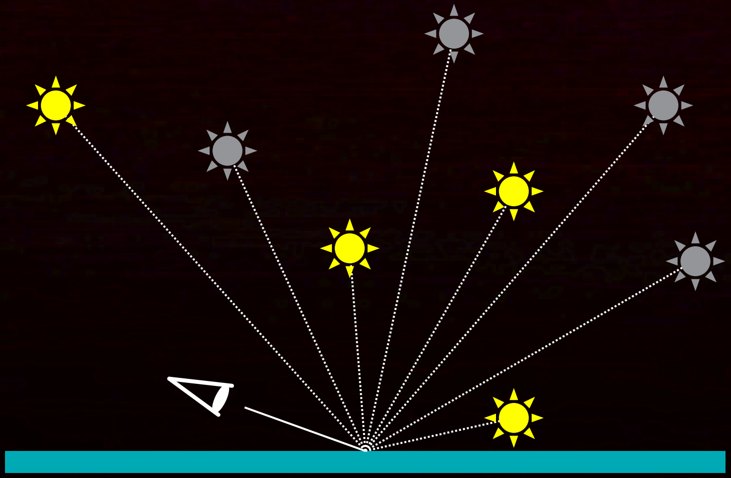
# OUR METHOD

**Light range determination based on Russian roulette** → **Any culling method** → **Shading with Russian roulette**

- Random influence ranges based on ***Russian roulette*** [Arvo90]
  - Can sample distant point lights with low probability
- Unbiased sampling
- Variance is produced instead of bias
- Unlike the darkening bias, this variance does not accumulate as lights increase ☺

# RUSSIAN ROULETTE

- Kill each light stochastically
- Probability: proportional to the fall-off function
- Divide the energy of a surviving light by the probability



Fall-off: $f(l) = \frac{1}{l^2}$

Distance from a light

probability: $p(l) = \min\left(\frac{f(l)}{\alpha}, 1\right)$

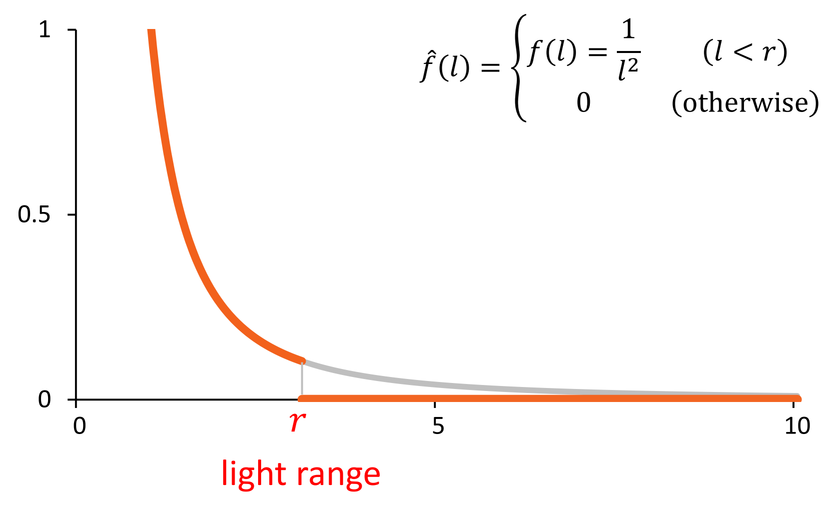
User-specified parameter

to control variance

# STOCHASTIC FALL-OFF FUNCTION

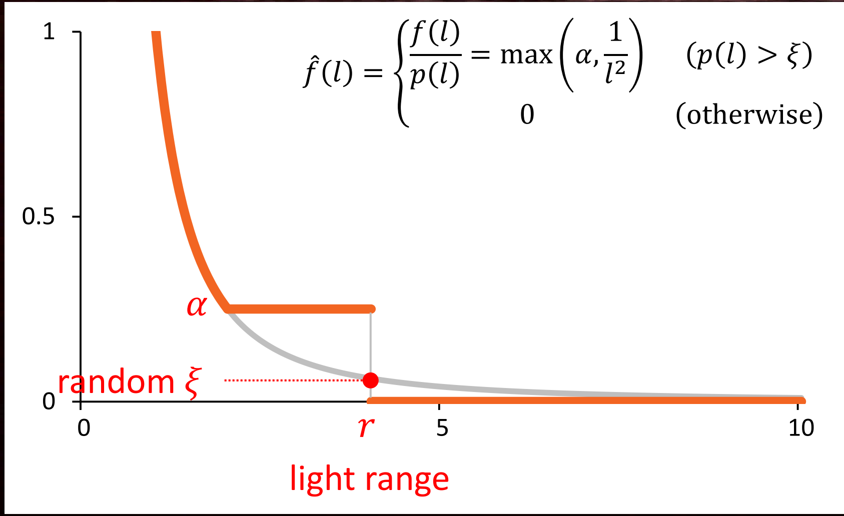
## Clamping

$$\hat{f}(l) = \begin{cases} f(l) = \dfrac{1}{l^2} & (l < r) \\ 0 & (\text{otherwise}) \end{cases}$$



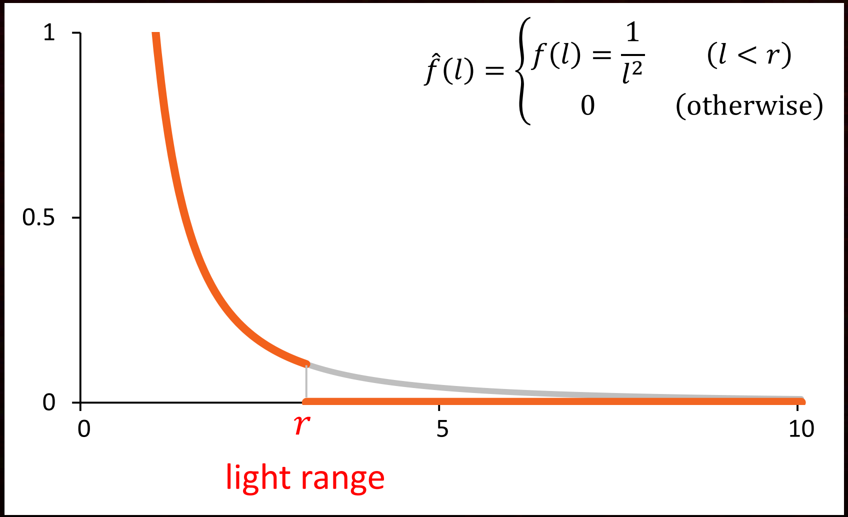

## Stochastic

$$\hat{f}(l) = \begin{cases} \dfrac{f(l)}{p(l)} = \max\left(\alpha, \dfrac{1}{l^2}\right) & (p(l) > \xi) \\ 0 & (\text{otherwise}) \end{cases}$$

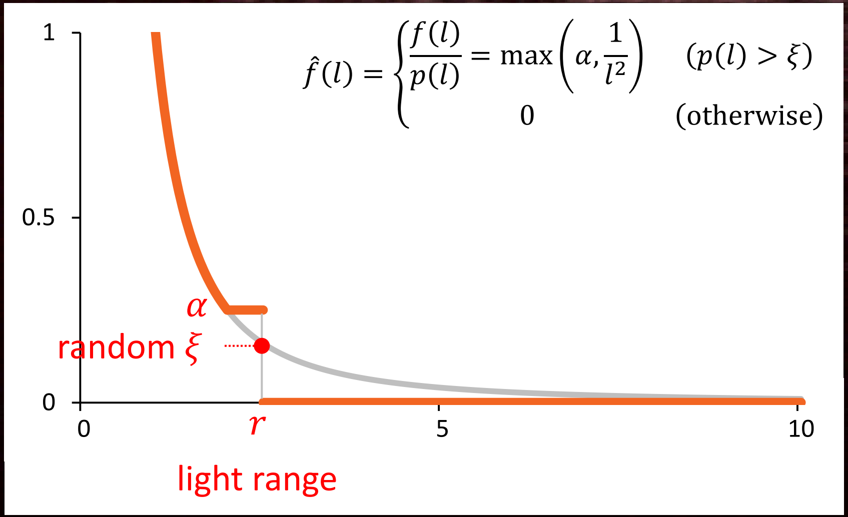


# STOCHASTIC FALL-OFF FUNCTION

## Clamping



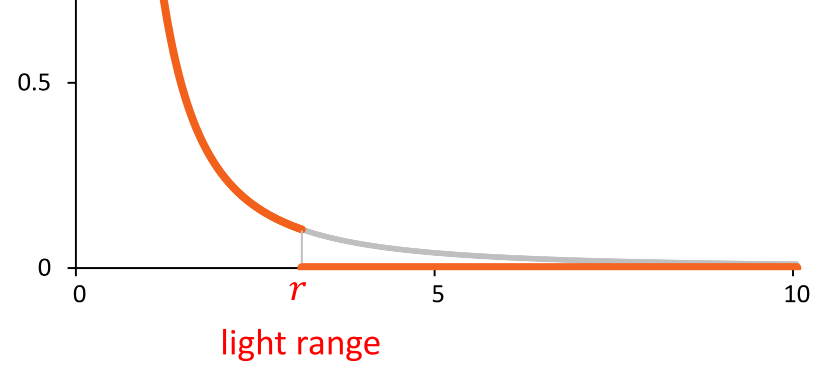

## Stochastic



# STOCHASTIC FALL-OFF FUNCTION

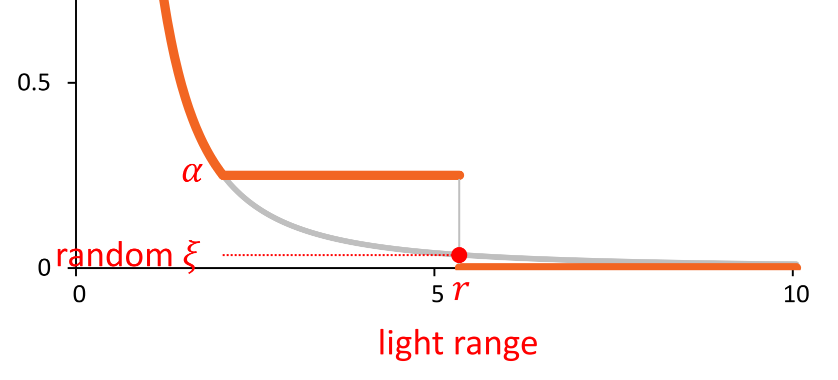
## Clamping

$$\hat{f}(l) = \begin{cases} f(l) = \dfrac{1}{l^2} & (l < r) \\ 0 & (\text{otherwise}) \end{cases}$$

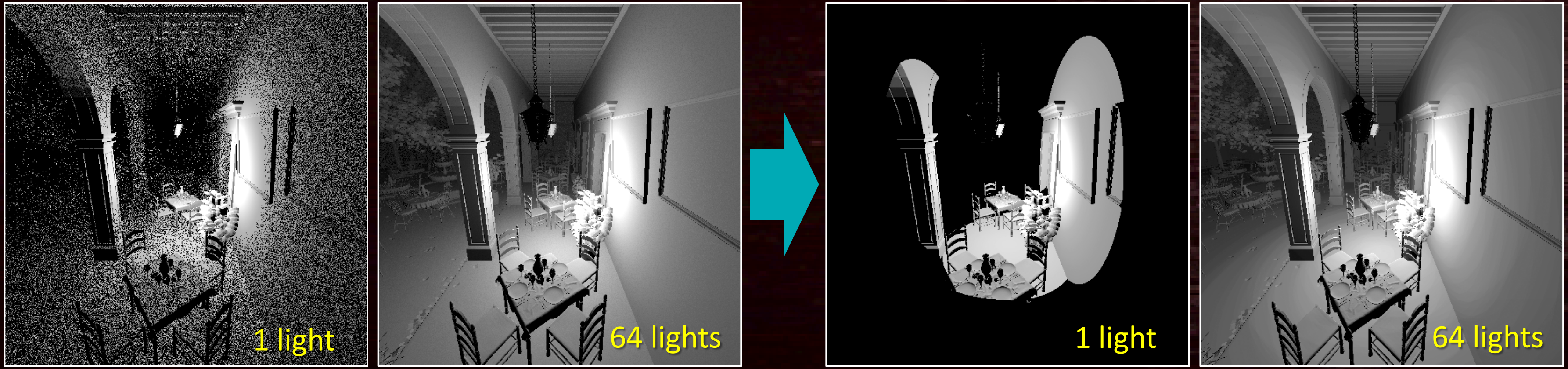



## Stochastic

$$\hat{f}(l) = \begin{cases} \dfrac{f(l)}{p(l)} = \max\left(\alpha, \dfrac{1}{l^2}\right) & (p(l) > \xi) \\ 0 & (\text{otherwise}) \end{cases}$$



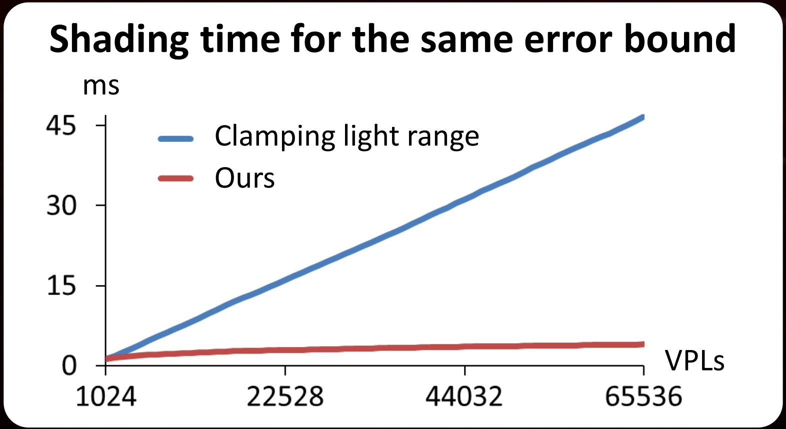
# RANDOM INFLUENCE RANGE




- Different ranges between shading points ☹
- For culling, we have to use an identical range for each light
- Solution: Single random number for each light
  - Unbiased coherent sampling ☺
  - Variance is visible as banding artifacts instead of noise

# ERROR BOUND-BASED LIGHT RANGE

- Tradeoff between variance and computation time
- Employ a user-specified error bound to avoid oversampling
  - Lower sampling probability (i.e., smaller light range) for smaller radiant intensity
- For VPLs, the number of intersecting lights is sublinear ☺

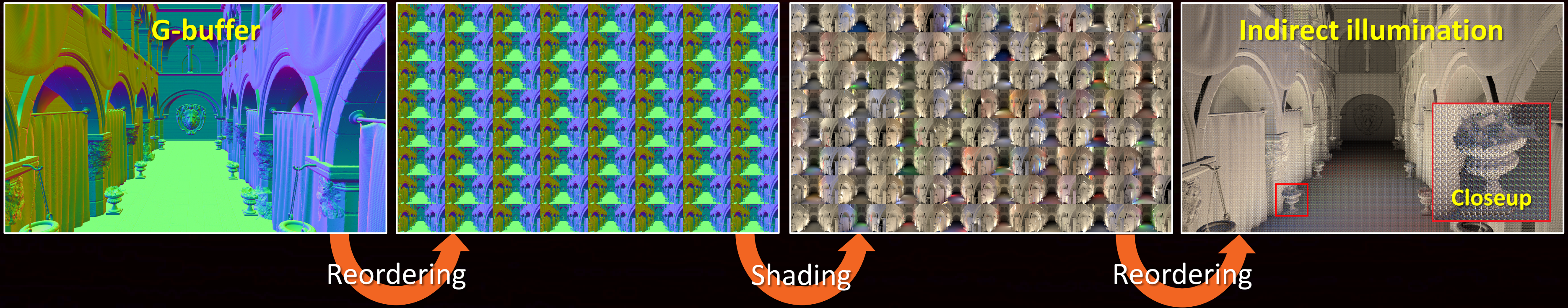


# EXAMPLE IMPLEMENTATION OF REAL-TIME INDIRECT ILLUMINATION

# CLASSIC ALGORITHM

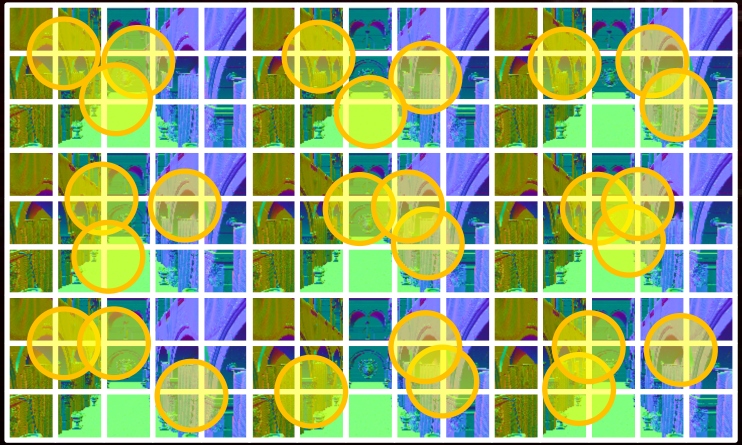
## PREVIOUS WORK

- (1) Generate 65536 VPLs by rendering a reflective shadow map [Dachsbacher05]
- (2) Shade using 8x8 interleaved sampling of VPLs [Segovia06]
  - Different VPL subsets between neighboring pixels (i.e., 1024 VPLs per pixel)
  - Reorder pixels into 8×8 subregions to reduce the divergence of threads
  - Variance is visible as noise
- (3) Denoise in post processing (cross bilateral filtering)



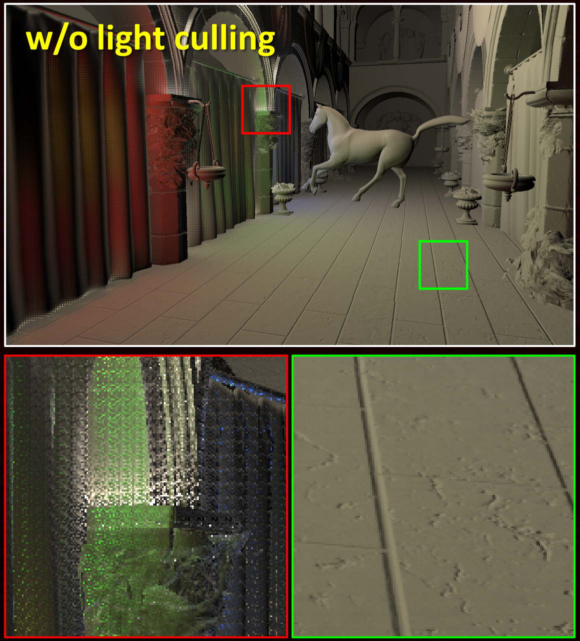
# APPLY STOCHASTIC LIGHT CULLING

- Combination of interleaved sampling and stochastic light culling
- Tiled deferred shading [Andersson11] for each subregion
  - 8×8 interleaved sampling for 65536 VPLs -> 1024 VPLs per subregion
  - Then, stochastic light culling is performed for each 1024 VPLs

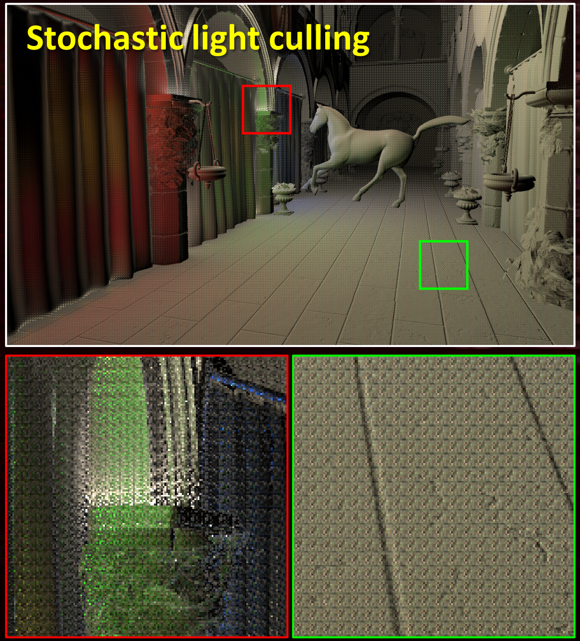


# RESULTS

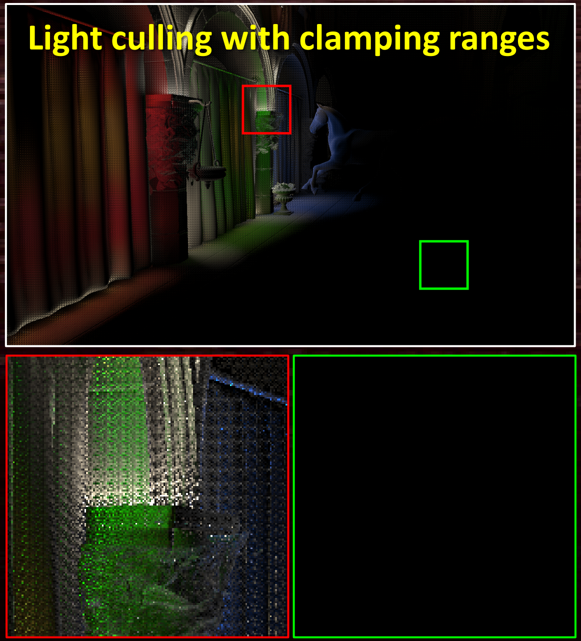
## BEFORE DENOISING




Shading time: 44.4 ms




Shading time: 2.87 ms

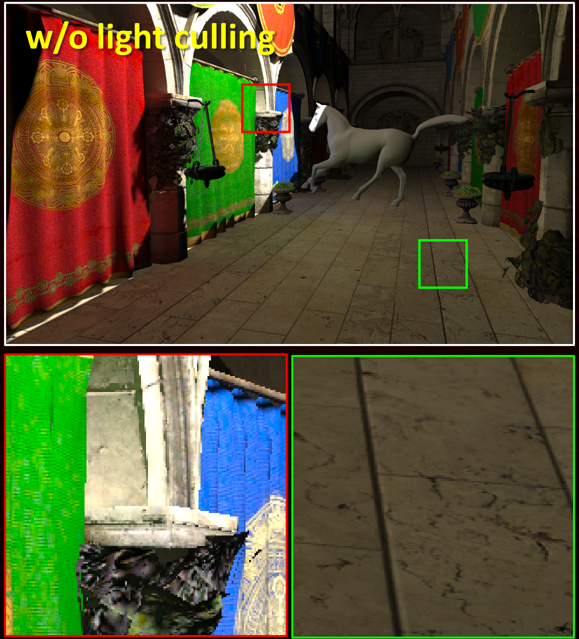



Shading time: 2.87 ms

65536 VPLs, Resolution: 1920×1152, Error bound: 0.0005, GPU: AMD Radeon$^{TM}$ R9 290X

# RESULTS

## AFTER DENOISING, TEXTURING, AND ADDING DIRECT ILLUMINATION




Total rendering time: 48.5 ms

RMSE: 0.0017


Total rendering time : 7.0 ms

RMSE: 0.0026


Total rendering time : 7.0 ms

RMSE: 0.0377

65536 VPLs, Resolution: 1920×1152, Error bound: 0.0005, GPU: AMD Radeon™ R9 290X

# RESULTS

## EQUAL-TIME COMPARISON

1024 VPLs (shading time: 1.19 ms)

Stochastic light culling

Light culling with clamping ranges

Resolution: 1920×1152, Error bound: 0.0005, GPU: AMD Radeon™ R9 290X

# RESULTS

## EQUAL-TIME COMPARISON

4096 VPLs (shading time: 1.59 ms)

Stochastic light culling

Light culling with clamping ranges

Resolution: 1920×1152, Error bound: 0.0005, GPU: AMD Radeon™ R9 290X

# RESULTS

## EQUAL-TIME COMPARISON

16384 VPLs (shading time: 2.15 ms)

Stochastic light culling

Light culling with clamping ranges

Resolution: 1920×1152, Error bound: 0.0005, GPU: AMD Radeon™ R9 290X

# RESULTS

## EQUAL-TIME COMPARISON

65536 VPLs (shading time: 2.87 ms)

Stochastic light culling

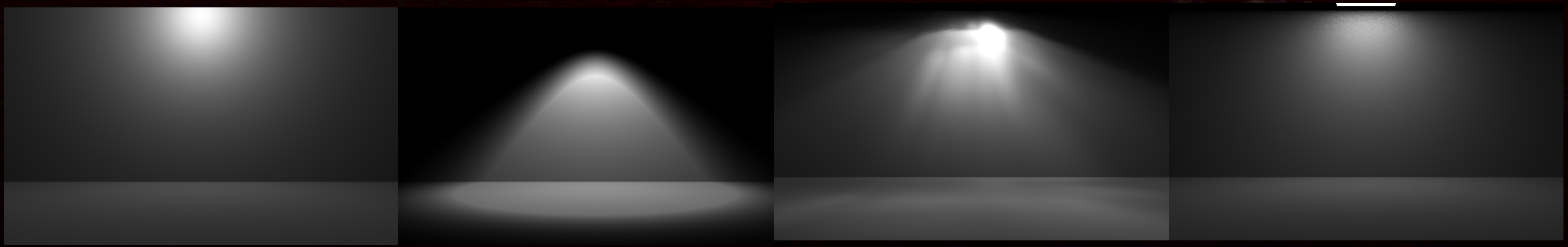
Light culling with clamping ranges

Resolution: 1920×1152, Error bound: 0.0005, GPU: AMD Radeon™ R9 290X

# STOCHASTIC LIGHT CULLING FOR PROGRESSIVE PATH TRACING

# VARIETY OF LIGHT TYPES

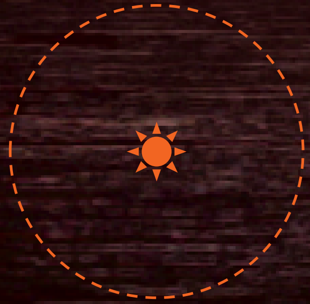
- We have other light types in path tracing
- Most of them are area lights
  - What should we do for area lights?



Point ☺ Spot ☺ IES ☺ Area ☹

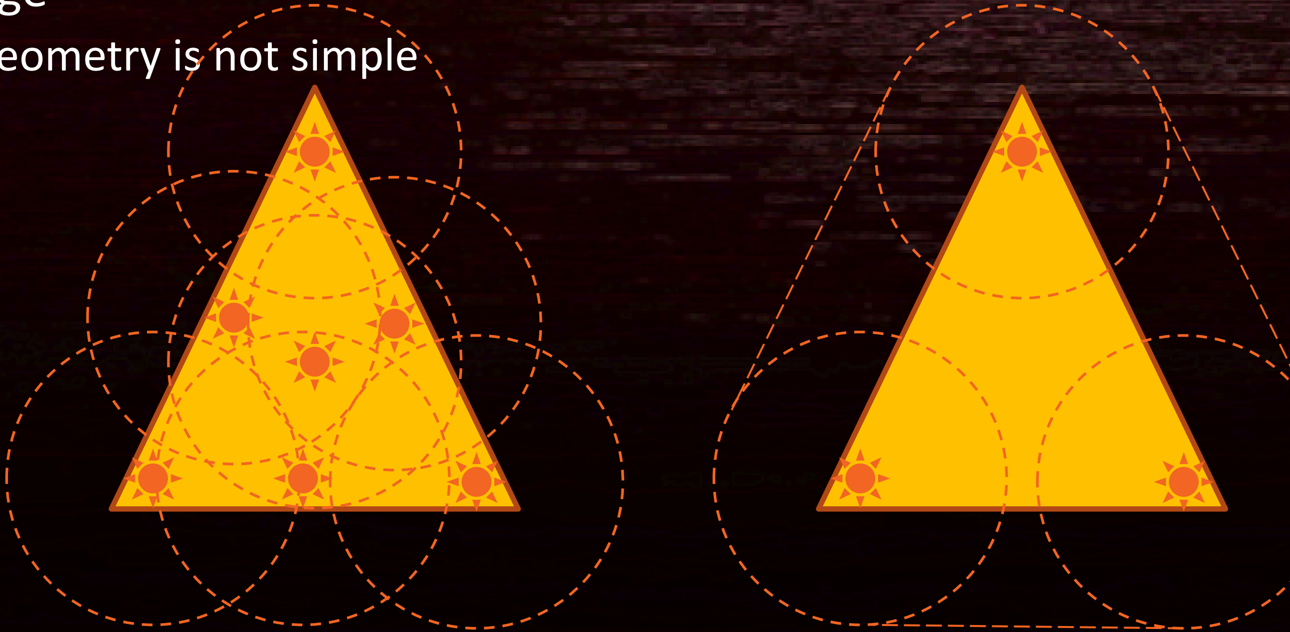
# AREA LIGHT BOUND

- Need to compute light bound
- How??



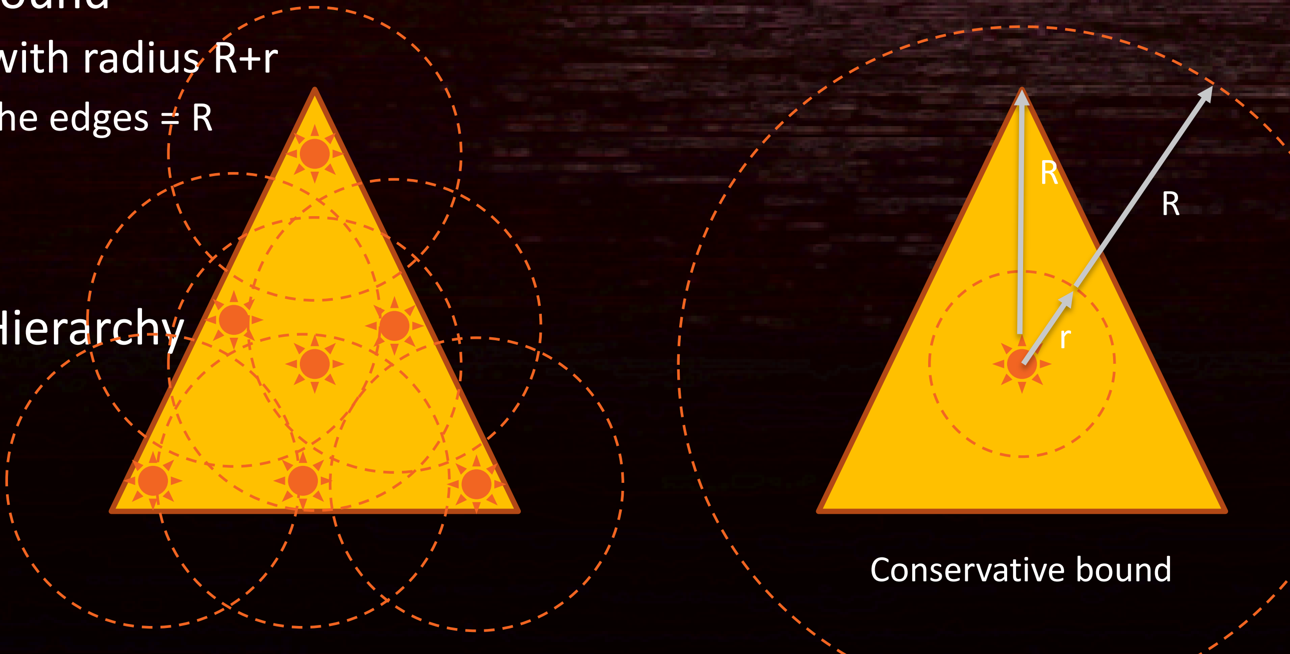

# AREA LIGHT BOUND

- Need to compute light bound
- Area light == Union of point lights
- Sweep sphere on the edge
  - Overlapping test to this geometry is not simple



# AREA LIGHT BOUND

- Need to compute light bound
- Area light == Union of point lights
- Compute conservative bound
  - Represent it as a sphere with radius R+r
    - Where maximum distto the edges = R
    - Radius of a point light = r
- Build Bounding Sphere Hierarchy




Conservative bound

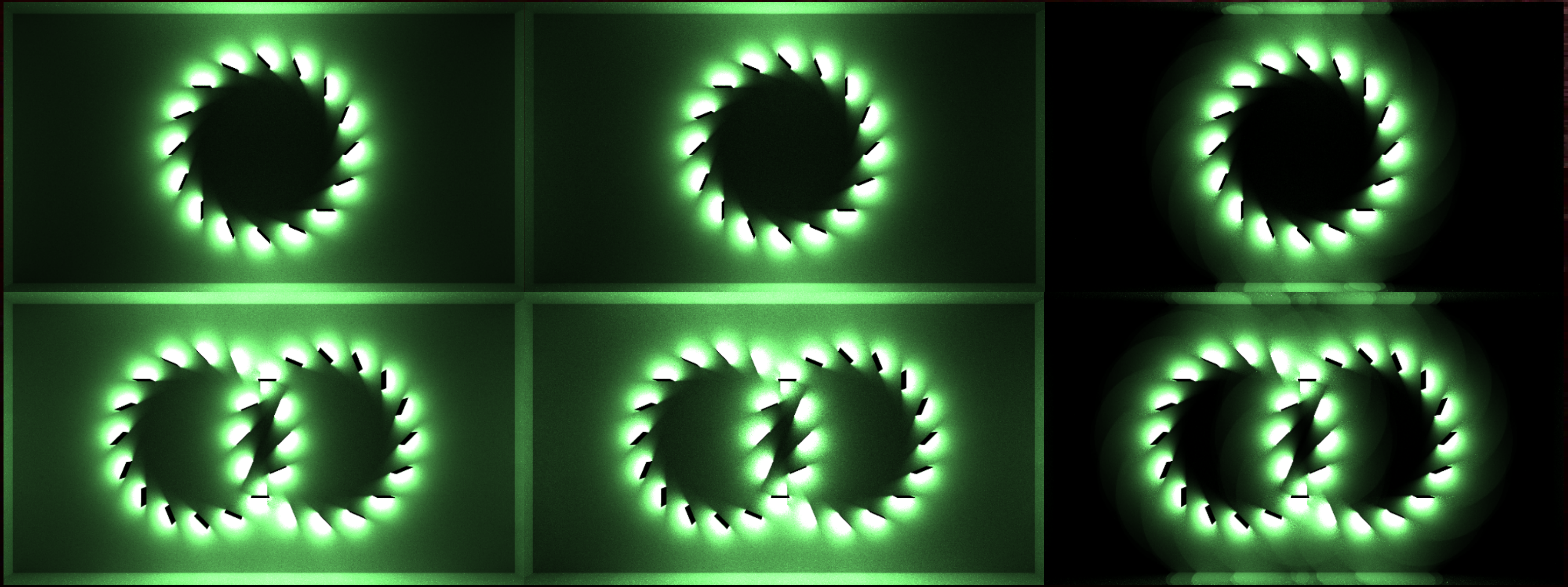
# MULTIPLE IMPORTANCE SAMPLING (MIS)

- Probability is well defined
- Easy to apply MIS [Veach95]
  - Explicit connection + implicit connection
- At implicit connection, light sampling probability is
  - [pdf of sampling the light vertex] x [SLC (Russian Roulette) probability]

# RESULTS

## CONVERGE TO REFERENCE

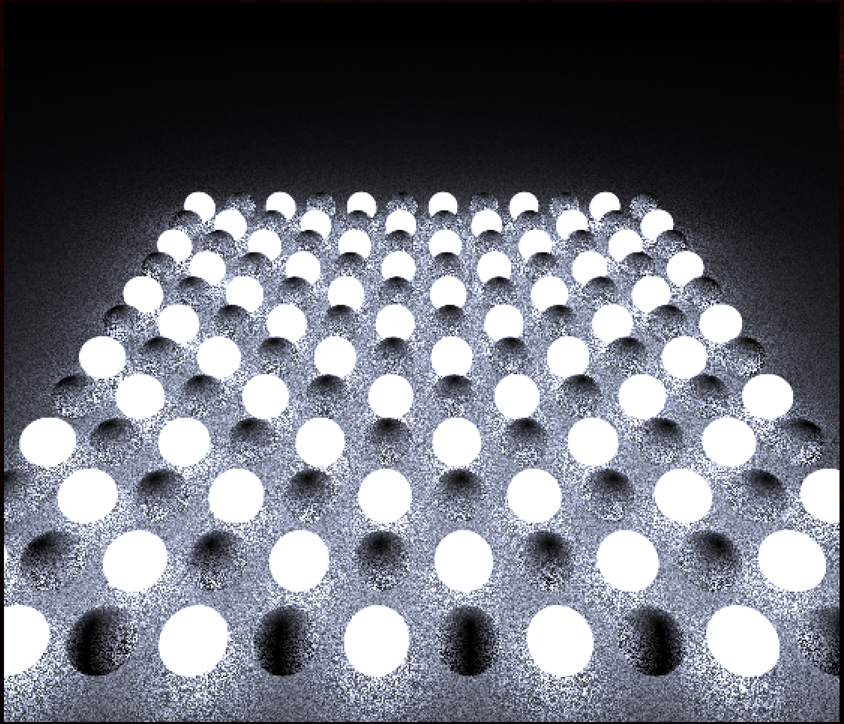
| Reference | Stochastic Light Culling | Clamping |
| --- | --- | --- |



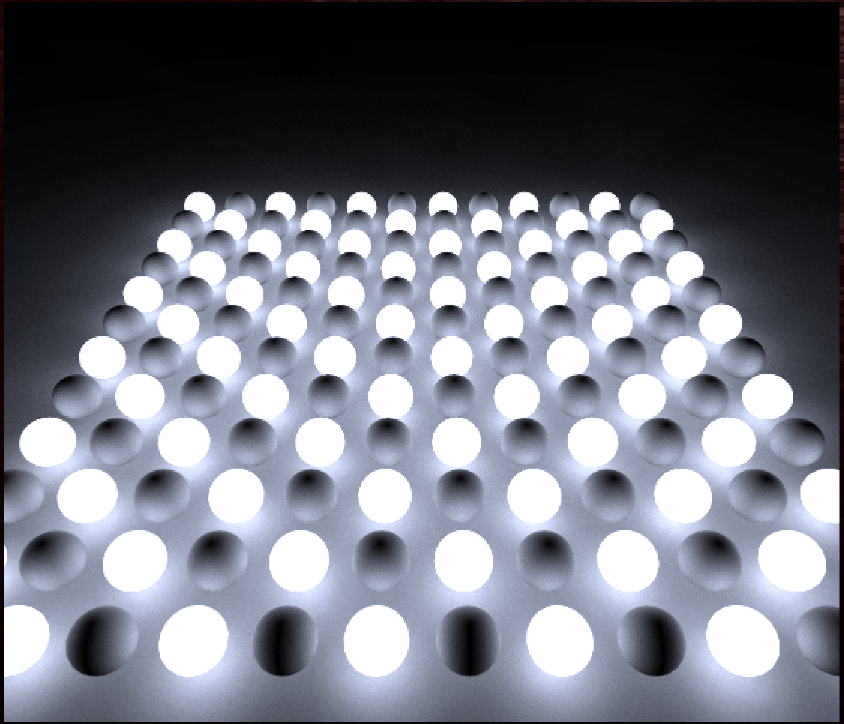
# RESULTS

## CONVERGENCE SPEED COMPARISON (EQUAL TIME ON THE CPU)

**Uniform sampling ☹☹**



**Stochastic Light Culling ☺☺**



**55,000 triangle lights**, after 30s

# STOCHASTIC LIGHT CULLING ON THE GPU

# CHALLENGES IN GPU PATH TRACING

- Stochastic light culling works very well on the CPU
- Porting algorithm as it is causes performance issues on the GPU

1. Work item (thread) divergence
   - Execute shading & visibility test on a leaf visit leads to a large divergence
2. Memory footprint
   - Many WIs are running in parallel
   - Even the storage of hits is small for a WI, preparing it for all WIs isn’t realistic
     - [# of lights] x [# of WIs]

# IMPROVING TREE TRAVERSAL & SHADING

## WORK ITEM DIVERGENCE

- BVH traversal
- When we hit a leaf node
  - Shadow ray casting
  - Shading
  - Accumulation
- Not all WI hits leaf at the same time
  - Divergence

```
while( nodeIdx )
{
    Node node = getNode( nodeIdx );
    if( hit( node, ray ) )
    {
        if( isLeaf( node ) )
        {
            Ray shadowRay = createRay( node, ray );
            if(!intersect( shadowRay) )
            {
                pixel += shade( node, ray );
            }
            nodeIdx = node.m_next;
        }
        else
        {
            nodeIdx = node.m_child;
        }
    }
    else
    {
        nodeIdx = node.m_next;
    }
}
```

Expensive computation deep in branches => Very bad

# IMPROVING TREE TRAVERSAL & SHADING

## MEMORY FOOTPRINT

- Don’t shade in the tree traversal
- Store light index in a buffer, process (shade) them later
- Divergence in the tree traversal is resolved, but
  - Don’t know how many lights overlaps
  - Storage of hit index can be huge
    - Don’t know the upper bound

```
while( nodeIdx )
{
    Node node = getNode( nodeIdx );
    if( hit( node, ray ) )
    {
        if( isLeaf( node ) )
        {
            hitList[nHits++] = nodeIdx;
            nodeIdx = node.m_next;
        }
        else
        {
            nodeIdx = node.m_child;
        }
    }
    else
    {
        nodeIdx = node.m_next;
    }
}

for(i=0; i<nHits; i++)
{
    Node node = getNode( hitList[i] );
    Ray shadowRay = createRay( node, ray );
    if(!intersect( shadowRay) )
    {
        pixel += shade( node, ray );
    }
}
```

Isolate expensive computation
Loop over nHits, which varies a lot => Bad

# IMPROVING TREE TRAVERSAL & SHADING

## RESERVOIR SAMPLING

- Reservoir sampling [Vitter85]
  - Select at most *k* items without storing all the candidates
  - Only need storage for *k* items

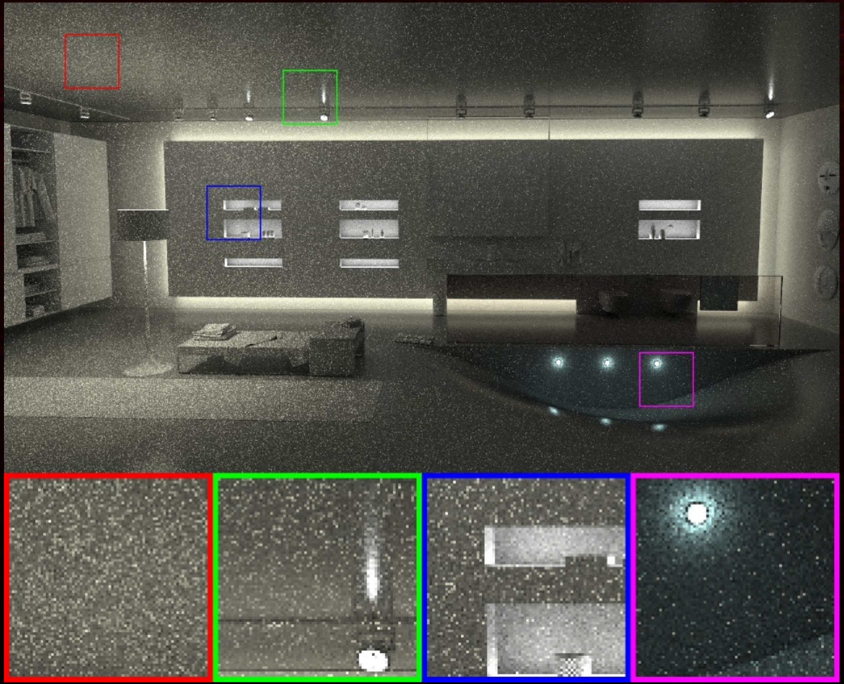
```
while( nodeIdx )
{
    Node node = getNode( nodeIdx );
    if( hit( node, ray ) )
    {
        if( isLeaf( node ) )
        {
            resevoirSampling( hitList, nodeIdx );
            nodeIdx = node.m_next;
        }
        else
        {
            nodeIdx = node.m_child;
        }
    }
    else
    {
        nodeIdx = node.m_next;
    }
}

for(i=0; i<resevoirMax; i++)
{
    if( nHits <= i )
        continue;
    Node node = getNode( hitList[i] );
    Ray shadowRay = createRay( node, ray );
    if(!intersect( shadowRay) )
    {
        pixel += shade( node, ray );
    }
}
```

Loop at most reservoirMax (constant) => Good ☺

# RESULTS

**Uniform Sampling (RSME:0.0749)**

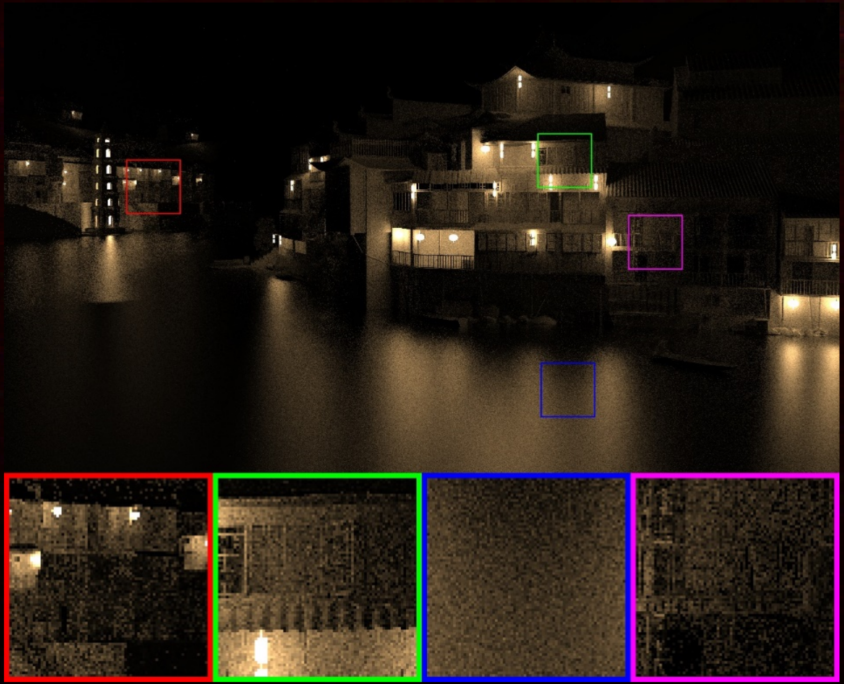


**Stochastic Light Culling (RSME:0.0464)**

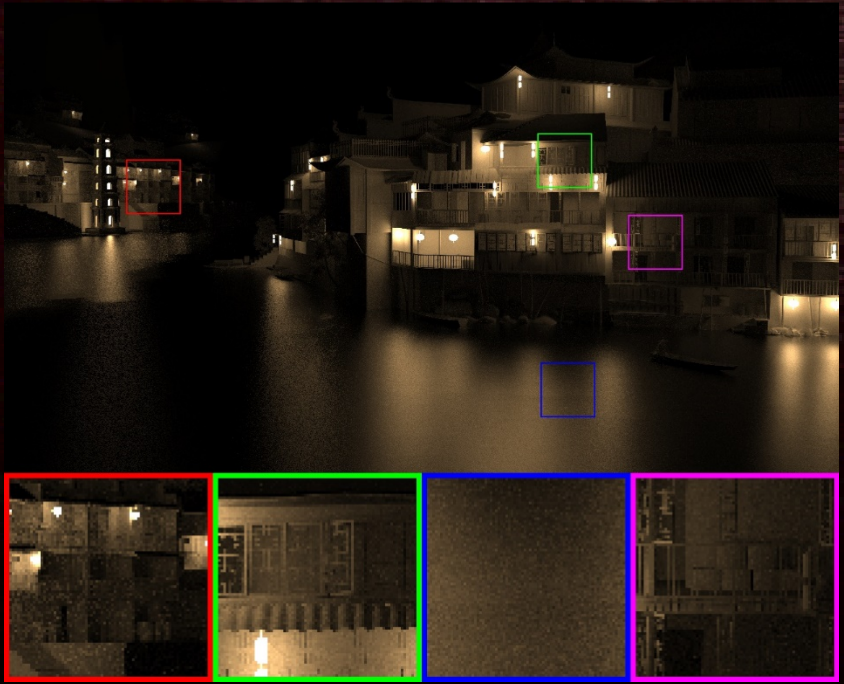
5,000 triangle lights, after 2min

# RESULTS

**Uniform Sampling (RSME:0.0355)**



**Stochastic Light Culling (RSME:0.0203)**



59,000 triangle lights, after 2min

# CLOSING

- Introduced Stochastic Light Culling
  - Can cull lights without bias (darkening)

- Presented two applications
  - Real-time GI using VPLs
    - VPLs + interleaved sampling
  - Interactive GI using path tracing
    - Extension to area lights
    - GPU optimization

# QUESTIONS?

**Yusuke Tokuyoshi and Takahiro Harada, Stochastic Light Culling, JCGT, vol. 5, no. 1, 35-60, 2016**

# REFERENCES


- ARVO, J., AND KIRK, D. 1990. Particle transport and image synthesis. SIGGRAPH Comput. Graph. 24, 4, 63–66.
- ANDERSSON, J. 2011. Directx 11 rendering in Battlefield 3. In GDC ’11.
- DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In I3D ’05, 203–231.
- DACHSBACHER, C., AND STAMMINGER, M. 2006. Splatting indirect illumination. In I3D ’06, 93–100.
- HARADA, T., MCKEE, J., AND YANG, J. C. 2012. Forward+: Bringing deferred lighting to the next level. In Eurographics ’12 Short Papers.
- KELLER, A. 1997. Instant radiosity. In Proc. SIGGRAPH’97, 49–56.
- OLSSON, O., AND ASSARSSON, U. 2011. Tiled shading. J. Graph. GPU, and Game Tools, 235–251.
- OLSSON, O., BILLETER, M., AND ASSARSSON, U. 2012. Clustered deferred and forward shading. In HPG’12, 87–96.
- SEGOVIA, B., IEHL, J. C., MITANCHEY, R., AND PE ROCHE, B. 2006. Non-interleaved deferred shading of interleaved sample patterns. In GH’06, 53–60.
- VEACH, E., AND GUIBAS, L. J. 1995. Optimally combining sampling techniques for Monte Carlo rendering. In SIGGRAPH ’95, 419–428.
- VITTER, J. S. 1985. Random sampling with a reservoir. ACM Trans. Math. Softw. 11, 1, 37–57.

# STOCHASTIC LIGHT CULLING FOR POINT LIGHTS

## DIRECT ILLUMINATION

### Path Tracing (Base)

```
HitInfo hit = scene.intersect( from, to );
if( !hit.hasHit() )
    continue;

float4 hp = from + ( to - from ) * hit.m_f;


// explicit connection
for(int il=0; il<nLightSamples; il++)
{
    const SampleInfo& l = ls[il];



    float g = geomTerm( hp, hit.m_ns, l.m_x, l.m_n );
    if( !scene.intersect( hp, l.m_x ).hasHit() )
    {
        float4 f = scene.brdfEvaluate( hit.m_ns, m );

        dst += f * l.m_le * g / l.m_pdfArea;
    }
}
```

### Path Tracing + SLC

```
HitInfo hit = scene.intersect( from, to );
if( !hit.hasHit() )
    continue;

float4 hp = from + ( to - from ) * hit.m_f;


// explicit connection (SLC)
for(int il=0; il<nLightSamples; il++)
{
    const SampleInfo& l = ls[il];
    const float d2 = l2( hp - l.m_x );
    if( SlcImpl::radius2( l.m_le, ALPHA, xi[il] ) < d2 )
        continue;

    float g = geomTerm( hp, hit.m_ns, l.m_x, l.m_n );
    if( !scene.intersect( hp, l.m_x ).hasHit() )
    {
        float4 f = scene.brdfEvaluate( hit.m_ns, m );
        float rrPdf = SlcImpl::computeRrPdf( hp, l, ALPHA );
        dst += f * l.m_le * g / (l.m_pdfArea * rrPdf);
    }
}
```

# STOCHASTIC LIGHT CULLING CODE

```
class SlcImpl
{
    public:
        static
        float computeRt( const float4& le, float alpha )
        {
            return sqrtf( dot3F4( float4(0.33f,0.33f,0.33f), le ) * ( 1.f/(M_PI*alpha) ) );
        }

        static
        float computeRrPdf( const float4& vtx, const float4& lvtx, const float4& le, float alpha )
        {
            float d2 = dot3F4( vtx-lvtx, vtx-lvtx );
            float r_t = computeRt( le, alpha );
            if( d2 > r_t*r_t )
                return r_t*r_t / d2;
            return 1.f;
        }

        static
        float radius2( float r_t, float xi )
        {
            return ( r_t*r_t / (1.f-xi) );
        }
};
```