CEDEC 2016
Computer Entertainment Developers Conference

確率的ライトカリング -理論と実装-

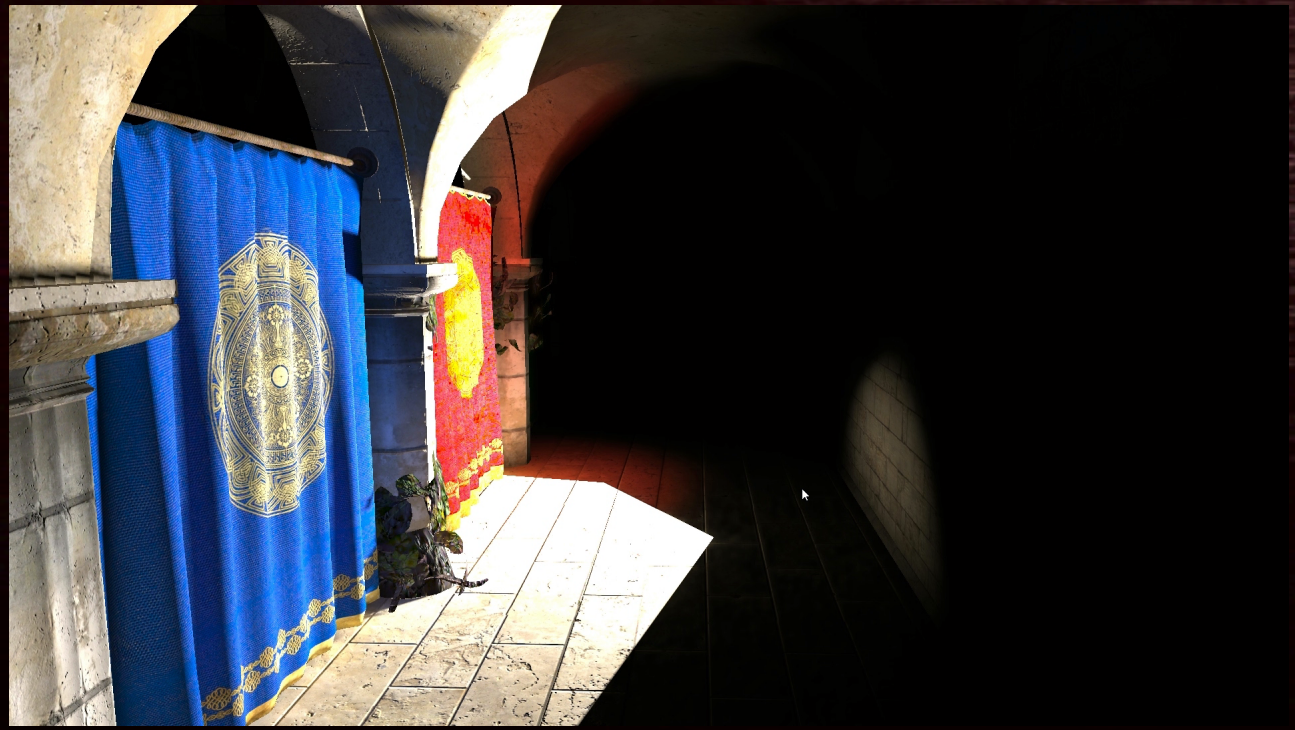
YUSUKE TOKUYOSHI (SQUARE ENIX CO., LTD.)
TAKAHIRO HARADA (ADVANCED MICRO DEVICES, INC.)

本講演の目的

- 数万の光源で照らされたシーンを比較的小さい誤差で描画する方法のご紹介
  - 論文: “Stochastic Light Culling” (http://jcgt.org/published/0005/01/02/)
- GPU実装の解説
  - リアルタイムレンダリング
  - オフラインパストレーシング

- 講演の対象者
  - レンダリングの基礎知識をもっている人
  - これからレンダリングの研究を始める人

VIDEO



間接照明（65536 virtual point lights）

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

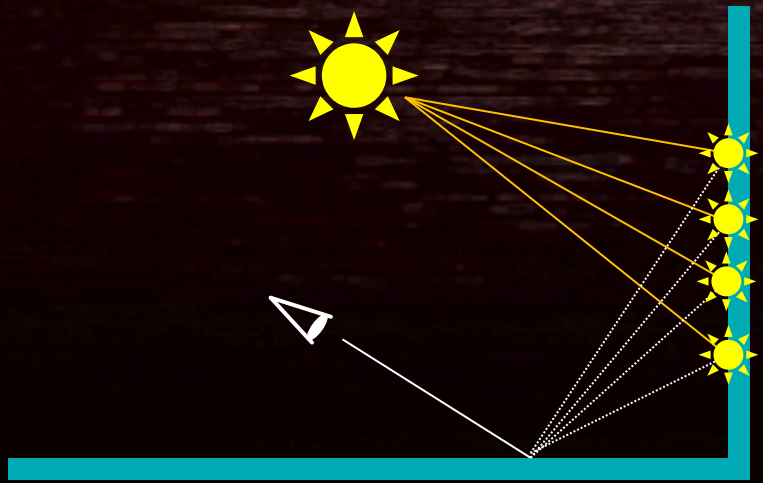
本日の流れ

- ライトカリングの説明
- 確率的ライトカリング
- 間接照明のリアルタイム実装例
- オフラインレンダリングへの応用
  - パストレーシング用のカリングアルゴリズム
  - GPU実装

LIGHT CULLING

多光源レンダリング

- 光源が大量に存在
  - 自動生成される光源
  - 間接照明を表現するvirtual point lights (VPLs) [Keller97]
- まじめにやると膨大な計算量

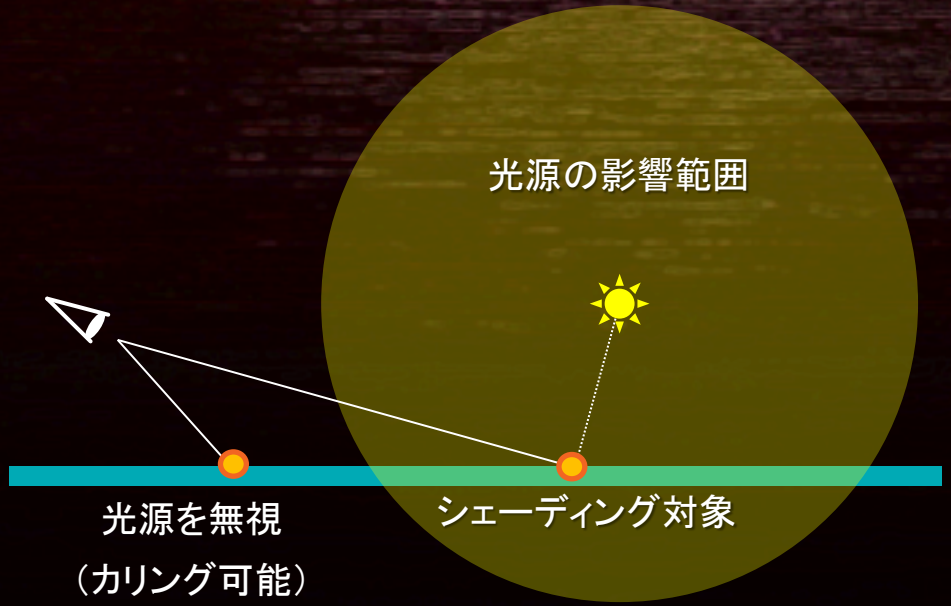


VPLsを使った間接照明の光路

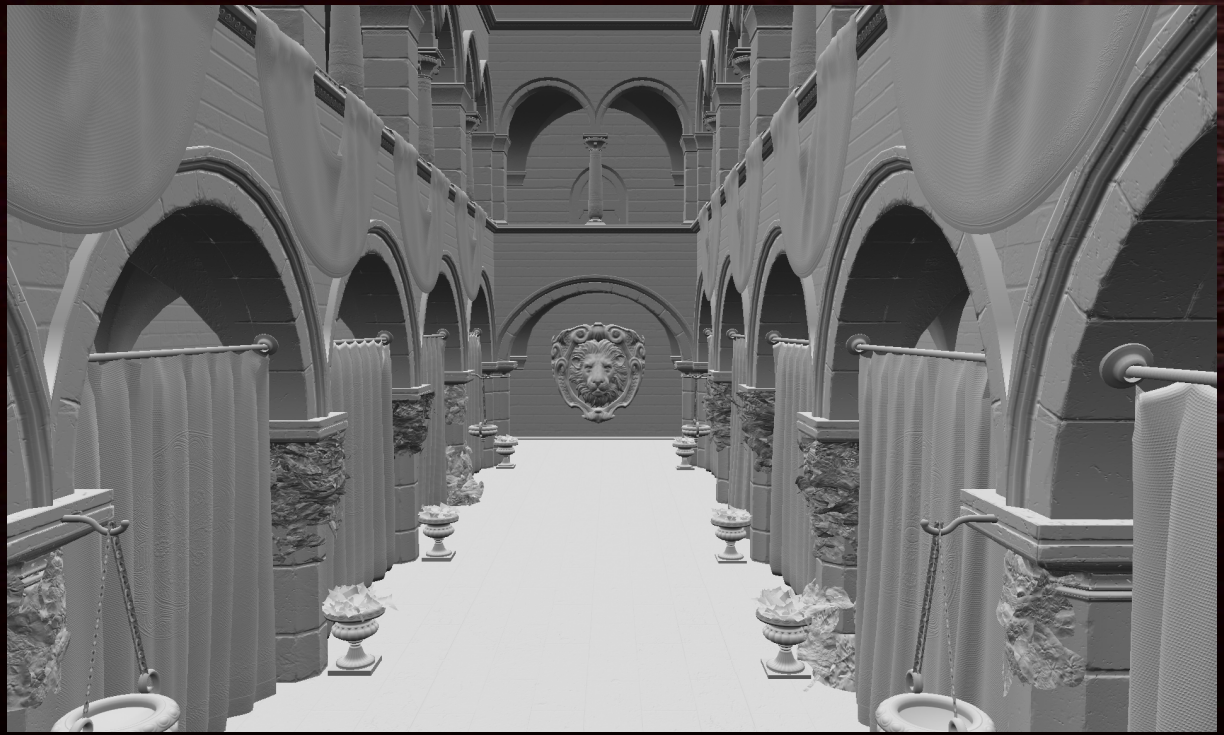
ライトカリング

シェーディング点から遠い光源は寄与が小さいので無視

- 光源毎に光の影響範囲を制限
- 範囲外はシェーディングの対象外
- 光源を事前にカリングして高速化

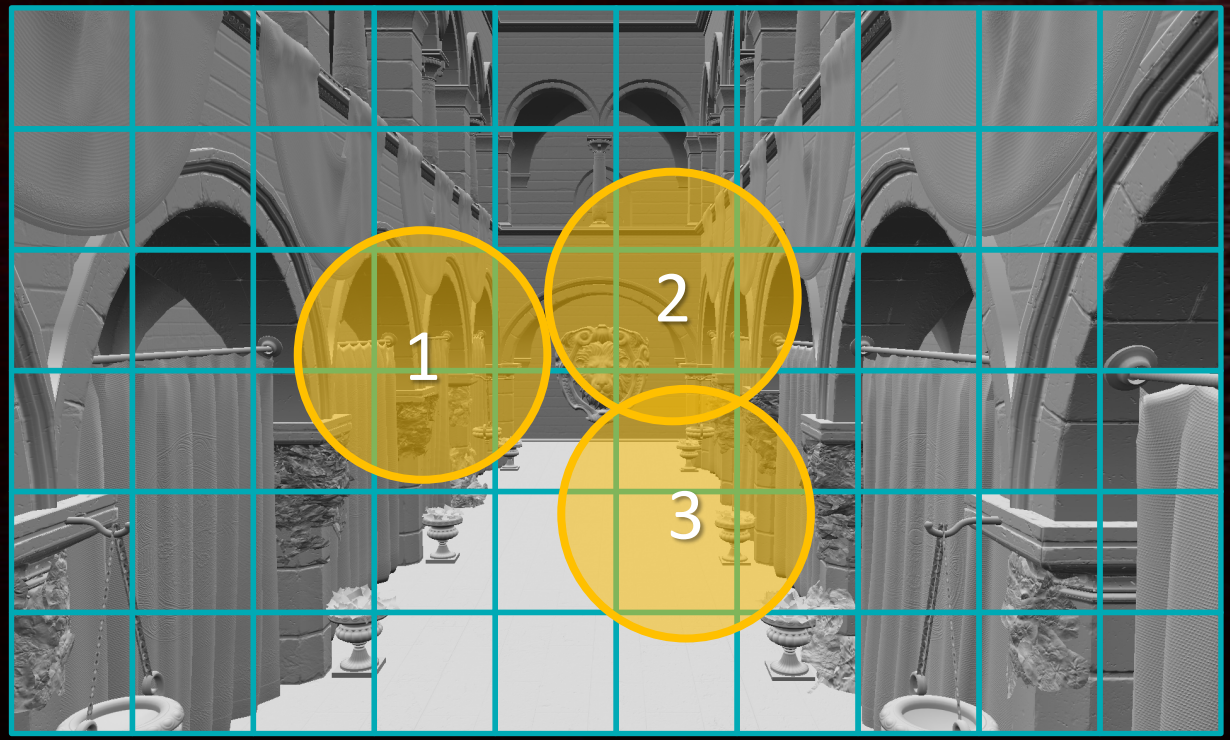


TILE-BASED CULLING [Olsson11; Harada12]

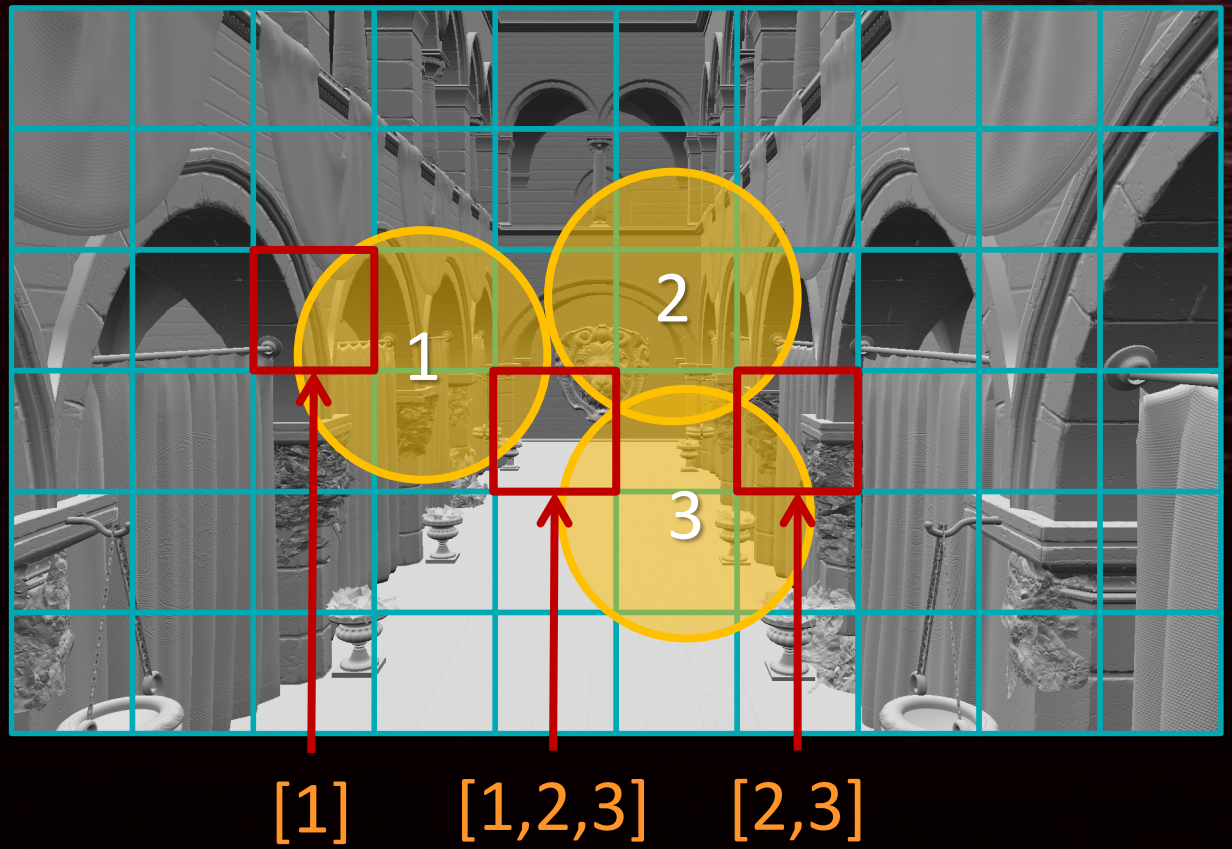


TILE-BASED CULLING [Olsson11; Harada12]

TILE-BASED CULLING [Olsson11; Harada12]

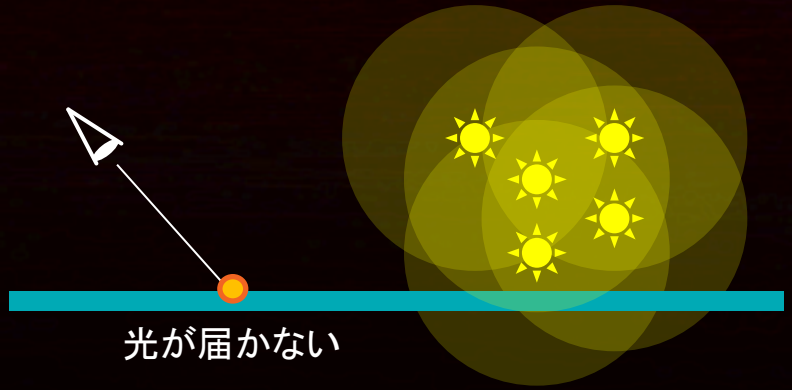


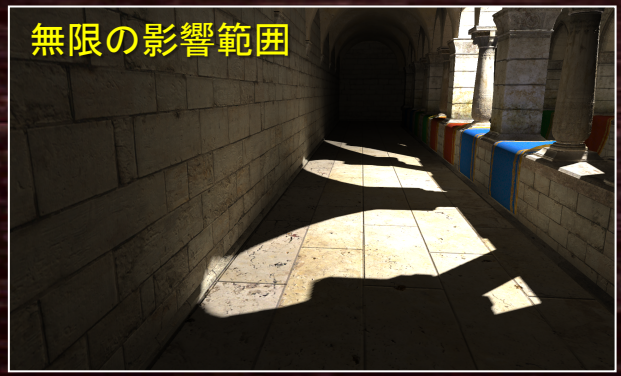
TILE-BASED CULLING [Olsson11; Harada12]

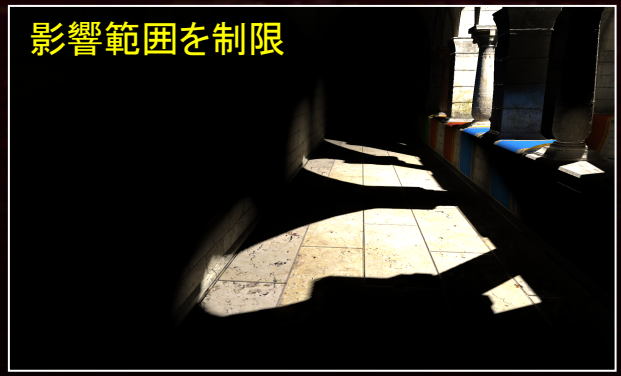


問題点

- 光源から遠い所が暗くなる
  - –実際の光は無限の影響範囲を持つ
  - –物理ベースの光源を扱えない (e.g., VPLs)
- 光源が多いほど誤差(bias)が累積







65536 VPLs使った間接照明

BIAS

- 解に対する推定値の偏りを表す誤差
- ライトカリングは全光源が暗めに偏っている
- Biasのある推定が常に解に収束しないとは限らないが、この場合は収束しない☹

CEDEC 2016
Computer Entertainment Developers Conference

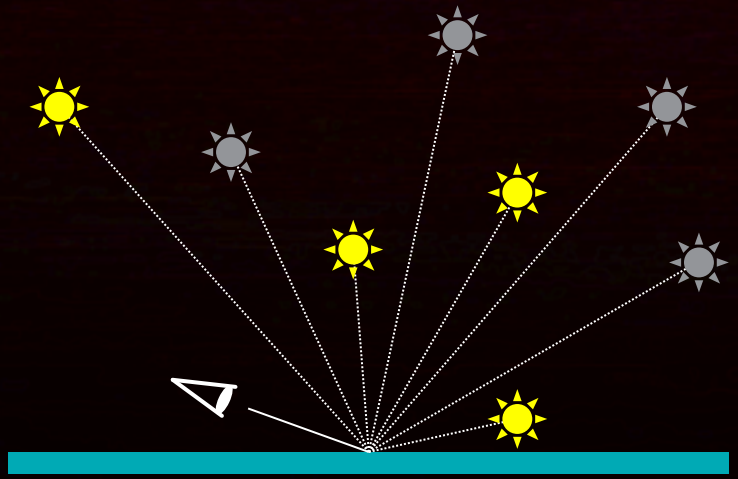
STOCHASTIC LIGHT CULLING

手法の概要

- 基本的にロシアンルーレット法[Arvo90]
- 光の影響範囲をランダムに変更
  – 遠い光源も低確率でサンプリング
- Biasを分散（推定値のばらつきを表す誤差）に変える
- 光源数が多くなっても誤差が累積しない

ロシアンルーレット法

- 各光源を確率的に殺す
- 生存確率: シェーディング点から見た光源の明るさに比例
- 生き残った光源のエネルギーは生存確率で割る



光源までの距離

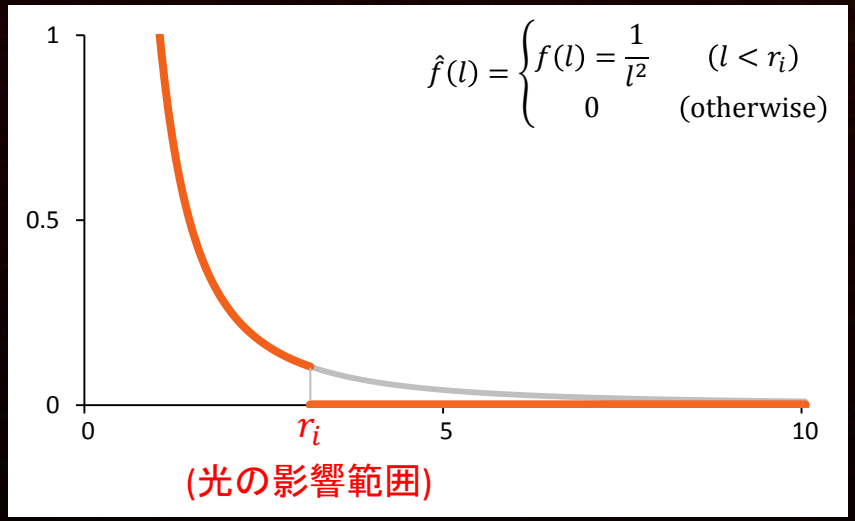
光源のfall-off関数: $f(l) = \frac{1}{l^2}$

$$\text{生存確率: } p_i(l) = \min\left(\frac{f(l)}{\alpha_i}, 1\right)$$

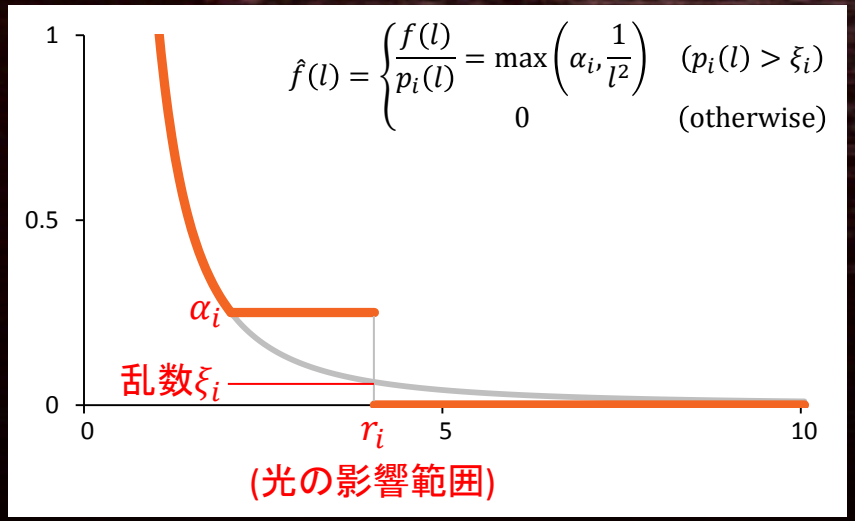
ユーザー定義の係数
大きいほど死にやすい

確率的FALL-OFF関数

Clamping

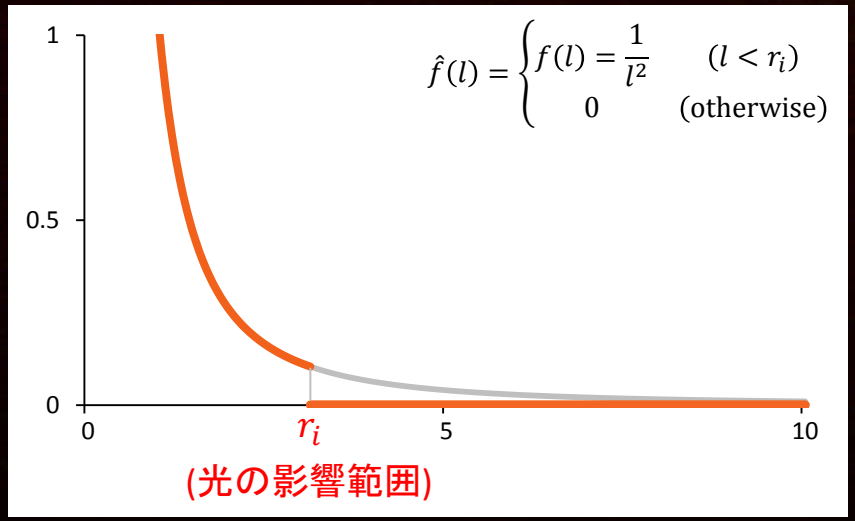


Stochastic

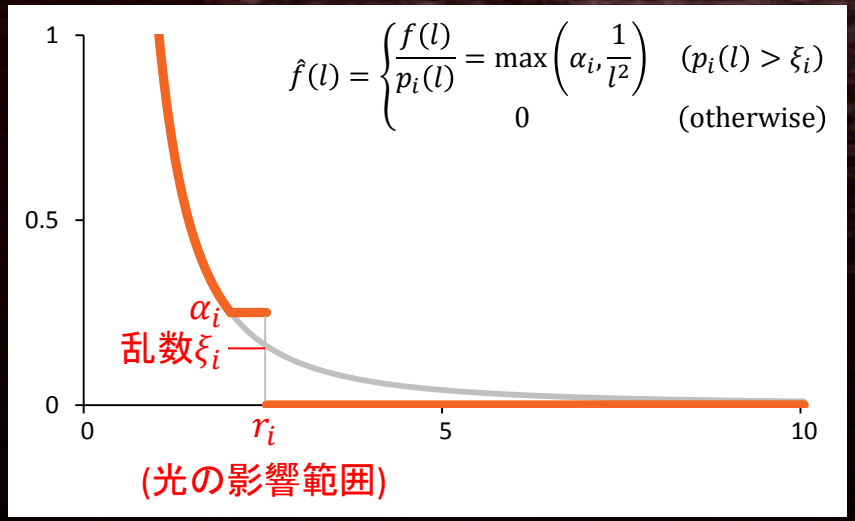


確率的FALL-OFF関数

Clamping

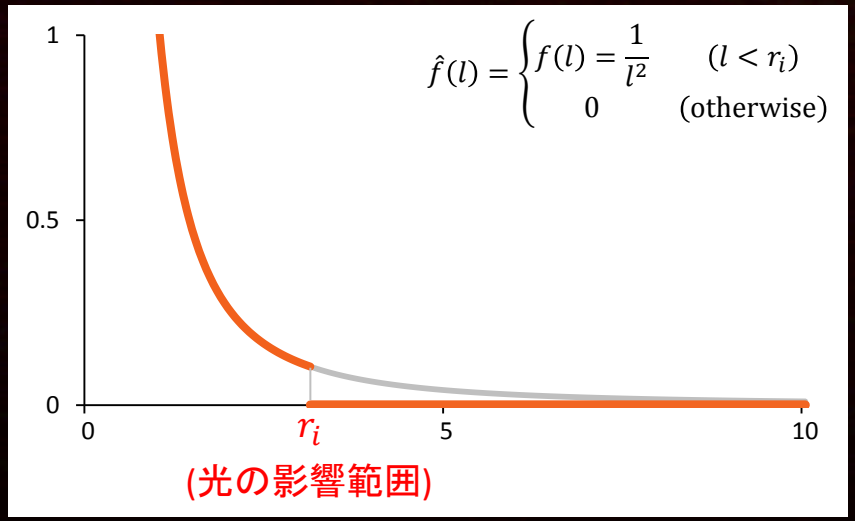


Stochastic

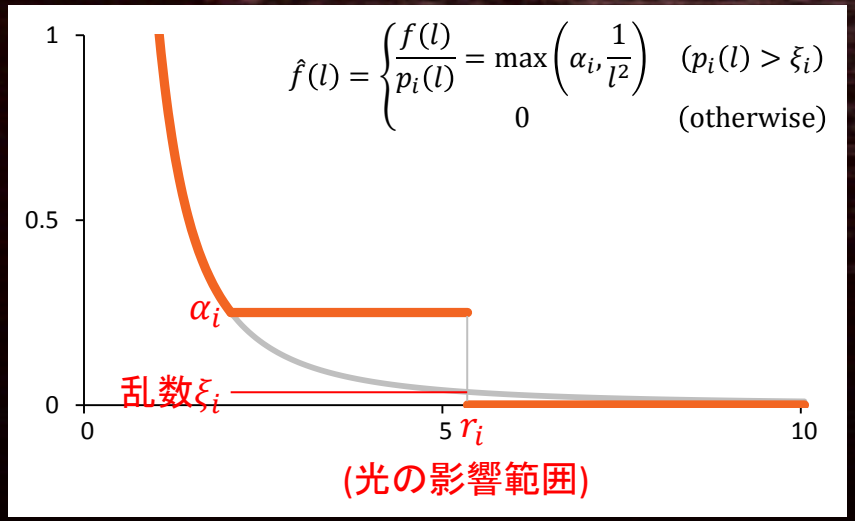


確率的FALL-OFF関数

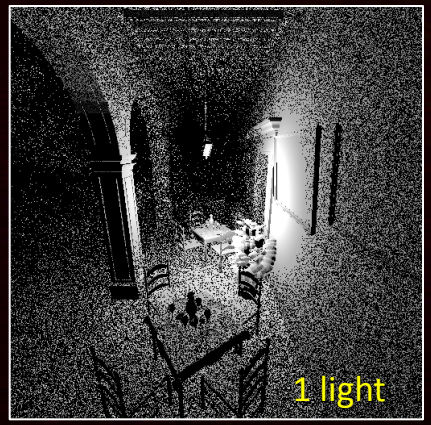
Clamping



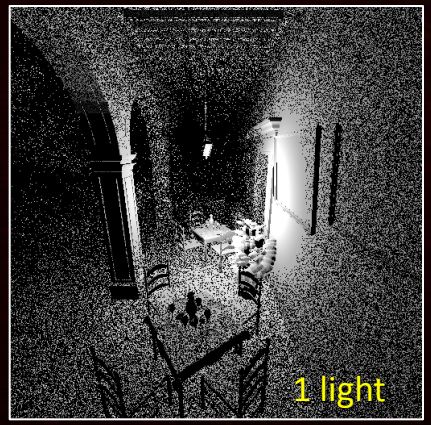
Stochastic



ランダムな光の影響範囲



ランダムな光の影響範囲



ランダムな光の影響範囲

- シェーディング点毎に異なる光の影響範囲。事前にカリングできない☹
- 共通の範囲を使いたい

ランダムな光の影響範囲

- シェーディング点毎に異なる光の影響範囲。事前にカリングできない☹
- 共通の範囲を使いたい
- 解決法: 全シェーディング点で同じ乱数を使用

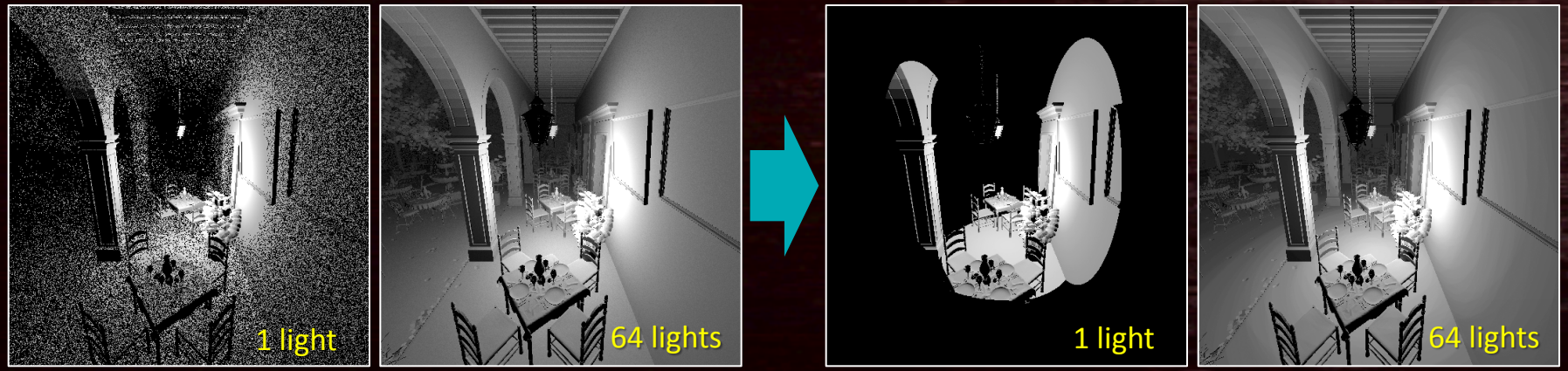
ランダムな光の影響範囲

- シェーディング点毎に異なる光の影響範囲。事前にカリングできない☹
- 共通の範囲を使いたい
- 解決法: 全シェーディング点で同じ乱数を使用

ランダムな光の影響範囲

- シェーディング点毎に異なる光の影響範囲。事前にカリングできない☹
- 共通の範囲を使いたい
- 解決法: 全シェーディング点で同じ乱数を使用
  - 推定に偏りがあるように見えるかもしれないが、実は偏ってない
  - Unbiased coherent sampling

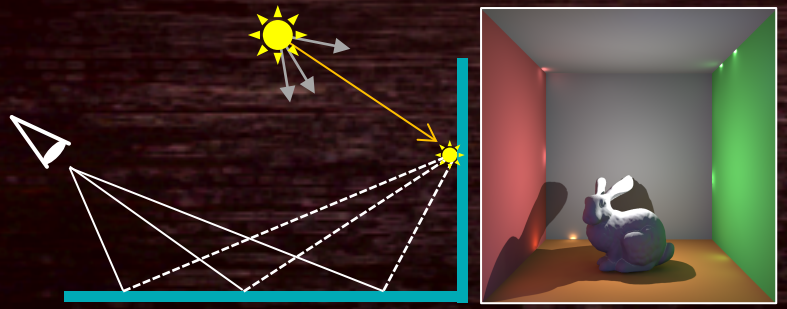
ランダムな光の影響範囲



- シェーディング点毎に異なる光の影響範囲。事前にカリングできない☹
- 共通の範囲を使いたい
- 解決法: 全シェーディング点で同じ乱数を使用
  - 推定に偏りがあるように見えるかもしれないが、実は偏ってない
  - Unbiased coherent sampling

UNBIASED COHERENT SAMPLING

- 関連研究
  - Virtual point lights (VPLs) [Keller97]
  - Lightcuts [Walter05]
  - Coherent path tracing [Sadeghi09]

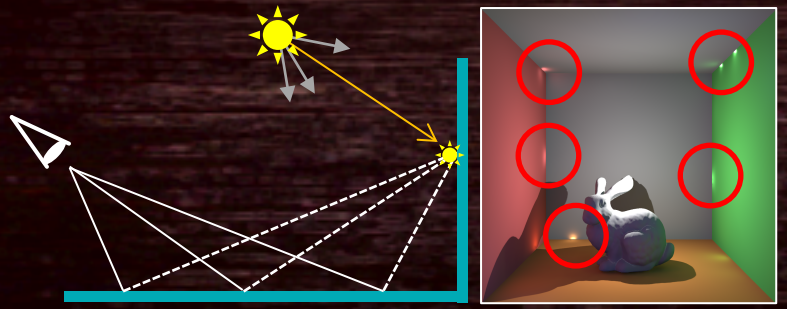


複数のシェーディング点が同じサンプル光路（VPL）を共有

UNBIASED COHERENT SAMPLING

- 関連研究
  - Virtual point lights (VPLs) [Keller97]
  - Lightcuts [Walter05]
  - Coherent path tracing [Sadeghi09]

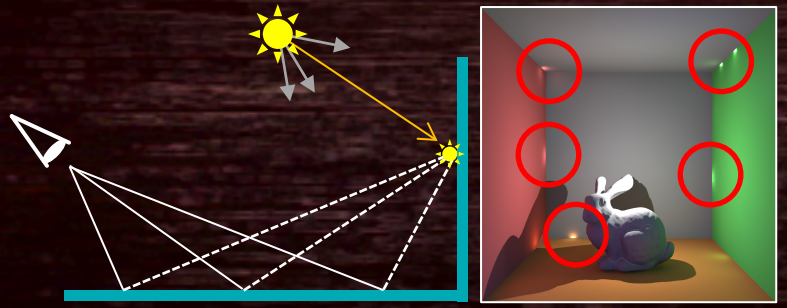
- “分散 = 高周波ノイズ”とは限らない
  - 明るい斑点模様
  - フリッカリング



複数のシェーディング点が同じサンプル光路（VPL）を共有

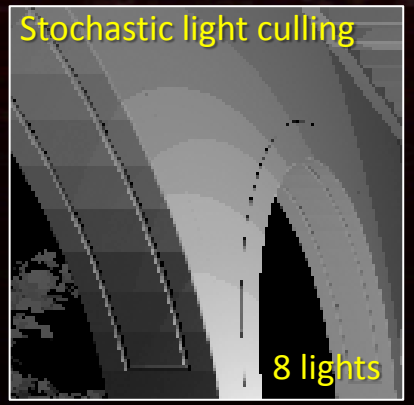
UNBIASED COHERENT SAMPLING

- 関連研究
  - Virtual point lights (VPLs) [Keller97]
  - Lightcuts [Walter05]
  - Coherent path tracing [Sadeghi09]



複数のシェーディング点が同じサンプル光路（VPL）を共有

- “分散 = 高周波ノイズ”とは限らない
  - 明るい斑点模様
  - フリッカリング
  - バンディングアーティファクト



“UNBIASED”の意味

- 誤差が偏りなくばらつくこと
- 誤差が0に収束することは意味してない
  - よくある誤解のひとつ
    - T. Hachisuka. 2013. "*Five Common Misconceptions about Bias in Light Transport Simulation*"

- あえて誤差を0に収束させないunbiasedな手法もある (e.g., lightcuts)
  - 許容誤差量を設定して過剰な計算を省く
  - 誤差は許容量以下になるが、0には収束しない
  - 実用的☺

許容誤差量の設定

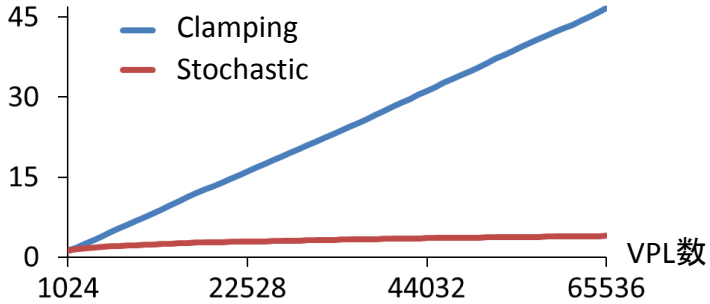
- 分散の量を簡単にコントロール☺
- VPLsではシェーディング対象の平均光源数がほぼ一定に収束☺
- ユーザー定義の許容誤差量ϵ_{max}から生存確率のパラメータα_iを計算
  - 拡散反射を仮定
  - 光源の放射強度$I(\omega')$も低周波な分布を仮定

$$\alpha_i = \frac{2\pi\epsilon_{max}}{E\max_{\omega'}\left(I(\omega')\right)}$$

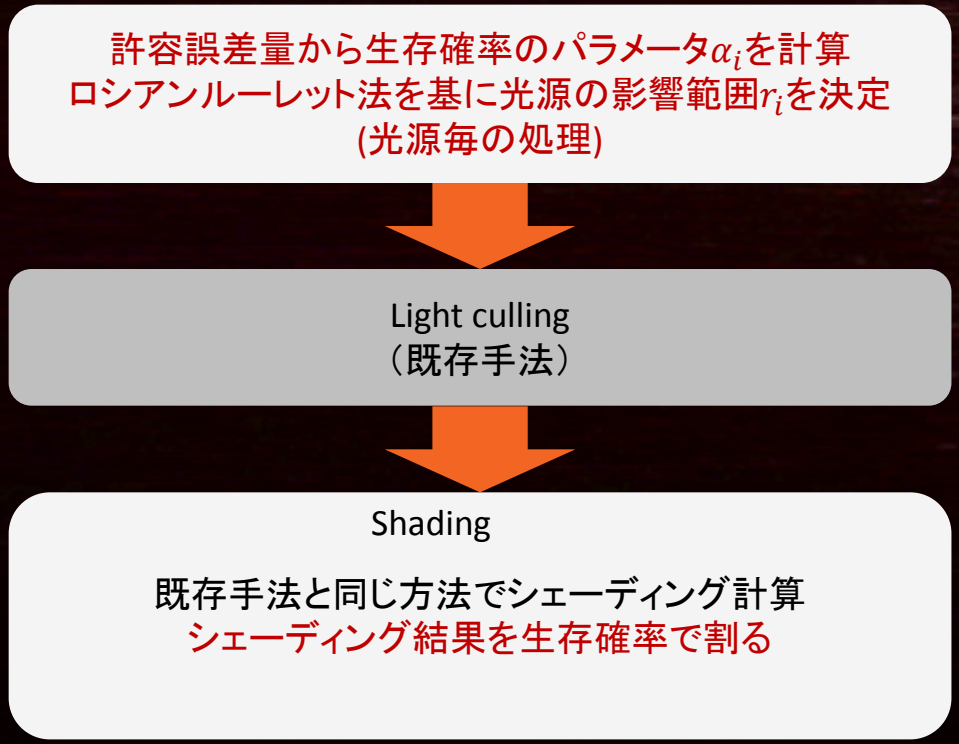
カメラの露出

VPLの場合、VPL数Nに反比例: $I(\omega') \propto \frac{1}{N}$

同じ許容誤差でのライティング時間 (ms)



本手法の処理の流れ



← カリング手法に依存しない

Splatting [Dachsbacher06]

Tile-based culling [Olsson11; Harada12]

Clustered shading [Olasson12]

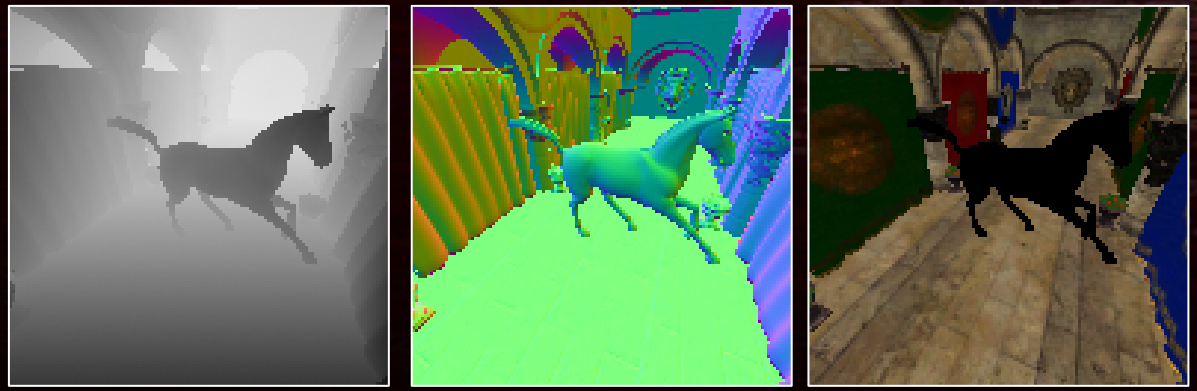
…etc.

赤字が新たに追加された処理

EXAMPLE IMPLEMENTATION OF REAL-TIME INDIRECT ILLUMINATION

VIRTUAL POINT LIGHTS (VPLS)の生成

- Reflective shadow maps (RSMs) [Dachsbacher05]
- 各テクセル = VPLs

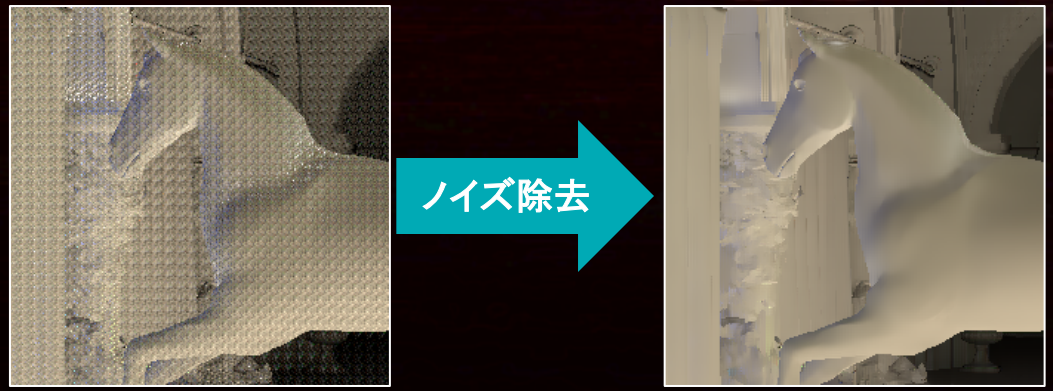


RESAMPLING VPLS

- 論文ではRSMから重要なVPLを再サンプリングした
- 重要度: bidirectional RSM [Ritschel11]
- 速度優先なら、再サンプリングせず代わりに低解像度のRSMを使ってもいい

INTERLEAVED SAMPLING

- ピクセル毎に異なる光源を使ってシェーディング [Wald02]
  - 8×8ピクセルのサンプリングパターン → ピクセルあたりの光源数は1/64に
- 誤差（分散）は高周波ノイズとして画面に現れる
- 後処理でノイズ除去フィルタリング



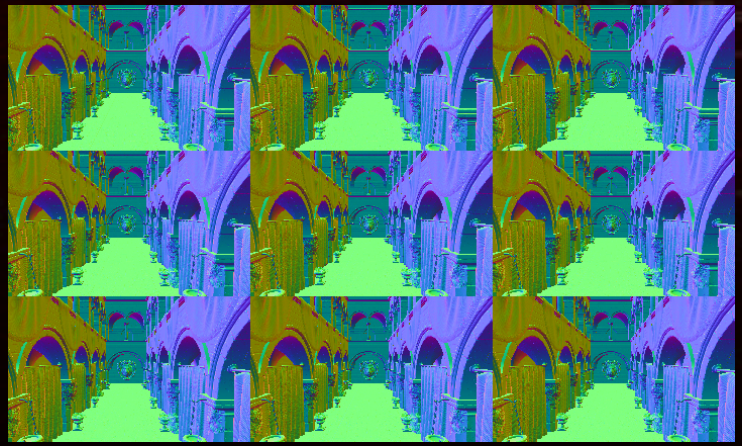
8×8 interleaved sampling

INTERLEAVED SAMPLINGのGPU実装

- ピクセルに合わせてスレッドを割り当てるとdivergenceが発生☹
- 画面を8×8の部分領域にピクセルを並び替えて回避 [Segovia06]
  - 部分領域内では同じ光源が使用される☺
  - Single passで実装

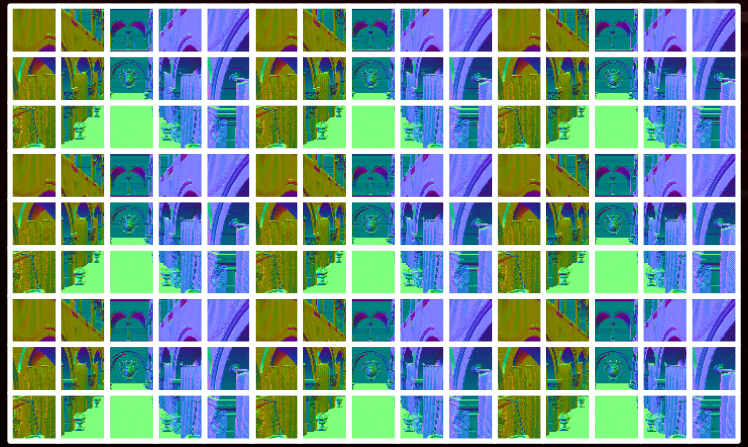
ライトカリングを適用

- Interleaved samplingと確率的ライトカリングの組み合わせ
- 並び替えた部分領域に対してtiled deferred shading [Andersson11]
  - 部分領域あたりの光源数は1/64に減っている
  - さらに確率的にカリングしてピクセルあたりの光源数を減らす



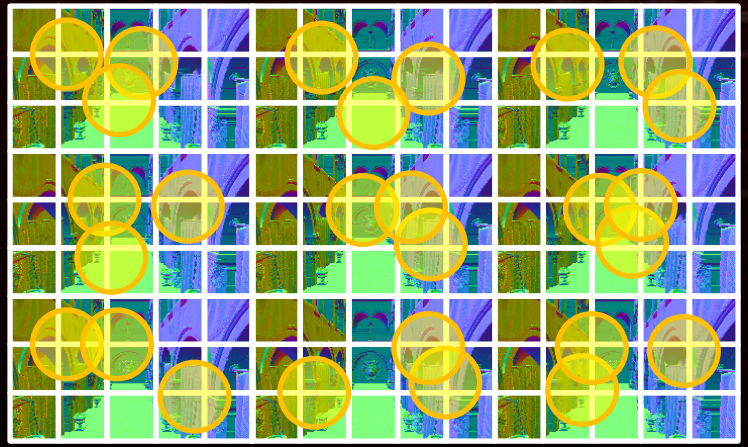
ライトカリングを適用

- Interleaved samplingと確率的ライトカリングの組み合わせ
- 並び替えた部分領域に対してtiled deferred shading [Andersson11]
  - 部分領域あたりの光源数は1/64に減っている
  - さらに確率的にカリングしてピクセルあたりの光源数を減らす



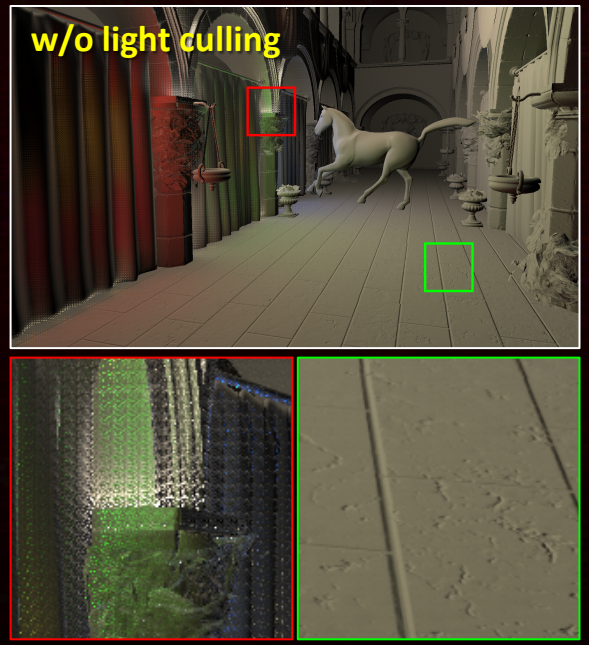
ライトカリングを適用

- Interleaved samplingと確率的ライトカリングの組み合わせ
- 並び替えた部分領域に対してtiled deferred shading [Andersson11]
  - 部分領域あたりの光源数は1/64に減っている
  - さらに確率的にカリングしてピクセルあたりの光源数を減らす

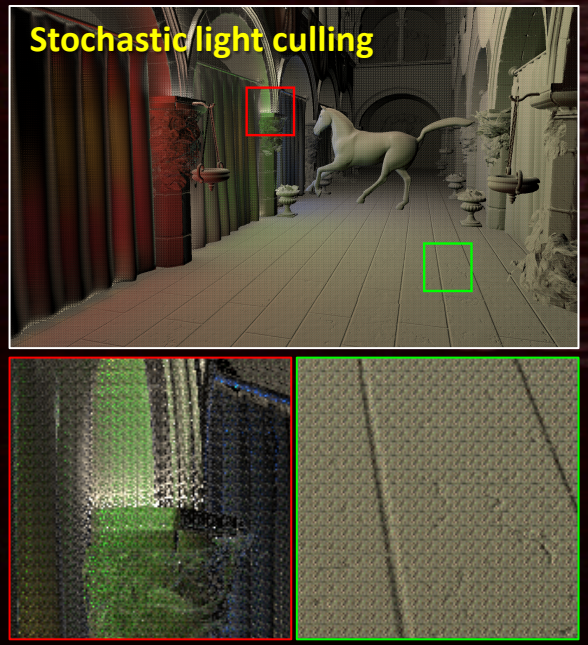


RESULTS

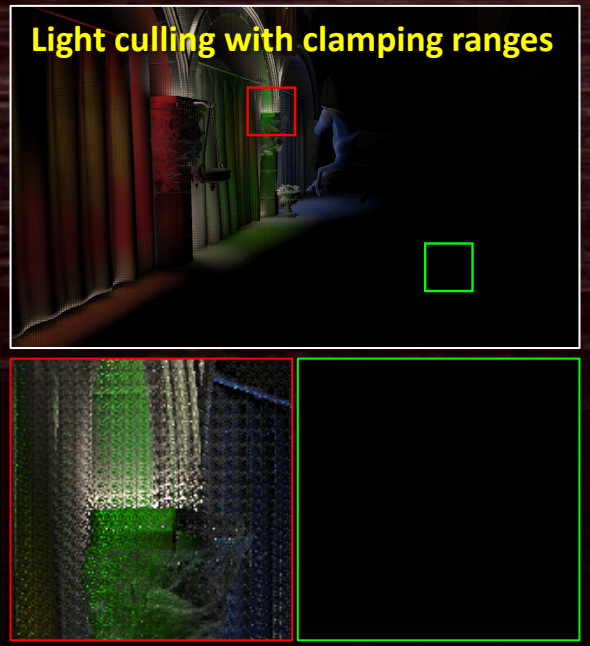
フィルタリング前の間接照明成分



Shading time: 44.4 ms



Shading time: 2.87 ms



Shading time: 2.87 ms

65536 VPLs, Resolution: 1920×1152, $\epsilon_{max} = 0.0005$, GPU: AMD Radeon™ R9 290X

RESULTS

ノイズ除去フィルタリング、テクスチャ適用、直接照明追加

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, $\epsilon_{max} = 0.0005$, GPU: AMD Radeon™ R9 290X

RESULTS
同じ計算時間での比較

1024 VPLs

Stochastic light culling

Light culling with clamping ranges

Resolution: 1920×1152, $\epsilon_{max} = 0.0005$, GPU: AMD Radeon™ R9 290X

RESULTS
同じ計算時間での比較

4096 VPLs

Stochastic light culling

Light culling with clamping ranges

Resolution: 1920×1152, $\epsilon_{max} = 0.0005$, GPU: AMD Radeon™ R9 290X

RESULTS
同じ計算時間での比較

16384 VPLs

Stochastic light culling

Light culling with clamping ranges

Resolution: 1920×1152, $\epsilon_{max} = 0.0005$, GPU: AMD Radeon™ R9 290X

RESULTS
同じ計算時間での比較

65536 VPLs

Stochastic light culling

Light culling with clamping ranges

Resolution: 1920×1152, $\epsilon_{max} = 0.0005$, GPU: AMD Radeon™ R9 290X

ここまでのまとめ

- 確率的ライトカリング
  - 誤差を考慮したライトカリング
  - Biasを分散に変えて誤差を収束
- リアルタイム実装
  - カリング自体の実装に変更なし
  - Interleaved samplingと組み合わせると効果的
- 制限事項
  - 低周波のBRDFを仮定

CEDEC 2016
Computer Entertainment Developers Conference

STOCHASTIC LIGHT CULLING FOR PROGRESSIVE PATH TRACING

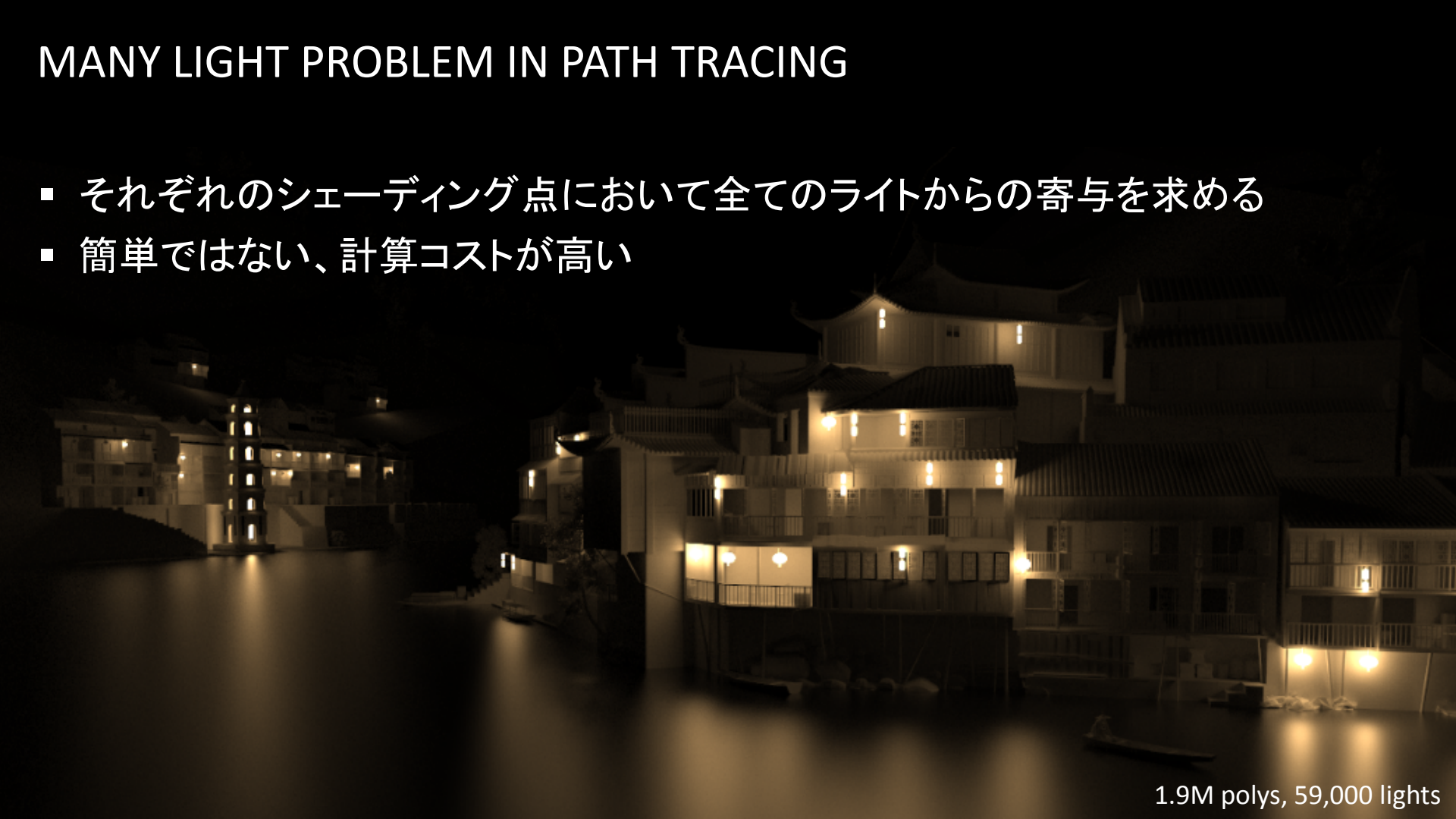
AGENDA

1. イントロ
   – 確率的ライトカリング + パストレーシング（ポイントライトのみ）
2. さらなる詳細
   – 確率的ライトカリング + パストレーシング（全部入り）
3. GPUでの実装

- Demo time!!

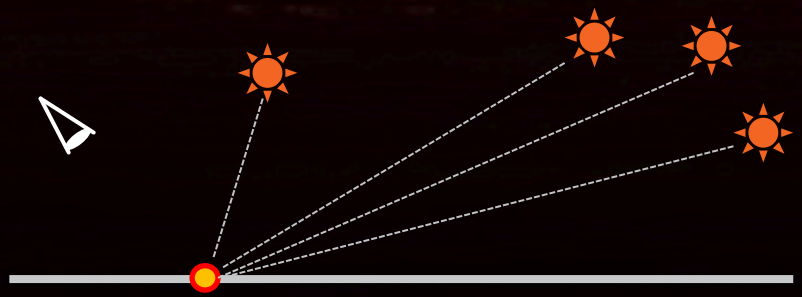
MANY LIGHT PROBLEM IN PATH TRACING

- それぞれのシェーディング点において全てのライトからの寄与を求める
- 簡単ではない、計算コストが高い



MANY LIGHT PROBLEM IN PATH TRACING

- それぞれのシェーディング点において全てのライトからの寄与を求める
- 手法として二つの可能性
  1. それぞれのライトにおいて、一つ頂点をサンプリング
     - 多くライトがあるシーンにおいて計算負荷が高い
     - 1000ライトあれば、1000回シャドウレイをトレースしなければならない



MANY LIGHT PROBLEM IN PATH TRACING

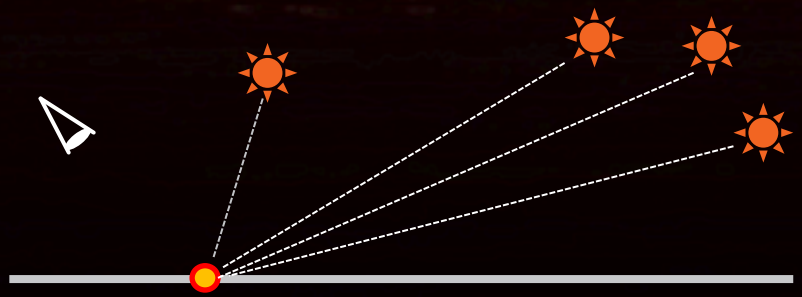
- それぞれのシェーディング点において全てのライトからの寄与を求める
- 手法として二つの可能性
  1. それぞれのライトにおいて、一つ頂点をサンプリング
     - 多くライトがあるシーンにおいて計算負荷が高い
     - 1000ライトあれば、1000回シャドウレイをトレースしなければならない
  2. ライトを一つ選択し、一つ頂点をサンプリング
     - 収束はライト選択する戦略に依存する
     - プログレッシブパストレーシングではより実用的

MANY LIGHT PROBLEM IN PATH TRACING

- それぞれのシェーディング点において全てのライトからの寄与を求める
- 手法として二つの可能性
  1. それぞれのライトにおいて、一つ頂点をサンプリング
     - 多くライトがあるシーンにおいて計算負荷が高い
     - 1000ライトあれば、1000回シャドウレイをトレースしなければならない
     - 確率的ライトカリング => バイアスを加えることなくシャドウレイの数を減らす
  2. ライトを一つ選択し、一つ頂点をサンプリング
     - 収束はライト選択する戦略に依存する
     - プログレッシブパストレーシングではより実用的

W.O. STOCHASTIC LIGHT CULLING (POINT LIGHT)

- それぞれのシェーディング点において
  - For 全てのライト (4ライト)
    - シャドウレイをトレース (visibility)
    - シェーディングし、寄与を足し合わせる

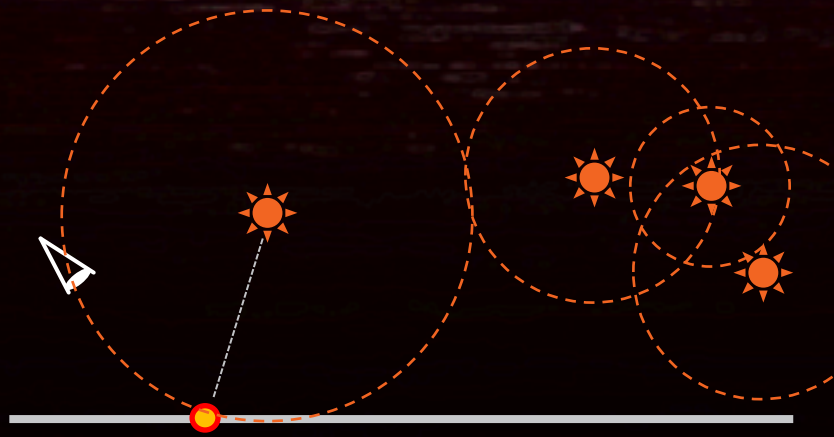


STOCHASTIC LIGHT CULLING (POINT LIGHT)

UNOPTIMIZED

- それぞれのシェーディング点において
  - For 全てのライト (4ライト)
    - If(!overlaps(ライトの影響範囲, 点))
      - continue
    - シャドウレイをトレース (visibility)
    - シェーディングし、寄与を足し合わせる

- ライトの影響範囲は全てのシェーディング点に対して固定
- それぞれの点に重なっているライトを代わりに見つける方が効率的

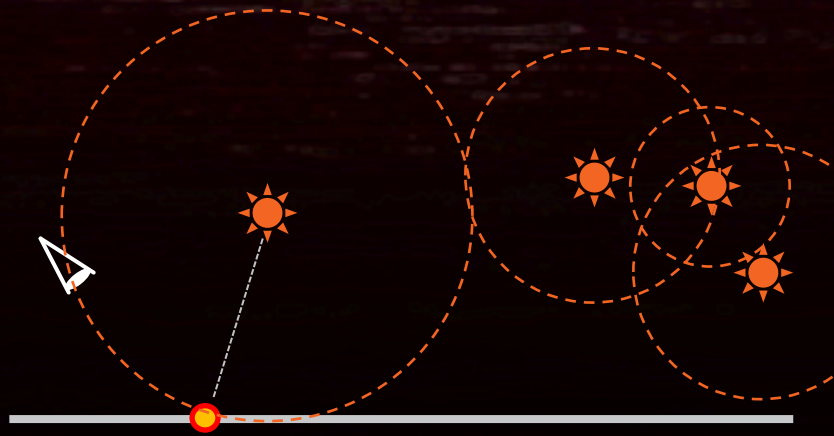


Stochastic Light Culling

STOCHASTIC LIGHT CULLING (POINT LIGHT)

OPTIMIZED

- それぞれのシェーディング点において
  - For 重なっている ライト (1ライト)
    - ライト上の頂点を一つ選ぶ
    - シャドウレイをトレース (visibility)
    - シェーディングし、寄与を足し合わせる



Stochastic Light Culling

STOCHASTIC LIGHT CULLING FOR POINT LIGHTS

DIRECT ILLUMINATION

PT (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;
    }
}
```

PT + 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);
    }
}
```

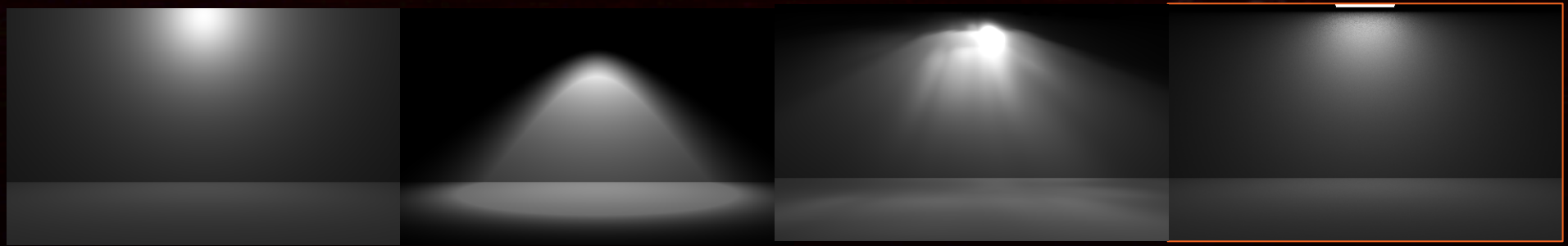
STOCHASTIC LIGHT CULLING CODE


```
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) );
        }
};
```

SPECIFICS FOR PATH TRACING

様々なライトの種類

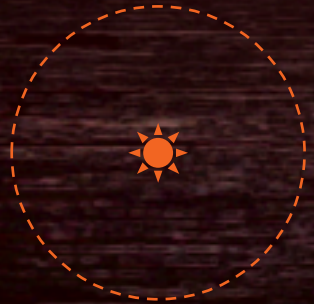
- ポイントライトだけの説明をした
- パストレーシングでは他にも様々なライトが使われる
- ほとんどのライトはエリアライト
  - 確率的ライトカリングはエリアライトに使える??



Point ☺ Spot ☺ IES ☺ Area ☹

エリアライトの影響範囲

- ライトの影響範囲を計算する必要あり
- どうやって??

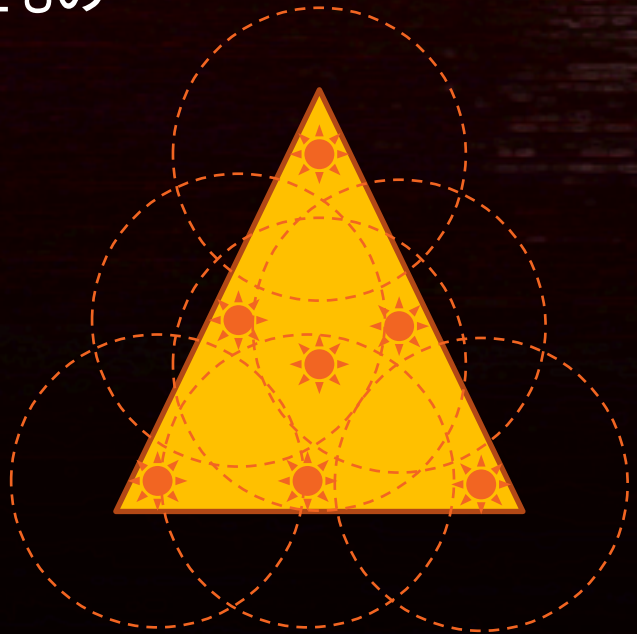


Point light

Area light

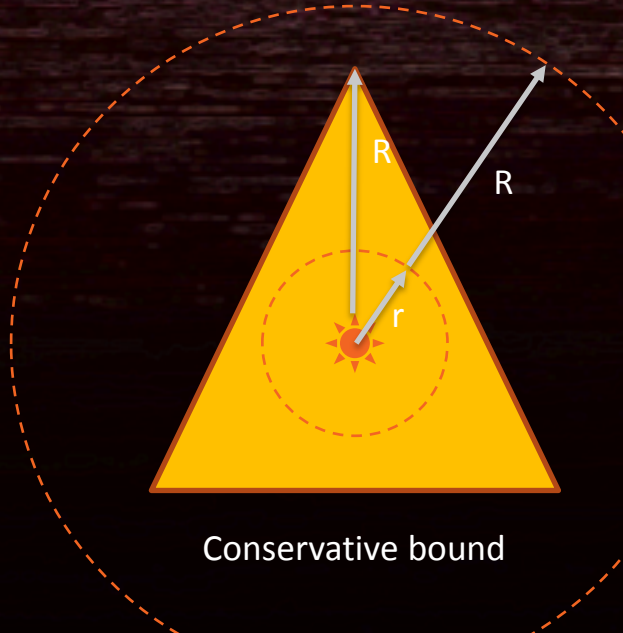
エリアライトの影響範囲

- ライトの影響範囲を計算する必要あり
- エリアライト == ポイントライトの集合
- 球をエッジ上で掃引したもの
  - 簡単には求まらない



エリアライトの影響範囲

- ライトの影響範囲を計算する必要あり
- エリアライト == ポイントライトの集合
- Conservativeな影響範囲を計算
  - 一つの球として表現
  - エリアライトの中心を求める
  - 中心からエッジまでの最大距離(R)
  - 球の半径を求める
    - R+r



MULTIPLE IMPORTANCE SAMPLING (MIS)

- カリングされる確率がはっきりと定義されている
- MISを導入するのが容易 (implicit connection + explicit connection)
- Implicit connectionにおいて、ライトサンプリングでの確率は
  - [pdf of sampling the light vertex] x [SLC (Russian Roulette) probability]

BOUNDING SPHERE TREEを用いたカリング

- フラスタムカリングを使うことができない
  - シェーディング点がスクリーンスペースに並んでいない
    - カメラから直接見える点は例外
  - ランダムに分布したシェーディング点
  - => 一般的なカリング方法を用いる必要がある
  - Bounding sphere treeを用いた
    - BVHの一種

BOUNDING SPHERE TREEを用いたカリング

実装

- それぞれのステップ(フレーム)において影響範囲が変わる
- それぞれのステップ(フレーム)において完全に再構築することもできる
- 確率的ライトカリングのオーバーヘッドを減らすために、木構造は一度だけ計算されて、その後はリフィットに止める

*n*th step

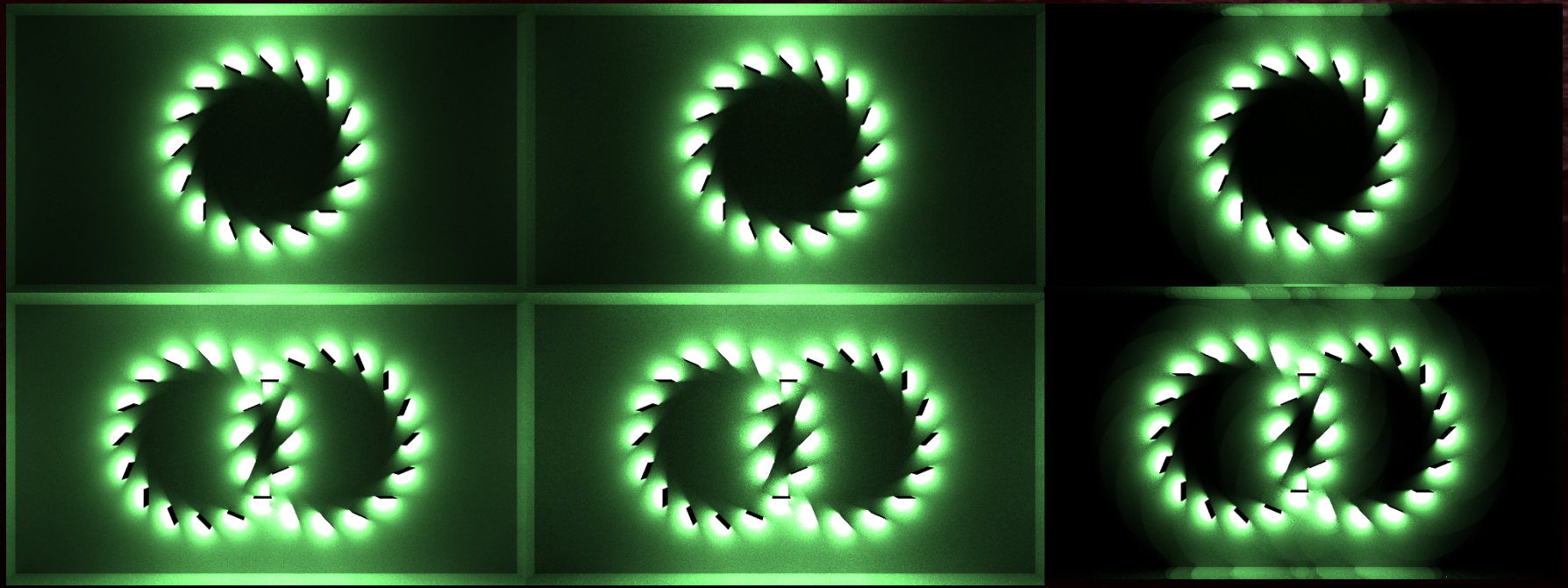
*n+1*th step

RESULTS
収束の比較

Reference

Stochastic Light Culling

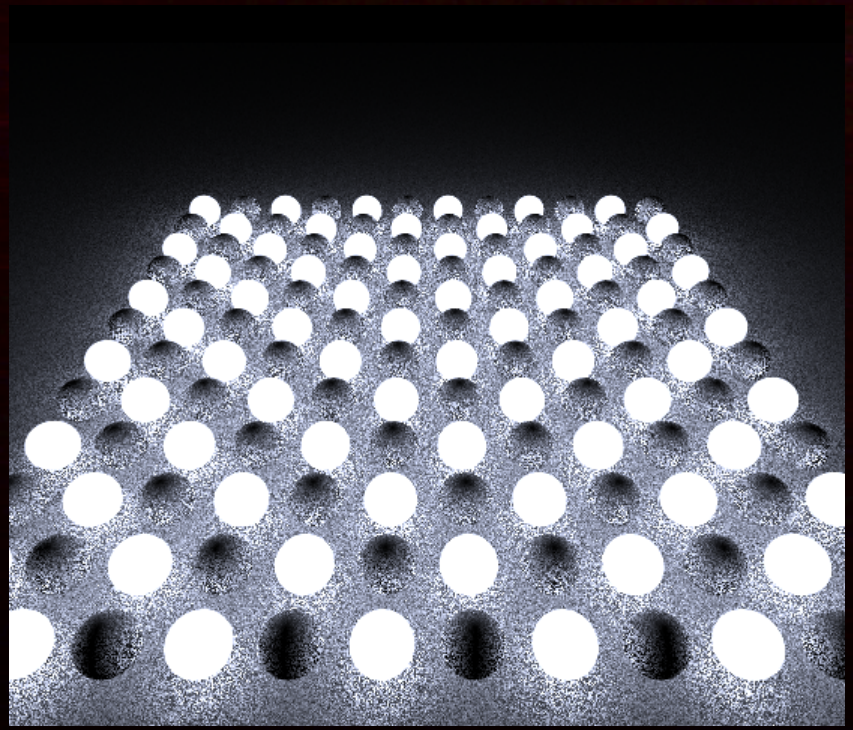
Clamping



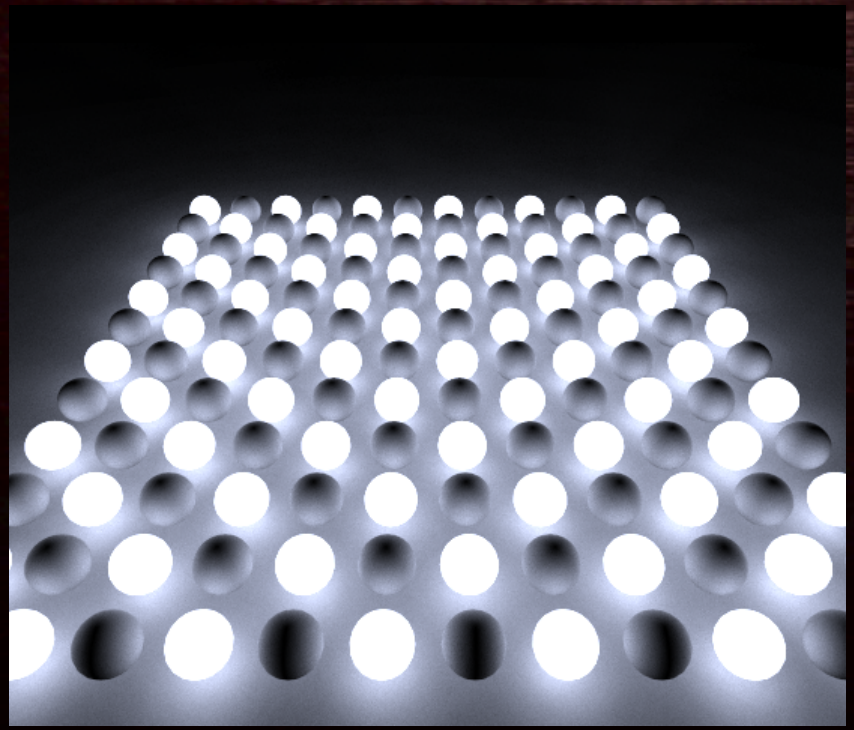
RESULTS

収束の速さ比較(CPU実装、同じ計算時間での比較)

Uniform sampling ☹☹



Stochastic Light Culling ☺☺



55,000 triangle lights, after 30s

CEDEC2016
Computer Entertainment Developers Conference

STOCHASTIC LIGHT CULLING ON GPU

CHALLENGES

- 確率的ライトカリングはCPU実装ではとても良い
- GPUで実装するときに少し注意が必要

1. Work item (thread) の発散
   - シェーディングとビジビリティのテストをそれぞれのリーフノードにおいて行うとコードの発散がひどい

2. Memory使用量
   - 多くのWIが並列に走っている
   - それぞれのWIで必要なメモリ量が少なくても、多くのWIだと問題になる
     - [# of lights] x [# of WIs]

IMPROVING TREE TRAVERSAL & SHADING

WORK ITEM DIVERGENCE

- BVH巡回
- リーフノードでシェーディング

```
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

WORK ITEM DIVERGENCE

- ツリー巡回しているときに、シェーディングを行わないためには、ライトの番号をバッファに保持し、後でシェーディングをする

```
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

- ツリー巡回しているときに、シェーディングを行わないためには、ライトの番号をバッファに保持し、後でシェーディングをする
- ツリー巡回中の発散は解決したが
  - シェーダー実行の時間がばらつく (ray cast + shade)
  - ライトの番号を蓄えるバッファが大きすぎる (# of lights x # of WIs)

```
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

- ツリー巡回中の発散は解決したが
  - ライトの番号を蓄えるバッファが大きすぎる
    - => 最大k個のライトだけを蓄える
    - => 最大k回のシェーディングを行う
- リザーバサンプリング
  - 全ての候補を蓄えずに最大k個を選択することができる

```
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 ☺

BOUNDING SPHERE TREE REFIT

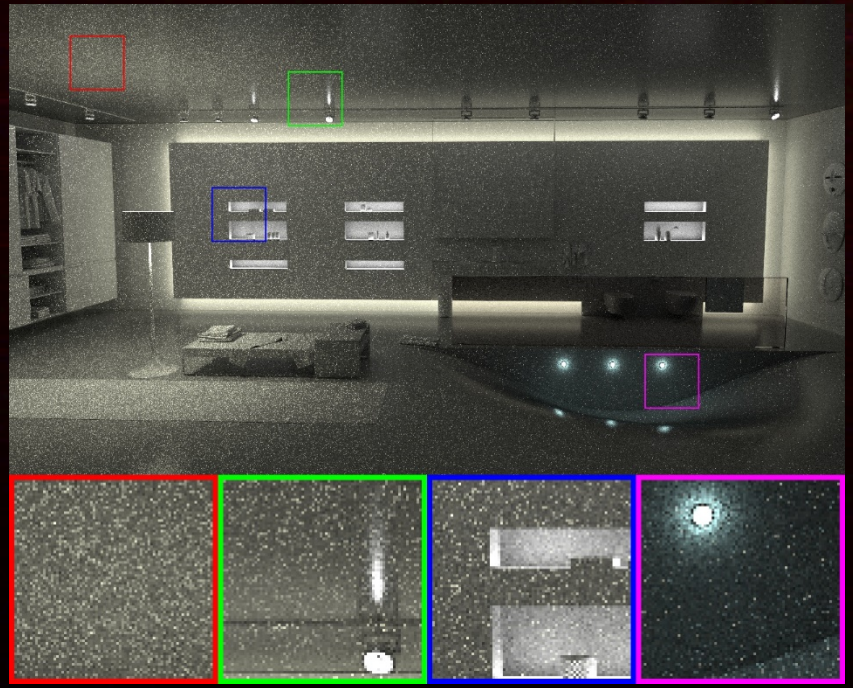
- 前処理としてツリーはCPUで構築して、GPUに転送

- 複数のカーネルを実行してリフィット
- それぞれのカーネルで1レベル、リフィット
- for それぞれのツリーの深さ:
  –リフィットのカーネルを実行

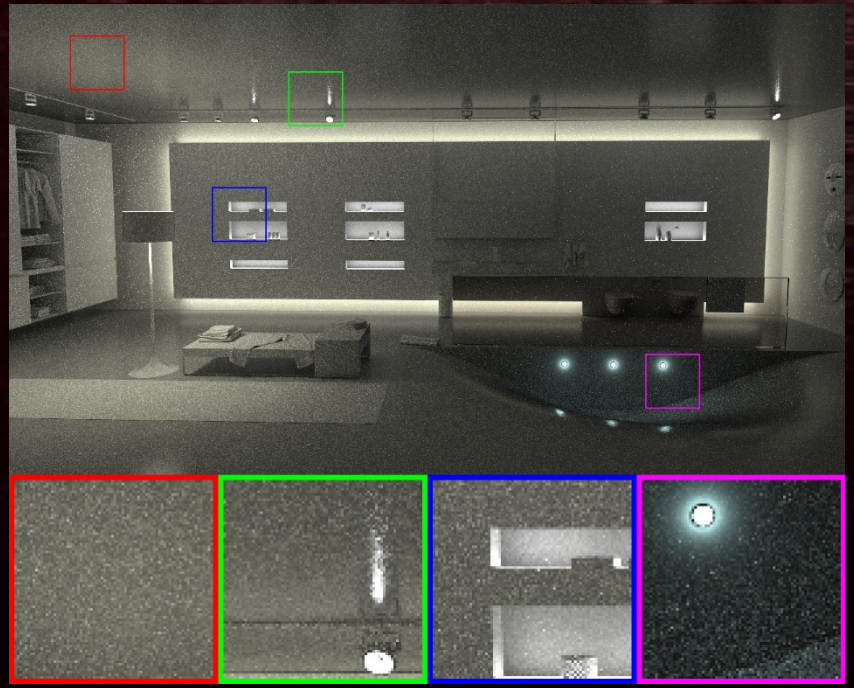
```
_kernel
void RefitOneLevelKernel( GLOBAL(BSTreeNode) gTree, GLOBAL(u32)
gLevels, int cb_n, int cb_level )
{
    int gIdx = GET_GLOBAL_IDX;

    if( gIdx >= cb_n ) return;

    if( gLevels[gIdx] != cb_level )
        return;

    int nodeIdx = gIdx;

    BSTreeNode* dst = &gTree[nodeIdx];

    u32 c0 = dst->m_child0;
    u32 c1 = dst->m_child1;

    float4 v0 = gTree[ c0 ].m_volume;
    float4 v1 = gTree[ c1 ].m_volume;

    mergeChildVolumes( dst );
}
```

RESULTS

Uniform Sampling (RSME:0.0749)



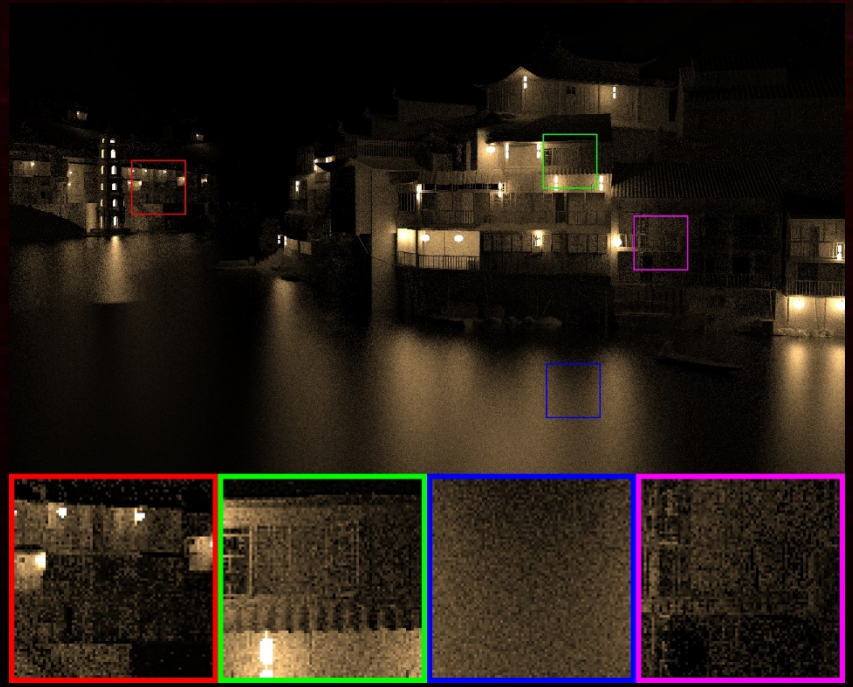
Stochastic Light Culling (RSME:0.0464)



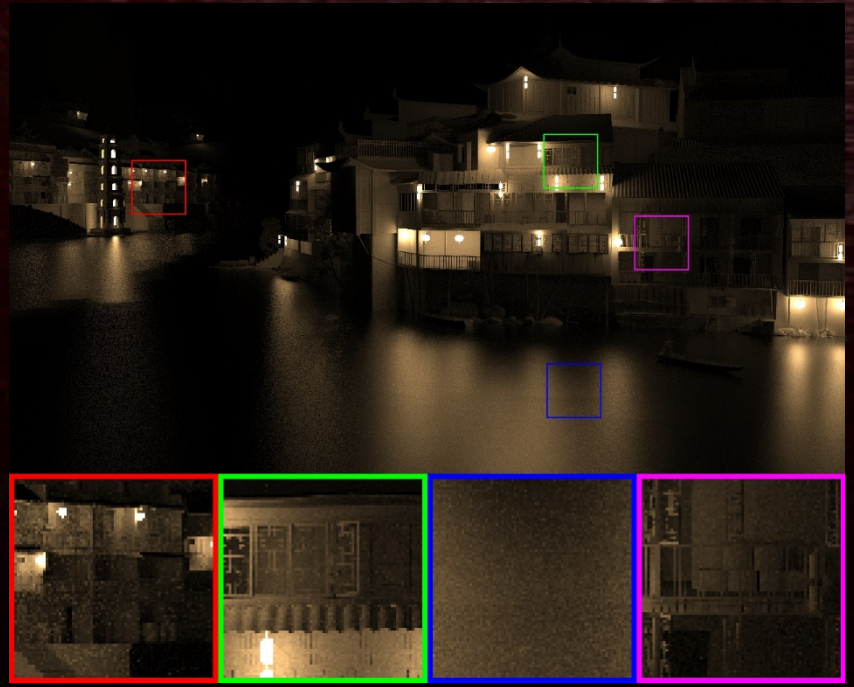
5,000 triangle lights, after 2min

RESULTS

Uniform Sampling (RSME:0.0355)



Stochastic Light Culling (RSME:0.0203)



59,000 triangle lights, after 2min

DEMO TIME!!

- Do we have time??

CLOSING

- 確率的ライトカリングを紹介した
  - バイアスを加えないライトカリング

- 二つの応用例を紹介した
  - VPLを用いたリアルタイムGI
    - VPL + interleaved sampling
  - パストレーシングを用いたインタラクティブGI
    - エリアライトへの拡張
    - GPUへの最適化

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.
- 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.
- RITSCHEL T., EISEMANN E., HA I., KIM J. D., SEIDEL H.-P. 2011. Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. Comput. Graph. Forum 30, 8, 2258–2269.
- SADEGHI, I., CHEN, BIN., AND JENSEN, H. W. 2009. Coherent path tracing. Journal of Graphics, GPU, and Game Tools, 14, 2.
- 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.
- T. Hachisuka. 2013. Five common misconceptions about bias in light transport simulation.
- 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.
- WALD I., KOLLIG T., BENTHIN C., KELLER A., SLUSALLEK P. 2002. Interactive global illumination using fast ray tracing. In EGWR’02, pp. 15–24. 2
- WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: A scalable approach to illumination. ACM Trans. Graph. 24, 3, 1098–1107.