GPGPUによる高速なグローバルイルミネーションベイクツールの作り方

株式会社スクウェア・エニックス

関根 崇

アジェンダ

第1章　静的GIの基礎

第2章　Ray Bundle Tracing 解説

第3章　実践編

第1章 静的GIの基礎

静的GI

- 屋外 / 太陽の光

ディフューズ

静的GI

- 屋外 / 太陽の光

ライトマップ

静的GI

- 屋外 / 環境光

ディフューズ

静的GI

- 屋外 / 環境光

ライトマップ

静的GI

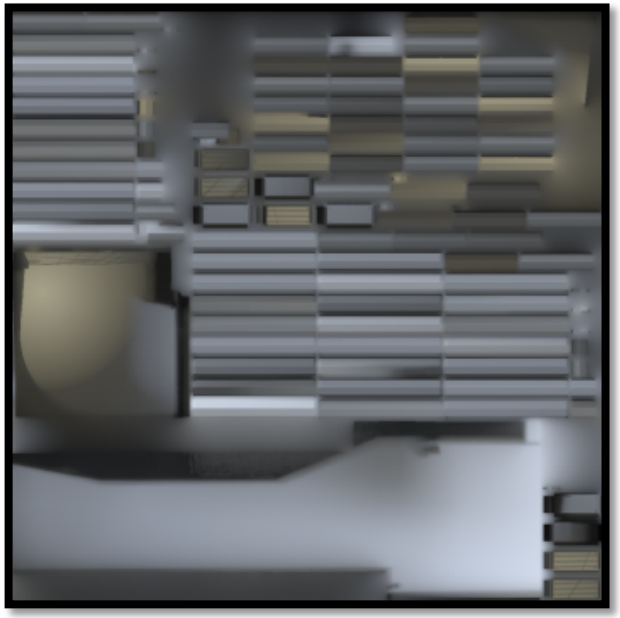
- 屋内 / 間接光

静的GI

- GIはランタイムで処理するには重い……
  - コンソール機のリソースでは特に難しい
  - 事前に計算しておこう！
    - ライトマップ
    - Irradiance Volume

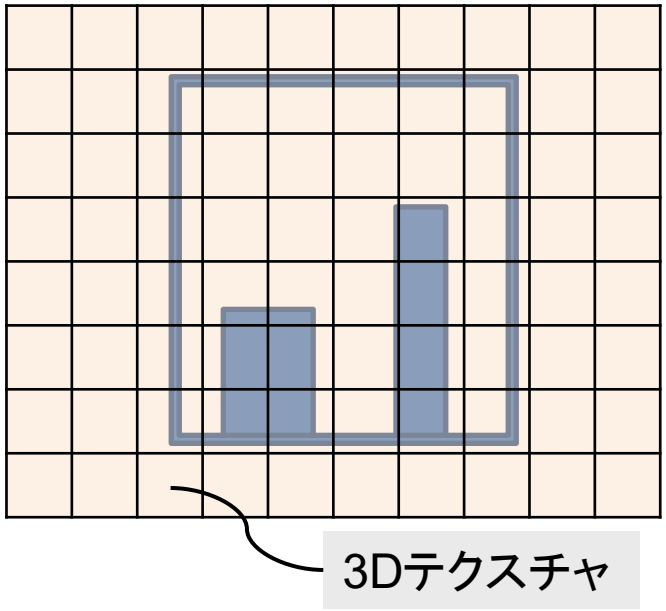
ライトマップ

- 「どんな光が当たっているか」のテクスチャ
- ランバート反射 (拡散反射)



Irradiance Volume

- キャラクターへのGI (環境光)
  - ライトマップの3次元バージョン
  - ライトマップと同様に作成できる



静的GIの良い所

- ランタイムでのパフォーマンスが良い
- クオリティが高い

パフォーマンス

- ランタイムではデータを読み出すだけ
  - 処理時間が変動しない
- ライトの個数に関係ない
  - いくつ置いても同じパフォーマンス

クオリティ

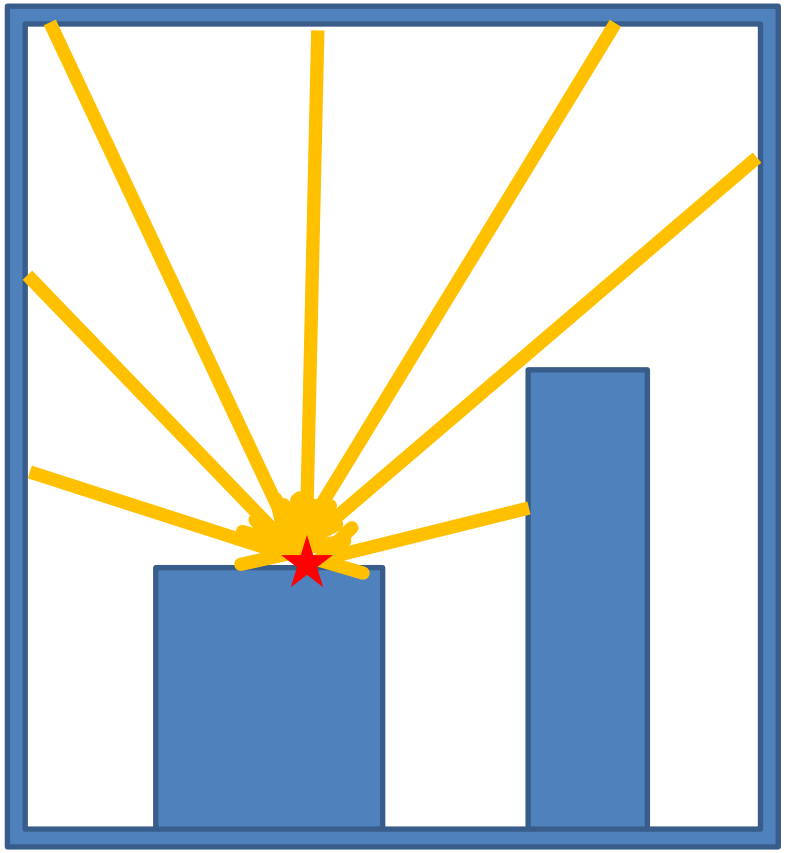
- ランタイムと無関係に任意のGI手法を使える
- 複数回の相互反射

静的GIの困る所

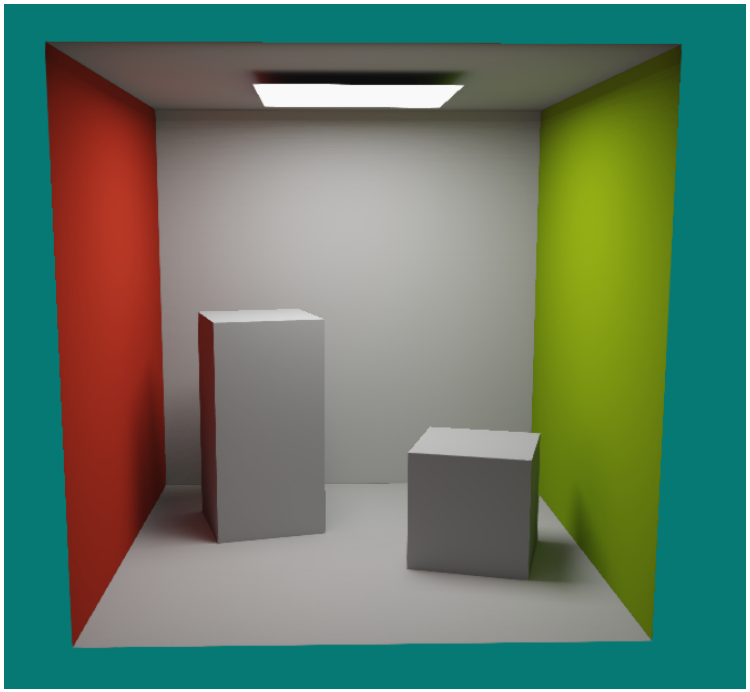
- メモリ使用量
  - モデル毎に共有不可能なテクスチャ
- シーンの変化に弱い
  - ライトの色,位置
  - 地形の位置
- 事前計算(ベイク)に時間がかかる！

ライトマップの作り方

- テクセルに入ってくる光を集める

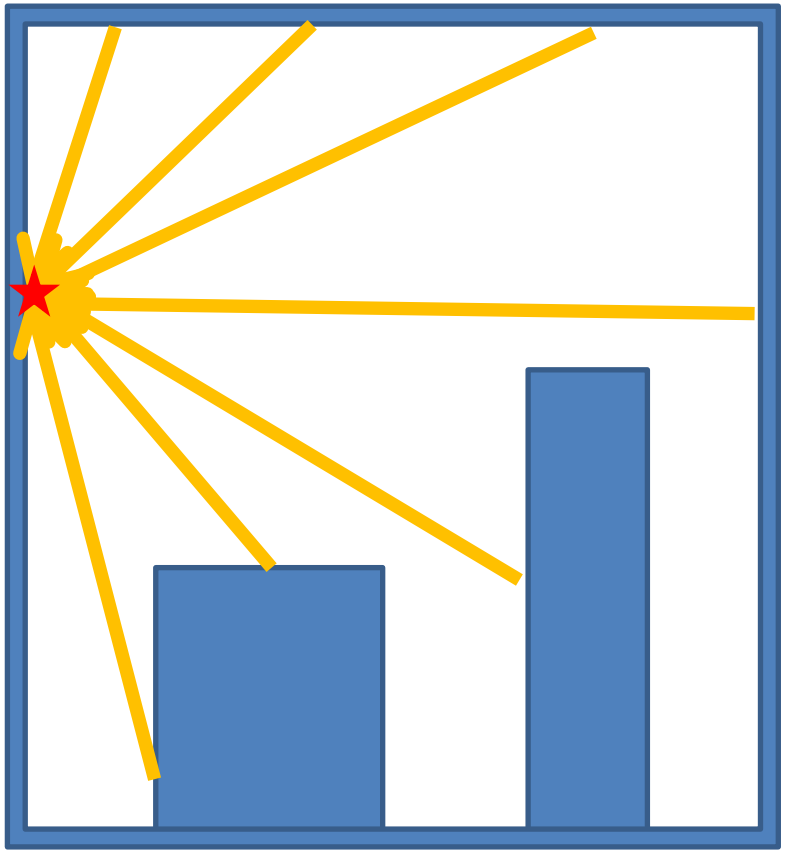


全てのモデルに
ライトマップがある

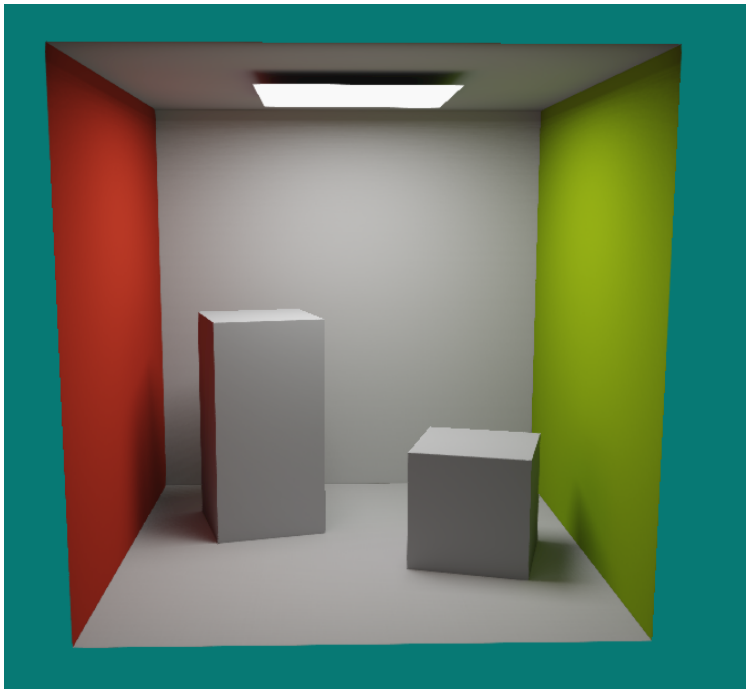


ライトマップの作り方

- テクセルに入ってくる光を集める

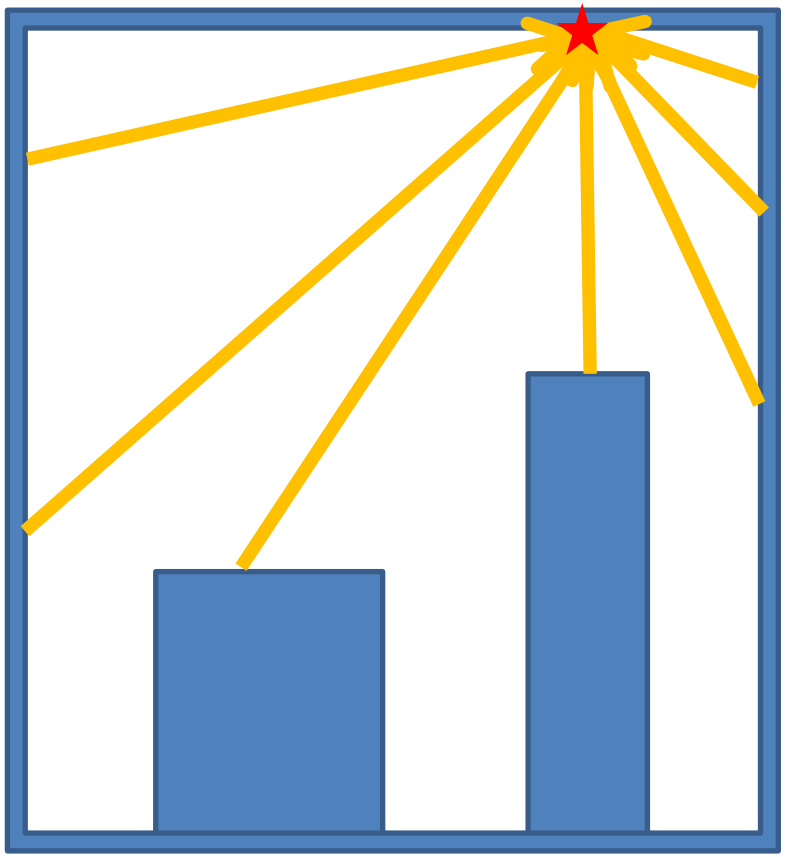


全てのモデルに
ライトマップがある

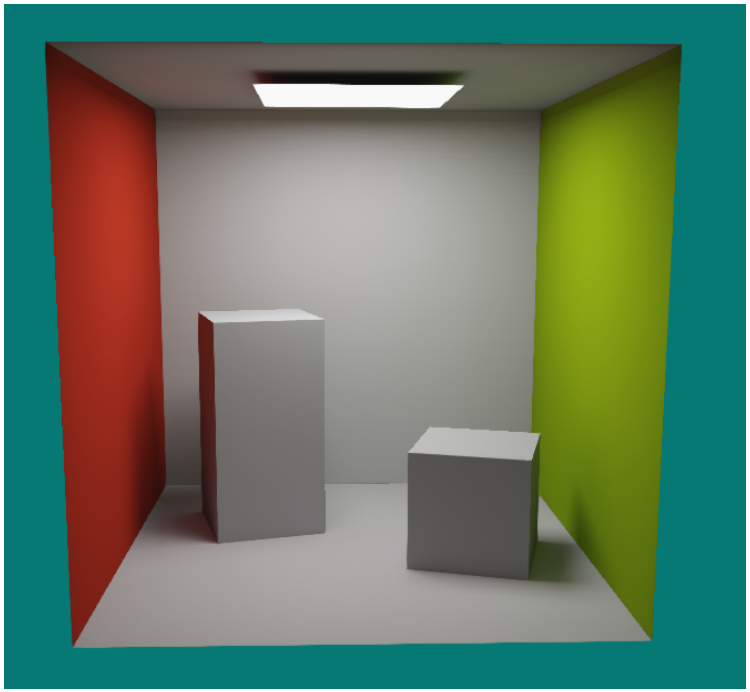


ライトマップの作り方

- テクセルに入ってくる光を集める



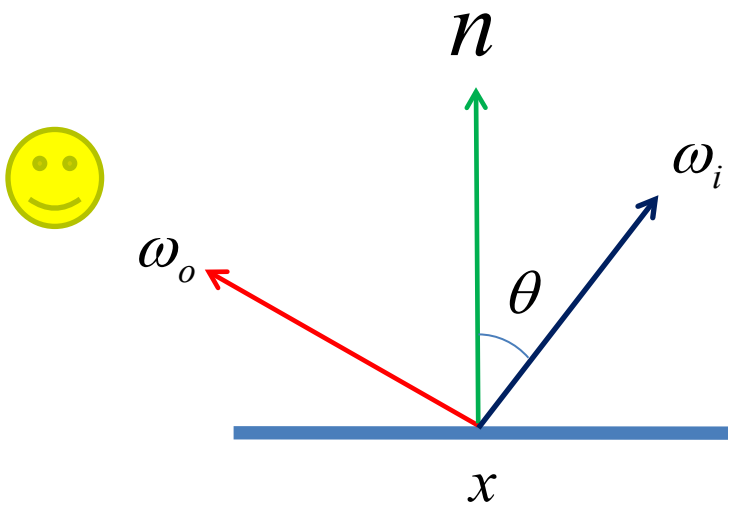
全てのモデルに
ライトマップがある



レンダリング方程式

$$L_o(x,\omega_o) = \int_\Omega f_r(x,\omega_i,\omega_o) L_i(x,\omega_i) \cos\theta d\omega_i$$

出ていく光　　半球　　BRDF　　入ってくる光



レンダリング方程式

- ランバート反射

$$f_r(x,\omega_i,\omega_o)=\frac{\rho(x)}{\pi}$$

反射率
(ディフューズ
テクスチャ)

$$L_o(x,\omega_o)=\int_\Omega f_r(x,\omega_i,\omega_o)L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}\int_\Omega L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}E(x)$$

E(x)をライトマップに格納

ディフューズ x ライトマップでOK！

レンダリング方程式

- ランバート反射

$$f_r(x,\omega_i,\omega_o)=\frac{\rho(x)}{\pi}$$

反射率
(ディフューズ
テクスチャ)

$$L_o(x,\omega_o)=\int_\Omega f_r(x,\omega_i,\omega_o)L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}\int_\Omega L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}E(x)$$

E(x)をライトマップに格納

ディフューズ x ライトマップでOK！

レンダリング方程式

- ランバート反射

$$f_r(x,\omega_i,\omega_o)=\frac{\rho(x)}{\pi}$$

反射率
(ディフューズ
テクスチャ)

$$L_o(x,\omega_o)=\int_\Omega f_r(x,\omega_i,\omega_o)L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}\int_\Omega L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}E(x)$$

E(x)をライトマップに格納

ディフューズ x ライトマップでOK！

レンダリング方程式

- ランバート反射

$$f_r(x,\omega_i,\omega_o)=\frac{\rho(x)}{\pi}$$

反射率
(ディフューズ
テクスチャ)

$$L_o(x,\omega_o)=\int_\Omega f_r(x,\omega_i,\omega_o)L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}\int_\Omega L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}E(x)$$

E(x)をライトマップに格納

ディフューズ x ライトマップでOK！

レンダリング方程式

- ランバート反射

$$f_r(x,\omega_i,\omega_o)=\frac{\rho(x)}{\pi}$$

反射率
(ディフューズ
テクスチャ)

$$L_o(x,\omega_o)=\int_\Omega f_r(x,\omega_i,\omega_o)L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}\int_\Omega L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}E(x)$$

E(x)をライトマップに格納

ディフューズ x ライトマップでOK！

レンダリング方程式

- ランバート反射

$$f_r(x,\omega_i,\omega_o)=\frac{\rho(x)}{\pi}$$

反射率
(ディフューズ
テクスチャ)

$$L_o(x,\omega_o)=\int_\Omega f_r(x,\omega_i,\omega_o)L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}\int_\Omega L_i(x,\omega_i)\cos\theta d\omega_i$$

$$=\frac{\rho(x)}{\pi}E(x)$$

E(x)をライトマップに格納

ディフューズ x ライトマップでOK！

モンテカルロ法で計算

- ライトマップの値

$$E(x) = \int_{\Omega} L_i(x, \omega_i) \cos\theta d\omega_i$$

モンテカルロ法

$$E(x) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{L_i(x, \omega_i) \cos\theta}{p(\omega_i)}$$

一様サンプルなら

$$p(\omega_i) = \frac{1}{2\pi}$$

ω_i ：半球上のランダム方向

確率密度関数

モンテカルロ法で計算

- 一様サンプルの場合

$$E(x) = \int_{\Omega} L_i(x, \omega_i) \cos\theta d\omega_i$$

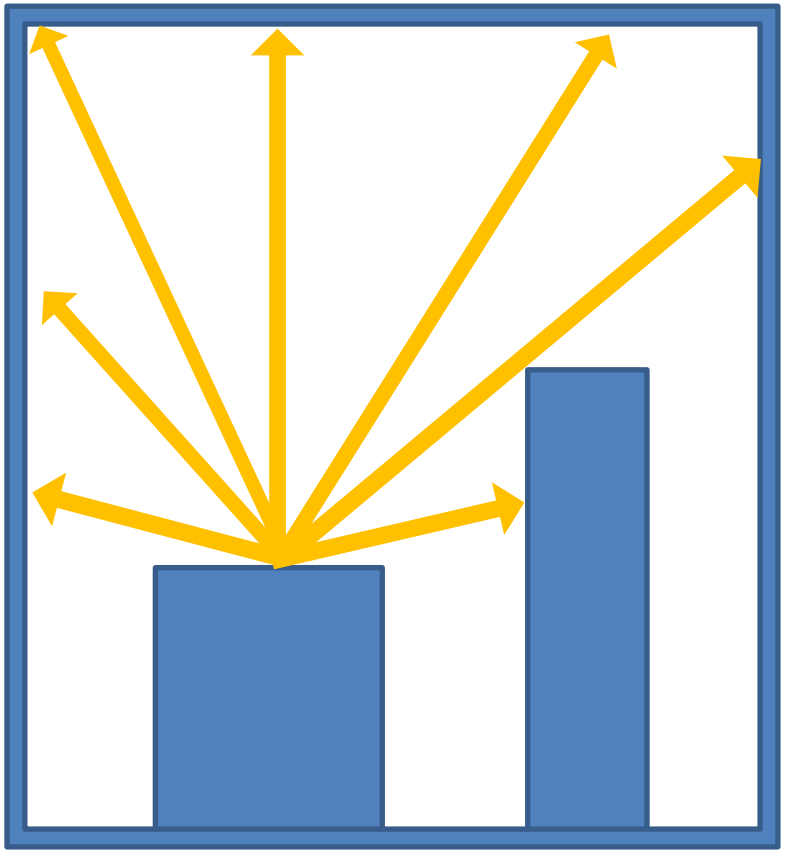
モンテカルロ法

$$E(x) \approx \frac{2\pi}{N} \sum_{i=1}^{N} L_i(x, \omega_i) \cos\theta$$

半球方向からの光を足すとライトマップに！

ライトマップの作り方

- テクセル毎に計算



```
for each ライトマップテクセル {
    for each サンプル方向 {
        レイ衝突判定;
        衝突位置の輝度を足す;
    }
}
```

$$E(x) \approx \frac{2\pi}{N} \sum_{i=1}^{N} L_i(x, \omega_i) \cos\theta$$

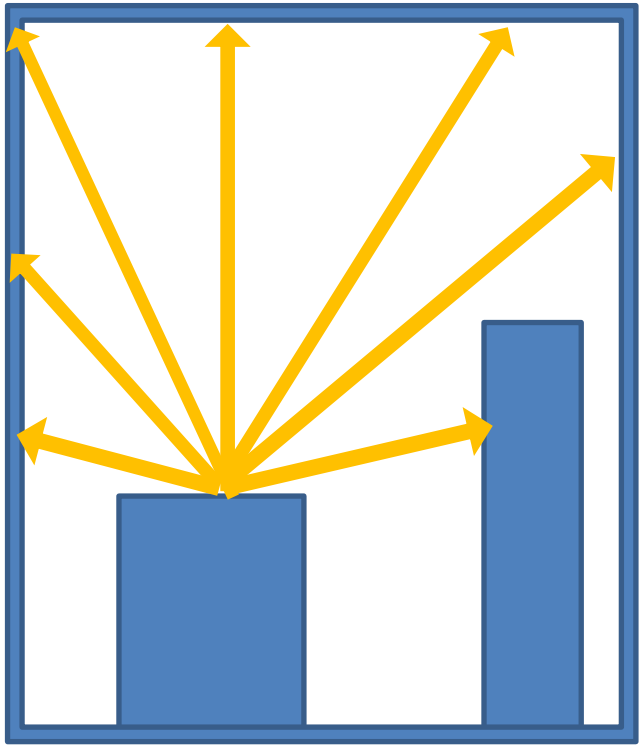
重い！

第2章 RAY BUNDLE TRACING

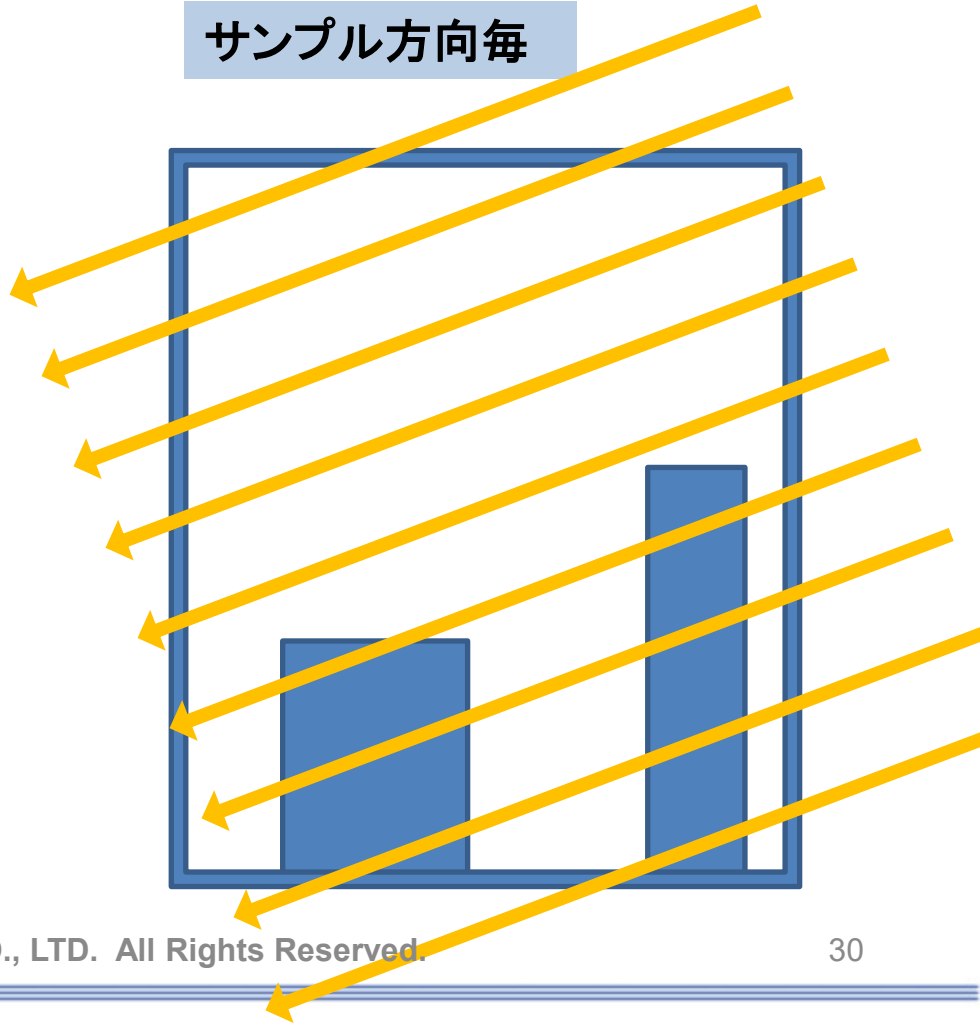
Ray Bundle Tracing

- 発想の転換

テクセル毎

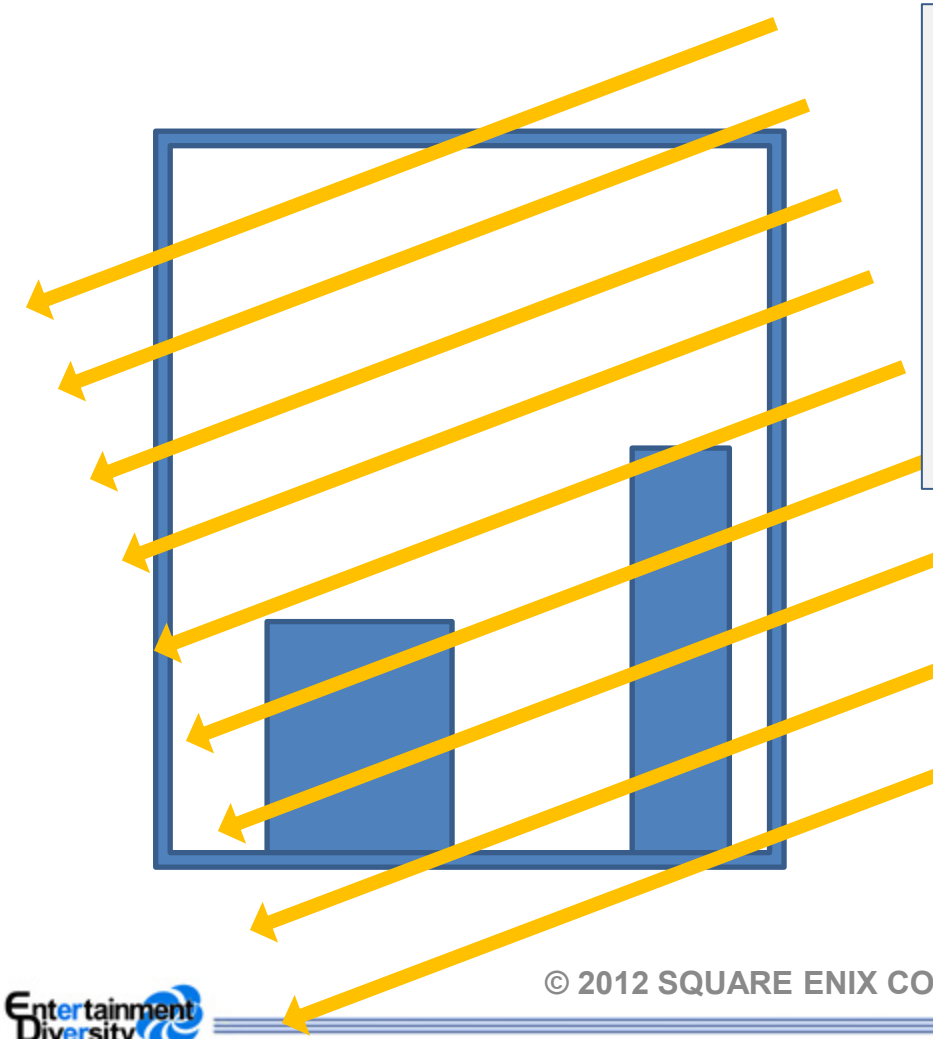


サンプル方向毎



Ray Bundle Tracing (イメージ)

- サンプル方向毎に計算

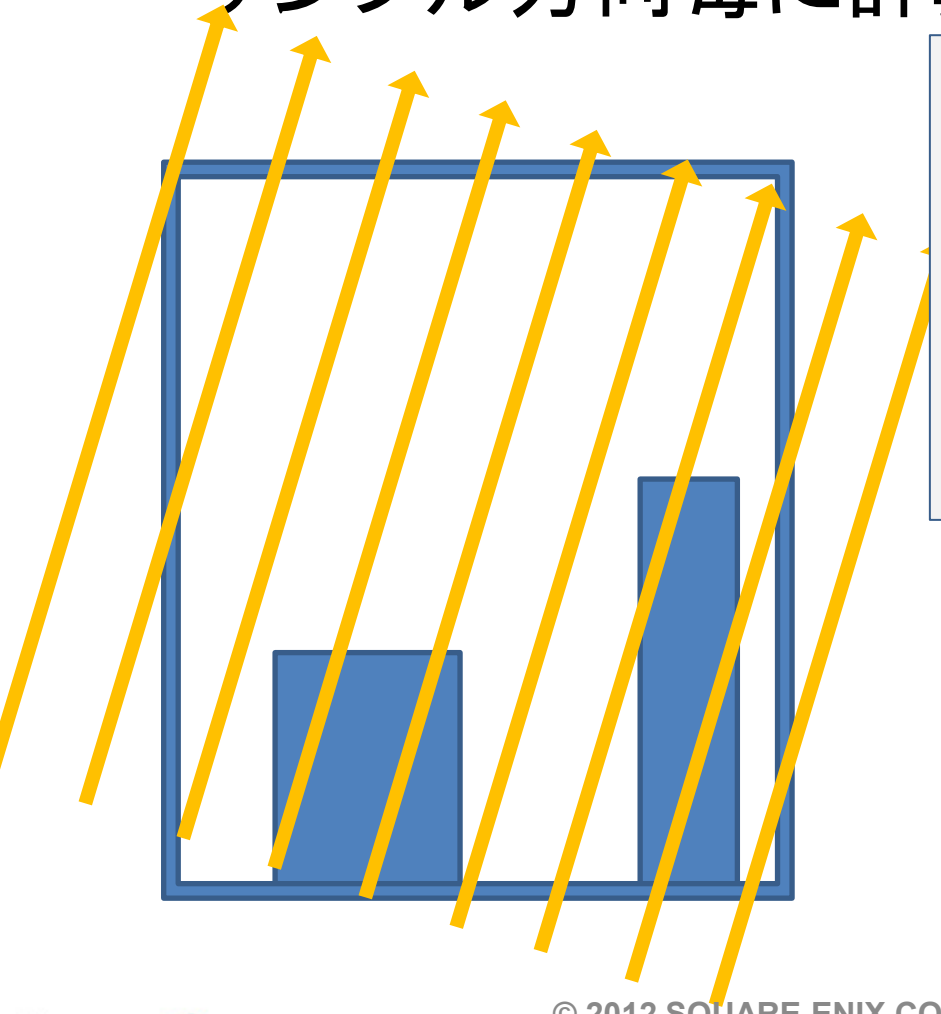


```
for each サンプル方向 {
    レイ衝突判定;
    for each ライトマップテクセル {
        衝突位置の輝度を足す;
    }
}
```

$$E(x) \approx \frac{2\pi}{N} \sum_{i=1}^{N} L_i(x, \omega_i) \cos\theta$$

Ray Bundle Tracing (イメージ)

- サンプル方向毎に計算

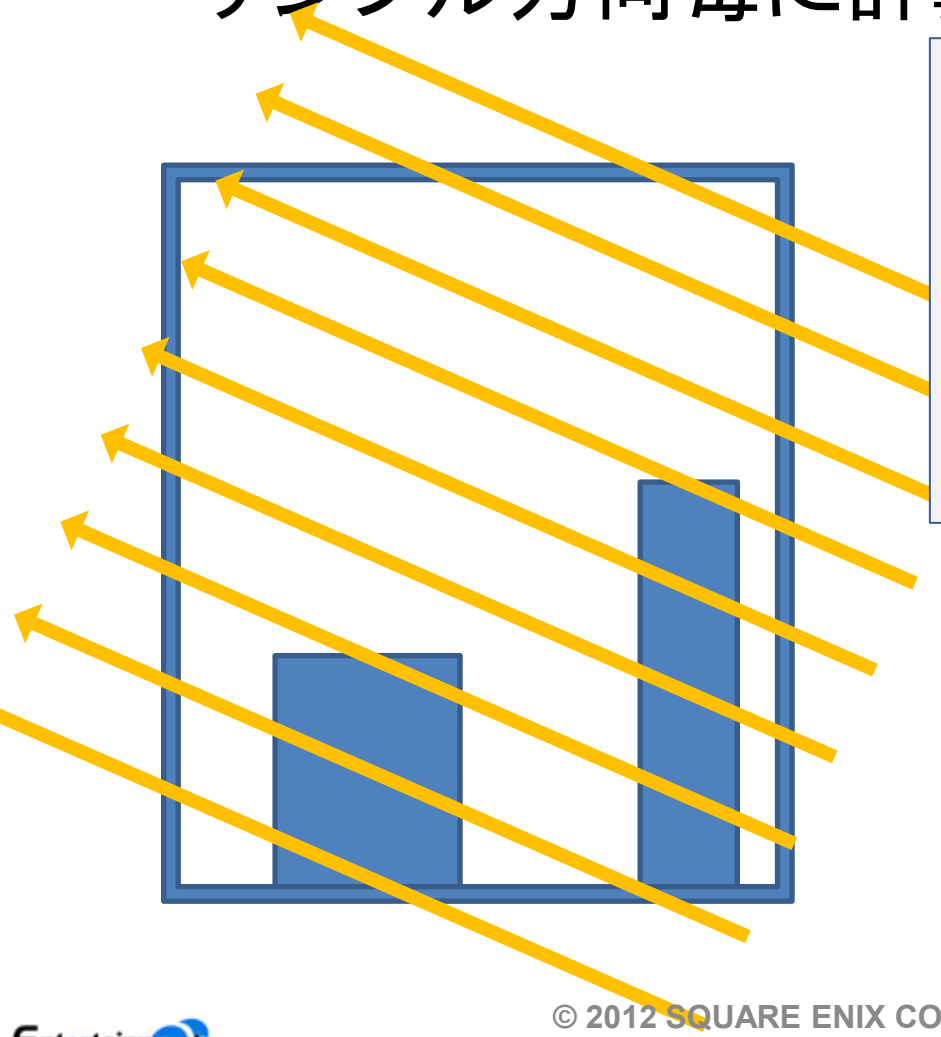


```
for each サンプル方向 {
    レイ衝突判定;
    for each ライトマップテクセル {
        衝突位置の輝度を足す;
    }
}
```

$$E(x) \approx \frac{2\pi}{N} \sum_{i=1}^{N} L_i(x, \omega_i) \cos\theta$$

Ray Bundle Tracing (イメージ)

- サンプル方向毎に計算

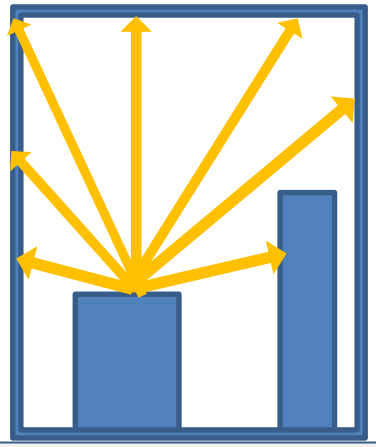


```
for each サンプル方向 {
    レイ衝突判定;
    for each ライトマップテクセル {
        衝突位置の輝度を足す;
    }
}
```

$$E(x) \approx \frac{2\pi}{N} \sum_{i=1}^{N} L_i(x, \omega_i) \cos\theta$$

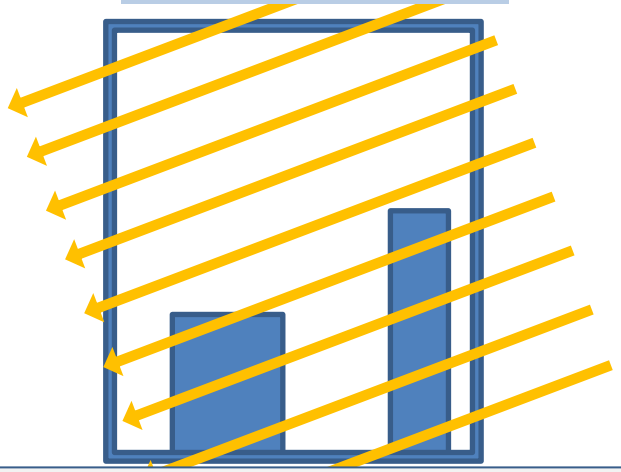
Ray Bundle Tracing

テクセル毎



```
for each ライトマップテクセル {
    for each サンプル方向 {
        レイ衝突判定;
        衝突位置の輝度を足す;
    }
}
```

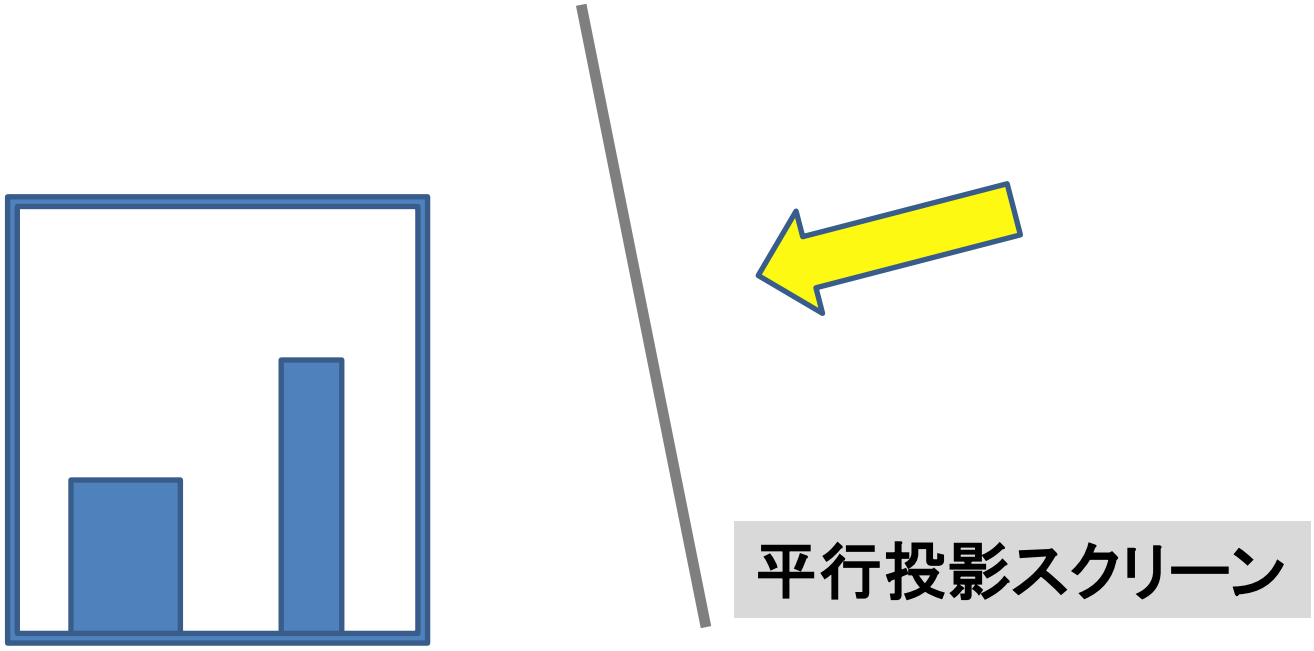
サンプル方向毎



```
for each サンプル方向 {
    レイ衝突判定;
    for each ライトマップテクセル {
        衝突位置の輝度を足す;
    }
}
```

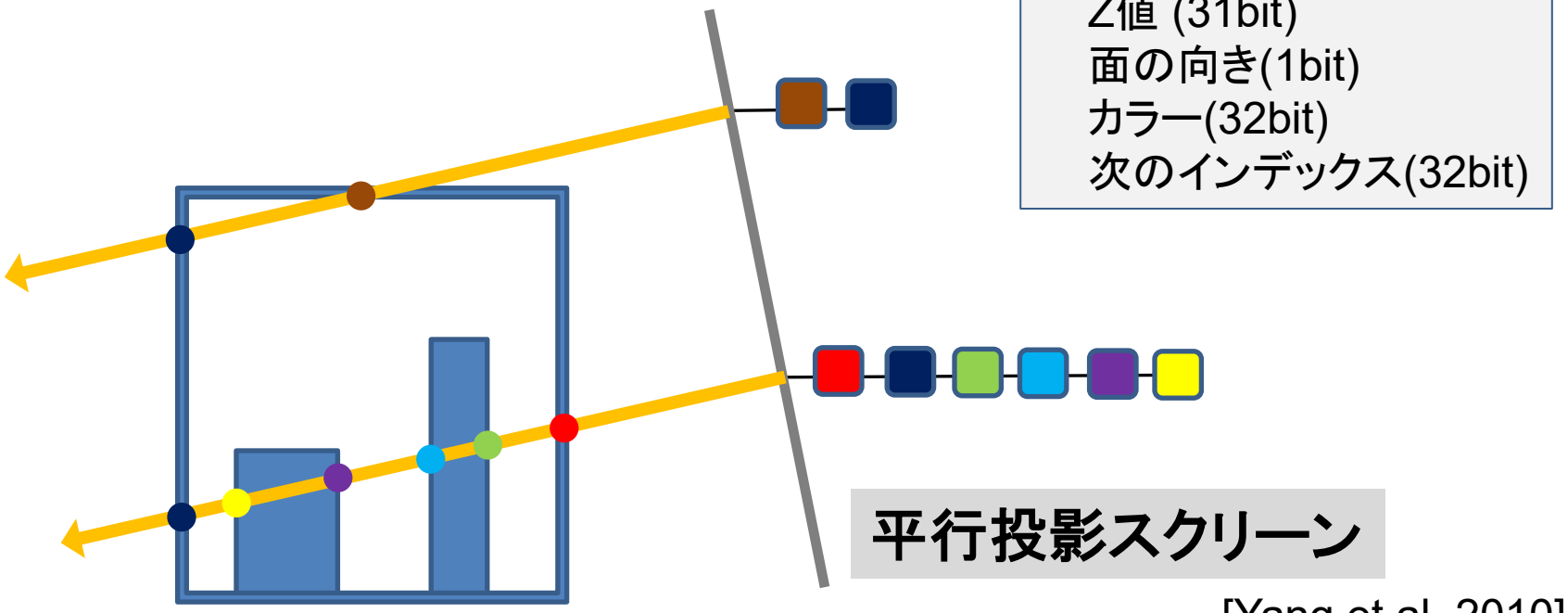
衝突判定 (ピクセルシェーダー)

- ラスタライズで衝突判定
  - シーン全体を平行投影ラスタライズ



衝突判定 (ピクセルシェーダー)

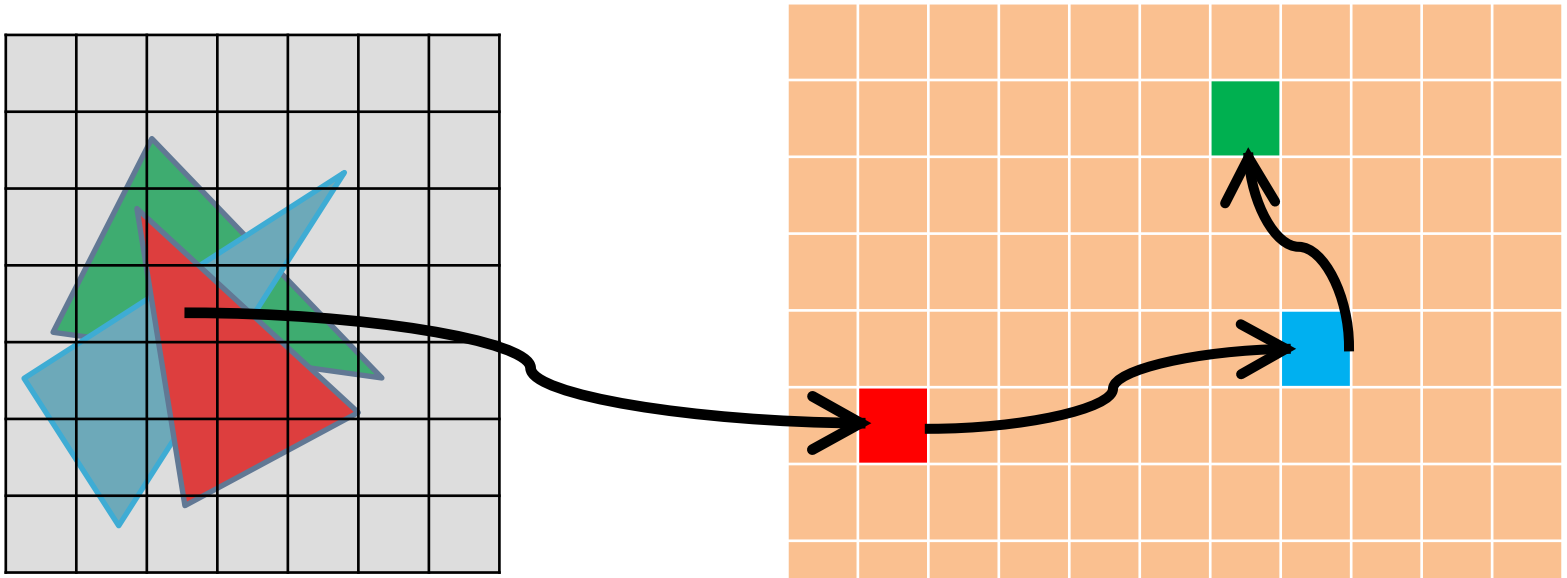
- Z値、裏表でのカリングをしない
  - ピクセル毎に LinkedList を作る



[Yang et al. 2010]

衝突判定 (ピクセルシェーダー)

- UAV (Unordered Access View)



Ray Bundle ヘッダー
(平行投影スクリーン)

LinkedList の先頭インデックス

Fragment バッファ (UAV)

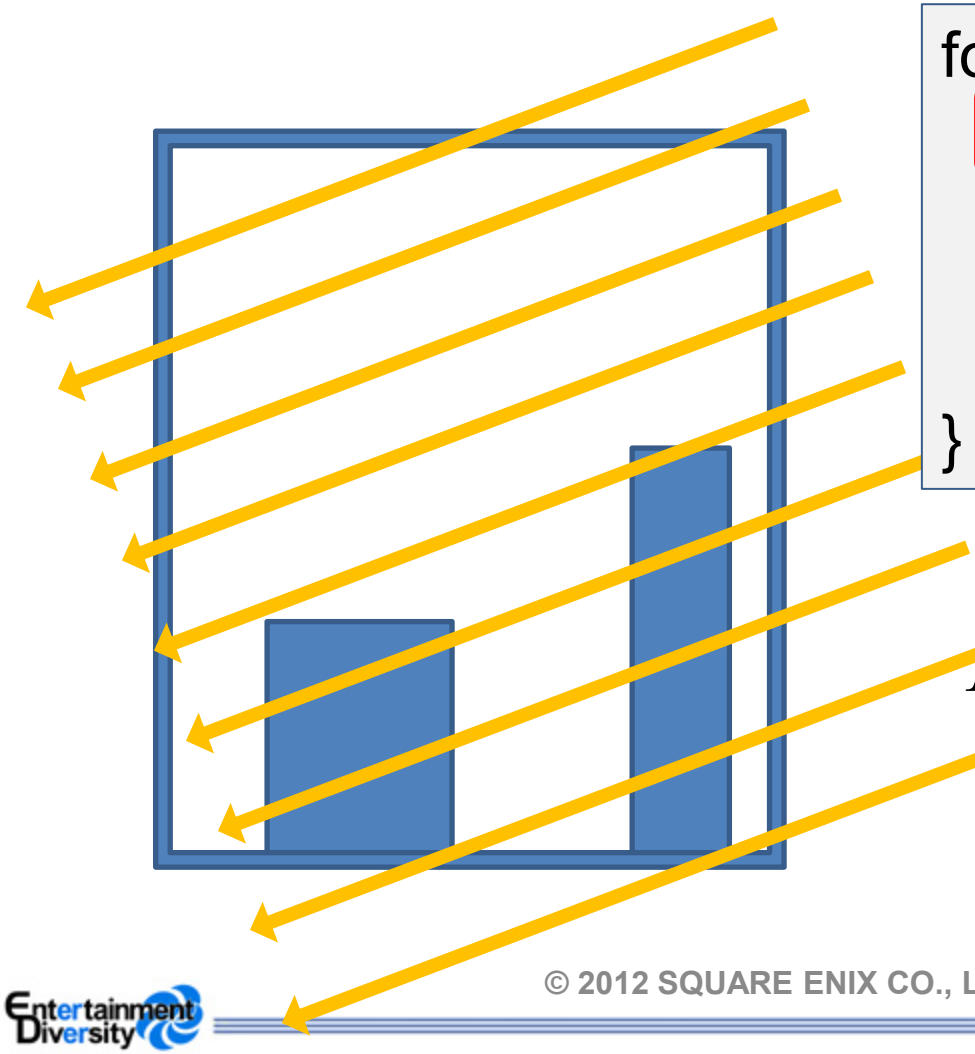
LinkedList のノード実体

衝突判定 (ピクセルシェーダー)

```
RWStructuredBuffer<FragmentLink> g_fragment_buffer;
RWTexture2D<uint> g_ray_bundle_header;

void CreateRayBundlePS(const PS_INPUT input)
{
    // 輝度計算
    const float3 color = ....;

    // Fragmentバッファからメモリ確保 / リンク差し替え
    const uint new_index = g_fragment_buffer.IncrementCounter();
    InterlockedExchange(g_ray_bundle_header[input.pos.xy], new_index, old_index);

    // 新しいノード
    FragmentLink node;
    node.data.depth = (input.is_front_face * 2.0f - 1.0f) * input.pos.z;
    node.data.color = color;
    node.next_index = old_index;

    // LinkedList の先頭に新しいノードを挿入
    g_fragment_buffer[new_index] = node;
}
```

Ray Bundle Tracing (イメージ)

- サンプル方向毎に計算

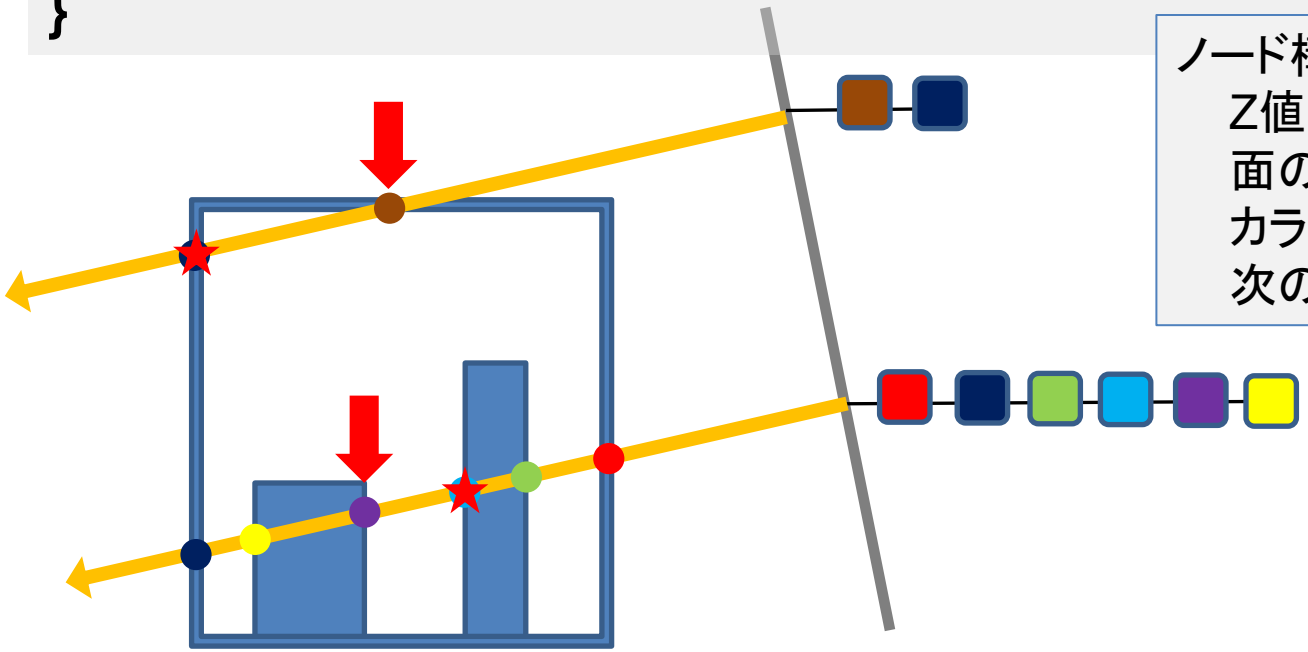


```
for each サンプル方向 {
    レイ衝突判定;
    for each ライトマップテクセル {
        衝突位置の輝度を足す;
    }
}
```

$$E(x) \approx \frac{2\pi}{N} \sum_{i=1}^{N} L_i(x, \omega_i) \cos\theta$$

輝度を足す(ComputeShader)

```
for each ライトマップテクセル {
    テクセルのワールド位置をスクリーンに投影;
    対応する LinkedList 内を全探索;
    Z値を見てレイのヒット位置のフラグメントを見つける;
    輝度を足す;
}
```



ノード構造体
- Z値 (31bit)
- 面の向き(1bit)
- カラー(32bit)
- 次のインデックス(32bit)

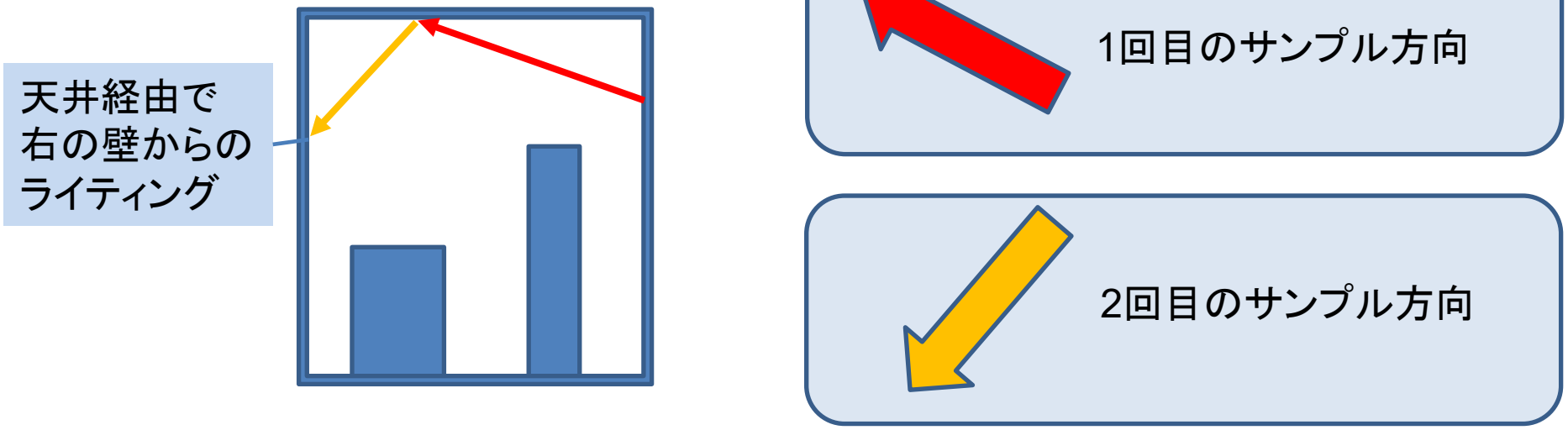
Ray Bundle Tracing まとめ

```
for each サンプル方向 {
    シーン全体をラスタライズ;
    for each ライトマップテクセル {
        LinkedList探索;
        輝度を足す;
    }
}
```

$$E(x) \approx \frac{2\pi}{N} \sum_{i=1}^{N} L_i(x, \omega_i) \cos\theta$$

複数回の相互反射

- ライトマップは計算途中のキャッシュ
- 輝度計算に再利用
- 複数回の相互反射を計算可能

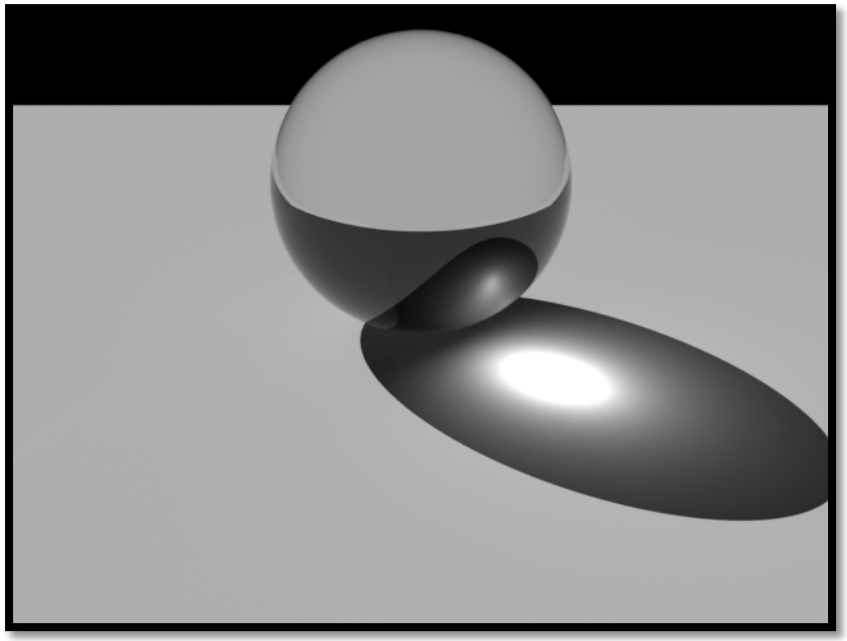


制限

- 光沢反射
- 鏡面反射
- 誤差
- メモリ

光沢反射 / 鏡面反射

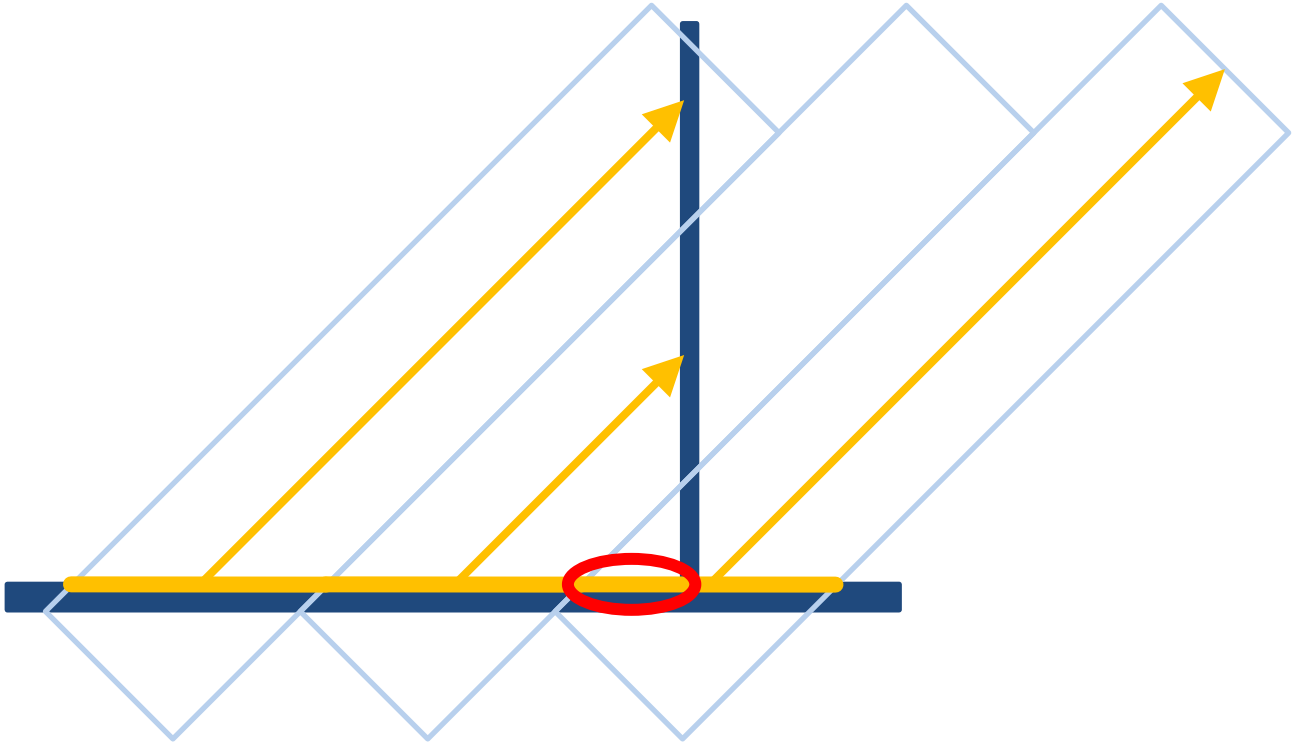
- 基本はランバート反射で
- ランバート以外は苦手
- 鏡を扱うことは不可能



コースティクス

誤差

- ライトリーク



メモリ

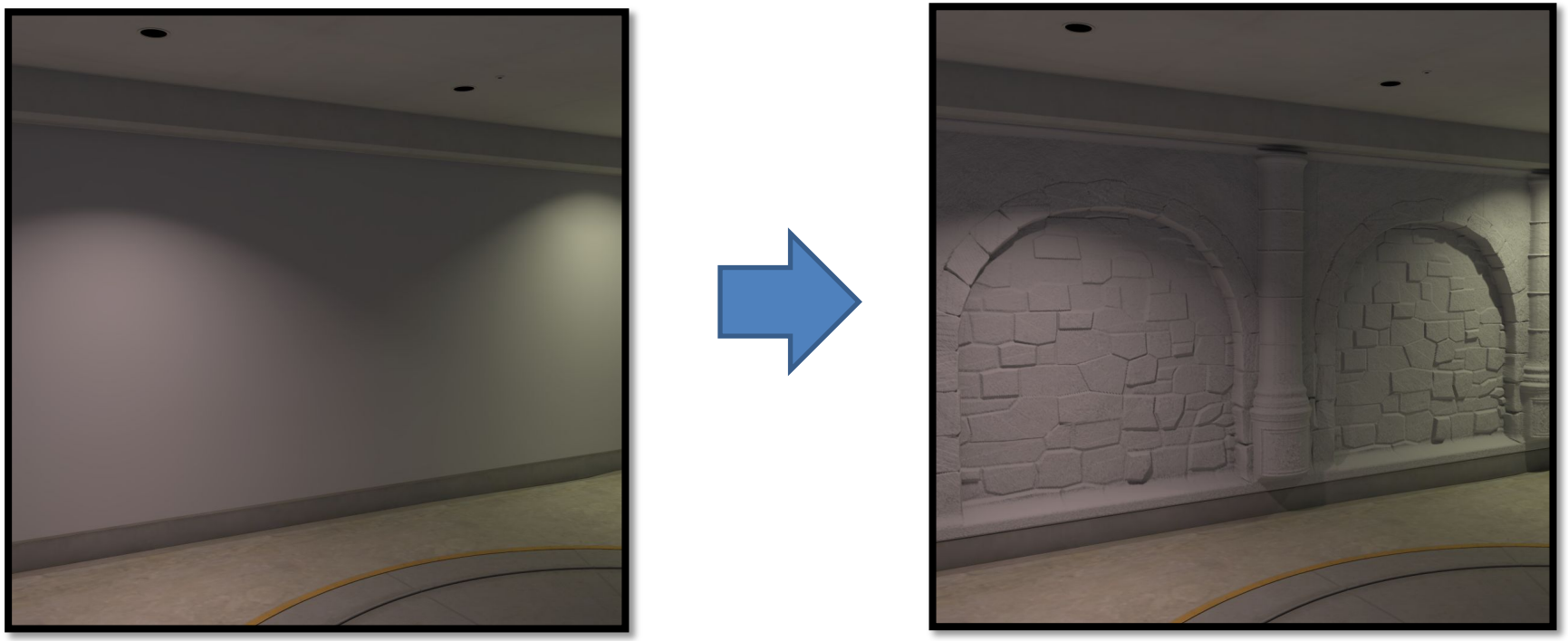
- 計算用のバッファたくさん
- 全部VRAM
- LinkedListノード (Fragmentバッファ)
  - 充分なサイズを確保しておく必要
- 広いシーンが難しい

第3章 実践編

Ray Bundle Tracing 実践

- テッセレーション
- ランバート以外の相互反射
- 広大なシーン

テッセレーション



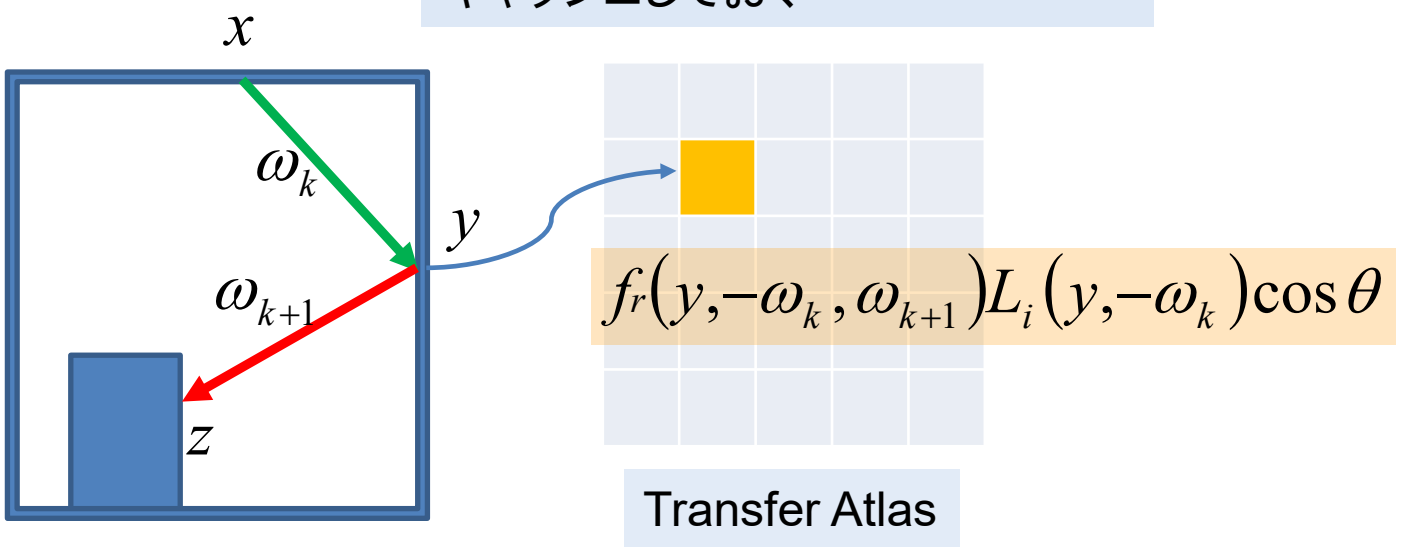
テッセレーション

- CPUベースの手法だと難しい
  - テッセレーション済みのモデル作成
    - メモリ使用量が増加
  - ソフトウェアで対応
    - 実装が複雑
- Ray Bundle Tracing では自然に可能
  - ハードウェアテッセレーション
  - ランタイムとベイクで同じテッセレーション処理
  - 省メモリ

ランバート以外の相互反射

- 1サンプル分の輝度をキャッシュするバッファ

k番目の計算の時点で、
k+1番目の方向へ反射する輝度を
キャッシュしておく



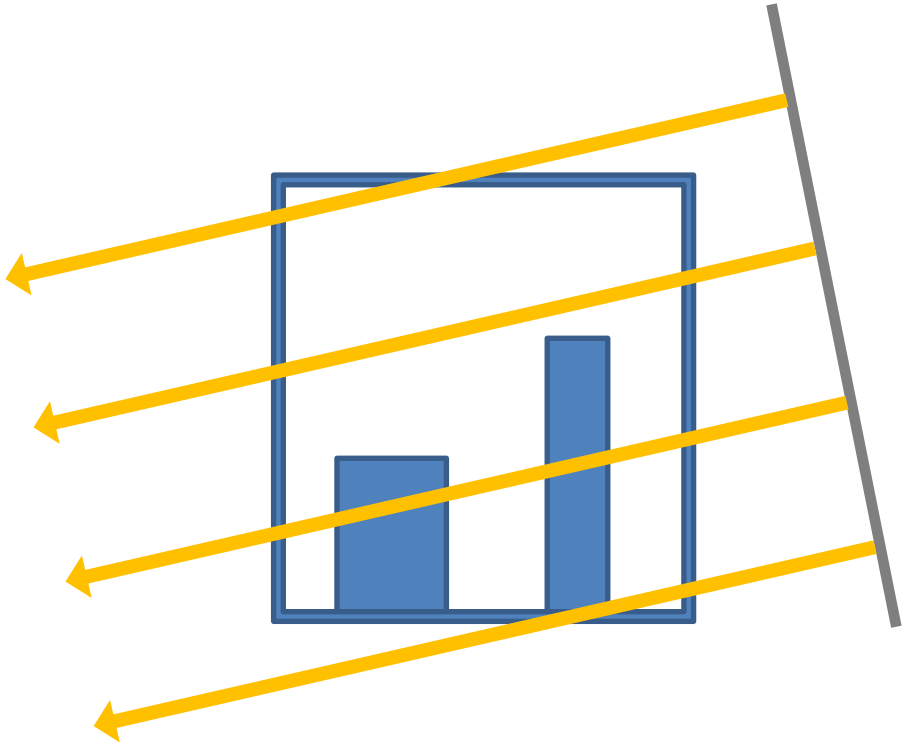
[Hermes et al. 2010]

空間的に大きいシーン

Ray Bundle の密度が足りない → ライトリーク

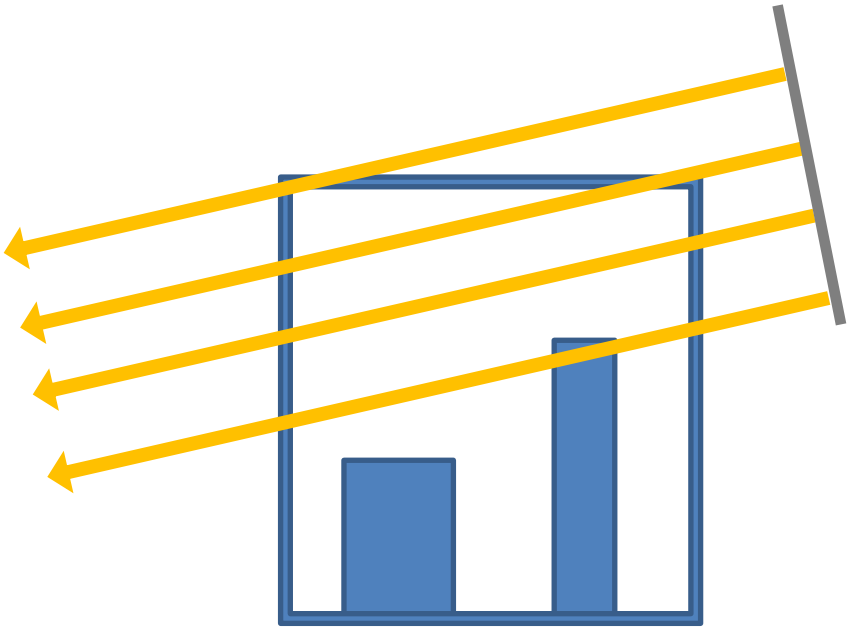
空間的に大きいシーン

- Ray Bundle の密度を上げる
  - タイリング
    - 1方向の計算を複数回に分ける



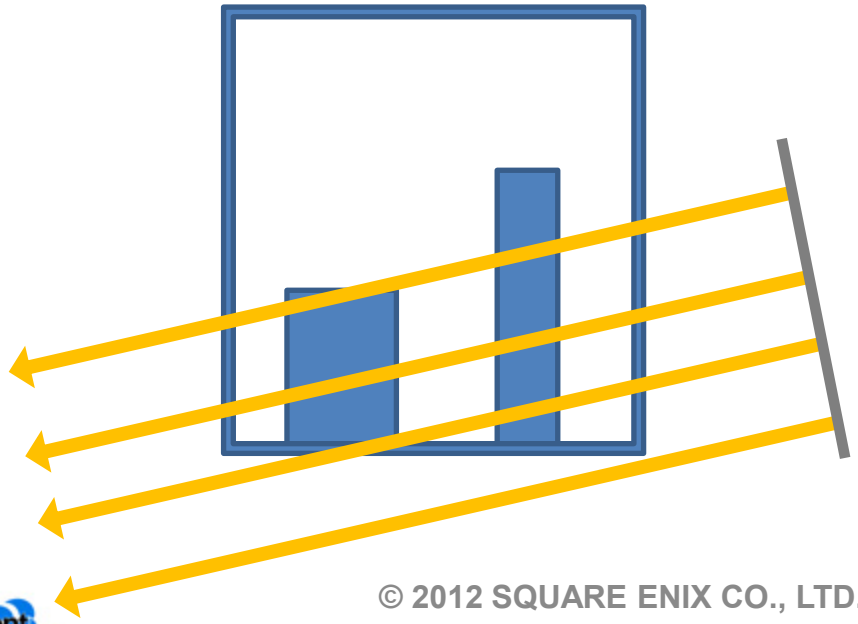
空間的に大きいシーン

- Ray Bundle の密度を上げる
  - タイリング
    - 1方向の計算を複数回に分ける



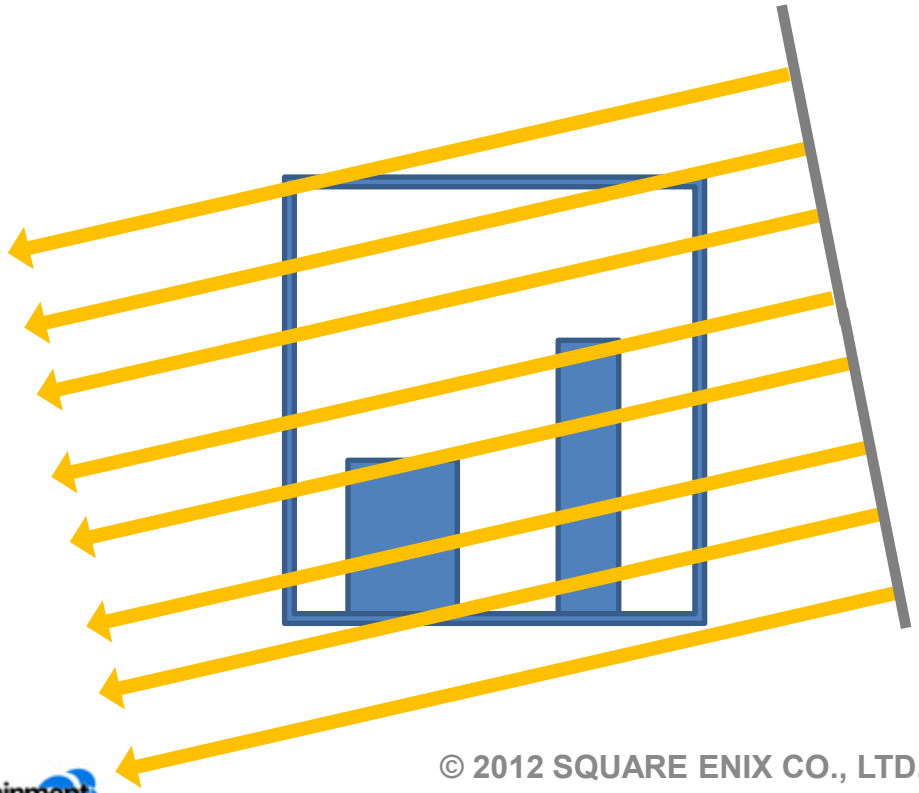
空間的に大きいシーン

- Ray Bundle の密度を上げる
  - タイリング
    - 1方向の計算を複数回に分ける



空間的に大きいシーン

- Ray Bundle の密度を上げる
  - タイリング
    - 1方向の計算を複数回に分ける



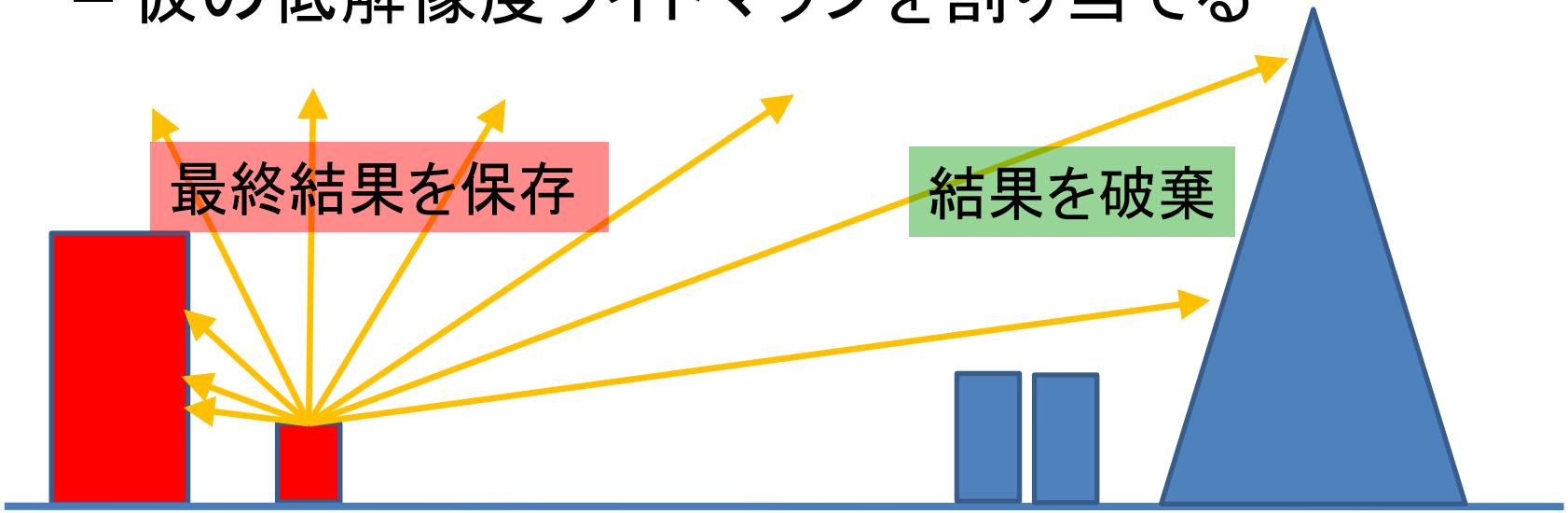
密度は無限大！

データ量的に大きいシーン

- データ量はライトマップサイズに依存
- 全てVRAMにのせる必要がある
  - 大きすぎてのらない！

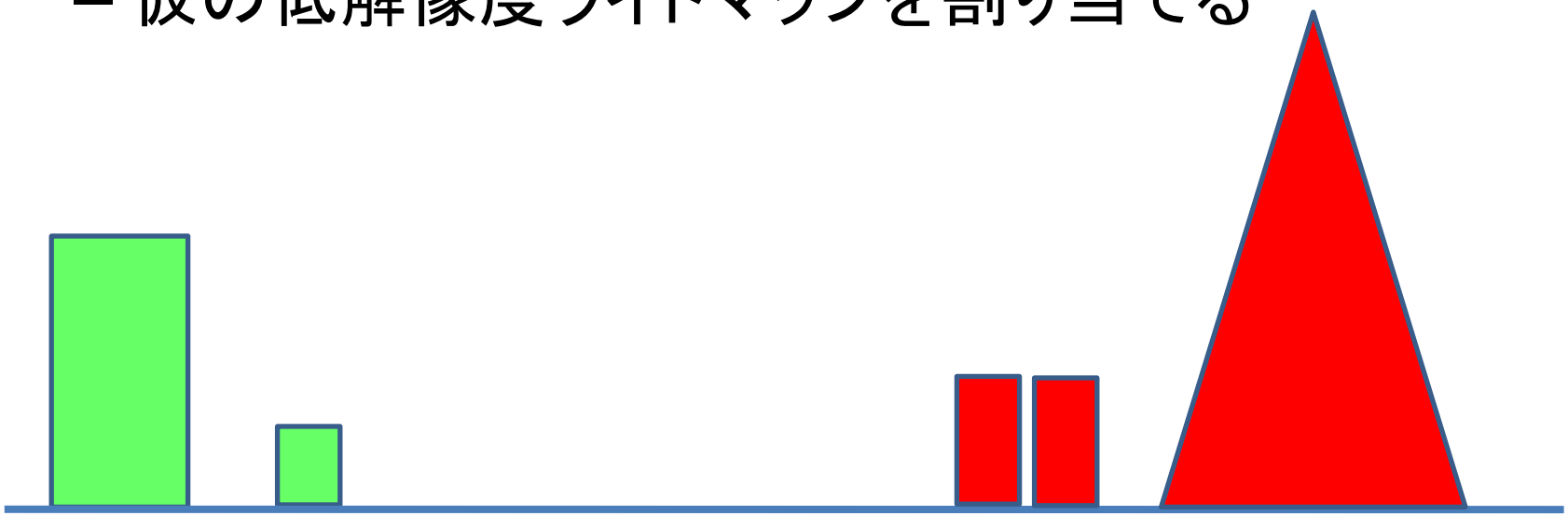
分割ベイク

- シーンを分割してLOD
  - Ray Bundle 密度調整
  - 仮の低解像度ライトマップを割り当てる



分割ベイク

- シーンを分割してLOD
  - Ray Bundle 密度調整
  - 仮の低解像度ライトマップを割り当てる



低解像度のライトマップ

通常の解像度

部分ベイク

- 良くある要望 ＝ 一部だけ焼き直したい
  - モデルをちょっと横に移動
  - ライトマップ1枚の解像度を変更
- 大きいシーン全体を焼き直すのは無駄
  - 特定部分を選んで分割ベイク
  - 遠くは低解像度で処理

制限への対応

- 光沢反射 — 対応
- 鏡面反射 — 非対応
- 誤差 — 対応
- メモリ — 対応

静的GI

- 屋外 / 太陽の光

静的GI

- 屋外 / 太陽の光

静的GI

- 屋外 / 環境光

静的GI

- 屋外 / 環境光

参考文献

- HACHISUKA, T. 2005. High-quality global illumination rendering using rasterization. In GPU Gems 2. Addison-Wesley Professional, ch. 38, 615-634.
- HERMES, J., HENRICH, N., GROSCH, T., AND MUELLER, S. 2010. Global illumination using parallel global ray-bundles. In Vision, Modeling and Visualization.
- KAJIYA, J. T. 1986. The rendering equation. SIGGRAPH Comput. Graph. 20, 143-150.
- LLOYD, B., TUFT, D., YOON, S.-E., AND MANOCHA, D. 2006. Warping and partitioning for low error shadow maps. In Proc. of EGSR 2006, 215-226.
- NIESSNER, M., SCHAFER, H., STAMMINGER, M. 2010. Fast indirect illumination using layered depth images. The Visual Computer, 26, 6-8, 679-686.
- SMITS, B., SHIRLEY, P., AND STARK, M. M. 2000. Direct ray tracing of smoothed and displacement mapped triangles. Tech. rep.
- THIBIEROZ, N. 2011. Order-independent transparency using per pixel linked lists. In GPU Pro 2. AK Peters, ch. VII, 2, 409-431.
- VEACH, E., AND GUIBAS, L. J. 1995. Optimally combining sampling techniques for monte carlo rendering. In Proc. of SIGGRAPH'95, 419-428.
- YANG, J. C., HENSLEY, J., GRUN, H., AND THIBIEROZ, N. 2010. Real-time concurrent linked list construction on the gpu. Comput. Graph. Forum 29, 1297-1304.
- The polygon models are courtesy of Robert W. Sumner, Jovan Popovic (http://people.csail.mit.edu/sumner/research/deftransfer/data.html)

ご清聴ありがとうございました