# Wavelet Lighting

田村尚希 †　　藤田将洋
† 株式会社ピラミッド

平成 16 年 3 月 26 日作成
平成 16 年 3 月 26 日改訂

図 1: wavelet lighting 実行結果。アテナ像のモデルの著作権は © De Espona(http://www.3dshort.com/) に帰属する。

# 1　概要

近年の Graphics Hardware、Programmable Shader の発展と共に、従来オフラインにおける手法であったグローバルイルミネーションの技術が、リアルタイム環境においても実現されつつある。特に SIGGRAPH 2002 において Sloan らによって発表された Precomputed Radiance Transfer(事前計算済み放射輝度伝達)[2] は、RealTime Global Illumination 技術に対して大きな可能性を示した。さらに SIGGRAPH 2003 においては前述した PRT のさらなる高速化と共に、Wavelet 変換を利用した新たな RealTime Global Illumination 技術の発表が行われた。今回はこの技術に注目し、その実装を通して、RealTime Grahics の新たな可能性を探ると共に、実装を通して得られた成果を報告する。

# 2　All-frequency shadows について

Ng ら [1] によって発表された手法は、Sloan ら [2] の手法と以下の点で類似している。

- 事前計算によって遮蔽を計算する
- 基本的にオブジェクトは固定である
- 環境マップによる Lighting
- 視点・光源は自由に動かせる

しかし、さらなる利点として以下のような点が挙げられる

- Wavelet 変換によるデータの圧縮
- Wavelet 変換の局所性による高周波照明効果の正確な再現

# 3　本手法の特徴

本手法の特徴は、以下の通りである。

- **HDRI からの正確な光積分**
  HDRI 画像を全方向からの照明環境と見立て、それらすべての方向からの照明の寄与 (照明の積分) をウェーブレット展開を持ちいて正確に表現することができる。
- **正確な遮蔽と影の表現**
  HDRI は照明の強さの変化が非常に大きいが、照明が強い方向から生じる

くっきりした影と全体からの柔らかい影の両方をノイズなしに表現することができる。これは球面調和ライティング (PRT) が柔らかい影しか表現できないことと、通常のオフラインでのモンテカルロ法では影に大きなノイズが出ることに比べて大きなアドバンテージである。

- **動的な照明変化**
  双方物面マッピングや動的キューブマッピングを用いることで、周りの照明環境を動的に取得し、照明をリアルタイムに変化させることができる。このとき影なども同時に変化する。

# 4 実装

## 4.1 前処理

前処理は、以下のような手続きで行っている。

1. モデルの各頂点において
   (a) Graphics Hardware を用いキューブマップレンダリング (pbuffer によるオフスクリーンレンダリング) によって遮蔽情報を取得する。
   (b) キューブマップ画像を、CPU に読み戻して Dual Paraboloid 画像に変換。
   (c) Dual Paraboloid 画像に関して Wavelet 変換を行う。
   (d) Wavelet 変換後の画像データに対して、係数の小さなデータの破棄、残った係数の量子化を行う。
2. Wavelet 変換・圧縮後のデータをファイルに保存。

## 4.2 レンダリング時の処理

1. 各頂点の Wavelet 係数をファイルから取得する。
2. 動的に照明環境の DualParaboloid 画像を生成し Wavelet 変換を適用する。
3. モデルの各頂点において、照明環境の Wavelet 係数ベクトルと頂点の遮蔽 Wavelet 係数ベクトルの内積を計算し、頂点色を求める。
4. 得られた頂点色を Graphics Hardware に転送してレンダリングを行う。
5. 照明環境に変更が生じた場合は (2) から繰り返す。

# 5 実装の詳細

## 5.1 定式化

環境マップによる直接光を考える。頂点 $\boldsymbol{x}$ から方向 $\omega$ へ出射していく放射輝度は、以下の式で求められる。

$$B(\boldsymbol{x}, \omega_o) = \int_\Omega L(\omega) V(\boldsymbol{x}, \omega) f_r(\boldsymbol{x}, \omega \to \omega_o)(\omega \cdot n(\boldsymbol{x}))\, d\omega$$

ここで、$B, L$ は放射輝度、$V$ は可視関数、$n(\boldsymbol{x})$ は点 $\boldsymbol{x}$ での法線方向、 $f_r$ は双方向反射分布関数、$\Omega$ は法線上の半球を表す。

双方向反射分布関数 $f_r$ をディフューズと仮定することで、積分の中身は $\omega_o$ に依存しなくなる。積分の中の $V, f_r, n$ をまとめて、

$$T(\boldsymbol{x}, \omega) = V(\boldsymbol{x}, \omega) f_r(\boldsymbol{x})(\omega \cdot n(\boldsymbol{x})).$$

と定義できる。

これを用いて各頂点の放射輝度 $B(\boldsymbol{x}_i)$ は、

$$B(\boldsymbol{x}_i) = \sum_j T(\boldsymbol{x}, \omega_j) L(\omega_j).$$

と表すことができる。これをすべての頂点についてまとめると、以下の行列・ベクトル表現を得る。

$$B = TL$$

ここで $B, L$ は要素数が頂点個のベクトルであり、 $T$ は行の要素が方向 $\omega$ の分割サンプル数、列が頂点個の行列となる。

## 5.2 Wavelet 変換

上で求めた $T$,$L$ は非常に大きな行列、ベクトルとなる。(入力データをキューブマップとしサンプリングサイズを $64 \times 64$ とすると、一頂点あたり $64 \times 64 \times 6 = 24,576$ 要素のベクトルになる。) このため、計算量的にもデータ容量的にも圧縮を行わなくては現実的な速度でレンダリングは行えない。

そこで $B, L$ に対して Wavelet 変換を行い、前述の問題点を克服する。Wavelet 変換の数学的背景は [3, 4] などを参照する事を薦め、ここでは割愛する。

具体的に行った Wavelet 変換は、画像処理などの分野で広く利用される Multi Resolution Analysis(多重解像度解析,MRA) で、マザーウェーブレットには最も基本的な Wavelet 基底である Haar Wavelet を用いている。Wavelet 変換の定式化は以下のようになる。ある位置 $x$ での放射輝度は以下のように表せる

$$B(\boldsymbol{x}) = \int_\Omega T(\boldsymbol{x}, \omega) L(\omega) d\omega. \tag{1}$$

ここで Wavelet 基底関数 $\psi$ を用いて T,L の Wavelet 変換を以下のように行う。

$$\begin{aligned} T_i(\boldsymbol{x}) &= \int_\Omega T(\boldsymbol{x},\omega)\psi_i(\omega)d\omega. \\ L_i &= \int_\Omega L(\omega)\psi_i(\omega)d\omega. \end{aligned}$$

これらを Wavelet 係数と呼ぶ。逆に Wavelet 係数から、元の関数の復元は以下のように記述できる。

$$\begin{aligned} \tilde{T}(\boldsymbol{x},\omega) &= \sum_i T_i(\boldsymbol{x})\psi_i(\omega). \\ \tilde{L}(\omega) &= \sum_i L_i\psi_i(\omega). \end{aligned}$$

ここで上式を用い式 (1) は以下のように変形できる。

$$B(\boldsymbol{x}) = \int_\Omega \left(\sum_i T_i(\boldsymbol{x})\psi_i(\omega)\right)\left(\sum_j L_j\psi_j(\omega)\right)d\omega.$$

ここで Haar Wavelet は直交基底であるので以下のような性質を持つ。

$$< \psi_i, \psi_j > = \begin{cases} 1 & (i = j) \\ 0 & (i \neq j) \end{cases}$$

よってに最終的に以下を得る。

$$B(\boldsymbol{x}) = \sum_i T_i(\boldsymbol{x})L_i.$$

これは Wavelet 空間での内積値と、球面空間 $\Omega$ での積分値が等しい事を示している。

## 5.3 量子化・圧縮

Wavelet 変換のみを行っただけでは、データ量自体は変化しない。しかし変換により係数が局所化し、非常に少ない係数のみが結果に寄与するようになる。

そこで、0 に近いような係数を破棄し、さらに残った少数の係数も量子化することにより、データの圧縮を図る事ができる。またさらに頂点色の計算時においても非常に少ない数の係数同士をかければ良い事になるので、劇的な計算速度な向上が図れる。具体的には、各頂点において係数の最大値、最小値を求め、その区間を 8bit(255 段階) に量子化している。

## 5.4 疎行列計算

疎行列とは、行列内の多くの要素が 0 になるような行列の事である。このような行列の演算を通常のように全要素同士をかけるという方法では効率の上から好ましくない。前述した圧縮をかけた Wavelet 係数行列はまさに疎行列であり、その演算の効率化のため以下のような手順で高速化を図った。

1. Wavelet 係数行列の各行ベクトルを 256 要素毎のブロックに分け、そのブロックに含まれる 0 でない係数の個数を記録する。256 要素毎のブロックに分けるのは、要素の index が 8bit で収まるようにするためである。

2. 0 でない係数は、そのブロックの中での index 及び量子化されたデータの二つを保存する。よって 0 でない Wavelet 係数は、index 8bit + 実データ 8bit で 16bit のデータとなる。

3. 以下のような処理を行い、頂点色を計算する。

---

```
NBlocks = PARABOLOIDWIDTH * PARABOLOIDHEIGHT * 2 / 256
count = 0;

for ( i = 0; i < NBlocks; ++i ) {
    // もしブロックに含まれる Wavelet 係数がゼロだったらスキップする
    if (CoeffNum[i] <= 0) continue;
    for( j = 0; j < CoeffNum[i]; ++j ) {
        // 対応する元の行ベクトルでの位置を計算する
        index = i * 256 + waveletcoeff[ count + j ].index;

        // 量子化した Wavelet 係数を復元する
        fval = (float)(waveletcoeff[ couunt + j ].val) *
                       (maxval - minval) / 256.0f + minval);

        color[ 0 ] += fval * Lred  [ index ];
        color[ 1 ] += fval * Lgreen[ index ];
        color[ 2 ] += fval * Lblue [ index ];

    }

    count += waveletcoeff[ i ];
}
```

---

## 5.5 双放物面マップ

**双放物面マップ** (*dual paraboloid map*) は、2 枚のテクスチャマップで 360 度の視点を表現する。通常の環境マップの表現には、スフィアマップ (sphere map) やキューブマップ (cube map) があるが、スフィアマップには特異点がありまた歪みが大きく、キューブマップは 6 枚のテクスチャマップを使うためにリソースを多く使うという欠点がある。

双放物面マップによる環境マップは、スフィアマップやキューブマップに比べて歪みが少ない。また 2 枚のテクスチャで表現できるので、GPU で動的に生成するときにキューブマップでは 6 回レンダリングする必要があるのに対して、 2 回で済むという非常に有益な形式である。

双放物面マップは、今までの固定パイプラインの GPU で生成するのは困難であったが、近年のシェーダを搭載する GPU を用いれば、頂点シェーダで双放物面の変形式を簡単に記述することが可能である。

## 5.6 グレアフィルタ

照明環境が HDR であることの知覚的な効果を強調するために、グレアフィルタをポストエフェクトで適用することにした。グレアフィルタの手順は以下の通りである。

1. シーンを画面の 1/4 の大きさの浮動小数点 pbuffer オフスクリーンにレンダリングする。このとき、照明値が 1.0 以上の場合のみを pbuffer に出力するようにする (フラグメントシェーダを用いて)

2. (1) で照明値が高い部分を抜き出した画像に、Graphics Hardware でガウスフィルタシェーダを適用してぼかし処理を加える

3. シーンを通常にレンダリングし、(2) のグレア画像を合成する。

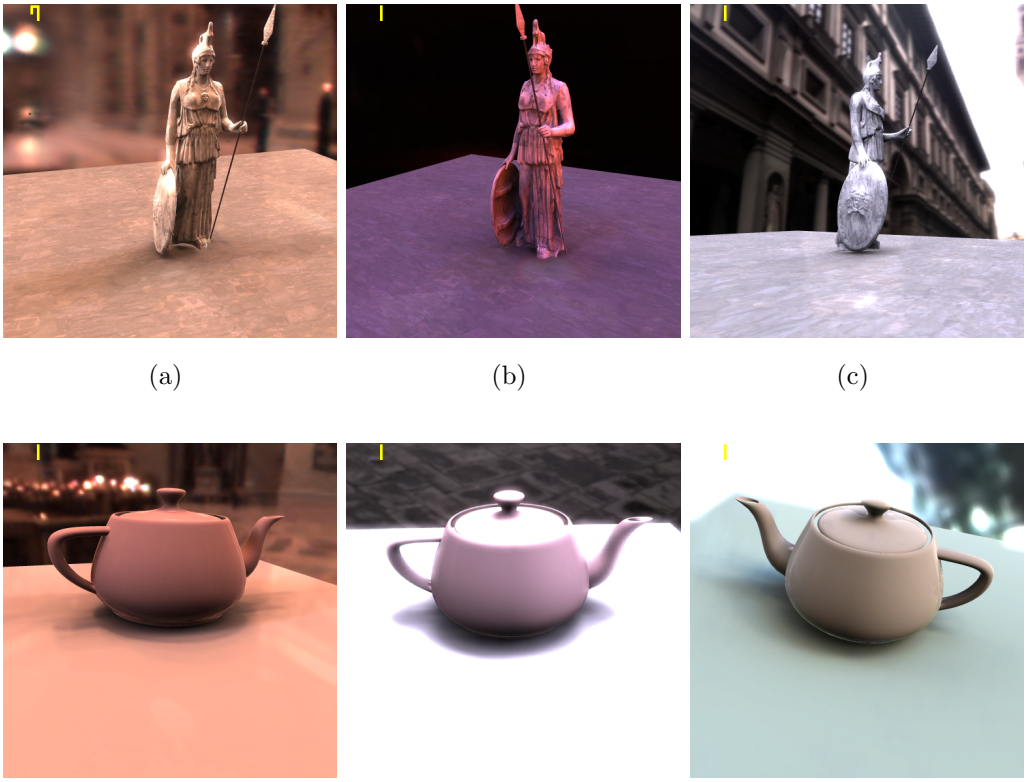
以上により、眩しさの知覚的な表現が可能になる。

## 5.7 実装環境

CPU:Pentium4 2.6G Hz
メモリ:512MB
GraphicsCard:GeForceFX 5200
言語:C++,OpenGL

# 6　実行結果

## 6.1　実行画像



(a)　(b)　(c)

(d)　(e)　(f)

図 2:　各種 HDRI 照明環境でのレンダリング結果画像

## 6.2　実行速度

| モデル名 | 頂点数 | フレームレート |
|---|---|---|
| アテナ像 | 40,000 頂点 | 3 FPS |
| ティーポット | 20,000 頂点 | 5 FPS |
| ティーポット | 3,000 頂点 | 12 FPS |

### 6.3 前計算時間

| モデル名 | 頂点数 | 時間 |
|---|---|---|
| アテナ像 | 40,000 頂点 | 30 minutes |
| ティーポット | 20,000 頂点 | 15 minutes |
| ティーポット | 3,000 頂点 | 4 minutes |

## 参考文献


[1] Ren Ng, Ravi Ramamoorthi and Pat Hanrahan, All-frequency shadows using non-linerar wavelet lighting approximation, ACM Transaction on Graphics(SIGGRAPH 2003), pp. 376 - 381, 2003.

[2] Peter-Pike Sloan, Jan Kautz, and John Snyder, Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments, SIGGRAPH 2002, pp. 527 - 536,July, 2002.

[3] 新井康平著 ウェーブレット解析の基礎理論 森北出版

[4] 酒井幸一著 ディジタル画像処理の基礎と応用 CQ 出版