レイトレーシングによるコンピュータグラフィクス入門
- 環境マッピング -

金森 由博*

2014 年 5 月 16 日

1 課題

ソースコード `EnvironmentMap.cpp` ファイルの `fetchColor` 関数および `readTexture` 関数を実装し、環境マッピングを実現しよう。

2 環境マッピング

金属のような材質の表面では鏡面反射成分が強く、周囲の環境にある様々な物体が映り込む。映り込みを正確に表現するには、レイトレーシングを行えばよい。しかし一般にレイトレーシングは計算負荷が高いので、近似的に映り込みを計算する方法として、環境マッピング (environment mapping) あるいはリフレクションマッピング (reflection mapping) と呼ばれるものがある。この方法では、周囲の環境を画像として与え (この画像を環境マップと呼ぶ)、対象となる物体を囲む 1 つの非常に大きな仮想球の球面上に貼り付ける (マッピングする)。そして、レイトレーシングと同様に、レイと物体の交点で反射ベクトルを計算し、この延長上で仮想球との交点を求め、その点の色を物体の色としてマッピングする (図 1)。普通のレイトレーシングと異なるのは、もし反射ベクトルと物体が交差してもそれを無視する、つまりレイと物体との交差判定は 1 回しかしない、というところである。同様に、ガラスのような材質の屈折を扱う場合に、屈折ベクトルを 1 回だけ計算してその方向の色を取得する、という方法も考えられる。これはリフラクションマッピング (refraction mapping) と呼ばれる。

環境マッピングにおいて、画像のどの画素の色を取得するかは、レイと物体の交点の位置にはよらず、反射ベクトル (あるいはリフラクションマッピングなら屈折ベクトル) だけに依存する。これは、周囲の環境が対象となる物体から十分離れている、つまり周囲の光が無限遠から届くと仮定しているので、交点の位置は相対的に無視できるからである。

この実験ではレイトレーシングを扱うので、環境マッピングのようにレイの交点計算を 1 回だけに限る必要はない。とはいえ他に適切な用語が見当たらないため、以下の説明で「環境マッピング」という言葉は、周囲の環境を画像 (環境マップ) で与える方法を指し、交点計算の回数は 1 回に限定しないものとする。

環境マップは、球の内面をスクリーンとして、周囲の環境を映した画像を表す。実際には球面画像のままでは取り扱いにくいので、通常は平面に展開したものが用いられる。代表的なものとして、球をサイコロのよう

* kanamori@cs.tsukuba.ac.jp

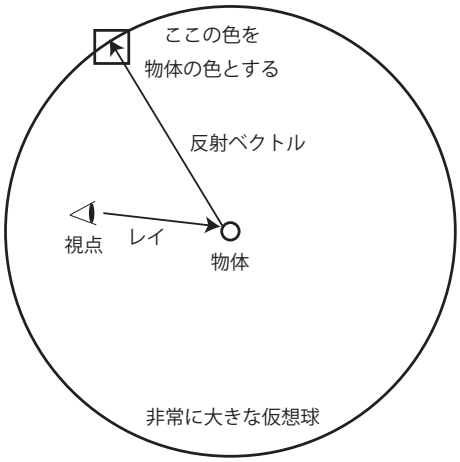

図 1　環境マッピング.

に立方体と見なして 6 面の画像を与えるもの (キューブマップ; Cube Map)、データ量を減らすために球面を魚眼レンズを使って撮影したような 1 枚の画像に収めるもの (Angular Map またはスフィアマップ; Sphere Map) などがある (図 2)。この実験では、キューブマップのみを対象とする。

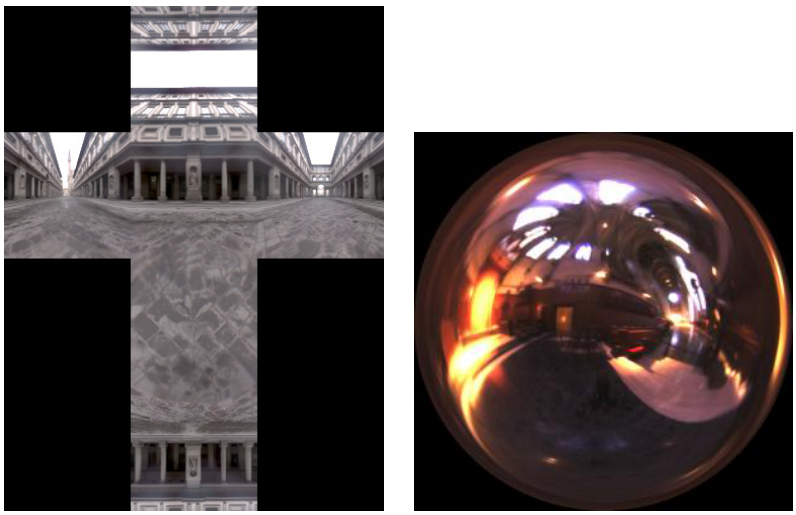

図 2　キューブマップ (左) と Angular Map (右) の例.

3　High Dynamic Range (HDR) 画像

CG 画像の色は、赤 (Red)、緑 (Green)、青 (Blue) からなる 3 値 (RGB) で表されることが多く、それぞれ 8 ビット、すなわち 0 から 255 までの 256 段階で表現されることが一般的である。しかし、実世界の光の強さは 256 段階では到底表現できない。この光の強さの段階をダイナミックレンジ (Dynamic Range) と呼び、256 段階を超える画像を高ダイナミックレンジ (High Dynamic Range; HDR) 画像と呼ぶ。これに対して、ダイナミックレンジの低い画像を低ダイナミックレンジ (Low Dynamic Range; LDR) 画像と呼ぶ。

この実験で扱う HDR 画像フォーマットは、Radiance 形式 (拡張子 .hdr) である。R, G, B それぞれを浮動小数点数とするとデータ量が多くなるため[*1]、R, G, B それぞれが 8 ビットで表現され、さらに浮動小数点数の指数部を表すために E(Exponent) という要素が加えられている。つまり各画素ごとに RGBE の 4 つの要素があり、計 8×4=32 ビットである。このファイルの読み取りは `hdr{.cpp/.h}` で行っており、特に修正する必要はない。Radiance 形式の環境マップは、この分野の世界的権威である Paul Debevec のサイト[*2]からダウンロードできるので、いろいろ試してみてほしい。

4 プログラムについて

前述の通り、この実験のプログラムでは環境マップとしてキューブマップ (図 2 左) を用いる[*3]。この立方体の各面を、図 3 のように名前を付ける[*4]。

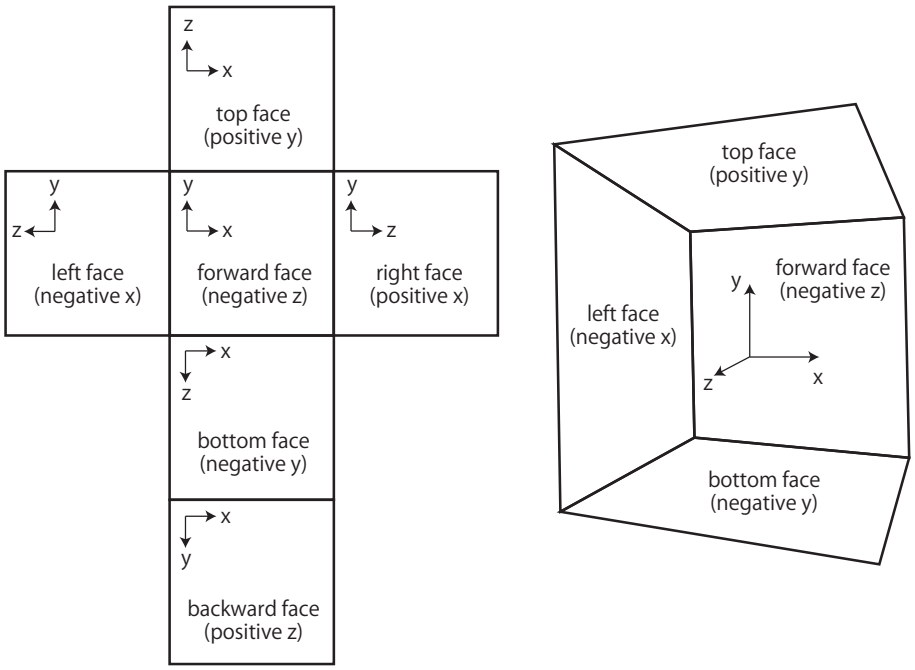

図 3 キューブマップの展開図 (左) と立方体に復元した図 (右).

実験で用いるプログラムでは、`EnvironmentMap.cpp` の `loadFromCrossHDR` 関数で環境マップのデータを読み込んでいる。読み込み先のバッファのサイズは、横幅がキューブマップの各面の横幅 1 つ分、縦幅が各面の 6 つ分であり、各面の配置は図 4 のようになっている。

まず `EnvironmentMap.cpp` でプログラムコードを追加すべきところは、`fetchColor` 関数である。各面の座標は [0,1]×[0,1] の範囲にパラメータ化されており (各面の左下が (0,0) で、右上が (1,1) に対応)、与えられ

[*1] Radiance 形式が登場した当初はメモリが貴重であったため、メモリ消費量が少なくなるよう工夫されている。が、その後に登場した OpenEXR フォーマットは、RGB それぞれを 32 ビット浮動小数点数で保持する。このフォーマットはスターウォーズで有名な Industrial Light&Magic (ILM) 社が開発したもので、映画の編集作業などに用いられている。

[*2] http://www.debevec.org/probes/

[*3] さらに言えば、展開図が縦向きのものと横向きのものがあるが、縦向きのものを仮定している。

[*4] z 方向の負の向きが前面 (forward face) となっているのは、右手系を採用している OpenGL に対応させているためである。

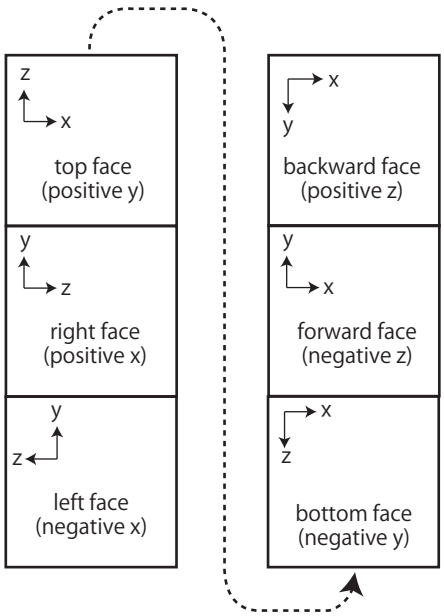

図 4　キューブマップの各面のメモリ配置図. 紙面の都合で途中で分断されているが, 実際のメモリには 6 面が縦につながるよう配置されている.

たレイの方向ベクトルに基づいて、レイが面に交差する点のパラメータを計算する。この計算のためには、立方体の中心を頂点とし、対象となる面を底面とするような四角錐を考え、底面との交点を計算する。

次に修正すべきは、`readTexture` 関数である。現状では、特定の 1 画素の色だけを取得するようになっているが、これでは描画結果にブロック状の視覚的不具合が出てしまう。そこで、各面の座標を表すパラメータを用いて、2×2 画素から色を線形補間によって求めるよう、修正する。

線形補間の計算方法について解説する。`readTexture` 関数の引数は、キューブマップのひとつの面の先頭アドレス `addr`、その面でのパラメータ座標 $u \in [0,1]$ および $v \in [0,1]$ の 3 つである。パラメータ u, v で表されるピクセル位置 $\mathbf{p}$ は、面の横幅を w、縦幅を h とすると、(uw, vh) である (図 5 左)。ここで、点 $\mathbf{p}$ の座標値は整数値ではなく実数値を取り、単に一番近い画素の色を取得すると (最近傍サンプリング; nearest-neighbor sampling)、色の変化が不連続になってしまう。そこで、近傍の画素から色を補間し、色の変化を滑らかにする。点 $\mathbf{p}$ を取り囲む整数の座標値を持つ点 (格子点) を、$\mathbf{p}_{00}$, $\mathbf{p}_{10}$, $\mathbf{p}_{11}$, $\mathbf{p}_{01}$ とする (図 5 右)。点 $\mathbf{p}$ の座標値について、横方向の座標値の小数部分を α、縦方向の座標値の小数部分を β とする。また、点 $\mathbf{p}$ での色 $\mathbf{c}$ を決めるには、格子点 $\mathbf{p}_{00}$, $\mathbf{p}_{10}$, $\mathbf{p}_{11}$, $\mathbf{p}_{01}$ での画素値 $\mathbf{c}_{00}$, $\mathbf{c}_{10}$, $\mathbf{c}_{11}$, $\mathbf{c}_{01}$ を、横方向に $\alpha : 1-\alpha$、縦方向に $\beta : 1-\beta$ の比で混合すればよい。すなわち、

$$\mathbf{c}_0 = (1-\alpha)\mathbf{c}_{00} + \alpha\mathbf{c}_{10} \tag{1}$$
$$\mathbf{c}_1 = (1-\alpha)\mathbf{c}_{01} + \alpha\mathbf{c}_{11} \tag{2}$$
$$\mathbf{c} = (1-\beta)\mathbf{c}_0 + \beta\mathbf{c}_1 \tag{3}$$

このように、2×2 の値から一次式を用いて値を補間することを、双一次補間 (bilinear interpolation) と呼ぶ。

なお、画像の色を参照するとき、画像の縁を超えてデータにアクセスしないよう注意が必要である。どういうことかというと、画像の横幅が W のとき x 座標として $x = -1$ や $x = W$ にアクセスしないようにする、ということである。そのような座標にアクセスすると、場合によってセグメンテーション違反でプログラムが

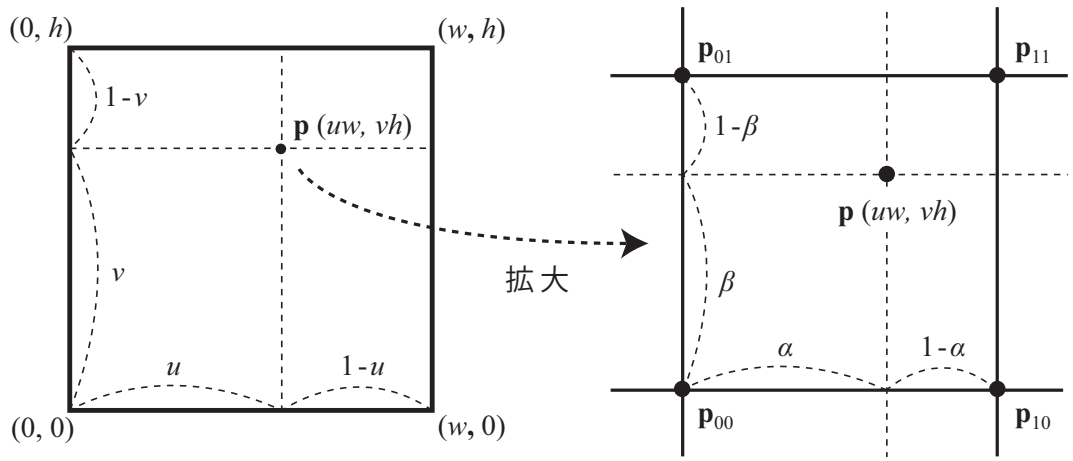

図 5　双一次補間 (bilinear interpolation). パラメータ u, v で表されるピクセル位置 $\mathbf{p}$(左) における画素値は、近傍の画素 $\mathbf{p}_{00}, \mathbf{p}_{10}, \mathbf{p}_{11}, \mathbf{p}_{01}$(右) から補間して求められる.

クラッシュする。これを回避するため、readTexture 関数のサンプルコードでは次のように座標を切り詰めて (clamp して) いる。

readTexture 関数

```
const int x0 = max(0, min((int)(u * m_FaceWidth), m_FaceWidth-1));
const int y0 = max(0, min((int)(v * m_FaceHeight), m_FaceHeight-1));
```

この実験で用いるシーン定義ファイルは envmap_test.cfg である。このファイルには次のような記述がある。

pixel_filter.cfg

```
/* ...（略）... */

environment_light
{
//  filename = "Cubemap/beach_cross.hdr";
//  filename = "Cubemap/building_cross.hdr";
//  filename = "Cubemap/campus_cross.hdr";
//  filename = "Cubemap/galileo_cross.hdr";
//  filename = "Cubemap/grace_cross.hdr";
//  filename = "Cubemap/building_cross.hdr";
//  filename = "Cubemap/kitchen_cross.hdr";
//  filename = "Cubemap/rnl_cross.hdr";
//  filename = "Cubemap/uffizi_cross.hdr";
  filename = "Cubemap/test_environment_map.hdr";
```

```
  hdr_gamma = 1.0;
}

/* ...（略）... */
```

上記の filename から好きなものを選べば、環境マップの画像を切り替えることができる。デバッグの際は test_environment_map.hdr を用いること。この画像を使うと、球面に赤、緑、青の矢印がマッピングされて表示される。これらの矢印はそれぞれ x, y, z の正の方向を表している。これらの矢印が球面にマッピングされたとき、それぞれ同じ方向を向くようになれば fetchColor 関数は完成である。