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
- Image-Based Lighting (IBL) -

金森 由博*

2014 年 5 月 16 日

CG で写実的な画像を作るとき、実世界の照明を再現するのは難しい。だが、環境マッピングで用いた、周囲の景色を撮影した画像 (環境マップ) を使えば、比較的簡単に実現できる。環境マップによって光源の配置を与えてレンダリングを行うことを、Image-Based Lighting (IBL) と呼ぶ。IBL は研究分野では 2000 年前後に登場した技術であり、最近の映画制作では一般的に使われている。今回のレジュメでは、この実験での IBL の実現方法について説明する。

1 課題

シーン定義ファイル `sample_ibl.cfg` を読み込んで、Image-Based Lighting を試してみよう。

2 環境マッピングと Image-Based Lighting

環境マッピングでは、鏡のような物体の表面に、周囲の景色が映り込む様子を実現した。これは次の 2 つの仮定に基づいている。

- 物体は周囲から距離が十分離れている
- 物体表面が完全鏡面 (屈折を扱う場合は屈折面) である

上記の 1 つ目の仮定により、周囲から届く光の入射方向は物体が多少動いても変わらない、つまり届く光は**平行光**として扱われる。また 2 つ目の仮定により、入射した光は反射する一方向 (屈折の場合は、反射と屈折の二方向) のみを追跡すれば、レイトレーシングが実現できた。

Image-Based Lighting (IBL) においても、「物体は周囲から距離が十分離れている」という仮定は有効である。つまり「光がいろいろな方向から平行光として届く」とする。が、IBL では一般に、鏡面反射や屈折以外の反射も扱う。具体的には、この実験で言えば拡散反射や光沢反射などである。拡散反射や光沢反射では、ある方向から入射した光は、単一方向だけではなく、幅広い範囲に向かって反射する。逆に言えば、ある方向に反射する光を計算したければ、いろいろな方向から入射した光の反射光をすべて足し合わせる (積分する) 必要がある。「いろいろな方向から入射」といっても、例えば環境マップの 1 ピクセルごとに入射方向を計算して平行光として明るさを計算、としていては計算時間が膨大になる。そこでレイトレーシングでは、環境マップの中で主に明るいところのピクセルをサンプリングして、それらのサンプル点のみから平行光を計算する。

* kanamori@cs.tsukuba.ac.jp

物体表面から視点に向かう方向が ω_o だとしたとき、その方向の明るさ $I(\omega_o)$ は次式で計算される。

$$I(\omega_o) = I_a + \sum_{\omega_i} \{I_d(\omega_i, \omega_o) + I_s(\omega_i, \omega_o)\} \tag{1}$$

ここで I_a は環境光、ω_i は入射方向、$I_d(\omega_i, \omega_o)$ は入射方向 ω_i に対する出射方向 ω_o への拡散反射による明るさ、そして I_s は入射方向 ω_i に対する出射方向 ω_o への光沢反射による明るさである。要するに、シーン中に平行光源がたくさん存在する、というだけで、計算方法はこれまでと何ら変りない。

3 プログラミング内容について

ここまでのレジュメの内容がすべて実装できていれば、IBL を実現するために新たにプログラミングする必要は何もない。配布された IBL 用のシーンファイルに、環境マップからサンプリングされた平行光を利用するための設定 `environment_light` が記述されているので、そのファイルを読み込んでレンダリングすればIBL が実現できる。IBL 用のシーンファイルには、次のような記述がある。

sample_ibl.cfg

```
environment_light
{
  filename = "Cubemap/uffizi_cross.hdr"; // 環境マップのファイル

  num_ibl_samples = 128; // サンプリングする平行光源の数
  light_scale = 0.001; // 明るさを調整するための係数

  hdr_gamma = 0.5; // HDR 画像のガンマ補正のための係数
}
```

上記の `environment_light` の各種パラメータを変更することで、レンダリングの品質や計算時間が変化する。特に重要なのが、サンプリングする平行光源の数を指定する `num_ibl_samples` というパラメータである。これを大きくすれば品質は向上するが、ある程度まで大きくすると結果はあまり変わらなくなる。なお、サンプル数を増やすと全体的な明るさが変わってしまうので、`light_scale` を調整する必要がある。

対応する環境マップを使えば、鏡面反射や屈折する物体もシーン中に混在させることができる。ただし、計算時間が非常に長くなることに注意してほしい。1 つのレイあたり `num_ibl_samples` で指定した数の平行光による明るさを計算することになる。よって、レイの数が増えるような設定 (スクリーンの解像度を大きくする、アンチエリアシングのために 1 ピクセルあたり複数のレイを飛ばす) はいきなり行わず、まずレイの数を減らす設定でレンダリングし、カメラの設定を済ませてから改めてレンダリングするようにしてほしい。

4 トラブルシューティング

上で「新たにプログラミングする必要は何もない」と書いたが、それは実装が完璧な場合の話で、次のような問題が起きる場合がある。

■シーンファイルを読み込ませるとプログラムがクラッシュする 課題のプログラムでは、シーンファイルを読み込むと同時に、`EnvironmentMap.cpp` の `fetchColor` 関数を呼び出し、さらにその中で `readTexture` 関

数を使って、環境マップから画素値をサンプリングしている。プログラムがクラッシュするのは、readTexture 関数の実装にバグがある可能性がある。特に、バイリニアフィルタリングを実装したときに、メモリの範囲外アクセスをしていないか確認してみること。

■実行してみると真っ暗になってしまう EnvironmentMap.cpp の fetchColor 関数の実装が間違っていて、上下逆さまのところから画素値を参照している可能性がある。試しに、シーンファイルの一番下に定義されている plane というのをコメントアウトして、球の底の部分が明るくなるか調べてみるとよい。もし底の部分が明るくなるなら、上下逆さまになっている。その場合は fetchColor 関数の実装を修正すること。