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
- レイトレーシング法による画像生成について -

金森 由博*

2014 年 4 月 23 日

1　レイトレーシングとは

コンピュータにおける画像は、画素 (ピクセル; pixel) と呼ばれる正方形の集合[*1]からなり、 コンピュータによる画像生成とは各画素の色を決定していく処理になる。 画像をカメラのフィルムに見立ててコンピュータ内の 3D シーンを撮影するには、観測される光の色を画素ごとに計算することになる。これを行うのがレイトレーシング法 (光線追跡法) である。レイトレーシング法は、画像を 3D 空間にあるスクリーンだと見なし、視点から各画素を通過するような光線 (レイ; ray) を飛ばし、そのレイが物体に衝突したら、その物体に届く光の色を計算して画素の色とする。ちなみに「光の挙動を追跡するならなぜ光源から光線を出さないのか」と疑問に思うかもしれないが、光源からたくさん光線を飛ばしても、それらのうち有限の計算で視点まで到達するような光線というのは必ずしも多くなく、非効率だからである[*2]。図 1 に Tuner Whitted による世界初のレイトレーシングの画像を示す。

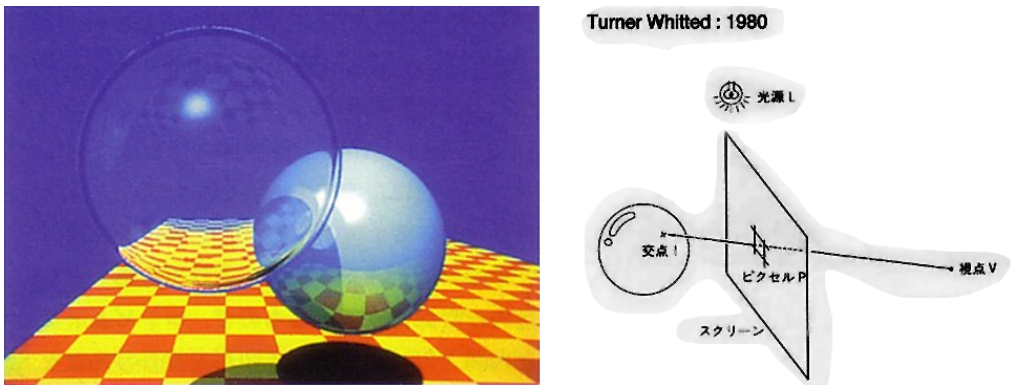

図 1　Tuner Whitted による世界初のレイトレーシング画像 (左) とその模式図 (右).

レイトレーシングの計算は大まかに言うと、次のような手順になる。

* kanamori@cs.tsukuba.ac.jp

*1 最近のディスプレイは解像度が高いのでわかりにくいが、例えば Windows なら「拡大鏡」を使って画面を拡大すれば正方形の集合が見える。

*2 ただし、より高度なアルゴリズムでは光源からも光の経路を計算する。その場合でも、最終的に画像を生成する際には視点からレイを飛ばして画素の色を計算する。

画像の表示

```
カメラの情報をセットアップ;

for (画像の各画素について) {
  カメラを視点とするレイを生成;
  レイを追跡; // 「レイの追跡」を実行
  計算結果の色を画素に記録する;
}

画像を表示;
```

レイの追跡

```
再帰の深さが指定以上ならリターン;

for (すべての物体について)
  レイが最初にぶつかる物体との交点を探す;

if (レイがどの物体ともぶつからなかった)
  背景の色を取得してリターン;

if (完全鏡反射する材質の場合) {
  レイを反射させて再帰的に計算;
  計算結果を画素の色としてリターン;
} else if (屈折する材質の場合) {
  反射成分について再帰的に計算;
  屈折成分について再帰的に計算;
  反射成分と屈折成分をブレンドした色をリターン;
} else {
  for (すべての光源について) {
    if (レイと物体との交点に光が届いている) {
      光源の反射光の色を累積;
    }
  }
  環境光の光の色を加算;
  計算した色をリターン;
}
```

図 2　レイトレーシングの擬似コード.

2　この実験の内容

レイトレーシングはいまや映画など映像制作に広く使われており、コンピュータグラフィクス (CG) の基礎をなしている。この実験では、レイトレーシングのプログラムを作成することを通じて、CG の基礎を学んでいく。より具体的には、C++ 言語で書かれた雛形のソースコードを、レジュメを見ながら完成させる、という形式で行う。この実験で扱う内容は以下の通りである。

■視線と物体との交差判定　視点から見える物体を調べるために、 視点から放射される直線 (視線) と物体とが互いに交差するかを計算する。 そして視線上の交点のうち、一番視点に近いものを出力する。 まず、球や三角形などを対象として交差判定を行う。

■物体の陰影の計算　簡単のため、光が物体表面で 1 回だけ反射して視点に届くものと仮定すると、 物体表面での反射光の色を計算すればよい。 この色は、物体の材質によって異なる。 いくつかの単純化された材質モデルを用いて、この計算を行う。

■テクスチャマッピング　物体の表面に模様を入れたりするために、画像を取り込むことがある。 物体表面をパラメータ化し、視線と物体表面の交点でのパラメータから、 入力画像の対応する画素の色を取得する。このとき用いる入力画像をテクスチャと呼び、 この操作をテクスチャマッピングという。 簡単のため、白黒 2 色のチェッカーパターンをテクスチャとして、 平面にテクスチャマッピングを行う。 チェッカーパターンは、以下の実験でのデバッグに便利である。

■アンチエリアシング　チェッカーパターンを描画すると、色の変わり目でギザギザが目立つ。 このような視覚的不具合をエリアシングという。 この原因は、画素ごとに、視点と物体とが交差するか否かの 2 値的判断をしているためである。 エリアシングを解消する処理をアンチエリアシングと呼ぶ。 アンチエリアシングにはいくつか方法があるが、 ここでは 1 画素あたり複数の視線を放射し、 それらの色の加重平均によって画素の色を決定することで、エリアシングを低減する。

■光の反射・屈折の計算　ここまでは反射が 1 回のみであるとして単純化したが、実際には 光は視点に届くまでに反射や屈折を繰り返す。 光が鏡面・屈折面のみで反射・屈折するものとして、 光が複数回反射した結果得られる色を計算する。

■環境マッピング　物体を取り囲む背景を画像として与え、 反射光をその背景画像に基づいて計算すると、比較的容易に写実的な画像が得られる。 物体が鏡面であり、光が物体に遮られることなく物体表面で 1 回だけ反射すると仮定する。 このとき、反射光を背景画像の対応する画素によって決定する処理は 一種のテクスチャマッピングであり、 特に、環境マッピング (反射マッピング) と呼ばれる。 サイコロのような六面体の展開図として与えられる背景画像を用い、 環境マッピングを行う。

■イメージベースドライティング　環境マッピングと同様に、周囲から入ってくる光の分布を画像として与え、その照明環境で物体の陰影計算を行う。映画などで一般的に用いられている技術である。

■複雑な 3 次元モデルの読み込み　コンピュータグラフィクスにおいて複雑な形状は、 主に三角形の集合として表現される (いわゆるポリゴンメッシュ)。 このような形状のデータ形式を読み取れるようにし、 形状を構成する各三角形との交差判定を行うことで、画像を出力する。

■空間データ構造を用いたレイトレーシングの高速化　描画対象となるシーンが複雑になると、次第に計算時間が膨大になる。 空間的データ構造を用いて、各視線で検査すべき物体数を減らし、計算を高速化する。

■形状のインスタンス化と座標変換　データサイズの大きな形状データを複数表示したい場合、その個数分だけデータを読み込むとメモリ消費が増えてしまう。そこでデータは 1 つのままで、物体に座標変換を適用しつつ、使いまわすようにする。

■メタボールで表現された物体のレンダリング　メタボール (metaball) という技術を用いると、有機的な形状を表現することができる。 レイとメタボールの交点を求めるには、方程式の根を求める必要がある。

■アニメーションの作成　3D シーン内の物体やカメラを少しずつ動かしながら、レイトレーシングで画像を生成し、それらの画像をつなげれば、アニメーションを作ることができる。別途プログラムによってシーンを記述したファイルを多数生成し、それをプログラムにバッチ処理させて画像を生成して、アニメーションを作る。

3　雛形のプログラムについて

レイトレーシングは計算に時間がかかるため、この実験で雛形として提供するプログラムではプレビュー用に物体を描画する機能を加えてある。この機能を実装するために、OpenGL というグラフィクスライブラリ、そのユーティリティライブラリである GLUT、GUI を提供する GLUI というライブラリを利用している。基本的には OpenGL/GLUT/GLUI のことを勉強しなくてもよいように作ってあるが、機能を追加・修正したい場合は OGLDisplayDriver.cpp/.h を編集してほしい。OpenGL/GLUT については下記 (1) を、GLUI については下記 (2) や、インターネットの他のサイトを参考にしてほしい。

(1) GLUT **による「手抜き」**OpenGL **入門**　http://www.wakayama-u.ac.jp/~tokoi/opengl/libglut.html
(2) GLUI **日本語訳**　http://ktm11.eng.shizuoka.ac.jp/glui/glui.html

プログラムの実行時のスクリーンショットを次に示す。

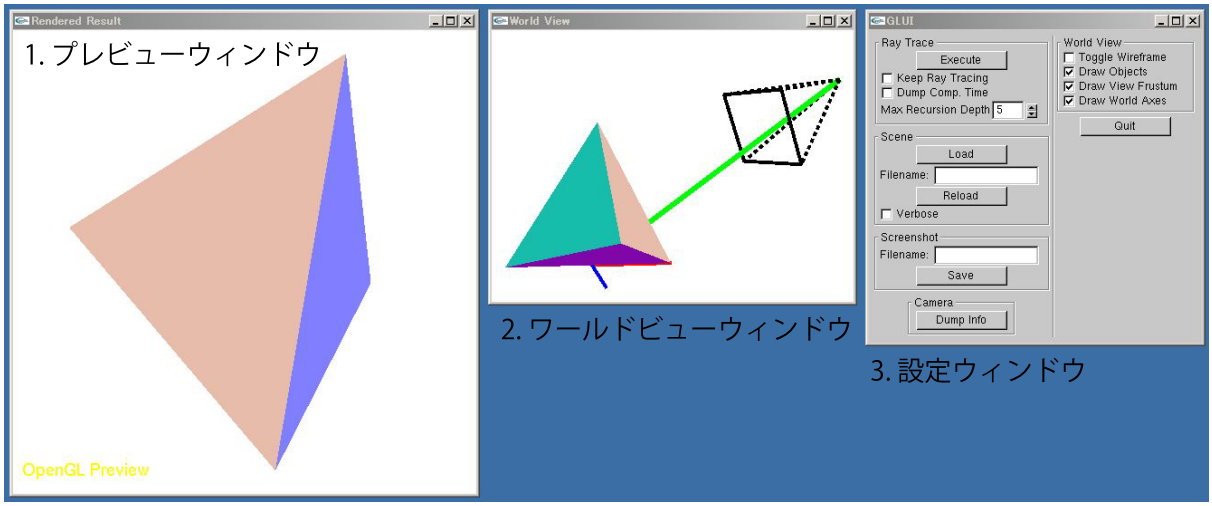

図 3　雛形となるプログラムのスクリーンショット.

以下、各ウィンドウの説明である。

プレビューウィンドウ:　OpenGL で描画されたプレビュー結果が表示される。マウスの左ドラッグでカメラの回転、中央ドラッグでズームイン/ズームアウト、右ドラッグで平行移動を操作できる。後述する設定ウィンドウでレイトレーシングを実行するよう指定すると、この画面にレイトレーシングの描画結果が表示される。

ワールドビューウィンドウ:　上記のプレビューウィンドウとは別の視点から、現在のカメラ位置や物体の配置を確認できる。併せて、ワールド座標の原点に x,y,z 軸がそれぞれ赤 (R)、緑 (G)、青 (B) で表示さ

れる。

設定ウィンドウ: レイトレーシングの実行や、その他設定を行う。

Execute ボタン: このボタンを押すと、プレビューウィンドウにレイトレーシングの計算結果が表示される。

Keep Ray Tracing チェックボックス: これをチェックしておくと、Execute ボタンを押さなくても、プレビューウィンドウでカメラを操作するたびにずっとレイトレーシングが実行される。計算に時間がかかるシーンではチェックを外しておくのをお薦めする。

Dump Comp. Time チェックボックス: これをチェックしておくと、レイトレーシングの計算時間がコンソールに表示される。

Max Recursion Depth: レイトレーシングの再帰計算の深さを指定する。

Scene タブ: **Load** ボタンを押すと、その下の **Filename** で指定されたファイル名のシーン設定ファイルが読み込まれる。シーン設定ファイルを書き換えて再読み込みしたいときは **Reload** ボタンを押せばよい。**Verbose** をチェックしておくとコンソールに読み込み時の細かい情報が表示される。

Screenshot タブ: **Save** ボタンを押すと、**Filename** で指定されたファイル名で、レイトレーシング結果が画像として保存される。

Camera タブ: **Dump Info** ボタンを押すと、カメラの情報がコンソールに表示される。

World View タブ: **Toggle Wireframe** にチェックを入れると**ワールドビューウィンドウ**の形状がワイヤフレームで表示される。その他、**Draw Objects** にチェックを入れるとシーン中の物体、**Draw View Frustum** にチェックを入れると視錐台、**Draw World Axes** にチェックを入れると x,y,z 軸が、それぞれ表示される。

4 プログラム起動時に読み込むシーンファイルの指定方法

この実験では、レジュメごとに “.cfg” という拡張子のシーンファイルを指定する必要があるが、プログラム起動時に毎回、手作業でシーンファイルのファイル名を指定するのは不便である。そこでプログラム起動時に読み込むシーンファイルを指定する方法を説明する。

プログラム起動時に読み込むシーンファイルは、`MiscSettings.cpp` というソースコードの `init` 関数の中で指定されている。具体的には次の行である。

シーンファイル名の指定

```
m_SceneFilename = "default.cfg";
```

この行のシーンファイル名を、レジュメで指定されたシーンファイル名に書き換えてほしい。

5 課題について

ソースコード中には、課題のためにわざと機能を削っている部分（TODO という文字を探してほしい）がある。その部分を各回のレジュメを参考にしながら、自分で考えて埋めてほしい。ソースコードは C++ の知識がなくても、それほど支障がないよう配慮したつもりだが、必要に応じてネットなどの資料で勉強してほしい。

最初の課題として、視線と三角形の交差判定のコードを完成させてほしい。視線と球、平面の交差判定が参考になると思われる。