pythonを用いた3次元グラフィックス

逗子開成高等学校 2年 菅原 瑞 [#36 yawn]

ラスタライズ

はじめに

3次元の物体のデータを、スクリーンに表示できる画像に変換することを、3Dレンダリングという。レンダリングにはいろいろな手法があるが、ここでは、ラスタライズという方法と、レイトレーシングという方法を試した。多く、レンダラーは実行速度が一般に速いとされているc, c++, c#などの言語で書かれることが多いが、あえて、遅いといわれるpythonを用いた。また、OpenGL等のライブラリは用いず，3次元物体のデータ構造から、座標変換までpythonで書いた。

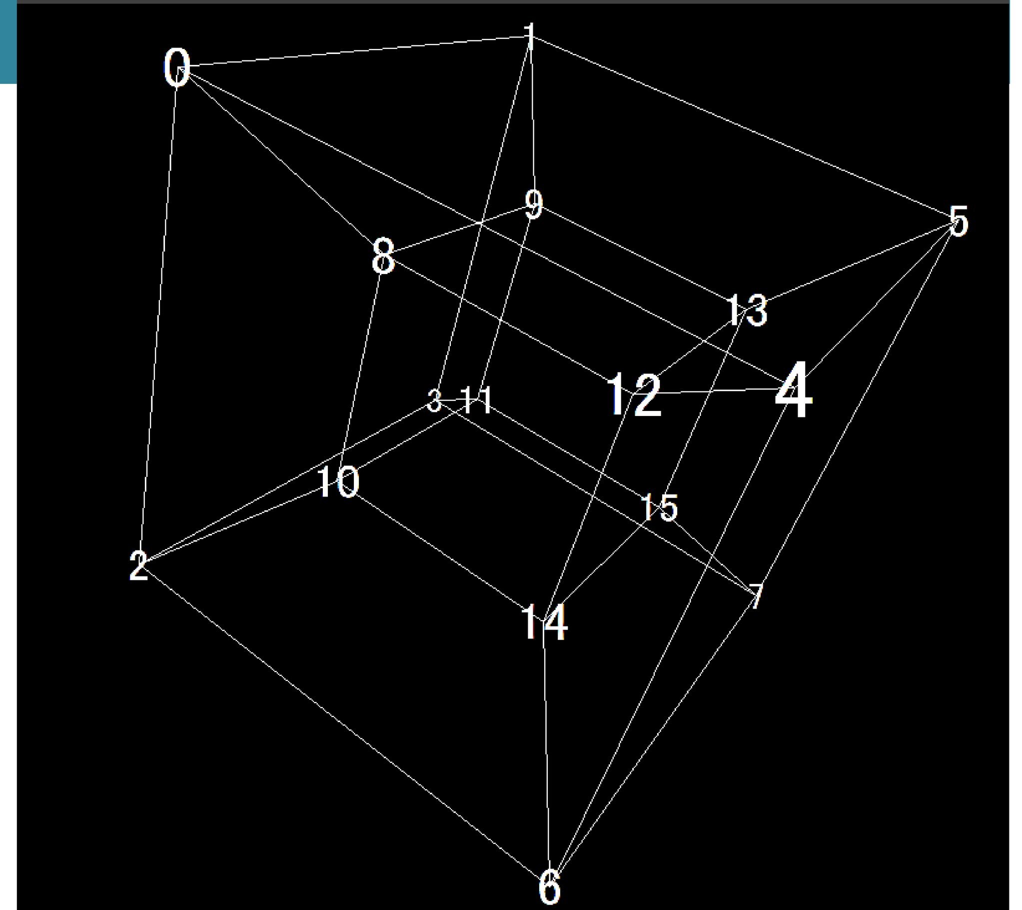
実装

三次元の物体は、頂点の座標を辞書型で記述し、辺はその辺が結ぶ2つの頂点のインデックスを格納し、面は、その面を作る3つの頂点のインデックスを格納するようにした。そして、表示するときには、頂点をスクリーンに投影したときの座標を計算し、それらを結ぶ辺を線として描画した。また、三角関数の逆関数を用いて、指定した軸まわりの回転ができるようにした。

結果

Fig.1は、実際に描画された立体の画像である。回転させて、表示するのには**0.0468秒**であり、リアルタイムでさくさく好きなように動かすことができる。

Fig.1：頂点と辺のみの描画，0.0468sec



今後の課題

どの面が最前面にあるかを判定するためには、その面上の点と視点との距離を比較する必要があり、計算量も、コードを書くこと自体も難しい。今回はできなかった。

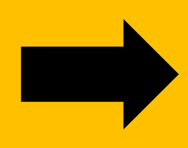
レイトレーシング

レイトレーシングとは、カメラが向いている画角中のそれぞれの方向から受け取るはずの光線（レイ）を、仮想的に逆方向に追跡し、その方向に何が見えるかを判定する方法である。また今回は、レイトレーシングの中でも、パストレーシングと言う手法を用いた。詳しい説明は長くなるのでここでは省く。実際に、pythonを使って、レイトレーシングのコードを自作し、実行した。しかし、5分経っても処理が終わらず、途中で止めてしまった。そこで、何とかして高速化する方法を考えた。

高速化#1：pythonの数値計算ライブラリ「numpy」を用いる

numpyの配列を用いると、繰り返しの処理がfor文で回すより圧倒的に速くなる。 また、pythonは、他の言語と比べて、関数の呼び出しに掛かる時間が長い。そこで、画像をnumpyの配列にし、関数の呼び出し回数を減らした。

```
for スクリーンのそれぞれのyの値について：
  for スクリーンのそれぞれのxの値について:
    raytrace(x,yの方向) ←ピクセル数分呼び出される
```

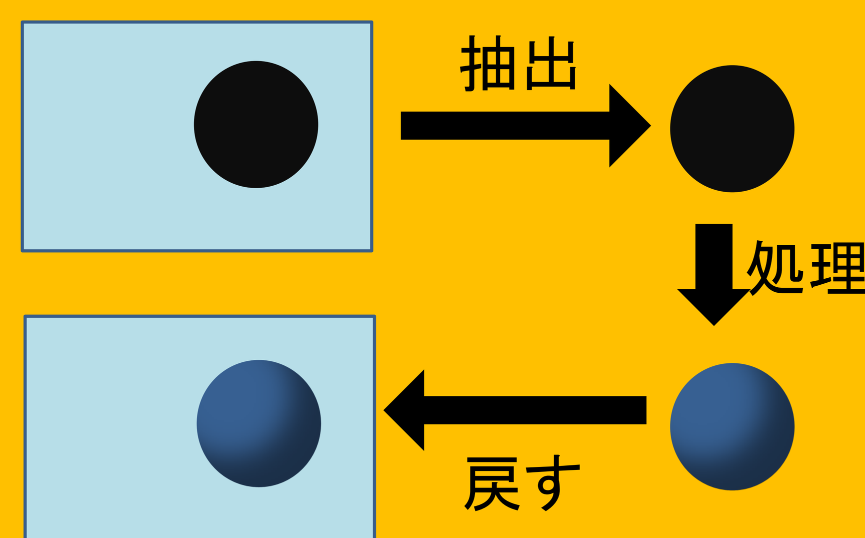


```
raytrace(スクリーン上のすべてのピクセルへの
  方向の配列) ←1回だけ呼び出される
```

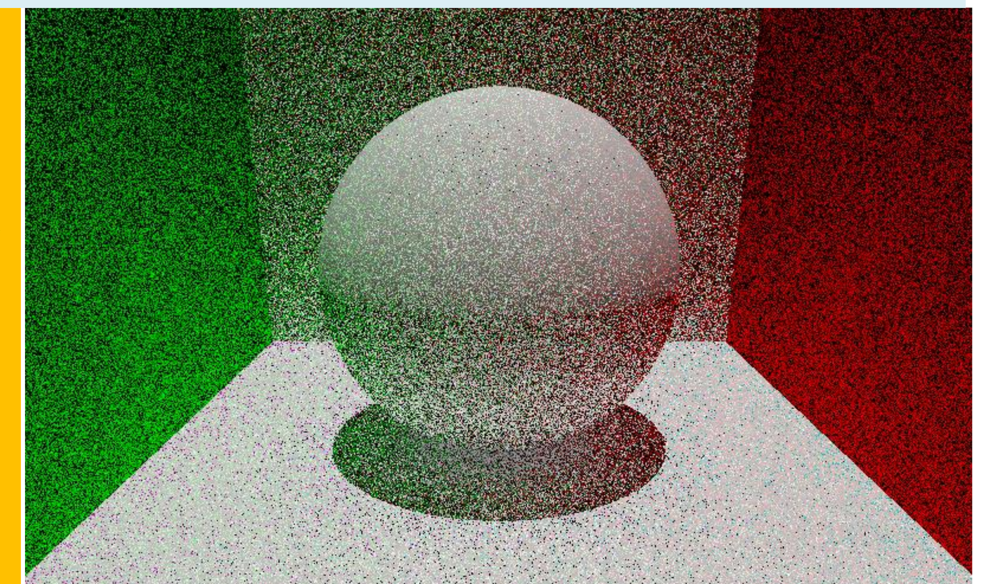
結果：5分でもできなかった処理が、**31.9秒**で終了した。

高速化#2：Rayが物体に衝突したところだけ反射を考える

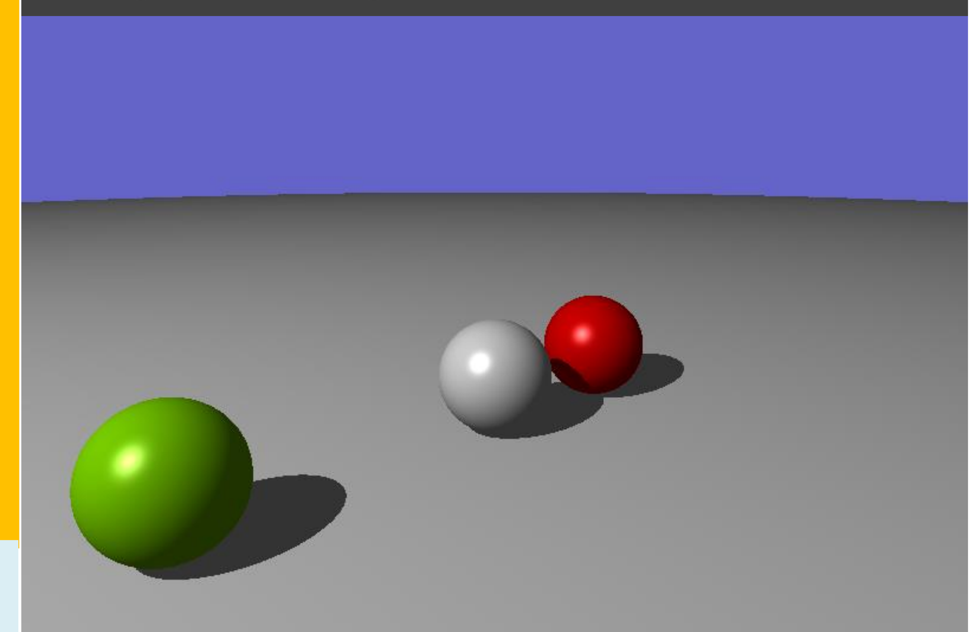
numpyでは、特定の条件の値だけを取り出したり、特定の場所に値をコピーしたりすることができる。それを使って、反射等の計算が、物体に当たったところのみに対して行われるようにした。そうすることで、どこにも当たらなかったところ（あるいは光源に当たったところ）は無駄な処理をせずに済む。



結果：31.9秒かかっていた処理が、**7.02秒**で終了した(Fig.2)。
また、Phongの反射モデルという、二次反射を考えないものでは、**0.892秒**で描画できた(Fig.3)。
ただし、Fig.2で球に壁の色が映り込んでいるのに対し、Fig.3では映り込みはない。



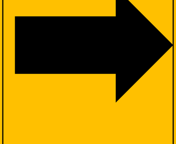
↑Fig.2：パストレ1回, 7.02 -> 5.28sec
↓Fig.3：Phongの反射モデル, 0.892sec



高速化#3 :衝突したところは、オブジェクトごとではなくまとめて反射

今まで、オブジェクトそれぞれに対して、反射等の計算を行っていたのを、できる限りまとめた。それだけで、関数raytraceの呼び出し回数が大幅に減り、速度上昇につながる。

```
for それぞれのオブジェクトに対して：
  ray =そのオブジェクトから反射されたレイ
    raytrace(ray)
```



```
ray = それぞれのオブジェクトから
    反射されたレイの配列
raytrace(ray)
```

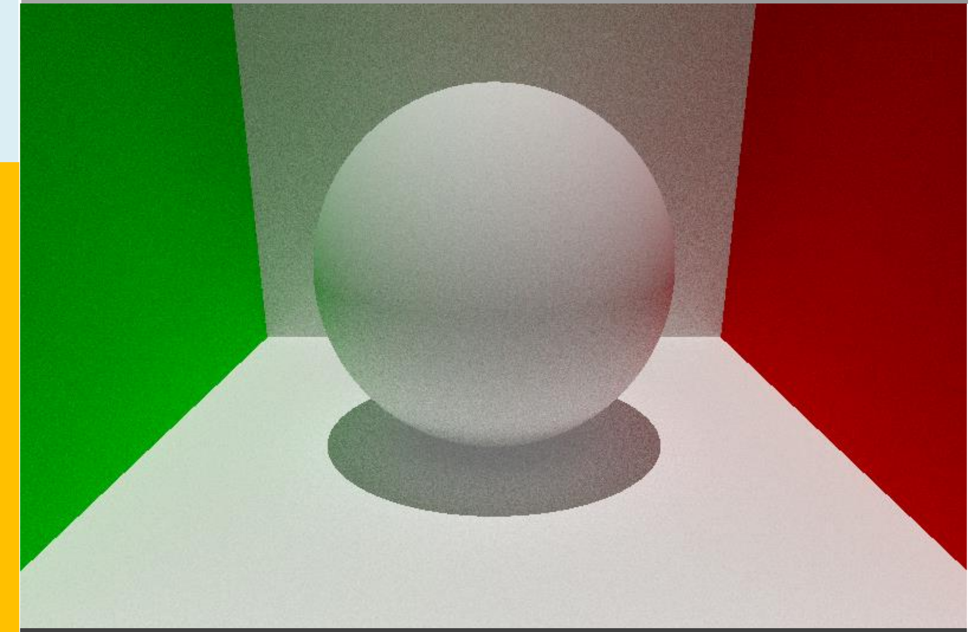


Fig.3：パストレ50回平均値, 269sec

結果：7.02秒かかっていた処理が、**5.28秒**で終了した(Fig.2と同じ画像)。pythonでも、工夫すれば、レンダラーを作れるといえるだろう。ただし、まだまだ、乱数の取り方などに問題があるので、改善していきたい。

補足：画像生成は全て800x600ピクセルであり、サンプルは1ピクセル1回のみ。パストレーシングにはNEEを用いている。

参考文献

「パストレーシング」『memoRANDOM』<https://rayspace.xyz/CG/contents/path_tracing/>(参照2020.03.01)
「A reasonably speedy Python ray-tracer」『excamera』https://excamera.com/sphinx/article/ray.html(参照2020.03.01)