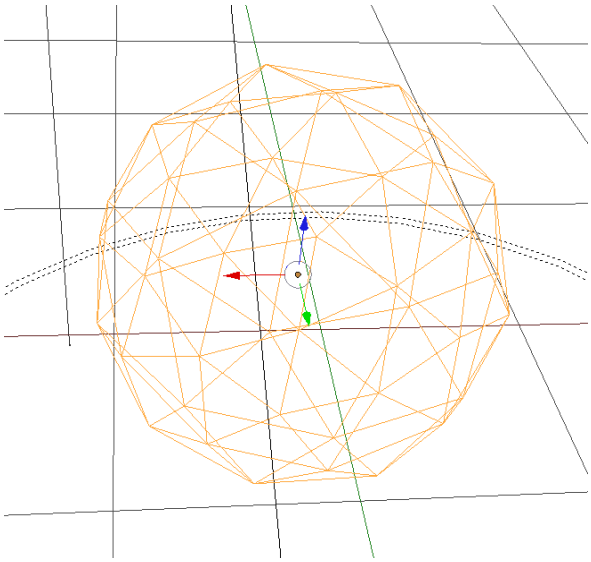
Part3 3DCG に挑戦

1. 3DCG 制作の流れ
2. とりあえず 3D 化してみる
3. 視点について
4. キー操作で移動
5. 3 次元物体を描く
6. 3 次元物体を操作する 3 つの命令
7. ライト
8. 物体の材質

１．3DCG 制作の流れ



(1) モデリング

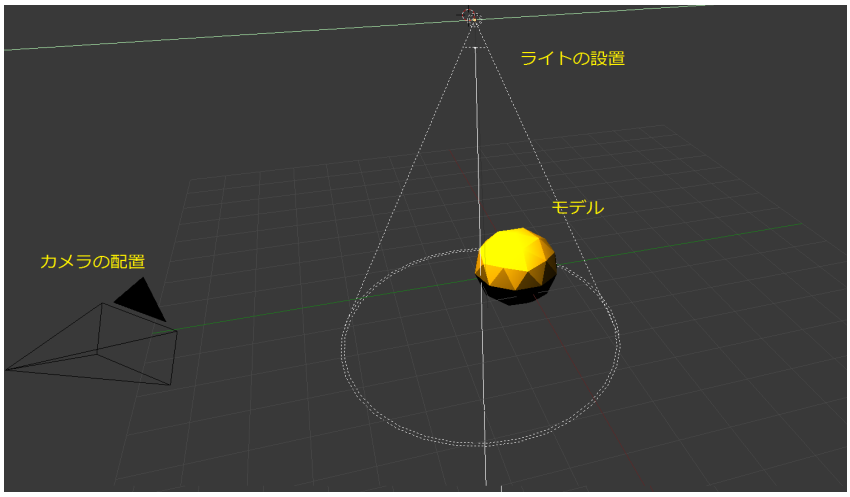
3DCG に登場する物体を作る過程である。

よく用いられる手法では、物体を三角形が集まった表面だけで表現してしまう。つまりハリボテである。

専用のソフトウェアを使って手動で、あるいはプログラムによって自動的に CG にしたい物体のハリボテを作る。右の図は、Blender というソフトで作った多面体のワイヤーフレーム（頂点と辺だけ見えるようにしたもの）である。

形が出来たら、今度は面に色をつけたり、絵を描いたりして、物体の見え方を決める。

こういった作業を「モデリング」という。



(2) 配置

モデリングした物体たちを配置する。物体以外にもカメラやライトも配置する。これがゲームなら、プログラムによって自動的に配置される。

(3) レンダリング

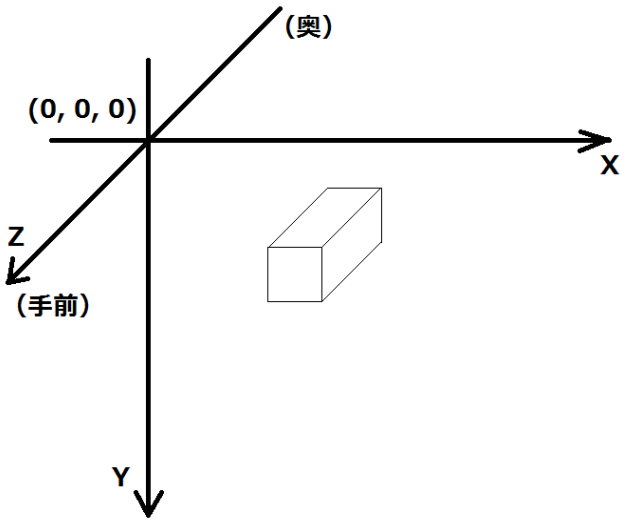
配置したカメラに映る画像をシミュレーションして生成する。

２．とりあえず 3D 化してみる

・Z 軸の導入

二次元から三次元への移行ということで、第三の軸 Z 軸が登場する。

今まで X 軸は画面右方向、Y 軸は画面下方向だったが、Z 軸は画面奥方向から手前方向に伸びる軸である。



・とりあえずプログラム

下のプログラムを書いてみよう。基本的には 2D の時と変わらない。

描かれる図形は下の図のような三角形だが、ん一なんだか本当に 3D なのかよく分からない。

とりあえず、(100, 300, 100)の点が(100, 300)の点よりも左にずれているということは分かる。（手前側にあるからそう見える）

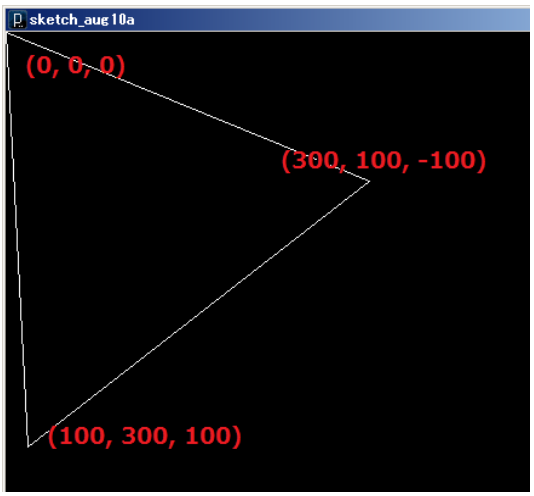
```
void setup(){
  size(600, 400, P3D);
  colorMode(HSB, 100, 100, 100);
  background(0, 0, 0);
}

void draw(){
  background(0, 0, 0);
  stroke(0, 0, 100);
  line(0, 0, 0, 100, 300, 100);
  line(0, 0, 0, 300, 100, -100);
  line(100, 300, 100, 300, 100, -100);
}
```

size命令の第三引数を"P3D"にする

line命令は

(x0, y0, z0, x1, y1, z1)

の形になる

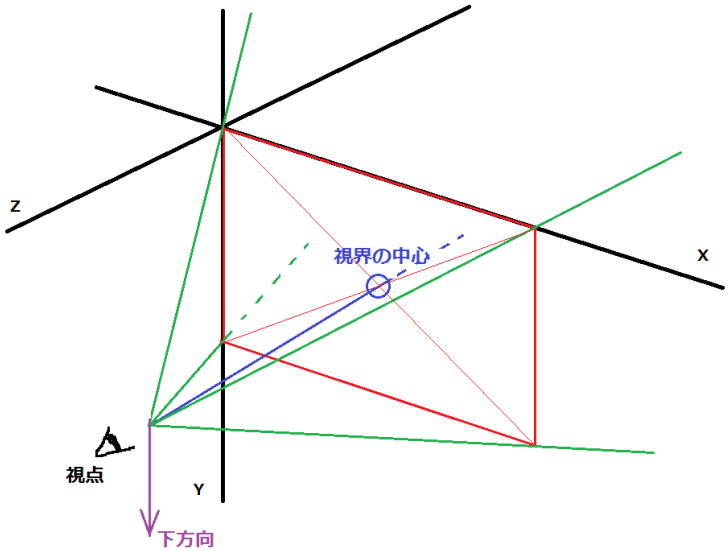


３．視点について

(1)視点とは

視点とは、立体をどこから見ているかという点のことである。

3DCG では視点を移動することで、自分が移動しているような感じにすることができる。



(2)視点の指定

視点を指定するには、

・空間上のどの点から見ているか
　(eyeX, eyeY, eyeZ)
・視界の中心
　(centerX, centerY, centerZ)
・下はどっちか
　(downX, downY, downZ)

の９個の座標が必要である。どれか一つでも欠けると、視点を完全に決定できない（どっちを向いてるかわかんなかったりする）ことがわかるだろうか？

(3)プログラム

下の方向は、視点が(0, 0, 0)にあるときの方向で指定するということに注意。

先ほどの画像に X,Y,Z 軸を書き込んで、視点を左右方向に動かしてみよう。

さっき描いた三角形が、立体っぽく見えることが分かるだろうか？

```
float x;  //視点のX座標をあらわす変数

void setup(){
  size(600, 400, P3D);
  colorMode(HSB, 100, 100, 100);
  background(0, 0, 0);

  x = -200;
}

void draw(){
  camera(x, height/2, 350, x, height/2, 0, 0, 1, 0);  //視点の設定。
  //          視点の位置       視界の中心     下方向(ここではY軸方向が下)
  x += 1.0; //drawが呼ばれるたびに視点を右方向に動かす

  background(0, 0, 0);
  stroke(0, 100, 100);       //X軸は赤
  line(0, 0, 0, 300, 0, 0);

  stroke(33, 100, 100);      //Y軸は緑
  line(0, 0, 0, 0, 300, 0);

  stroke(66, 100, 100);      //Z軸は青
  line(0, 0, 0, 0, 0, 300);

  stroke(0, 0, 100);
  line(0, 0, 0, 100, 300, 100);
  line(0, 0, 0, 300, 100, -100);
  line(100, 300, 100, 300, 100, -100);
}
```

・問題

このプログラムのように、視点の設定は

```
camera(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, downX, downY, downZ)
```

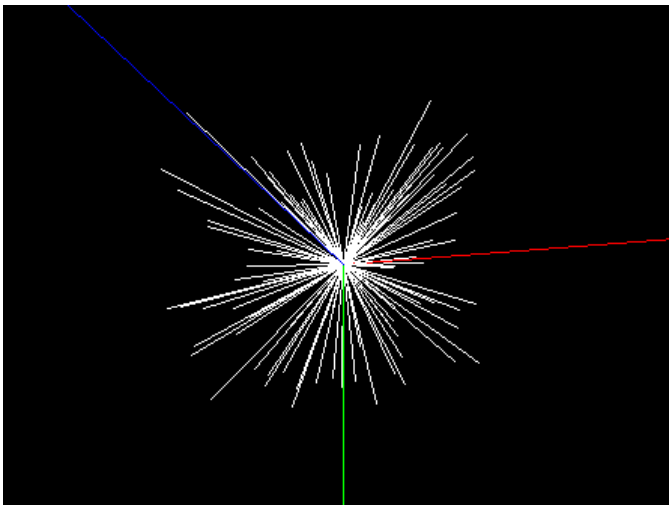
命令で行う。引数を時間変化させることで、視点を動かすことができる。

昨日やった、**「setup 命令は最初に一回だけ呼ばれる」、「draw 命令は一秒間になんども呼ばれる」、「どっちでも使う変数は、その外側に書いておく」**ということに注意しておこう。

・(0, 0, 0)の点が**常に**視界の中心に来るように改良しよう

・今は三角形を若干下から見上げているが、上から見下ろすように改良してみよう

・三角形ではなく次のような火花が散ってるみたいなアニメーションをつくろう。
draw が呼ばれるたびに、(0, 0, 0)からランダムな方向に直線を引けば良い。



・**直線の代わりに**、適当なりんかくと塗りつぶしの色をつけて、

```
sphere(200);
```

と書いてみよう。

・同様に、

```
box(100, 200, 300);
```

と書いてみよう。

４．キー操作で移動

(1) キー操作

キー操作は、mousePressed と同じように、keyPressed という命令を作ることで扱う。

keyPressed 命令の中では、key という名前の変数があり、これには最も最近押されたキーの英語が入る。もし a が押されたときに何かしたいのならば、

```
if(key == 'a'){
        したいこと
}
```

と書けば良い。

(2) プログラム

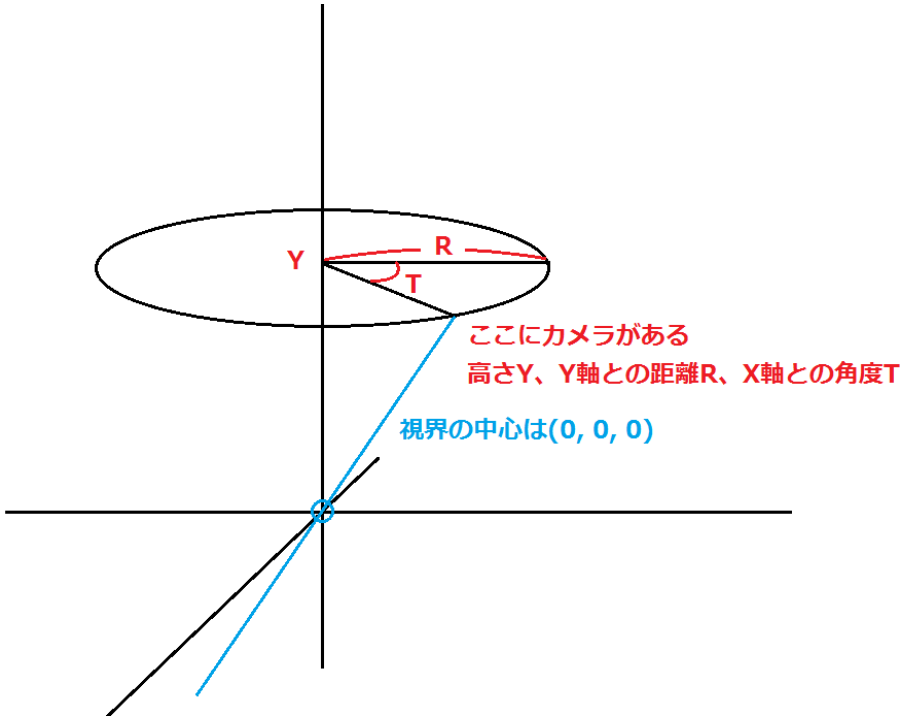
```
float x;  //視点のX座標をあらわす変数

void setup(){
  size(600, 400, P3D);
  colorMode(HSB, 100, 100, 100);
  background(0, 0, 0);

  x = 0;
}

void draw(){
  camera(x, 100, 350, 0, 0, 0, 0, 1, 0);  //視点の設定。
  //         視点の位置      視界の中心     下方向(ここではY軸方向が下)

  background(0, 0, 0);
  stroke(0, 100, 100);       //X軸は赤
  line(0, 0, 0, 300, 0, 0);

  stroke(33, 100, 100);      //Y軸は緑
  line(0, 0, 0, 0, 300, 0);

  stroke(66, 100, 100);      //Z軸は青
  line(0, 0, 0, 0, 0, 300);

  stroke(0, 0, 0);
  fill(50, 100, 100);
  box(100, 200, 300);
}

void keyPressed(){            //これがkeyPressed命令
  if(key == 'a') x -= 3.0;    //押されたキーがaならxから3を引く
  if(key == 'd') x += 3.0;
}
```

a キーを押すと視点（カメラ）は X 軸と逆方向（つまり最初は左方向）に、d キーで X 軸方向に動く。

・問題

(1)ｗキーを押すとＺ軸と逆方向、ｓキーでＺ軸方向、
ｒキーでＹ軸と逆方向（つまり上方向）、ｆキーでＹ軸方向に移動するようにしよう。
もちろん新しい変数ｙとｚを作る必要がある。
このとき、視界の中心は常に（0,0,0)にあるようにすること。

(2)このままだと、ｗを押しても常に前に進むわけじゃないし、ｄを押しても常に右に進むわけじゃないから、やや使いにくい。
ここで、下の図のような位置にカメラを置くことにして、今までのX,Y,Zの代わりにR,T,Yを使うことにする。RはＹ軸からの距離、TはＸ軸との角度、YはＹ座標を表す。
ｗを押すとRが小さくなり、ｓを押すとRが大きくなる。
ｄを押すとTが小さくなり、ａを押すとTが大きくなるようにしてみよう。



sin、cosに慣れていない人のために。
上のカメラの位置は、
　(R*cos(T), Y, R*sin(T))
である。

完成したら、キーボードを使ってしばらくこの世界を動きまわってみよう。ここから何章かは、この移動方法をそのまま使う。

数学が好きな人のために。
(1)の移動方法はデカルト座標、(2)の移動方法は円柱座標という。
もう一つ**極座標**というものがあるので、帰ったら調べてみよう。

５．３次元物体を描く

(1)球
3.の問題で出てきたが、

```
sphere(R);
```

と書くと、(0, 0, 0)を中心とした**半径**Ｒの球が書ける。

(2)直方体
これも今まで使ってきたが、

```
box(X, Y, Z);
```

と書くと、(0, 0, 0)を中心とした、**Ｘ軸方向の辺の長さ**がＸ、Ｙ軸方向がＹ、Ｚ軸方向がＺの直方体が書ける。

(3)直線

```
line(x0, y0, z0, x1, y1, z1);
```

(4)平面図形

```
ellipse、rect、arc
```

などは、Ｘ軸とＹ軸によって張られる、Z=0 の平面上に描かれる。

(5)三角形
好きな場所に三角形を描く。

```
beginShape();
vertex(x0, y0, z0);
vertex(x1, y1, z1);
vertex(x2, y2, z2);
endShape();
```

こう書くと、この３点を結ぶような三角形が描かれる。

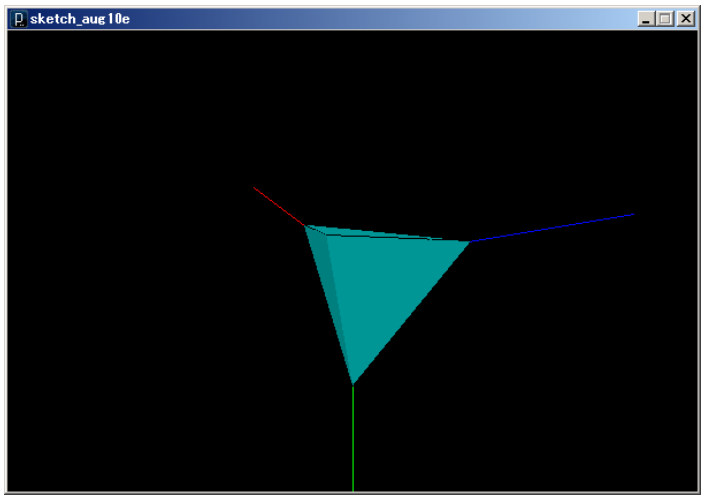
・プログラム

これは 4. の問題(2)の解答にもなっている。
赤枠一つ目の lights()は、勝手に光を当ててくれる命令である。これを書くと、光と影が描かれるので少し綺麗に見える。
赤枠二つ目は、今まで使ってきた rect が三次元上でどうなるかを調べるためのものである。
さっそくプログラムにして、見てみよう。

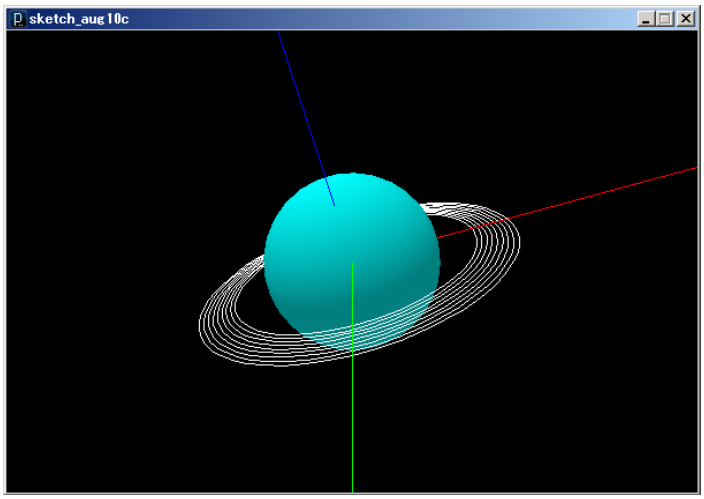
```
float y;  //視点のX座標をあらわす変数
float r;  //視点のX座標をあらわす変数
float t;  //視点のX座標をあらわす変数

void setup(){
  size(600, 400, P3D);

  colorMode(HSB, 100, 100, 100);
  background(0, 0, 0);

  y = 0;
  r = 500;
  t = 0;
}

void draw(){
  lights(); //魔法の呪文lights()を唱えると影をつけてくれる
  camera(r * cos(t), y, r * sin(t), 0, 0, 0, 0, 1, 0);  //視点の設定。
  //         視点の位置       視界の中心     下方向(ここではY軸方向が下)

  background(0, 0, 0);
  stroke(0, 100, 100);       //X軸は赤
  line(0, 0, 0, 300, 0, 0);

  stroke(33, 100, 100);      //Y軸は緑
  line(0, 0, 0, 0, 300, 0);

  stroke(66, 100, 100);      //Z軸は青
  line(0, 0, 0, 0, 0, 300);

  for(int i=-10; i<=10; i++){
    stroke(0, 0, 0);
    fill((i+10)*5, 100, 100, 80);
    rect(i*10, -100, 8, 200);
  }
}

void keyPressed(){            //これがkeyPressed命令
  if(key == 'a') t += 0.1;    //押されたキーがaならxから3を引く
  if(key == 'd') t -= 0.1;
  if(key == 'w') r -= 3.0;
  if(key == 's') r += 3.0;
  if(key == 'r') y -= 3.0;
  if(key == 'f') y += 3.0;
}
```

・問題

・三角形
三角形を４枚組み合わせて、下の絵のような三角錐を作ってみよう。
変更するのは、二つ目の赤枠の中だけでよい。



・土星
sphere と ellipse を使って土星を描いてみよう。



６．３次元物体を操作する３つの命令

(1) 平行移動

```
translate(x, y, z);
```

X 軸方向に x、Y 軸方向に y、Z 軸方向に z 平行移動する。

(2) 回転

```
rotateX(t);
rotateY(t);
rotateZ(t);
```

それぞれ、X 軸、Y 軸、Z 軸まわりに時計回りに角度 t だけ回転する。

(3) 拡大縮小

```
scale(s);
```

図形を s 倍に拡大する。

(4) 効果を打ち消す

```
pushMatrix();
・・・
popMatrix();
```

間に書かれた命令は、pushMatrix();より前の translate などの影響を受けない。また、間に translate などを書くと、popMatrix();より後の命令には影響を与えない。

・注意

・これらの命令は、それより後ろにある全ての図形を書く命令に対して適用される。

・同じ命令を続けて書くと、**重ねがけされる**。

たとえば scale(2.0)の後に scale(1.5)を書くと、scale(3.0)と同じ意味になる。

・操作する順序

たとえば

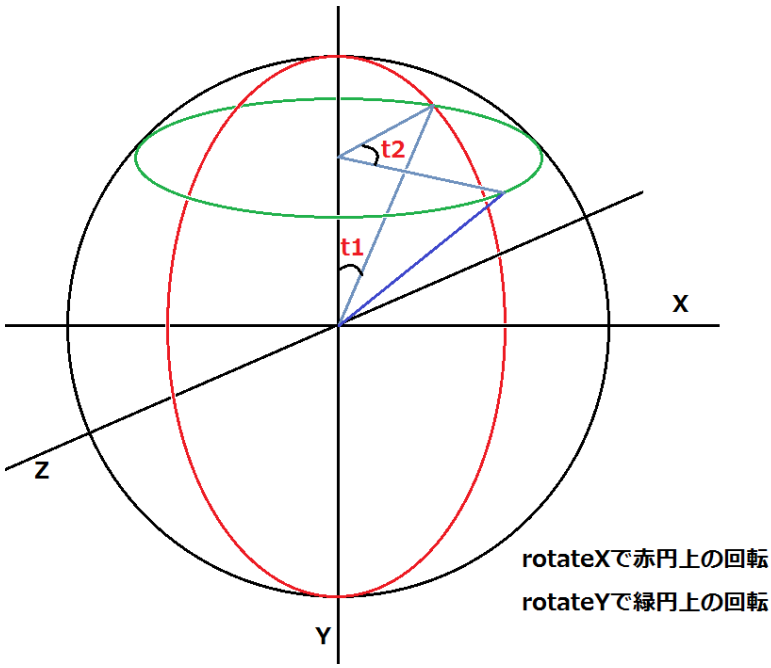
```
rotateY(t);
translate(x, y, z);
```

の順番に書くと、**先に平行移動して、後に回転される**。直感と逆なので注意。

基本的に、**下から順番に効果が適用される**。

これらをふまえると、ある一つの物体を移動するときは、次のようにすると良い。

```
pushMatrix();
translate(x, y, z);
rotateY(t2);
rotateX(t1);

scale(s);
立体を描く命令
popMatrix();
```



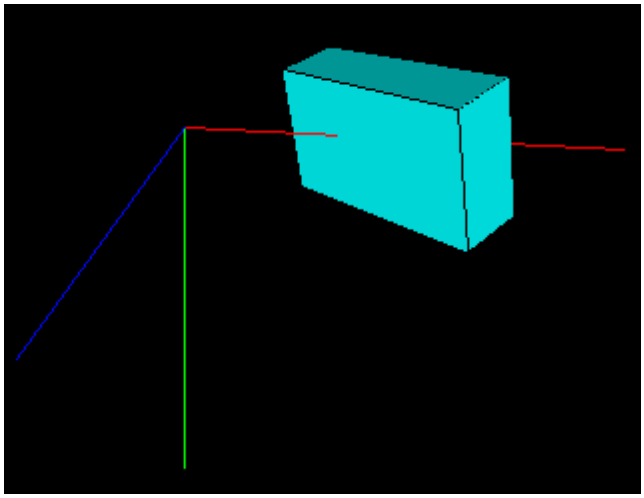
・プログラム

5. のプログラムの、二つ目の赤線の中を次に書き換えてみよう。

```
pushMatrix();
translate(150, 0, 0);
rotateY(135 * PI / 180);
rotateX(15 * PI / 180);
scale(0.5);
box(300, 200, 100);
popMatrix();
```

辺の長さが 300, 200, 100 の直方体は、最初 (0, 0, 0) を中心として存在する。
まず scale(0.5) で辺の長さが 150, 100, 50 の直方体に縮小される。
次に、rotateX で、X 軸（赤）回りに 15 度回転する。（前頁の地球儀みたいなの参照）
次に、rotateY で、Y 軸（緑）周りに 135 度回転する。
最後に、X 軸方向に 150 平行移動する。

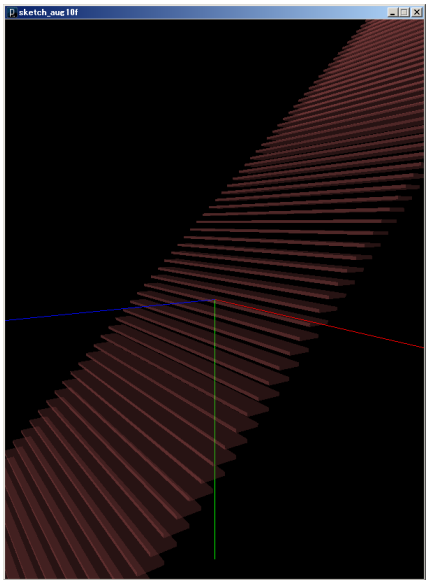
その結果が下の図である。



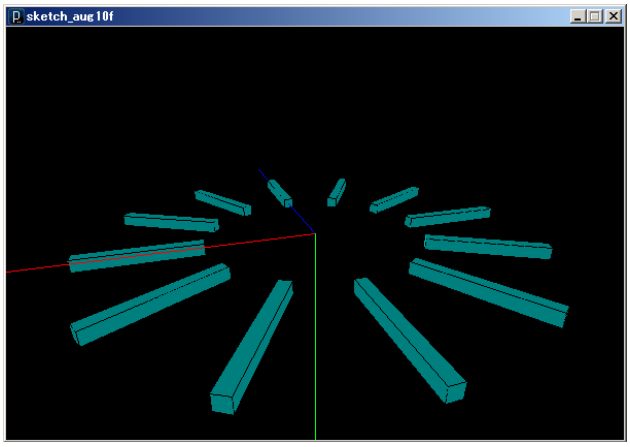
・問題

次の CG を描いてみよう。

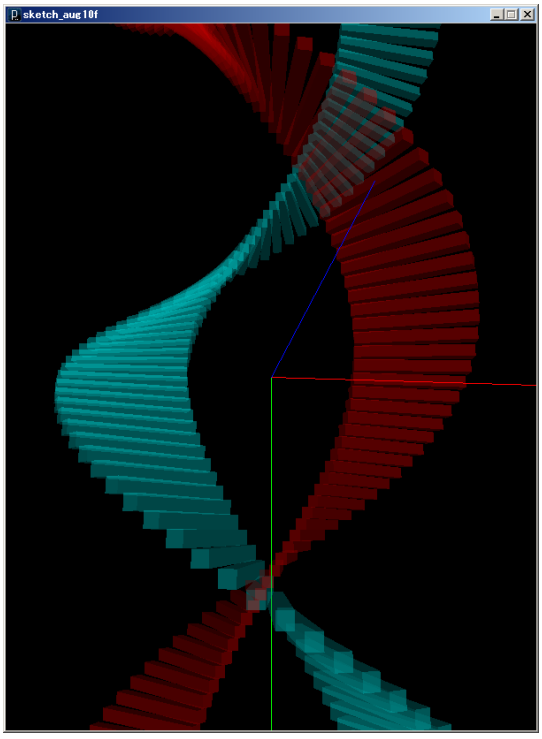
半透明な階段



時計盤



DNA



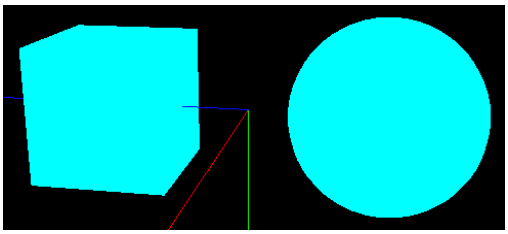
７．ライト

Processing で使えるライトは、主に次の 3 種類である。

(1)ambientLight(H, S, B);

「環境光」

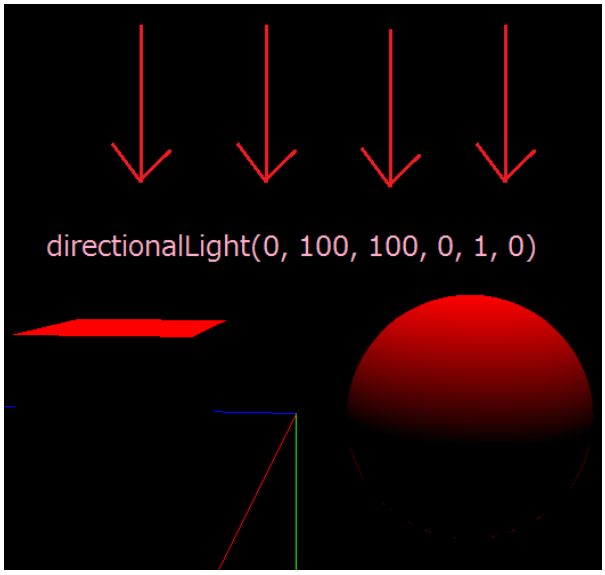
物体全体を同じように色 H, S, B で照らす。



(2)directionalLight(H, S, B, X, Y, Z);

「指向光」

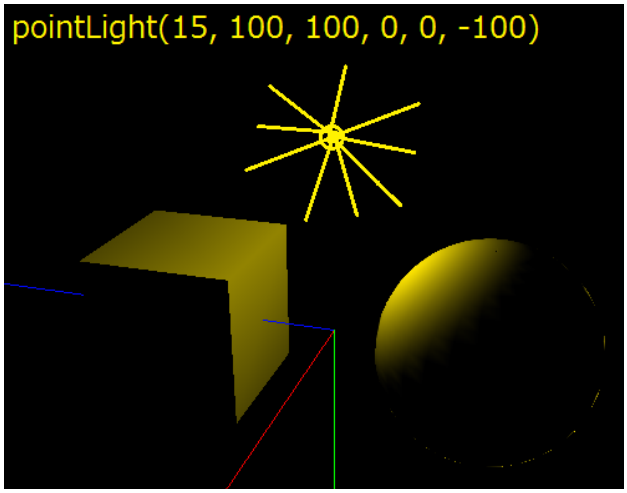
X, Y, Z 方向に色 H, S, B で照らす。（平行光線）



(3)pointLight(H, S, B, X, Y, Z);

「点光源光」

(X, Y, Z)においた点光源から色 H, S, B で照らす。



・プログラム

ライトは、5.のプログラムの一つ目の赤枠で、light();と書かれている文の代わりに書く。
次のようなプログラムを書いてみよう。

・一つ目の赤枠

```
ambientLight(0, 0, 30);   //全体を弱く白で照らす
pointLight(15, 100, 100, -(mouseX-width/2), (mouseY-height/2), 0);  //マウスで操作できる点光源（X, Y面内）
```

・二つ目の赤枠

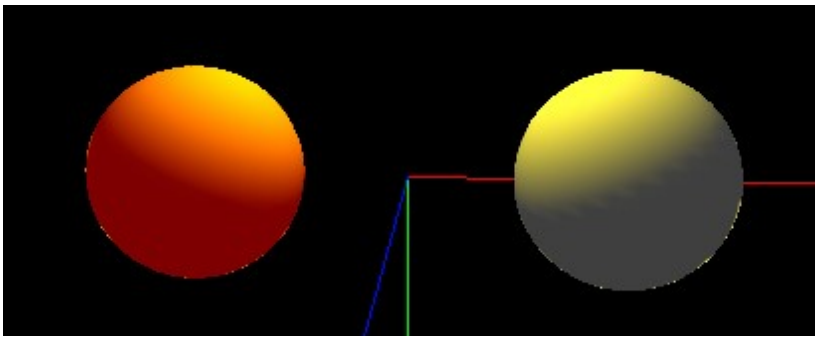
```
noStroke();
fill(0, 0, 100);

pushMatrix();
translate(0, 0, -100);
sphere(70);    //球
popMatrix();

pushMatrix();
translate(0, 0, 100);
box(100, 100, 100);      //立方体
popMatrix();

fill(15, 100, 100);
pushMatrix();
translate(-(mouseX-width/2), (mouseY-height/2), 0);
sphere(10);    //光源の場所がわかるように、球を描く
popMatrix();
```

８．物体の材質

物体の色は、光源から出る光の色と、観察者と物体の位置関係、さらに反射率等の物体固有の性質によって決定されます。

(1)ambient(H, S, B)
　環境光（ambientLight）に対する照らされ方。環境光の色のうち、ambient で指定した色だけが見える。

(2)emissive(H, S, B)
　発光。ただし、あくまでこの物体の見え方なので、他の物体を照らすにはここに点光源を置く必要がある。



```
ambientLight(0, 0, 50);   //全体を弱く白で照らす
pointLight(15, 100, 100, 0, -200, 0);
```

```
pushMatrix();
ambient(0, 0, 50);        //環境光対して白色に色づく
emissive(0, 0, 0);        //発光なし
translate(100, 0, 0);
sphere(50);
popMatrix();

pushMatrix();
ambient(0, 0, 0);         //環境光を感じない
emissive(0, 100, 50);     //赤色に発光
translate(-100, 0, 0);
sphere(50);
popMatrix();
```

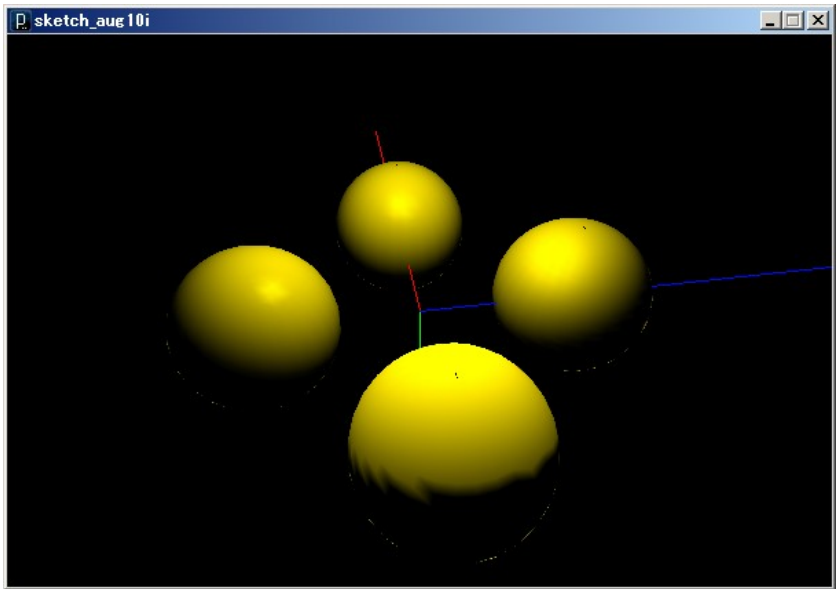
(3) 鏡面反射

金属やツルツルのプラスチックのような光沢を出す方法。

・directionalLight か pointLight の前に、
　　lightSpecular(H, S, B)
と書く。こうすると、ここで指定した色が光沢から反射されるようになる。

・さらに、ambient、emissive を指定しているところで、
　　specular(H, S, B);
　　shininess(sh);
を書く。光源からの色のうち、specular で指定した色が光沢から反射される。sh が高いほど、鋭い反射が起きる。

下の図で、下:sh=0, 右:sh=10, 上:sh=50, 左:sh=100

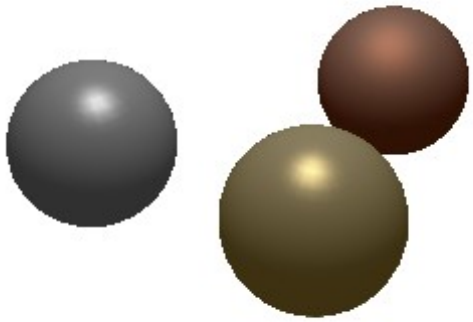



```
lightSpecular(15, 100, 100);
ambientLight(0, 0, 50);   //全体を弱く白で照らす
pointLight(15, 100, 100, 0, -200, 0);
```

```
pushMatrix();
ambient(0, 0, 0);
emissive(0, 0, 0);
specular(0, 0, 30);
shininess(10.0);
translate(0, 0, 100);
sphere(50);
popMatrix();
```

・問題

金銀銅の球を作りましょう。



金

ambient(12, 69, 24)
specular(12, 41, 62)
shininess(52.0)

銀

ambient(0, 0, 19)
specular(0, 0, 50)
shininess(51.2)

銅

ambient(5, 88, 19)
specular(5, 66, 25)
shininess(12.8)