K会2012年度冬季講習　情報講座

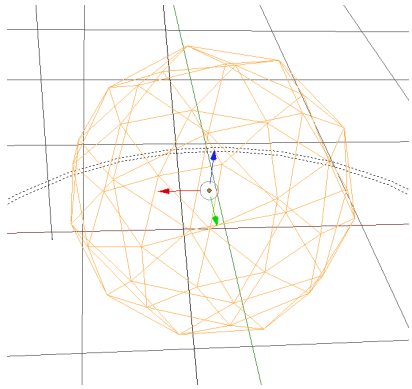
リピーターコース　Part2　3DCGを描いてみよう

目次

2-1. 3DCG 制作の流れ

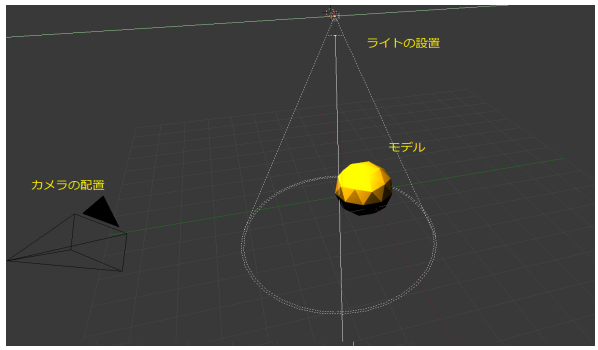
（１）モデリング

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

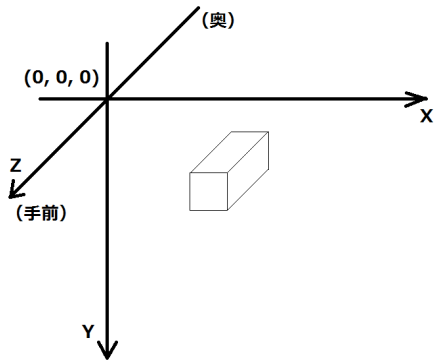
仮想的な 3 次元空間にモデリングした物体たちを配置する。
物体の他に、カメラとライトも設置する。
例えば 3D シューティングゲームなら、カメラは主人公の機体である。



（３）レンダリング

配置されたカメラに映る画像をシミュレーションして生成する。
具体的には、スクリーンのこの位置にはどの物体が映るか、その面にはどのくらい光が当たり、どのくらいの明るさになるか、といった計算がされる。

2-2. とりあえず 3D 化してみる



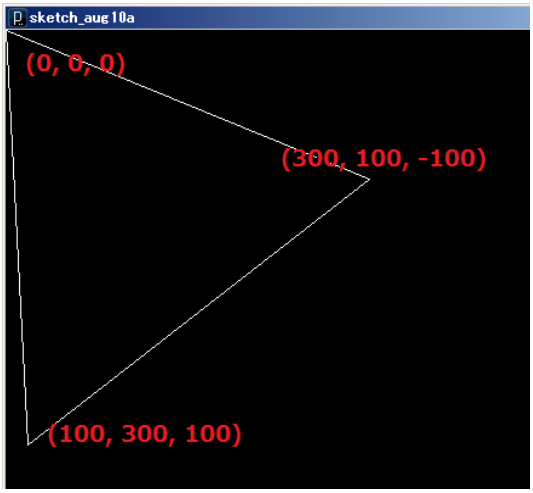
（１）Z 軸の導入
今までの X 軸（画面右方向）、Y 軸（画面下方向）に加えて、第三の軸 Z 軸が登場する。これは画面奥方向から手前方向に伸びる軸である。

（２）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)の点が二次元での(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）

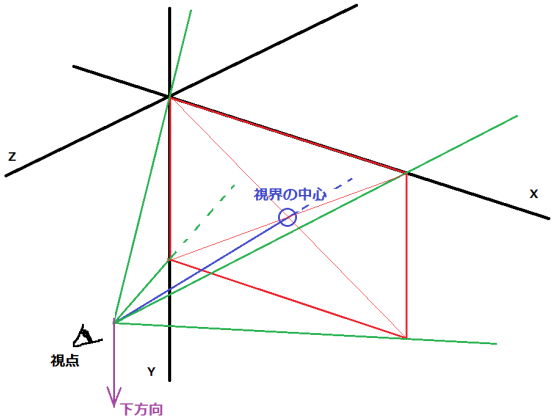
```
void setup(){
  size(600, 400, P3D);
  colorMode(HSB, 100, 100, 100);
  background(0, 0, 0);
}

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



2-3. カメラ



（１）カメラ

2DCG のときは、画面には x が 0 から width, y が 0 から height までの範囲が必ず表示されていた。3DCG では、3 次元空間を画面という 2 次元空間に投影して表示するので、「どこから見た図を表示するか」が問題となる。この位置を指定するのが、次の camera 命令である。

　camera 命令では、カメラの位置と方向を指定するために 9 つの引数をとる。

カメラ

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

・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)

（２）プログラム

次のプログラムは、さっきのプログラムに camera 命令を追加したものである。とはいえ表示されるものは変わらない。なぜなら、ここで引数に与えられている 9 つの数は、デフォルトの値（camera 命令を書かなかった時の値）と同じだからである。

　カメラの位置を、マウスの x 座標と同じになるようにせよ。マウスを動かすと物体が立体的に動くのが分かるだろう。

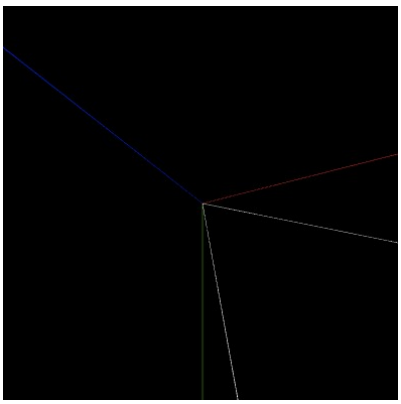
```
void setup(){
  size(400, 400, P3D);
  colorMode(HSB, 100);
  background(0);
}

void draw(){
  background(0);
  camera(mouseX, height/2, 350, width/2, height/2, 0, 0, 1, 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);
}
```

（３）課題

・X 軸、Y 軸、Z 軸をそれぞれ赤、緑、青の直線で描き込もう。X 軸なら(0, 0, 0)から(1000, 0, 0)などに線を引っ張れば良い。

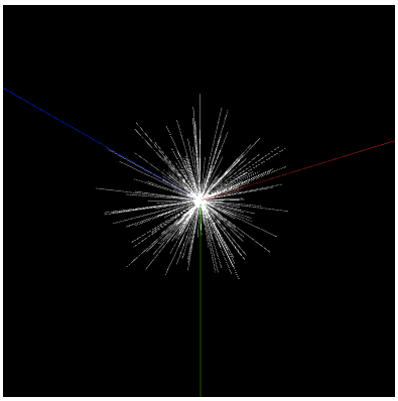
・視界の中心が(0, 0, 0)になるようにせよ。

　右図は表示される画像の例



これからのプログラムでは、方向を見失わないためにこの 3 軸は常に表示させるようにする。

・三角形の代わりに直線をたくさん(0, 0, 0)からランダムな方向に引くようなプログラムを書け。直線の引く先は、配列 x, y, z を用意してそこで指定せよ。



・立体図形の表示

　　sphere(100);　（半径 100 の球を(0, 0, 0)に描く）

　と書いてみよ。次に、

　　lights();　　（Z 軸方向からライトを照射）

　と書いてみよ。Sphere の代わりに、

　　box(100, 200, 300);

　　（辺の長さが 100, 200, 300 の直方体を(0, 0, 0)中心に描く）

　も試してみよ。

　これらの立体図形も、fill と stroke 命令で色を指定できる。

2-4. キー操作で移動

（１）キー操作の方法

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）次のプログラムを書こう。

このプログラムでは、カメラの x 座標が、ブロックの外側で定義された変数 x によって決まる。変数 x は、キー'a'が押された時に 10 ずつ減り、キー'd'が押された時に 10 ずつ増える。

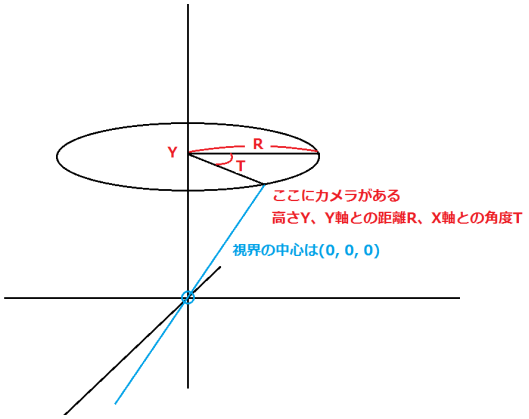
キーを押しっぱなしにすると、視点が連続的に動くのが分かるだろう。

また、マウスで視界の中心位置を調整できるようにしてある。

```
float x;

void setup(){
  size(400, 400, P3D);
  colorMode(RGB, 100);
  background(0);
}

void draw(){
  lights();
  background(0);
  camera(x, height/2, 350, mouseX-width/2, mouseY-height/2, 0, 0, 1, 0);

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

  stroke(100, 0, 0);
  line(0, 0, 0, 1000, 0, 0);
  stroke(0, 100, 0);
  line(0, 0, 0, 0, 1000, 0);
  stroke(0, 0, 100);
  line(0, 0, 0, 0, 0, 1000);
}

void keyPressed(){
  if(key == 'a') x -= 10.0;
  if(key == 'd') x += 10.0;
}
```

（３）課題
(i) w キーを押すとカメラの Z 座標が減り、増える、r キーで Y 座標が減り、f キーで増えるようにしよう。もちろん新しい変数 y と z を作る必要がある。

(ii) この移動方法もまあまあ使えるのだが、もっと直感的に移動できる方法がある。それは、カメラを下の図のような円柱状の位置に置くことである。
　今までの変数 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。
上の図でのカメラの位置は、
　　(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。

2-5. 基本図形

2-3 の問題で sphere と box 命令を使ってもらったが、これらは 3DCG を作る上での基本パーツとなる図形のひとつである。Processing には次のような基本図形がある。

基本図形

```
sphere(R)
   半径R, 中心(0, 0, 0)の球を描く
box(X, Y, Z)
   辺の長さが(X, Y, Z), 中心が(0, 0, 0)の直方体を描く
line(X1, Y1, Z1, X2, Y2, Z2)
   点(X1, Y1, Z1)から点(X2, Y2, Z2)への直線

円と長方形の描き方
   2DCGの時と同じようにellipse, rectが使える。
   これらはZ=0の平面上で描かれる

三角形の描き方
   beginShape();
   vertex(x0, y0, z0);
   vertex(x1, y1, z1);
   vertex(x2, y2, z2);
   endShape();
```

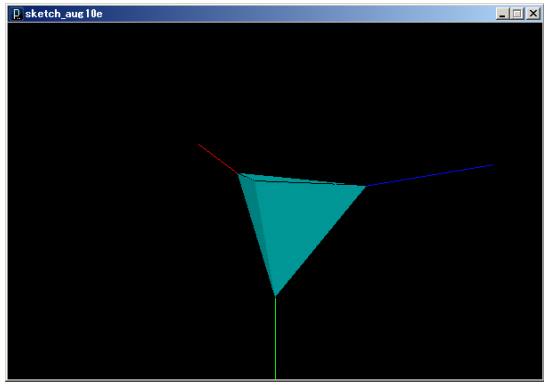
ライトは後で詳しく扱うが、2-3 でも出てきた次の命令はこの段階から常に使うことにする。

自動ライト

```
lights()
  Z軸正方向から、負方向へと向かう平行光線を照射する。
```

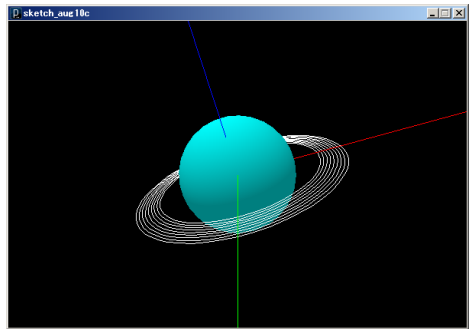
（１）三角錐

三角形を４枚描いて、三角錐を作れ。



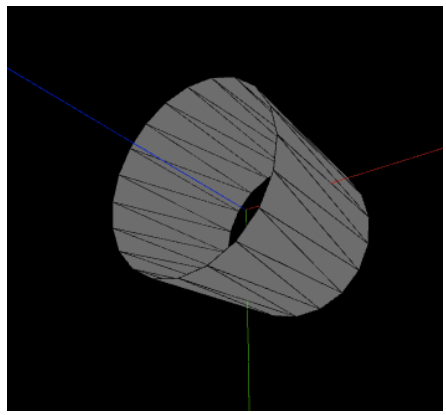
（２）土星

sphere 命令と ellipse 命令で土星を描け。



（３）円柱

三角形を組み合わせて円柱を描け。



2-6. 立体図形を動かす 3 つの命令

sphere と box は(0, 0, 0)中心のものしか描かれないので、このままでは使いものにならない。好きな位置にこれらを描くためには、(0, 0, 0)に書いたものを平行移動の命令を使って目的の位置に移動させる。

立体図形を動かす命令

translate(x, y, z)
　X軸方向にx, Y軸方向にy, Z軸方向にz平行移動させる。

rotateX(t)
rotateY(t)
rotateZ(t)
　それぞれX軸、Y軸、Z軸まわりに角度tだけ回転させる。

scale(s)
　図形をs倍相似拡大する。

※これらの命令は、後に書いたものから順に適応される

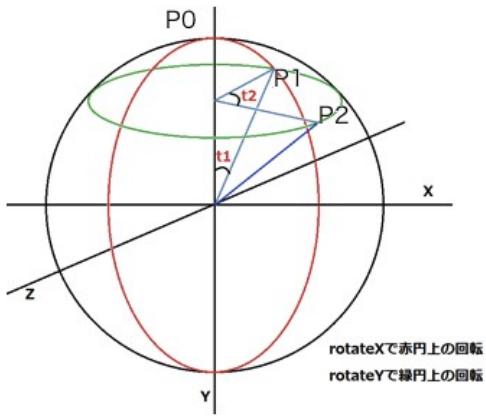
※通常、次の…の部分に記述する
pushMatrix()
……
popMatrix()
　移動や回転の処理は、この２つの命令の間の部分に書く

ある(0, 0, 0)に描かれた図形を、s 倍に拡大し、Y 軸となす角が t1, Z 軸となす角が t2、中心が(x, y, z)になるように移動させるには、次のように書けばいい。

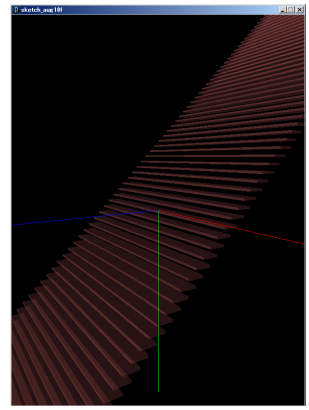
典型的な移動命令の使い方

```
pushMatrix();
translate(x, y, z);
rotateY(t2);
rotateX(t1);
scale(s);
  (立体を描く命令)
popMatrix();
```

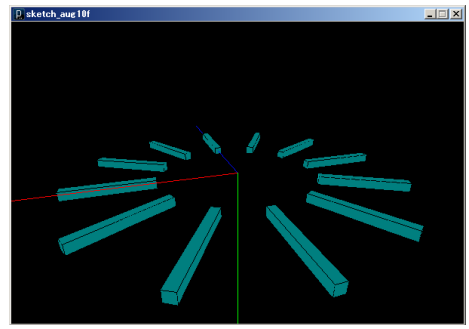
回転処理はなかなかわかりづらいが、まず rotateX(t1)により、右図の P0 は P1 に回転移動し、次に rotateY(t2)により P2 まで移動する。



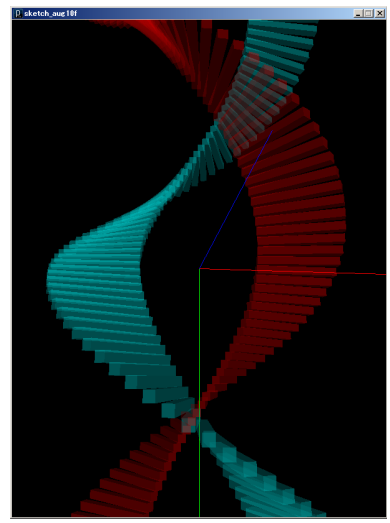
（１）階段
右の階段を描くプログラムを作れ。
for ループ内で box を作り、translate で移動する。
色を半透明にすると綺麗。



（２）時計盤
右の時計盤を描くプログラムを作れ。
rotateY と translate を組み合わせる。



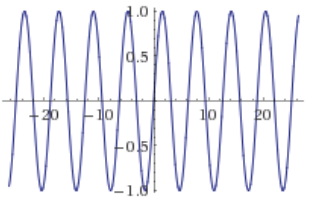
（３）DNA



2-7. 立体視

3DCG は、あくまで 3 次元空間の物体を 2 次元のスクリーンに投影した図を表示する技術である。しかし、目の錯覚を使うことによって、あたかも本当に 3 次元物体があるように見せることができる。これを立体視という。

（１）小刻み振動による立体視
カメラの位置を小刻みに振動するようにすると、目の錯覚によって立体感が増す。



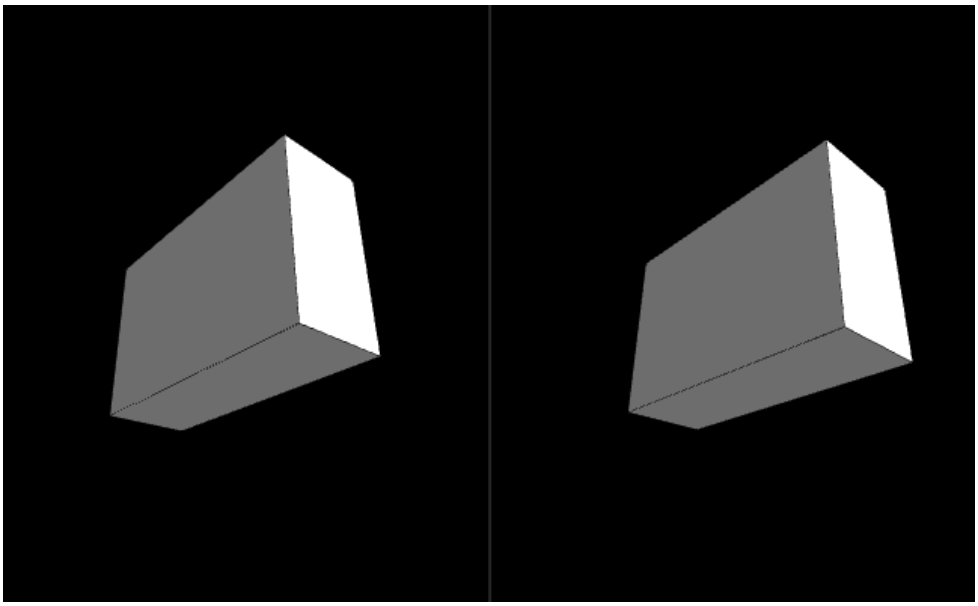
　この振動は、三角関数を使うと簡単にかける。右のグラフは横軸に x, 縦軸に sin(x)を書いたもので、x が増えると sin(x)は振動する。これより、カメラの位置を sin(x)に比例する量だけ動かして、毎ステップ x の値を変えてやれば良い。
　これをさっきの問題のプログラムなどに実装して、本当に立体的になるか確認せよ。

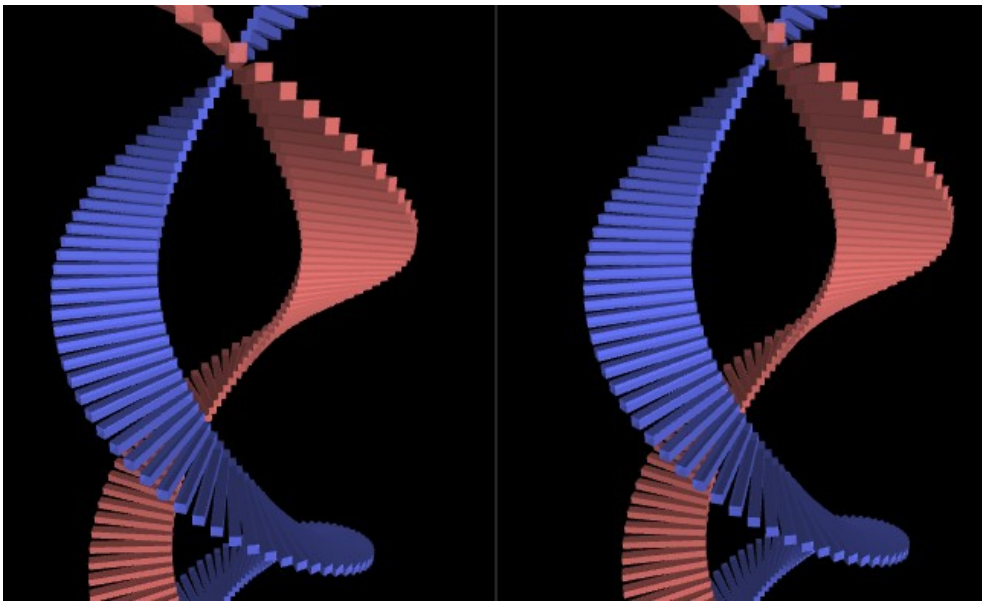
（２）交差法による立体視
人間は、左右の目から見える画像のズレ（視差という）を元に立体感を感じている。つまり、プログラムでそれを再現してやれば、立体視ができるはずである。
　「交差法」は、横に並んだ二枚の画像で立体視をする方法である。

　プログラムを書く前に、まずは目の練習をしよう。

　左の画像を右目が、右の画像を左目が見るようにしたい。そのためには、視点を紙の上ではなく、紙よりも少し手前を見るようにする。紙の前に指をおいてそこを見るようにし、ふたつの図形が重なるところで重なった図形を眺めると、立体的に見えるはずである。

まずは紙の上で練習してみよう！
直方体が飛び出てみえるはずである。





（３）PGraphics

直方体を描画したプログラム。

```
float y;
float r;
float t;

PGraphics pg1, pg2;

void setup(){
  size(650, 400, P3D);

  pg1 = createGraphics(300, 400, P3D);
  pg2 = createGraphics(300, 400, P3D);

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

void draw(){
  pg1.beginDraw();
    pg1.colorMode(RGB, 100);
    pg1.lights();
    pg1.camera(r * cos(t-0.05), y, r * sin(t-0.05), 0, 0, 0, 0, 1, 0);
    pg1.background(0, 0, 0);

    pg1.stroke(0);
    pg1.fill(100);
    pg1.box(100, 200, 300);
  pg1.endDraw();


  pg2.beginDraw();
    pg2.colorMode(RGB, 100);
    pg2.lights();
    pg2.camera(r * cos(t+0.05), y, r * sin(t+0.05), 0, 0, 0, 0, 1, 0);
    pg2.background(0, 0, 0);

    pg2.stroke(0);
    pg2.fill(100);
    pg2.box(100, 200, 300);
  pg2.endDraw();

  image(pg1, 0, 0);
  image(pg2, 350, 0);

  stroke(100);
  line(325, 0, 325, 400);
}

void keyPressed(){
  if(key == 'a') t += 0.1;
  if(key == 'd') t -= 0.1;
  if(key == 'w') r -= 10.0;
  if(key == 's') r += 10.0;
  if(key == 'r') y -= 10.0;
  if(key == 'f') y += 10.0;
}
```