コンピュータ基礎演習
追加課題(1) 3Dプログラミング

理工学部 情報科学科 隅田 康明
sumida@ip.kyusan-u.ac.jp

追加課題について

- 本来の授業内容から削った内容をいくつか出題
  - 14回授業が13回になったことの補填
  - 遠隔授業により削減した内容の提示

- 制作課題まで提出して、余裕があれば取り組みましょう
  - あくまで、やる気のある学生向けの内容

- いくつか出題する追加課題のうち、1つを提出すれば加点
  - 追加課題の提出で最大+5点
  - ただし、合計の評点は100点を上限とする

想定動作端末

- 追加課題は大学のパソコン教室等での動作を想定
  - 所持端末によっては動作しないものもあるので、動作するものを選んで作業すること

3Dプログラミング

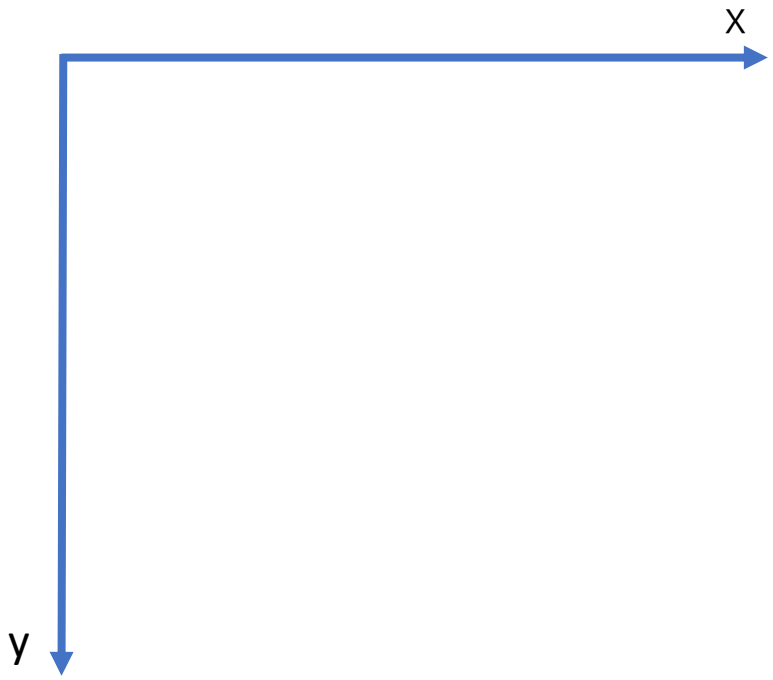
3Dプログラミング

- これまでのプログラムは、縦と横の2軸（２D)
  - 平面状に平面の図形を描画

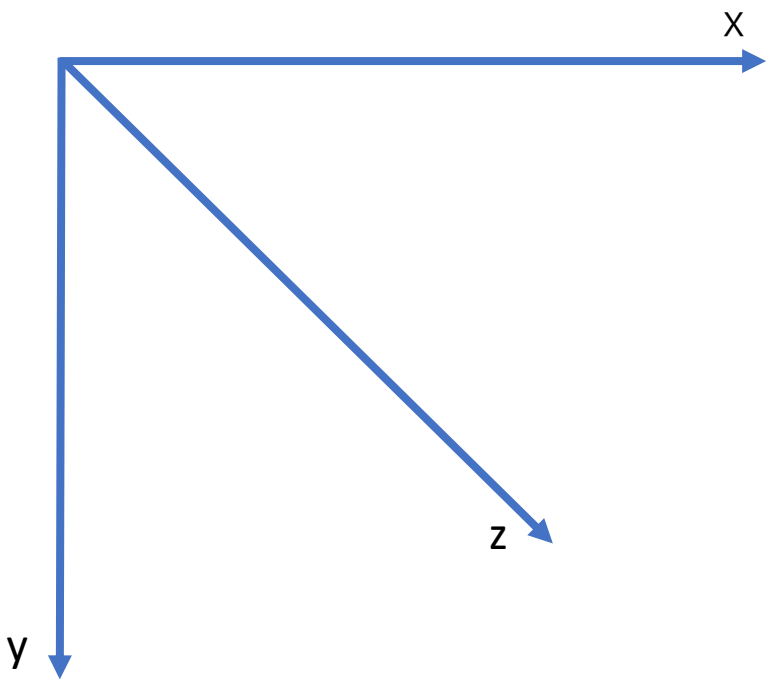
- 3次元では、縦と横に加えて奥行きが追加
  - 最終的に描画するのは2Dディスプレイ
  - 奥行きは擬似的に表現する
    - 奥に行くほど小さくなるなど

3D空間での座標系

2D空間ではXとYの2軸



3D空間ではZ軸が追加

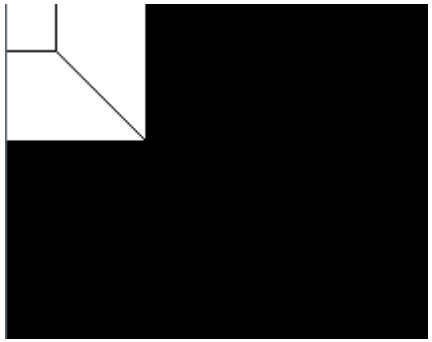


zが大きくなると手前に、小さくなると奥に

3D空間に図形を配置

- 3D空間を作るには、
  size(幅,高さ,P3D); と書く

```
void setup(){
  size(400,400,P3D);
}
void draw(){
  background(0);
  box(100);
}
```



まだ見た目は2Dと同じ

Processingに用意されている３D図形

- 立方体（キューブ）を描くメソッド
  - box(size);
    - 原点(0,0,0)を中心に立方体を設置
  - box(幅, 高さ, 奥行き);
    - 原点を中心に指定したサイズの直方体を設置
- 球（スフィア）を描くメソッド
  - sphere(半径);
    - 原点を中心に指定半径の球を描く

boxとsphere

- Processingに用意されている3D図形はこの２つ
  - どちらも描画する座標を指定できない
    ⇨　座標変換で動かすしかない
  - Processingでの3D図形描画は

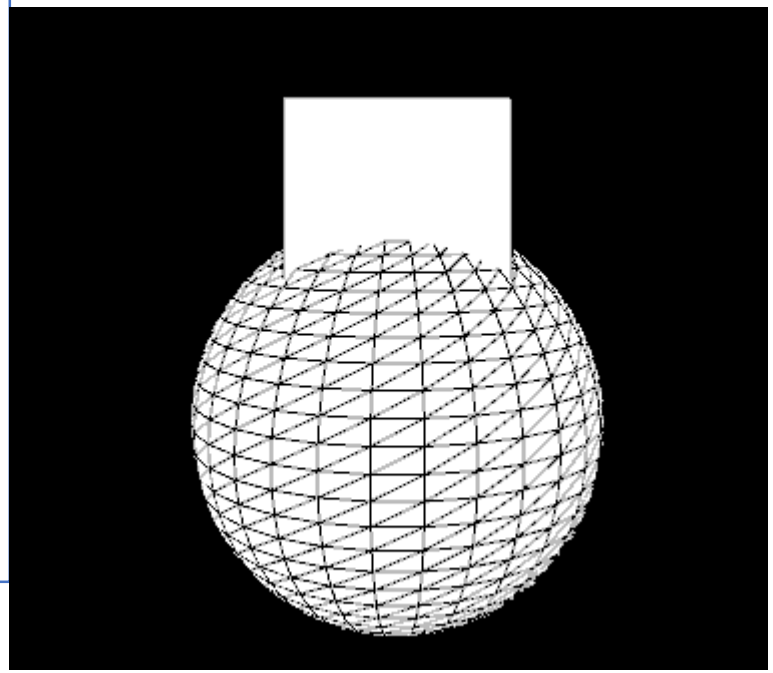
1. translate(x,y,z)で描画キャンバスを動かす
2. box()やsphere()で図形を描画

  - この手順で動かしていくのが基本
    - 座標変換を使いこなせなければ厳しい

boxとsphereを配置してみる

- 球は分かるが、boxは平面？
  ⇒　真正面から見ているだけだとよく分からない

```
void setup() {
  size(640, 480, P3D);
}
void draw() {
  background(0);
  translate(width/2, height/2, 0);
  box(100);
  translate(0, 100, 0);
  sphere(100);
}
```



描画キャンバスを回転させる

- 2Dの座標変換
  - 描画キャンバスを移動：translate(x, y)
  - 描画キャンバスを回転：rotate(角度)

- 3Dの座標変換
  - 描画キャンバスを移動：translate(x, y, z)
  - 描画キャンバスを回転：rotateX(角度)
    rotateY(角度)
    rotateZ(角度)

演習：boxとsphereを配置してみよう

1. 座標を保存
   - pushMatrix();
2. 座標変換（移動と回転）
3. 図形を描く（boxかsphere）
4. 座標を戻す
   - popMatrix();

↑の手順を繰り返して、思った場所に図形を配置

```
float angleX = 0;
float angleY = 0;
float angleZ = 0;
void setup() {
  size(640, 480, P3D);
}
void draw() {
  background(0);
  translate(width/2, height/2, 0);
  //rotateX(radians(angleX)); //縦に回したいなら
  rotateY(radians(angleY)); //横に回したいなら
  //rotateZ(radians(angleZ)); //時計回転させたいなら
  angleX += 1;
  angleY += 2;
  angleZ += 0.5;
  box(100);
  translate(0, 100, 0);
  sphere(100);
}
```

回転させたい方向以外をコメントアウト

boxやsphereを動かす

```
float angleX = 0, angleY = 0, angleZ = 0;
float x1 = 0;
void setup() {
  size(640, 480, P3D);
}
void draw() {
  background(0);
  translate(width/2, height/2, 0);
  //rotateX(radians(angleX));  //縦に回したいなら
  //rotateY(radians(angleY)); //横に回したいなら
  //rotateZ(radians(angleZ));   //時計回転させたいなら
  angleX += 1;
  angleY += 2;
  angleZ += 0.5;
  translate(x1, 0, 0);
  box(100);
  translate(0, 100, 0);
  sphere(100);
  x1++;
}
```

変数を使えば動かせる、
ただし、座標変換をよく理解出来ていないと難しい

他の図形を描くには？

- rect(),ellipse() などの2D図形はそのまま描ける
  - 平面図形なので平べったい

- 立方体と球以外の3D図形を描くには？
  1. openGLライブラリにはいくつか図形がある
  2. ないなら自分で作る
     - 平面図形を組み合わせて立体的にしていく
     - 作ったプログラムをメソッドにまとめる

円形模様のプログラムの3D版と思えばいい

これまでのプログラムの利用

- 3D円形模様を描いてみよう

1. 2Dの円形模様メソッドを作成（コピー）
2. x軸またはy軸の回転を加える
   - 円形模様全体を回転させる
3. 円形模様メソッドを呼び出す

深く考えなくても、
3Dのオリジナル模様を描けるようにしている
（移動は難しいので、レポート内容からは省略）

3Dの円形模様を作る

- 2Dの円形模様
  - 時計回りに回転させながら図形を配置

- 3Dの円形模様
  - 時計回りに回転させながら、
    更に別方向に回転させながら図形を配置
  - 2軸以上の回転をさせながら図形を配置していくことで、
    立体的な図形を作ることが出来る

2Dの場合

```
void enkeiMethod(float x, float y) {
  pushMatrix();
  translate(x, y);
  for(int i = 0; i < 10; i++){
    rect(0,0,100,50);
    rotate(radians(36));
  }
  popMatrix();
}
```

3Dの場合

```
void enkeiMethod(float tx, float ty, float tz) {
  pushMatrix();
  translate(tx, ty, tz);
  for (int j = 0; j < 10; j++) {
    for (int i = 0; i < 10; i++) {
      rect(0, 0, 100, 50);
      rotateZ(radians(36));
    }
    rotateY(radians(36));  //Xでも良い
  }
  popMatrix();
}
```

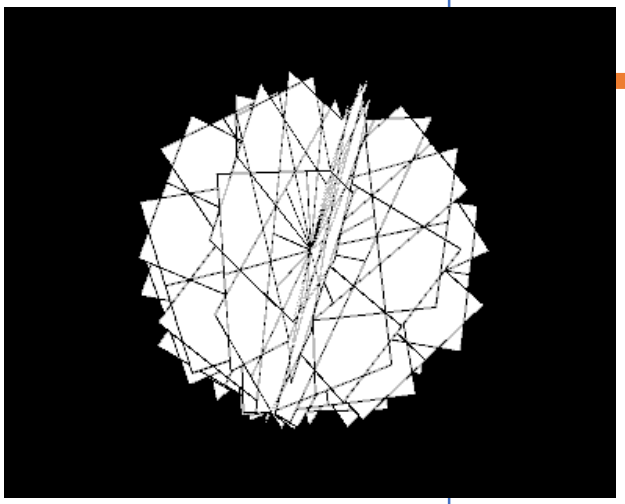
円形模様の配置(雛型で配布)


```
float angleX = 0;
float angleY = 0;
float angleZ = 0;
void setup() {
  size(640, 480, P3D);
  fill(255, 255, 255);
}
void draw() {
  background(0);
  translate(width/2, height/2, 0);

  //rotateX(radians(angleX));
  rotateY(radians(angleY));
  rotateZ(radians(angleZ));
  angleX += 1;
  angleY += 2;
  angleZ += 0.5;

  enkeiMethod(0, 0, 0);
}


void enkeiMethod(float x, float y, float z) {
  pushMatrix();
  translate(x, y, z);
  for (int j = 0; j < 10; j++) {
    for (int i = 0; i < 10; i++) {
      rect(0, 0, 100, 50);
      rotateZ(radians(36));
    }
    rotateY(radians(36));
  }
  popMatrix();
}
```



応用：沢山の円形模様を動かす

- 円形模様を沢山配置（配列を応用）
- 跳ね返る図形のプログラムに、
  Z方向の移動と跳ね返り条件を追加

- 円形模様メソッドの繰り返し回数によっては、
  まともに動かないので気を付けること

3D空間で沢山の図形を動かす(雛型配布)


```
float angleX = 0;
float angleY = 0;
float angleZ = 0;
float [] x = new float[20];
float [] y = new float[20];
float [] z = new float[20];
float [] vx = new float[20];
float [] vy = new float[20];
float [] vz = new float[20];
void setup() {
 size(400, 400, P3D);
 fill(255, 255, 255);

 for (int i = 0; i < x.length; i++) {
  x[i] = random(400);
  y[i] = random(400);
  z[i] = random(100);
  vx[i] = random(3)-1.5;
  vy[i] = random(3)-1.5;
  vz[i] = random(3)-1.5;
 }
}
void draw() {
 background(0);
 angleX += 1;
 angleY += 2;
 angleZ += 0.5;
 for (int i = 0; i < x.length; i++) {
  enkeiMethod(x[i], y[i], z[i]);
  x[i] += vx[i];
  if (x[i] < 0 || x[i] > width) {
   vx[i] *= -1;
  }
  y[i] += vy[i];
  if (y[i] < 0 || y[i] > height) {
   vy[i] *= -1;
  }
  z[i] += vz[i];
  if (z[i] < -100 || z[i] > 100) {
   vz[i] *= -1;
  }
 }
}
```

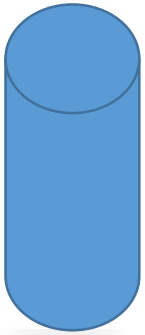
```
void enkeiMethod(float x, float y, float z) {
 pushMatrix();
 translate(x, y, z);
 scale(0.1);
 for (int j = 0; j < 10; j++) {
  for (int i = 0; i < 10; i++) {
   rect(0, 0, 100, 50);
   rotateZ(radians(36));
  }
  rotateY(radians(36));
 }
 popMatrix();
}
```

追加課題レポート：１

- 前ページのプログラム（雛型）をアレンジし、実行画像とプログラムを提出
- 最低基準や加点項目は明記しない
  - プログラムの工夫に応じて点数をつける
- 基本的には1つを選んで提出すれば良いが、複数提出してもよい

カメラの配置と移動（資料のみ）

- 3D空間での描画では、
  図形の他に、カメラと光源が必要



撮影する場所によって映る範囲が変わる
撮影する角度によって映る角度が変わる

描画キャンバスを回転させると、カメラや光源も回転する
（カメラ、光源を上手く設定しないと何も映らないことも）

カメラを設置する

camera(視点x,視点y,視点z,
注視点x,注視点y,注視点z,
天地X, 天地Y, 天地Z);

camera(0, 0, 0, width/2, height/2, 0, 0, 1, 0);

この位置から、　この位置を見る

天地はどこかを指定
（通常はy=1）

```
void draw() {
  background(0);
  camera(0, 0, 0, width/2, height/2, 0, 0, 1, 0);
  translate(width/2, height/2, 0);
  box(100);
  translate(0, 100, 0);
  sphere(100);
}
```

カメラを動かす（参考）

- このままでは、カメラの位置は固定
  - 所謂、定点カメラの状態

- 設置した物体を色々な角度から確認するためには、カメラを動かさないといけない

カメラの視点を変える

- １カメ：左上から画面中心を見る

```
camera(0, 0, 0, width/2.0, height/2.0, 0, 0, 1, 0);
```

- ２カメ：中心手前から画面中心を見る

```
camera(width/2, height/2, 500, width/2.0, height/2.0, 0, 0, 1, 0);
```

- ３カメ：中心奥側から画面中心を見る

```
camera(width/2, height/2, -500, width/2.0, height/2.0, 0, 0, 1, 0);
```

キーを押してカメラを切り替える

押したキーによってcamModeを変える

```
void keyPressed() {
  if (keyPressed) {
    if (key == '1') {
      camMode = 1;
    } else if (key == '2') {
      camMode = 2;
    } else if (key == '3') {
      camMode = 3;
    }
  }
}
```

同様の手順で好きにカメラを増やせる

camModeによって呼び出すcameraを変える

```
if(camMode == 1) {
   camera(0, 0, 0, width/2.0, height/2.0, 0, 0, 1, 0);
 } else if ( camMode == 2){
   camera(width/2, height/2, 500, width/2.0, height/2.0, 0, 0, 1, 0);
 } else if ( camMode == 3){
   camera(width/2, height/2, -500, width/2.0, height/2.0, 0, 0, 1, 0);
 }
```

カメラを自動で回転させる例

```
float camX;
float camY;
float camZ;
float radianS, radianT;
void setup() {
  size(400, 400, P3D);
  colorMode(HSB, 360, 100, 100, 100);
  camX = width/2;
  camY = height/2;
  camZ = 500;
}
void draw() {
  background(0);
  translate(width/2, height/2, 0);
  box(100);
  camX = 400 * sin(radianS) * cos(radianT);
  camY = 400 * sin(radianS) * sin(radianT);
  camZ = 400 * cos(radianS);
  camera(camX, camY, camZ, width/2, height/2, 0, 0, 1, 0);
  radianS += radians(1);
  radianT+= radians(1);
}
```