おまけ

- この資料は授業とは関係ないので，参考程度に読んでください
- 解説はしません

Processing

- Processingは「高度なオーサリングツール」とも言われる[誰によって?]通り，グラフィックスを手軽に扱えるプログラミング言語としてデジタルアート関連の分野で注目されている
- 例えば，東京工科大学メディア学部のメディアアートの授業でProcessingを教材として活用している
- C/C++に文法が似ているので親しみやすい（C/C++というかJavaだけど）

浮動小数点

- Processingが提供する関数が使用する浮動小数点数は全てfloat型で，double型は使われていない
- 基本的にProcessingではfloat型しか使わない，というのがProcessing開発陣の理念のようなものである
  https://processing.org/reference/float.html
- Processingの理念に賛同するのであれば，科学技術計算などdouble型が必要となる用途には使わないようにしたほうがいい

3DCG

- Processingでは主に2DCGプログラミングを想定しているようで，3DCGプログラミングには力を入れていないように思える
  - Processingは左手系
  - 基本的な機能しか搭載されていない
    - ProcessingではJavaの機能を利用できるので，Javaのライブラリを使うことにより，本格的な3DCGプログラミングをおこなうことができる
      - それならば最初からJavaを使えばいいのでは？
      - Processingは本格的な3DCGプログラミングには向いていない
    - 出来ることが限られている分，複雑な内容が削減されていて，分かりやすくて理解しやすい
      - 少ない行数で実装できる
      - アート関係の人にも理解しやすい
      - 教育目的に利用するのに適している
  - Processingは2DCGプログラミングに適している

サンプルプログラム

- 私の作成したサンプルソースコードを載せるが，参考にしないように
  - 私はProcessing初心者である
  - このサンプルはかなり雑に作ってある
- 3Dのライブラリとしては P3Dを使った

サンプルソースコード

```
float[] self = new float[3];
float[] item = new float[3];
float[] enemy = new float[3];
float[] enemy_v = new float[3];
float RADIUS = 0.5;

void setup(){
  size(640, 480, P3D);
  frameRate(10);
  noStroke();
  self[0] = -3.0;
  self[1] = 0.0;
  self[2] = RADIUS;
  item[0] = 3.0;
  item[1] = 0.0;
  item[2] = RADIUS;
  enemy[0] = 0.0;
  enemy[1] = 0.0;
  enemy[2] = RADIUS;
  enemy_v[0] = 0.0;
  enemy_v[1] = 0.1;
  enemy_v[2] = 0.0;
}

void draw() {
  float dx, dy, dz;
  float R2 = 2.0 * RADIUS * 2.0 * RADIUS;
  background(200, 120, 160);
  ambientLight(50, 50, 50);
  directionalLight(200, 200, 200, 0.0, 0.0, 1.0);
  perspective(PI/3.0, float(width)/float(height), 0.1, 100.0);
  camera(0.0, -10.0, 10.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0);
  PMatrix3D mat = (PMatrix3D)getMatrix();
  mat.m10 = -mat.m10;
  mat.m11 = -mat.m11;
  mat.m12 = -mat.m12;
  resetMatrix();
  applyMatrix(mat);
  myGround();
  pushMatrix();
  translate(self[0], self[1], self[2]);
  fill(160, 160, 160);
  sphere(RADIUS);
  popMatrix();
  pushMatrix();
  translate(item[0], item[1], item[2]);
  fill(40, 40, 160);
  sphere(RADIUS);
  popMatrix();
  pushMatrix();
  translate(enemy[0], enemy[1], enemy[2]);
  fill(160, 40, 40);
  sphere(RADIUS);
  popMatrix();
  enemy[0] += enemy_v[0];
  enemy[1] += enemy_v[1];
  enemy[2] += enemy_v[2];
  if(enemy[1] > 5.0) {
    enemy[1] = 5.0;
    enemy_v[1] = -enemy_v[1];
  }
  if(enemy[1] < -5.0) {
    enemy[1] = -5.0;
    enemy_v[1] = -enemy_v[1];
  }
```

サンプルソースコード

```
  ortho();
  resetMatrix();
  noLights();
  fill(255, 255, 255);
  textAlign(CENTER);
  textSize(48);
  dx = self[0] - enemy[0];
  dy = self[1] - enemy[1];
  dz = self[2] - enemy[2];
  if(dx * dx + dy * dy + dz * dz < R2) {
    text("GAME OVER", 0, 0);
  }
  dx = self[0] - item[0];
  dy = self[1] - item[1];
  dz = self[2] - item[2];
  if(dx * dx + dy * dy + dz * dz < R2) {
    text("GAME CLEAR", 0, 0);
  }
}

void myGround() {
  for(int j = -5; j < 5; j++) {
    for(int i = -5; i < 5; i++) {
      if(abs(i + j) % 2 == 0) fill(120, 120, 120);
      else fill(60, 60, 60);
      rect(i, j, 1, 1);
    }
  }
}

void keyPressed() {
  float WALKSPEED = 0.1;
  if(keyCode == LEFT) self[0] -= WALKSPEED;
  if(keyCode == UP) self[1] += WALKSPEED;
  if(keyCode == RIGHT) self[0] += WALKSPEED;
  if(keyCode == DOWN) self[1] -= WALKSPEED;
}
```