流体工学セミナー　金野担当

2011年9月15日

## Processingの起動と最初のプログラミング

　流体工学セミナーの金野担当日程では，プログラミングに慣れることを目的とし，Processingを用いた作品作りに取り組む．今回の演習では，プログラミング環境Processingを起動し，最初のプログラムを書くところまでを実習する．

## Processingとは

|  |
| --- |
| Processingとは何か  ProcesssingとはMITのBen FryとCasey Reasによってつくられた電子メディアを素材とし、視覚デザインを行うためのプログラミング言語と開発環境です。このソフトウェアは視覚的な表現を視野に入れつつ基礎的なプログラミングを教育するためにつくられ、またソフトウェアのスケッチブックとして提供されています。  現在Processingはベータリリースの段階です。現在ProcessingはベータバージョンをFreeにダウンロードできるようになり、MacやPC、そしてLinuxでも使うことができます。  Processingは個人だけでなく、多くの研究機関や大学でも使われています。 Processingに興味が出てきたら、Processingの公式サイト、Processing.orgのダウンロードページからファイルをダウンロードし、さっそくインストールしてみましょう。  （http://processing.jp/about/ より引用．2011年9月15日） |

　この授業では，プログラミング言語・環境としてProcessingを選択する．

何かひとつのプログラミング言語を表から裏まできちんと理解することができれば，ほかのプログラミング言語に応用することは容易なので，どの言語を選択するかはあまり重要ではない．ただし世の中で多く使われている言語のほうが，使うための環境がそろっていて，参考書も多く，習得しやすいことが多い．

ProcessingはC／C++言語に近い言語構造で，参考書や関連ウェブサイトが充実しており，またプログラムを書き始める敷居が非常に低いので，この授業で採用することにした．フリーソフトウェアなので，自由にダウンロードし，ファイルを展開すればすぐに使いはじめることができる．授業でのプログラミングを通して，プログラムの要素（変数の方など），言語構造（判断，繰り返しなど）の基本的な考え方を習得し，プログラミングに慣れてほしい．

## Processingの起動まで

|  |  |
| --- | --- |
| セミナーフォルダ選択.png | processing選択.png |
| 1. user（U:）ディスクの「流体工学セミナー（金野）」フォルダを選択 | 1. processing-1.5.1フォルダを開く |
| processing起動.png | processing-splash.png |
| 1. processing.exeをダブルクリックして起動 | 1. Processingの起動画面 |

図1: Processingの起動方法

　演習室のPCからProcessingを利用するには，userディスクの「流体工学セミナー（金野）」フォルダの中にあるprocessing-1.5.1フォルダを開き，その中にあるprocessing.exeをダブルクリックして起動する．するとProcessingの起動画面が現れたあと，しばらくすると図2のエディタ画面が現れる．

　エディタ画面の白い四角い部分が「エディタ」，つまりプログラムを編集する部分になっている．ここにプログラムを書いて，上の実行ボタン（右三角ボタン）を押すと，書かれているプログラムが実行される（か，エラーがあればエラーが報告される）．

　自宅のPCなどでProcessingを利用するには，processing.orgにアクセスして「Download Processing」のページに行き，自分が使っているPCのOSに合わせたファイルをダウンロードすればよい．ダウンロードしたファイルを展開すれば，すぐ実行できる．

|  |
| --- |
| (2)このボタンをクリックしてプログラムを実行  (1)この部分にプログラムを書く． |
| 図2: Processingのエディタ画面 |

## ３．はじめてのProcessingプログラミング

　早速，簡単なプログラムを実行してみよう．

プログラム例：ただ点を打つだけ．

point(50, 50);



実行結果．中心に点があるのが分かるだろうか?

Processingが自動的に100×100のキャンバスを作り，その中の座標(50, 50)の位置に点（point）を配置したために，このような絵が出る．

プログラム例：点の位置を変えてみる．

point(50, 80);



実行結果．点を打つy座標を大きくしたのに，点が前よりも下に打たれている．なぜだろうか?

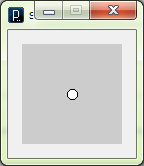
|  |
| --- |
| *y*  *x*  原点はここ |
| 図3: Processingの座標系 |

　これは，Processingの座標系が「下向き正」になっているせいである．これはコンピュータグラフィックではよくある座標系なので，慣れてほしい．

Processingが自動的に100×100のキャンバスを作り，その中の座標(50, 50)の位置に点（point）を配置したために，このような絵が出る．

プログラム例：円を描いてみる．

ellipse(50, 50, 10, 10);



実行結果．円が描かれる．

ellipseは実際には楕円を描く命令で，今回は横直径10，縦直径10の楕円を描いたので，正円が表示されている．縦横比を変えてみよう．

プログラム例：横長の楕円を描いてみる．

ellipse(50, 50, 40, 20);



実行結果．長径40，短径20の楕円が描かれる．

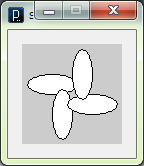
プログラム例：楕円をいくつか描いてみる．

ellipse(30, 40, 50, 20);

ellipse(40, 70, 20, 50);

ellipse(70, 60, 50, 20);

ellipse(60, 30, 20, 50);



実行結果．楕円が重なって表示される．後から描いたものが上に表示されていることに注意．

プログラム例：色を変えた楕円をいくつか描いてみる．背景の色も変えてみる．

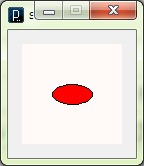
colorMode(RGB, 100); // RGBモードで，色の濃さを

// 100段階で設定

background(100, 98, 97); // 背景は薄いクリーム色

fill(100, 0, 0); // 塗りつぶし部分の色は赤

ellipse(50, 50, 40, 20);



実行結果．色は光の3原色であるR（赤），G（緑），B（青）を混ぜた色で表現する．また，プログラム中で“//”から後はコメントで，プログラムからは無視されるので，プログラムの説明などのメモを書き込んでおくとよい．

プログラム例：今度は長方形（実際には正方形）を，いろいろな色の組み合わせで描いてみる．

colorMode(RGB, 100);

background(100, 100, 100); // 背景は白

fill(100, 0, 0);

rect(10, 10, 20, 20); // (10, 10)の位置に

// 20x20の長方形を描く

fill(0, 100, 0);

rect(40, 10, 20, 20);

fill(0, 0, 100);

rect(70, 10, 20, 20);

fill(100, 100, 0); // 色を混ぜる

rect(10, 40, 20, 20);

fill(0, 100, 100);

rect(40, 40, 20, 20);

fill(100, 0, 100);

rect(70, 40, 20, 20);

fill(0, 0, 0);

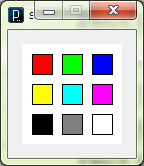
rect(10, 70, 20, 20);

fill(50, 50, 50); // 明るさを変える

rect(40, 70, 20, 20);

fill(100, 100, 100);

rect(70, 70, 20, 20);



実行結果．「rect」が長方形を描く命令で，このプログラム例では幅20，高さ20の「長方形」（正方形）が，指定した色で描かれる．標準では長方形の左上隅の位置を指定した座標に合わせて描かれる（これは変更可能である）．

プログラム例：透明度を指定して，2つの正方形を重ねて描く．

colorMode(RGB, 100);

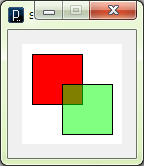
background(100, 100, 100);

fill(100, 0, 0);

rect(10, 10, 50, 50);

fill(0, 100, 0, 50); // 不透明度を指定．50%

rect(40, 40, 50, 50);



実行結果．正方形が重なって描かれるが，不透明度を指定した方の正方形は薄く描かれ，背景や下にある図形が透けて見える．

プログラム例：画面を大きくし，縁取りのない長方形を描く．また，座標の平行移動と回転の機能を使って，斜めの長方形を描く．

size(120, 240); // 画面の大きさを120x240にする

colorMode(RGB, 100);

background(100, 100, 100);

fill(0, 0, 100);

noStroke(); // 縁取りの線を引かない

rect(0, 0, 64, 240);

fill(100, 80, 0);

translate(0, 160); // 原点を移動

rotate(radians(-45)); // 座標軸を回転

rect(0, 0, 90, 8);

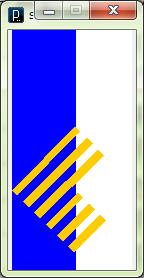
rect(0, 16, 90, 8);

rect(0, 32, 90, 8);

rect(0, 48, 45, 8);

rect(0, 64, 45, 8);

rect(0, 80, 45, 8);



実行結果．長方形に縁取りがないこと（noStroke()しているため），斜めの長方形が描かれていることに注意．

## ４．Processingプログラム例の実行

　ここまでは自分でプログラムを書いてみたが，四角や丸を描くだけではまだあまりおもしろくないであろう．Processingの能力を理解する一助として，Processingに付属しているプログラム例（Examples）を実行してみよう．

　Processingのエディタ画面から，File 🡪 Examples … を選択する．

|  |
| --- |
|  |
| 図4: Examplesの選択画面 |

　非常に多くのプログラム例が用意されているので，その中からいくつかを実行してみるとよい．

|  |  |
| --- | --- |
|  |  |
| Topics 🡪 Animation 🡪 Sequential | Topics 🡪 Fractals and L-Systems 🡪 PenroseSnowflake （印刷の都合から白黒反転しています） |
| 図5: Examplesの実行例 | |

　Processingのウェブサイト（http://processing.org/）には，さらに発展したプログラム例が紹介されている．芸術作品と呼べるものもある．

## 参考文献

　Built with Processing[Ver. 1.x対応版] -デザイン/アートのためのプログラミング入門，田中孝太郎・前川峻志・著，ビー・エヌ・エヌ新社（2010），ISBN-13: 978-4861007071．

　Processingの書籍はこの本のほかにも，ここ1，2年に多数出版されているので，是非購入し，いろいろ試してみてほしい．

## 課題

以下の課題に取り組み，その結果をレポートにまとめて提出せよ．

1. 今回の演習で実行した結果の画面をどれかひとつ，レポートに添付せよ．できるだけ後半の課題にするのが望ましい．
2. Processingを用いて独自のプログラムを作成し，適当な図形を作れ．そのプログラムおよび図形をレポートに添付せよ．凝った図形が望ましい．単純でつまらない図形は対象外とする．
3. 参考書，ウェブサイトなどでProcessingの機能を調べ，この資料には書かれていない機能を使った図形を作成し，その図形をレポートに添付するとともに，使用した機能を説明せよ．アニメーションなど動作を伴う機能の場合は，動作が分かるよう数画面を添付せよ．

提出期限：2011年9月28日（水）まで．

提出方法：MS WordファイルまたはPDFファイルにまとめ，金野までメールで送付すること．金野のメールアドレス：konno@researchers.jp

## 連絡

　次回9月22日（木）のセミナーは水野教授担当．A514室に集合すること．金野担当はその次の9月29日．

　また，9月22日（木）午後には流体研・スポーツ流体研の卒論中間発表会が予定されている．都合がつく学生は是非参加し，卒論生の発表を聴講してほしい．会場は1165教室の予定．

|  |
| --- |
|  |
| 図6: processing.orgのLearning 🡪 Getting Startedから，マウスを動かすとそれに追随して円が描かれるプログラムの実行例．この例のように，マウスの動きなどに合わせて動作を変えるプログラムを作ることもできる．なおGetting Startedにはエディタ画面のHelpからアクセスできる． |

レポートはメールで提出させるべき．

レポートのファイル名は指定するべき．流体研報告書方式．指定どおりにしなければ減点，など．

流体工学セミナー　金野担当

2011年9月29日

## Processingによる構造化プログラミング

　前回の演習ではProcessingを用いて画面に簡単な図形を描画したり，サンプルプログラムを実行したりした．今回の演習では「構造化プログラミング」を課題を通して学び，より高度なプログラムの作成にチャレンジする．

## 今回の演習の目標

* 構造化プログラミングの意味を知ること．
* Processingによる順次，反復，分岐のプログラミングができるようになること． また，関数を用いたプログラミングができるようになること．

## プログラミング言語を学ぶ上で必要な知識―なぜ，ある言語をきちんと学べばほかの言語に応用できるのか?

本セミナーの金野担当授業では，「ひとつのプログラミング言語を完璧にマスターすれば，他の言語に容易に応用できて簡単に習得できる」との考えに基づき，Processingという言語を高いレベルまでマスターすることを目標としている．では，なぜある言語をきちんと学べば，ほかの言語に応用できるのであろうか?

その理由は，多くのプログラミング言語の構成要素が共通しているからに他ならない．以下のような構成要素を学べば，多くのプログラミング言語，特に手続き型と呼ばれるプログラミング言語に応用することができる．

* データ型：整数，実数（浮動小数点数），文字，文字列，真偽値（ブール代数），配列，構造体，リスト，辞書（ハッシュ），オブジェクト，…
* 式：代入，四則演算，論理演算，ビット操作，…
* 構造化プログラミング（後述）
* プロシージャ（VBAでは「サブルーチン」），関数
* 入出力．ファイルを読み書きする，マウスのクリックを読む，画面に図形を表示する，…
* プログラミング言語が提供している機能や，ほかのプログラム構成要素（ライブラリ）の機能を自分のプログラムから呼び出して使う方法

　たとえば簡単な例として，整数型の変数aを宣言するには，次のようにする．

VBA：Dim a As Integer

C，Processing：int a;

Pascal：var a: integer;

書き方はプログラミング言語によって大きく異なるが，やっていることは同じだ．この場合は整数型の変数aを使いたいので，それに先だって使うことを宣言している．整数型の変数を宣言する，ということが分かっていれば，変数宣言の方法を調べればよい．

ほかの場合でも同様で，あるプログラミング言語でプログラムが書けるなら，ほかのプログラミング言語を使う場合にもやりたいことに対応した書き方を調べて使えばよいから，容易に応用できる．したがって，何かひとつのプログラミング言語を高いレベルでマスターすることが，突破口となる．

## データ型

Processingでは整数，実数（浮動小数点数），論理値（boolean，trueまたはfalse），文字などのデータ型がある．変数は宣言してから使う．

プログラム例：座標を変数で指定して，4つの正方形を描く．

int a; // int（整数）の変数aを宣言

a = 10;

rect(a, a, 10, 10);

a = a + 20; // aを20増加

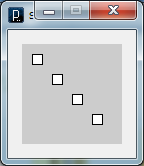
rect(a, a, 10, 10);

a = a + 20;

rect(a, a, 10, 10);

a = a + 20;

rect(a, a, 10, 10);



実行結果．(a, a)の位置に正方形が描かれる．

最初の2行は，まとめて「int a = 10;」と書くこともできる．また「a = a + 20;」は「a += 20;」と書いてもよい．

## 構造化プログラミングとは

構造化プログラミングとは，大規模プログラムを効率よく記述しプログラム設計上のミスが起こりづらいようにするための方法論で，「構造化定理」に基づいている．構造化定理とは，次のような定理である．

1つの入り口と1つの出口を持つようなプログラムは，「順次・反復・分岐」の3つの基本的な論理構造によって記述できる

したがってプログラムを，順次（書いてある順番に実行すること），反復（あるプログラム単位を繰り返し実行すること），分岐（プログラムの中で何らかの条件が成立するか否かを調べ，成立するときとしないときとで実行するプログラム単位を変える）の組み合わせのみで記述する．現在のほとんどのプログラミング言語には，この構造化プログラミング言語が最低限のプログラム構成要素として含まれている．

Processingの場合は，上記の論理構造を次のように書く．

* 順次：単に文を並べて書くだけ．

実行したい内容1;

実行したい内容2;

実行したい内容3;

　これまでのプログラム例は，すべて順次に実行される例であった．

* 反復：いくつか書き方があるが，ここでは2通りの書き方を紹介する．基本となるのは以下のwhile文である．

while (条件式) {

条件式が成り立つ（正しい，真の）あいだ繰り返し実行する内容

}

　　または，

for (文1; 条件式2; 文3) {

条件式2が成り立つ（正しい，真の）あいだ繰り返し実行する内容

}

for文では，はじめに文1を実行し，次に条件式2が成り立っているあいだ，{ }で囲まれたブロックの内容と文3とを実行する．上のfor文をwhile文に書き直すと次のようになる．

文1

while (条件式2) {

条件式2が成り立つ（正しい，真の）あいだ繰り返し実行する内容

文3

}

プログラム例：反復構造を用い，画面の対角線に正方形を描く．

int a; // int（整数）の変数を宣言

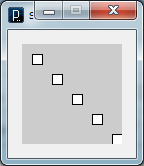
a = 10;

while (a < 200) {

rect(a, a, 10, 10);

a = a + 20; // aを20増加

}



実行結果．(a, a)の位置に正方形が描かれる．

プログラム例：上記のプログラム例と同じことをfor文で行う．

int a; // int（整数）の変数を宣言

for (a = 10; a < 200; a = a + 20) {

rect(a, a, 10, 10);

}

* 分岐：いわゆるif文

if (条件式) {

条件式が成り立つ（正しい，真の）ときに実行する内容

}

または，

if (条件式) {

条件式が成り立つ（正しい，真の）ときに実行する内容

} else {

条件式が成り立たない（偽の）ときに実行する内容

}

分岐のプログラム例は，次の節で紹介する．

## Processingの特殊な関数―setup()とdraw()

Processingは，はじめに何らかの準備をしてから，画面に何かを繰り返し描画したり，人間の操作に対応して作業をしたりするプログラムを作成するのに使われることが多い．このため，「初期化」（プログラムの最初の準備をすること）のための関数setup()と，その後，繰り返し何らかの処理を行う関数draw()を作り，それに処理させる．

プログラムを実行すると，はじめに setup()が呼び出される．変数の初期化や，キャンバスの大きさの指定，背景色の変更などを行えばよいだろう．

実際の描画動作を司るのはdraw()である．この関数は，プログラムを停止するまで繰り返し何度も呼び出される．

プログラム例：画面サイズを大きくし，マウスの位置に繰り返し円を描く．

void setup() { // 初期化関数

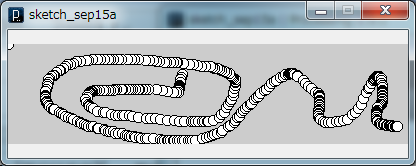
size(400, 100);

}

void draw() { // メインループ

ellipse(mouseX, mouseY, 10, 10); // マウス位置

}



実行結果．mouseX，mouseYがマウスの位置を示しており，その位置に円が描かれる．繰り返し描かれているので，マウスを動かすとマウスを追いかけるように円が描かれる．

プログラム例：上の例とほぼ同じだが，マウスボタンが押されているときは黒い円を，押されていないときは白い円を描く．if文を使っている．

void setup() { // 初期化関数

size(400, 100);

}

void draw() { // メインループ

if (mousePressed) { // もし，マウスが押されていたら

fill(0); // 明るさ0の色（＝黒）で塗りつぶす

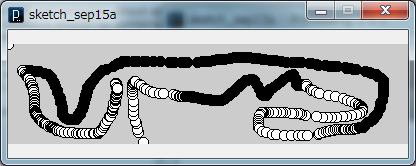
} else { // そうでなければ

fill(255); // 明るさ255の色（＝白）で塗りつぶす

}

ellipse(mouseX, mouseY, 10, 10); // マウス位置

}



実行結果．マウスを押しながら動かした部分は，黒い円で描かれる．

## Processingでアニメーション

Processingでアニメーションを描くには，画面上に絵を描いてから，background()で消去してまた描く，ということを繰り返す．backgroundは画面を指定した背景色で塗りつぶすコマンドである．

プログラム例：前節で取り上げた，マウスの位置に円を描く例とほぼ同じだが，画面の背景を毎回消去する．すると円が1つだけ描かれるので，アニメーションのようにマウスの位置を円が追いかける．

void setup() { // 初期化関数

size(400, 100);

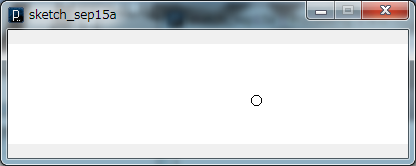
}

void draw() { // メインループ

background(255); // 画面を白で消去

ellipse(mouseX, mouseY, 10, 10); // マウス位置

}



実行結果．円を繰り返し描いているが，毎回画面を消去してから描いているので，1個しか表示されない．

プログラム例：壁で跳ね返るボールのアニメーション

int x, y; // int x; int y;と同じこと．まとめて書いている．

int vx, vy;

void setup() { // 初期化関数

size(400, 100);

x = 100; y = 50; // ボールの位置

vx = 2; vy = 2; // ボールの速度

}

void draw() { // メインループ

background(255); // 画面を白で消去

ellipse(x, y, 10, 10);

x += vx; y += vy; // ボールの位置を速度に合わせ更新

if (x < 5 || x > 395) // 左右の壁にぶつかったら

vx = -vx; // x方向速度を反転させる

if (y < 5 || y > 95) // 上下の壁にぶつかったら

vy = -vy; // 1行しかないときは，{}は省略可

}

## 課題その1：前回の課題の修正・再提出

前回課題のレポートを審査したが，内容はどれもよいものの，レポートの書き方やメールの書き方が不適切なものが多く，目に余った．就職活動も近づいているので，メールの書き方やレポートの書き方は気をつけてほしい．次の点に留意してレポートを修正し，再度金野までメールで送信せよ．

1. レポートには表紙または表題を付ける．そこには授業名，レポート課題名，学籍番号，氏名が記載されていること．つまり表紙または表題を見れば，誰の何のレポートかが分かるようにすること．諸君が出すレポートはこれで最後ではない．
2. レポートにはページ番号を振ること．
3. レポート課題の内容を書くこと．前回の課題は(1)～(3)まであったが，(1)，(2)，(3)は書いてあっても，その課題内容がなんだったかを書いていないものが多数見受けられた．レポートとして完結した内容にすること．
4. 図の下には図番号（通し番号）と表題を付けること．

ここまで述べたことは，2年生の「機械実験及演習」のレポート審査でも指摘していることです!

1. レポートのファイル名には，自分の学籍番号と，レポート課題が識別できる情報（たとえば課題番号，日付など）を入れること．諸君のレポートのファイル名はたとえば「流体工学セミナー課題」とか「流体セミナーレポート課題」など，誰のどのレポートかがまったく識別できない名前のものが半数以上であった．金野はレポートを人数分だけ受け取るということを考え，受け取る人の都合を想像せよ．
2. メールの件名（表題）は，内容が分かるように適切に付けること．
3. メールの中身がなく，ファイルを添付してきただけの学生が数名いるが，機械実験に例えれば，部屋に入ってきて一言も挨拶せずにレポートを突き出すのに等しい．メールを送るからには理由があるのだから，その趣旨を書くこと．
4. 名を名乗れ! メールに氏名が書かれていない学生数名（レポートにも書いていない学生もいました．）携帯電話のメールと同じだと思ってはならない．友達に携帯電話でメールやショートメッセージを送るときと同じような感覚で，教員（や就職先の人事担当者）にメールを送ってはならない．

提出期限：2011年10月1日（土）まで．

提出方法：MS WordファイルまたはPDFファイルにまとめ，金野までメールで送付すること．金野のメールアドレス：konno@researchers.jp

## 課題その2：テニスゲーム

プログラム例「壁で跳ね返るボールのアニメーション」を改造し，テニスゲームを作成せよ．最低でも，ラケットでボールを打ち返す（跳ね返す?）ことができるようにすること．点数が表示されるようになるとなおよい．

テニスゲームの枠を超えたものに変更してもよい．（ブロック崩し，エアホッケーなど）凝ったものを作ろう．

作成したプログラムをレポートに掲載せよ．またプログラムの動作が分かるように，数画面を添付せよ．

提出期限：2011年10月5日（水）まで．

提出方法：MS WordファイルまたはPDFファイルにまとめ，金野までメールで送付すること．金野のメールアドレス：konno@researchers.jp

|  |
| --- |
|  |
| 図1: 金野が即席で作った，壁打ちテニスゲームもどき．テニスというよりひとりスカッシュに近いか．ラケットはマウスで縦にのみ動かすことができる．上下と左の壁，およびラケットにボールが当たると跳ね返る． |

流体工学セミナー　金野担当

2011年10月6日

## Processingによるシミュレーション物理

　前回の演習では衝突・反発するボールのアニメーションと，それを発展させたゲームを作成した．プログラム例ではボールは等速で直線に動き，壁面では速さを保ったまま跳ね返る（完全弾性衝突）ものだった．実際のボールは重力や空気抵抗が作用するので等速直線運動はしないし，衝突も完全弾性衝突とはならない．今回は力学の知識をアニメーションに取り入れて，ボールにより現実的な動きをさせるプログラムを作成する．

## 今回の演習の目標

* 実空間で起こる現象を，画面上に反映させられるようになること．
* 微分方程式を数値的に解くことができるようになること．運動方程式（これも微分方程式だ）を，数値的に解くことができるようになること．ボールに運動方程式に基づく運動をさせ，アニメーションを作成すること．
* 複数のボールが運動するアニメーションを作成すること．それぞれのボールの直径をランダムに決められること．
* ボール同士の衝突を表すことができるようになること．
* 作成したアニメーションをJavaアプレットとしてウェブサーバにアップロードし，金野や他のセミナー生が鑑賞・実演できるようにすること．

## 座標変換―物理空間の座標から画面上の座標へ

これから物理的な運動をシミュレーションするにあたり，まずは物理空間（実空間）と，画面上に表示される空間との対応をつけるところからはじめる．

前回の演習で行った「壁で跳ね返るボールのアニメーション」では，画面のサイズを” size(400, 100);”と指定していた．これは表示する部分のサイズを横400ピクセル，縦100ピクセル（ピクセルは画面の表示要素1個分）にしているが，画面の大きさと物理空間の大きさとは直接対応していない．たとえば物理空間の大きさが400 m×100 mというわけではない．

そこで，物理空間の大きさをたとえば横幅10 mとし，それを画面上に表示する場合を考えよう．物理空間の横幅と高さをそれぞれfx，fyで表し，表示する画面の大きさをcx，cyで表すと，物理空間上の点(x, y)を画面の座標(px, py)に対応させるには，図1のような対応関係を作ればよい．つまり，

px = x×cx÷fx

py = (fy  y)×cy÷fy

という対応関係になる．Processingの画面ではy座標が下向き正になっているので，y=0のときpy=cy，y=fyのときpy=0となるので注意．

|  |
| --- |
| 物理空間 （実空間）  y  x  画面上の空間  （仮想空間）  py  px |
| 図1: 物理空間の座標を画面上の座標に変換する．Processingの画面ではy座標が下向き正になっているので特に注意． |

物理空間の座標を表すので，x，yは整数ではなく実数である．Processingでは実数（浮動小数点数）はfloat型の変数として宣言する．

int a; // 整数型の変数aを宣言

float b; // 実数（浮動小数点数）型の変数bを宣言

演習課題1 前回の演習で行った「壁で跳ね返るボールのアニメーション」を，物理空間上で動くボールを表すプログラムに書き換えよ．物理空間の横幅は10 mとする．

## 微分方程式を数値的に解く―Euler法

微分方程式とは要するに方程式に微分項が含まれているもので，最も単純な微分方程式はdy/dx = f(x,y)のように表される．この微分方程式を数値的に解く方法として，初歩的な「Euler法」（オイラー法，陽的Euler法）を説明する．

といっても話は簡単で，x=0のときのyがy=y0と表されるとすると，

y1 = y0 + f(x0, y0)x (x0=0)

y2 = y1 + f(x1, y1)x (x1=x0+x)

y3 = y2 + f(x2, y2)x (x2=x1+x)

:

のように，順番に近似計算する方法である．

　もう少し数学的に説明すると，dy/dxとは数学的には次のような意味を持つ．

ここで，本当はxは0になる極限を取らなければならないが，実際に極限を取るのは難しいので，十分に小さいx（たとえば，感覚的だが，x=0.001など）であれば，

と近似計算できる．上の逐次計算で行っているのは，つまりこれである．

　Euler法のいいところは，f(x,y)の計算さえできれば何も難しいところがない，というところである．何せかけ算して足し算するだけだ．欠点は計算精度が低い（悪い）ことで，研究レベルではあまり使えないと考えた方がよい．もし卒業研究などで微分方程式を解く必要がでたときには，もう少し精度の高い方法を使おう．5節で高精度な方法を説明している．

## 運動方程式を数値的に解く

運動方程式は加速度と外力との関係を記した式である．加速度は位置の2階微分なので，運動方程式も微分方程式である．1階ではなくて2階の微分方程式だが，これを2つの1階微分方程式に分解することで，前の節で述べた方法を使って解くことができる．

もとの微分方程式：

2つに分解した微分方程式：

t=0のときの位置をx=x0，速度をv=v0で表すと，それ以降の位置と速度は，

x1 = x0 + v0t, v1 = v0 + Ft/m

x2 = x1 + v1t, v2 = v1 + Ft/m

x3 = x2 + v2t, v3 = v2 + Ft/m

:

のように計算できる．

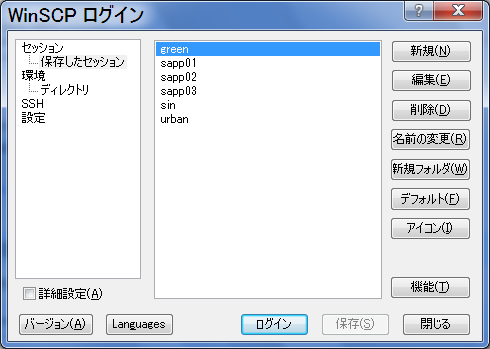
演習課題2 演習課題1のプログラムを改造し，ボールに重力がはたらくようにせよ．（y方向に加速度gを作用させる．）

## ProcessingプログラムのJavaアプレット化

プログラムを作ったら，せっかくなので金野やほかのセミナー生にも見てもらえるようにしよう．ProcessingではプログラムをJavaアプレットとして出力できるので，これをウェブサーバにアップロードすれば，Java（とJavaScript）が使えるブラウザから，そのプログラムを閲覧，実行できるようになる．

流体工学セミナー用に，ウェブサーバを用意しているので，そこに作成したJavaアプレットをアップロードしよう．以下に手順を示す．

1. Fileメニューにある”Export Applet”を選択し，プログラムをJavaアプレットに出力する．”applet”というフォルダが新たに作られ，その中にいくつかファイルが置かれるはずだ．
2. WinSCPを使って，Javaアプレットを流体工学セミナー用サーバにアップロードする．
   1. スタートメニューからWinSCPを起動する．
   2. WinSCPで，流体工学セミナー用サーバ”mizu6”にログインする．

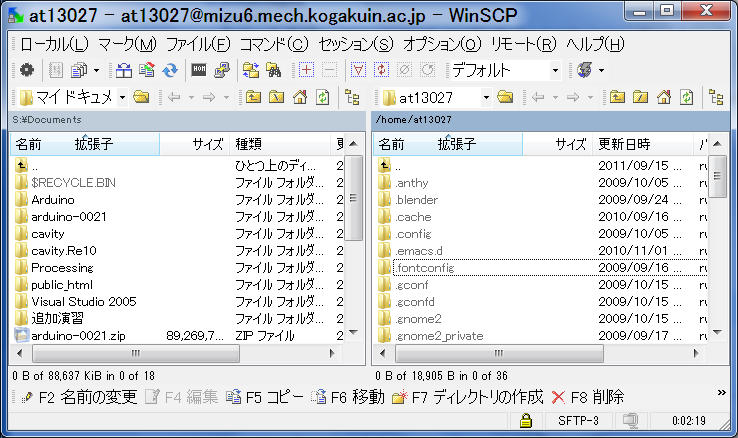


起動画面．ここで「新規」を選択

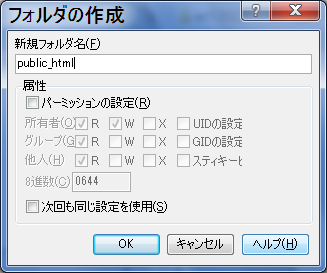


ホスト名は”mizu6.mech.kogakuin.ac.jp”，ユーザ名はログインID，パスワードは配布するパスワードを入力する．その後「ログイン」をクリック．

* 1. ログインしたフォルダ（右側）に”public\_html”というフォルダを新規に作成する．

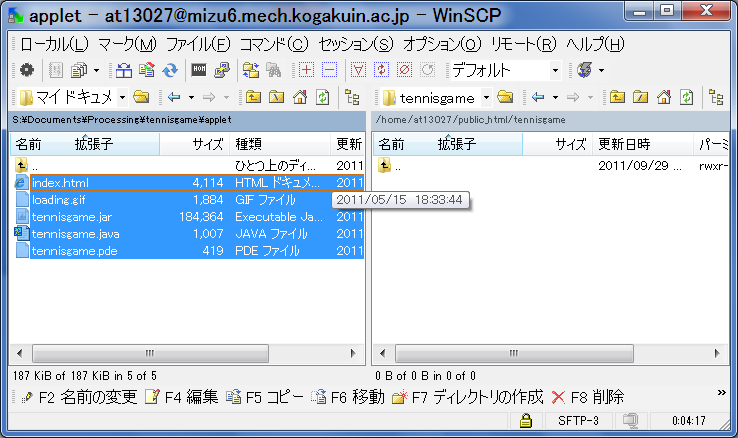


右側のペインを選択してから，「ディレクトリの作成」をクリック．ディレクトリとはWindowsではフォルダにあたるもの．左側を選択したままクリックすると，サーバではなくWindowsマシンのほうにフォルダが作られるので注意．

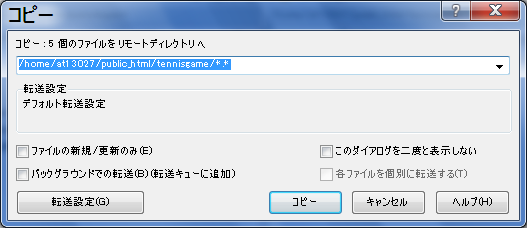


最初に作るディレクトリ名は，必ず”public\_html”とする．Javaアプレットはこのディレクトリの中にアップロードすること．

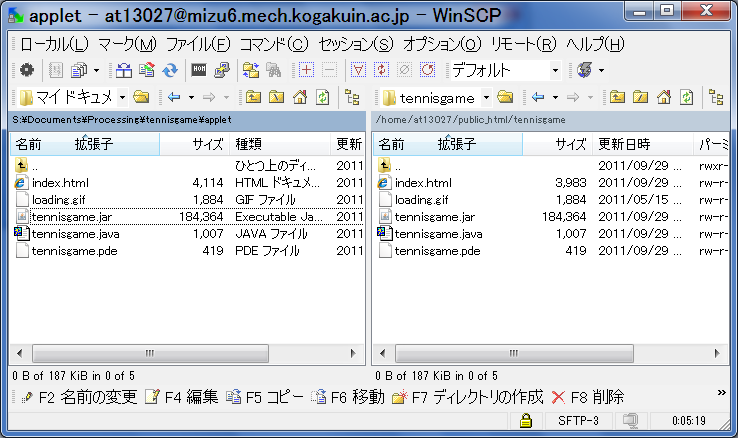
* 1. そのフォルダの中に入り，プログラムを表す適当な名前のフォルダを作成する．
  2. 先ほど作成したアプレットのファイルを，そのフォルダの中にコピーする．



左側のペインではJavaアプレットがあるフォルダに行き，右側のペインでは新しく作ったディレクトリに入る．その上でProcessingによって作られたファイルを，左から右にドラッグ＆ドロップしてコピーする．



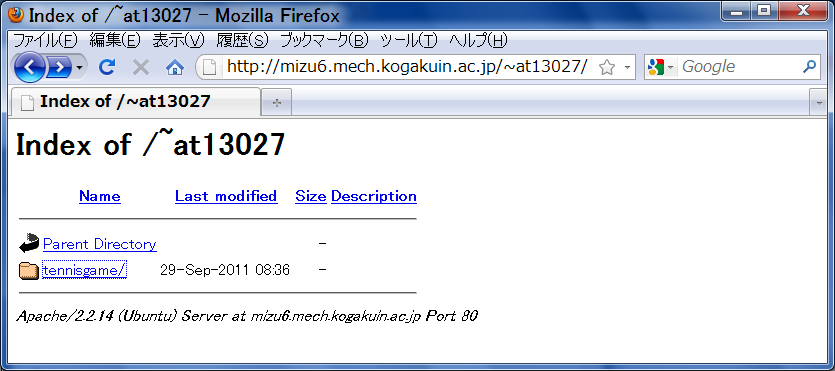
確認が促されるので「コピー」をクリック．



コピーされると，右側にも左側と同じファイルが現れる．

ファイルをコピーし終わったら，WinSCPを終了してよい．

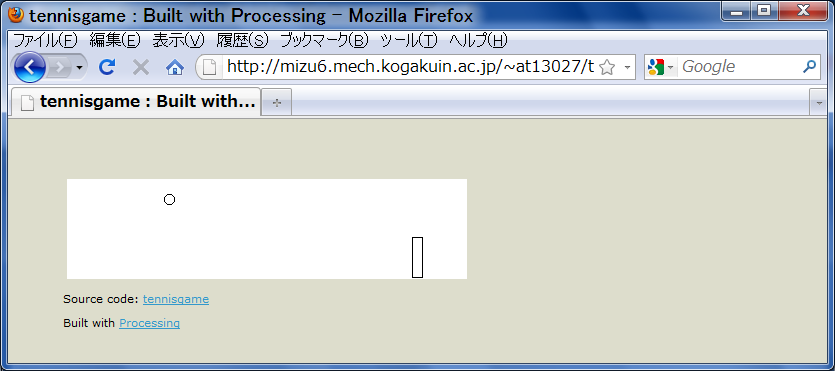
1. Mozilla Firefoxを使ってJavaアプレットを実行する（センターのWindowsマシンのInternet Explorerでは，なぜかJavaアプレットを実行できない）．



Windowsのスタートメニューから，Mozilla Firefoxを起動する．アドレスバーに以下のアドレスを打ち込む．

http://mizu6.mech.kogakuin.ac.jp/~a1*XXXXX*

a1*XXXXX*の部分は各々の学籍番号．すると上記のような画面が表示されるはずなので，先ほど自分が作ったディレクトリ（上記の例ではtennisgame）をクリックする．



Processingで作ったプログラムが，Javaアプレットとしてブラウザ上で実行される．

演習課題3 演習課題2のプログラムをJavaアプレットに変換してセミナー用サーバにアップロードし，ブラウザ上で実行せよ．

### 壁に衝突した際のエラー対策

　前回の演習では，壁にぶつかったときの判定を次のようにしていた．

if (x < 5 || x > 395) // 左右の壁にぶつかったら

vx = -vx; // x方向速度を反転させる

　速度が遅い場合にはこれでよいが，速くなってきて計算誤差が大きくなったときにはこれでは対応できない場合がある．たとえば計算誤差のためxがとても小さくなった場合，速度を反転させた次のステップでもまだx < 5かもしれない．そこでもういちど速度を反転させると，もとの速度に戻ってしまい壁で反発しないことになる．

　これの対策として，x < 5のときはvx < 0のときのみ速度を反転させ，x > 395のときはvx > 0のときのみ速度を反転させるほうがよい．

if ((x < 5 && vx < 0) || (x > 395 && vx > 0))

vx = -vx; // x方向速度を反転させる

&&はAND（「かつ」），||はOR（「または」）の意味で，上の場合はx < 5かつvx < 0の場合，またはx > 395かつvx > 0の場合にのみ，vxを反転させる．

## 計算精度を上げる―改良Euler法

演習課題2のプログラムでは，おそらく予想どおりの結果にならなかったはずだ．跳ね返りでは速度の方向を変えているだけなので力学エネルギーが保存するから，物理的にはずっと同じ高さで跳ね返り続けるはずである．しかしプログラム上では，おそらくボールの跳ね返りがどんどん高くなり，ついには上下の壁のあいだを高速に跳ね返り続けるような結果になるだろう．（逆に，だんだん低くなる場合もあり得る）これはなぜだろうか?

　これはEuler法の欠点が出たためで，つまり計算精度が悪いからである．今回の問題は加速度が常に一定だが，このような計算をEuler法で行うと誤差が出やすい．

　これを改善するために，Euler法を改良Euler法に変えてみよう．改良Euler法は次のように計算する．（dy/dx=f(x,y)を解く場合）

1. はじめにEuler法と同じ計算で，次のステップでの仮のyを求める．これをy’と書くことにする．

つまり，

1. y’を使って次のステップでの微分係数を求め，Euler法で使った微分係数と平均を取り，それを使って次のステップでのyの値を求める．

つまり，

このように2段階に解くことで，計算精度を上げることができる．現在のステップの微分係数だけを使うEuler法よりも，現在のステップの微分係数と，（仮の）次のステップの微分係数とで平均を取って計算したほうが，計算精度がよい，ということを表しており，直感的にも理解しやすいであろう．

演習課題4 演習課題2のプログラムを改造し，ボールが跳ね返って上がる高さが一定になるように（非物理的な変化をしないように）せよ．上記の改良Euler法を使うのが望ましいが，今回は加速度が一定の計算なので，それを利用して精度の高い式に変更してもよい．

## ボールを複数にする―配列

これまでの例ではボールを1つだけ動かしていたが，複数のボールが動くように変更してみよう．

1つか2つのボールであれば，それぞれのボールのために変数を用意してもよいが，10個を超えると変数の数が増えて面倒だ．変数を一括して扱いたい．このためには配列を使うのがよいだろう．Processingでは次のようにして配列を作ることができる．

float[] x = new float[10]; // float型の配列を宣言

演習課題5 演習課題4のプログラムを改造し，マウスをクリックした位置にボールが発生し，複数のボールが跳ね返り続けるプログラムを作成せよ．ボールの数は10個とする．x方向の速度は適当に定めよ．

## ボールの大きさを変える―乱数の利用

図2の実行例ではすべてのボールの大きさが等しいが，これではあまりおもしろくないので，ボールの直径をランダムに変えるようにしよう．このためには乱数を使うとよい．

乱数とはランダムな数（列）のことで，値に規則性がない．たとえばボールの直径r[i]を0.1～0.4の範囲でランダムに決めるためには，

r[i] = random(0.1, 0.4);

のようにrandom()命令を使う．

演習課題6 演習課題5のプログラムを改造し，ボールの直径をランダムに決めるようにせよ．

発展課題1 演習課題6のプログラムを改造し，ボールの色をランダムに決めるようにせよ．（発展課題は時間に余裕があるときに挑戦せよ．）

|  |  |
| --- | --- |
|  |  |
| 図2: 演習課題5（左）および6（右）のプログラムの実行例．10個のボールが弾んでいる．演習課題6（右）ではボールの直径をランダムに変えている．なおボールの軌跡が分かるように，画面を真っ白にする代わりに半透明の四角形を描画することで，前のボールの絵を薄く残している． | |

## ボールとボールの衝突―心向き斜め衝突

ここまでのプログラムでは，複数のボールが運動するが，お互いのボールは衝突せずにすれ違うだけで，単独で運動しているときと変わらない．ボール同士が衝突するようにしよう．

ボール同士が衝突しているかどうかは，ボールの中心間距離がそれぞれの半径の和よりも小さいかどうかで判定できる．つまり，それぞれのボールの半径を*rA*，*rB*，中心間距離を*d*とすると，*d< rA*+*rB*のとき衝突している．

ただし，*d< rA*+*rB*であってもお互いに離れていくときは，衝突の計算は必要ない．計算の安定性を考えると，お互いに離れていくときには衝突の計算はせず，いまの速度を維持したほうがよい．ボール同士が近づいてくるか，遠ざかっていくかの判定は，相対位置ベクトルと相対速度ベクトルの内積から判定できる．

|  |  |
| --- | --- |
|  |  |
| 図3: 相対位置ベクトル（図中の黒矢印）と相対速度ベクトル（図中の白抜き矢印）とのなす角で，物体同士が近づいているか，遠ざかっているかを判別できる．なす角が90°未満（左図）なら近づいている．90°以上（右図）なら遠ざかっている．ベクトルの内積はベクトルのなす角が90°未満なら正，90°以上なら負になるから，内積を使って判定する． | |

さて衝突の計算だが，球同士の衝突なので，工業力学の授業で学んだ「心向き斜め衝突」になる．このときの衝突後の速度を工業力学の知識からまじめに求めることもできるが，今回は手を抜いて，相対位置ベクトルの方向に適当な運動量を交換する，というプログラムを作ろう．たとえば，適当な係数を決めて，衝突後の速度ベクトルを

とする．この計算は手抜きなので非物理的な反発をするが，衝突すると反発する方向に速度が変化する，という動作は実現される．

|  |
| --- |
|  |
| 図4: 現在の速度ベクトル（図中の白抜き矢印）に，相対位置ベクトルの方向の速度ベクトルを加える．これだと運動量が保存されないので物理的にはおかしい結果になるが，衝突すると速度が変化する，という動作は実現される． |

演習課題7 演習課題5または6のプログラムを改造し，ボール同士の衝突を考慮したプログラムにせよ．

発展課題2 演習課題7のプログラムを改造し，ボール同士の衝突において運動量保存則を満たすようにせよ．ボールの質量はすべて等しいとする．

発展課題3 演習課題7のプログラムを改造し，ボール同士の衝突において運動量保存則を満たすようにせよ．ボールの質量は半径の3乗に比例することにする．

## 参考になるウェブサイト

inutch / Processing

http://inutch.net/processing/

今回の演習で取り組んだ課題が，高いレベルで実現されています．

## 課題

以下の課題に取り組み，その結果をJavaアプレットに変換してセミナー用サーバにアップロードせよ．

1. 演習課題5または演習課題6のいずれか
2. 演習課題7．衝突の動作が物理的に見ておかしくても，衝突したときに速度が変化するようなプログラムであれば可とする．
3. 発展課題に取り組んだ場合はその結果．

セミナー用サーバへのアップロードは，学内ネットワークからしか行えないので注意．学外，たとえば自宅からセミナー用サーバにアップロードするためには，大学のネットワークにVPN接続する必要がある．

提出期限：2011年10月26日（水）まで．

## 連絡

　次回10月13日（木）および20日のセミナーは水野教授担当．A514室に集合すること．次回の金野担当授業は10月27日（金野担当の最終回）．

流体工学セミナー　金野担当

2011年10月27日

## Processingによるインタラクティブ・アート

　前回の演習ではシミュレーション物理に挑戦し，実空間で起こる現象に対応するアニメーションの作成とJavaアプレット化までを行った．今回は前回の成果をさらに発展させ，より正確な衝突や運動を実現させたライブラリを利用し，外力（引力，バネの復原力など）が作用したときの動作を模擬する．最終的にはこれを利用した作品を作る．

## 今回の演習の目標

* ライブラリを利用して，自分のプログラムの機能を拡張できるようになること．
* 重力以外の外力（バネの復原力，万有引力など）を作用させることができるようになること．
* 固定した物体を配置できるようになること．

## ライブラリの利用

今回の演習のために，ボール（画面上では円盤）の運動方程式に基づく運動と，衝突および反発を実装したライブラリ，およびそれを利用したプログラムを作成した．下記のアドレスからダウンロードして利用できる．

http://mizu6.mech.kogakuin.ac.jp/~at13027/bouncingball10/

|  |  |
| --- | --- |
| 1. 上記のウェブサイトにアクセスし，Processingのソースコードをダウンロードする．   ここから2つのファイルをダウンロード |  |

1. Processingを起動し，Fileメニュー 🡪 Open…でbouncingball10.pdeを開く．

|  |  |
| --- | --- |
|  | このようなダイアログボックスが表示されたら，”OK”をクリック．フォルダが作られファイルが移動する． |

1. Sketchメニュー 🡪 Add File…でbouncingballs.pdeを選択，追加する．この場合はファイルは移動されずにコピーされる．
2. これでbouncingball10プログラム（スケッチ）が実行できるはずである．

　ダウンロードした2つのファイルのうち，bouncingballs.pdeが運動や衝突の計算を司るライブラリである．もう一方のbouncingball10.pdeはライブラリを利用して動作するプログラムで，今回の演習ではこのプログラムを改造することで新しい機能を実現していく．

演習課題1 ダウンロードしたプログラムbouncingball10を，自分のProcessing環境で実行し，動作を確認せよ．

## 外力を変えてみる

これまではすべてのボールに同じ加速度（重力加速度）を与えて運動させていた．これは，すべてのボールに同じ重力をかけていたことに相当する．ボールの運動を変えるため，いくつかのボールには重力をかけ，それ以外には重力をかけないようなプログラムを書いてみよう．

ボールにはたらく力は，プログラムbouncingball10の中のforceという関数で定義されている．この関数を改造することで，ボールにはたらく力を変えることができる．

演習課題2 演習課題1のプログラムを改造し，重力がはたらかないようにせよ．

演習課題3 演習課題1のプログラムを改造し，最初の5個のボールには重力がはたらくが，それ以降のボールには重力がはたらかないようにせよ．なおボールの番号は関数forceの引数\_iで調べることができるが，\_iは0からはじまるので，\_iが5より小さければ（\_i<5），最初の5個である．

演習課題4 演習課題1のプログラムを改造し，最初の3個のボールにははじめに置かれた位置に引き戻されるような外力を作用させよ．バネでつながれたボールのような動きをするはずである．（はじめの位置を配列に記憶しておき，移動したときに外力を作用させればよい．kはバネ定数）

発展課題1 演習課題4のプログラムを改造し，速度に比例した空気抵抗がはたらくようにせよ．これによってバネによる振動を減衰振動にすることができる（つまり，だんだん振動が止まる）．

## 相互作用力を入れる

前の節で与えた外力は，そのボールの位置などの条件のみで決まり，周りにあるボールの影響を受けないものだった．ここでは，たとえば万有引力のように，周りのボールから（衝突以外で）力を受けるように変更してみよう．これができれば，太陽系のシミュレーションなどができるようになる．

プログラム例：万有引力を作用させる

float[] force(int \_i, int \_count, float[] \_x, float[] \_y,

float[] \_vx, float[] \_vy)

{

float rx, ry, d;

int j;

fret[0] = fret[1] = 0;

for (j = 0; j < \_count; ++j) {

if (\_i == j)

continue; // 自分の引力は考えない

rx = \_x[j] - \_x[\_i]; ry = \_y[j] - \_y[\_i];

d = sqrt(rx\*rx + ry\*ry);

fret[0] += G\*m[\_i]\*m[j]/(d\*d)\*rx/d;

fret[1] += G\*m[\_i]\*m[j]/(d\*d)\*ry/d;

}

return fret;

}

このプログラムでは重力定数Gを使っているので，どこかでGを定義する必要がある．

演習課題5 演習課題1のプログラムを改造して万有引力を作用させ，プログラムの動作を確認せよ．なお，速度と反発係数が小さいと引き寄せあってすぐに1カ所にまとまってしまうので，速度は大きめ，反発係数は1.0にしておくとよい．

## 物体を固定する

すべての物体が動くのではなく，固定した物体を置きたいときもあるであろう．たとえば演習のプログラムを発展させてパチンコゲームのプログラムを作るには，釘にあたるものがなければならない．チューリップも必要かもしれない．これはどうすればよいだろうか?

　固定した物体を表すプログラムをまじめに書いてもよいが，これは意外に面倒である．「ほぼまったく動かない」条件でよければ，動かしたくない物体を大きな質量にしてしまえばよい．たとえばプログラム上であるボールを1,000,000キログラムにすれば，10キログラムぐらいの物体がぶつかったところでびくともしないであろう．

演習課題6 演習課題2のプログラムを改造し，いくつかのボールを固定せよ．固定するためには，物体の質量を大きく（106ぐらいに）し，初速度を0にする．また，重力を作用させると物体の質量に関係なく落下運動するので注意せよ．

演習課題7 演習課題6のプログラムを改造し，プログラムを起動した直後から複数の物体が固定されているようにせよ．小さなボールを並べてコースを作る，パチンコ台を作る，ピンボールゲーム風のものを作る，など．

## 物体を指示どおりに動かす

前の節では物体を固定したが，ここでは物体を自由に動かすことに挑戦する．たとえばマウスの動きに合わせて動かすようにしよう．

　これはすでに第2回の演習で行っていることなので，簡単だと感じるかもしれないが，今回行っている課題では物体の運動や衝突をすべて力学の法則に基づいて行っているので，それに反した方法で物体を動かすとプログラムの動作が破綻する．注意しなければならない．

演習課題8 演習課題1のプログラムを改造し，最後（10番目）のボールにはマウスの位置に引きつけられるような外力を作用させよ．

演習課題4でバネにつながれたボールのような動きをするプログラムを作ったが，それを応用すればよい．ただし画面上のマウスの位置（座標）は，実空間での座標とは異なるので，きちんと変換する必要がある．

課題

今回の演習で学んだ知識を用い，Processingによるインタラクティブ・アート作品を制作せよ．

作品は以下の条件を満たすものとする．

1. 「シミュレーション物理」の課題で学んだ知識を利用すること．授業で学んだ知識を使っていないアート作品は原則として不可．（優れた芸術作品であれば許容する．）
2. インタラクティブなものにすること．マウスのクリックやキーボード入力に応じて，動作が変わるようなプログラムにすること．
3. 金野が観て「すごい!」または「おもしろい!」と感じること．（ここ重要）

提出方法：

1. 制作した作品を Javaアプレットに変換し，セミナー用サーバにアップロードせよ．
2. その作品について説明したレポートをMS WordファイルまたはPDFファイルにまとめ，金野までメールで送付せよ．レポートには最低限，以下の情報を含めること．
   1. Javaアプレットのアドレスを明記
   2. プログラムの動作が分かるように，数画面を添付すること．（レポートだけでプログラムの内容がある程度伝わるようにすること．）
   3. その作品の使い方や，作品を楽しむのに必要な知識（もしあれば）．たとえば，ゲーム作品であればゲームのルールなど．
   4. そのプログラムの中で使われている，授業で学んだ知識
   5. そのプログラムの中で使われている，自分で調べた機能，知識など
   6. 作品のアピールポイント

提出期限：2011年12月2日（金）まで．

金野のメールアドレス：konno@researchers.jp

## 補遺1・衝突の判定再考

|  |  |
| --- | --- |
| *i*  *j* | *i*  *j* |
| 図1: 物体*i*からみた物体*j*の相対位置ベクトル（図中の黒矢印）と相対速度ベクトル（図中の白抜き矢印）とのなす角で，物体同士が近づいているか，遠ざかっているかを判別できる．なす角が90°未満（左図）なら遠ざかっている．90°以上（右図）なら近づいている．ベクトルの内積はベクトルのなす角が90°未満なら正，90°以上なら負になるから，内積を使って判定する． | |

前回の演習で衝突の判定について説明したが，一部不適切だと思われる説明だったので，改めて説明する．

ボール同士が衝突しているかどうかは，ボールの中心間距離がそれぞれの半径の和よりも小さいかどうかで判定できる．つまり，それぞれのボールの半径を*rA*，*rB*，中心間距離を*d*とすると，*d< rA*+*rB*のとき衝突している．

ただし，*d< rA*+*rB*であってもお互いに離れていくときは，衝突の計算は必要ない．計算の安定性を考えると，お互いに離れていくときには衝突の計算はせず，いまの速度を維持したほうがよい．ボール同士が近づいてくるか，遠ざかっていくかの判定は，相対位置ベクトルと相対速度ベクトルの内積から判定できる．

## 補遺2・ボールとボールの衝突―心向き斜め衝突（再考）

|  |  |
| --- | --- |
| 質量*mA*  速度*vA*  質量*mB*  速度*vB* |  |
| 図2: 心向き直衝突 | 図3: 心向き斜め衝突では，中心軸を結んだ線の方向（図中の破線方向）の速度成分は心向き直衝突と同じように変化する． |

演習のプログラムではボールとボールとが斜めにぶつかる場合を考えなければならないが，はじめから斜めに考えると難しいので，最初は図2にあるような「心向き直衝突」の場合の，衝突後の速度を求めよう．このような問題では，運動量保存則の式と反発の法則の式とを連立させて解くことで，速度を求めることができる．（まさか運動量保存則の式を忘れていないだろうね!?）

球Aの質量を*mA*，衝突前後の速度をそれぞれ*vA*，*vA’*と書く．球Bは*mB*，*vB*，*vB’*である．すると運動量保存則より，



また反発の法則より



を得る．この2式を連立させて，*vA’*， *vB’*を求める．まず反発の法則より，

．

これを運動量保存則の式に代入して，





を得る．

　球Aの運動量に着目すると，*mAvA*だった運動量が*mAvA’*へと変化するから，運動量がだけ変化したことになる．球Bも同様である．

　図3のような斜め方向の衝突（心向き斜め衝突と呼ぶ）の場合は，球の中心と中心とを結んだ直線（図中破線）方向の運動量が，衝突前後に上記の式に従って変化する．そのため，まずは運動量を中心軸の方向とそれに垂直な方向に分解し，中心軸方向の運動量を変化させた後，再び垂直方向の運動量と合成すれば，衝突後の運動量を求められる．

（bouncingballsライブラリで実現されているので，詳細は省略する．）