Processing によるプログラミング基礎教育の試み

倉田 和夫 Kazuo KURATA

島森 功 Isao SHIMAMORI

概要:筆者らは、2006 年度の本部例会でフリーの CG ソフト POV-Ray を使った初心者のためのプログラミング教育事例の発表を行った。今回は、その研究の第2弾として MIT で開発された Java ベースの言語開発環境である Processing によるプログラミング初心者向けのプログラミング教材について発表する。この Processing はデザイン/アートの分野での作品制作に特化されたプログラミング環境で、作品制作を楽しみながらプログラミングに必要な論理的な思考を教育する授業教材の使えるのではないかと考え授業に取り入れているその試みを発表する。

キーワード:造形教育／プログラミング／CG

1. はじめに

筆者らの所属する短期大学部は、簿記会計などの事務処理技術者を養成するマネジメント分野 200 名、プログラマを養成するネットコミュニケーション分野 80 名、CG や DTP、Web デザイン分野のデザイナーやプログラマを養成するデジタルデザイン分野 40 名、ハードウェア技術者を養成する 20 名の 4 つ分野で学ぶ約 350 名の学生が在籍している。本学では、プログラミング教育を全学生の必修科目として位置付けているためデザインを学ぼうとする学生もプログラミングを履修しなければならない。将来プログラマを目指すネットコミュニケーション分野以外の学生にとって、とりわけデザイン系の学生にとってはプログラミングの授業は、数学や物理と並んで、つまらない、難しい、退屈であると敬遠される科目となっている。プログラミング教育を担当する筆者らにとって学生たちの関心をいかにプログラミングに向けさせるかが大きな課題となっている。これまで昨年発表した POV-Ray を使ったプログラミングの基礎教育の試みなどを行ってきた。筆者らのプログラミング基礎教育はプログラマ養成のために文法を覚えさせることよりその論理的思考力、思考法の修得に重点を置いている。従来の C や Java のプログラミング入門が文字の表示や入力といったテキスト操作を中心として行われていることが、つまらない、退屈といった学生の声の大きな要因ではないかと考えている。筆者らが直接学生から聞いた経験からも最初のプログラム体験がその後のプログラムへの興味の有無に大きな影響を与えることは想像できる。

2. 教育の内容

2. 1. Processing について

2. 1. 1. なぜ Processing か

Processing は MIT メディアラボでプログラミング教育を受けていないアーティストやデザイナーのために開発された教育環境 Design By Numbers (DBN) に起源をもつと言われている。DBN は 10 個程度の命令を使って簡単なプログラミングを体験できるもので Java アプレットとして実行されている。この DBN は 100×100 程度の小さな画面にモノクロで 2 次元の作画命令を用意し繰り返しや条件分岐などの基本的なプログラミングの体験ができるもので、プログラミングがどのようなものであるかを大雑把に掴むことができる。この DBN を基にして大きな画面でフルカラー表示や 3 次元処理などができるように機能拡張して Processing が開発された。2001 年秋から MIT のメディアラボの授業で使われ始め、2002 年夏に α バージョンが公開されて注目されるようになった。現在 β バージョンの開発が続けられている。筆者らがこの Processing に注目した理由のひとつは Processing の記述が POV-Ray より実際のプログラミング言語の表記に近いという点がある。CG 表現に関しては POV-Ray が優れているものの、プログラミングの記述としては、C 言語の表記に非常に親和性が高い。また、POV-Ray 同様にフリーソフトで、学生が自宅で勉強することができる点も大きい理由となっている。

2. 1. 2. Processing の動作環境

すでに紹介したように Processing はフリーソフトである。インターネットから誰でも無料で入手できる。また、Windows 以外の代表的な OS 上でも動かすことができる。筆者らは、Windows で動かしている。また、POV-Ray のようなインストールをする必要は無い。ダウンロードした圧縮ファイルを解凍して生成したプログラムをダブルクリックして動かせる。2007 年 11 月 2 日の最新バージョンは β 版の Processing-0133 となっている。

2. 2. カリキュラム

本稿では、次の 2 つの授業で Processing を利用した授業を行っている。ひとつは島森が担当する武蔵野美術大学の造形学部基礎デザイン学科の 3 年次開講科目「デザイン演習-j(CG)」(受講者 6 名)と、倉田が担当する情報・ネットワーク学科の 2 年次開講科目「総合実習 B(プログラミングによる CG 表現)」(受講者 7 名)である。

2. 3. 講義内容とテキスト

「デザイン演習-j(CG)」(表 1)の授業の目的は、身の回

りにある自然物(動植物や自然現象等)の観察から美の規則性を発見し、それをプログラミングを通して再現することにより造形に対する認識と理解を深める。また、それを効果的なプレゼンテーション(静的・動的)に結びつける技術を習得することである。受講する学生にはプログラミング経験は要求していないため、受講者のほとんどがプログラミング未経験者である。しかしプログラミングによるヴィジュアルコミュニケーションを目的としているため、ある程度のプログラミングの理解は不可欠である。改めてプログラミング教育を行う時間的な余裕は無く、効率よくプログラミングの概念が理解できること、かつ授業のもうひとつの目的でもあるインタラクティブアートの機能が組み込まれていることから Processing を採用することになっている。授業の流れは、最初に全体を通しての課題の目的を説明し、第 5 週から 8 週までに Processing の使い方を習得させ、残りの週で Processing によるプレゼンテーション課題の制作を行う。テキストはすべて筆者らが独自に作成したものを使用している。このテキストは授業開始時に配布し、これに沿って授業を進めている。

表1 「デザイン演習-J(CG)」の講義スケジュール

週	講義内容
1週	ガイダンス
2週	課題説明・班分け
3週	アイデアの持ち寄り／研究対象の決定
4週	研究対象の確認
5週	プログラミング・素材画像の制作
6週	プログラミング・素材画像の制作
7週	プログラミング・素材画像の制作
8週	プログラミング・素材画像の制作／中間発表
9週	プレゼンテーションの計画と立案
10週	プレゼンテーションの準備
11週	プレゼンテーションの準備
12週	プレゼンテーションの準備
13週	プレゼンテーション／講評会

もうひとつの「総合実習 B(プログラミングによる CG 表現)」(表 2)は、コンピュータグラフィックスで使われている技術の中からいくつか代表的なものをピックアップしてその内容をプログラミングを通して体験的に学習する授業である。具体的にはパーティクルシステム、絵画のような表現手法やセルアニメ風のレンダリングを実現するノンフォトレンドリング技法の2つを採りあげてプログラミングを通して技術を理解する。履修の条件としてプログラミング科目が履修済みであることを要望しているが、受講者の半数以上がプログラミング初心者となっている。この授業ではプログラミング概念、とくに C 言語をベース

に講義を進めることから C や Java についてある程度の理解が前提となる。この矛盾を解決するために効率よくプログラミング概念を理解し、なおかつ C 言語の文法にも抵抗無く親しめる Processing を採用している。授業は秋学期の授業で現在進行中である。10 月末現在第 5 週の Processing を使ったプログラミング課題が終わったところである。後半の授業では簡単なパーティクルシステムを使った噴水のプログラムと写真画像を元にモザイク画を自動生成するノンフォト風レンダリングのプログラミングを C+OpenGL を使って体験させることになる。

表 2 「総合実習(プログラミングによる CG 表現)」の講義スケジュール

週	講義内容
1週	ガイダンス
2週	Processing プログラミング
3週	Processing アニメーション
4週	プロセッシングによる作品制作(1)
5週	プロセッシングによる作品制作(2)
6週	講評会
7週	パーティクルシステムとは (煙、水、雲の表現)
8週	パーティクルシステム体験① 課題(噴水を作る)
9週	パーティクルシステム体験②
10週	パーティクルシステム体験③
11週	ノンフォトレンドリング体験① 課題(モザイク画自動生成)
12週	ノンフォトレンドリング体験②
13週	ノンフォトレンドリング体験③
14週	講評会

2. 3. 1 テキスト教材

テキスト教材は HTML で作成している。プログラミングの予備知識なしに理解できるように考えて作成した。教材を開いたときウィンドウを縦長に表示できるように 1 行の文字数を少なくし、Processing を動かしながら教材を見ることができるよう配慮している。(図1)

教材の内容は以下の通りである。

0. マニュアルの使い方と注意点

1. 基本操作

- ・Processing の概要 ・ダウンロードとインストール
- ・エディタの使い方 ・エラーが出たら

2. 図形の描画

- ・画面と座標 ・色の指定 ・点 ・直線
- ・矩形 ・三角形 ・楕円 ・円弧 ・曲線
- ・点を結んだ形 ・画像 ・文字

3. プログラミング手法(1)

- ・変数 ・乱数 ・配列 ・関数 ・繰り返し

- ・条件判断
- 4. アニメーション
 - ・イベントと無関係に動く ・マウスに反応させる
 - ・キーボードに反応させる ・日付時刻を読み込む
 - ・DVカメラ/USBカメラ ・その他の入力機器
- 5. 出力
 - ・画像 ・動画 ・PDF ・アプリケーション
 - ・Web
- 6. プログラミング手法(2)
 - ・座標の変換 ・再帰(自己相似図形)
 - ・ピクセル操作 ・三次元表現
- 7. CG数学基礎
 - ・三平方の定理 ・三角関数 ・放物線
 - ・行列 ・座標系について
- 8. CG色彩学基礎
 - ・色の三属性 ・CGにおける表色系
 - ・色度図と色空間 ・カラーマネージメント

現在のマニュアルは暫定版であり一部の説明が抜けているので随時追加していく予定である。

図1 教材を表示している画面

2. 4. Processing のプログラムと実行

Processing はプログラムを起動すると、エディタを内蔵した統合環境のプログラムが立ち上がる。(図 2)エディタ画面でプログラムを書き、メニュー上部の実行(Run)ボタンを押して実行する。実行結果は別のウィンドウが開き表示される。ソースファイルは自動的にマイドキュメント内に Processing という名前のフォルダが作られそこに

保存される。ファイル名は Sketch+日付+a と自動的に割り振られる。

2007 年 11 月 2 日に作成した場合 Sketch_20071102a となる。

図2 Processing 起動時の画面

2. 4. 1. 画面の定義

Processingでは結果を表示する画面を作成する必要がある。画面の作成はsizeという命令を使う。

```
size(320,240);
```

引数は最初の数字が横幅、後の数字が高さの画素数である。

図3 size(320,240)で表示される画面

2. 5. Processing でプログラムを書く

Processing は Java をベースに開発されているために非常に C や Java に類似している。いくつか実例を示そう。筆者らがプログラミング概念として理解してほしいと考えている要素は

- ①変数の概念
- ②繰り返し処理の考え方
- ③条件分岐の考え方
- ④関数の考え方

などである。ここではこの各要素を Processing でどのように記述されるかを説明する。

2. 5. 1 変数を使う

Processing ではほぼ C や Java と同様の変数を使うことができる。

使用例

```
int    x,y;
float  AspectRatio;
```

変数は基本のデータ型として

- int(整数)
- float(浮動小数点数)
- char(文字)
- byte(バイト)
- boolean(ブーリアン)
- color(色)

が用意されている。

変数は定義される場所によってグローバル変数とローカル変数とに区別される。他に特殊なものとして width(画面横幅のデータ)、height(画面の高さデータ)や PI(円周率)などがある。また型の変換を行う以下の関数が用意されている。

- int(変数)
- float(変数)
- char(変数)
- byte(変数)

配列変数も定義は C などと若干文法が違うものの使うことができる。

配列の定義例

```
int [] array = new int[30];
```

この例では整数型で要素数が 30 の配列 array を定義している。

2. 5. 2 繰り返し処理をする

Processing の繰り返し処理は for ループと while ループが用意されており、記述は C などとほとんど同じである。

以下に for ループを使った繰り返しの例を示す。for(){ } の間に記述された内容が繰り返される。

```
size(640,480);
colorMode(RGB);
background(#FFFFFF);
```

```
int x,y,d,r,g,b;
for(int i=0; i<50; i++){
    x=int(random(width));
    y=int(random(height));
    d=int(random(5, 50));
    r=int(random(255));
    g=int(random(255));
    b=int(random(255));
    noStroke();
    fill(r, g, b);
    ellipse(x, y, d, d);
}
```

colorMode(RGB)は色を RGB で設定するか HSV で設定するかを指定する。background(#FFFFFF)は背景の色を設定する。#FFFFFF は色を 6 桁の 16 進数で指定するもので#に続く最初の 2 文字が赤のレベル、続く FF が緑のレベル、最後の FF が青のレベルを指定する。#FFFFFF は白をあらわす。

それぞれのレベルは 00 から FF までの 256 段階で設定する。random()は擬似乱数を発生させる関数で発生する乱数の範囲を()の中に指定する。

random(); 0.0~1.0 の範囲で乱数発生

random(N); 0.0~N の範囲で乱数発生

random(N,M); N~M の範囲で乱数発生

noStroke()は図形の線を描かないという指定。fill(r,g,b)は塗り潰しの色を乱数で決めた r,g,b で指定している。ellipse(x,y,d,r,g,b);は半径 d の円を中心(x,y)に表示するプログラムである。

図4 繰り返しサンプルの実行結果画面

このように繰り返した結果が可視化されることで繰り返しの概念が理解しやすい。

2. 5. 3 条件によって処理を分ける。

条件分岐の処理も C と同じ書き方で表現できる。条件分岐を行う場合は次のような命令を使う。

```
if(条件)
  処理 A
else if
  処理 B
else
  処理 C
```

単に条件が成立したときのみ処理をしたい場合は else if 以下を省略することができる。以下に条件分岐を使った例を示す。

```
size(320, 240);
colorMode(RGB);
background(#FFFFFF);
```

```
int x, y, d, r, g, b;
```

```
for(int i=0; i<50; i++){
  x=int(random(width));
  y=int(random(height));
  d=int(random(5, 50));
  r=int(random(255));
  g=int(random(255));
  b=int(random(255));
  if (i % 2 == 0) {
    noFill();
    stroke(r, g, b);
  }else{
    noStroke();
    fill(r, g, b);
  }
  ellipse(x, y, d, d);
}
```

ここでは 50 個の図形を、偶数番目の表示では線画で描き、奇数番目の表示は塗り潰すようにプログラムしている。 $i \% 2$ は i を 2 で割った余りを計算している。条件分岐の文法は言語 C と同じであり、少なくとも繰り返しや条件分岐に関しては Processing での学習は C 言語の学習と同じとみなすことができる。演算子についても $i++$ や $\%$ 、 $+=$ や $==$ や $<$ 、 $>$ などの関係演算子、 $\|$ や $\&\&$ などの論理演算子も同じように使うことができる。

図5 条件分岐サンプル実行結果画面

2. 6. 関数を定義する。

関数は、C などではプログラムを作ることは関数を作ることと同じ意味を持つほど重要な概念である。Processing では C や Java と同様に関数を作ってプログラムを構造化することができる。

2. 6. 1. 自作の関数を作る。

Processing で自作関数を作る場合はセットアップ関数と呼ばれる `setup()` と自動的に繰り返し呼び出されるループ関数 `draw()` を準備しなければならない。ループ関数を自動的に繰り返し呼び出すタイミングを `framerate()` で設定する。`framerate(10);` と指定すると一秒間に 10 回呼び出される。

`setup()` は初期設定処理を行うもので

画面サイズを決める。(size)

背景の色を決める。(background)

フレームレートを決める。(framerate)

色の指定の仕方を決める。(colorMode)

直線の作画モードを決める(rectMode)

などの処理を行う。

`draw()` は `framerate()` で指定されたタイミングで繰り返し実行される。

以下に自作関数の例を示す。

```
void setup(){
  size(320, 240);
  background(#FFFFFF);
  noFill();
}

void draw(){
  sikaku(70, 50, 60);
  sikaku(30, 20, 80);
  sikaku(90, 30, 50);
}
```



```
void sikaku(int x, int y, int s){
  line(x, y, x + s, y);
  line(x + s, y, x + s, y + s);
  line(x + s, y + s, x, y + s);
  line(x, y + s, x, y);
}
```

setup()関数内では画面サイズを 320×240 にし、背景を白、塗り潰しをしないと指定している。draw() では一辺の大きさを指定して正方形を書く自作関数 sikaku()を呼び出している。

図6 自作関数サンプル実行結果画面

3. 評価

3. 1. Processingを採用することへの評価

今回はプログラミング初心者に対して Processing を教材として使う試みを紹介した。現時点では、ここで我々が用意した教材の評価と、Processing が本当に効果的にプログラミングの理解に繋がるかどうかなどの客観的な評価を下すほどのデータがない。今後より詳細な分析と調査ができた辞典で改めて発表させていただく。少人数の受講者であったが、これまでCやJavaを使ったプログラミング言語での授業の経験から Processing によるプログラミング課題に対して学生たちがスムーズに課題に取り組み、また、意欲的に取り組んでいたことは感じている。また、筆者らが以前から取り組んでいる POV-Ray による試みと比べると POV-Ray では CG 制作に対する過度の没頭がみられたが、Processing では 2 次元の単純な図形を扱うのでむしろプログラミングに集中できることがわかった。今のところプログラミング概念の理解という目的には POV-Ray より Processing が勝るということがいえるだろう。

3. 2. 学生による授業評価アンケート結果から

筆者が行った Processing の課題制作直後に行ったア

ンケートをまとめたものを以下に示す。

表3 学生による授業アンケート結果

プログラミング経験の有無について	
まったくない	1
履修したが理解していない	4
大体理解している	1
よく理解している	0
はじめてプログラミングを学ぶとき、C や Java に比べて Processing のほうが理解しやすいと思いますか？	
特に思わない	2
多少は理解しやすいと思う。	1
理解しやすい	3

アンケート結果を見るとプログラミングについてあまり理解していないと思われる学生のうち半数程度が Processing を理解しやすいと答えている。しかし全く理解できないと答えた学生が 1 名、逆に Processing のほうが難しいと答えた学生が 1 名いた。これはプログラミングをビジュアルにただけですぐに理解度が上がるという前提が必ずしも正しくないとも考えられる。

4. まとめ

今回の研究では筆者らは Processing がプログラミングの初歩の教育に有効に使えるということを実感することが出来た。筆者らは短期大学でデザイン系の学生に対して 15 年ほどプログラミング教育をやってきた。デザインを勉強するために入学したのになぜプログラミングなんかやらせるのかと学生に言われ、デザイナーにもプログラミングのセンスが必要だと説き、コンピュータを道具として使えることが美術系の学生と対等に渡り合える武器になると訴えてきた。同時にデザイン系の学生でも自然に理解できるプログラミングの授業のあり方を考えた結果 POV-Ray や Processing を登場させることになった。ところが筆者らの短大は 2008 年度で長い歴史に終止符を打ち、新しい学部へと生まれ変わることが決まっている。とりあえず筆者らの研究は一旦休むことになるが、新しい環境で改めて研究を再開できることを願っている。

参考文献

- [1] Built With Processing デザイン/アートのためのプログラミング入門
前川峻志、田中幸太郎著 (株ビー・エヌ・エヌ新社)

著者紹介

倉田 和夫 : 東海大学短期大学部情報・ネットワーク学科, 108-0074, 東京都港区高輪 2-3-23
島森 功 : 東海大学短期大学部非常勤講師
武蔵野美術大学・女子美術大学・日本大学非常勤講師