(正会員) 高 橋 裕 樹†

キーワード：CG (Computer Graphics)，画像処理，視覚芸術，java

1．ま え が き

Processingは，MIT (Massachusetts Institute of Technology) メディアラボに所属していたCasey ReasとBen Fryによって開発された，視覚芸術のためのプログラミング教育用に作られたJavaをベースにした開発環境である．図の描画，3次元グラフィックス，ビデオのキャプチャや再生などのプログラムを簡単に書くことができ，それらを動かすことによって，動作をすばやく確認することができる．また，Processingはオープンソースで，だれもが自由に利用することができ，Mac OS (Operating System)，Windows，GNU (General Public License) /Linux上で動作する．さらに，開発環境から簡単にJavaのアップレットやアプリケーションプログラムに変換することができる．また，CV (Computer Vision) やデータ可視化，音楽，ネットワークなどの70以上の拡張ライブラリーがあり，非常に簡単にマルチメディア処理を行うプログラムを作成することが可能である．

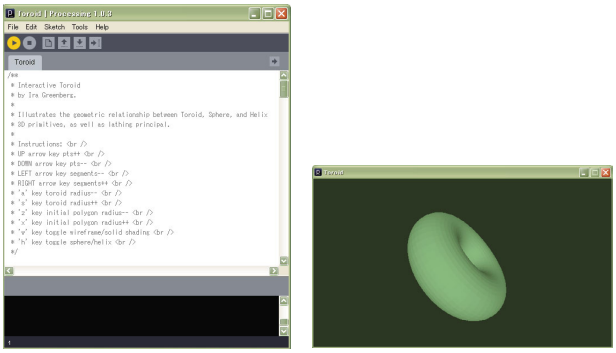
本稿では，このProcessingを用いたビジュアル情報処理の基本的な部分について概説する．2章でProcessingの準備について述べる．3〜7章で，2次元図形の描画法，アニメーション，簡単な画像処理，および，3次元CG (Computer Graphics)のプログラミングについて述べる．Processingは，プログラミングやCGの学習用としても利用されるが，本稿では，プログラミングとCG，画像処理の基礎知識があることを前提として説明する．

なお，本稿は当初2回に分けた連載企画であったが，編集の都合で1回にまとめて掲載することになったため，長文となっていることをお許しいただきたい．

2．Processingの準備

2.1 Processingのインストール

Processingは，http://processing.orgのdownloadページから，利用するOSに応じたソフトウェアをダウンロードし，解凍するだけで利用可能である．本稿では，windowsで利用することを前提として述べる．processing-1.2.1.zip*1 (約63.2MB) を適当な場所にダウンロードし，適切な解凍ソフトウェアで解凍すると利用することができる．もし，現在，利用しているwindowsにJavaがインストールされている場合には，Windows [Without Java] (processing-1.2.1-expert.zip，約23.6MB) をダウンロードしても構わない．



(a) エディタ　(b) 実行ウィンドウ

図1　プログラム実行の様子

2.2 Processing Development Environment

zipファイルを解凍すると，processing-1.2.1という名前のフォルダが生成される．そのフォルダにあるprocessing.exeをクリックして起動すると，**図1** (a) に示すProcessingのエディタウィンドウが表示される．このウィンドウが，プログラムを書いて，実行し，動作を確認するためのProcessingの開発環境となっている．

Processingの開発環境で作られたプログラムは，"sketch"と呼ばれている．新しいプログラムを作成すると，作成した日付に対応したsketchフォルダがsketchbookフォルダに作成され，拡張子pde (Processing Development Environment) が付いたファイルが保存される．sketchbookフォルダは，デフォルトで"マイドキュメント/Processing"フォルダに設定されている．メニューのFile→Preferencesで開かれるウィンドウで

† 電気通信大学　大学院情報理工学研究科　総合情報学専攻
"Processing" by Hiroki Takahashi (Department of Informatics, Graduate School of Informatics & Engineering, the University of Electro-Communications, Tokyo)

*1　2010年8月現在．

Processingの環境設定を変更することができる．一番上のSketchbook locationと表示されている場所にsketchを保存したいフォルダを指定することができる．

Processingでは，メニューのFile→Examples以下にサンプルが多数用意されている．図1(b)に3D→Form→Toroidを選択し，実行した例を示す．プログラムの実行は，ファイルメニュー下にある円に右向き▲が描かれたボタンをクリックするか，Sketch→Runメニューの選択，あるいはCtr+Rで行うことができる．ProcessingはJavaをベースにした開発環境であるため，作成したスケッチをメニューのFile→Export ApplicationでJavaのアプリケーションプログラムに，File→Exportでアップレットに変換することができる．

3．ウィンドウの表示と基本図形の描画

Processingでは，他のプログラミング言語における関数やメソッドを命令とよんでいる．Processingには，さまざまな機能を実現するための命令が用意されているが，その多くは命令の内容を英単語にしたものになっている．**プログラム3.1**は，2次元の点，線，矩形を描画するプログラムである．

Processingでは，描画することが前提となっているため，特にウィンドウの描画設定をすることなく，描画命令を指定することで，100画素×100画素のウィンドウが開き，そのウィンドウに図形を描画する．

図2（a）にプログラム3.1の実行結果を示す．Processingの座標系は，図2(b)に示すように，左上を原点とし，水平右向きがx軸の正方向，鉛直下向きがy座標の正方向となっている．

点・線・矩形の描画

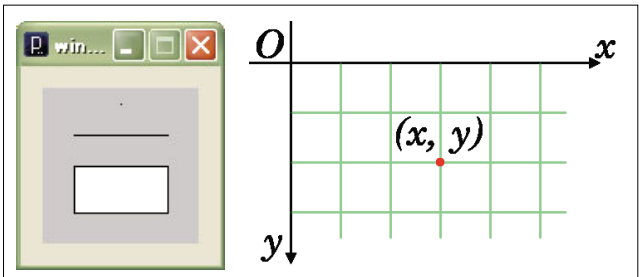
```
point(x, y); //位置(x, y)への点の描画
line(x1, y1, x2, y2);
    //(x1, y1)と(x2, y2)の線分
rect(x, y, w, h);
    //左上頂点(x, y)，w画素×h画素の矩形
```

プログラム3.2に文字列を表示するプログラムを示す．表示するウィンドウの大きさを変更するためには，size命令を用いて，ウィンドウの幅と高さを画素数で指定する．

描画は，デフォルトで，背景が灰色，図形は黒で描画されるようになっている．色の指定は，グレースケールおよび色の3原色で設定することができる．これらは，命令の引数の個数によって自動的に判別される．デフォルトで色はRGB表色系で設定するようになっており，色要素は0～255の整数値で指定する．これらの値は，背景の色を設定するbackground命令，線分の色を設定するstroke命令，塗りつぶしの色を設定するfill命令で利用する．

```
point(50, 10);
line(20, 30, 80, 30);
rect(20, 50, 60, 30);
```

プログラム3.1　基本描画(window.pde)



(a) 基本描画　　(b) 座標系

図2　ウィンドウの表示と基本図形の描画

```
size(400, 80);
background(255);
PFont font = loadFont("DejaVuSans-24.vlw");

textFont(font, 24); textAlign(CENTER, CENTER);
fill(128, 0, 234);
text("Welcome to My Class!!", width / 2, height / 2);
```

プログラム3.2　文字列の表示(text.pde)

ウィンドウの大きさの指定

size(幅，高さ);

背景色の指定

background(gray);
background(R, G, B);

文字列を表示するためには，loadFont命令でフォントをロードし，textFont命令でフォントを指定した後に，text命令でテキストを表示する．また，利用するフォントは，事前に作成する必要がある．フォントの作成は，Tools→Create Fontメニューをクリックする．利用したいフォント名と，大きさを指定し，"OK"ボタンをクリックすると，現在作成しているスケッチフォルダに"data"フォルダを作成し，フォントファイルを生成する．プログラム3.2では，DejaVuSans-24.vlwというファイルが生成される．**図3**にプログラム3.2の実行結果を示す．

図3　文字列の表示

フォント

```
PFont font; // font変数の宣言
flont=loadFont("フォントファイル") ;
    //フォントの読み込み
textFont(font); //フォントの指定
textFont(font, size);
textAlign(ALIGN); //フォントの描画位置設定
textAlign(ALIGN, YALIGN);
    // ALIGN = LEFT, CENTER, RIGHT
    // YALIGN = TOP, CENTER, BOTTOM
text("文字列", x, y); //(x, y)に文字列を描画
```

4．2次元図形の描画

本章では，国旗の描画を例に，前章よりも少し複雑な2次元図形の描画方法について説明する．**プログラム4.1**に日本の国旗*2を描画するプログラムを示す．

コンピュータで扱う画像は，画素を単位とした離散的な要素で構成されているため，斜めの線や曲線を描画する場合に正確に描画することができず凹凸が生じてしまう．このような凹凸をエイリアシングとよぶ．このエイリアシングを軽減し，目立たなくするための処理として，対応する画素や隣接する画素の色を滑らかにして表示するアンチエイリアシング処理がある．Processingには，アンチエイリアシングを行う命令としてsmoothが用意されている．プログラム4.1では，円を描画するため，3行目でsmooth命令を利用してアンチエイリアシング処理を施している．

円や楕円の描画には，ellipse命令を用いる．図形を描画する場合，デフォルトでは輪郭線が描画されるようになっている．日の丸を描画する際には，輪郭線が必要ないため，輪郭線を非表示にするnoStroke命令を円の描画の前に施す．プログラム4.1の実行結果を**図4**(a)に示す．

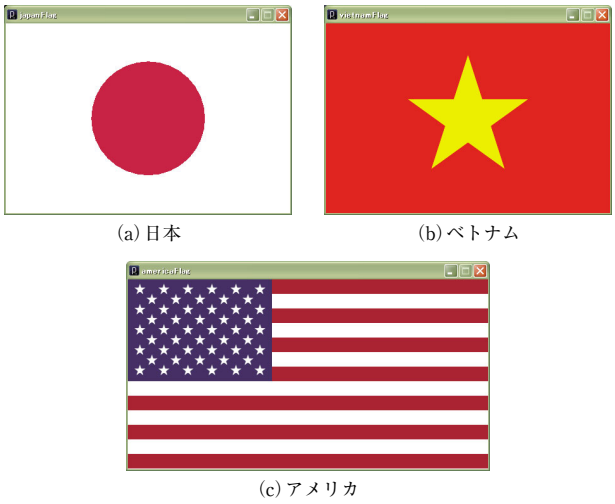
円，楕円の描画

```
ellipse(x, y, dx, dy);
    //中心(x, y)，dx, dyが横と縦の直径
```

*2　日本国旗の仕様は，http://ja.wikipedia.org/wiki/日本の国旗　を参考にした．

```
size(450, 300);
background(255); // 背景白
smooth();

ellipseMode(CENTER); // 楕円の中心を指定して描画
fill(208, 19, 69); // 塗りつぶしの色を指定
noStroke(); // 輪郭線を表示しない
ellipse(225, 150, 180, 180);
```

プログラム4.1　日本国旗(japanFlag.pde)



(a) 日本　(b) ベトナム

(c) アメリカ

図4　国旗の描画

```
void setup(){ // 初期設定
  size(450, 300);
  noStroke(); // 外枠の非表示
  smooth();
}

void fiveStar(){ // 星
  beginShape(); // 多角形の描画
  for(int i = 0; i < 5; i++){
    float theta = TWO_PI / 5 * 2 * (i + 1) + HALF_PI;
    int x = int(100 * cos(theta)) + width / 2;
    int y = -int(100 * sin(theta)) + height / 2;
    vertex(x, y); // 頂点指定
  }
  endShape();
}

void draw(){ // 描画
  background(255, 0, 0);
  fill(255, 255, 0);
  fiveStar();
}
```

プログラム4.2　ベトナム国旗(vietnamFlag.pde)

次に，多角形を描画する例として，**プログラム4.2**にベトナムの国旗を描画するプログラムを示す．

Processingでは，いくつかの処理をまとめて新たな命令を作成することもできる．Processingでプログラムの構造化を行うためには，プログラムが最初に一度だけ実行するsetup命令とプログラム実行中に何度も繰り返し呼び出され，描画を行うdraw命令を必ず作成する必要がある．それぞれの型はvoidである．

多角形の描画には，beginShape命令とendShape命令の間に，頂点の座標を指定するvertex命令を必要な頂点数だけ記述する．プログラム4.2の9行目から14行目では，一筆書きした★型の内部を塗りつぶしている．Processingでは，ウィンドウサイズをシステム変数としてwidthとheightという名前の変数に格納している．その値をプログラム4.2の11，12行目で利用している．プログラム4.2の実行結果を図4(b)に示す

多角形の描画

```
beginShape();
    vertex(頂点のx座標，頂点のy座標);
    ....
endShape();
    //閉多角形の場合endShape(CLOSE);
```

Processingでは，平行移動，回転，拡大・縮小などの幾何学的変換を行う命令も用意されている．その一例として，**プログラム4.3**にアメリカの国旗*3を描画するプログラムを示す．

アメリカの国旗では，左上のUnion内に50個の★が描画されている．プログラム4.3では，★を描画する命令を作成し，その命令を幾何学変換を施すことで50個描画している．30，31行目で描画位置を計算し，半径1の外接円に内接する★を仕様にしたがって拡大し，平行移動することによって描画している．幾何学変換は，元になっている座標系に対してある対象を描画する局所的な座標系を変換する処理であるため，複数の幾何学変換を行う場合には，変換の順番や変換の施し方に注意する必要がある．32〜34行目の処理では，★の局所的な座標系に対し，拡大・縮小を行ったあとに平行移動を施すことで，ウィンドウ座標系のある位置に★が一つ描画される．そのため，次の★を描画する際に，単純に同様な幾何学変換を施すと，その局所座標系にある★をさらに変換することになり，意図した場所に意図した大きさの★が表示されなくなる．そのため，29行目と35行目で，変換行列をスタックすることで，一つ一つの★をウィンドウ座標系に対して異なる変換行列で幾何学変換して描画している．プログラム4.3の実行例を図4(c)に示す．

*3 http://en.wikipedia.org/wiki/Flag_of_the_United_States にあるアメリカ国旗の仕様を参考にした．

```
void setup(){
  size(570, 300);
  smooth();
  noStroke();
}

void fiveStar(){ // 星
  beginShape(); // 多角形の描画
  for(int i = 0; i < 5; i++){
    float theta = TWO_PI / 5 * 2 * (i + 1) + HALF_PI;
    float x = cos(theta);
    float y = -sin(theta);
    vertex(x, y); // 頂点指定
  }
  endShape();
}

void draw(){
  background(0xff); // 白
  fill(0xb2, 0x22, 0x34); // old glory red
  for(int i = 0; i < 7; i++)
    rect(0, 2 * i * height / 13, width, 23);
  fill(0x3c, 0x3b, 0x6e); // old glory blue
  rect(0, 0, width * 2 / 5, height * 7 / 13);

  fill(0xff);
  for(int l = 0; l < 9; l++){
    for(int k = 0; k < (((l % 2) == 0) ? 6 : 5); k++){
      pushMatrix(); // 幾何変換行列のスタック
      int sx = width * 2 / 60 * (2 * k + 1 + l % 2);
      int sy = height * 7 / 130 * (l + 1);
      translate(sx, sy); // 平行移動
      scale(0.0616 / 2 * height); // 拡大・縮小
      fiveStar(); // 星の描画
      popMatrix();
    }
  }
}
```

プログラム4.3　アメリカ国旗(americaFlag.pde)

2次元幾何学変換

```
translate(xt, yt); //(xt, yt)平行移動
rotate(r); // r(rad)回転
scale(sx, sy); // x方向にsx倍，y方向にsy倍
pushMatrix(); //変換行列のプッシュ
popMatrix(); //変換行列のポップ
```

5．アニメーション

本章では，アニメーションの描画手法について説明する．**プログラム5.1**にウィンドウの壁面で反射するボールのアニメーションを行うプログラムを示す．

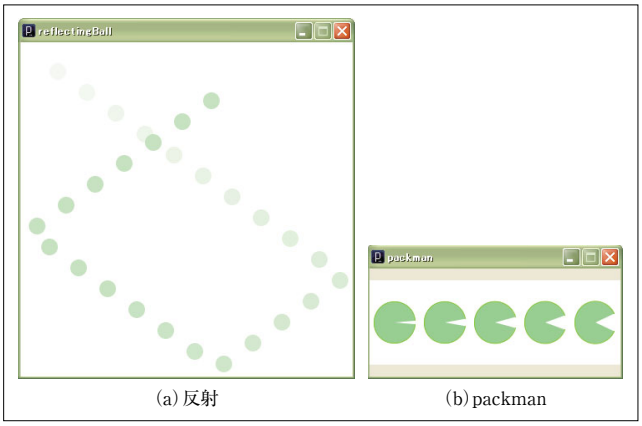
Processingでアニメーションを実現するためには，プログラムが最初に一度だけ実行されるsetup命令とプログラム実行中に何度も繰り返し呼び出されるdraw命令を用いる．draw命令を呼び出す頻度は，frameRate命令で設定する．frameRate命令の引数には，フレームレート


```
int w = 20, x = 0, y = 0, vx = 4, vy = 3;
void setup(){
  size(400, 400); smooth(); noStroke();
  ellipseMode(CORNER);
  fill(200, 255, 200);
  frameRate(30);
}

void draw(){
  background(255);
  x += vx; y += vy;
  if ((x < 0) j j (x + w > width)){
    vx *= -1;
    if (x < 0) x *= -1;
    else if (x + w > width) x = 2 * (width - w) - x;
  }
  if ((y < 0) j j (y + w > height)){
    vy *= -1;
    if (y < 0) y *= -1;
    else if (y + w > width) y = 2 * (height - w) - y;
  }
  ellipse(x, y, w, w);
}
```

プログラム5.1　ボールの反射（reactingBall.pde）

```
int angle = 0, aStep = 2;
int x, vx = 2;
void setup(){
  size(600, 100); smooth();
  stroke(154, 205, 50); fill(152, 251, 152);
  frameRate(15);
  x = height / 2;
}

void packman(int d, int t){
  arc(0, 0, d, d, radians(t), TWO PI - radians(t));
}

void draw(){
  background(255);
  angle += aStep;
  if (angle < 0 ｜｜ angle >= 15 * abs(aStep)) aStep *= -1;

  if (x < width - (height - 20) / 2) x += vx;
  else x = (height - 20) / 2;

  translate(x, height / 2);
  packman(height - 20, angle);
}
```

プログラム5.2　packman（packman.pde）



(a) 反射　　(b) packman

図5　アニメーション

(framerate) を指定する．フレームレートは，一秒間に描画する静止画の数であり，fps (Frames Per Second) という単位で表される．プログラム5.1では，6行目で30fpsを設定している．

フレームレートの設定

```
frameRate(fps); // fps(Frames Per Second)
```

アニメーションプログラムでは，フレーム毎に変化する変数は，外部変数として宣言し，draw命令内でそれらの値を変化させることで時間的な変化を実現している．また，描画処理は上書きされるので，10行目にあるbackground命令でdraw命令が呼び出されるたびに背景を初期化する必要がある．**図5** (a) にプログラム5.1の実行例を示す．ただし，印刷のためにボールの軌跡を表示している．

アニメーションと幾何学変換の関係を示す例として，**プログラム5.2**にpackmanが口を開け閉めしながら移動するプログラムを示す．

前章では，幾何学変換命令を連続して施すと，幾何学行列が連結される例を示した．しかし，アニメーションを行う際には，draw命令が呼び出される毎に，新たに幾何学変換を施すため，プログラム5.2では，単にpackmanを描画する位置 (x, height / 2) への平行移動をしている．図5 (b) にプログラムの実行例を示す．

6．簡単な画像処理

ここでは，画像処理を行う命令について説明する．**プログラム6.1**に画像を読み込んで，ディスプレイに表示するプログラムを示す．

Processingには，画像データを扱うためのクラスとしてPImageクラスが用意されている．画像の読み込みにはloadImage命令を用い，gif, jpg, tga, pngの4種類の画像フォーマットを読み込むことができる．画像は，image命令を用いて簡単にディスプレイに表示することができる．画像ファイルは，sketchフォルダ内にdataフォルダを作成し，その中に保存する．

次に，読み込んだ画像の画素値を取得し，順次ディスプレイに表示するプログラムを**プログラム6.2**に示す．

PImageでは，画像の幅や高さを保存するwidthやheightがフィールドとして用意されているとともに，画像のデータを扱う命令も複数用意されている．get命令は，

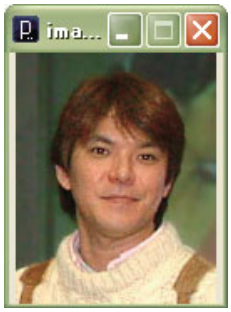
```
size(120, 150);
PImage img = loadImage("rocky.jpg");
image(img, 0, 0);
```

プログラム6.1　画像の表示（imageDisplay.pde）

画像中の位置（i, j）における画素値を取得するための命令であり，colorクラスの変数を返す．colorは色を保持するクラスであり，プログラム6.2ではRGBの値を保持している．特定のインスタンスに対する命令の呼び出しには，メンバ演算子"."を用いて実行する．**図6**（b）にプログラム6.2の実行例を示す．

```
PImage img;

void setup(){
  size(120, 150);
  background(255);
  img = loadImage("rocky.jpg");
}

void draw(){
  int i = int(random(img.width));
  int j = int(random(img.height));
  color c = img.get(i, j); // 画素(i, j) の画素値の取得
  stroke(c);
  point(i, j);
}
```

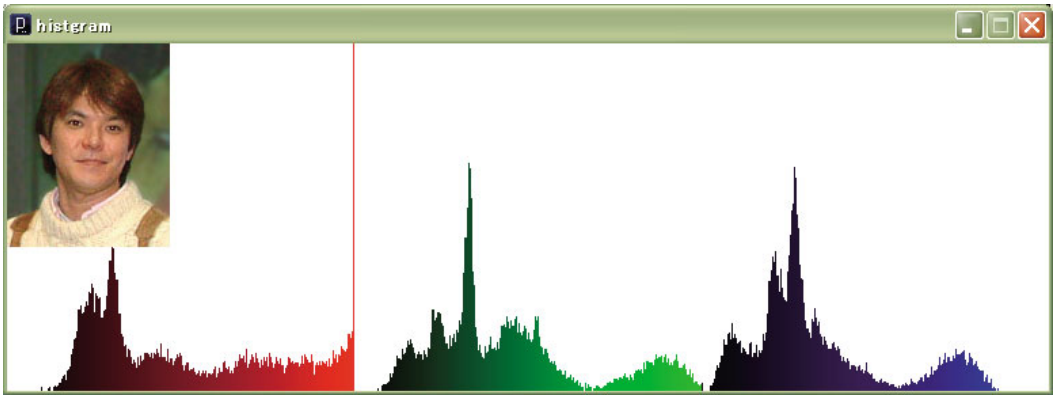
プログラム6.2　画素値の取得（pointImage.pde）



（a）画像の表示

（b）画素値の取得

（c）フィルタリング処理



（d）画像のヒストグラム

図6　簡単な画像処理


```
size(480, 150);
PImage img = loadImage("rocky.jpg");
image(img, 0, 0);
img.
img.filter(BLUR, .8); image(img, img.width, 0);
img.filter(INVERT); image(img, 2 * img.width, 0);
img.filter(INVERT);
img.filter(THRESHOLD, .7); image(img, 3 * img.width, 0);
```

プログラム6.3　フィルタリング（Iter.pde）

画像に関する命令

```
PImage img = loadImage("ファイル名");
image(img, x, y); // img を位置(x, y) に表示
img.save("ファイル名"); // img を保存
img.get(i, j); // 画素(i, j) の画素値の取得
```

プログラム6.3に，processingに用意されているフィルタリング処理の一部を用いた例を示す．filter命令では，引数に施すフィルタリング処理のmode（THRESHOLD, GRAY, INVERT, POSTERIZE, BLUR, OPAQUE, ERODE, DILATE）と必要なパラメータを指定することで，簡単にフィルタリング処理を施すことができる．プログラム6.3は，平滑化，画素の反転，および2値化を施した例であり，実行結果を図6（c）に示す．

プログラム6.2では画素値を取得する命令の例を示したが，次に画素値を配列に保存し，直接データの読み書きが行える方法について説明する．例として，画像の画素値のヒストグラムを表示するプログラムを**プログラム6.4**に示す．

画像全体の画素値を保存する配列

```
img.loadPixels(); // 画素データの取り込み
img.updatePixels(); // 画素データの更新
img.pixels[p]; // p 番目の画素値の取得
    // 画素(i, j) はp = j * img.width + i
```

PImageクラスには，画素値を保存する配列としてpixelsと名付けられた1次元配列が用意されている．pixelsのデータを扱う場合には，事前にloadPixels命令を用いて画素値データを配列に保存する必要がある．また，画素値の値を変更した場合には，変更後，updatePixels命令で画素値の変更を有効にする必要がある．pixelsは，画素値を1次元配列として扱っているため，2次元座標系で位置(i, j)にある画素は，p = j * img.width + i番目の要素に対応する．図6（d）にプログラム6.4の実行例を示す．

```
void setup(){
  size(768, 256);
  background(255);
}

void histgram(PImage img){
  img.loadPixels();
  int[ ] r = new int[256], g = new int[256], b = new int[256];
  for(int j = 0; j < img.height; j++){
    for(int i = 0; i < img.width; i++){
    int p = j * img.width + i; // 画素の一次元座標
    r[int(red(img.pixels[p]))]++; // 頻度計算
    g[int(green(img.pixels[p]))]++;
    b[int(blue(img.pixels[p]))]++;
  }
}

int max = 0;
for(int i = 0; i < 256; i++){// 正規化
  if (max < r[i]) max = r[i];
  if (max < g[i]) max = g[i];
  if (max < b[i]) max = b[i];
}

for(int i = 0; i < 256; i++){// histgram の描画
  stroke(i, 0, 0);
  line(i, height, i, height < 256 * r[i] / max);
  stroke(0, i, 0);
  line(i + 256, height, i + 256, height - 256 * g[i] / max);
  stroke(0, 0, i);
  line(i + 512, height, i + 512, height - 256 * b[i] / max);
  }
}

void draw(){
  PImage img = loadImage("rocky.jpg");
  image(img, 0, 0);
  histgram(img);
}
```

プログラム6.4　ヒストグラム（histgram.pde）

```
void setup(){
  size(400, 200, P3D); // 3 次元描画
  background(255);
}

void draw(){
  noFill(); // wireframe model
  translate(110, 100, 0); sphere(65);

  lights(); fill(196); noStroke(); // surface model
  translate(180, 0, 0); sphere(65);
}
```

プログラム7.1　3次元モデル（model3D.pde）

7．3次元CGの描画

続いて，本章では，3次元CGを描画する方法について簡単に説明する．**プログラム7.1**にワイヤフレームモデルとサーフェスモデルの球を描画する例を示す．3次元モデルを描画するためには，size命令の3番目の引数にP3Dを指定する．ワイヤフレームモデルを描画するためには，noFill命令で面の描画をせずに，稜線だけ表示するようにする．一方，サーフェスモデルの描画には，noStroke命

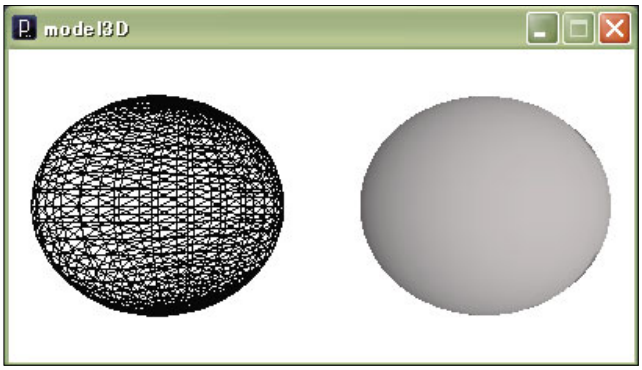
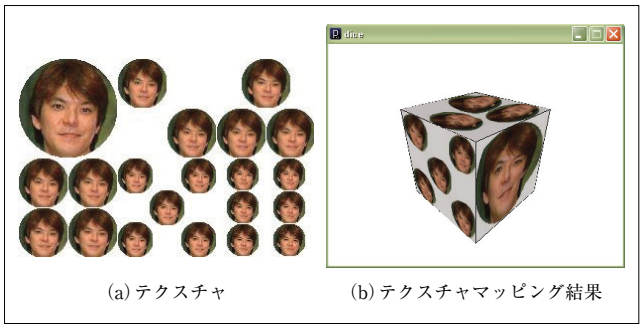


図7　3次元モデル



(a) テクスチャ　　(b) テクスチャマッピング結果

図8　テクスチャマッピング

```
PImage img;
void setup(){
  size(400, 300, P3D);
        I mg = loadImage("diceTexture.jpg");
}

void texturedCube(){
  pushMatrix();
    scale(.5, .5, .5);
    beginShape(QUADS);
      textureMode(IMAGE);
      texture(img);
      vertex(-1, -1, 1, 0, 0); vertex(-1, 1, 1, 0, 100);
        vertex(1, 1, 1, 100, 100); vertex(1, -1, 1, 100, 0);
      vertex(1, -1, 1, 100, 0); vertex(1, 1, 1, 100, 100);
        vertex(1, 1, -1, 200, 100); vertex(1, -1, -1, 200, 0);
      vertex(1, -1, -1, 200, 100); vertex(1, 1, -1, 200, 200);
        vertex(-1, 1, -1, 300, 200); vertex(-1, -1, -1, 300, 100);
      vertex(-1, -1, -1, 100, 100); vertex(-1, 1, -1, 100, 200);
        vertex(-1, 1, 1, 200, 200); vertex(-1, -1, 1, 200, 100); 20
      vertex(-1, -1, 1, 200, 0); vertex(1, -1, 1, 200, 100);
        vertex(1,-1, -1, 300, 100); vertex(-1, -1,-1, 300, 0);
      vertex(1, 1, 1, 0, 100); vertex(-1, 1, 1, 0, 200);
        vertex(-1, 1, -1, 100, 200); vertex(1, 1, -1, 100, 100);
    endShape();
  popMatrix();
}

void draw(){
  background(255); 30
  lights();
  translate(200, 150, -50);
  rotateX(map(mouseY, 0, height, PI, -PI));
  rotateY(map(mouseX, 0, width, -PI, PI));
  scale(150, 150, 150);
  texturedCube();
}
```

プログラム7.2　テクスチャマッピング（dice.pde）

令で稜線の描画をやめ，fill命令で面の色を指定する．また，3次元モデルのレンダリングでは，光源を指定することで，shadingが施され，立体らしさを表現することができる．デフォルトの光源の指定には，lights命令を用いる．lights命令では，環境光（ambientLights（128，128，128）），平行光（directionalLight（128，128，128，0，0，－1）），光源の減衰率（lightFalloff（1.0，0，0）），光源の鏡面反射（lightSpecular（0，0，0））となっている．

3次元モデルのプリミティブとしては，球を描画するsphere命令，直方体を描画するbox命令がある．プログラム7.1の実行結果を**図7**に示す．

プログラム7.2にテクスチャマッピングを行うプログラムを示す．テクスチャの指定には，texture命令を用いる．テクスチャマッピングは，作成した立体の各頂点の座標とテクスチャ画像の座標をそれぞれ対応付けることによって行うことができる．Processingでは左手デカルト座標系を採用しており，3次元描画の際には，$\left(\frac{\mathrm{width}}{2}, \frac{\mathrm{height}}{2}, \frac{\mathrm{height}}{2\tan\pi/6}\right)$にカメラが設置され，$\left(\frac{\mathrm{width}}{2}, \frac{\mathrm{height}}{2}, 0\right)$の方向を向いている．Processingでは，マウスポインタの位置をmouseX，mouseYという名前のシステム変数に保存しているため，プログラム7.2では，33，34行目でマウスポインタの座標値をmap命令で$-\pi$とπの間の角度に写像して，マウス操作で立方体を回転するようにしている．プログラム7.2で用いたテクスチャを**図8**（a）に，テクスチャマッピング結果を図8（b）に示す．

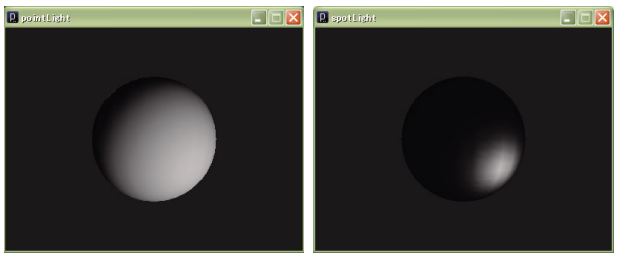
テクスチャマッピングの方法

```
PImage img = loadImage("テクスチャ画像");
beginShape();
    texture(img); // テクスチャの指定
    textureMode(NORMALIZED);
        // テクスチャ画像の座標の指定方法
        // NORMALIZED: 0-1 の範囲
        // IMAGE: 画像オリジナルの座標
    vertex(x, y, z, u, v);
        // 3 次元モデル上の点(x, y, z) と
        // テクスチャ上の点(u, v) が対応
    ...
endShape();
```

最後に，平行光源，点光源，スポットライトを設定するプログラムを**プログラム7.3**に示す．directional命令，

```
final int DLIGHT = 0, PLIGHT = 1, SLIGHT = 2;
int light status = DLIGHT;

void setup(){
  size(400, 300, P3D);
  noStroke();
  fill(192);
}

void draw(){
  background(32);
  foat theta = map(mouseY, 0, height, -PI / 2, PI / 2);
  foat phi = map(mouseX, 0, width, -PI / 2, PI / 2);
  foat x = 500 * sin(phi), y = 500 * sin(theta),
    z = 500 * cos(phi) * cos(theta);

  translate(width / 2, height / 2, 0);
  switch(light status){
    case DLIGHT:
      directionalLight(255, 255, 255, -x, -y, -z); 20
      break;
    case PLIGHT:
      pointLight(255, 255, 255, x, y, z);
      break;
    case SLIGHT:
      spotLight(255, 255, 255, x, y, z, -x, -y, -z, PI / 2, 500);
      break;
  }
  sphere(80);
}

void keyPressed(){
  switch(key){
    case 'd': light status = DLIGHT; break;
    case 'p': light status = PLIGHT; break;
    case 's': light status = SLIGHT; break;
  }
}
```

プログラム7.3　光源（lighting.pde）



(a) 点光源　　(b) スポットライト

図9　光　源

pointLight命令，spotLight命令でそれぞれの光源を容易に変更することができる．プログラム7.3の実行例を**図9**に示す．

光源の設定

```
directionalLight(r, g, b, nx, ny, nz);
    // 色(r, g, b) で方法(nx, ny, nz) の平行光源
pointLight(r, g, b, x, y, z);
    // 色(r, g, b) で位置(x, y, z) の点光源
spotLight(r, g, b, x, y, z, nx, ny, nz,
    angle, concentration);
    // 位置(x, y, z) に色(r, g, b) で方向(nx, ny, nz)
    // のスポットライト
    // angle: スポットライトの円錐の角度
    // concentration: 光の集中度
```

8. むすび

本稿では，視覚芸術のためのプログラム開発環境であるProcessingの基本的な使い方について概説した．Processingには，70以上もの拡張ライブラリーがあり，音やネットワークなども利用したプログラムを比較的簡単に作成することができる．これらを組合せることによって，さらに多様なプログラムを作成することができるので，是非挑戦して頂けると幸いである．

（2010年10月12日受付）

高橋（たかはし）　裕樹（ひろき）　1990年，東京工業大学工学部制御工学科卒業．1992年，同大大学院博士前期課程（物理情報工学専攻）了．1994年，同大大学院博士後期課程中退．同年，同大工学部情報工学科助手．同大大学院情報理工学研究科助手，電気通信大学電気通信学部人間コミュニケーション学科准教授を経て，現在，同大大学院情報理工学研究科総合情報学専攻准教授．画像処理，パターン認識，CGなどの研究に従事．博士（工学）．正会員．