Media Programing 1 - Processing

- proc.06 - 2009.06.09 hrr

今回のテーマ

座標変換による図形の操作

音声処理

座標変換（アフィン変換）による図形移動

画像処理とは？

OpenGLなど3D系での描画では行列演算によって座標変換を行う事で、
移動や回転などの配置や動きの制御を指定するのが一般的である。
例えば、ある座標に描画するにはtranslateを使って座標を移動し、
その場所で描画するという処理を行う。このとき、座標系に位置の変更を行う
変換行列をかけることで座標系全体が平行移動している。

一度translateすると座標系全体が移動したと考えることになるため、
移動した場所を原点と考えて、さらにtranslateをする必要がある。
例えば、(x,y) と（x+10, y)にrectを描画するには、

```
translate(x,y);
rect(0, 0, 10, 10);
translate(10,0);
rect(0,0, 10, 10);
```

とする必要がある。
このとき、新しい原点は、元々の座標での(x+10,y)になっている。

rotate()では座標系の回転を行う。同じように座標系全体が回転するので
translateとの組み合わせでは、よく考えて行う必要がある。

ちなみに、このような変換は変換行列を使った座標系の変換、
いわゆるアフィン変換と呼ばれる物である。
変換は、4x4の行列（３次元のため）によって表すことができ、
この座標の演算によって、空間上の移動や回転、拡大縮小が表現できる。
プログラム内でも実際に行列によって変換が保持されている。

なお、今回は2Dで処理をしているため、translateはx,y 、
rotateは、平面上の回転（3Dでのz軸周りの回転）になっている。
3Dでは、x,y,zになり、回転も回転軸を指定して行う必要がある。

以下の２つの例を実行してみよう。
別々のプログラムにしないと予期しない結果となるので注意。

```
translate(100, 100);
rotate(PI/4.0);
rect(0, 0, 10, 10);
```


```
rotate(PI/4.0);
translate(100, 100);
rect(0, 0, 10, 10);
```

試しにこの２つのプログラムを一つのプログラムで行うと
２つ目のrectが見えなくなってしまう。
これは、前半部分で、原点が元々の(100,100)に移った上、
さらに45度回転しているため、その状態でさらにrotateとtranslateを
行ったために、このような結果となったためである。
size(400,400)にすると微かにみえる。
先頭部分に translate(100,100) を加えて全体を100,100移動すると
全体がみえるようになる。

このように座標変換が累積してしまうと思ったように描画できない。
そのために、pushMatrix(), popMatrix()を利用する。
push、popとは座標系の状態を示す変換行列を一時的に保管する操作と
保管した物を呼び出す操作を示す言葉である。
保管も取り出しもスタック状になっているので、pushするたびに上に積まれ、
popすると上から取り出される。
スタック構造なので、荷物を積み上げていくのと同じように、一番最後に
積んだ物から順番に積みおろされる。
一番最後にpushしたものが、popすると取り出される。
スタック構造にするとpush/popの入れ子構造が実現しやすい。
これを使うと、push,popで囲まれた部分で座標変換を行っても
pushするまえの変換行列に戻すことができる。

```
    pushMatrix();                   // 変換行列の保存（積み上げ）
    translate(100, 100);            // 変換行列の変更
    rotate(PI/4.0);
    rect(0, 0, 10, 10);
    popMatrix();                    // 変換行列を元に戻す（積み降ろし）

    rotate(PI/4.0);
    translate(100, 100);
    rect(0, 0, 10, 10);
```

とすれば、意図した結果が得られるようになる。
さらに描画処理を続けるなら、後半部分もpush, popで囲む必要がある。

なお、座標変換の順番を考えるとき、
・座標系が変化すると考えると、順番通りになる
・図形に対し適応すると考えると、逆順になる。
とするとわかりやすい。

練習問題１：

rotateを使って、四角が同じ場所でグルグル回るアニメーションを作成する。
rectMode(CENTER)を使うと簡単。

練習問題２：

translateとrotateを使って、(100,100)の座業を中心に回転するアニメーションを作成する。

授業内課題１：

太陽の周りを地球が回り、地球の周りを月が回るアニメーションを作成する。
実際のスケールは考慮する必要はない。

授業内課題２：

太陽の周りを違う周期で回転するもう１つの惑星を追加する。
pushMatrix(), popMatrix() で２つの動きを分けると良い。

授業内課題３：

これまで使ってきたAreaクラスに、四角を回転させて表示するために
rotR(), rotL()メソッドを追加する。

追加したメソッドを利用して、キーボード操作で回転するプログラムを作成する。

音声処理

標準ライブラリに入っているMiminを使う。
Minim では、音声ファイルの再生、音声の録音、波形の描画、
スペクトラムの表示、音声合成など、かなりのことが行えます。
ライブラリのドキュメントにサンプルがありますので、これを参考にすれば、
基本的なことはすぐできます。
http://code.compartmental.net/
このドキュメントには、processingのLibraryのページから辿れます。

まずは、サンプルに従って、音声ファイルの再生をしてみましょう。
Processingのメニューから、Sketch/Import Libraryを選びプルダウンメニューから
minim を選択すると、必要なライブラリのimport記述がされます。

音声ファイルは、スケッチがあるフォルダにdataフォルダを作成して、その
フォルダに入れる。フォントの場合と同様。
サンプル用のファイルはad1にあります。
この曲は、卒業生の林君がCCライセンスでOpsoundで公開しているものです。

ここで、stop()メソッドに注意。stop()メソッドは、setup()やdraw()と同様に
processingであらかじめ定義されているメソッドです。
通常は、プログラムの停止時（Processingから"Stop"ボタンを押すか、実行している
ウインドウを閉じるとき）に勝手に呼ばれている。
minimではプログラム終了時にclose()処理をして、使用してたメモリなどの
リソースを解放する処理があるため、再定義（オーバーライド）している。

```
import ddf.minim.*;

Minim minim;
AudioPlayer player;
AudioInput input;

void setup()
{
  size(100, 100);
```

```
  // minimオブジェクトの生成
  minim = new Minim(this);

  // playerオブジェクトを音声ファイルを指定して生成
  player = minim.loadFile("clicker.mp3");

  //  playerオブジェクトを再生
  player.play();
}

void draw()
{
}

void stop()
{
  // ロードしたプレイヤーオブジェクトをクローズして解放。
  player.close();
  // minimオブジェクトの停止処理
  minim.stop();

  // これまで stop()メソッドは使ってこなかったが、実は勝手に呼ばれていた。
  //  今回は、minimにより大量のメモリなどを使ったものを解放するため、
  // 終了時にclose処理を行うため、自分で定義している。
  // 自分で定義したために勝手に呼ばれなくなったので、明示的に呼んでいる。
  // オブジェクト指向の例題でやったように、super.stop()で、親クラスの
  // stopメソッドを呼んでいる。
  super.stop();
}

void mousePressed(){
  if(player.isPlaying()){
    player.pause();
  } else {
    player.start();
  }
}
```

練習問題３：

MinimのドキュメントのQuickstart Guideを見ながら、上のサンプルに
波形表示を追加する。

コンピュータで扱う音声とは何か？

音声は空気の振動によって発せられ、聴く事ができる。
例えば、440hzの音では一秒間に440回振動していることになる。
振動の最大値と最小値を各々１と-1で表す事にすると、１と－１が一秒間に
440回繰り返し再生すれば440Hzの音として聞こえるようになる。
これは、440Hzの矩形波になる。
これをコンピュータ上で扱うには、配列を使えば良い事は簡単に想像る。
１と－１の繰り返しが440回だから、880個の配列（矩形波なら整数の配列）が
あれば、440Hzの矩形波を表す事ができることになる。
しかし、440Hzのサイン波だったら1から-1までsinを使って徐々に変化するため
880個の大きさの配列では足りない。
ここで、サンプリングレートが重要になる。
サンプリングレート（サンプルレート）とは一秒間を何個の値で示すか
どの程度短い時間間隔で音を表現するかを示したものである。
CDでは44100Hzが採用されている。矩形波の例でわかるように44100Hzの
サンプリングレートで表現できる最大周波数は22050Hzとなっている。
音声は通常はサイン波などに見られるように連続的な変化をするので、
サンプリングレートは、音のスムーズさを示してもいる。
さらに、音の大きさの何ビットで表現するかも重要な要素となっている。
8bitでは、256段階で1から-1の値を表現しているため、波形がガタガタになって
しまう。人間の声程度なら十分に判別できるが、音楽には向いていない。
CDでは、16bitつまり、65536段階で1から-1を表現している。

このように、コンピュータ上では整数の配列を用いて、
一秒間の音量の変化をサンプリングレートの回数で表現している。
これらを連続的に行う事で音として扱っている。
16bit 48kHz(44000Hz)の音を10秒間分示すためには、
2byte * 44000 * 10 = 880000
88万バイト（= 880kバイト）のメモリが必要となる。

これらのメモリを常に確保するのは、大変なので、音声処理は
バッファを確保して、少し読み込んでは再生するというような

処理を繰り返すことで実現しているのが一般的である。

練習問題４：

ドキュメントのOsillatorを見ながら、マウスの位置によってサイン波と
のこぎり波の周波数を変化させながら音を合成するプログラムを作成する。

授業内課題４：

Areaクラスを拡張して、音声ファイルを読み込むメソッド setSound()
音を再生する playSound()メソッドを追加し、壁に当たったときなどに
音を出すようなプログラムを作成する。

最終課題：

各自、自分で決めた課題を実現し、6/16(火)14:00からC2で発表する。

来週は、４限目からビジネスマナー講座があり、ちょうど３限が空いているので、
この時間に発表を行う。

なお、最終課題は、事前にad1にもあげておく事。