Media Programing 1 - openFrameworks
- frame.06 - 2010.06.01 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

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ofTranslate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ofTranslate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、

```
ofTranslate(x,y,0);
ofRect(0, 0, 10, 10);
ofTranslate(10,0);
ofRect(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。

ofRotate()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
ofTranslate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。
ここでは、まずは2次元で処理をしているため、ofTranslateのz軸は0、
ofRotateでは、z軸を1.0にして（またはofRotateZを使って）、

平面上の回転（3Dでのz軸周りの回転）になっている。

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。

```
---
void testApp::draw(){

   ofTranslate(100, 100, 0);
   ofRotate(45, 0, 0, 1);
   ofRect(0, 0, 30, 30);

}

----
void testApp::draw(){

   ofRotateZ(45);
   ofTranslate(100, 100, 0);
   ofRect(0, 0, 30, 30);

}

----
```

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。
先頭部分に ofTranslate(100,100,0) 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、ofPushMatrix(), ofPopMatrix()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。

```
void testApp::draw(){

   ofPushMatrix();
   ofTranslate(100, 100, 0);
   ofRotate(45, 0, 0, 1);
   ofRect(0, 0, 30, 30);
   ofPopMatrix();

// ofPushMatrix();
   ofRotateZ(45);
   ofTranslate(100, 100, 0);
   ofRect(0, 0, 30, 30);
// ofPopMatrix();

}
```

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：
ofRotate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。
ofSetRectMode(OF_RECTMODE_CENTER)を使うと簡単。

練習問題２：
ofTranslateとofRotate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。
ofPushMatrix(), ofPopMatrix() で２つの動きを分けると良い。
ちなみに、z軸以外を回転軸にすると円（球ではない）が前後方法に回るようになる。
ofRotateX() あるいは、ofRotate(angle, 1, 0, 0)　などを使う。

授業内課題３：
これまで使ってきたZukeiクラスに、回転させて表示するためのrotR(), rotL()という
２つのメソッドを追加する。
追加したメソッドを利用して、Sikakuクラスのインスタンスを
キーボード操作で回転するプログラムを作成する。
なお、これに合わせて、SikakuクラスのpaintメソッドをofTranslateとofRotateを使った
ものに変更する必要がある。

OpenGLとの連携
openFrameworksでは、OpenGLをそのまま使う事もできる。
OpenGLを使うと様々な3D表示を行えるようになるので、必要に応じて
直接OpenGLを呼ぶと良い。

例えば、以下のようにOpenGLのglut（ライブラリ）を使って、球や立方体を
描画できる。
なお、OpenGLでは、Lightの位置や視点の設定ができるので、これらをうまく
やらないと思ったように表示できない。

```
void testApp::draw(){

   glDisable(GL_LIGHTING);

   sikaku.paint();

   float lightpos[] = {200,200,-150,0};

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glLightfv(GL_LIGHT0, GL_POSITION, lightpos);

   ofTranslate(100, 100, 0);
   glutSolidCube(30);
   glTranslatef(130, 130, 50);
   glutSolidSphere(50, 30, 30);

}
```

また、addonsにあるofxVectorMathを使っての描画や、ofx3dsModelLoaderで
3ds形式のオブジェクトを読み込む事ができる。
詳細は、サンプルプログラムを各自参照すること。

音声処理

openFrameworksでは、標準でofSoundPlayerクラスを使って、音の再生ができる。
対応している圧縮形式は、.wav, .aif, .mp3, .mp2, .ogg .raw　となっている。
また、音声ストリームを入出力するための仕組みとして、
ofSoundStreamSetup() でセットアップし、
void testApp::audioReceived(){} で入力された音声を処理し、
void testApp::audioRequested(){} で音声データを再生する
という方法もある。
この手法については、音声データをそのまま扱うため、それなりの知識が必要となるが
音を生成できるので音声のシンセサイズができる。

まずは、既存の音声ファイルを手軽に扱える ofSoundPlayerを利用する。
その前に音声ファイルを用意する必要がある。
卒業生が作成したCreativeCommonsライセンスの音声として
OpsoundにあるYOUSUKE HAYASHI の作品を利用する。
http://www.opsound.org/artist/yosukehayashi/
ここから適当な曲をダウンロードし、
作成したプロジェクトのbin/data フォルダに置く。
#配布したテンプレートを使うと、音声ファイルを正しい場所に置いても
#認識されない事があります。
#その場合は、プロジェクトの上の階層にdataフォルダを移動してみるか、
#openframeworks.jp で配布しているテンプレートを使ってください。

testApp.h へ追加
--

```
	ofSoundPlayer player;
	float playpos;
	bool isPlaying;
```

---

testApp.cpp

```
void testApp::setup(){
```

```
    ofSetWindowShape(400, 400);
    ofBackground(0, 0, 0);
    ofSetVerticalSync(true);
    ofSetFrameRate(30);

    player.loadSound("scrapper.mp3"); //サウンドファイルのロード
    player.setVolume(0.75);         // 音量の設定

    isPlaying = false;
    playpos = 0.0;

}

//--------------------------------------------------------------
void testApp::draw(){

    if(isPlaying){
        ofDrawBitmapString("Playing Now", 50, 100);
    } else {
        ofDrawBitmapString("Play Stopped", 50, 100);
    }

}
void testApp::mousePressed(int x, int y, int button){

    if(isPlaying){                      //再生中なら
        playpos = player.getPosition();   //再生位置を保存
        player.stop();                  // ストップ
        isPlaying = !isPlaying;
    } else {                            // 再生中じゃなかったら
        player.play();                  // 再生
        player.setPosition(playpos);    // 前回止めた場所に移動
        isPlaying = !isPlaying;
    }
}
```

練習問題３：
setPan()やsetSpeed()を利用して、マウスをドラッグするとパンやスピードが変化するようにしてみる。

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。

testApp.h へ追加

```
	void windowResized(int w, int h);
	void audioRequested(float *output, int bufferSize, int
```

```
nChannels);
      float samplingRate;
      float amplitude;
      float phase;
      float pan;
      float frequency;
      float mx, my;
```

testApp.cpp

```
void testApp::setup(){
   ofSetWindowShape(400, 400);
   ofBackground(0, 0, 0);
   ofSetVerticalSync(true);
   ofSetFrameRate(30);

   samplingRate = 44100;
   amplitude = 0.8;
   phase = 0;
   pan = 0.5;
   frequency = 880 * 2;

   ofSoundStreamSetup(2, 0, this);
}

void testApp::mousePressed(int x, int y, int button){
   mx = mouseX;
   my = mouseY;
}

void testApp::audioRequested(float *output, int bufferSize, int
nChannels){
   float sample;
   float phaseDiff;

   pan = mx / ofGetWidth() - 0.5;
   phaseDiff = TWO_PI * frequency / samplingRate;

   for(int i = 0; i < bufferSize; i++){
      phase += phaseDiff;
      if(phase > TWO_PI){
         phase -= TWO_PI;
      }
      sample = sin(phase);

      output[i*nChannels] = sample * pan * amplitude;
```

```
      output[i+nChannels+1] = sample * (1 -pan) * amplitude;
   }
}
```

練習問題４：
amplitudeの値も、時間に応じて一定の周波数で変化するようにする。
元の正弦波に変調を加えて新しい音にする。AM変調。
また、FM変調的なものにも挑戦してみる。

最終課題：

各自、自分で決めた課題を実現し、6/8(火)14:00からC2で発表する。
なお、最終課題は、事前にfsにもあげておく事。

どっかで一回補修やる？