

チュートリアル

スマートフォンと呼ばれる携帯型端末の普及は目覚ましいものがあり、アプリと呼ばれるスマートフォン向けアプリケーションも、日々膨大な数が世界中でリリースされています。今回は、代表的な2つのスマートフォン用OS上でのソフトウェア開発に関するチュートリアル(全4回シリーズ)の第2回目として、東洋大学の塩谷隆二先生と中林靖先生に解説していただきます。なお、チュートリアル記事は1ページ目のみを本誌掲載し、続きは日本計算工学会HP上で公開していますので、そちらも併せてご参照ください。

スマートフォンアプリ開発入門 - iOS vs. Android (2)

塩谷 隆二
中林 靖

1 はじめに

前号のチュートリアルで「スマートフォンアプリ開発入門」第1回目として、iOSとAndroidそれぞれの開発環境構築手法について解説した。iOS用のアプリ開発には、まずMac (Mac OS X 10.6以降)が必須であり、その上でXcodeと呼ばれる統合開発ツールを用いてObjective-C言語によるアプリ開発環境が構築出来る。一方、Android用のアプリ開発環境としては、広くJavaやC言語等のアプリケーション開発に用いられている統合開発環境Eclipseを用いて、その上にAndroid Development Tool (ADT)やAndroid SDKなどを追加インストールすることで、Java言語によるアプリ開発環境が構築出来る。

ここで、前号で紹介した開発環境以外にもスマートフォンアプリの開発手法があるので紹介しておく。まず、Android用アプリ開発用ツールとしては、Google Labsで公開されている「App Inventor」というWebベースのアプリ開発ツールがある。App InventorはEclipseも

Javaも用いずに、Web上でパズルのように部品を組み立てていくイメージでAndroidアプリを開発するツールで、もともとはプログラミングを専門的に学んだことがない学生がプログラムの作り方の基礎を学ぶために作られたもので、米国では教育機関で広く利用されている。

また、JavaScriptやC#で記述されたインタラクティブな3DアプリケーションをiOS/Androidの両方用に変換する「Unity」、HTMLとJavaScriptで記述されたWebアプリケーションをiOS/Android用に変換する「Titanium」、HTML5で記述されたコンテンツをiOS/Android用に変換する「BPR」など、iOSとAndroidの垣根を超えたアプリ開発環境も次々と開発されている。これらのツールの詳細についても、次号以降で紹介していきたい。

本号では、チュートリアルの第2回目として、前号で構築した開発環境の動作確認も兼ね、iOSとAndroidそれぞれについて基礎的なアプリを実際に開発することを目的とし解説する。例えば、スマートフォンの入力方法として最も良く用いられる、画面を直接触れるタッチ操作に反応して、何らかの図形を表示するといったシンプルなものであるが、本格的なアプリ開発の準備段階として、また、スマートフォンアプリの基本的な開発方法を理解する上では重要なものである。

次章でまず、Androidで簡単な図形の表示とタッチイベントの処理方法について、次に、iOSでOpenGLによる図形の描画とタッチイベントの処理方法についてそれぞれ解説していく。

続きはWebで

日本計算工学会誌「計算工学 (Vol.17, No.1)」HP:
<http://www.jscs.org/Issue/Journal/>

筆者紹介



しおや りゅうじ

1996年東京大学大学院工学系研究科博士課程修了、東京大学助手、九州大学准教授を経て、現在、東洋大学総合情報学部教授。超並列計算機による数値解析、大規模CAEソフト「ADVENTURE」システム開発などの研究に従事。



なかばやし やすし

1999年東京大学大学院工学系研究科博士課程修了、東京大学リサーチ・アソシエイト、東洋大学工学部講師を経て、現在、東洋大学総合情報学部准教授。数値流体力学、逆問題・最適化、モバイル・ユビキタスコンピューティングなどの研究に従事。

2 Android アプリ開発

(1) Android アプリの基本構造

Android アプリを開発するためには、まず、Android アプリの基本的な構造を理解する必要がある。本稿の読者は計算工学分野の様々なアプリケーションを開発した経験があると思われるが、その際に重要な点はデータ構造とアルゴリズムである。データ構造とアルゴリズムを問題に応じて適切に選択し、構造化プログラミングまたはオブジェクト指向プログラミングを行うのが通常の方法である。一方、グラフィカル・ユーザ・インターフェイス(GUI)を搭載したアプリケーションは、キーボードやマウス等からのイベント入力を持つループを持ち、所謂、イベントドリブン(event-driven)型のプログラム構造を持つ。

Android アプリの場合には、一般的な GUI アプリと同様にイベントドリブンなプログラム構造を持つが、更に重要な概念として「Activity」がある。Activity とは簡単に言うと、スマートフォンやタブレット PC に表示される画面の単位のことであり、Java 言語における Canvas や Panel に近いものであるが、単に画像情報としての画面ではなく、現在表示されているアプリの画面に付随する様々な属性情報を持ったクラスとして定義されている。

従って、Android アプリには通常一つ以上の Activity クラス、または Activity クラスから派生したクラスが存在し、この Activity の状態遷移をベースにアプリの構造を設計する。図 1 に Activity の状態遷移を示す。

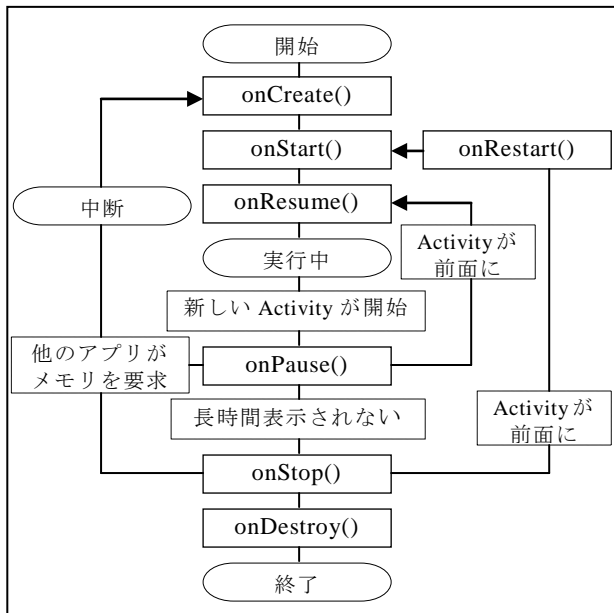


図 1 Activity の状態遷移

Android SDK で提供されている Activity クラスでは、図 1 の中で示された各メソッドが定義されており、ユーザはこれら全てのメソッドを必ずしも再定義する必要はないが、必要に応じてこれらのメソッドを override していく。

Android アプリは上述の Activity の他に、「ブロードキャストレシーバ」、「サービス」、「コンテンツプロバイダ」の合計 4 つの構成要素からなり、この 4 つの構成要素のうち一つ以上から成り立っている。本稿では Activity 以外の構成要素については割愛する。

(2) サンプル 1—グラフィックスの描画

最も簡単なサンプルを用いて Android アプリの基本的な作成方法を説明する。図 2 に示すリストは、白地の背景に赤の塗り潰しの円を表示するだけの簡単なアプリである。

```

package circle.android;

import android.app.Activity;
import android.content.Context;
import android.graphics.*;
import android.os.Bundle;
import android.view.View;

public class TestCircle extends Activity {
    //アプリの初期化
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        MyCircleView view =
            new MyCircleView(getApplication());
        setContentView(view);
    }
}

//グラフィックスの描画
class MyCircleView extends View {
    public MyCircleView(Context context) {
        super(context);
        setFocusable(true);
    }
    //描画
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        //円の描画
        canvas.drawColor(Color.WHITE);
        Paint paint = new Paint();
        paint.setAntiAlias(true);
        //円の塗り潰し
        paint.setStyle(Paint.Style.FILL);
        paint.setColor(Color.RED);
        canvas.drawCircle(150, 200, 100, paint);
    }
}

```

図 2 サンプルプログラム 1 (TestCircle)

このサンプルプログラムを作成するには、前号で解説した eclipse のファイルメニューから、新規 android プロジェクトを選択し、以下の各項目を入力する。

プロジェクト名 : TestCircle
 ビルド・ターゲット : Android 2.2
 アプリケーション名 : TestCircle
 パッケージ名 : circle.android
 アクティビティーの作成 : TestCircle

続いて、パッケージ・エクスプローラから TestCircle → src → TestCircle.java を選択し、図 2 のリストを入力する。入力後、eclipse の実行メニューから、実行 → 実行構成を選択し、プロジェクト欄の TestCircle を選択し実行ボタンのクリックにより、エミュレータが起動し、図 3 に示す実行結果が表示される。

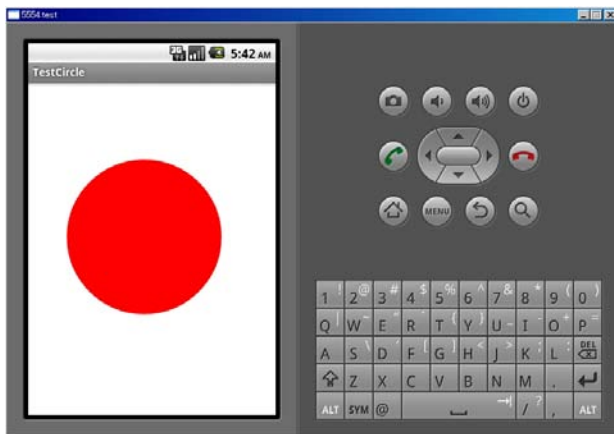


図 3 TestCircle の実行結果

続いて、図 2 のサンプルプログラムについて解説する。1 行目は android プロジェクトを作成した際に指定したパッケージ名である。他のプログラムから呼び出される際に、名称が重ならないように、開発者の PC のドメイン名等を付加することも多いが、本稿では簡単な名前のみとする。続いて、このプログラム中で必要となるクラスライブラリを import する。どのクラスライブラリが必要となるかは、マニュアルで確認する必要があるが、android.app.Activity と android.os.Bundle は Android アプリに必須のものであり、eclipse でプロジェクトを立ち上げた際にデフォルトで import されている。

ここからがプログラムの本体であり、まず、Activity クラスを継承し、TestCircle クラスを作成する。TestCircle クラスで最初に記述することは、図 1 で解説した Activity に関するメソッドのうち、最初に呼ばれる onCreate() を override することである。このサンプルプログラムでは TestCircle クラスが onCreate() された際に、MyCircleView クラスのインスタンスが作成され、TestCircle クラスの content として表示される。

次に、MyCircleView クラスについて解説する。このクラスは、View クラスのサブクラスとして定義されており、setFocusable(true) の時点で onDraw() メソッドが呼び出され、canvas に指定されたグラフィックスを表示するものである。具体的には、プログラム中の「// 円の描画」や「// 円の塗り潰し」以下に記述されてい

る通り、canvas の背景を白にする、円を赤色で塗り潰し表示し、円の中心座標を(150, 200)、半径を 100 にするという具合である。

このサンプルプログラムのように、多くの Android アプリでは、Activity (またはその派生) クラスから View (またはその派生) クラスを呼び出し、画面表示を行うことが一般的である。このサンプルでは円を描画する drawCircle() のみを用いているが、この他にも android.graphics.* クラスライブラリには、線を描画する drawLine() や軌跡を描画する drawPath()、矩形を描画する drawRect() などが用意されており、これらを組み合わせる事により、任意のグラフィックスの描画が可能となる。

(3) サンプル 2-タッチイベントの処理

次に、スマートフォンアプリの特徴の一つである、画面タッチによる入力機能に関するサンプルを紹介する。このアプリは画面上のタッチ入力された箇所に小さな円を表示するという簡単なものであるが、多くの Android アプリに必須なタッチイベントの処理方法を理解する上で重要である。

前述のサンプル 1 作成と同様に、新規 Android プロジェクトを作成し、以下の各項目を入力する。

プロジェクト名 : TestTouch
 ビルド・ターゲット : Android 2.2
 アプリケーション名 : TestTouch
 パッケージ名 : touch.android
 アクティビティーの作成 : TestTouch

続いて、図 5 のリストを入力し、エミュレータを実行すると図 4 の実行結果が得られる。図 4 で青い丸が描かれている箇所が画面タッチされた場所であるが、エミュレータで実行するには、画面タッチ入力の代わりに、PC のマウスで左クリックにより場所を指定し実行することになる。

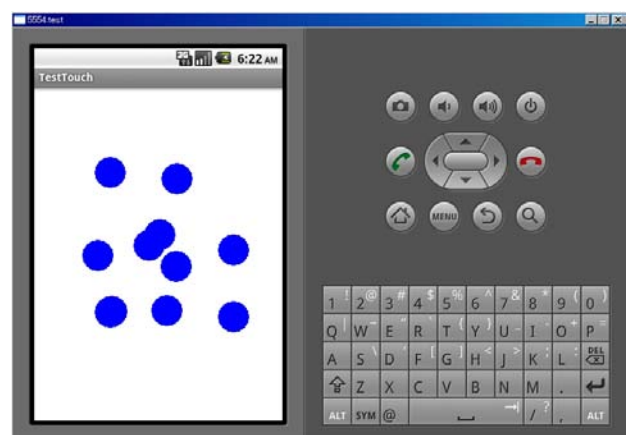


図 4 TestTouch の実行結果

```

package touch.android;

import java.util.ArrayList;
import android.app.Activity;
import android.content.Context;
import android.graphics.*;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;

public class TestTouch extends Activity {
    //アプリの初期化
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TestTouchView view =
            new TestTouchView(getApplicationContext());
        setContentView(view);
    }
    //タッチイベントの処理
    class TestTouchView extends View {
        private ArrayList<Point> points;
        //コンストラクタ
        public TestTouchView(Context c) {
            super(c);
            setFocusable(true);
            points = new ArrayList<Point>();
        }
        //タッチイベント
        public boolean onTouchEvent(MotionEvent event) {
            Point p = new Point();
            p.x = (int)event.getX();
            p.y = (int)event.getY();
            points.add(p);
            this.invalidate();
            return true;
        }
        //描画
        protected void onDraw(Canvas canvas) {
            canvas.drawColor(Color.WHITE);
            Paint paint = new Paint();
            paint.setColor(Color.BLUE);
            paint.setStyle(Paint.Style.FILL);
            for (int i=0; i<points.size(); i++) {
                Point p = points.get(i);
                canvas.drawCircle(p.x, p.y, 20, paint);
            }
        }
    }
}

```

図 5 サンプルプログラム 2 (TestTouch)

続いて、図 5 のプログラムについて解説する。最初の package と import の部分はサンプル 1 とほぼ同様であるが、タッチ入力された複数の点を記憶するために

java.util.ArrayList を使い、また、タッチイベント処理のために android.view.MotionEvent が必要となる。

プログラムの全体的な構成についてもサンプル 1 とほぼ同様で、Activity クラスの派生である TestTouch クラスから View クラスの派生である TestTouchView クラスが呼び出されている。

実際にユーザが画面をタッチしてタッチイベントとして Android OS が認識すると、onTouchEvent()メソッドが呼び出され、新しく Point が追加された後、event.getX()、event.getY()によりタッチされた画面の位置情報が Point に付加される。続いて、TestTouchView クラス内の onDraw()メソッドが呼び出され、背景を白色、円の色を青色、円の描画方法を塗り潰しに指定した後、for ループを用いることにより、これまでにタッチ入力された全ての点に対して円を描画する。このように、Android アプリで最もよく用いるタッチ入力は android.view.MotionEvent クラスを用いて、単純に event.getX()などとするだけで簡単にプログラム中から利用することが出来る。

また、このプログラムは、最後の for ループの部分を図 6 のように修正することにより、簡単なお絵かきツールになる。

```

if( points.size()>1 ) {
    for (int i=1; i<points.size(); i++) {
        Point p = points.get(i-1);
        Point q = points.get(i);
        canvas.drawLine(p.x, p.y, q.x, q.y, paint);
    }
}

```

図 6 TestTouch の修正

ただし、変数 q を新たに定義する必要があるため、プログラム中の Point p = new Point();の次の行に Point q = new Point();の一行を付け加える必要がある。この修正は、入力された点が 2 点以上ある場合に、各点を順に結ぶ線を表示するものであり、お絵かきツールの最も簡単な例となっている。図 6 の修正を施したアプリの実行結果を図 7 に示す。

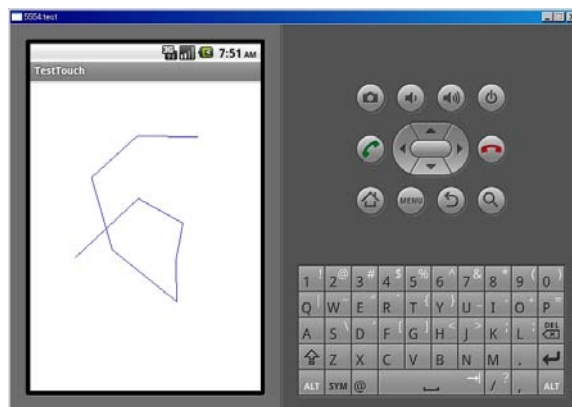


図 7 TestTouch 修正後の実行結果

3 iOS アプリ開発

(1) OpenGL ES による描画

前号の4章「Xcode プロジェクト作成」では、OpenGL をベースにしたサンプルプロジェクトの動作確認までを行った。本稿ではこれを利用し、三角形の2D描画、また4面体による3D描画、タッチスクリーンによる操作を行うアプリ作成について解説する。

iOS と Android ではグラフィックス用ライブラリとして OpenGL のサブセットである OpenGL ES (OpenGL for Embedded Systems) が採用されている。前節の Android における描画についても OpenGL を利用することも可能であり、iOS と Android 用アプリを開発する時に、描画部分については OpenGL をベースにしたソースを利用できることになる。本稿では OpenGL や iOS の記述言語である objective-c については、一部のみの解説となるため、詳細については他の文献などを参考にしていきたい。

なお、前号で使用した Xcode のバージョンは 3.2.5 であったが、iOS 5 のリリースにともない、Xcode も iOS 5 に対応したバージョン 4 が主流となりつつある。Xcode のバージョン 3 と 4 では大幅に変更が加えられているため、今後はバージョン 4 での解説が望ましいが、前号からの継続性を考え、今回もバージョン 3.2.5 を使用したサンプルを用いた。

(2) 2D グラフィックス

(i) レンダラー指定

レンダラー (rendering: 画像を生成し表示させる) には ES1Renderer (OpenGL ES 1.1) と ES2Renderer (OpenGL ES 2.0) が利用できるが、今回は ES1 のみを使用する。前号で作成した「NAME01」プロジェクトを開き、Classes 内のファイルから「EAGLView.m」をクリックする (図 8)。下部ウィンドウにソースプログラムが表示されるので、ソース内の任意の場所をクリックすると、行数が上部に表示される。これを参考に 45 行目の行頭に「//」を挿入する (図 9)。「//」はコメントアウトであり、この行を無効とする。

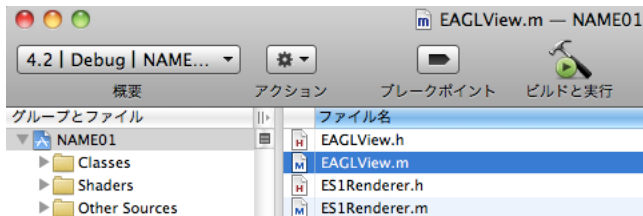


図 8 ソースプログラムの選択の様子

```
//renderer = [[ES2Renderer alloc] init];
```

図 9 修正部分 1

(ii) 座標系の設定

表示画面の座標系初期状態は、縦横比が 1:1 に設定

されているが、iPad のスクリーンでは「横:縦」が「768:1020」で構成されているため、初期状態では表示形状に差が生じる。そこで、glOrthof 関数により座標系の設定を変更する。例えば画面左上の(x,y,z)座標を(-1.0,1.33,1.0)、右下を(1.0,-1.33,-1.0)とするには、「ES1Renderer.m」の 69 行目に、図 10 の 2 行目を追加する (1 と 3 行目は記載済)。ただし 2D 画面では z 座標は意味を持たない。

```
glLoadIdentity();
glOrthof(-1.0f, 1.0f, -1.33f, 1.33f, 1.0f, -1.0f);
glMatrixMode(GL_MODELVIEW);
```

図 10 修正部分 2

次にサンプルで設定されているグラフィックの上下移動を止めるために、56 行目と 72, 73 行目 (上記の 69 行目を挿入しているため、オリジナルファイルからは行数が異なる) をコメントアウトする (図 11)。

```
//static float transY = 0.0f;

//glTranslatef(0.0f, (GLfloat)(sinf(transY)/ 2.0f),
0.0f);
//transY += 0.075f;
```

図 11 修正部分 3

ここまでの修正を行い「ビルドと実行」により、図 12(左)に示す長方形が静止して表示される。

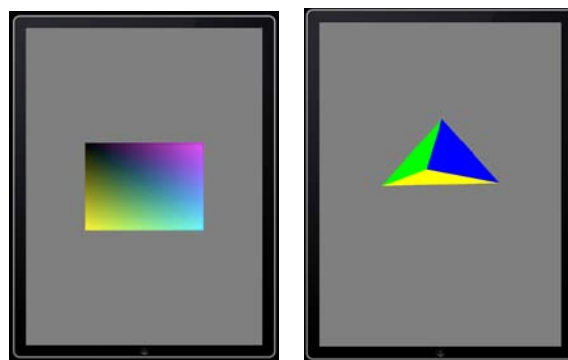


図 12 2D(左)と 3D(右)の表示画面

(iii) 三角形の描画

「ES1Renderer.m」のソースを見てみると、表示した四角形の形状と色の定義の場所が 42 行目以降の図 13 の部分であることがわかる。配列 squareVertices で、4 つの頂点(x,y)座標を定義し、squareColors で 4 つの頂点の色 (Red, Green, Blue, 透明度) を定義している。図 12(左)の表示結果から、頂点間の色については補完されていることがわかる。78 行目の glVertexPointer 関数 (図 14) により squareVertices を登録するが、第 1 引数は 2D では「2」、3D では「3」を指定する。83 行目の glDrawArrays 関数 (図 14) の第 1 引数は描画モードを指定するが、「GL_TRIANGLE_STRIP」は三角形を連続

して描画するため、ここでは2つの三角形を連続させ四角形を描画している。squareVerticesの最初の3つの頂点で三角形を1つ描画し、2から4番目の3つの頂点で2つ目の三角形を描画している。第3引数は描画する頂点の数である。

```
static const GLfloat squareVertices[] = {
    -0.5f, -0.33f,
     0.5f, -0.33f,
    -0.5f,  0.33f,
     0.5f,  0.33f,
};
static const GLubyte squareColors[] = {
    255, 255,  0, 255,
     0, 255, 255, 255,
     0,  0,  0,  0,
    255,  0, 255, 255,
};
```

図 13 座標と色の定義部分

```
glVertexPointer(2, GL_FLOAT, 0, squareVertices);
```

図 14 glVertexPointer 関数

```
glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
```

図 15 glDrawArrays 関数

ここで、描画モードを「GL_TRIANGLES」にすることにより、独立した三角形の描画が可能となる。これらの部分を図16のように変更を行うことにより、三角形が描画される。なお、配列名はtriangleVerticesなどと変更した方が望ましいが、ここではそのままsquareVerticesを使用している。

```
static const GLfloat squareVertices[] = {
    -0.5f, -0.5f,
     0.5f, -0.5f,
     0.0f,  0.5f,
};
static const GLubyte squareColors[] = {
    255, 255,  0, 255,
     0, 255, 255, 255,
    255,  0, 255, 255,
};
glDrawArrays(GL_TRIANGLES, 0, 3);
```

図 16 修正部分 4

(3) 3D グラフィックス

3D描画ではz座標情報を、画面から手前方向がz正方向、奥方向がz負方向として付加する。76行目の「glVertexPointer」第1引数は、3Dではx,y,zの3つの座標になるため「3」を指定する。三角形を4つ張り合わせ

て四面体を1つ描写する場合、描画する頂点数は、3角形の頂点数(3)×個数(4)=12となるため、81行目のglDrawArraysの最後の引数、頂点の数は12を指定する。これらの変更と、座標、色を図17のように指定すると、四面体が1つ描画される。

```
static const GLfloat squareVertices[] = {
    0.0f, 0.7f, 0.0f,
   -0.7f, 0.0f, 0.0f,
    0.7f, 0.0f, 0.0f,

   -0.7f, 0.0f, 0.0f,
    0.0f, 0.0f, 0.7f,
    0.0f, 0.7f, 0.0f,

    0.0f, 0.0f, 0.7f,
    0.7f, 0.0f, 0.0f,
    0.0f, 0.7f, 0.0f,

    0.7f, 0.0f, 0.0f,
   -0.7f, 0.0f, 0.0f,
    0.0f, 0.0f, 0.7f,
};
static const GLubyte squareColors[] = {
    100, 100, 100, 255,
     25, 100, 100, 255,
     55,  0, 200, 255,

    100, 100, 100, 255,
     25, 100, 100, 255,
     55,  0, 200, 255,

    100, 100, 100, 255,
     25, 100, 100, 255,
     55,  0, 200, 255,

    100, 100, 100, 255,
     25, 100, 100, 255,
     55,  0, 200, 255,
};
```

図 17 修正部分 5

ただし、このまま表示しても正面からの静止画しか描画されないため、グラフィックを回転させ3D描画を確認するために、glOrthof関数からglDrawArrays関数までの変更後の部分を図18に示す。また、3D描画では手前の面に隠れている奥の面を描画しない陰面処理を行わないと、視点からの正しい画像が描画されない。そこで深度バッファを用いてこの処理を行った[1]などを参考に「ES1Renderer.m」の修正を加えることにより、図12(右)に示す四面体が回転されて表示される。

```

glOrthof(-1.0f, 1.0f, -1.33f, 1.33f, 1.0f, -1.0f);
glMatrixMode(GL_MODELVIEW);
glRotatef(0.3f, 1.0f, 1.0f, 0.0f);
//glLoadIdentity();
//glTranslatef(0.0f, (GLfloat) (sinf(transY)/
2.0f), 0.0f);
//transY += 0.075f;

glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);

glVertexPointer(3, GL_FLOAT, 0, squareVertices);
glEnableClientState(GL_VERTEX_ARRAY);
glColorPointer(4, GL_UNSIGNED_BYTE, 0,
squareColors);
glEnableClientState(GL_COLOR_ARRAY);

//glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
glDrawArrays(GL_TRIANGLES, 0, 12);

```

図 18 修正部分 4

(4) マルチタッチ機能の実装

タッチイベントの処理を理解する上で図 19 に簡単な例を示す。スクリーンがタッチされた時に呼ばれるメソッド(関数)、`touchesBegan` を利用し、タッチされた座標をコンソールに表示するソースである。

```

- (void)touchesBegan:(NSSet *)touches
withEvent:(UIEvent *)event
{
    UITouch* touch = [touches anyObject];
    CGPoint pt = [touch locationInView:self];
    CGFloat glx = (pt.x / 768.0f) * 2.0f - 1.0f;
    CGFloat gly = (pt.y / 1020.0f) * -2.67f + 1.33f;
    NSLog(@"(%f, %f)", glx, gly);
}

```

図 19 タッチイベント関数

タッチされた座標が(pt.x, pt.y)に返されるが、これらの値は画面左上が(0,0)、右下が(768,1020)に設定されているスクリーン座標となっているため、これらを図10で指定した座標系(glx, gly)に変換し表示している。図19の部分でEAGLView.mの、最終行「@end」の上に加え、「ビルドと実行」を行い、「実行」メニューの「コンソール」をクリックし、表示されるコンソールウィンドウに、タッチされた座標が表示される。

同様に、ドラッグされたときに呼ばれるメソッド `touchesMoved`、タッチが終了したときに呼び出されるメソッド `touchesEnded` などがあり、これらの中で移動や回転の座標情報を獲得し描画に反映させることができる。図20にその一例を掲載する。

```

CGPoint pt0, pt1, center;
CGFloat x, y, length, rotate;
NSArray* array;

array = [[event touchesForView:self] allObjects];
pt0 = [[array objectAtIndex:0] locationInView:self];
pt1 = [[array objectAtIndex:1] locationInView:self];
x = pt0.x - pt1.x;
y = pt0.y - pt1.y;
length = sqrt(x * x + y * y);
center.x = x / 2 + pt1.x;
center.y = y / 2 + pt1.y;
rotate = atan2(y, x);

```

図 20 マルチタッチ座標

ここでは、2本指でタッチされた座標情報を、pt0, pt1に格納し、そこから2点間の距離、中心座標、回転用角度を計算している様子がわかる。

最後に、今回紹介した例を応用し、ADVENTURE System(Open Souce CAE)による解析結果から、表面パッチ情報(三角形パッチおよび相当応力コンター)を読み込み、iPad 上に表示し、タッチ操作により解析結果を確認するシステムの様子を図 21 に示す[2]。

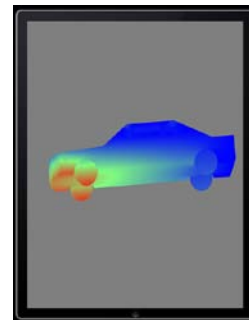


図 21 CAE 解析結果のタブレット端末での表示

4 おわりに

本チュートリアルでは第1回目にAndroidアプリとiOSアプリを開発するための開発環境設定について解説を行い、本稿では各アプリ開発の基礎的な内容について解説を行った。次回はより高度なアプリの開発手法について解説を行うとともに、非常に便利で有用なアプリ開発ツールが多くリリースされ、広く利用されているため、それらについても紹介する予定である。

参考文献

- [1] WebOS Goodies: http://webos-goodies.jp/archives/getting_started_with_opengl_on_iphone.html
- [2] 塩谷隆二, 大橋秀樹, 白柳翔太, スマートフォン端末を用いたクラウド型CAEシステム, 第16回計算工学講演会, Vol.16, F-2-3.pdf, pp.1-2, 2011.