Android

OpenGL ES 1.0

Tomasz Dzieniak

Wymagania

- JRE & JDK 5.0 +
- IDE (Eclipse 3.3.1 + / Netbeans 7.0.0 +)
- Android SDK Starter Package
- Android SDK Components

Pierwszy program

Project name:	OpenGL
Build Target:	Android 2.0
Application name:	OpenGL
Package name:	org.umk.opengl
Create Activity:	OpenGL
Min SDK Version:	8

OpenGL/src/org/umk/opengl/OpenGL.java

```
package org.umk.opengl;

import android.app.Activity;
import android.os.Bundle;

public class OpenGL extends Activity {



    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);



        setContentView(R.layout.main);
    }









}
```

OpenGL/src/org/umk/opengl/OpenGL.java

```
package org.umk.opengl;

import android.app.Activity;
import android.os.Bundle;

public class OpenGL extends Activity {
    WidokGL widok;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        widok = new WidokGL(this);

        setContentView(R.layout.main);
    }

    @Override
    protected void onPause() {
        super.onPause();
        widok.onPause();
    }

    @Override
    protected void onResume() {
        super.onResume();
        widok.onResume();
    }
}
```

Przesłaniamy metody odpowiedzialne za pauzowanie i kontynuowanie zdarzeń.

Będą wywoływać swoje odpowiedniki na rzecz wystąpienia obiektu klasy WidokGL.

OpenGL/src/org/umk/opengl/WidokGL.java

```
package org.umk.opengl;

import android.content.Context;
import android.opengl.GLSurfaceView;

class WidokGL extends GLSurfaceView {



   WidokGL(Context kontekst) {
      super(kontekst);



      setRenderer(renderer);
   }
}
```

OpenGL/src/org/umk/opengl/WidokGL.java

```
package org.umk.opengl;

import android.content.Context;
import android.opengl.GLSurfaceView;

class WidokGL extends GLSurfaceView {

    private final RendererGL renderer;

    WidokGL(Context kontekst) {
        super(kontekst);

        renderer = new RendererGL(kontekst);

        setRenderer(renderer);
    }
}
```

Tworzymy instancję klasy WidokGL, która będzie odpowiedzialna za wykonywanie całej pracy:

- konfigurację frustum
- rysowanie aktora
- animację aktora
- oświetlanie sceny
- teksturowanie
- przezroczystość

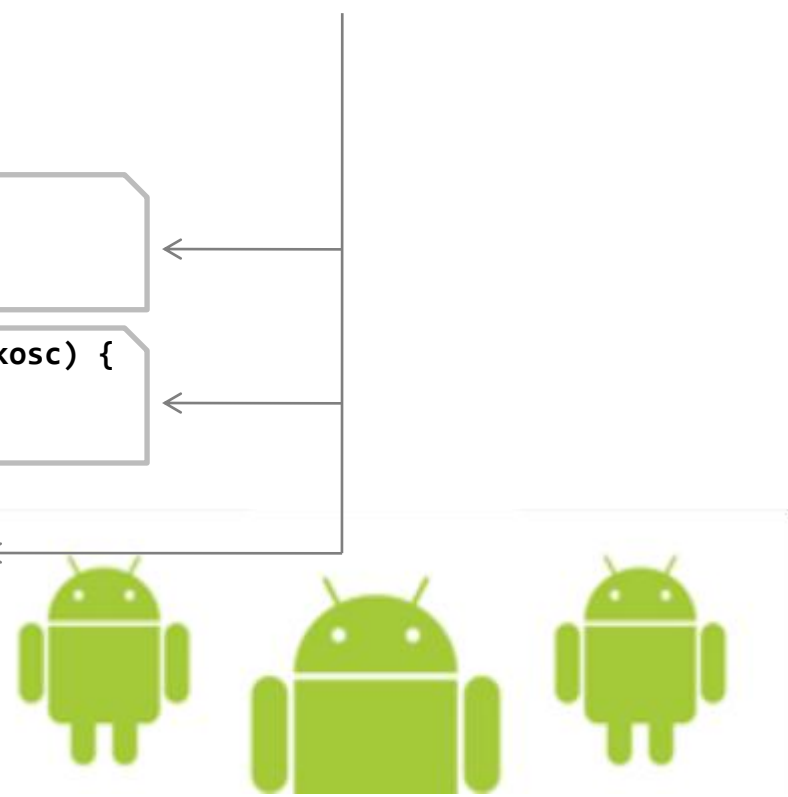
OpenGL/src/org/umk/opengl/RendererGL.java

```
package org.umk.opengl;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

class RendererGL implements GLSurfaceView.Renderer {

   private final Context kontekst;

   RendererGL(Context kontekst) {
      this.kontekst = kontekst;
   }









}
```

OpenGL/src/org/umk/opengl/RendererGL.java

```
package org.umk.opengl;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;

class RendererGL implements GLSurfaceView.Renderer {

   private final Context kontekst;

   RendererGL(Context kontekst) {
      this.kontekst = kontekst;
   }

   public void onSurfaceCreated(GL10 gl, EGLConfig konfig) {
      // ...
   }

   public void onSurfaceChanged(GL10 gl, int szerokosc, int wysokosc) {
      // ...
   }

   public void onDrawFrame(GL10 gl) {
      // ...
   }
}
```

Przeciążamy metody odpowiedzialne za tworzenie ramki, jej odświeżanie i wyświetlanie.



OpenGL/src/org/umk/opengl/RendererGL.java

```
public void onSurfaceCreated(GL10 gl, EGLConfig konfig) {

   gl.glEnable(GL10.GL_DEPTH_TEST);
   gl.glDepthFunc(GL10.GL_LEQUAL);
   gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

}
```

GL_DEPTH_TEST – porównanie głębi i aktualizacja bufora głębokości

GL_LEQUAL – określa, które piksee bufor Z ma zignorować, a które wyświetlić na ekranie

GL_VERTEX_ARRAY – uruchamia vertex array

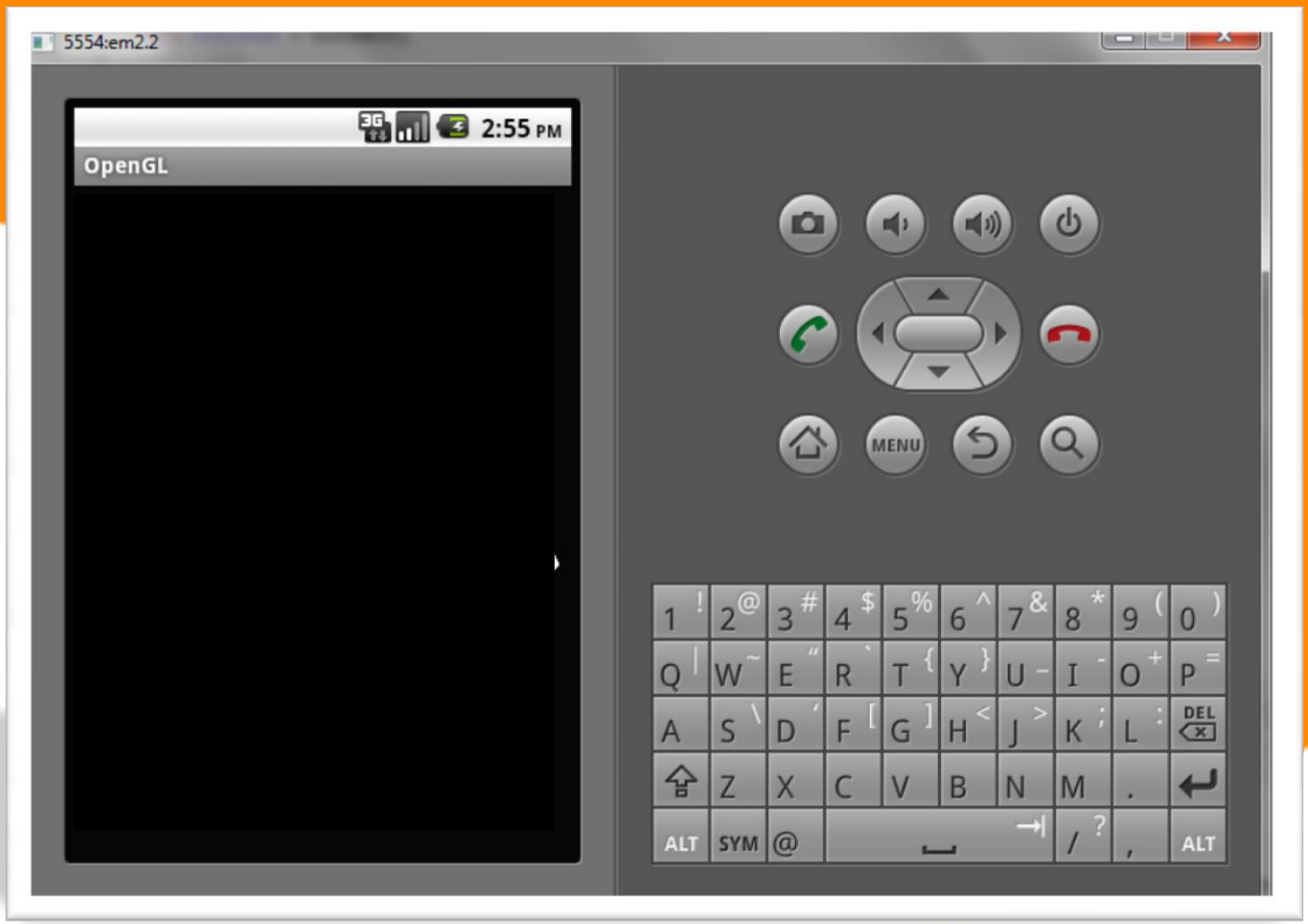
OpenGL/src/org/umk/opengl/RendererGL.java

```
public void onSurfaceChanged(GL10 gl, int szerokosc, int wysokosc) {

   gl.glViewport(0, 0, szerokosc, wysokosc);
   gl.glMatrixMode(GL10.GL_PROJECTION);
   gl.glLoadIdentity();

   float proporcje = (float) szerokosc / wysokosc;
   GLU.gluPerspective(gl, 45.0f, proporcje, 1, 100f);

}
```

- ustawiamy ostrosłup widzenia
- uruchamiamy macierz modelu
- resetujemy macierz modelowania

- określamy perspektywę i parametry wyświetlania obrazu
  - kąt widzenia
  - proporcje obrazu
  - bliższą płaszczyznę ograniczającą
  - dalszą płaszczyznę ograniczającą

Pierwszy rzut oka



OpenGL/src/org/umk/opengl/RendererGL.java

```
public void onDrawFrame(GL10 gl) {

   gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
   gl.glMatrixMode(GL10.GL_MODELVIEW);
   gl.glLoadIdentity();
   gl.glTranslatef(0, 0, -3.0f);

}
```

- oczyszczamy bufor głębi i bufor koloru
- resetujemy macierz modelowania
- przesuwamy obraz tak, aby coś zobaczyć

OpenGL/src/org/umk/opengl/SzescianGL.java

```
package org.umk.opengl;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.IntBuffer;

import javax.microedition.khronos.opengles.GL10;

class SzescianGL {
   private final IntBuffer mBuforWierzcholka;

   public SzescianGL() {
      int jeden = 65536;
      int pol = jeden / 2;
      int[] wierzcholki = {
         // ściana przednia
         -pol, -pol, pol,
         pol, -pol, pol,
         -pol, pol, pol,
         pol, pol, pol,
         // ściana tylnia
         -pol, -pol, -pol,
         -pol, pol, -pol,
         pol, -pol, -pol,
         pol, pol, -pol,
         // ściana lewa
         -pol, -pol, pol,
         -pol, pol, pol,
         -pol, -pol, -pol,
         -pol, pol, -pol,
         // ściana prawa
         pol, -pol, -pol,
         pol, pol, -pol,
         pol, -pol, pol,
         pol, pol, pol,
         // szczyt
         -pol, pol, pol,
         pol, pol, pol,
         -pol, pol, -pol,
         pol, pol, -pol,
         // spód
         -pol, -pol, pol,
         -pol, -pol, -pol,
         pol, -pol, pol,
         pol, -pol, -pol,
      };

      ByteBuffer vbb =    ByteBuffer.allocateDirect(
         wierzcholki.length * 4);
      vbb.order(ByteOrder.nativeOrder());
      mBuforWierzcholka = vbb.asIntBuffer();
      mBuforWierzcholka.put(wierzcholki);
      mBuforWierzcholka.position(0);
   }
```

OpenGL/src/org/umk/opengl/SzescianGL.java CDN

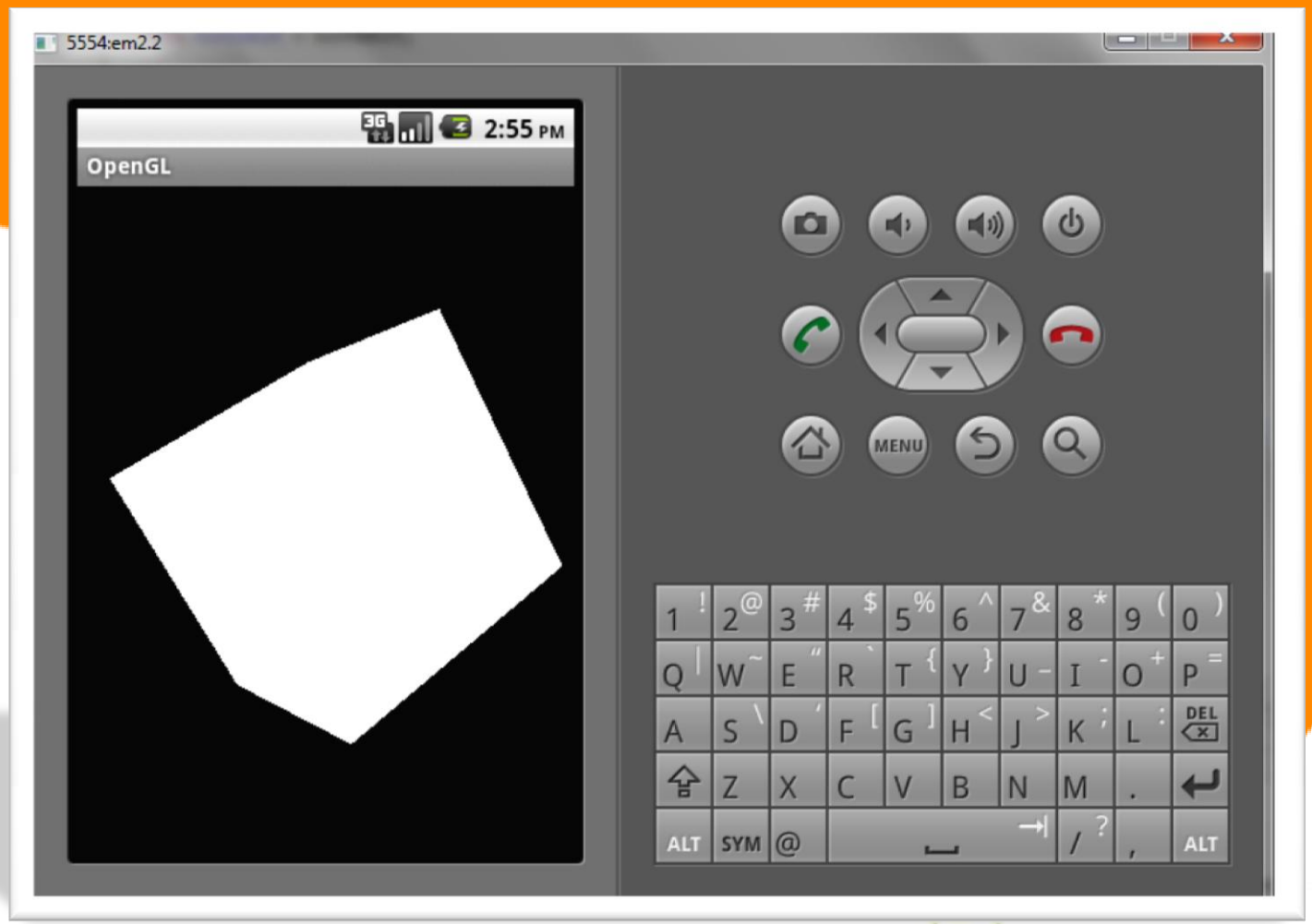
```
    public void rysuj(GL10 gl) {
        gl.glVertexPointer(3, GL10.GL_FIXED, 0, mBuforWierzcholka);

        gl.glColor4f(1, 1, 1, 1);
        gl.glNormal3f(0, 0, 1);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
        gl.glNormal3f(0, 0, -1);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 4, 4);

        gl.glColor4f(1, 1, 1, 1);
        gl.glNormal3f(-1, 0, 0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 8, 4);
        gl.glNormal3f(1, 0, 0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 12, 4);

        gl.glColor4f(1, 1, 1, 1);
        gl.glNormal3f(0, 1, 0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 16, 4);
        gl.glNormal3f(0, -1, 0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 20, 4);
    }
}
```

OpenGL/src/org/umk/opengl/RendererGL.java

```
// aktor
private final SzescianGL szescian = new SzescianGL();

public void onDrawFrame(GL10 gl) {

   (…)

   szescian.rysuj(gl);
}
```

- tworzymy instancję klasy SzescianGL (tworzymy aktora)
- w metodzie onDrawFrame wywołujemy funkcję rysującą

Gotowy sześcian



OpenGL/src/org/umk/opengl/RendererGL.java

```
// poniżej oświetlenie
float[] oswietlenieDookolne = new float[] { 0.2f, 0.2f, 0.2f, 1 };
float[] oswietlenieRozproszone = new float[] { 1, 1, 1, 1 };
float[] pozostaleZrodlaSwiatla = new float[] { 1, 1, 1, 1 };
gl.glEnable(GL10.GL_LIGHTING);
gl.glEnable(GL10.GL_LIGHT0);
gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_AMBIENT, oswietlenieDookolne, 0);
gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_DIFFUSE, oswietlenieRozproszone, 0);
gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, pozostaleZrodlaSwiatla, 0);

// symulacja materiału
float[] materialDookolne = new float[] { 1, 1, 1, 1 };
float[] materialRozproszone = new float[] { 1, 1, 1, 1 };
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT, materialDookolne, 0);
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE, materialRozproszone, 0);
```

- konfigurujemy parametry światła i materiału
  - położenie światła i kolor

Tworzone światło (w położeniu (1,1,1) jest nieukierunkowane, posiada składową światła dookolnego i rozproszonego.

Materiał zostaje określony jako matowy (podobny do papieru).

W efekcie prawy górny róg sześcianu, który znajduje się bliżej źródła światła będzie jaśniejszy.

OpenGL/src/org/umk/opengl/RendererGL.java

```
// animacja – klasa RendererGL
private long czasStartu;



// animacja – metoda onSurfaceCreate
czasStartu = System.currentTimeMillis();


// animacja – metoda onDrawFrame ->
long szacowany = System.currentTimeMillis() - czasStartu;
gl.glRotatef(szacowany * (30f / 1000f), 0, 1, 0);
gl.glRotatef(szacowany * (15f / 1000f), 1, 0, 0);
// <- animacja
```

Dodanie powyższych fragmentów kodu spowoduje ładną, płynną animację sześcianu (obrót o 30 stopni wzdłuż osi x i 15 stopni wzdłuż osi y).

Uwaga: emulowanie programu może powodować zmniejszenie wydajności wyświetlanej animacji. Na urządzeniu płynność jest płynniejsza… ;)

OpenGL/src/org/umk/opengl/SzescianGL.java

```
    // tekstura
    private final IntBuffer mBuforTekstury;

    // tekstura
    int[] wspolrzedneTekstury = {
    // ściana przednia
    0, jeden, jeden, jeden, 0, 0, jeden, 0,
    // ściana tylnia
    jeden, jeden, jeden, 0, 0, jeden, 0, 0,
    // ściana lewa
    jeden, jeden, jeden, 0, 0, jeden, 0, 0,
    // ściana prawa
    jeden, jeden, jeden, 0, 0, jeden, 0, 0,
    // szczyt
    jeden, 0, 0, 0, jeden, jeden, 0, jeden,
    // spód
    0, 0, 0, jeden, jeden, 0, jeden, jeden,
};

    ByteBuffer tbb = ByteBuffer.allocateDirect(wspolrzedneTekstury.length * 4);
    tbb.order(ByteOrder.nativeOrder());
    mBuforTekstury = tbb.asIntBuffer();
    mBuforTekstury.put(wspolrzedneTekstury);
    mBuforTekstury.position(0);
```

Powyższy kod należy umieścić w konstruktorze.

```
static void wczytajTeksture(GL10 gl, Context kontekst, int zasob) {
   Bitmap bmp = BitmapFactory.decodeResource(
   kontekst.getResources(), zasob);
   GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bmp, 0);
   gl.glTexParameterx(GL10.GL_TEXTURE_2D,
   GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_LINEAR);
   gl.glTexParameterx(GL10.GL_TEXTURE_2D,
   GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
}
```

Poniższy kod należy umieścić w metodzie rysuj.

```
   // tekstura ->
   gl.glEnable(GL10.GL_TEXTURE_2D);
   gl.glTexCoordPointer(2, GL10.GL_FIXED, 0, mBuforTekstury);
   // <- tekstura
```

Na koniec w klasie RendererGL w metodzie onSurfaceCreated wymuszamy obsługę tekstur.

```
   // tekstura
   gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
   gl.glEnable(GL10.GL_TEXTURE_2D);
```

Efekt końcowy

Przezroczystość

Pytania?

Pytania?