Programowanie Urządzeń Mobilnych

Laboratorium nr 11, 12

Android

Temat 3 – wykorzystanie sensorów i multimediów w Android SDK

Krzysztof Bruniecki

Zadania do wykonania na zajęciach 1

Zadanie 1. (opisane dalej)

Zadanie 2. (opisane dalej)

Zadania do wykonania na zajęciach 2

Zadanie 3. (opisane dalej)

Wstęp

Android SDK wyposażony jest w szereg możliwości tworzenia multimedialnych aplikacji. Podczas laboratorium zapoznamy się z wykorzystaniem następujących z nich:

• dostęp do sensorów położenia,

• dostęp do kamery cyfrowej urządzenia.

• tworzenie grafiki 3D z wykorzystaniem OpenGL ES,

Niektóre z funkcji w Android SDK są opisane poniżej.

Tabela 1. Hardwarowe funkcje wystawione w Android SDK

Cecha	Opis
android.hardware.Camera	Klasa, która umożliwia aplikacji interakcję z aparatem do zrobienia zdjęcia, zmiany parametrów używanych do zarządzania kamerą.
android.hardware.SensorManager	Klasa, która pozwala na dostęp do czujników dostępnych w ramach platformy Android. Nie każde urządzenie wyposażone w Androida obsługuje wszystkie czujniki w SensorManager.
android.hardware.SensorListener	Interfejs realizowany przez klasy, które chcą otrzymywać aktualizacje wartości czujnika, ponieważ zmiany zachodzą w czasie rzeczywistym. Aplikacja implementuje ten interfejs do nadzorowania jednego lub więcej czujników dostępnych w urzadzeniu.
android.FaceDetector	Klasa, która pozwala na podstawowe rozpoznanie twarzy osoby zawartej w postaci bitmapy. Traktując to jako blokadę urządzenia oznacza brak haseł do zapamiętania - możliwości technologii biometrycznych na telefon komórkowy.
android.os .*	Pakiet zawiera kilka użytecznych klas w interakcji ze środowiskiem pracy, w tym zarządzania energią, „watcher" plików, obsługa i klasa wiadomości.
java.util.Date java.util.Timer java.util.TimerTask	Podczas pomiaru wydarzeń w świecie rzeczywistym, data i czas są często znaczące. Na przykład,klasa java.util.Date pozwala uzyskać znacznik czasu, gdy określone zdarzenie lub stan są spotykane. Możesz użyć java.util.Timer i java.util.TimerTaskdo wykonywania okresowych zadań.

Android.hardware.SensorManager zawiera kilka stałych, które reprezentują różne aspekty systemu czujników Androida, w tym:

Sensor type

Orientacja, akcelerometr, światło, pole magnetyczne, zbliżeniowe, temperatury, itp.

Sampling rate

Najszybsza, gry, normalny, interfejs użytkownika. Gdy aplikacja żąda określonej częstotliwości próbkowania, to tak naprawdę tylko podpowiedź, czy sugestia, do podsystemu czujnika.

Accuracy

Wysoki, niski, średni, niewiarygodne.

Interfejs **SensorListener** dwie wymagane metody:

- **OnSensorChanged (int sensor, float values[])** metoda jest wywoływana zawsze, gdy wartość czujnika uległa zmianie. Metoda jest wywoływana tylko dla czujników, które są monitorowane przez program. Argumenty metody to: liczb całkowita, która określa czujnik, który zmienił wartości z tablicy typu float reprezentujących dane z czujników. Niektóre czujniki dostarczają tylko jedną wartość, podczas gdy inne zapewniają trzy wartości. Czujniki orientacji i akcelerometrdostarczają trzy wartości.
- **OnAccuracyChanged (int sensor, int accuracy)** metoda jest wywoływana, gdy dokładność czujnika została zmieniona. Argumenty są to dwie liczby: jedna przedstawia czujnik, a druga reprezentuje nową wartość dokładności dla tego czujnika.

Aby skontaktować się z czujnikiem, aplikacja musi zarejestrować się w celu wykrywania działalności jednego lub więcej czujników. Rejestracja odbywa się za pomocą metody registerListener z klasy SensorManager.

Pamiętaj, nie każde urządzenie wyposażone w Androida obsługuje wszystkich czujników określonych w SDK.

ZADANIE 1

Zadanie polega na demonstracji odczytu danych z sensorów zawartych w urządzeniu. Aplikacja ma wyświetlać dostępne sensory i umożliwiać wyświetlenie informacji z wybranego sensora w postaci natywnej – czyli jako wartości liczbowe wektora (najczęściej trójwymiarowego).

1. W Eclipse należy utworzyć nowy projekt wybierając **File | New | Project** z polecenia menu.

2. Wybierz **Android Project** z szablonów.

3. Uzupełnij **Project name** nazwą względem swojego wyboru.

4. Wybierz platformę: np. **Android 2.3**

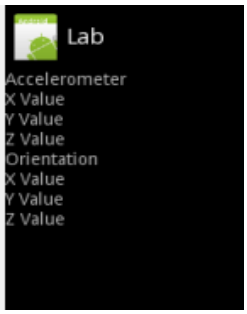
5. Uzupełnij **Application Name** oraz **Package Name**.

6. Kliknij **Finish**. Zostanie utworzony nowy projekt.

7. W **res → layout → main.xml**, stwórz odpowiedni widok dla aplikacji, np.:

Dla typu LinearLayout:

```
<TextView


    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Accelerometer"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="X Value"
    android:id="@+id/xbox"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Y Value"
    android:id="@+id/ybox"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Z Value"
    android:id="@+id/zbox"
    />

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Orientation"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="X Value"
    android:id="@+id/xboxo"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Y Value"
    android:id="@+id/yboxo"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Z Value"
    android:id="@+id/zboxo"
    />
```



Rys 1. Widok main.xml

8. Stworzona klasa typu Activity powinna także implementować interfejs **SensorListener**. Importuj klasy i zadeklaruj zmienne:

```
import android.hardware.*;

import android.widget.*;

[..............]

SensorManager sm = null;

//elementy widoku w których będą wyświetlane dane z sensorów
TextView xViewA = null;
TextView yViewA = null;
TextView zViewA = null;
TextView xViewO = null;
TextView yViewO = null;
TextView zViewO = null;
```

W celu wykonania ćwiczenia należy posłużyć się klasą **SensorManager** której instancję można uzyskać wywołując **Context.getSystemService()** z argumentem **SENSOR_SERVICE**. Należy to uczynić w metodzie **onCreate**:

```
sm = (SensorManager) getSystemService(SENSOR_SERVICE);
```

W **onCreate** należy też określić odniesienia do sześciu widgetów TextView w których następować będzie aktualizacja wartości danych z czujnika.

```
yViewA = (TextView) findViewById(R.id.ybox);
zViewA = (TextView) findViewById(R.id.zbox);
xViewO = (TextView) findViewById(R.id.xboxo);
yViewO = (TextView) findViewById(R.id.yboxo);
zViewO = (TextView) findViewById(R.id.zboxo);
xViewA = (TextView) findViewById(R.id.xbox);
```

9. Nadpisz metodę **onResume**, w metodzie wykorzystaj odniesienie do **SensorManager** do rejestracji obiektu klasy **SensorListener** z metody **registerListener**:

- Pierwszy parametr jest instancją klasy, która implementuje interfejs **SensorListener,** czyli tej w której obecnie pracujemy().

- Drugi parametr jest maską bitową pożądanych czujników.
- Trzeci parametr jest podpowiedzią dla systemu, aby wskazać, jak szybko aplikacja wymaga aktualizacji wartości czujnika.

```
sm.registerListener(this, Sensor.TYPE_ALL,
SensorManager.SENSOR_DELAY_NORMAL);
```

10. Klasa implementująca interfejs **SensorListener** musi implementować dwie metody **onSensorChange** i **onAccuracyChanged**. Metoda **onSensorChanged** wywoływana jest stale, jak akcelerometr i czujnik orientacji wysyłają dane. Pierwszy parametr określa, który czujnik wysyła dane. Gdy czujnik wysyła zostanie zidentyfikowany, wtedy odpowiednie elementy interfejsu użytkownika są aktualizowane z danych zawartych w tablicy typu float przekazywanej jako drugi argument do metody.

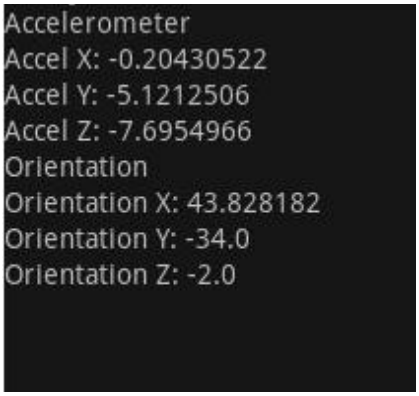
Tak wygląda jej przykładowa implementacja:

```
public void onSensorChanged(int sensor, float[] values) {

        synchronized (this) {
            if (sensor == SensorManager.SENSOR_ORIENTATION) {
                xViewO.setText("Orientation X: " + values[0]);
                yViewO.setText("Orientation Y: " + values[1]);
                zViewO.setText("Orientation Z: " + values[2]);
            }
            if (sensor == SensorManager.SENSOR_ACCELEROMETER) {
                xViewA.setText("Accel X: " + values[0]);
                yViewA.setText("Accel Y: " + values[1]);
                zViewA.setText("Accel Z: " + values[2]);
            }
        }
    }
```

```
    public void onAccuracyChanged(int sensor, int accuracy) {

    }
```

11. W metodzie **onPause** należy wyrejestrować obiekt klasy implementującej interfejs **SensorListener** (w naszym przypadku to klasa w której właśnie pracujemy) za pomocą metody **unregisterListener** klasy **SensorManager**.

```
protected void onPause() {

        super.onPause();
        sm.unregisterListener(this);
    }
```



Rys 2. Przykładowe działanie programu

ZADANIE 2

Zadanie polega na stworzeniu aplikacji umożliwiającej wyświetlanie obrazu z wbudowanej kamery. W tym celu należy podłączyć obraz z podglądu kamery do umieszczonej w aplikacji powierzchni.

1. W Eclipse należy utworzyć nowy projekt wybierając **File | New | Project** z polecenia menu.

2. Wybierz **Android Project** z szablonów.

3. Uzupełnij **Project name** nazwą względem swojego wyboru.

4. Wybierz platformę: np. **Android 2.3**

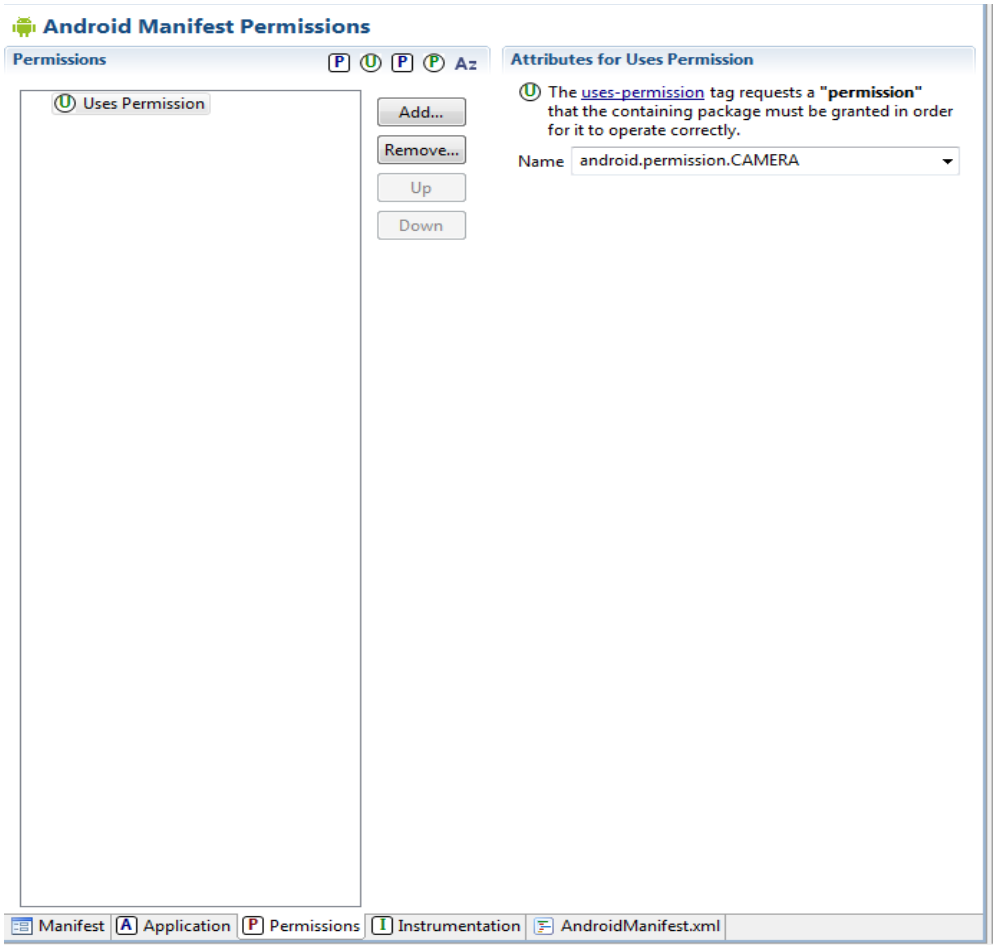
5. Uzupełnij **Application Name** oraz **Package Name**.

6. Kliknij **Finish**. Zostanie utworzony nowy projekt.

7. W **res → layout → main.xml**, stwórz odpowiedni widok dla aplikacji, np.:

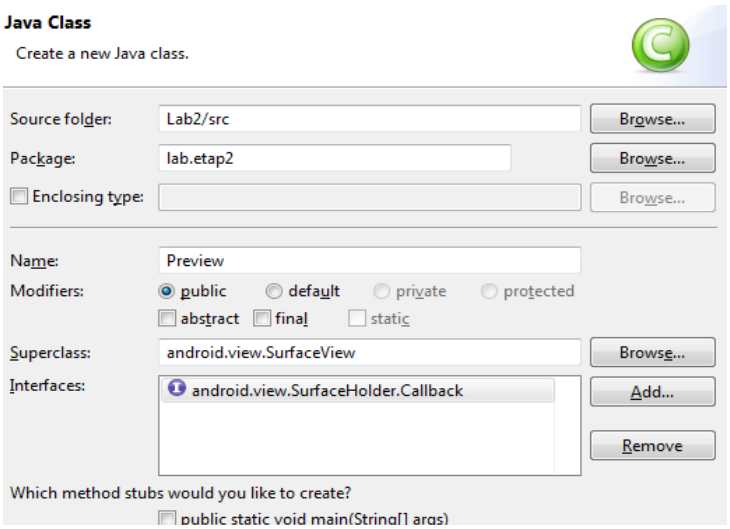
```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

            android:orientation="vertical"
android:layout_width="fill_parent"
            android:layout_height="fill_parent"
android:id="@+id/layout">
             <TextView android:layout_width="fill_parent"
                  android:layout_height="wrap_content"
android:text="Camera Demo"
                  android:textSize="24sp" />
            <FrameLayout android:id="@+id/preview"
                  android:layout_weight="1"
android:layout_width="fill_parent"
                  android:layout_height="fill_parent">
             </FrameLayout>
      </LinearLayout>
```

8. W celu umożliwienia aplikacji wykorzystywania sprzętowej kamery konieczne jest ustawienie uprawnienia z grupy Uses Permissions:
   android.permission.CAMERA
   W pliku **AndroidManifest.xml** w zakładce *permisions* kliknij Add.
   Wybierz „Uses permision” . Następnie wybierz jak na rysunku.



Rys 3. Wybór android.permission.CAMERA

9. Dodaj nową klasę do projektu. Nazwij ją ***Preview***. W polu Superclass wpisz **android.view.SurfaceView**, a w Interfaces wpisz **android.view.SurfaceHolder.Callback**.



Rys 4. Okno dodania klasy.

Klasa **Preview** obsługuje podgląd z kamery. Jest to podklasa klasy **SurfaceView**, tak więc można ją umieścić w interfejsie. Także implementuje interfejs **SurfaceHolder.Callback**. Klasa może implementować ten interfejs, aby otrzymywać informacje o zmianach powierzchni.W przypadku korzystania z **SurfaceView**, zmiany powierzchni odbywają się tylko pomiędzy wywołaniami **surfaceCreated (SurfaceHolder)** i **surfaceDestroyed (SurfaceHolder), surfaceChanged**.

10. W klasie dodaj zmienne:

```
//uchwyt sceny
public  SurfaceHolder mHolder;
//obiekt kamery
public Camera camera;
```

11. W konstruktorze klasy **Preview** zainstaluj **SurfaceHolder.Callback** aby otrzymywać powiadomienia, gdy podstawowa powierzchnia jest tworzona i niszczona.

```
super(context);
mHolder = getHolder();
mHolder.addCallback(this);
mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

SURFACE_TYPE_PUSH_BUFFERS generuje kilka buforów **SurfaceView**. Komponenty wypełniają danymi i wyświetlają dane tych buforów głęboko w

kodzie OS.

12. W metodzie **surfaceCreated** aby uzyskać dostęp do usług kamery, użyj statycznej metody **Open** z klasy **Camera**. Trzeba także przypisać uchwyt sceny do obiektu kamery, gotowa metoda powinna mieć postać:

```
public void surfaceCreated(SurfaceHolder holder) {
      camera = Camera.open();
      try {
          camera.setPreviewDisplay(holder);

      } catch (IOException e) {
          e.printStackTrace();
      }
}
```

Ciekawostka: Dla zasygnalizowania powodzenia utworzenia sceny można użyć powiadomienia **Toast.** Powiadomienie toast to komunikat, który pojawia się na powierzchni okna. Wypełnia ilość miejsca wymaganą dla wiadomości. Powiadomienie automatycznie znika i nie akceptuje zdarzeń interakcji. Powiadomienia tworzy się za pomocą metody **makeText** z klasy **Toast,** przyjmuje 3 argumenty: kontekst wyświetlenia, tekst do wyświetlenia, czas wyświetlenia. Powiadomienie wyświetla się metodą **show.**

```
Toast.makeText(this.getContext(), "surfaceCreated",
Toast.LENGTH_LONG).show();
```

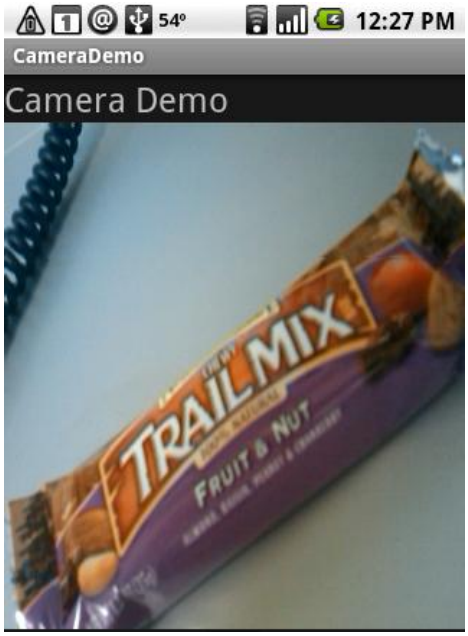
13. W metodzie **surfaceDestroyed** należy zatrzymać działanie kamery za pomocą metody **stopPreview,** zwolnić obiekt za pomocą metody **release** klasy Camera oraz należy wyczyścić obiekt **camera** przypisując mu wartość *null*.

14. W metodzie **surfaceChanged** :

Aktualne ustawienia aparatu są dostępne jako pole **Camera.Parameters**. Wywołaj metodę **getParameters** obiektu camera aby mieć dostęp do aktualnych parametrów. Użyj metody **setPreviewSize** ,aby uaktualnić rozmiary sceny podając jako argumenty wysokość i szerokość. Aby zastosować zmiany, wywołaj metodę **setParameters**. Aby podgląd kamery był wyświetlany w czasie rzeczywistym na powierzchni wywołaj metodę **startPreview** obiektu camera.

15. Tak przygotowaną kontrolkę można umieścić (w metodzie onCreate klasy Activity) używając mechanizmu dynamicznego ustalania layoutu jak na poniższym przykładzie:

```
Preview preview = new Preview(this);
FrameLayout frame = (FrameLayout)findViewById(R.id.preview);
frame.addView(preview);
```



Rys. 5. Okno aplikacji

16. Należy dodać przycisk umożliwiający zapisanie aktualnego obrazu w kamerze na karcie SD. W tym celu należy skorzystać z metody *takePicture* z obiektu klasy *Camera*. Nazwa pliku powinna pochodzić od daty i czasu wykonania zdjęcia.
    UWAGA!!
    Należy stworzyć ograniczenie polegające na tym że użytkownik może wykonać zdjęcie tylko wtedy gdy kamera jest nieruchoma.

ZADANIE 3

Zadanie polega na rozbudowie projektu aplikacji wykorzystującej

OpenGL ES.

Należy wykorzystać efekty zadania 1 w celu rozbudowy aplikacji animującej wielościan wyświetlany.

Jako podstawowe założenia należy przyjąć:

- **Animacja ma odbywać się zgodnie z kierunkiem ruchu urządzenia (dane o ruchu urządzenia należy czerpać z sensora położenia).**
- **Tekstura animowanego obiektu powinna być stworzona ilustracji w formacie BMP o rozmiarze 512x512, należy ją dołączyć do zasobów projektu (folder_projektu/res/drawable) .**

UWAGA: Poniższa instrukcja pokazuje tylko jak zrobić początkową część zadania, dzięki której uzyskamy wirujący sześcian.

1. Stwórz nowy projekt typu **Android Application.**
2. W klasie typu **Activity** w metodzie **onCreate** należy:
   - stworzyć nowy widok sceny, który zaimplementowany jest w klasie **GLSurfaceView**.
   - Za pomoca metody **setRenderer** ustawiamy dla widoku sceny nowy obiekt z implementacją klasy Renderer:

```
        GLSurfaceView glSurfView = new GLSurfaceView(this);
        glSurfView.setRenderer(new Renderer() {

                public void onSurfaceCreated(GL10 gl, EGLConfig
config) {
                        // TODO Auto-generated method stub

                }

                public void onSurfaceChanged(GL10 gl, int width, int
height) {
                        // TODO Auto-generated method stub

                }

                public void onDrawFrame(GL10 gl) {
                        // TODO Auto-generated method stub

                }
        });
        setContentView(glSurfView);
```

Klasa ta implementuje dane metody:

Metody publiczne	
abstract void	onDrawFrame(GL10 gl) Wywoływana w celu przerysowania bieżącej ramki.
abstract void	onSurfaceChanged(GL10gl, int szerokość, int wysokość) Wywoływana, gdy powierzchnia zmieniła rozmiar.
abstract void	onSurfaceCreated(GL10 gl, EGLConfig config) Wywoływana, gdy powierzchnia jest utworzona lub odtworzona.

3. W klasie **Renderer** dodaj następujące zmienne:

```
//zmienna przechowywująca zdjęcię
private Bitmap bmp;
//zmienna przechowywująca teksturę
private int tex;
//zmienna kąta rotacji
private float mAngle=0;
//bufor wierzchołków
private IntBuffer mVertexBuffer;
//bufor indeksów
private ByteBuffer mIndexBuffer;
//bufor koordynatów tekstury
```

```
private FloatBuffer mTextureBuffer;
//szerokość sceny
private int mWidth;
//wysokość sceny
private int mHeight;
//flaga stworzenia sceny
private boolean created = false;
```

4. Z powodu że OPEN GL ES korzysta się z liczb stałoprzecinkowych, typu GL-Fixed należy zapisać wartość 1 jako stałą:

```
private int one = 0x10000;
```

5. Należy dodać macierze:

- wierzchołków

```
private int vertices[] = {
            -one, -one, -one,
            one, -one, -one,
            one,  one, -one,
            -one,  one, -one,
       -one, -one,  one,
            one, -one,  one,
            one,  one,  one,
            -one,  one,  one,
};
```

- koordynatów tekstur

```
private float textures[] = {
0,0,
1,0,
1,1,
0,1,
1,0,
0,0,
0,1,
1,1
};
```

- wskaźników

```
private byte indices[] = {
            0, 4, 5,    0, 5, 1,
            1, 5, 6,    1, 6, 2,
            2, 6, 7,    2, 7, 3,
            3, 7, 4,    3, 4, 0,
            4, 7, 6,    4, 6, 5,
            3, 0, 1,    3, 1, 2
 };
```

6. W metodzie **onSurfaceCreated** klasy renderera:

– załadować bitamapę tekstury:

```
bmp = (Bitmap)BitmapFactory.decodeResource (context.getResources(),
R.drawable.mapa2);
```

– załadować macierze utworzone w poprzednim kroku do buforów:

```
ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
vbb.order(ByteOrder.nativeOrder());
mVertexBuffer = vbb.asIntBuffer();
mVertexBuffer.put(vertices);
mVertexBuffer.position(0);

mIndexBuffer = ByteBuffer.allocateDirect(indices.length);
mIndexBuffer.put(indices);
mIndexBuffer.position(0);

ByteBuffer texture_direct = ByteBuffer.allocateDirect (textures.length
* (Float.SIZE >> 3));

texture_direct.order (ByteOrder.nativeOrder ());
mTextureBuffer = texture_direct.asFloatBuffer ();
mTextureBuffer.put(textures);
mTextureBuffer.position(0);
```

– Tworzenie sceny następuje w metodzie **onSurfaceCreated**. Sygnalizujemy w niej stworzenie sceny ustawiając wartość flagi **created** na true.W niej wprowadzamy ustawienia dla sceny. Za pomocą metody **glEnable** włączamy test bufora głębokości oraz mozliwość teksturowania powierzchni. Za pomocą metody **glEnableClientState** ustawiamy stan wierzchołków oraz koordynatów tekstur. Ładujemy także bitmapy jako tekstury.

```
gl.glEnable (GL10.GL_DEPTH_TEST);
gl.glEnable (GL10.GL_TEXTURE_2D);
gl.glEnableClientState (GL10.GL_VERTEX_ARRAY);
gl.glEnableClientState (GL10.GL_TEXTURE_COORD_ARRAY);

tex = loadTexture (gl, bmp);
```

7. W metodzie onDrawFrame następuje wyrysowanie:
   – na początek ustalamy punkt widoku(pozycje kamery) oraz sprawdzić czy scena została stworzona co na sygnalizuje zmienna **created**:
   –

```
boolean c = false;
synchronized (this)
{
      c = created;
}
if (!c) return;

gl.glViewport(0, 0, mWidth, mHeight);
```

- Ustaw macierz projekcji. To nie musi być wykonane za każdym razem, ale zwykle nowa projekcja musi być ustawione przy zmianie rozmiaru rzutni.

```
float ratio = (float) mWidth / mHeight;
      gl.glMatrixMode(GL10.GL_PROJECTION);
      gl.glLoadIdentity();
      gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
```

- Następnie następuje tworzenie sceny 3D na macierzy GL_ModelView:

Rysowanie sześcianu:

```
gl.glMatrixMode(GL10.GL_MODELVIEW);
      gl.glLoadIdentity();

      gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
GL10.GL_DEPTH_BUFFER_BIT);
//tłumaczenie obecnej macierzy
      gl.glTranslatef(0, 0, -3.0f);
//rotacja
gl.glRotatef(mAngle,        0, 1, 0);
      gl.glRotatef(mAngle*0.25f,  1, 0, 0);
//
      //ustawienie zawartości buforów
      gl.glVertexPointer(3, GL10.GL_FIXED, 0,
mVertexBuffer);
      gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0,
mTextureBuffer);

      //podpinanie tekstur
      gl.glBindTexture(GL10.GL_TEXTURE_2D, tex);
      gl.glDrawElements(GL10.GL_TRIANGLES, 36,
GL10.GL_UNSIGNED_BYTE, mIndexBuffer);

gl.glFlush ();
gl.glFinish ();
      mAngle += 3.2f;
```

8. Przy zmianie szerokości lub długości widoku zmiany są przeprowadzane w metodzie **onSurfaceChanged**, należy w niej uaktualnić stare wymiary **mHeight** i **mWidth**.

9. Aby móc załadować bitmapę, w klasie renderera należy stworzyć metodę do przeprowadzenia przekształceń bitmap na teksturę:

```
private int loadTexture (GL10 gl, Bitmap bmp)
{

}
```

W ciele metody należy:

– Należy ulokować bitmapę w buforze jako bajty w natywnym porzadku.

```
ByteBuffer bb =
ByteBuffer.allocateDirect(bmp.getHeight()*bmp.getWidth()*4);
bb.order(ByteOrder.nativeOrder());
IntBuffer ib = bb.asIntBuffer();
```

– Bufor z bajtami bitmapy należy zapisać jako piksele, uwzględniając szerokość i wysokość bitmapy.

```
for (int y=0;y<bmp.getHeight();y++)
      for (int x=0;x<bmp.getWidth();x++) {
            ib.put(bmp.getPixel(x,y));
      }
ib.position(0);
bb.position(0);
```

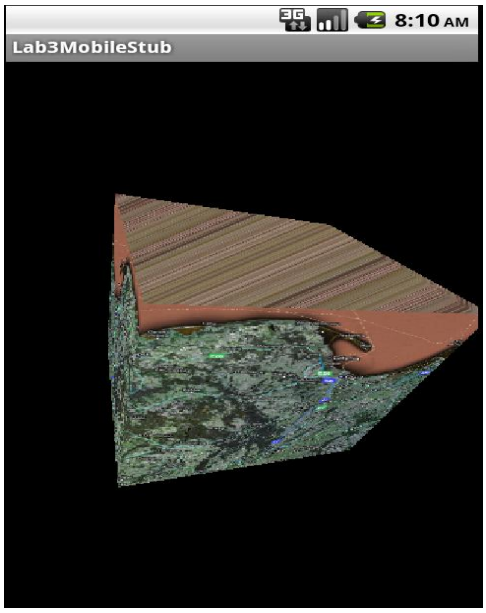
– Następnie należy wygenerować teksturę.

```
int[] tmp_tex = new int[1];

gl.glGenTextures(1, tmp_tex, 0);
int tekstura = tmp_tex[0];
```

– Utworzona tekstura zostaje podpieta i zwrócona:

```
gl.glBindTexture(GL10.GL_TEXTURE_2D, tekstura);
gl.glTexImage2D(GL10.GL_TEXTURE_2D, 0, GL10.GL_RGBA,
bmp.getWidth(), bmp.getHeight(), 0,
      GL10.GL_RGBA, GL10.GL_UNSIGNED_BYTE, bb);
return tekstura;
```

11. W wyniku działania programu powinniśmy otrzymać:



Rys 5. Przykładowa aplikacja