Programowanie urządzeń mobilnych

dr inż. Andrzej Grosser
na podstawie wykładu
dr inż. Juliusz Mikoda

Open GL

- Open GL – Open Graphics Library – specyfikacja uniwersalnego, wieloplatformowego API do generowania grafiki. Zestaw funkcji składa się z podstawowych wywołań, umożliwiających budowanie złożonych trójwymiarowych scen z podstawowych figur geometrycznych.
- OpenGL wykorzystywany jest często przez gry komputerowe i wygaszacze ekranu.
- Wiele programów przedstawiających wyniki badań naukowych (programy typu CAD) używa biblioteki OpenGL.

Open GL

- Biblioteki pomocnicze:
  - GLU (ang. GL Utility library) ,
  - GLUT (ang. GL Utility Toolkit),
  - GLX (w przypadku środowiska X Window System),
  - WGL (w przypadku Windows),
  - SDL (ang. Simple DirectMedia Layer),

Open GL ES

- OpenGL ES – OpenGL for Embedded Systems – to podzbiór OpenGL 3D zaprojektowany m.in. dla urządzeń mobilnych typu telefony komórkowe, palmtopy i konsole do gier.
- OpenGL ES jest oficjalnym API dla grafiki 3D w systemach Symbian, Android, iOS.
- Ważniejsze różnice do pełnej wersji OpenGL: usunięcie glBegin ... glEnd – semantyka dla prymitywów, wprowadzenie stałej długości atrybutów zmiennoprzecinkowych (przyspieszenie obliczeń) i wiele innych.

Open GL ES dla Androida

- OpenGL ES 1.0 – wspomagany na platformie android.
- OpenGL ES 1.1 – wprowadzony na platformie od wersji Android 1.6
- OpenGL ES 2.0 – wspierany na platformie Android od wersji 2.2
- OpenGL ES 2.0 – wspierany w bibliotece NDK od wersji Android 2.0
- OpenGL ES 3.0 – wspierany w bibliotece NDK od wersji Android 4.3

Open GL ES - składowe

- void glVertexPointer (int size, int type, int stride, Buffer pointer) – służy do określenia zbioru (tablicy) punktów (wierzchołków) dla elementów rysowanych w kontekście graficznym.
  - size – liczba współrzędnych wierzchołka,
  - type – typ przekazywanej wartości,
  - stride – przestrzeń pomiędzy wartościami (elementy pomijane),
  - pointer – bufor (tabela) danych.

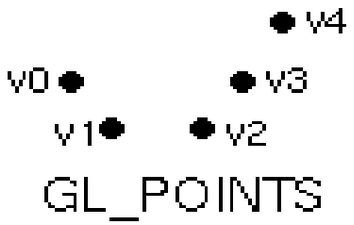
Open GL ES - składowe

- void glDrawElements (int mode, int count, int type, Buffer indices) – uruchomienie procedury rysowania prymitywów (elementów graficznych) według podanych parametrów oraz wczytanych wierzchołków:
  - mode – rodzaj rysowanego kształtu,
  - count – liczba indeksów (wierzchołków),
  - type – typ danych dla indeksów,
  - indices – bufor zawierający indeksy wierzchołków.

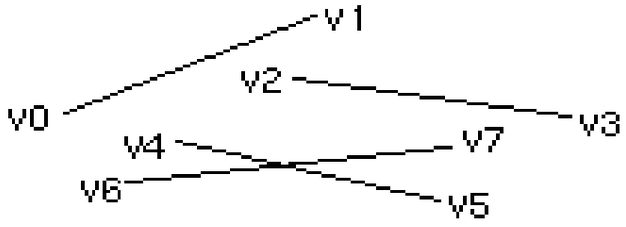
Open GL ES - składowe

- Rodzaj rysowanego kształtu (mode):
  - GL_POINTS lista punktów,
  - GL_LINES – lista linii (odcinków),
  - GL_LINE_STRIP – wstęga odcinków (łamana),
  - GL_LINE_LOOP – zamknięta wstęga odcinków,
  - GL_TRIANGLES – lista trójkątów,
  - GL_TRIANGLE_STRIP - wstęga trójkątów,
  - GL_TRIANGLE_FAN – wachlarz trójkątów,
  - GL_QUADS - czworościany,
  - GL_QUAD_STRIP - wstęga czworościanów,
  - GL_POLYGON - wielokąt.

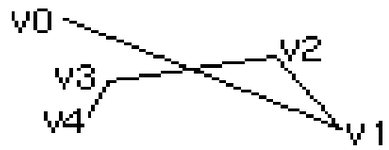
Open GL ES - składowe



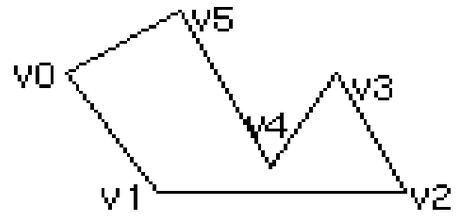
GL_POINTS



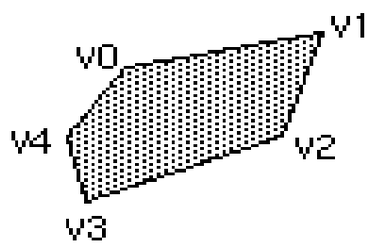
GL_LINES



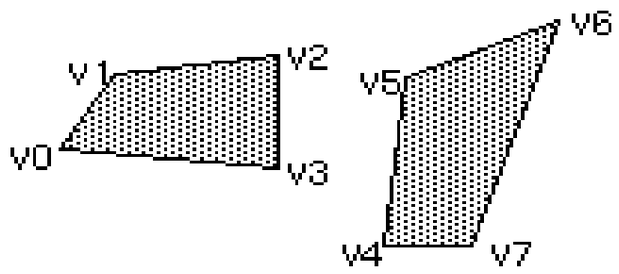
GL_LINE_STRIP



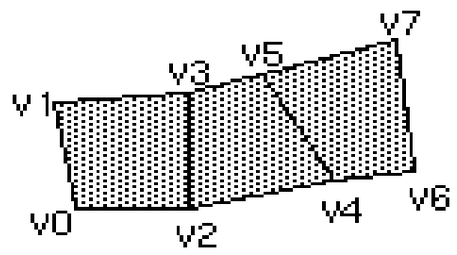
GL_LINE_LOOP



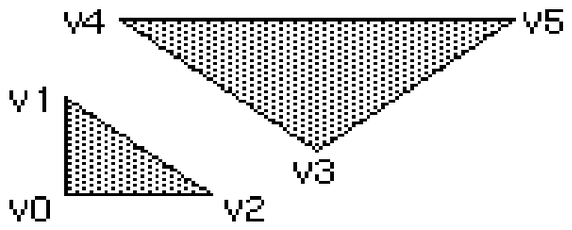
GL_POLYGON



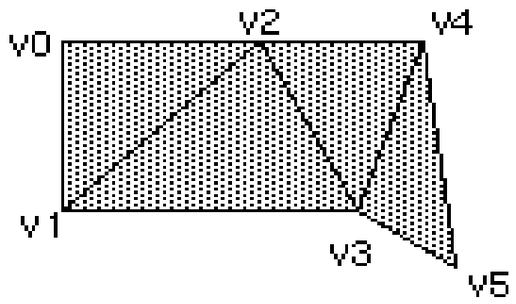
GL_QUADS



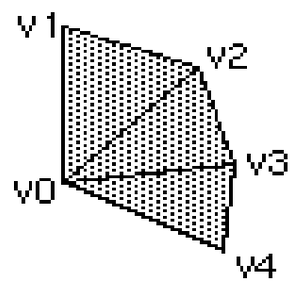
GL_QUAD_STRIP



GL_TRIANGLES



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

Open GL ES - składowe

- glClear(int mask) – wywołanie metody powoduje wymazanie odpowiedniego bufora danych:

- GL_COLOR_BUFFER_BIT – Bufor koloru (obrazu). Wymazanie tego bufora powoduje ustawienie domyślnego koloru,

- GL_DEPTH_BUFFER_BIT – Bufor głębokości – odpowiedzialny za badanie zakrytych obszarów rysunku,

- GL_STENCIL_BUFFER_BIT – Bufor szablonu - bufor stosowany do tworzenia obrazów z odbiciem obrazu (lustro).

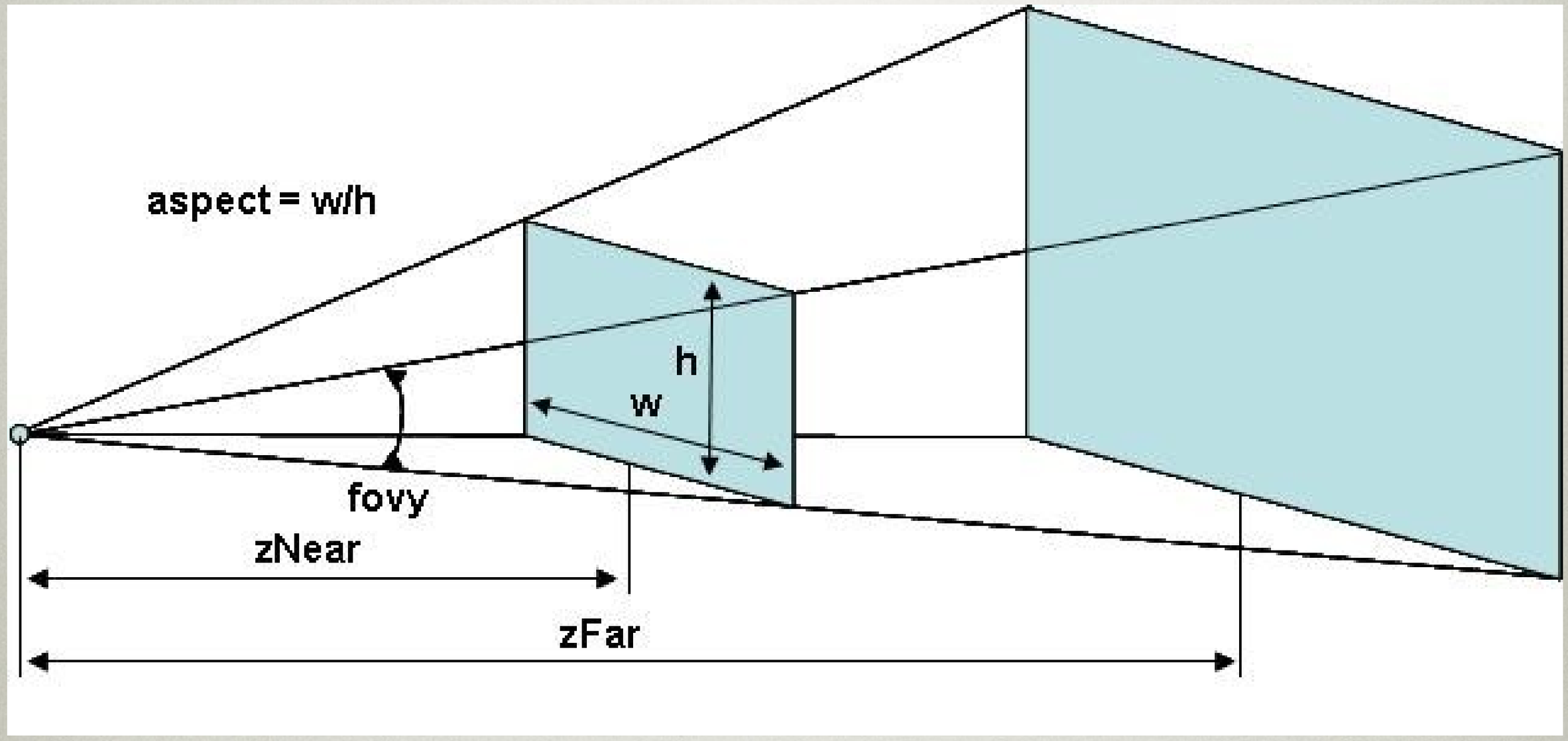
Open GL ES - składowe

- glColor4f(float red, float green, float blue, float alpha), glColor4x(int red, int green, int blue, int alpha) – ustawienie barwy, za pomocą której będzie narysowany prymityw.
  - Kolor można określić wartością rzeczywistą (0.0 – 1.0) oraz wartością całkowitą (0-255).
- glColorPointer(int size, int type, int stride, Buffer pointer) - kolor może być przekazany dla każdego wierzchołka (ściany rysowanych prymitywów będą cieniowane).
  - Wywołanie analogiczne jak w przypadku metody glVertexPointer.

Open GL ES - składowe

- glViewport(int x, int y, int width, int height) – określenie rozmiaru okna graficznego – obszaru w którym zostanie narysowany obraz.
- glOrthof(float left, float right, float bottom, float top, float zNear, float zFar) – rzut prostokątny – bez zastosowania perspektywy
- glFrustumf(float left, float right, float bottom, float top, float zNear, float zFar) - utworzenie tablicy projekcji (rzutowania) obrazu.
- gluPerspective(GL10 gl, float fovy, float aspect, float zNear, float zFar) – ustawienie widoku perspektywy.

Open GL ES - składowe




```
aspect = (float) w / (float) h;
```

fovy - kąt pola widzenia w stopniach, w kierunku y.

Open GL ES - składowe

- gluLookAt(GL10 gl, float eyeX, float eyeY, float eyeZ, float centerX, float centerY, float centerZ, float upX, float upY, float upZ) – kontroluje położenie kamery widoku.
- glLoadIdentity() – zerowanie macierzy transformacji.
- glRotatef(float angle, float x, float y, float z) – obroty wokół jednej z osi o zadany kąt.
- glTranslatef(float x, float y, float z) – translacja (przemieszczenie) obrazu o zadany wektor.

Open GL ES – ekran

- Kolejne etapy tworzenia kontekstu rysowania:
  - Implementacja interfejsu Renderer,
  - Ustawienie obiektu kamery w klasie implementującej interfejs Renderer,
  - Implementacja procedury rysowania w metodzie onDrawFrame (interfejs Renderer),
  - Konstrukcja obiektu GLSurfaceView,
  - Konfiguracja interfejsu renderującego GLSurfaceView,

Open GL ES – przykład

```
public class OpenGl extends Activity {
    private GLSurfaceView glsv;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    glsv = new GLSurfaceView(this);
    // Wyłączenie bufora głębokości
    glsv.setEGLConfigChooser(false);
    // Ustawienie obiektu renderującego
    glsv.setRenderer(new SimpleTriangleRenderer(this));
    // Statyczny tryb renderowania.
    // Przerysowanie wymaga wywołania requestRender ()
     glsv.setRenderMode(
        GLSurfaceView.RENDERMODE_WHEN_DIRTY);
     // Tryb automatcznego renderowania sceny
     // glsv.setRenderMode(
     //    GLSurfaceView.RENDERMODE_CONTINUOUSLY);
     setContentView(glsv);
}
```

Open GL ES – przykład

```
public class OpenGl extends Activity {

   // ... ... ...

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

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

Open GL ES – przykład

```
public abstract class AbstractRenderer
                implements Renderer {

  @Override
  public void onSurfaceCreated(GL10 gl, EGLConfig conf)
{
    gl.glClearColor(0.5f, 0.5f, 0.5f, 0.5f);
    gl.glEnable(GL10.GL_DEPTH_TEST);
  }

  @Override
  public void onSurfaceChanged(GL10 gl, int w, int h) {
    gl.glViewport(0,0,w,h);
    float aspect = (float) w / h;
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity();
    gl.glFrustumf(-aspect, aspect, -1,1,3,7);
    gl.glMatrixMode(GL10.GL_MODELVIEW);
  }
```

Open GL ES – przykład

```
public abstract class AbstractRenderer
                implements Renderer {

  @Override
  public void onDrawFrame(GL10 gl) {
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
               GL10.GL_DEPTH_BUFFER_BIT);

    gl.glLoadIdentity();
    GLU.gluLookAt(gl, 0,0,-5, 0f,0f,0f,0f,1.0f,0.0f);
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

    draw(gl);
  }

  protected abstract void draw(GL10 gl);

}
```

Open GL ES – przykład

```
public class SimpleTriangleRenderer
        extends AbstractRenderer {

  private final static float [] coords = {
                          -0.5f, -0.5f, 0.0f,
                           0.5f, -0.5f, 0.0f,
                           0.0f,  0.5f, 0.0f
                        };

  private final static short [] indexes = { 0, 1, 2 };

  private final static int VERTEX = 3;

  // Bufor współrędnych
  private FloatBuffer vertex;
  // Bufor indeksów wierzchołków
  private ShortBuffer index;
```

Open GL ES – przykład

```
public class SimpleTriangleRenderer
        extends AbstractRenderer {

  SimpleTriangleRenderer(Context context)
  {
    ByteBuffer bb;

    bb = ByteBuffer.allocateDirect(VERTEX * 3 * 4);
    bb.order(ByteOrder.nativeOrder());
    vertex = bb.asFloatBuffer();

    bb = ByteBuffer.allocateDirect(VERTEX * 3);
    bb.order(ByteOrder.nativeOrder());
    index = bb.asShortBuffer();

    for (int i = 0; i < VERTEX; ++i {
      for (int j = 0; j < 3; ++j) {
        vertex.put(coords[i * 3 + j]);
      }
    }
```

Open GL ES – przykład

```
public class SimpleTriangleRenderer
        extends AbstractRenderer {
  SimpleTriangleRenderer(Context context)
  {
    // ..........
    for (int i = 0; i < indexes.length; ++ i) {
      index.put(indexes[i]);
    }
    vertex.position(0);
    index.position(0);
  }

  protected void draw(GL10 gl) {
    gl.glColor4f(1.0f, 0.0f, 0.0f, 0.5f);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertex);
    gl.glDrawElements(GL10.GL_TRIANGLES, 3,
                      GL10.GL_UNSIGNED_SHORT, index);
  }
}
```

Open GL ES - Obsługa zdarzeń

- boolean onTouchEvent (MotionEvent event)
- final class MotionEvent
  - getAction() – typ zdarzenia ACTION_DOWN, ACTION_MOVE, ACTION_UP, or ACTION_CANCEL,
  - float getX(), float getY() – współrzędne wystąpienia zdarzenia,
  - int getPointerCount (), float getX (int index), loat getY (int index) – wszystkie punkty w tym zdarzeniu.
  - ...

Open GL ES - Obsługa zdarzeń

```
public class OpenGlTouch extends Activity {
private GLSurfaceView mGLView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mGLView = new ClearGLSurfaceView(this);
        setContentView(mGLView);
    }
    @Override
    protected void onPause() {
        super.onPause();
        mGLView.onPause();
    }
    @Override
    protected void onResume() {
        super.onResume();
        mGLView.onResume();
    }
}
```

Open GL ES - Obsługa zdarzeń

```
class ClearGLSurfaceView
      extends GLSurfaceView {

  ClearRenderer mRenderer;

  public ClearGLSurfaceView(Context context) {
     super(context);
     mRenderer = new ClearRenderer();
     setRenderer(mRenderer);
  }

  public boolean onTouchEvent(final MotionEvent event)
{
     mRenderer.setColor(event.getX() / getWidth(),
                        event.getY() / getHeight(),
                        1.0f );
     return true;
  }

}
```

Open GL ES - Obsługa zdarzeń

```
class ClearRenderer implements GLSurfaceView.Renderer {
  public void onSurfaceCreated(GL10 gl, EGLConfig config) {
  }

  public void onSurfaceChanged(GL10 gl, int w, int h) {
    gl.glViewport(0, 0, w, h);
  }

  public void onDrawFrame(GL10 gl) {
    gl.glClearColor(mRed, mGreen, mBlue, 1.0f);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
  }

  public void setColor(float r, float g, float b) {
    mRed = r;     mGreen = g;     mBlue = b;
  }

  private float mRed, mGreen, mBlue;
}
```

OGL – Sfera - OpenGLSphere

```
public class OpenGLSphere extends Activity {
  private GLSurfaceView mGLView;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Pełny ekran bez tytułu
    this.requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().setFlags(
       WindowManager.LayoutParams.FLAG_FULLSCREEN,
       WindowManager.LayoutParams.FLAG_FULLSCREEN);
    // Kontekst renderowania
    mGLView = new ViewSurface(this);
    setContentView(mGLView);
  }
  protected void onPause() { //...  }
  protected void onResume() { //...  }
}
```

OGL – Sfera - ViewSurface

```
class ViewSurface extends GLSurfaceView {


  ViewRenderer mRenderer;

  // Punkt dotyku ekranu
  private float x, y, o = 0;

  public ViewSurface(Context context) {
    super(context);
    // Moduł renderowania obrazu
    mRenderer = new ViewRenderer();
    setRenderer(mRenderer);
    SetRenderMode(
        GLSurfaceView.RENDERMODE_CONTINUOUSLY);
  }
```

OGL – Sfera - ViewSurface

```
public boolean onTouchEvent(final MotionEvent event) {
  switch (event.getAction()) {
    // Zdarzenie dotknięcia ekranu
    case MotionEvent.ACTION_DOWN :
      x = event.getX();   y =event.getY();   o = 1;
      break;
    // Zdarzenie zwolnienia nacisku
    case MotionEvent.ACTION_UP :
      o = 0;
      break;
    // Zdarzenie przesunięcia po ekranie
    case MotionEvent.ACTION_MOVE :
      if (o > 0) mRenderer.rotate(y - event.getY(),
                                  x - event.getX());
      x = event.getX(); y = event.getY();
      break;
  }
  return true;
 }
}
```

OGL – Sfera - ViewRenderer

```
class ViewRenderer implements GLSurfaceView.Renderer {
  // Stałe pomocnicze
  private final static float X = 0.52573111211913360 6f;
  private final static float Z = 0.85065080835203993 2f;

  // wierzchołki
  private final static float [] vdata = {
    -X, 0, Z,     X, 0, Z,     -X, 0,-Z,     X, 0,-Z,
     0, Z, X,      0, Z,-X,      0,-Z, X,      0,-Z,-X,
     Z, X, 0,     -Z, X, 0,      Z,-X, 0,     -Z,-X, 0
};

// kolory wierzchołków
private final static float [] cdata = {
  1,0,0,1,  0,1,0,1,  0,0,1,1,  1,1,0,1,
  1,0,1,1,  0,1,1,1,  X,0,0,1,  0,X,0,1,
  0,0,X,1,  X,X,0,1,  X,0,X,1,  0,X,X,1,
};
```

OGL – Sfera - ViewRenderer

```
// opis powierzchni dwudziestościanu
private final static short [] idata = {
    0,4,1,   0,9,4,   9,5,4,   4,5,8,   4,8,1,
   8,10,1,  8,3,10,   5,3,8,   5,2,3,   2,7,3,
   7,10,3,  7,6,10,  7,11,6,  11,0,6,   0,1,6,
   6,1,10,  9,0,11,  9,11,2,   9,2,5,  7,2,11 };

// bufory danych
private FloatBuffer vertex, color;
private ShortBuffer index;
// liczba trójkątów do narysowania
private int ni;

// współczynnik rotacji obrazy
private float rx = 0, ry = 0;

// macierz pomocnicza rotacji
private float trans[] =
            {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1};
```

OGL – Sfera - ViewRenderer

```
ViewRenderer()
{  // tworzenie buforów danych do renderowania
   vertex = ByteBuffer.allocateDirect(vdata.length * 4)
      .order(ByteOrder.nativeOrder()).asFloatBuffer();
   color = ByteBuffer.allocateDirect(cdata.length * 4)
      .order(ByteOrder.nativeOrder()).asFloatBuffer();
   index = ByteBuffer.allocateDirect(idata.length * 2)
      .order(ByteOrder.nativeOrder()).asShortBuffer();

   vertex.put(vdata);
   color.put(cdata);
   index.put(idata);

   vertex.position(0);
   color.position(0);
   index.position(0);

   ni = idata.length ;
}
```

OGL – Sfera - ViewRenderer

```
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
  // inicjacja kontekstu renderowania
  gl.glClearColor(0.1f, 0.1f, 0.1f, 0.5f);

  // test głębokości
  gl.glEnable(GL10.GL_DEPTH_TEST);

  // rysowanie widocznych ścianek
  gl.glEnable(GL10.GL_CULL_FACE);
  gl.glFrontFace(GL10.GL_CCW);
  gl.glCullFace(GL10.GL_BACK);
}
```

OGL – Sfera - ViewRenderer

```
public void onSurfaceChanged(GL10 gl, int w, int h)
{

  // inicjacja widoku
  gl.glViewport(0,0,w,h);
  float aspect = (float) w / (float) h;

  gl.glMatrixMode(GL10.GL_PROJECTION);
  gl.glLoadIdentity();
  // gl.glFrustumf(-aspect, aspect, -1,1,-1,1);

  // widok perspektywiczny
  GLU.gluPerspective(gl, 40.0f, aspect, -1f, 2.0f);

  gl.glMatrixMode(GL10.GL_MODELVIEW);

}
```

OGL – Sfera - ViewRenderer

```
public void onDrawFrame(GL10 gl) {
    // Czyszczenie buforów
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
               GL10.GL_DEPTH_BUFFER_BIT);

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glEnableClientState(GL10.GL_COLOR_ARRAY);

    // transformacje - obroty
    gl.glLoadIdentity();
    float tmp[] = {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1};
    Matrix.rotateM(tmp, 0, rx, 1, 0, 0);
    Matrix.rotateM(tmp, 0, ry, 0, 1, 0);
    Matrix.multiplyMM(trans, 0, tmp, 0, trans, 0);
    gl.glMultMatrixf(trans, 0);
    rx = ry = 0;

//   gl.glRotatef(rx, 1, 0, 0);
//   gl.glRotatef(ry, 0, 1, 0);
```

OGL – Sfera - ViewRenderer

```
    // rysowanie obrazy sfery
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertex);
    gl.glColorPointer(4, GL10.GL_FLOAT, 0, color);
    gl.glDrawElements(GL10.GL_TRIANGLES, ni,
                      GL10.GL_UNSIGNED_SHORT, index);
}
```

```
    public void rotate(float x, float y)
    {  // zmiana obrotu
       rx += x;
       ry += y;
    }

}
```

Open GL - Teksturowanie

```
public class OpenGLText extends Activity {
    private GLSurfaceView glsv;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        glsv = new GLSurfaceView(this);
        // Wyłączenie bufora głębokości
        glsv.setEGLConfigChooser(false);
        // Ustawienie obiektu renderującego
        glsv.setRenderer(new TexturedSquareRenderer(this));
        // Statyczny tryb renderowania.
        // Przerysowanie wymaga wywołania requestRender ()
        glsv.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
        setContentView(glsv);
    }
    protected void onResume() {
        super.onResume();  glsv.onResume();
    }
    protected void onPause() {
        super.onPause();   glsv.onPause();
    }
}
```

Open GL - Teksturowanie

```
abstract class AbstractRenderer implements Renderer {

    int mTextureID;
    int mImageResourceId;
    Context mContext;

    public AbstractRenderer(Context ctx, int imageResourceId) {
        mImageResourceId = imageResourceId;
        mContext = ctx;
    }

    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig conf) {
        gl.glClearColor(0.5f, 0.5f, 0.5f, 0.5f);
        gl.glEnable(GL10.GL_DEPTH_TEST);
        prepareTexture(gl);
    }
```

Open GL - Teksturowanie

```
private void prepareTexture(GL10 gl)
{
    int[] textures = new int[1];
    gl.glGenTextures(1, textures, 0);

    mTextureID = textures[0];
    gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);
    gl.glTexParameterf(GL10.GL_TEXTURE_2D,
        GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
    gl.glTexParameterf(GL10.GL_TEXTURE_2D,
        GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
    gl.glTexParameterf(GL10.GL_TEXTURE_2D,
        GL10.GL_TEXTURE_WRAP_S, GL10.GL_CLAMP_TO_EDGE);
    gl.glTexParameterf(GL10.GL_TEXTURE_2D,
        GL10.GL_TEXTURE_WRAP_T, GL10.GL_CLAMP_TO_EDGE);
    gl.glTexEnvf(GL10.GL_TEXTURE_ENV,
        GL10.GL_TEXTURE_ENV_MODE, GL10.GL_REPLACE);
```

Open GL - Teksturowanie

```
        InputStream is = mContext.getResources()
            .openRawResource(this.mImageResourceId);
        Bitmap bitmap;
        bitmap = BitmapFactory.decodeStream(is);
        GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
        bitmap.recycle();
    }

    public void onSurfaceChanged(GL10 gl, int w, int h) {
        gl.glViewport(0,0,w,h);
        float aspect = (float) w / h;
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        gl.glFrustumf(-aspect, aspect, -1,1,3,7);
        gl.glMatrixMode(GL10.GL_MODELVIEW);
    }
```

Open GL - Teksturowanie

```
public void onDrawFrame(GL10 gl) {
    gl.glDisable(GL10.GL_DITHER);
    gl.glTexEnvx(GL10.GL_TEXTURE_ENV,
        GL10.GL_TEXTURE_ENV_MODE, GL10.GL_MODULATE);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
        GL10.GL_DEPTH_BUFFER_BIT);
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 0f, 1.0f, 0.0f);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
        gl.glActiveTexture(GL10.GL_TEXTURE0);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);
        gl.glTexParameterx(GL10.GL_TEXTURE_2D,
            GL10.GL_TEXTURE_WRAP_S, GL10.GL_REPEAT);
        gl.glTexParameterx(GL10.GL_TEXTURE_2D,
            GL10.GL_TEXTURE_WRAP_T, GL10.GL_REPEAT);
        draw(gl);
    }
    protected abstract void draw(GL10 gl);
}
```

Open GL - Teksturowanie

```
class TexturedSquareRenderer extends AbstractRenderer
{
    //Nieskompresowany bufor natywny, współrzędne punktów
    private FloatBuffer mFVertexBuffer;
    //Nieskompresowany bufor natywny, współrzędne tekstury
    private FloatBuffer mFTextureBuffer;
    // Nieskompresowany bufor natywny, przechowujący indeksy
    // pozwalające na wielokrotne wykorzystywanie punktów.
    private ShortBuffer mIndexBuffer;

    private int numOfIndices = 0;

    private int sides = 4;

    public TexturedSquareRenderer(Context context)
    {
        super(context, R.drawable.icon);
        prepareBuffers(sides);
    }
```

Open GL - Teksturowanie

```
private void prepareBuffers(int sides)
{
    RegularPolygon t = new RegularPolygon(0,0,0,0.8f,sides);
    this.mFVertexBuffer = t.getVertexBuffer();
    this.mFTextureBuffer = t.getTextureBuffer();
    this.mIndexBuffer = t.getIndexBuffer();
    this.numOfIndices = t.getNumberOfIndices();
    this.mFVertexBuffer.position(0);
    this.mIndexBuffer.position(0);
    this.mFTextureBuffer.position(0);
  }

  protected void draw(GL10 gl)
  {
    prepareBuffers(sides);
    gl.glEnable(GL10.GL_TEXTURE_2D);
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFVertexBuffer);
    gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mFTextureBuffer);
    gl.glDrawElements(GL10.GL_TRIANGLES, this.numOfIndices,
    GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
  }
}
```