```
    <scale
        android:duration="2000"
        android:startOffset="2000"
        android:fromXScale="1"
        android:fromYScale="1"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="3"
        android:toYScale="3" />
    <translate
        android:startOffset="6000"
        android:duration="2000"
        android:fromXDelta="0"
        android:toXDelta="-100" />
    <alpha android:fromAlpha="1.0"
        android:startOffset="8000"
        android:toAlpha="0.0"
        android:duration="1000"
        android:repeatMode="reverse"
        android:repeatCount="infinite"/>
</set>
```

9.5. Podstawy grafiki 3D

Tworzenie grafiki 3D w urządzeniach z systemem Android jest możliwe dzięki zastosowaniu modułu OpenGL ES. OpenGL ES jest podzbiorem standardu OpenGL dostosowanym do urządzeń z systemem Android. Podstawowym sposobem korzystania z OpenGL ES jest zastosowanie następujących klas:

- `GLSurfaceView`,
- `GLSurfaceView.Renderer`.

Klasa `GLSurfaceView` obsługuje uruchomienie niezbędnych bibliotek i zarządza wątkami związanymi z rysowaniem. Jej zastosowanie pozwala także na synchronizację cyklu życia klasy z cyklem życia Aktywności, w której została zastosowana. Klasa `GLSurfaceView.Renderer` zarządza procesem rysowania i umożliwia programiście definiowanie własnych implementacji elementów grafiki.

Zastosowanie klas Open GL ES jest dużym ułatwieniem dla programisty, ponieważ umożliwia:

- łatwą implementację wyświetlania grafiki;
- łatwą integrację z cyklem życia Aktywności;

- odciążenie od obowiązku ręcznego zarządzania wątkami związanymi z wyświetlaniem grafiki;
- odciążenie od obowiązku ręcznego zarządzania bibliotekami OpenGL ES.

W pierwszej kolejności przedstawiony został przykład zastosowania prostej aplikacji z wykorzystaniem klas `GLSurfaceView` oraz `GLSurfaceView.Renderer`. Przykład ten koncentruje się na odpowiednim sposobie wkomponowania tych obiektów do własnej aplikacji, aby w prawidłowy sposób z nią współdziałały. W efekcie aplikacja wyświetli szare tło na całym ekranie (`gl.glClearColor (0.5f, 0.5f, 0.5f, 1.0f);`).

```
package com.example.openglszkielet;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.FloatBuffer;
import java.nio.ShortBuffer;
import java.util.Random;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {

   private GLSurfaceView glView;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Utworzenie powierzchni do rysowania
      glView = new GLSurfaceView(this);
      // Utworzenie obiektu zarządzającego rysowaniem,
      glView.setRenderer(new ClearRenderer());
      // Ustawienie utworzonej powierzchni jako interfejsu
      // Aktywności
      setContentView(glView);
   }
   @Override
   protected void onPause() {
```

```
        //Synchronizacja cyklu Aktywności z cyklem
        //powierzchni rysowania
            super.onPause();
            glView.onPause();
    }

    @Override
    protected void onResume() {
        //Synchronizacja cyklu Aktywności z cyklem
        //powierzchni rysowania
            super.onResume();
            glView.onResume();
    }

    class ClearRenderer implements GLSurfaceView.Renderer {

        public void onSurfaceChanged(GL10 gl, int w, int h) {
            //Ustawienie wielkości obszaru rysowania na cały ekran
            gl.glViewport(0, 0, w, h);
        }

        public void onDrawFrame(GL10 gl) {
        //Metoda wywoływana non stop (wywoływana co każdą ramkę)
        //Działa tylko wtedy, gdy działa aktywność
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10 ⏎
                                            .GL_DEPTH_BUFFER_BIT);
        //Ustawienie koloru tła na szary bez przezroczystości
        gl.glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
    }

    @Override
    public void onSurfaceCreated(GL10 arg0, EGLConfig arg1) {
        // Operacje wykonywane w momencie utworzenia obszaru
        // rysowania

        }
    }

}
```

W metodach Aktywności `onPause()` i `onResume()` zaimplementowano synchronizację w taki sposób, aby wątek związany z grafiką był wstrzymywany

i wznawiany w przypadku zatrzymania lub wznowienia działania Aktywności. Takie rozwiązanie przede wszystkim oszczędza zasoby urządzenia.

W metodzie `onSurfaceCreated` należy zaimplementować elementy wymagane przy pierwszym tworzeniu powierzchni przeznaczonej do rysowania grafiki. W metodzie `onSurfaceChanged()` należy zaimplementować elementy związane ze zmianą wielkości obszaru wyświetlania. Przykładowo, metoda umożliwia uzyskanie odpowiedniej reakcji na zmianę obszaru rysowania w momencie zmiany orientacji ekranu urządzenia. Istotną metodą umożliwiającą programiście implementację własnej funkcjonalności jest metoda `onDrawFrame()`. Wywoływana jest ona cyklicznie (ramka obrazu) i umożliwia wprowadzanie zmian w obszarze rysowania.

Dotychczasowe działania nie umożliwiały wprowadzenia interakcji z użytkownikiem. Aby prawidłowo zaimplementować zdarzenia związane z interakcją z użytkownikiem, należy zdefiniować własną klasę korzystającą z klasy `GLSurfaceView` z implementacją metod obsługujących odpowiednie zdarzenia. W kolejnym przykładzie przedstawiono interakcję polegającą na odczytywaniu współrzędnych miejsca, w którym użytkownik dotknął ekranu. Odczytane współrzędne w dodanej metodzie `onTouchEvent()` wykorzystywane są następnie do ustawienia nasycenia poszczególnych barw RGB składających się na kolor tła. Ponieważ nasycenie danego koloru reprezentowane jest przez wartości z przedziału od 0 do 1, wartości składowych obliczono zgodnie ze wzorem:

- `r = event.getX() / getWidth()`,
- `g = event.getY() / getHeight()`,
- `b = 1- event.getX() / getWidth()`.

Obliczone wartości przekazywane są do obiektu typu `GLSurfaceView.Renderer`, który odpowiedzialny jest za rysowanie tła, z wykorzystaniem dodanej metody `setColor()`. Pełny kod aplikacji wygląda następująco:

```
package com.example.openglinterakcja;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.app.Activity;
import android.content.Context;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
import android.view.MotionEvent;

public class MainActivity extends Activity {
   private GLSurfaceView glView;
   @Override
```

```
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      glView = new MojeGLSurfaceView(this);
      setContentView(glView);
   }

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

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

// Rozszerzenie klasy GLSurfaceView
// Dodanie metody onTouchEvent
class MojeGLSurfaceView extends GLSurfaceView {
         MojRenderer renderer;

   public MojeGLSurfaceView(Context context) {
      super(context);
      renderer = new MojRenderer();
      setRenderer(renderer);
   }

   public boolean onTouchEvent(final MotionEvent event) {
      queueEvent(new Runnable(){
         public void run() {
            renderer.setColor(
               event.getX() / getWidth(),
               event.getY() / getHeight(),
               1- event.getX() / getWidth());
         }});
         return true;
      }
}
```

```
// Rozszerzenie klasy GLSurfaceView.Renderer
// Dodanie metody setColor umożliwiającej
// zmianę kolorów powierzchni do rysowania
class MojRenderer implements GLSurfaceView.Renderer {

   private float red;
   private float green;
   private float blue;

   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(red, green, blue, 1.0f);
      gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10 ↵
                                          .GL_DEPTH_BUFFER_BIT);
   }

   public void setColor(float r, float g, float b) {
      red = r;
      green = g;
      blue = b;
   }
}
```

Kolejny przykład prezentuje sposób rysowania trójkąta (figura płaska) z możliwością zobrazowania obrotów przestrzennych. W przykładzie zaimplementowano także interakcję polegającą na zmianie kąta ustawienia figury przez dotknięcie ekranu w wybranym miejscu. W dalszej części znajduje się pełny kod z dodatkowymi komentarzami. Schemat implementacji jest identyczny do zastosowanego we wcześniejszym przykładzie. Dodana została klasa `Trojkat()` opisująca właściwości graficzne figury z metodami jej rysowania. W metodach obiektu `Renderer` wprowadzono odpowiednie instrukcje umożliwiające prawidłowe wyświetlanie trójkąta. Szczegółowe omówienie wszystkich instrukcji wykracza poza ramy publikacji.

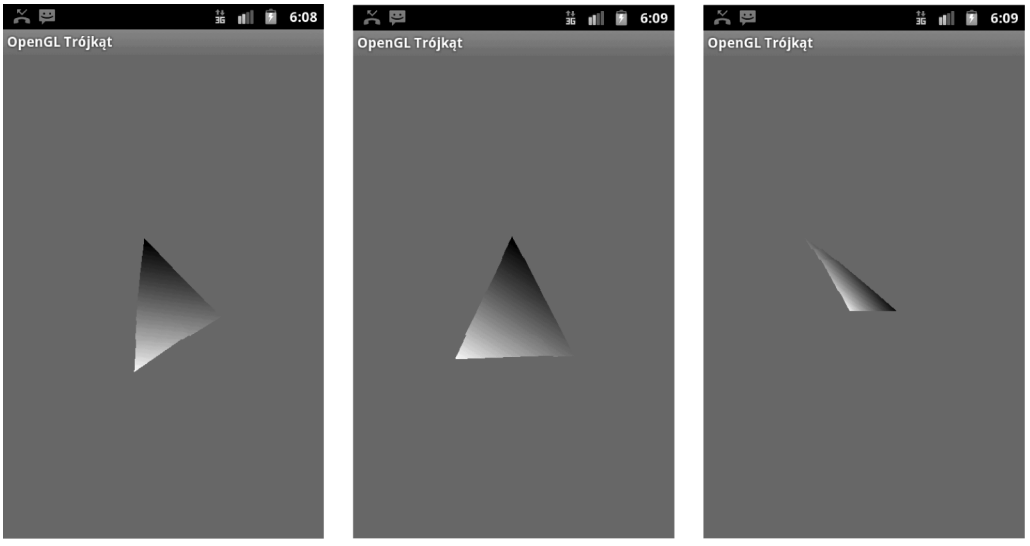
```
package com.example.opengltrojkat;

import java.nio.ByteBuffer;
```

```
...
public class MainActivity extends Activity {

   private GLSurfaceView glView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      // j.w.
   }

   @Override
   protected void onPause() {
      // j.w.
   }

   @Override
   protected void onResume() {
      // j.w.
   }
}

// Rozszerzenie klasy GLSurfaceView
// Dodanie metody onTouchEvent
class MojeGLSurfaceView extends GLSurfaceView {
      MojRenderer renderer;

   public MojeGLSurfaceView(Context context) {
      super(context);
      renderer = new MojRenderer();
      setRenderer(renderer);
   }

   public boolean onTouchEvent(final MotionEvent event) {
      queueEvent(new Runnable(){
         public void run() {
            // Na podstawie współrzędnej X miejsca dotknięcia
            // ekranu wyznaczany jest kąt ustawienia trójkąta
            renderer.UstawKat(event.getX());
         }});
         return true;
      }
}
```

```
// Rozszerzenie klasy GLSurfaceView.Renderer
// Dodanie metody setColor umożliwiającej
// zmianę kolorów powierzchni do rysowania
class MojRenderer implements GLSurfaceView.Renderer {

   private Trojkat tr = new Trojkat();
   private float kat = 100f;
   private float zmiana = -1f;

   public void onSurfaceCreated(GL10 gl, EGLConfig config) {
         // Ustalenie koloru tła
      gl.glClearColor(0.5f, 0.5f, 0.5f, 1f);
      gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10 ⏎
                                                    .GL_NICEST);
      gl.glEnable(GL10.GL_DEPTH_TEST);

   }

   public void onSurfaceChanged(GL10 gl, int w, int h) {
      gl.glViewport(0, 0, w, h);
      float wsp = (float)w / h;
      gl.glMatrixMode(GL10.GL_PROJECTION);
      gl.glLoadIdentity();
      gl.glFrustumf(-wsp, wsp, -1.0f, 1.0f, 1.0f, 10.0f);

   }

   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(0.0f, 0.0f, -4.0f);
      gl.glRotatef(kat, 1.0f, 1.0f, 1.0f);
      tr.draw(gl);
      if (kat <0 ) zmiana = 1f;
      if   (kat > 100 ) zmiana = -1f;
      kat += zmiana;
   }

   public void UstawKat(float k) {
      // Ustawienie wartości aktualnego kąta
      kat = k;
   }
```

```
// Definicja trójkąta
public class Trojkat {

   // Zmienne buforów dla danych trójkąta
   private FloatBuffer wierzcholkiBuffer = null;
   private ShortBuffer indeksyBuffer = null;
   private FloatBuffer koloryBuffer = null;

   // Definicja wierzchołków trójkąta
   // "Lewy dolny", "prawy dolny", "górny"
   private float wierzcholkiList[] = {
      -1.0f,-1.0f,0.0f,
      1.0f,-1.0f,0.0f,
      0.0f, 1.0f,0.0f };

   // Numery indeksów wierzchołków
   private short indeksyList[] = { 0,1,2 };

   // Kolory wierzchołków
   private float koloryList[] = {
      1.0f, 1.0f, 1.0f, 1.0f,
      0.5f, 0.5f, 0.5f, 1.0f,
      0.0f, 0.0f, 0.0f, 1.0f };

public Trojkat(){

   // Bufor wierzchołków
   ByteBuffer wBuffer = ByteBuffer.allocateDirect ↵
                              (wierzcholkiList.length * 4);
   wBuffer.order(ByteOrder.nativeOrder());
   wierzcholkiBuffer = wBuffer.asFloatBuffer();
   wierzcholkiBuffer.put(wierzcholkiList);
   wierzcholkiBuffer.position(0);

   // Bufor indeksów wierzchołków
   ByteBuffer iBuffer = ByteBuffer.allocateDirect ↵
                                  (indeksyList.length * 2);
   iBuffer.order(ByteOrder.nativeOrder());
   indeksyBuffer = iBuffer.asShortBuffer();
   indeksyBuffer.put(indeksyList);
   indeksyBuffer.position(0);

   // Bufor kolorów wierzchołków
   ByteBuffer cBuffer = ByteBuffer.allocateDirect ↵
                                   (koloryList.length * 4);
```

```
        cBuffer.order(ByteOrder.nativeOrder());
        koloryBuffer = cBuffer.asFloatBuffer();
        koloryBuffer.put(koloryList);
        koloryBuffer.position(0);

    }

    public void draw(GL10 gl){
        // Rysowanie trójkąta
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);

        gl.glColorPointer(4, GL10.GL_FLOAT, 0, this.koloryBuffer);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, ↵
                                        this.wierzcholkiBuffer);
        // Właściwe rysowanie
        gl.glDrawElements(GL10.GL_TRIANGLES, 3, ↵
                                    GL10.GL_UNSIGNED_SHORT, ↵
                                        this.indeksyBuffer);
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
        }
    }
}
```



Rysunek 72. Obroty przestrzenne figury z wykorzystaniem OpenGL ES