Лабораторна робота №2. Вказування кольорів об’єктів засобами OpenGL ES та організація інтерфейсу користувача застосунку

Мета: отримати навички програмування кольорів об’єктів для графіки OpenGL ES та меню користувача.

У цій лабораторній роботі буде продовжено знайомство з Android OpenGL ES, зокрема, будуть розглянуті питання визначення кольорів об’єктів.

Завдання

Потрібно створити у середовищі Android Studio проєкт з ім’ям **Lab2_GLES**, зокрема

- написати вихідний текст програми згідно варіанту завдання. Використати мову Java або Kotlin – на вибір
- Налагодити програму. Перевірити роботу програми на емуляторі та на фізичному пристрої Android.

Засоби для виконання лабораторної роботи

Android Studio – середовище розробки Android-застосунків. Потрібно було його інсталювати відповідно до вказівок для виконання попередньої лабораторної роботи №1.

Теоретичні положення та методичні рекомендації

Створення проєкту Android Studio – так само, як і для попередньої лаб. роботи №1. Варто нагадати, що для успішного використання OpenGL ES версії 3.2 потрібно вказати цільовий API рівня не нижче 28 (Android 9.0).

Спочатку розглянемо питання, які не відносяться безпосередньо до графіки, а саме організація меню застосунку Android.

Програмування меню застосунку Android

Перший крок. Потрібно додати стрічку для меню головного activity. При створенні нового проекту по замовчуванням цієї стрічки немає.

Знайдіть у списку компонентів проєкту – у вікні Project розділ: app–res–values–themes. Завантажте файл **themes.xml**.

У рядку визначення стилю, там де було записано

```
parent="Theme.Material3.DayNight.NoActionBar">
```

вилучіть **.NoActionBar.**

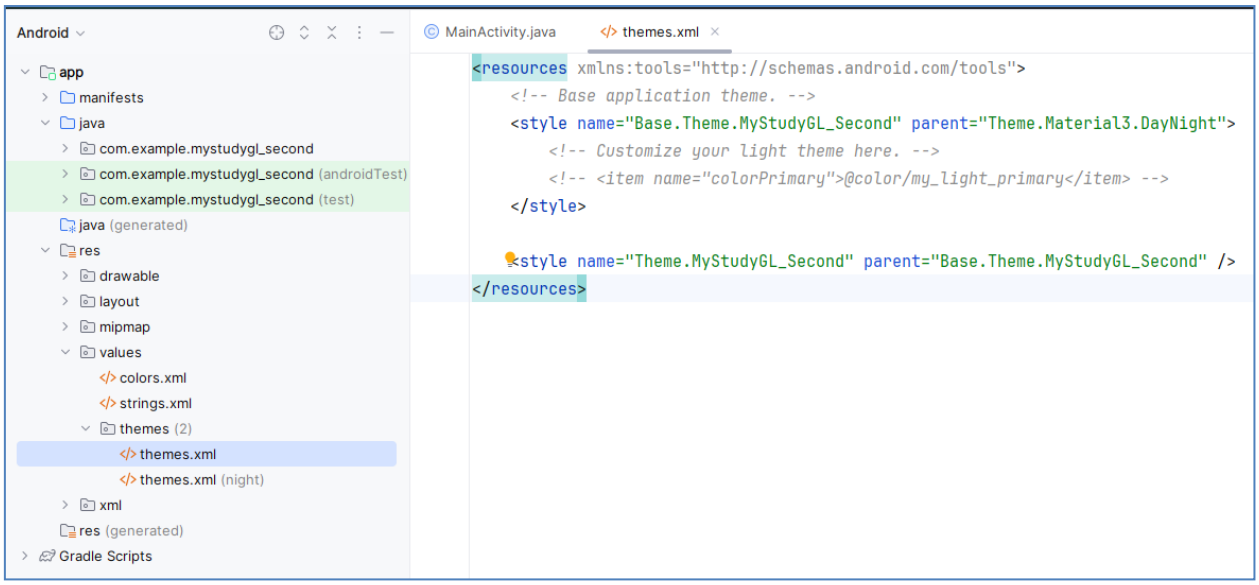


Рис. 1. Файл themes.xml

Так само вилучіть **.NoActionBar** у файлі themes.xml (night).

Другий крок. У класі MainActivity треба визначити метод onCreateOptionsMenu( ), у якому можна записати додавання потрібних пунктів меню з будь-якими іменами. Нижче наведено приклад формування меню з чотирма рядками з іменами “Menuitem 1”, “Menuitem 2” і т. д.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, 1, 0, "Menuitem 1");
    menu.add(0, 2, 0, "Menuitem 2");
    menu.add(0, 3, 0, "Menuitem 3");
    menu.add(0, 4, 0, "Menuitem 4");
    return true;
}
```

На цьому кроці рекомендується перевірити роботу застосунку в емуляторі. Переконайтеся, що меню відображається у верхній стрічці вікна застосунку.

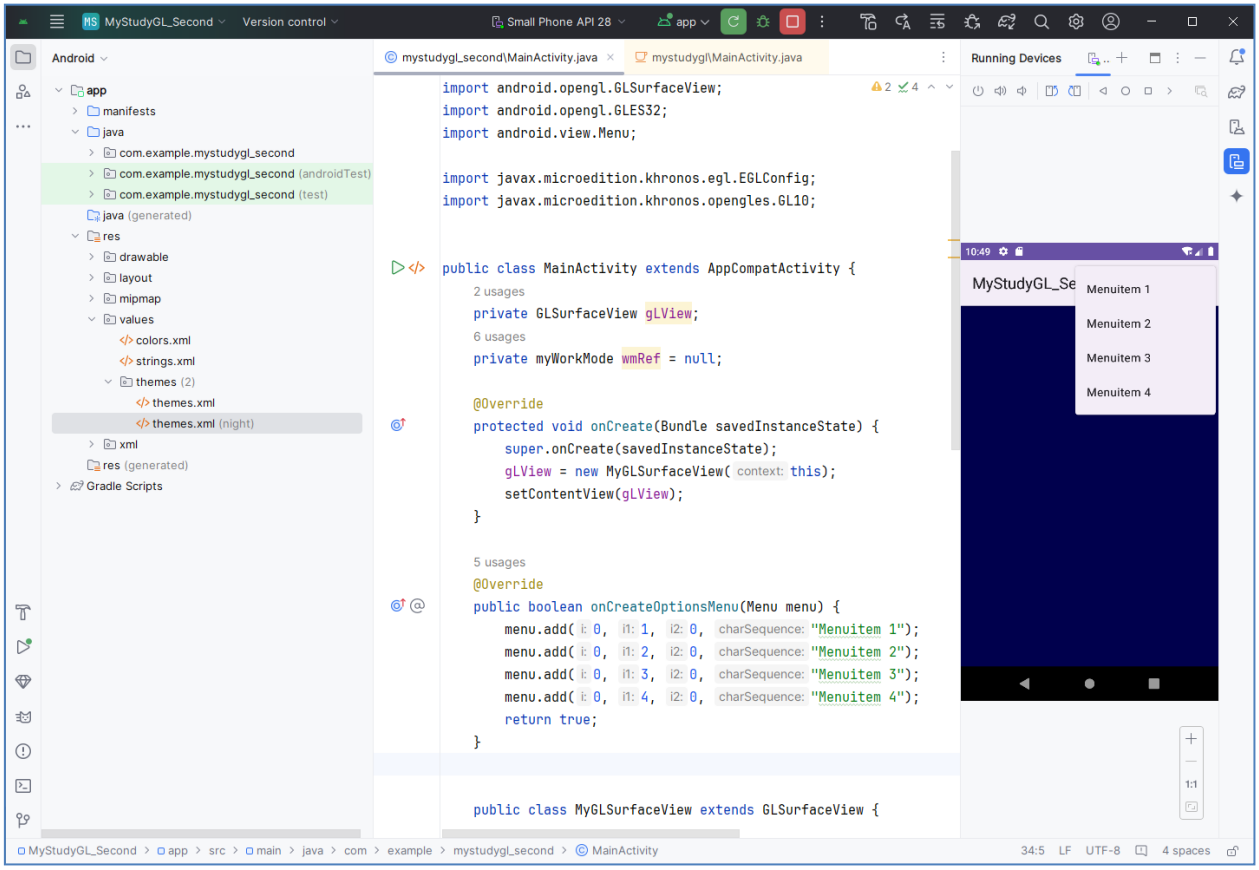


Рис. 2. Перевірка наявності меню застосунку

Звісно, у якості назви кожного з пунктів меню треба вказувати не “Menuitem”, а якусь назву події, яка має відбуватися після вибору відповідного пункту меню.

Третій крок програмування меню. Необхідно запрограмувати обробники для кожного пункту меню. Для цього потрібно визначити тіло метода onOptionsItemSelected( ).

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    setTitle(item.getTitle());
    switch (item.getItemId()) {
        case 1:
            // ... any code for Menuitem1 selection event
            return true;
        // ... responding to selection of other menu items
        default:break;
    }
    return super.onOptionsItemSelected(item);
}
```

Програмний код обробки вибору пунктів меню рекомендується робити структурованим. У цій лабораторній роботі кожний пункт меню буде позначати якійсь приклад зображення певних обєктів. Можна сказати, що за допомогою меню користувач буде робити вибор режиму работи застосунку.

Для кожного режиму роботи застосунку рекомендується створити окремий клас, який буде похідним від нашого власного базового класу myWorkMode. Так само, як вже було запропоновано у попередній лаб. роботі №1, цей клас буде інкапсулювати базові операції підготовки сцени, компіляцію шейдерних програм, завантаження масивів вершин та функції рендерінгу. Класи, які будуть похідними від myWorkMode, будуть успадковувати деякі операції, перевизначати існуючі і додавати власні.

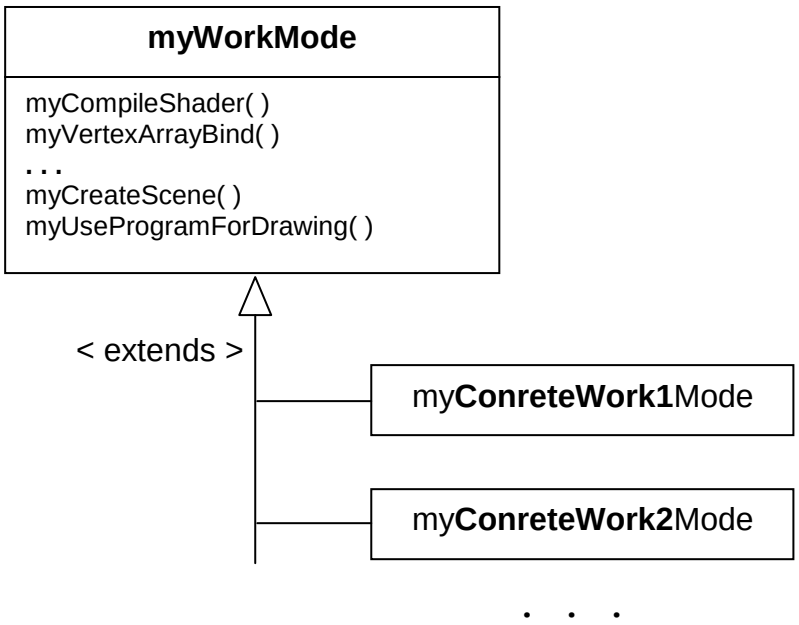


Рис. 3. Основи архітектури багаторежимного застосунку

Шаблон програмного коду

Шаблон програмного коду для лаб. №2 назвемо Шаблоном 3. Він базується на Шаблоні 2 для OpenGL ES, рекомендованому у попередній лаб. роботі №1. Нижче надано програмний код Шаблону 3 на Java.

MainActivity.java

```
import android.content.Context;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

import android.opengl.GLSurfaceView;
```

```
import android.opengl.GLES32;
import android.view.Menu;
import android.view.MenuItem;

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


public class MainActivity extends AppCompatActivity {
    private GLSurfaceView gLView;
    private myWorkMode wmRef = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        gLView = new MyGLSurfaceView(this);
        setContentView(gLView);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(0, 1, 0, "First example");
        menu.add(0, 2, 0, "Color animation");
        //... add other menuitems
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        setTitle(item.getTitle());
        switch (item.getItemId()) {
            case 1:
                myModeStart(new myExampleFirst(),
                            GLSurfaceView.RENDERMODE_WHEN_DIRTY);
                return true;
            case 2:
                myModeStart(new myExampleColorAnimation(),
                            GLSurfaceView.RENDERMODE_CONTINUOUSLY);
                return true;
            // ... responding to selection of other menu items
            default:break;
        }
        return super.onOptionsItemSelected(item);
    }

    public void myModeStart(myWorkMode wmode, int rendermode) {
        wmRef = wmode;
        gLView.setRenderMode(rendermode);
        gLView.requestRender();
    }

    public class MyGLSurfaceView extends GLSurfaceView {

        // ... усе так само, як у шаблоні 2 попередньої лаб. №1

    }

    public class MyGLRenderer implements GLSurfaceView.Renderer {

        // ... усе так само, як у шаблоні 2 попередньої лаб. №1

    }
}
```

Складовою шаблону 3 також є клас myWorkMode –той самий, який описано та використано у попередній лаб. роботі №1. Вихідний текст цього класу міститься у файлі myWorkMode.java.

Перевіримо коректність Шаблону 3 – створимо спочатку порожні класи myExampleFirst та myExampleColorAnimation – похідні від класу myWorkMode.java, а потім перевіримо роботу застосунку в емуляторі.

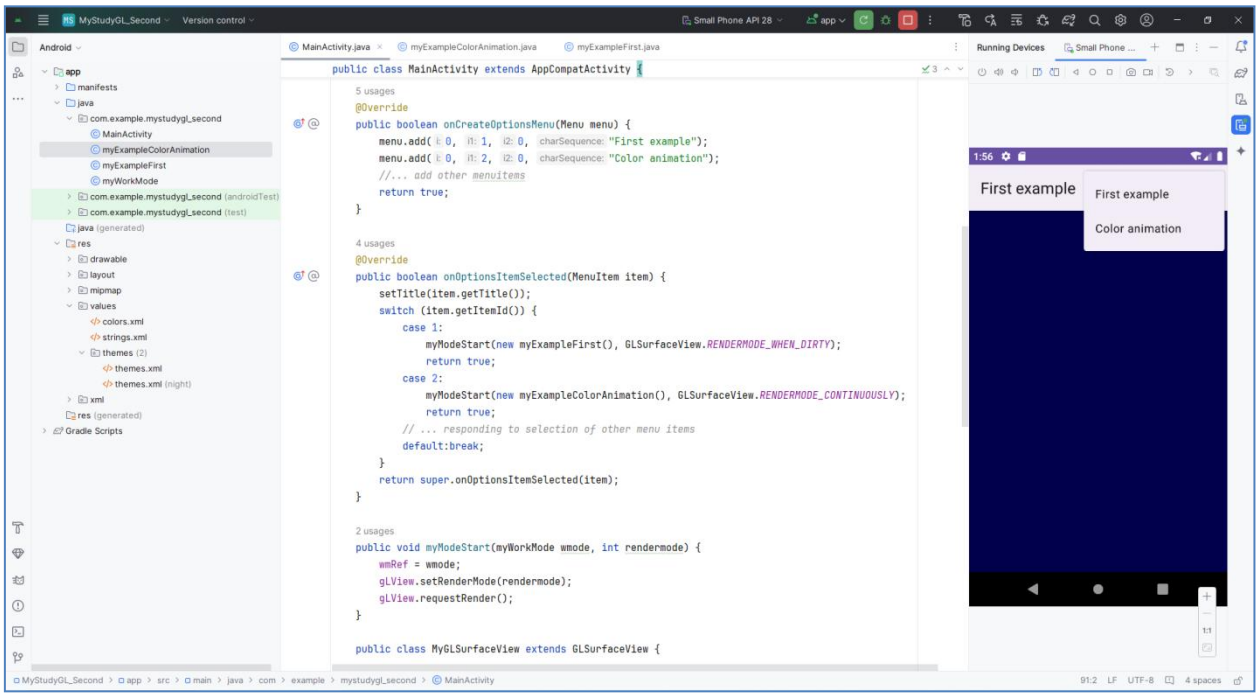


Рис. 4. Перевірка коректності Шаблону 3

Подальші зусилля по створенню застосунку будуть зосереджені на програмуванні класів, які тут названі як myExampleFirst, myExampleColorAnimation та інших класів.

Визначення кольору об'єкта у ході виконання програми

Одною з особливостей попередньої лаб. роботи №1 є те, що усі об'єкти малювання мають один колір. Це визначено у програмному коі фрагментного шейдера fragmentShaderCode1

```
#version 300 es
precision mediump float;
out vec4 outColor;
void main() {
   outColor = vec4(1.0f, 0.5f, 0.0f, 1.0f);
}
```

Цей шейдер через вихідну перемінну outColor виводить у графічний конвейєр константне значення помаранчевого кольору – RGBA(1, 0.5, 1, 1). А як запрограмувати можливість змінювати кольори об'єктів в ході виконання програми?

Для цього введемо у фрагментний шейдер uniform-перемінну, значення якій можна передавати зовні у будь-який момент роботи застосунку. Втілимо це у наступному зразку фрагментного шейдера, який назвемо fragmentShaderCode2

```
precision mediump float;
uniform vec4 vColor;
out vec4 outColor;
void main() {
   outColor = vColor;
}
```

Запрограмуємо для першого прикладу роботи з кольорами відповідний клас у файлі **myExampleFirst.java**

```
import android.opengl.GLES32;

public class myExampleFirst extends myWorkMode {

    protected float UserColor[] = { 0.7f, 0.5f, 0.9f, 1.0f };  // Magenta

    myExampleFirst() {
        super();
        myCreateScene();
    }

    @Override
    protected void myCreateScene() {
        arrayVertex = new float[] {     //triangle vertex coords
                0.5f, -0.5f, 0.0f,      // Bottom Right
                -0.5f, -0.5f, 0.0f,     // Bottom Left
                0.0f,  0.5f, 0.0f};     // Top
        numVertex = 3;
    }

    @Override
    public void myCreateShaderProgram() {
        myCompileAndAttachShaders(
                myShadersLibrary.vertexShaderCode1,
                myShadersLibrary.fragmentShaderCode2);
        myVertexArrayBind(arrayVertex,"vPosition");
    }

    @Override
    public void myUseProgramForDrawing(int width, int height) {
        // Set color for drawing object(s)
        // get handle to fragment shader's vColor member
        int colorHandle = GLES32.glGetUniformLocation(gl_Program, "vColor");
        GLES32.glUniform4fv(colorHandle, 1, UserColor, 0);

        GLES32.glBindVertexArray(VAO_id);
        GLES32.glDrawArrays(GLES32.GL_TRIANGLES, 0, numVertex);
        GLES32.glBindVertexArray(0);
    }
}
```

Програмні коди шейдерів для цього проєкту зосередимо у окремому класі – файлі **myShadersLibrary.java**

```
public class myShadersLibrary {
    //--- Vertex Shaders
    public static final String  vertexShaderCode1 =
            "#version 300 es\n" +
            "in vec3 vPosition;\n" +
            "void main() {\n" +
            "    gl_Position = vec4(vPosition, 1.0f);\n" +
            "}\n";

    //--- Fragment Shaders
    public static final String fragmentShaderCode2 =
            "#version 300 es\n" +
            "precision mediump float;\n" +
            "uniform vec4 vColor;\n" +
            "out vec4 outColor;\n" +
            "void main() {\n" +
            "    outColor = vColor;\n" +
            "}\n";
}
```

Питання для самоперевірки: а як можна ще запрограмувати на Java вміст таких String-обєктів, окрім конкатенації множини рядків “ ..\n“ +. . . + “ ..\n“ ?

Перевіримо роботу застосунку в емуляторі. Після запуску застосунку обираємо пункт меню “First example” і маємо побачити рожевий трикутник.

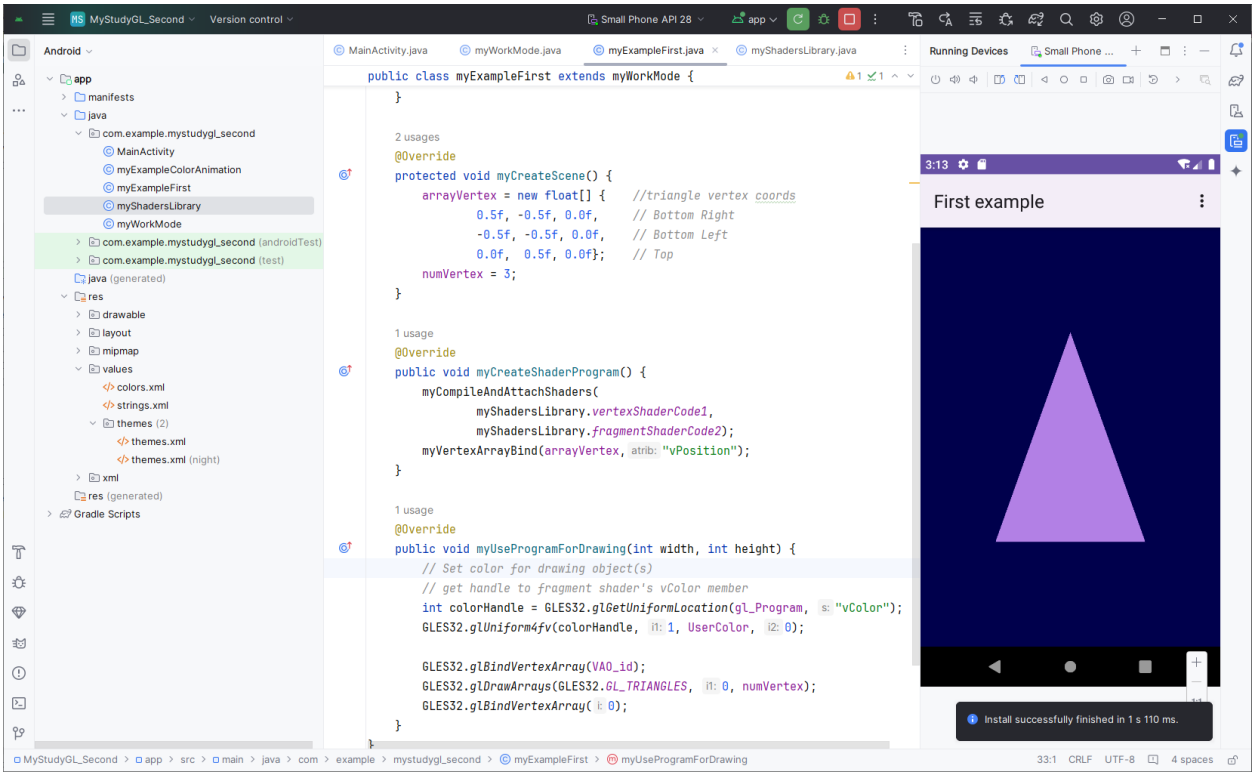


Рис. 5. Робота застосунку в емуляторі Virtual Android Device

Зміни кольору об'єкта у режимі анімації

Поставимо таке завдання. Як запрограмувати зображення об'єкту, щоб його колір постійно плавно змінювався впродовж показу?

Це буде нашою першою спробою анімації. Анімація – це постійний вивід на екран кадрів відображення сцени з певною циклічністю, причому кожний кадр може показувати зображення сцени, яке якось може змінюватися у часі. В ході анімації може змінюватися усе: ракурс показу об'єктів, розташування об’єктів тощо. У нашому випадку впродовж циклу анімації буде змінюватися колір обєкта–трикутника.

У запропонованому вище шаблоні програмного коду застосунку OpenGL ES режим рендерінгу сцени налаштовується в конструкторі класу MyGLSurfaceView

```
public MyGLSurfaceView(Context context){
    super(context);
    setEGLContextClientVersion(2);
    setRenderer(new MyGLRenderer());
    setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);  //default
}
```

В OpenGL ES рендерінг сцени може виконуватися, як мінімум, у двох режимах. Потрібний режим вказується значенням параметру метода setRenderMode( ).

– у режимі RENDERMODE_WHEN_DIRTY рендеринг виконується лише один раз – у відповідь на виклик метода requestRender( ). Цей режим у нас встановлюється по замовчуванню
– у режимі RENDERMODE_CONTINUOUSLY рендеринг автоматично безперервно повторюється з певним періодом часу у декілька мілісекунд.

Рендеринг – малювання вмісту екрану (або його фрагменту) в обох цих режимах відбувається в ході виклику метода onDrawFrame( ).

Таким чином, як наведено у тексті програмного коду Шаблону 3, при виборі відповідного пункту меню окрім створення об'єкту класу myExampleColorAnimation, ще встановлюється режим безперервного повторення рендерингу

```
    case 2:          //вибір пункту меню "Color animation"
        myModeStart(new myExampleColorAnimation(),
                    LSurfaceView.RENDERMODE_CONTINUOUSLY);
        return true;
```

А далі потрібно забезпечити змінюваність кольору об'єкта. Виконаємо це у методі myUseProgramForDrawing() класу myExampleColorAnimation. Вказаний метод, у свою чергу, викликається у тілі метода onDrawFrame( ).

Оскільки у класі myExampleColorAnimation створюється той самий трикутник і використовуються ті самі шейдери, що і у першому прикладі, то доцільно зробити клас myExampleColorAnimation спадкоємцем класу myExampleFirst.

Файл **myExampleColorAnimation.java**

```
import android.opengl.GLES32;
import android.os.SystemClock;

public class myExampleColorAnimation extends myExampleFirst {
    myExampleColorAnimation() {
        super();
    }

    @Override
    public void myUseProgramForDrawing(int width, int height) {
        //using time for color varying
        long time = SystemClock.uptimeMillis() % 2000L;
        long clrvar;
        if (time <= 1000)
            clrvar = time;
        else clrvar = 1999 - time;
        UserColor[0] = 0.001f*(float)clrvar;
        int colorHandle = GLES32.glGetUniformLocation(gl_Program, "vColor");
        GLES32.glUniform4fv(colorHandle, 1, UserColor, 0);

        GLES32.glBindVertexArray(VAO_id);
        GLES32.glDrawArrays(GLES32.GL_TRIANGLES, 0, numVertex);
        GLES32.glBindVertexArray(0);
    }
}
```

У циклі анімації у кожному виклику метода myUseProgramForDrawing() виконується вимір часу у мілісекундах від початкового кадру. Якщо залишок від ділення на 2000 від 0 до 1000 (перемінна time)', то це значення використовується при зростанні компоненти R, яка зберігається у першому елементі масиву UserColor[ ]. Значення time від 1001 до 1999 використовуються вже для зворотного зменшення R. Оскільки кожна компонента (R, G, B) має бути у діапазоні від 0 до 1, то введено масштабний коефіцієнт 0.001. Таким чином, цикл зміни кольору повторюється кожні 2 секунди.

У даному прикладі у кожному кадрі анімації у uniform-перемінну vColor фрагментного шейдеру перезаписується оновлений вміст масиву UserColor[ ].

Для перевірки роботи застосунку виконайте Run на емуляторі віртуального пристроя. Виберіть меню “Color animation” і спостерігайте, як колір трикутника плавно змінюється від рожевого до блакитного і навпаки.

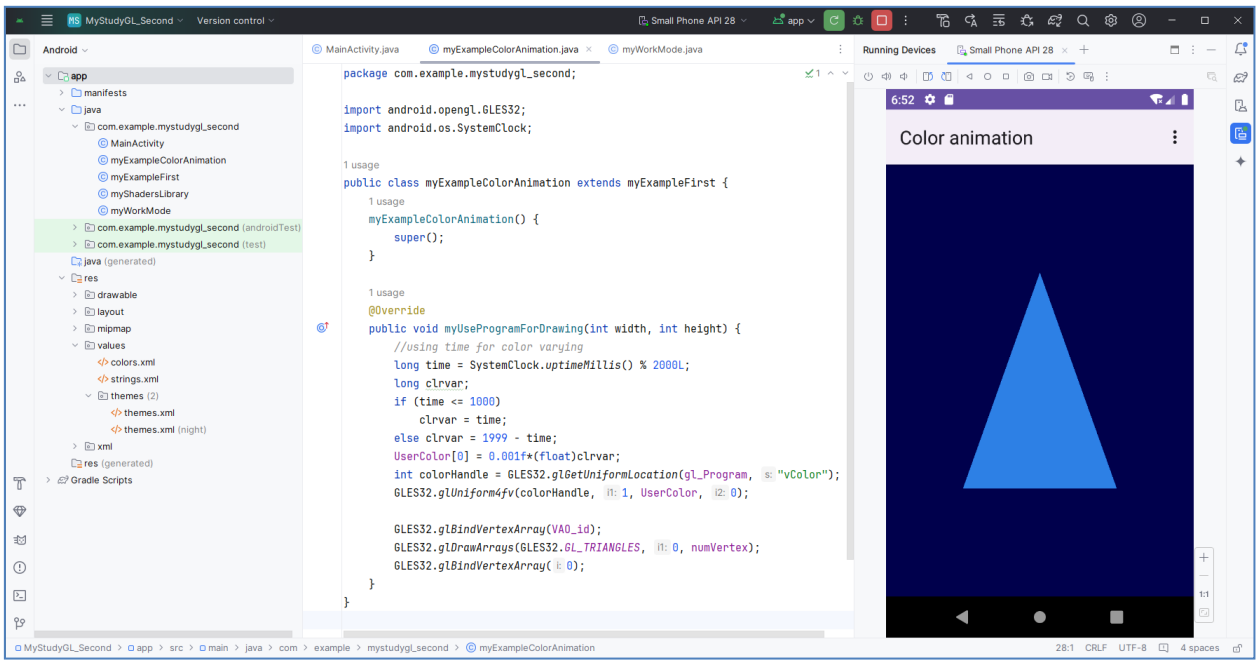


Рис. 6. Перевірка роботи застосунку в емуляторі

Для кожної вершини індівідуальний колір

У наведених прикладах визначається однаковий колір для усіх об'єктів, скільки б їх би не було (хоча поки що маємо лише один об'єкт-трикутник). Крім того, цей колір є однаковим для усіх точок трикутника.

Можливо для кожної вершини приписати власні кольори. Для того, щоб кожній вершині співставити індивідуальний колір, потрібно відповідним чином визначити масив вершин. Задля досягнення максимальної швидкодії бажано об’єднати координати та кольори в одному масиві і завантажити цей масив у пам'ять відеокарти і доручити графічному процесору знаходити колір для кожної вершини у конвеєрі рендерингу.

Основна ідея – по-можливости, намагатися звільняти центральний процесор від роботи, яку варто покласти на графічний процесор, спеціально оптимізований для обробки великих масивів даних з високою швидкістю.

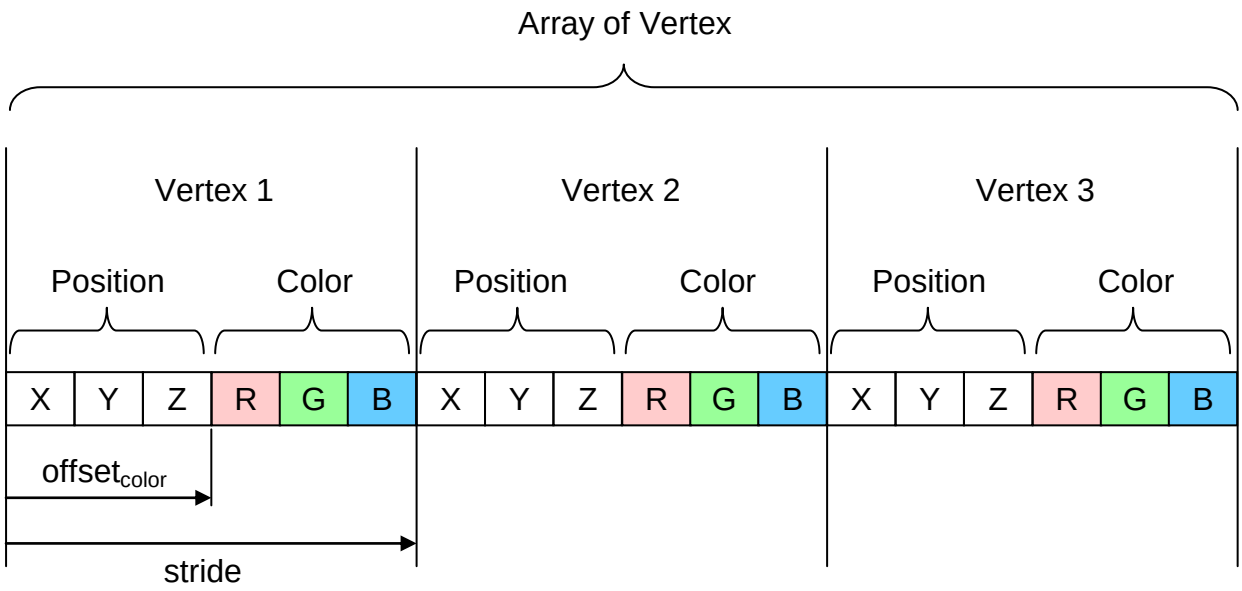


Рис. 7. Поєднання координат та кольорів у єдиному масиві вершин

Позначення:

- stride – визначає зміщення байтів між однаковими атрибутами послідовних вершин (період). У даному прикладі stride = 6 x 4 = 24 байтів, оскільки у масиві усі елементи X, Y, Z, R, G, B – чотирьохбайтові float
- $offset_{color}$ – зміщення байтів атрибутів кольору відносно початку опису атрибутів вершини. У даному прикладі $offset_{color}$ = 3 x 4 = 12 байтів

Для того, щоб реалізувати таке, потрібно при завантаженні масиву вершин вказати параметри розташування значень атрибутів вершин. Такі параметри вказуються при виклику метода glVertexAttribPointer(). Для даного прикладу буде

```
glVertexAttribPointer(handle, 3, GLES32.GL_FLOAT, false, stride, 0);
glVertexAttribPointer(handle, 3, GLES32.GL_FLOAT, false, stride, offsetColor);
```

Задля цього вдосконалимо шаблон програмного коду – у базовий клас myWorkMode додамо метод myVertexArrayBind2(), який повинен завантажувати масив вершин з двома атрибутами – позиція та колір.

Вдосконалений шаблон програмного коду

Назвемо його Шаблон 4. Програмний код міститься у класах MainActivity та myWorkMode. Оскільки програмний код MainActivity.java той самий, що і для попереднього шаблону 3, то тут наводимо лише текст класу **myWorkMode**. Але оскільки цей клас і дотепер фактично без змін використовувався ще від попередньої лаб. №1 (Шаблон 2), то оновлений програмний код нижче наводиться спрощено.

myWorkMode.java

```
import android.opengl.GLES32;
... //усе те ж саме, що у Шаблоні 2 для лаб1

//The base class for various concrete OpenGL ES examples
public class myWorkMode {
    ... //усе те ж саме, що у Шаблоні 2 для лаб1

    myWorkMode() {
    ... //усе те ж саме, що у Шаблоні 2 для лаб1
    }

    public int getProgramId() {
        ... //усе те ж саме, що у Шаблоні 2 для лаб1
    }

    protected int myCompileShader(int shadertype, String shadercode) {
        ... //усе те ж саме, що у Шаблоні 2 для лаб1
    }

    protected void myCompileAndAttachShaders(String vsh, String fsh) {
        ... //усе те ж саме, що у Шаблоні 2 для лаб1
    }

    protected void getId_VAO_VBO() {
        ... //усе те ж саме, що у Шаблоні 2 для лаб1
    }

    protected void myVertexArrayBind(float[] src, String atrib) {
        ... //усе те ж саме, що у Шаблоні 2 для лаб1
    }

    protected void myVertexArrayBind2(float[] src, int stride,
                                      String atrib1, int offset1,
                                      String atrib2, int offset2) {
        if (gl_Program <= 0) return;
        ByteBuffer bb = ByteBuffer.allocateDirect(src.length*4);
        bb.order(ByteOrder.nativeOrder());

        FloatBuffer vertexBuffer = bb.asFloatBuffer();
        vertexBuffer.put(src);
        vertexBuffer.position(0);

        getId_VAO_VBO();
        // Bind the Vertex Array Object first
        GLES32.glBindVertexArray(VAO_id);

        //then bind and set vertex buffer
        GLES32.glBindBuffer(GLES32.GL_ARRAY_BUFFER, VBO_id);
        GLES32.glBufferData(GLES32.GL_ARRAY_BUFFER,src.length*4,
                            vertexBuffer, GLES32.GL_STATIC_DRAW);
```

```
        //then define attribute pointers
        int handle = GLES32.glGetAttribLocation(gl_Program, atrib1);
        GLES32.glEnableVertexAttribArray(handle);
        GLES32.glVertexAttribPointer(handle, 3, GLES32.GL_FLOAT,
                                     false, stride*4, offset1);

        handle = GLES32.glGetAttribLocation(gl_Program, atrib2);
        GLES32.glEnableVertexAttribArray(handle);
        GLES32.glVertexAttribPointer(handle, 3, GLES32.GL_FLOAT,
                                     false, stride*4, offset2);

        GLES32.glEnableVertexAttribArray(0);
        GLES32.glBindBuffer(GLES32.GL_ARRAY_BUFFER, 0);
        GLES32.glBindVertexArray(0);
    }

    public void myCreateShaderProgram() { }

    protected void myCreateScene() { }

    public void myUseProgramForDrawing(int width, int height) {
        //Default implementation for simply triangulated scenes
        GLES32.glBindVertexArray(VAO_id);
        GLES32.glDrawArrays(GLES32.GL_TRIANGLES, 0,numVertex);
        GLES32.glBindVertexArray(0);
    }
}
```

А тепер запрограмуємо рішення для прикладу відображення трикутника з вершинами різного кольору. Спочатку шейдер вершин. У нашій бібліотеці шейдерів це буде vertexShaderCode2

```
#version 300 es
in vec3 vPosition;
in vec3 vColor;
out vec3 outColor;
void main() {
   gl_Position = vec4(vPosition, 1.0f);
   outColor = vColor;
}
```

У цьому шейдері маємо дві вхідні перемінні – vPosition та vColor, які сприймають відповідні атрибути вершин – елементи завантаженого масиву вершин. Значення вихідної перемінної outColor буде використано у фрагментному шейдері. Його вихідний текст назвемо fragmentShaderCode3

```
#version 300 es
precision mediump float;
in vec3 outColor;
out vec4 resultColor;
void main() {
    resultColor = vec4(outColor, 1.0f);
}
```

Далі запрограмуємо клас myOwnColorForEachVertexMode

```
public class myOwnColorForEachVertexMode extends myWorkMode  {

    myOwnColorForEachVertexMode() {
        super();
        myCreateScene();
    }

    @Override
    protected void myCreateScene() {
        //coords and color for each vertex individually
        arrayVertex = new float[] {
                //      coords                  color
                0.5f, -0.5f, 0.0f,     0.0f, 0.8f, 1.0f,
                -0.5f, -0.5f, 0.0f,    0.3f, 1.0f, 0.0f,
                0.0f,  0.5f, 0.0f,     1.0f, 0.0f, 0.0f};
        numVertex = 3;
    }

    public void myCreateShaderProgram() {
        myCompileAndAttachShaders(
                myShadersLibrary.vertexShaderCode2,
                myShadersLibrary.fragmentShaderCode3);
        myVertexArrayBind2(arrayVertex, 6,
                "vPosition", 0,
                "vColor", 3*4);
    }
}
```

І насамкінець для MainActivity запрограмуємо інтерфейс користувача – у метод onCreateOptionsMenu() додамо пункт меню “Own color for each vertex”. Також додамо відповідний обробник події вибору цього пункту меню у методі onOptionsItemSelected().

MainActivity.java (наведені лише фрагменти коду, де є зміни)

```
. . .
public class MainActivity extends AppCompatActivity {
. . .

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(0, 1, 0, "First example");
        menu.add(0, 2, 0, "Color animation");
        menu.add(0, 3, 0, "Own color for each vertex");
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        setTitle(item.getTitle());
        switch (item.getItemId()) {
            case 1:
                myModeStart(new myExampleFirst(),
                            GLSurfaceView.RENDERMODE_WHEN_DIRTY);
                return true;
            case 2:
                myModeStart(new myExampleColorAnimation(),
                            GLSurfaceView.RENDERMODE_CONTINUOUSLY);
                return true;
            case 3:
                myModeStart(new myOwnColorForEachVertexMode(),
                            GLSurfaceView.RENDERMODE_WHEN_DIRTY);
                return true;
            default:break;
        }
        return super.onOptionsItemSelected(item);
    }
. . .

}
```

Перевіримо роботу застосунку. Викличемо програму на виконання у емуляторі. Оберемо пункт меню “Own color for each vertex” – маємо побачити зображення різноколірного трикутника, як наведено на скриншоті нижче.

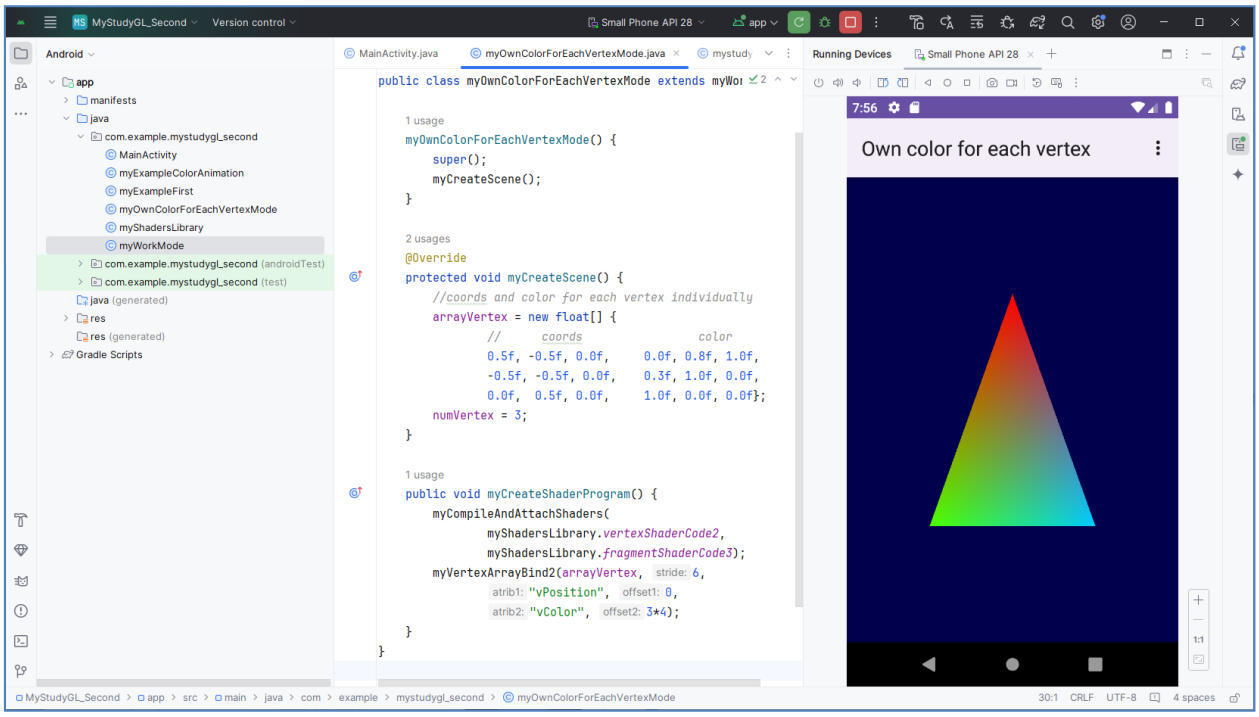


Рис. 8. Інтерполяція кольорів вершин у площині трикутника

Моделювання затемнення-підсвітлення у режимі анімації

А як запрограмувати затемнення-підсвілення такого різноколірного трикутника у режимі анімації? Під затемненням-підсвітленням розуміється зміна рівня випромінювання або відбиття світла від джерела світла змінної інтенсивності. Уявимо собі, що деяка грань має колір *Color = (R, G, B)*. Тоді, якщо помножити усі компоненти на деякий коєфіцієнт *Light*, то новий колір

$$NewColor = (Light \times R, Light \times G, Light \times B)$$

при $Light < 1$ буде означати затемнення, а при $Light > 1$ – підсвітлення.

Напишемо фрагментний шейдер

```
#version 300 es
precision mediump float;
uniform float vLight;
in vec3 outColor;
out vec4 resultColor;
void main() {
    resultColor = vec4(vLight*outColor, 1.0f);
}
```

Uniform-перемінна vLight буде глобальною перемінною, значення якої можна змінювати в ході роботи застосунку, зокрема, у режимі анімації.

Нижче наведено програмний код метода, який виконує рендеринг об'єктів із періодичним плавним затемненням

```
public void myUseProgramForDrawing(int width, int height) {
    long time = SystemClock.uptimeMillis() % 2000L;
    float vlight;
    if (time <= 1000)
        vlight = time;
    else vlight = 1999 - time;
    int colorHandle = GLES32.glGetUniformLocation(gl_Program, "vLight");
    GLES32.glUniform1f(colorHandle, 0.001f*(float)vlight);

    GLES32.glBindVertexArray(VAO_id);
    GLES32.glDrawArrays(GLES32.GL_TRIANGLES, 0, numVertex);
    GLES32.glBindVertexArray(0);
}
```

Тут формується значення для uniform-перемінної “vLight” фрагментного шейдера у діапазоні (0 . . . 1) з періодичністю 2 секунди.

Колірне коло Ньютона та його імітація

Видатний вчений Ісаак Ньютон запропонував модель опису кольору на основі 7 основних кольорів: червоного, помаранчевого, жовтого, зеленого, блакитного, синього, фіолетового, які розташував по колу. Така ідея використовується у деяких сучасних моделях опису кольорів, таких як HSV (HSB), HSL тощо.

У цій лабораторній роботі пропонується запрограмувати різноколірний семикутник. У центрі білий колір, по периметру – 7 кольорів для вершин:

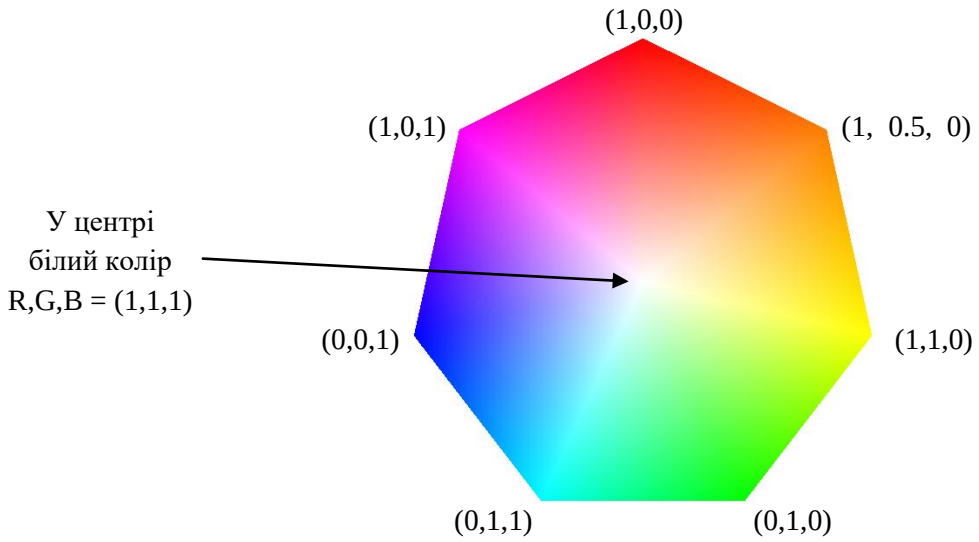


Рис. 9. Визначення основних кольорів у вершинах семикутника

Семикутник представимо у вигляді 7 трикутників зі спільною вершиною [0] у центрі – усього 9 вершин

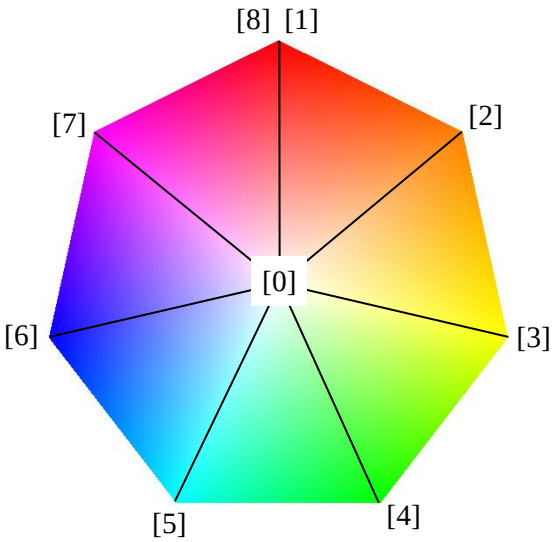


Рис. 10. Індекси вершин

Рекомендується відображати такий семикутник викликом функції

```
GLES32.glDrawArrays(GLES32.GL_TRIANGLE_FAN, startVertexFan, 9);
```

Стрічка кольорів веселки

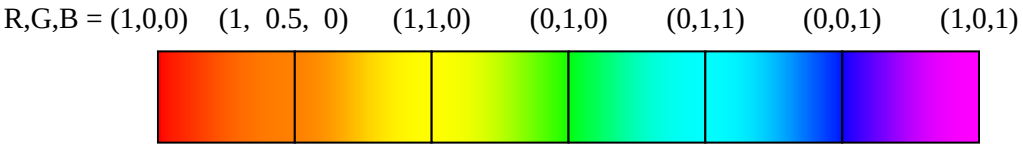


Рис. 11. Стрічка–веселка

Рекомендується відображати таку стрічку сіткою звязаних трикутників:

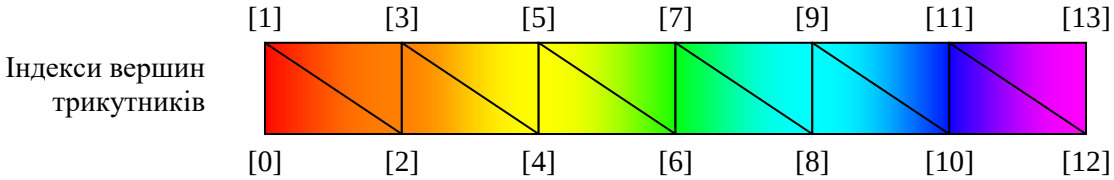
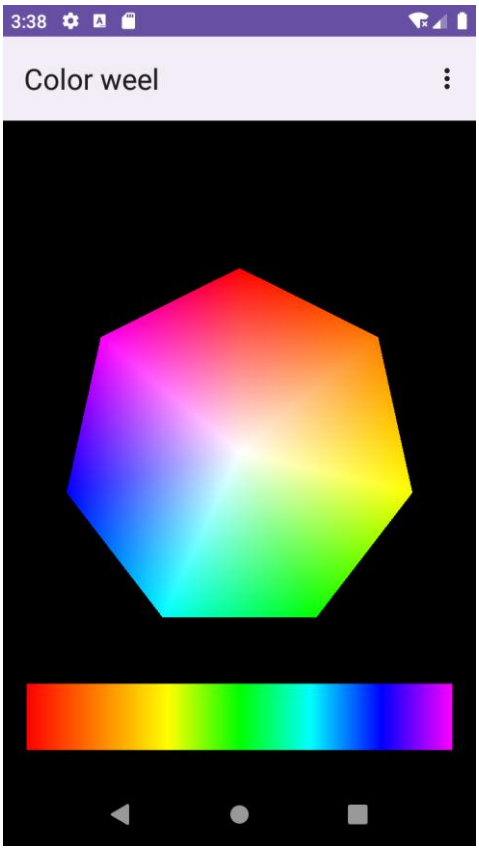


Рис. 12. Індекси вершин сітки трикутників

Для рендерінгу такої сітки трикутників зручно використати

```
GLES32.glDrawArrays(GLES32.GL_TRIANGLE_STRIP, startVertexStrip, 14);
```

Варіанти завдань та основні вимоги



1. Застосунок **Lab2_GLES** для вибору режиму роботи повинен мати меню з двома пунктами:
- Color Weel
- Color Weel animation

2. У режимі **Color Weel** потрібно відображати статичну картинку – на чорному фоні семикутник та стрічку веселки знизу.

3. У режимі **Color Weel animation** також відображаються ці два об'єкти, але один з них у режимі анімації плавно затемнюється до суцільного чорного і потім плавно відновлює первісну яскравість. Так повторюється до кінця роботи застосунку, або доти, поки не буде обрано інший режим роботи застосунку. Який з об’єктів має анімовану змінну яскравість – це визначається варіантом завдання.

4. Вибір варіанту завдання:

- для парних номерів залікових книжок варіант 1: анімоване змінне затемнення лише у семикутника, а у стрічки веселки зображення незмінне;

- для непарних номерів варіант 2: має бути змінне затемнення у нижньої стрічки веселки, а зображення семикутника незмінне.

Зміст звіту

1. Титульний аркуш
2. Варіант завдання
3. Вихідний тексти
4. Ілюстрації (скріншоти)
5. Висновки

Контрольні запитання

1. Що таке uniform-змінні у шейдерів?
2. Як передати одне значення типу float uniform-змінній шейдера?
3. Як передати вектор з трьох компонент кольору uniform-змінній шейдера?
4. Як запрограмувати перехід у інший режим роботи програми?
5. Чим відрізняються GL_TRIANGLE_FAN та GL_TRIANGLE_STRIP?
6. Як запрограмувати затемнення об’єктів?
7. Що таке stride у масиві вершин?