Kpi-best

Міністерство освіти та науки України

Національний технічний університет України “Київський політехнічний інститут”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота № 6

з курсу "Графічне та геометричне моделювання"

Виконав: студент 4 курсу

групи ІП-32

Ковтун А.В.

Київ – 2016

**Завдання:**

Виконати одночасно обернення навколо двох вісей заданої каркасної моделі геометричної фігури використовуючи матриці тривимірного перетворення. Одна з вершин обов’язково повинна мати координати (0, 0, 0) та бути розташована по центру екрану.

**Код програми**

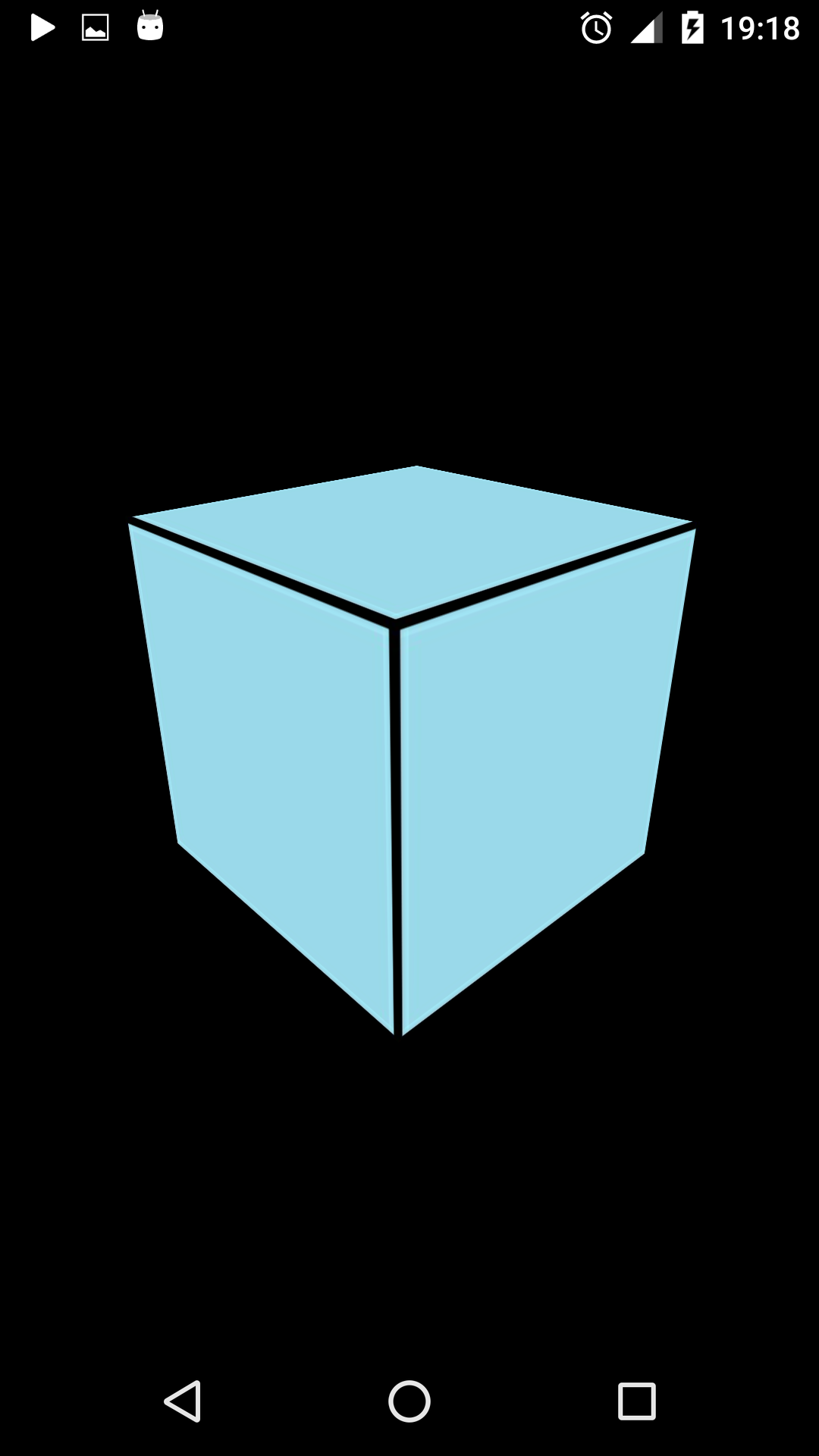
**package** ru.startandroid.l176texturecube;  
  
**import** android.content.Context;  
**import** android.graphics.Bitmap;  
**import** android.graphics.BitmapFactory;  
**import** android.opengl.GLES20;  
**import** android.opengl.GLES30;  
**import** android.opengl.GLUtils;  
  
**import static** android.opengl.GLES20.***GL\_TEXTURE0***;  
**import static** android.opengl.GLES20.***GL\_TEXTURE\_2D***;  
**import static** android.opengl.GLES20.***GL\_TEXTURE\_CUBE\_MAP***;  
**import static** android.opengl.GLES20.*glActiveTexture*;  
**import static** android.opengl.GLES20.*glBindTexture*;  
**import static** android.opengl.GLES20.glDeleteTextures;  
**import static** android.opengl.GLES20.glGenTextures;  
  
**public class** TextureUtils {  
  
 **public static int** loadTextureCube(Context context, **int** resourceId) {  
 *// создание объекта текстуры* **final int**[] textureIds = **new int**[1];  
 *glGenTextures*(1, textureIds, 0);  
 **if** (textureIds[0] == 0) {  
 **return** 0;  
 }  
 *// получение Bitmap* **final** BitmapFactory.Options options = **new** BitmapFactory.Options();  
 options.**inScaled** = **false**;  
  
 Bitmap[] bitmaps = **new** Bitmap[6];  
 **for** (**int** i = 0; i < 6; i++) {  
 bitmaps[i] = BitmapFactory.*decodeResource*(  
 context.getResources(), resourceId, options);  
  
 **if** (bitmaps[i] == **null**) {  
 *glDeleteTextures*(1, textureIds, 0);  
 **return** 0;  
 }  
 }  
 *// настройка объекта текстуры  
 glActiveTexture*(***GL\_TEXTURE0***);  
 *glBindTexture*(***GL\_TEXTURE\_CUBE\_MAP***, textureIds[0]);  
  
 GLES20.*glTexParameteri*(GLES20.***GL\_TEXTURE\_CUBE\_MAP***, GLES20.***GL\_TEXTURE\_MIN\_FILTER***, GLES20.***GL\_LINEAR***);  
 GLES20.*glTexParameteri*(GLES20.***GL\_TEXTURE\_CUBE\_MAP***, GLES20.***GL\_TEXTURE\_MAG\_FILTER***, GLES20.***GL\_LINEAR***);  
  
 GLUtils.*texImage2D*(GLES20.***GL\_TEXTURE\_CUBE\_MAP\_NEGATIVE\_X***, 0, bitmaps[0], 0);  
 GLUtils.*texImage2D*(GLES20.***GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X***, 0, bitmaps[1], 0);  
  
 GLUtils.*texImage2D*(GLES20.***GL\_TEXTURE\_CUBE\_MAP\_NEGATIVE\_Y***, 0, bitmaps[2], 0);  
 GLUtils.*texImage2D*(GLES20.***GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_Y***, 0, bitmaps[3], 0);  
  
 GLUtils.*texImage2D*(GLES20.***GL\_TEXTURE\_CUBE\_MAP\_NEGATIVE\_Z***, 0, bitmaps[4], 0);  
 GLUtils.*texImage2D*(GLES20.***GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_Z***, 0, bitmaps[5], 0);  
 **for** (Bitmap bitmap : bitmaps) {  
 bitmap.recycle();  
 }  
 *// сброс target  
 glBindTexture*(***GL\_TEXTURE\_CUBE\_MAP***, 0);  
 **return** textureIds[0];  
 }  
}

**package** ru.startandroid.l176texturecube;  
  
**import** android.content.Context;  
  
**import static** android.opengl.GLES20.***GL\_COMPILE\_STATUS***;  
**import static** android.opengl.GLES20.***GL\_LINK\_STATUS***;  
**import static** android.opengl.GLES20.*glAttachShader*;  
**import static** android.opengl.GLES20.*glCompileShader*;  
**import static** android.opengl.GLES20.*glCreateProgram*;  
**import static** android.opengl.GLES20.*glCreateShader*;  
**import static** android.opengl.GLES20.*glDeleteProgram*;  
**import static** android.opengl.GLES20.*glDeleteShader*;  
**import static** android.opengl.GLES20.glGetProgramiv;  
**import static** android.opengl.GLES20.glGetShaderiv;  
**import static** android.opengl.GLES20.*glLinkProgram*;  
**import static** android.opengl.GLES20.*glShaderSource*;  
  
**public class** ShaderUtils {  
  
 **public static int** createProgram(**int** vertexShaderId, **int** fragmentShaderId) {  
  
 **final int** programId = *glCreateProgram*();  
 **if** (programId == 0) {  
 **return** 0;  
 }  
 *glAttachShader*(programId, vertexShaderId);  
 *glAttachShader*(programId, fragmentShaderId);  
  
 *glLinkProgram*(programId);  
 **final int**[] linkStatus = **new int**[1];  
 *glGetProgramiv*(programId, ***GL\_LINK\_STATUS***, linkStatus, 0);  
 **if** (linkStatus[0] == 0) {  
 *glDeleteProgram*(programId);  
 **return** 0;  
 }  
 **return** programId;  
 }  
 **static int** createShader(Context context, **int** type, **int** shaderRawId) {  
 String shaderText = FileUtils  
 .*readTextFromRaw*(context, shaderRawId);  
 **return** ShaderUtils.*createShader*(type, shaderText);  
 }  
  
 **static int** createShader(**int** type, String shaderText) {  
 **final int** shaderId = *glCreateShader*(type);  
 **if** (shaderId == 0) {  
 **return** 0;  
 }  
 *glShaderSource*(shaderId, shaderText);  
 *glCompileShader*(shaderId);  
 **final int**[] compileStatus = **new int**[1];  
 *glGetShaderiv*(shaderId, ***GL\_COMPILE\_STATUS***, compileStatus, 0);  
 **if** (compileStatus[0] == 0) {  
 *glDeleteShader*(shaderId);  
 **return** 0;  
 }  
 **return** shaderId;  
 }  
}

**package** ru.startandroid.l176texturecube;  
**import** android.content.Context;  
**import** android.content.res.Resources;  
  
**import** java.io.BufferedReader;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.io.InputStreamReader;  
  
**public class** FileUtils {  
  
 **public static** String readTextFromRaw(Context context, **int** resourceId) {  
 StringBuilder stringBuilder = **new** StringBuilder();  
 **try** {  
 BufferedReader bufferedReader = **null**;  
 **try** {  
 InputStream inputStream =  
 context.getResources().openRawResource(resourceId);  
 bufferedReader = **new** BufferedReader(**new** InputStreamReader(inputStream));  
 String line;  
 **while** ((line = bufferedReader.readLine()) != **null**) {  
 stringBuilder.append(line);  
 stringBuilder.append(**"\r\n"**);  
 }  
 } **finally** {  
 **if** (bufferedReader != **null**) {  
 bufferedReader.close();  
 }  
 }  
 } **catch** (IOException ioex) {  
 ioex.printStackTrace();  
 } **catch** (Resources.NotFoundException nfex) {  
 nfex.printStackTrace();  
 }  
 **return** stringBuilder.toString();  
 }  
}

**package** ru.startandroid.l176texturecube;  
  
**import** android.content.Context;  
**import** android.opengl.GLSurfaceView.Renderer;  
**import** android.opengl.Matrix;  
**import** android.os.SystemClock;  
  
**import** java.nio.ByteBuffer;  
**import** java.nio.ByteOrder;  
**import** java.nio.FloatBuffer;  
  
**import** javax.microedition.khronos.egl.EGLConfig;  
**import** javax.microedition.khronos.opengles.GL10;  
  
**import static** android.opengl.GLES20.***GL\_COLOR\_BUFFER\_BIT***;  
**import static** android.opengl.GLES20.***GL\_DEPTH\_BUFFER\_BIT***;  
**import static** android.opengl.GLES20.***GL\_DEPTH\_TEST***;  
**import static** android.opengl.GLES20.***GL\_FLOAT***;  
**import static** android.opengl.GLES20.***GL\_FRAGMENT\_SHADER***;  
**import static** android.opengl.GLES20.***GL\_TEXTURE0***;  
**import static** android.opengl.GLES20.***GL\_TEXTURE\_2D***;  
**import static** android.opengl.GLES20.***GL\_TEXTURE\_CUBE\_MAP***;  
**import static** android.opengl.GLES20.***GL\_TRIANGLES***;  
**import static** android.opengl.GLES20.***GL\_TRIANGLE\_STRIP***;  
**import static** android.opengl.GLES20.***GL\_UNSIGNED\_BYTE***;  
**import static** android.opengl.GLES20.***GL\_VERTEX\_SHADER***;  
**import static** android.opengl.GLES20.*glActiveTexture*;  
**import static** android.opengl.GLES20.*glBindTexture*;  
**import static** android.opengl.GLES20.*glClear*;  
**import static** android.opengl.GLES20.*glClearColor*;  
**import static** android.opengl.GLES20.*glDrawArrays*;  
**import static** android.opengl.GLES20.glDrawElements;  
**import static** android.opengl.GLES20.*glEnable*;  
**import static** android.opengl.GLES20.*glEnableVertexAttribArray*;  
**import static** android.opengl.GLES20.*glGetAttribLocation*;  
**import static** android.opengl.GLES20.*glGetUniformLocation*;  
**import static** android.opengl.GLES20.*glUniform1i*;  
**import static** android.opengl.GLES20.glUniformMatrix4fv;  
**import static** android.opengl.GLES20.*glUseProgram*;  
**import static** android.opengl.GLES20.glVertexAttribPointer;  
**import static** android.opengl.GLES20.*glViewport*;  
  
**public class** OpenGLRenderer **implements** Renderer {  
  
 **private final static int *POSITION\_COUNT*** = 3;  
  
 **private** Context **context**;  
  
 **private** FloatBuffer **vertexData**;  
 **private** ByteBuffer **indexArray**;  
  
 **private int aPositionLocation**;  
 **private int uTextureUnitLocation**;  
 **private int uMatrixLocation**;  
  
 **private int programId**;  
  
 **private float**[] **mProjectionMatrix** = **new float**[16];  
 **private float**[] **mViewMatrix** = **new float**[16];  
 **private float**[] **mModelMatrix** = **new float**[16];  
 **private float**[] **mMatrix** = **new float**[16];  
  
 **private int texture**;  
  
 **public** OpenGLRenderer(Context context) {  
 **this**.**context** = context;  
 }  
 @Override  
 **public void** onSurfaceCreated(GL10 arg0, EGLConfig arg1) {  
 *glClearColor*(0f, 0f, 0f, 1f);  
 *glEnable*(***GL\_DEPTH\_TEST***);  
  
 createAndUseProgram();  
 getLocations();  
 prepareData();  
 bindData();  
 createViewMatrix();  
 Matrix.*setIdentityM*(**mModelMatrix**, 0);  
 }  
 @Override  
 **public void** onSurfaceChanged(GL10 arg0, **int** width, **int** height) {  
 *glViewport*(0, 0, width, height);  
 createProjectionMatrix(width, height);  
 bindMatrix();  
 }  
 **private void** prepareData() {  
  
 **float**[] vertices = {  
 *// вершины куба* -1, 1, 1, *// верхняя левая ближняя* 1, 1, 1, *// верхняя правая ближняя* -1, -1, 1, *// нижняя левая ближняя* 1, -1, 1, *// нижняя правая ближняя* -1, 1, -1, *// верхняя левая дальняя* 1, 1, -1, *// верхняя правая дальняя* -1, -1, -1, *// нижняя левая дальняя* 1, -1, -1 *// нижняя правая дальняя* };  
  
 **vertexData** = ByteBuffer  
 .*allocateDirect*(vertices.**length** \* 4)  
 .order(ByteOrder.*nativeOrder*())  
 .asFloatBuffer();  
 **vertexData**.put(vertices);  
  
 **indexArray** = ByteBuffer.*allocateDirect*(36)  
 .put(**new byte**[] {  
 *// грани куба  
 // ближняя* 1, 3, 0,  
 0, 3, 2,  
 *// дальняя* 4, 6, 5,  
 5, 6, 7,  
 *// левая* 0, 2, 4,  
 4, 2, 6,  
 *// правая* 5, 7, 1,  
 1, 7, 3,  
  
 *// верхняя* 5, 1, 4,  
 4, 1, 0,  
 *// нижняя* 6, 2, 7,  
 7, 2, 3  
 });  
 **indexArray**.position(0);  
  
 **texture** = TextureUtils.*loadTextureCube*(**context**, R.drawable.***rect***);  
 }  
 **private void** createAndUseProgram() {  
 **int** vertexShaderId = ShaderUtils.*createShader*(**context**, ***GL\_VERTEX\_SHADER***, R.raw.***vertex\_shader***);  
 **int** fragmentShaderId = ShaderUtils.*createShader*(**context**, ***GL\_FRAGMENT\_SHADER***, R.raw.***fragment\_shader***);  
 **programId** = ShaderUtils.*createProgram*(vertexShaderId, fragmentShaderId);  
 *glUseProgram*(**programId**);  
 }  
 **private void** getLocations() {  
 **aPositionLocation** = *glGetAttribLocation*(**programId**, **"a\_Position"**);  
 **uTextureUnitLocation** = *glGetUniformLocation*(**programId**, **"u\_TextureUnit"**);  
 **uMatrixLocation** = *glGetUniformLocation*(**programId**, **"u\_Matrix"**);  
 }  
 **private void** bindData() {  
 *// координаты вершин* **vertexData**.position(0);  
 *glVertexAttribPointer*(**aPositionLocation**, ***POSITION\_COUNT***, ***GL\_FLOAT***,  
 **false**, 0, **vertexData**);  
 *glEnableVertexAttribArray*(**aPositionLocation**);  
  
 *// помещаем текстуру в target CUBE\_MAP юнита 0  
 glActiveTexture*(***GL\_TEXTURE0***);  
 *glBindTexture*(***GL\_TEXTURE\_CUBE\_MAP***, **texture**);  
  
 *// юнит текстуры  
 glUniform1i*(**uTextureUnitLocation**, 0);  
  
 }  
  
 **private void** createProjectionMatrix(**int** width, **int** height) {  
 **float** ratio = 1;  
 **float** left = -1;  
 **float** right = 1;  
 **float** bottom = -1;  
 **float** top = 1;  
 **float** near = 2;  
 **float** far = 12;  
 **if** (width > height) {  
 ratio = (**float**) width / height;  
 left \*= ratio;  
 right \*= ratio;  
 } **else** {  
 ratio = (**float**) height / width;  
 bottom \*= ratio;  
 top \*= ratio;  
 }  
  
 Matrix.*frustumM*(**mProjectionMatrix**, 0, left, right, bottom, top, near, far);  
 }  
  
 **private void** createViewMatrix() {  
 *// точка положения камеры* **float** eyeX = 0;  
 **float** eyeY = 2;  
 **float** eyeZ = 4;  
  
 *// точка направления камеры* **float** centerX = 0;  
 **float** centerY = 0;  
 **float** centerZ = 0;  
  
 *// up-вектор* **float** upX = 0;  
 **float** upY = 1;  
 **float** upZ = 0;  
  
 Matrix.*setLookAtM*(**mViewMatrix**, 0, eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ);  
 }  
  
 **private void** bindMatrix() {  
  
 Matrix.*multiplyMM*(**mMatrix**, 0, **mViewMatrix**, 0, **mModelMatrix**, 0);  
 Matrix.*multiplyMM*(**mMatrix**, 0, **mProjectionMatrix**, 0, **mMatrix**, 0);  
 *glUniformMatrix4fv*(**uMatrixLocation**, 1, **false**, **mMatrix**, 0);  
 }  
 **long TIME** = 10000L;  
  
 @Override  
 **public void** onDrawFrame(GL10 arg0) {  
 *glClear*(***GL\_COLOR\_BUFFER\_BIT*** | ***GL\_DEPTH\_BUFFER\_BIT***);  
 Matrix.*setIdentityM*(**mModelMatrix**, 0);  
 *// вращение* setModelMatrix();  
 *glDrawElements*(***GL\_TRIANGLES***, 36, ***GL\_UNSIGNED\_BYTE***, **indexArray**);  
 }  
 **private void** setModelMatrix() {  
 **float** angle = (**float**)(SystemClock.*uptimeMillis*() % **TIME**) / **TIME** \* 360;  
 Matrix.*rotateM*(**mModelMatrix**, 0, angle, 0, 1, 0);  
 bindMatrix();  
 }  
  
}

**Результат виконання**



**Высновки:** В даній лабораторній роботі був побудований куб. Для побудови була використана матриця відображення із вказанням координат вершин куба. Були використана наступна матриця координатів:

-1, 1, 1, 1, 1, 1,-1, -1, 1, 1, -1, 1, -1, 1, -1, 1, 1, -1, -1, -1, -1, 1, -1, -1

Для створення прокрутки фігури був встановлений об’єкт камера який змінює свій вектор направлення тим самим прокручуючи фігури навколо осі Z. Час прокрутки 10сек.