Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Кафедра ПМиК

Лабораторная работа №5

по дисциплине

«Программирование мобильных устройств»

Выполнил:   
студент гр. ИП-813

Бурдуковский И.А.

Проверила:

Павлова У.В.

Новосибирск 2021

Оглавление

[Задание 3](#_Toc36477188)

[Выполнение 3](#_Toc36477189)

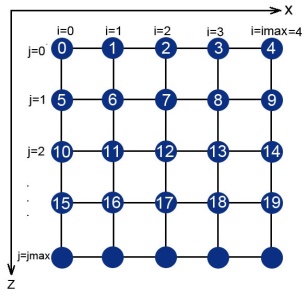
[Листинг проекта 6](#_Toc36477190)

# Задание

Создать водную поверхность, прозрачную до дна (взять произвольный рисунок). По поверхности должна идти волна.

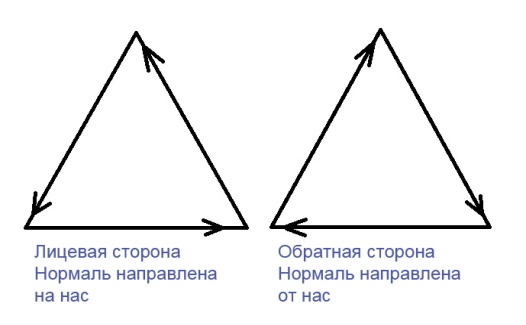
# Выполнение

Пусть сетка будет лежать в плоскости XZ, а значение Y будет вычисляться как функция от X и Z, т.е. y=f(x,z). В узлах сетки будут находиться вершины. Обозначили порядковый номер  узла сетки вдоль оси X как i, а вдоль оси Z как j. Номера узлов могут меняться от нуля до imax или jmax соответственно.

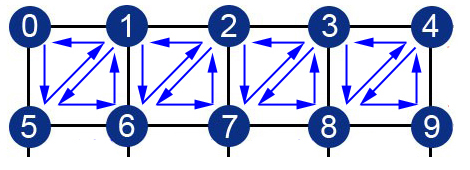
[](http://2.bp.blogspot.com/-X5gQo7vbd6I/UL_8aD7EjkI/AAAAAAAAAUQ/SYp9ML5-GDE/s1600/setka.jpg)

Обозначили шаг сетки вдоль оси X как dx, а шаг сетки вдоль оси Z как dz. Тогда мы можем легко вычислить координаты X и Z для всех вершин. Координата Y у нас меняется в зависимости от двух координат других координат X и Z. Поэтому будем хранить ее в двумерном массиве.

Для того чтобы использовать освещение нужно было вычислить нормаль для каждой  вершины сетки. Вектор нормали - это вектор единичной длины, перпендикулярный к поверхности в данной точке этой поверхности и направленный от обратной стороны поверхности к лицевой стороне. Чтобы нарисовать поверхность, потребовалось ее разбить на множество треугольников. Лицевой стороной треугольника является сторона, которая при рисовании обходится по вершинам против часовой стрелки:

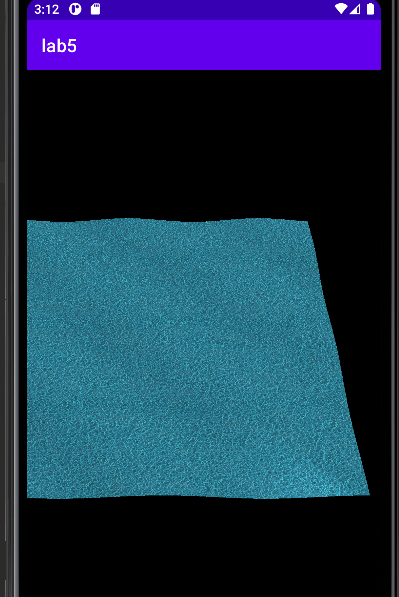
[](http://1.bp.blogspot.com/-9rm-XwZJknQ/UMAdS9UoozI/AAAAAAAAAUs/xXO3Z8lsuvY/s1600/obhod.jpg)

Далее рисовали поверхность в одном проходе, т.е. с использованием одной команды glDrawElements и правила обхода вершин GL\_TRIANGLE\_STRIP. Разбили сетку на ленты из треугольников. Правило GL\_TRIANGLE\_STRIP автоматически создает ряд треугольников 0-5-1, 1-5-6, 1-6-2, 2-6-7, 2-7-3, 3-7-8, 3-8-4, 4-8-9. Порядок перечисления вершин выбран так, чтобы обход выполнялся против часовой стрелки и верхняя сторона сетки считалась лицевой.

[](http://3.bp.blogspot.com/-s8Uvv8ubJ98/UMFxDgX1O5I/AAAAAAAAAWc/meNT6rsnmt4/s1600/lenta2.jpg)

Таким образом, чтобы сделать единую цепь вершин для правила GL\_TRIANGLE\_STRIP нужно дублировать последнюю вершину в каждой ленте. Порядок перечисления вершин называют массивом индексов и передают OpenGL в виде буфера.

**Результат**:



# Листинг проекта

MainActivity.java

package com.example.lab5;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.os.Bundle;  
import android.opengl.GLSurfaceView;  
  
  
public class MainActivity extends AppCompatActivity {  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 MyRenderer render = new MyRenderer(this);  
 super.onCreate(savedInstanceState);  
 GLSurfaceView canvas = new GLSurfaceView(this);  
 canvas.setEGLContextClientVersion(2);  
 canvas.setRenderer(render);  
 canvas.setRenderMode(GLSurfaceView.RENDERMODE\_CONTINUOUSLY);  
 setContentView(canvas);  
 }  
}

MyRenderer.java

package com.example.lab5;  
  
import android.annotation.SuppressLint;  
import android.content.Context;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.opengl.GLES20;  
import android.opengl.GLSurfaceView;  
import android.opengl.GLUtils;  
import android.opengl.Matrix;  
  
import java.io.InputStream;  
import java.nio.ByteBuffer;  
import java.nio.ByteOrder;  
import java.nio.FloatBuffer;  
import java.nio.ShortBuffer;  
  
import javax.microedition.khronos.egl.EGLConfig;  
import javax.microedition.khronos.opengles.GL10;  
  
  
class MyRenderer implements GLSurfaceView.Renderer {  
 Context c;  
 private int[] texture = new int [1];  
 private float x\_camera, y\_camera, z\_camera;  
 private float x\_light\_position, y\_light\_position, z\_light\_position;  
 private float[] model\_matrix;  
 private float[] view\_matrix;  
 private float[] model\_view\_matrix;  
 private float[] projection\_matrix;  
 private float[] model\_view\_projection\_matrix;  
 private int max\_size\_x = 60;  
 private int max\_size\_z = 60;  
 private int size\_index;  
 private float x0 =- 1.3f;  
 private float z0 = -2f;  
 private float dx = 0.05f;  
 private float dz = 0.05f;  
 private float [] x;  
 private float [][] y;  
 private float [] z;  
 private float [] vertexes;  
 private float [][] normales\_x;  
 private float [][] normales\_y;  
 private float [][] normales\_z;  
 private float [] normales;  
 private FloatBuffer vertexes\_buffer, normales\_buffer;  
 private ShortBuffer index\_buffer;  
 private Shader m\_shader;  
  
 public MyRenderer(Context context) {  
 c = context;  
 x\_light\_position = 5f;  
 y\_light\_position = 30f;  
 z\_light\_position = 5f;  
 model\_matrix = new float [16];  
 view\_matrix = new float [16];  
 model\_view\_matrix = new float [16];  
 projection\_matrix = new float [16];  
 model\_view\_projection\_matrix = new float [16];  
 Matrix.setIdentityM(model\_matrix, 0);  
 x\_camera = -3.0f;  
 y\_camera = 3.0f;  
 z\_camera = 0.0f;  
 Matrix.setLookAtM(view\_matrix, 0, x\_camera, y\_camera, z\_camera, -0.5f, 0.2f, 0, 0, 1, 0);  
 Matrix.multiplyMM(model\_view\_matrix, 0, view\_matrix, 0, model\_matrix, 0);  
 x = new float [max\_size\_x + 1];  
 z = new float [max\_size\_z + 1];  
 y = new float [max\_size\_z + 1][max\_size\_x + 1];  
 vertexes = new float [(max\_size\_z + 1) \* (max\_size\_x + 1) \* 3];  
 normales\_x = new float [max\_size\_z + 1][max\_size\_x + 1];  
 normales\_y = new float [max\_size\_z + 1][max\_size\_x + 1];  
 normales\_z = new float [max\_size\_z + 1][max\_size\_x + 1];  
 normales = new float [(max\_size\_z + 1) \* (max\_size\_x + 1) \* 3];  
 for (int i = 0; i <= max\_size\_x; i++) {  
 x[i] = x0 + i \* dx;  
 }  
 for (int j = 0; j <= max\_size\_z; j++) {  
 z[j] = z0 + j \* dz;  
 }  
 ByteBuffer vb = ByteBuffer.allocateDirect((max\_size\_z + 1) \* (max\_size\_x + 1) \* 3 \* 4);  
 vb.order(ByteOrder.nativeOrder());  
 vertexes\_buffer = vb.asFloatBuffer();  
 vertexes\_buffer.position(0);  
 ByteBuffer nb = ByteBuffer.allocateDirect((max\_size\_z + 1) \* (max\_size\_x + 1) \* 3 \* 4);  
 nb.order(ByteOrder.nativeOrder());  
 normales\_buffer = nb.asFloatBuffer();  
 normales\_buffer.position(0);  
 short[] index;  
 size\_index = 2 \* (max\_size\_x + 1) \* max\_size\_z + (max\_size\_z - 1);  
 index = new short [size\_index];  
 int k = 0;  
 int j = 0;  
 while (j < max\_size\_z) {  
 for (int i = 0; i <= max\_size\_x; i++) {  
 index[k] = chain(j, i);  
 k++;  
 index[k] = chain(j + 1, i);  
 k++;  
 }  
 if (j < max\_size\_z - 1) {  
 index[k] = chain(j + 1, max\_size\_x);  
 k++;  
 }  
 j++;  
 if (j < max\_size\_z) {  
 for (int i = max\_size\_x; i >= 0; i--) {  
 index[k] = chain(j, i);  
 k++;  
 index[k] = chain(j + 1, i);  
 k++;  
 }  
 if (j < max\_size\_z - 1) {  
 index[k] = chain(j + 1,0);  
 k++;  
 }  
 j++;  
 }  
 }  
 ByteBuffer bi = ByteBuffer.allocateDirect(size\_index \* 2);  
 bi.order(ByteOrder.nativeOrder());  
 index\_buffer = bi.asShortBuffer();  
 index\_buffer.put(index);  
 index\_buffer.position(0);  
 get\_vertexes();  
 get\_normales();  
 }  
  
 private short chain(int j, int i) {  
 return (short) (i + j \* (max\_size\_x + 1));  
 }  
  
 private void get\_vertexes() {  
 double time = System.currentTimeMillis();  
 for (int j = 0; j <= max\_size\_z; j++) {  
 for (int i = 0; i <= max\_size\_x; i++){  
 y[j][i] = 0.02f \* (float) Math.cos(0.005 \* time + 5 \* (z[j] + x[i]));  
 }  
 }  
 int k = 0;  
 for (int j = 0; j <= max\_size\_z; j++) {  
 for (int i = 0; i <= max\_size\_x; i++) {  
 vertexes[k] = x[i];  
 k++;  
 vertexes[k] = y[j][i];  
 k++;  
 vertexes[k] = z[j];  
 k++;  
 }  
 }  
 vertexes\_buffer.put(vertexes);  
 vertexes\_buffer.position(0);  
 }  
  
 private void get\_normales() {  
 for (int j = 0; j < max\_size\_z; j++) {  
 for (int i = 0; i < max\_size\_x; i++) {  
 normales\_x[j][i] = -(y[j][i+1] - y[j][i]) \* dz;  
 normales\_y[j][i] = dx \* dz;  
 normales\_z[j][i] = -dx \* (y[j+1][i] - y[j][i]);  
 }  
 }  
 for (int j = 0; j < max\_size\_z; j++) {  
 normales\_x[j][max\_size\_x] = (y[j][max\_size\_x - 1] - y[j][max\_size\_x]) \* dz;  
 normales\_y[j][max\_size\_x] = dx \* dz;  
 normales\_z[j][max\_size\_x] = -dx \* (y[j + 1][max\_size\_x] - y[j][max\_size\_x]);  
 }  
 for (int i = 0; i < max\_size\_x; i++) {  
 normales\_x[max\_size\_z][i] = -(y[max\_size\_z][i + 1] - y[max\_size\_z][i]) \* dz;  
 normales\_y[max\_size\_z][i] = dx \* dz;  
 normales\_z[max\_size\_z][i] = dx \* (y[max\_size\_z - 1][i] - y[max\_size\_z][i]);  
 }  
 normales\_x[max\_size\_z][max\_size\_x]= (y[max\_size\_z][max\_size\_x - 1] - y[max\_size\_z][max\_size\_x]) \* dz;  
 normales\_y[max\_size\_z][max\_size\_x] = dx \* dz;  
 normales\_z[max\_size\_z][max\_size\_x] = dx \* (y[max\_size\_z - 1][max\_size\_x] - y[max\_size\_z][max\_size\_x]);  
 int k = 0;  
 for (int j = 0; j <= max\_size\_z; j++) {  
 for (int i = 0; i <= max\_size\_x; i++) {  
 normales[k] = normales\_x[j][i];  
 k++;  
 normales[k] = normales\_y[j][i];  
 k++;  
 normales[k] = normales\_z[j][i];  
 k++;  
 }  
 }  
 normales\_buffer.put(normales);  
 normales\_buffer.position(0);  
 }  
  
 @SuppressLint("ResourceType")  
 @Override  
 public void onSurfaceCreated(GL10 gl, EGLConfig config) {  
 gl.glGenTextures(1, texture, 0);  
 InputStream stream;  
 Bitmap bitmap;  
 stream = c.getResources().openRawResource(R.drawable.water);  
 bitmap = BitmapFactory.decodeStream(stream);  
 gl.glBindTexture(GL10.GL\_TEXTURE\_2D, texture[0]);  
 gl.glTexParameterf(GL10.GL\_TEXTURE\_2D, GL10.GL\_TEXTURE\_MIN\_FILTER,  
 GL10.GL\_LINEAR);  
 GLUtils.texImage2D(GL10.GL\_TEXTURE\_2D, 0, bitmap, 0);  
 GLES20.glEnable(GLES20.GL\_DEPTH\_TEST);  
 String vertex\_shader =  
 "uniform mat4 u\_modelViewProjectionMatrix;"+  
 "attribute vec3 a\_vertex;"+  
 "attribute vec3 a\_normal;"+  
 "varying vec3 v\_vertex;"+  
 "varying vec3 v\_normal;"+  
 "void main() {"+  
 "v\_vertex = a\_vertex;"+  
 "vec3 n\_normal = normalize(a\_normal);"+  
 "v\_normal = n\_normal;"+  
 "gl\_Position = u\_modelViewProjectionMatrix \* vec4(a\_vertex, 1.0);"+  
 "}";  
  
 String fragment\_shader =  
 "precision mediump float;" +  
 "uniform vec3 u\_camera;" +  
 "uniform vec3 u\_lightPosition;" +  
 "uniform sampler2D u\_texture0;" +  
 "varying vec3 v\_vertex;" +  
 "varying vec3 v\_normal;" +  
 "vec3 myrefract(vec3 IN, vec3 NORMAL, float k) {" +  
 " float nv = dot(NORMAL,IN);" +  
 " float v2 = dot(IN,IN);" +  
 " float knormal = (sqrt(((k \* k - 1.0) \* v2) / (nv \* nv) + 1.0) - 1.0) \* nv;" +  
 " vec3 OUT = IN + (knormal \* NORMAL);" +  
 " return OUT;" +  
 "}" +  
 "void main() {" +  
 " vec3 n\_normal = normalize(v\_normal);" +  
 " vec3 lightvector = normalize(u\_lightPosition - v\_vertex);" +  
 " vec3 lookvector = normalize(u\_camera - v\_vertex);" +  
 " float ambient = 0.1;" +  
 " float k\_diffuse = 0.7;" +  
 " float k\_specular = 0.3;" +  
 " float diffuse = k\_diffuse \* max(dot(n\_normal, lightvector), 0.0);" +  
 " vec3 reflectvector = reflect(-lightvector, n\_normal);" +  
 " float specular = k\_specular \* pow( max(dot(lookvector,reflectvector),0.0), 40.0);" +  
 " vec4 one = vec4(1.0,1.0,1.0,1.0);" +  
 " vec4 lightColor = (ambient + diffuse + specular) \* one;" +  
 " vec3 OUT = myrefract(-lookvector, n\_normal, 1.2);" +  
 " float ybottom = -1.0;" +  
 " float xbottom = v\_vertex.x + OUT.x \* (ybottom - v\_vertex.y) / OUT.y;" +  
 " float zbottom = v\_vertex.z + OUT.z \* (ybottom - v\_vertex.y) / OUT.y;" +  
 " vec2 texCoord = vec2(xbottom, zbottom);" +  
 " vec4 textureColor = texture2D(u\_texture0, texCoord);" +  
 " gl\_FragColor = lightColor \* textureColor;" +  
 "}";  
 m\_shader = new Shader(vertex\_shader, fragment\_shader);  
 m\_shader.link\_vertex\_buffer(vertexes\_buffer);  
 m\_shader.link\_normal\_buffer(normales\_buffer);  
 m\_shader.link\_texture(texture);  
 }  
  
 @Override  
 public void onSurfaceChanged(GL10 gl, int width, int height) {  
 gl.glViewport(0, 0, width, height);  
 float ratio = (float) width / height;  
 float k = 0.055f;  
 float left = -k \* ratio;  
 float right = k \* ratio;  
 float bottom = -k;  
 float top = k;  
 float near = 0.1f;  
 float far = 10.0f;  
 Matrix.frustumM(projection\_matrix, 0, left, right, bottom, top, near, far);  
 Matrix.multiplyMM(model\_view\_projection\_matrix, 0, projection\_matrix, 0, model\_view\_matrix, 0);  
 }  
  
 @Override  
 public void onDrawFrame(GL10 gl) {  
 m\_shader.link\_model\_view\_projection\_matrix(model\_view\_projection\_matrix);  
 m\_shader.link\_camera(x\_camera, y\_camera, z\_camera);  
 m\_shader.link\_light\_source(x\_light\_position, y\_light\_position, z\_light\_position);  
 get\_vertexes();  
 get\_normales();  
 GLES20.glClear(GLES20.GL\_COLOR\_BUFFER\_BIT | GLES20.GL\_DEPTH\_BUFFER\_BIT);  
 GLES20.glDrawElements(GLES20.GL\_TRIANGLE\_STRIP, size\_index, GLES20.GL\_UNSIGNED\_SHORT, index\_buffer);  
 }  
}

Shader.java

package com.example.lab5;  
  
import android.opengl.GLES20;  
import java.nio.FloatBuffer;  
  
public class Shader {  
 private int program\_handle;  
  
 public Shader(String vertex\_shader, String fragment\_shader) {  
 create\_program(vertex\_shader, fragment\_shader);  
 }  
  
 private void create\_program(String vertex\_shader, String fragment\_shader) {  
 int vertex\_shader\_handle = GLES20.glCreateShader(GLES20.GL\_VERTEX\_SHADER);  
 GLES20.glShaderSource(vertex\_shader\_handle, vertex\_shader);  
 GLES20.glCompileShader(vertex\_shader\_handle);  
 int fragment\_shader\_handle = GLES20.glCreateShader(GLES20.GL\_FRAGMENT\_SHADER);  
 GLES20.glShaderSource(fragment\_shader\_handle, fragment\_shader);  
 GLES20.glCompileShader(fragment\_shader\_handle);  
 program\_handle = GLES20.glCreateProgram();  
 GLES20.glAttachShader(program\_handle, vertex\_shader\_handle);  
 GLES20.glAttachShader(program\_handle, fragment\_shader\_handle);  
 GLES20.glLinkProgram(program\_handle);  
 }  
  
 public void link\_vertex\_buffer(FloatBuffer vertexBuffer) {  
 GLES20.glUseProgram(program\_handle);  
 int a\_vertex\_handle = GLES20.glGetAttribLocation(program\_handle, "a\_vertex");  
 GLES20.glEnableVertexAttribArray(a\_vertex\_handle);  
 GLES20.glVertexAttribPointer(a\_vertex\_handle, 3, GLES20.GL\_FLOAT, false, 0, vertexBuffer);  
 }  
  
 public void link\_normal\_buffer(FloatBuffer normalBuffer) {  
 GLES20.glUseProgram(program\_handle);  
 int a\_normal\_handle = GLES20.glGetAttribLocation(program\_handle, "a\_normal");  
 GLES20.glEnableVertexAttribArray(a\_normal\_handle);  
 GLES20.glVertexAttribPointer(a\_normal\_handle, 3, GLES20.GL\_FLOAT, false, 0, normalBuffer);  
 }  
  
 public void link\_model\_view\_projection\_matrix(float[] modelViewProjectionMatrix) {  
 GLES20.glUseProgram(program\_handle);  
 int u\_model\_view\_projection\_matrix\_handle = GLES20.glGetUniformLocation(program\_handle, "u\_modelViewProjectionMatrix");  
 GLES20.glUniformMatrix4fv(u\_model\_view\_projection\_matrix\_handle, 1, false, modelViewProjectionMatrix, 0);  
 }  
  
 public void link\_camera(float xCamera, float yCamera, float zCamera) {  
 GLES20.glUseProgram(program\_handle);  
 int u\_camera\_handle = GLES20.glGetUniformLocation(program\_handle, "u\_camera");  
 GLES20.glUniform3f(u\_camera\_handle, xCamera, yCamera, zCamera);  
 }  
  
 public void link\_light\_source(float xLightPosition, float yLightPosition, float zLightPosition) {  
 GLES20.glUseProgram(program\_handle);  
 int u\_light\_source\_handle = GLES20.glGetUniformLocation(program\_handle, "u\_lightPosition");  
 GLES20.glUniform3f(u\_light\_source\_handle, xLightPosition, yLightPosition, zLightPosition);  
 }  
  
 public void link\_texture(int[] texture) {  
 int u\_texture\_Handle = GLES20.glGetUniformLocation(program\_handle, "u\_texture0");  
 GLES20.glActiveTexture(GLES20.GL\_TEXTURE0);  
 GLES20.glBindTexture(GLES20.GL\_TEXTURE\_2D, texture[0]);  
 GLES20.glUniform1i(u\_texture\_Handle, 0);  
 }  
  
 public void use\_program() {  
 GLES20.glUseProgram(program\_handle);  
 }  
  
}