Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

Программирование мобильных устройств

Лабораторная работа №5

Водная гладь

Выполнил: студент 4 курса

Ф. ИВТ, группа: ИП-711

Мартасов И. О.

Проверил: доцент кафедры ПМиК

Нечта Иван Васильевич

Новосибирск, 2020

**Содержание**

1. **Постановка задачи..………………………………………………………3**
2. **Исходный код……..……………………………………………………….3**
3. **Результаты работы…..………………………………………………….10**

**Постановка задачи**

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

**Исходный код**

**MyGL20Renderer.java – исходники водной глади и создание шейдеров**

class P

{

P()

{

z = 0;

vz = 0;

}

public float x, y, z, vz;

}

class MyGL20Renderer implements GLSurfaceView.Renderer

{

private Shader mShader;

Texture mTexture0;

private final float xСamera = 0f;

private final float yCamera = 0f;

private final float zCamera = 1f;

private final float xLightPosition;

private final float yLightPosition;

private final float zLightPosition;

private final float[] viewMatrix;

private final float[] modelViewMatrix;

private final float[] projectionMatrix;

private final float[] modelViewProjectionMatrix;

private final Context context;

final int N = 80;

float K = 0.06f;

float DT = 0.1f;

int offs = 0;

public P [][] p;

float time;

float [] a;

float [] n;

FloatBuffer f;

ByteBuffer b;

float sqr(float x)

{

return x \* x;

}

void NioBuff()

{

b = ByteBuffer.allocateDirect(2 \* 2 \* 3 \* N \* N \* 4);

b.order(ByteOrder.nativeOrder());

f = b.asFloatBuffer();

f.put(a);

f.position(0);

}

void Init1()

{

time = 0;

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

p[i][j] = new P();

(p[i][j]).x = 1.0f \* j / N;

(p[i][j]).y = 1.0f \* i / N;

(p[i][j]).z = 0;

(p[i][j]).vz = 0;

}

}

}

void display()

{

offs = 0;

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N - 1; j++)

{

a[N \* i \* 3 \* 2 + j \* 3 \* 2] = (float) j / N;

a[N \* i \* 3 \* 2 + j \* 3 \* 2 + 1]=(float)i/N;

a[N \* i \* 3 \* 2 + j \* 3 \* 2 + 2] = (p[i][j]).z;

a[N \* i \* 3 \* 2 + j \* 3 \* 2 + 3] = (float)(j + 1) / N;

a[N \* i \* 3 \* 2 + j \* 3 \* 2 + 4] = (float)(i) / N;

a[N\*i\*3\*2+j\*3\*2+5]=(p[i][j+1]).z;

offs+=6;

}

}

for (int i = 0; i < N - 1; i++)

{

for (int j = 0; j < N; j++)

{

a[offs + N \* i \* 3 \* 2 + j \* 3 \* 2] = 1.0f \* j / N;

a[offs + N\*i\*3\*2+j\*3\*2+1] = 1.0f \* i / N;

a[offs + N \* i \* 3 \* 2 + j \* 3 \* 2 + 2] = 1.0f \* (p[i][j]).z;

a[offs + N \* i \* 3 \* 2 + j \* 3 \* 2 + 3] = 1.0f \* (j) / N;

a[offs + N \* i \* 3 \* 2 + j \* 3 \* 2 + 4] = 1.0f \* (i + 1) / N;

a[offs + N \* i \* 3 \* 2 + j \* 3 \* 2 + 5] = 1.0f \* (p[i + 1][j]).z;

}

}

}

@Override

public void onSurfaceChanged(GL10 gl, int width, int height)

{

Init1();

GLES20.glViewport(0, 0, width, height);

float ratio = (float) width / height;

float k = K;

float left = -k \* ratio;

float right = k \* ratio;

float bottom = -k;

float near = 0.1f;

float far = 20.0f;

Matrix.frustumM(projectionMatrix, 0, left, right, bottom, k, near, far);

Matrix.multiplyMM( modelViewProjectionMatrix, 0, projectionMatrix, 0, modelViewMatrix, 0);

}

void Push1()

{

if (Math.random() \* 500 > 10)

{

return;

}

int x0=(int)(Math.random()\*N/2+1);

int y0=(int)(Math.random()\*N/2+1);

for (int y = y0-5; y<y0+5; y++){

if ((y<1)||(y>=N-1)) continue;

for (int x = x0-5; x<x0+5; x++) {

if ((x<1)||(x>=N-1)) continue;

p[x][y].z = 10.0f / N - (float) (Math.sqrt(sqr(y - y0) + sqr(x - x0)) \* 1.0 / N);

}

}

}

void MyTimer()

{

final int []dx={-1,0,1,0};

final int []dy={0,1,0,-1};

Push1();

for (int y=1;y<N-1;++y){

for (int x=1;x<N-1;++x)

{

P p0=p[x][y];

for (int i=0;i<4;++i)

{

P p1=p[x+dx[i]][y+dy[i]];

float d=(float)Math.sqrt(sqr(p0.x-p1.x)+sqr(p0.y-p1.y) +sqr(p0.z-p1.z));

p0.vz+=K\*(p1.z-p0.z)/d\*DT;

}

}

}

for (int y=1;y<N-1;++y)

{

for (int x = 1; x < N - 1; ++x)

{

P p0 = p[x][y];

p0.z += p0.vz;

}

}

display();

}

private final float[] scalerMatrix = new float[16];

private final float[] mRotationMatrixY = new float[16];

private final float[] mRotationMatrixX = new float[16];

private final float[] mTempMatrix = new float[16];

private final float[] projView = new float[16];

public MyGL20Renderer(Context context)

{

this.context=context;

xLightPosition = 5f;

yLightPosition = 30f;

zLightPosition = 5f;

a = new float [12\*N\*N];

n = new float [12\*N\*N];

p = new P[N][N];

float[] modelMatrix = new float[16];

viewMatrix=new float[16];

modelViewMatrix=new float[16];

projectionMatrix=new float[16];

modelViewProjectionMatrix=new float[16];

Matrix.setIdentityM(modelMatrix, 0);

Init1();

MyTimer();

NioBuff();

}

public void onSurfaceCreated(GL10 unused, EGLConfig config)

{

GLES20.glEnable(GLES20.GL\_DEPTH\_TEST);

GLES20.glEnable(GLES20.GL\_CULL\_FACE);

GLES20.glHint(GLES20.GL\_GENERATE\_MIPMAP\_HINT, GLES20.GL\_NICEST);

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);"+

"}";

mTexture0 = new Texture(context, R.drawable.sun);

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, 1.0);" +

"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(0.5\*xbottom,0.5\*zbottom);" +

"vec4 textureColor=texture2D(u\_texture0, texCoord);" +

"gl\_FragColor=lightColor\*textureColor;" +

"}";

mShader = new Shader(vertex\_shader, fragment\_shader);

mShader.linkVertexBuffer(f);

mShader.linkNormalBuffer(f);

mShader.linkTexture(mTexture0, null);

mShader.linkModelViewProjectionMatrix(modelViewProjectionMatrix);

mShader.linkCamera(xСamera, yCamera, zCamera);

mShader.linkLightSource(xLightPosition, yLightPosition, zLightPosition);

}

public void onDrawFrame(GL10 unused)

{

Matrix.setLookAtM(viewMatrix, 0, xСamera, yCamera, zCamera, 0, 0, 0, 0, 1, 0);

Matrix.multiplyMM(projView, 0, projectionMatrix, 0, viewMatrix, 0);

Matrix.setRotateM(mRotationMatrixX, 0, 0.6f, 0, 1.0f, 0f);

Matrix.setRotateM(mRotationMatrixY, 0, 0.6f, 1.0f, 0, 0);

Matrix.multiplyMM(mTempMatrix, 0, projView, 0, mRotationMatrixX, 0);

Matrix.multiplyMM(modelViewProjectionMatrix, 0, mTempMatrix, 0, mRotationMatrixY, 0);

Matrix.setIdentityM(scalerMatrix, 0);

Matrix.scaleM(scalerMatrix, 0, 1f, 1f, 0f);

Matrix.multiplyMM(mTempMatrix, 0, modelViewProjectionMatrix, 0, scalerMatrix, 0);

Matrix.setIdentityM(mTempMatrix, 0);

Matrix.translateM(mTempMatrix, 0, -0.5f, -0.5f, 0f);

Matrix.multiplyMM(modelViewProjectionMatrix, 0, modelViewProjectionMatrix, 0, mTempMatrix, 0);

mShader.linkModelViewProjectionMatrix(modelViewProjectionMatrix);

mShader.linkCamera(xСamera, yCamera, zCamera);

mShader.linkLightSource(xLightPosition, yLightPosition, zLightPosition);

GLES20.glClear(GLES20.GL\_COLOR\_BUFFER\_BIT | GLES20.GL\_DEPTH\_BUFFER\_BIT);

GLES20.glDrawArrays(GL10.GL\_LINES,0,4\*N\*N);

MyTimer();

f.put(a);

f.position(0);

}

}

**Shader.java – класс работы с шейдерами**

public class Shader

{

private int program\_Handle;

public Shader(String vertexShaderCode, String fragmentShaderCode)

{

createProgram(vertexShaderCode, fragmentShaderCode);

}

private void createProgram(String vertexShaderCode, String fragmentShaderCode)

{

int vertexShader\_Handle = GLES20.glCreateShader(GLES20.GL\_VERTEX\_SHADER);

GLES20.glShaderSource(vertexShader\_Handle, String.valueOf(vertexShaderCode));

GLES20.glCompileShader(vertexShader\_Handle);

int fragmentShader\_Handle = GLES20.glCreateShader(GLES20.GL\_FRAGMENT\_SHADER);

GLES20.glShaderSource(fragmentShader\_Handle, String.valueOf(fragmentShaderCode));

GLES20.glCompileShader(fragmentShader\_Handle);

program\_Handle = GLES20.glCreateProgram();

GLES20.glAttachShader(program\_Handle, vertexShader\_Handle);

GLES20.glAttachShader(program\_Handle, fragmentShader\_Handle);

GLES20.glLinkProgram(program\_Handle);

}

public void linkVertexBuffer(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 linkNormalBuffer(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 linkModelViewProjectionMatrix(float [] modelViewProjectionMatrix)

{

GLES20.glUseProgram(program\_Handle);

int u\_modelViewProjectionMatrix\_Handle = GLES20.glGetUniformLocation(program\_Handle, "u\_modelViewProjectionMatrix");

GLES20.glUniformMatrix4fv(u\_modelViewProjectionMatrix\_Handle, 1, false, modelViewProjectionMatrix, 0);

}

public void linkCamera (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 linkLightSource (float xLightPosition, float yLightPosition, float zLightPosition)

{

GLES20.glUseProgram(program\_Handle);

int u\_lightPosition\_Handle=GLES20.glGetUniformLocation(program\_Handle, "u\_lightPosition");

GLES20.glUniform3f(u\_lightPosition\_Handle, xLightPosition, yLightPosition, zLightPosition);

}

public void linkTexture(Texture texture0,Texture texture1)

{

GLES20.glUseProgram(program\_Handle);

if (texture0 != null)

{

int u\_texture0\_Handle = GLES20.glGetUniformLocation(program\_Handle, "u\_texture0");

GLES20.glActiveTexture(GLES20.GL\_TEXTURE0);

GLES20.glBindTexture(GLES20.GL\_TEXTURE\_2D, texture0.getName());

GLES20.glUniform1i(u\_texture0\_Handle, 0);

}

if (texture1 != null)

{

int u\_texture1\_Handle = GLES20.glGetUniformLocation(program\_Handle, "u\_texture1");

GLES20.glActiveTexture(GLES20.GL\_TEXTURE1);

GLES20.glBindTexture(GLES20.GL\_TEXTURE\_2D, texture1.getName());

GLES20.glUniform1i(u\_texture1\_Handle, 1);

}

}

}

**Texture.java – класс установки текстуры**

public class Texture

{

private final int name;

public Texture(Context context, int picture)

{

int [] texture\_name = new int[1];

GLES20.glGenTextures(1, texture\_name, 0);

name = texture\_name[0];

GLES20.glPixelStorei(GLES20.GL\_UNPACK\_ALIGNMENT, 1);

GLES20.glBindTexture(GLES20.GL\_TEXTURE\_2D, name);

GLES20.glTexParameteri(GLES20.GL\_TEXTURE\_2D, GLES20.GL\_TEXTURE\_MIN\_FILTER, GLES20.GL\_LINEAR\_MIPMAP\_LINEAR);

GLES20.glTexParameteri(GLES20.GL\_TEXTURE\_2D, GLES20.GL\_TEXTURE\_MAG\_FILTER, GLES20.GL\_LINEAR);

GLES20.glTexParameteri(GLES20.GL\_TEXTURE\_2D, GLES20.GL\_TEXTURE\_WRAP\_S, GLES20.GL\_REPEAT);

GLES20.glTexParameteri(GLES20.GL\_TEXTURE\_2D, GLES20.GL\_TEXTURE\_WRAP\_T, GLES20.GL\_REPEAT);

Bitmap bitmap = BitmapFactory.decodeResource(context.getResources(), picture);

GLUtils.texImage2D(GLES20.GL\_TEXTURE\_2D, 0, bitmap, 0);

bitmap.recycle();

GLES20.glGenerateMipmap(GLES20.GL\_TEXTURE\_2D);

}

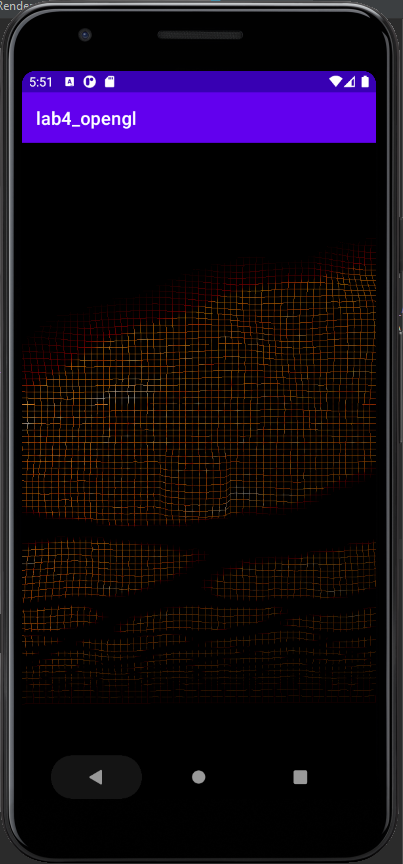
public int getName()

{

return name;

}

**Результаты работы**

****