# *Національний технічний університет України*

# *«Київський політехнічний інститут»*

#### ***Факультет інформатики та обчислювальної техніки***

## Лабораторна робота №5

*з курсу "****Комп’ютерна графіка****"*

***Виконав:***

*Бедь А.М.*

***Група*** *ІО-12,*

***Номер варіанту*** *1202*

***Номер підгрупи 15***

***Київ - 2013р.***

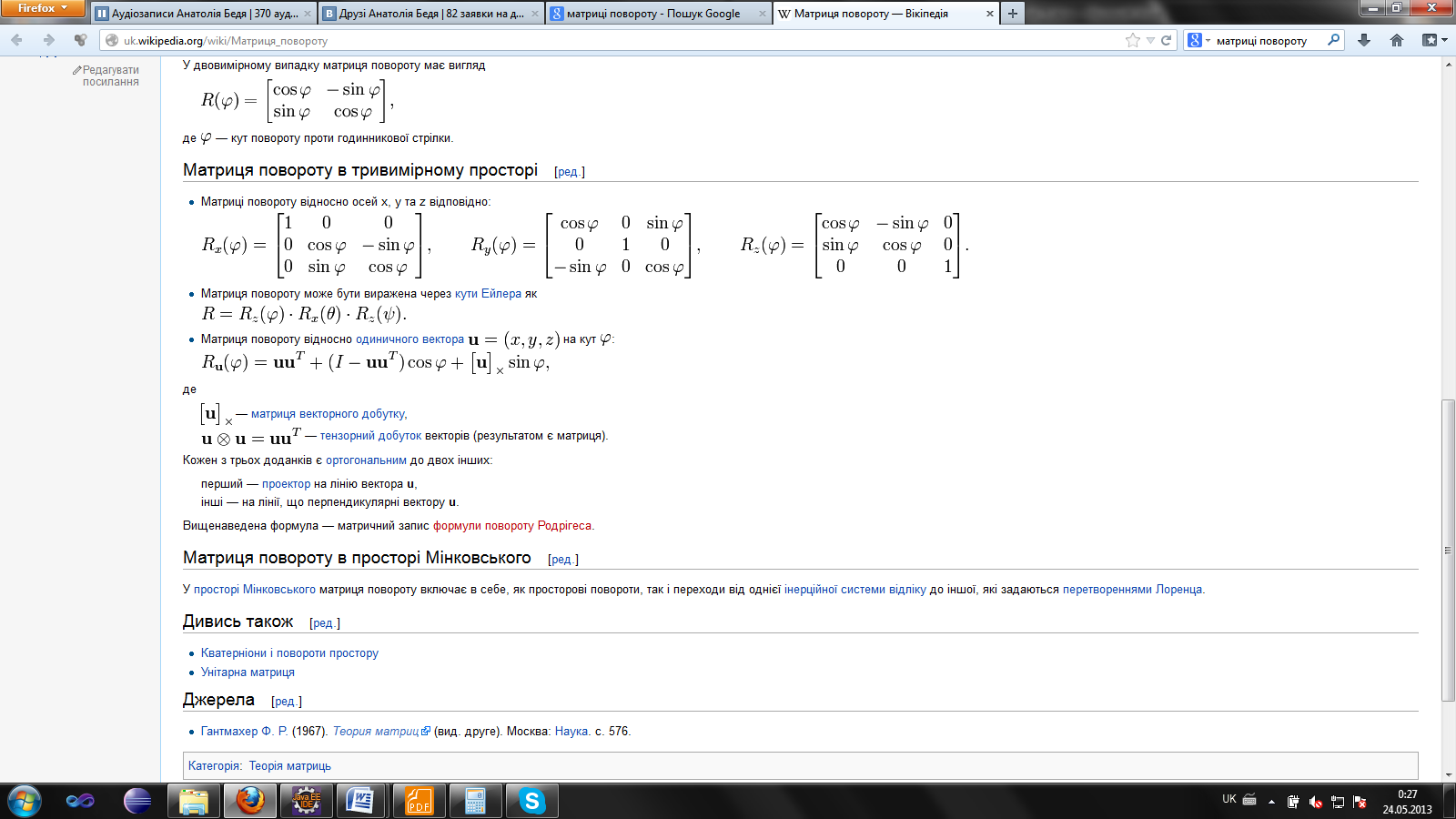
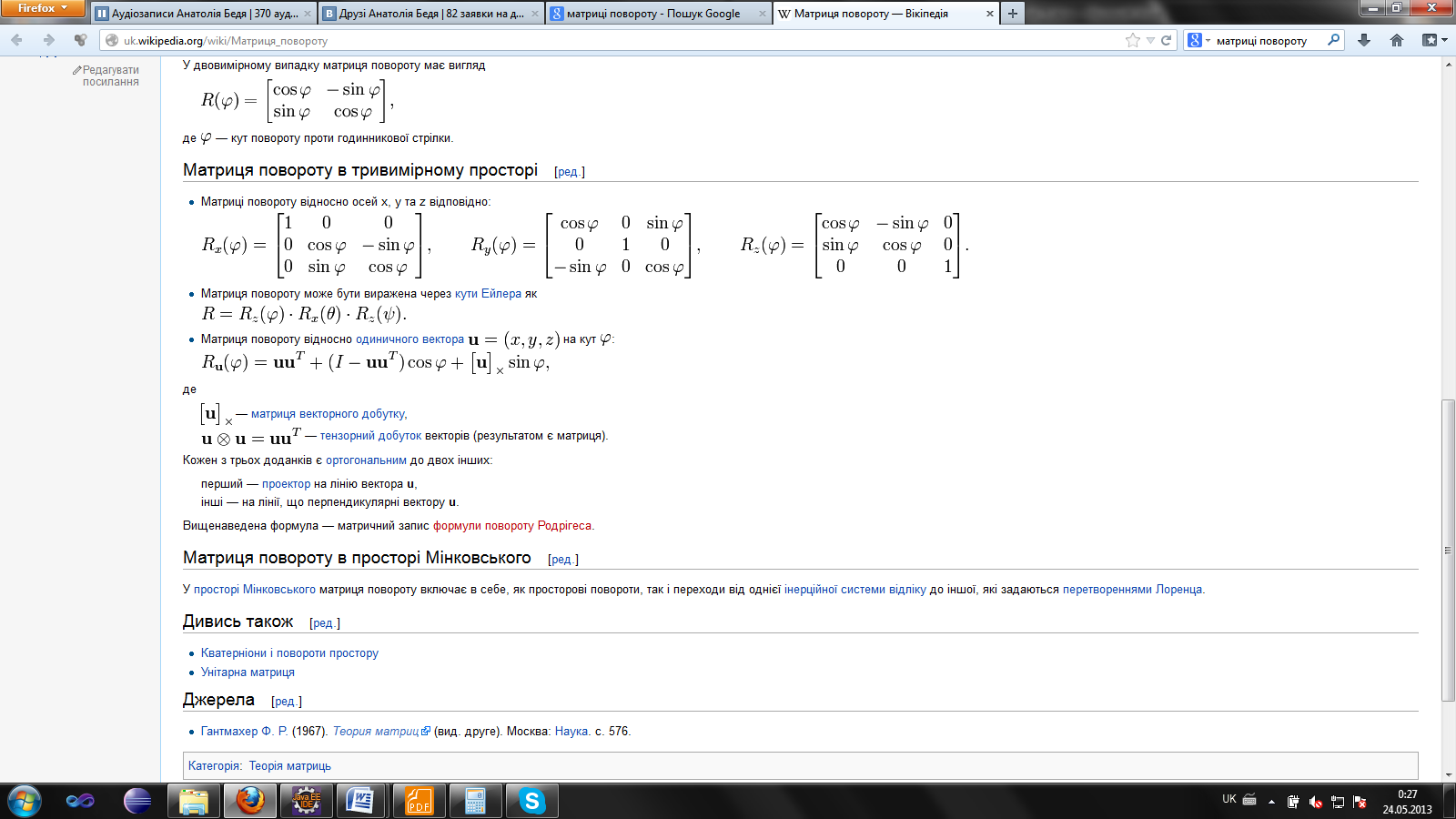
**Завдання.**

Виконати одночасно обернення навколо двох вісей заданої каркасної моделі геометричної фігури використовуючи матриці тривимірного перетворення. Координати однієї з вершин повинні бути (0,0,0).

Варіант:

**Теоретичні відомості.**

Для одночасного повороту навколо двох осей використовується матриця перетворень отримана добутком двох матриць повороту для кожної осі окремо



**Лістинг коду:**

**package** com.badlogic.androidgames.glbasics;

**import** java.util.Arrays;

**import** javax.microedition.khronos.opengles.GL10;

**import** com.badlogic.androidgames.framework.Game;

**import** com.badlogic.androidgames.framework.Screen;

**import** com.badlogic.androidgames.framework.gl.Vertices;

**import** com.badlogic.androidgames.framework.impl.GLGame;

**import** com.badlogic.androidgames.framework.impl.GLGraphics;

**public** **class** LineTest **extends** GLGame {

@Override

**public** Screen getStartScreen() {

**return** **new** LineTestScreen(**this**);

}

**class** LineTestScreen **extends** Screen {

GLGraphics glGraphics;

Vertices verticesLine;

LineModel lineModel;

Vertices vertices;

**float**[] center = **new** **float**[] { 0.5f, 0.5f, 0.5f };

**float**[] vecLine = **new** **float**[] { 0, 0, 0, 0, 0.5f, 0, 0, 0.5f, 0, 0, 1,

0.5f, 0, 1, 0.5f, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,

1, 0, 0, 1, 1, 0, 1, 1, 0, 0.5f, 1, 0, 0.5f, 1, 0, 0, 0.5f, 0,

0, 1, 0.5f, 0.5f, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0,

0, 0, 1, 0, 0 };

**public** LineTestScreen(Game game) {

**super**(game);

glGraphics = ((GLGame) game).getGLGraphics();

verticesLine = **new** Vertices(glGraphics, 24, 0, **false**, **false**, 3);

verticesLine.setVertices(Arrays.*copyOf*(vecLine, vecLine.length), 0,

72);

lineModel = **new** LineModel();

}

**private** **class** LineModel {

**float** angle;

**public** LineModel() {

angle = 0;

}

}

@Override

**public** **void** update(**float** deltaTime) {

game.getInput().getTouchEvents();

game.getInput().getKeyEvents();

**if** (game.getInput().isTouchDown(0)) {

lineModel.angle += 2;

}

}

@Override

**public** **void** present(**float** deltaTime) {

GL10 gl = glGraphics.getGL();

gl.glViewport(0, 0, glGraphics.getWidth(), glGraphics.getHeight());

gl.glClearColor(1, 1, 1, 1);

gl.glClear(GL10.*GL\_COLOR\_BUFFER\_BIT*);

gl.glMatrixMode(GL10.*GL\_PROJECTION*);

gl.glLoadIdentity();

gl.glOrthof(0, 320, 0, 480, -200, 200);

gl.glMatrixMode(GL10.*GL\_MODELVIEW*);

gl.glLoadIdentity();

gl.glTranslatef(150, 300, 0);

**float** v[];

v = verticesLine.getVerticesA();

scale(v, 100);

rotate(v, (**float**) (lineModel.angle \* Math.*PI* / 180));

verticesLine.setVertices(v, 0, v.length);

gl.glLineWidthx(5);

gl.glColor4f(0, 0, 0, 1);

verticesLine.bind();

verticesLine.draw(GL10.*GL\_LINES*, 0, 24);

verticesLine.unbind();

verticesLine.setVertices(Arrays.*copyOf*(vecLine, vecLine.length), 0,

vecLine.length);

center[0] = 0.5f;

center[1] = 0.5f;

center[2] = 0.5f;

}

**public** **void** scale(**float**[] t, **int** k) {

**for** (**int** i = 0; i < t.length; i++) {

t[i] = t[i] \* k;

}

**for** (**int** i = 0; i < center.length; i++) {

center[i] = center[i] \* k;

}

}

**public** **void** rotate(**float**[] t, **float** angle) {

**for** (**int** i = 0; i < t.length / 3; i++) {

t[i \* 3] = (**float**) (t[i \* 3] \* Math.*cos*(angle) + t[i \* 3 + 1]

\* Math.*pow*(Math.*sin*(angle), 2) + t[i \* 3 + 2]

\* Math.*sin*(angle) \* Math.*cos*(angle));

t[i \* 3 + 1] = (**float**) (t[i \* 3 + 1] \* Math.*cos*(angle) - t[i \* 3 + 2]

\* Math.*sin*(angle));

t[i \* 3 + 2] = (**float**) (-t[i \* 3] \* Math.*sin*(angle)

+ t[i \* 3 + 2] \* Math.*pow*(Math.*cos*(angle), 2) + t[i \* 3 + 1]

\* Math.*sin*(angle) \* Math.*cos*(angle));

}

center[0] = (**float**) (center[0] \* Math.*cos*(angle) + center[1]

\* Math.*pow*(Math.*sin*(angle), 2) + center[2]

\* Math.*sin*(angle) \* Math.*cos*(angle));

center[1] = (**float**) (center[1] \* Math.*cos*(angle) - center[2]

\* Math.*sin*(angle));

center[2] = (**float**) (-center[0] \* Math.*sin*(angle) + center[2]

\* Math.*pow*(Math.*cos*(angle), 2) + center[1]

\* Math.*sin*(angle) \* Math.*cos*(angle));

}

@Override

**public** **void** pause() {

}

@Override

**public** **void** resume() {

}

@Override

**public** **void** dispose() {

}

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**package** com.badlogic.androidgames.framework.gl;

**import** java.nio.ByteBuffer;

**import** java.nio.ByteOrder;

**import** java.nio.FloatBuffer;

**import** java.nio.ShortBuffer;

**import** javax.microedition.khronos.opengles.GL10;

**import** com.badlogic.androidgames.framework.impl.GLGraphics;

**public** **class** Vertices {

**final** GLGraphics glGraphics;

**final** **boolean** hasColor;

**final** **boolean** hasTexCoords;

**final** **int** vertexSize;

**final** **int** vec;

**final** FloatBuffer vertices;

**final** ShortBuffer indices;

**private** **float**[] bufVertices;

**public** Vertices(GLGraphics glGraphics, **int** maxVertices, **int** maxIndices,

**boolean** hasColor, **boolean** hasTexCoords, **int** vec) {

**this**.glGraphics = glGraphics;

**this**.hasColor = hasColor;

**this**.hasTexCoords = hasTexCoords;

**this**.vec = vec;

**this**.vertexSize = (vec + (hasColor ? 4 : 0) + (hasTexCoords ? 2 : 0)) \* 4;

ByteBuffer byteBuffer = ByteBuffer.*allocateDirect*(maxVertices

\* vertexSize);

byteBuffer.order(ByteOrder.*nativeOrder*());

vertices = byteBuffer.asFloatBuffer();

**if** (maxIndices > 0) {

byteBuffer = ByteBuffer.*allocateDirect*(maxIndices \* Short.*SIZE* / 8);

byteBuffer.order(ByteOrder.*nativeOrder*());

indices = byteBuffer.asShortBuffer();

} **else** {

indices = **null**;

}

}

**public** **void** setVertices(**float**[] vertices, **int** offset, **int** length) {

**this**.bufVertices = vertices;

**this**.vertices.clear();

**this**.vertices.put(vertices, offset, length);

**this**.vertices.flip();

}

**public** **void** setIndices(**short**[] indices, **int** offset, **int** length) {

**this**.indices.clear();

**this**.indices.put(indices, offset, length);

**this**.indices.flip();

}

**public** **void** bind() {

GL10 gl = glGraphics.getGL();

gl.glEnableClientState(GL10.*GL\_VERTEX\_ARRAY*);

vertices.position(0);

gl.glVertexPointer(vec, GL10.*GL\_FLOAT*, vertexSize, vertices);

**if** (hasColor) {

gl.glEnableClientState(GL10.*GL\_COLOR\_ARRAY*);

vertices.position(2);

gl.glColorPointer(4, GL10.*GL\_FLOAT*, vertexSize, vertices);

}

**if** (hasTexCoords) {

gl.glEnableClientState(GL10.*GL\_TEXTURE\_COORD\_ARRAY*);

vertices.position((hasColor ? 6 : 2));

gl.glTexCoordPointer(2, GL10.*GL\_FLOAT*, vertexSize, vertices);

}

}

**public** **void** draw(**int** primitiveType, **int** offset, **int** numVertices) {

GL10 gl = glGraphics.getGL();

**if** (indices != **null**) {

indices.position(offset);

gl.glDrawElements(primitiveType, numVertices,

GL10.*GL\_UNSIGNED\_SHORT*, indices);

} **else** {

gl.glDrawArrays(primitiveType, offset, numVertices);

}

}

**public** **void** unbind() {

GL10 gl = glGraphics.getGL();

**if** (hasTexCoords) {

gl.glDisableClientState(GL10.*GL\_TEXTURE\_COORD\_ARRAY*);

}

**if** (hasColor) {

gl.glDisableClientState(GL10.*GL\_COLOR\_ARRAY*);

}

}

@Override

**public** String toString() {

StringBuilder builder = **new** StringBuilder();

**for** (**int** i = 0; i < vertices.capacity(); i++) {

builder.append("" + vertices.get(i) + " ");

}

**return** builder.toString();

}

**public** **float**[] getVerticesA() {

**return** bufVertices;

}

}

**Висновок.**

Дана лабораторна робота була виконана під операційну систему Android. Використовувалося стандартне API, яке дозволило реалізувати поставлену задачу. Для відображення фігури було використано OpenGLES 1.0. Дане API дозволяє виконувати поворот фігур, але так як метою лабораторної роботи є вивчення афінних перетворень, то було реалізовано власний поворот фігури і її збільшення за допомогою матриці перетворень.