Programowanie i projektowanie obiektowe

Grafika 3D

Paweł Daniluk

Wydział Fizyki

Jesień 2011

JOGL – Java OpenGL

Instalacja

1. Pobrać plik `jogamp-linux-amd64.tar.gz`
2. Rozpakować
3. Dołączyć wszystkie pliki JAR do projektu (Libraries(RMB)>Add JAR/Folder)
4. We właściwościach projektu ustawić opcję VM options (kategoria Run) na:
   `-Djava.library.path=/<ścieżka>/jogamp-linux-amd64/lib`

Prosty przykład 2D

Interfejs GLEventListener

```
public class Triangle implements GLEventListener {

    @Override
    public void display(GLAutoDrawable drawable) {
        update();
        render(drawable);
    }

    @Override
    public void dispose(GLAutoDrawable drawable) {
    }

    @Override
    public void init(GLAutoDrawable drawable) {
    }

    @Override
    public void reshape(GLAutoDrawable drawable, int x, int y, int w, int h) {
    }
}
```

Prosty przykład 2D c.d.

Rysujemy trójkąt

```
private double theta = 0;
private double s = 0;
private double c = 0;

private void update() {
    theta += 0.01;
    s = Math.sin(theta);
    c = Math.cos(theta);
}

private void render(GLAutoDrawable drawable) {
    GL2 gl = drawable.getGL().getGL2();

    gl.glClear(GL.GL_COLOR_BUFFER_BIT);

    // draw a triangle filling the window
    gl.glBegin(GL.GL_TRIANGLES);
    gl.glColor3f(1, 0, 0);
    gl.glVertex2d(-c, -c);
    gl.glColor3f(0, 1, 0);
    gl.glVertex2d(0, c);
    gl.glColor3f(0, 0, 1);
    gl.glVertex2d(s, -s);
    gl.glEnd();
}
```

Prosty przykład 2D c.d.

Rozruch

```
public static void main(String[] args) {
    GLProfile glp = GLProfile.getDefault();
    GLCapabilities caps = new GLCapabilities(glp);
    GLCanvas canvas = new GLCanvas(caps);

    Frame frame = new Frame("Triangle");
    frame.setSize(300, 300);
    frame.add(canvas);
    frame.setVisible(true);

    frame.addWindowListener(new WindowAdapter() {

        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });

    canvas.addGLEventListener(new Triangle());

    FPSAnimator animator = new FPSAnimator(canvas,60);
    animator.add(canvas);
    animator.start();
}
```

Kostka w 3D

```
private GLU glu;
private GLUT glut;

private float r=3;
private float a = 0;

@Override
public void display(GLAutoDrawable drawable) {
    update();
    render(drawable);
}

@Override
public void init(GLAutoDrawable drawable) {
    glu=new GLU();
    glut=new GLUT();
}

private void update() {
    a = (a + (float) Math.PI / 300f);
}
```

Kostka w 3D c.d.

```
 private void render(GLAutoDrawable drawable) {
    GL2 gl = drawable.getGL().getGL2();

    int width = drawable.getWidth();
    int height = drawable.getHeight();

    gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);

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

    gl.glMatrixMode(GL2.GL_PROJECTION);
    gl.glLoadIdentity();

    camera(drawable, gl);
    lights(drawable, gl);

    gl.glMatrixMode(GL2.GL_MODELVIEW);
    gl.glLoadIdentity();

    scene(drawable,gl);
}
```

Kostka w 3D c.d.

```
private void camera(GLAutoDrawable drawable, GL2 gl) {
    int width = drawable.getWidth();
    int height = drawable.getHeight();

    glu.gluPerspective(70.0, (float) width / (float) height, 1, 50);
    glu.gluLookAt(r * Math.sin(a), 0, r * Math.cos(a), 0, 0, 0, 0.0, 1.0, 0.0);
}
```

```
private void scene(GLAutoDrawable drawable, GL2 gl) {
    gl.glEnable(GL2.GL_COLOR_MATERIAL);
    glut.glutSolidCube(1);
}
```

Kostka w 3D c.d.

```
private void lights(GLAutoDrawable drawable, GL2 gl) {
    float light_ambient[] = {0.2f, 0.2f, 0.2f, 1.0f};
    float light_diffuse[] = {0.8f, 0.8f, 0.8f, 1.0f};
    float light_specular[] = {0.5f, 0.5f, 0.5f, 1.0f};
    float light0_position[] = {-4.0f, 4.0f, -2.0f, 0.0f};

    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_AMBIENT, light_ambient, 0);
    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_DIFFUSE, light_diffuse, 0);
    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_SPECULAR, light_specular, 0);
    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_POSITION, light0_position, 0);

    gl.glEnable(GL2.GL_LIGHTING);
    gl.glEnable(GL2.GL_LIGHT0);

    gl.glShadeModel(GL2.GL_SMOOTH);
    gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    gl.glClearDepth(1.0f);
    gl.glEnable(GL.GL_DEPTH_TEST);
    gl.glDepthFunc(GL.GL_LEQUAL);
    gl.glHint(GL2.GL_PERSPECTIVE_CORRECTION_HINT, GL.GL_NICEST);
}
```

Interaktywna kamera

```
public class CubeInt implements GLEventListener, KeyListener {
    public static void main(String[] args) {

        ...

        canvas.addKeyListener(cube);

        ...

    }
```

Interaktywna kamera c.d.

```
private float r = 3;
private float a = 0;
private float b = 0;
private boolean keys[] = new boolean[256];

@Override
public void keyTyped(KeyEvent ke) {
}

@Override
public void keyPressed(KeyEvent ke) {
    try {
        char i = ke.getKeyChar();
        keys[(int) i] = true;
    } catch (Exception e) {
    };
}

@Override
public void keyReleased(KeyEvent ke) {
    try {
        char i = ke.getKeyChar();
        keys[(int) i] = false;
    } catch (Exception e) {
    };
}
```

Interaktywna kamera c.d.

```
private void update() {
    keyboardChecks();
}

public void keyboardChecks() {
    if (keys['q']) {
        b += Math.PI / 360;
    }

    if (keys['a']) {
        b -= Math.PI / 360;
    }

    if (keys['o']) {
        a -= Math.PI / 360;
    }

    if (keys['p']) {
        a += Math.PI / 360;
    }
}

private void camera(GLAutoDrawable drawable, GL2 gl) {
    int width = drawable.getWidth();
    int height = drawable.getHeight();

    glu.gluPerspective(70.0, (float) width / (float) height, 1, 50);
    glu.gluLookAt(r * Math.sin(a) * Math.cos(b), r * Math.sin(b), r * Math.cos(a) * Math.cos(b),
        0, 0, 0, 0.0, 1.0, 0.0);
}
```

Animacja 3D

```
private void scene(GLAutoDrawable drawable, GL2 gl) {
    float[] colorYellow = {1.0f, 1.0f, 0.0f};
    float[] colorWhite = {1.0f, 1.0f, 1.0f};
    float[] colorBlack = {0f, 0f, 0f, 0f};
    float[] colorBlue = {0f, 0.5f, 1.0f, 1f};

    gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_AMBIENT, colorYellow,0);
    gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_DIFFUSE, colorYellow, 0);
    gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_SPECULAR, colorWhite, 0);
    gl.glMateriali(GL.GL_FRONT_AND_BACK, GL2.GL_SHININESS, 4);
    gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_EMISSION, colorYellow, 0);

    gl.glColor3f(1f, 1f, 0f);
    glut.glutSolidSphere(1.0, 24, 24);

    gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_AMBIENT, colorBlack, 0);
    gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_DIFFUSE, colorBlue, 0);
    gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_EMISSION, colorBlack, 0);

    gl.glTranslated(5f * Math.sin(e), 0f, 5f * Math.cos(e));
    glut.glutSolidSphere(0.3, 24, 24);

    gl.glMaterialfv(GL.GL_FRONT_AND_BACK, GL2.GL_DIFFUSE, colorWhite, 0);

    gl.glTranslated(1f * Math.sin(e*12), 0f, 1f * Math.cos(e*12));
    glut.glutSolidSphere(0.1, 24, 24);
}
```

Animacja 3D c.d.

```
private void lights(GLAutoDrawable drawable, GL2 gl) {
    float light_ambient[] = {0.3f, 0.3f, 0.3f,1f};
    float light_diffuse[] = {1f, 1f, 1f,1f};
    float light_specular[] = {0.5f, 0.5f, 0.5f};
    float light0_position[] = {0f, 0f, 0f, 1f};


    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_AMBIENT, light_ambient, 0);
    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_DIFFUSE, light_diffuse, 0);
    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_SPECULAR, light_specular, 0);
    gl.glLightfv(GL2.GL_LIGHT0, GL2.GL_POSITION, light0_position, 0);

    gl.glEnable(GL2.GL_LIGHTING);
    gl.glEnable(GL2.GL_LIGHT0);

    gl.glShadeModel(GL2.GL_SMOOTH);
    gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    gl.glClearDepth(1.0f);
    gl.glEnable(GL.GL_DEPTH_TEST);
    gl.glDepthFunc(GL.GL_LEQUAL);
    gl.glHint(GL2.GL_PERSPECTIVE_CORRECTION_HINT, GL.GL_NICEST);
}
```

Animacja 3D c.d.

```
private void update() {
    keyboardChecks();

    if (running) {
        e += Math.PI / 2160;
        if (e > Math.PI * 2) {
            e -= Math.PI * 2;
        }
    }
}

private void render(GLAutoDrawable drawable) {
    GL2 gl = drawable.getGL().getGL2();

    int width = drawable.getWidth();
    int height = drawable.getHeight();

    gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);

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

    gl.glMatrixMode(GL2.GL_PROJECTION);
    gl.glLoadIdentity();

    gl.glMatrixMode(GL2.GL_MODELVIEW);
    gl.glLoadIdentity();

    camera(drawable, gl);
    lights(drawable, gl);
    scene(drawable, gl);
}
```