GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Prof Dr. Winfried Kurth,
Alex Tavkhelidze

# Praktikum Computergrafik, WiSe 17/18
# Übungsblatt 3

- ✓ **Abgabefrist**: 18.12.2017 09:00:00
- ✓ Abgabe erfolgt per E-Mail an jeos@mail.com
- ✓ **Betreff**: CG17WS ÜB3
- ✓ **Erste Zeilen der E-Mail**: Name der Autoren und Matrikelnummern.
- ✓ Der **lauffähige Code** soll **als Anhang** in der E-Mail mitgeschickt werden.
- ✓ Der Quelltext muss dabei mit Eclipse in **ein ZIP-Archiv** exportiert worden sein
  (*siehe die Anleitung in Folien zum Übungsblatt 1*)
  Bemerkung:
  Für jede programmierbezogene Aufgabe muss eine separate ZIP-Datei exportiert & beigefügt werden.

Quellen

**Aufgaben** http://www.uni-forst.gwdg.de/~wkurth/cg17_u03.pdf

**Vorlage** *siehe die Anlage zu diesem Übungsblatt oder die mitgeschickte Datei **0.txt** (für bessere Lesbarkeit in **Eclipse** oder in **Notepad++** mit abgehackter **Spracherkennung -> Java** öffnen)*

**Liste der Aufgaben**:

1. Mithilfe der Vorlage, erstellen Sie den Code, der die folgendermaßen beschriebene Szene aufbaut:
   - Quadratmaschenboden, der in der ganzen Szene verlegt wird (also die Szene soll mit dem Bodenhorizont rundum versehen sein);
   - Gleich (oder ein wenig gehoben, allerdings mit unerkennbarer Spalte) auf dem Boden platzierter Würfel, mit einfärbig und dabei unterschiedlich gefärbten Flächen;
   - Gleichmäßige Bewegung (d.h. mit einem konstanten Schritt) des Betrachters in 6 Richtungen durch die Szene mit den entsprechenden Tastaturtasten – *siehe Details in der **Vorlage*** (in Kommentarzeilen);
   - Gleichmäßige 360°-Rotation der Blickrichtung in 4 Richtungen – steuerbar durch die entsprechenden Tastaturtasten (*siehe Details in der **Vorlage,*** in Kommentarzeilen).

2. Ändern Sie die Szene in der Aufgabe #1 so, dass statt eines bewegungslosen Würfels ein gleichmäßig drehender Würfel in der Szene erscheint (*der vom Boden etwas abgehoben ist, der mit einfärbig und unterschiedlich gefärbten Flächen versehen ist und der so rotiert, dass alle Flächen in Sicht kommen*).
   Dabei ist zu beachten, dass die Rotation dieses Würfels durch den entsprechenden Tastendruck **glattgehend** (*d.h. ohne sichtbaren Sprung in Rotation*) angehalten oder (wenn schon angehalten) auch **leichtgängig** weitergelaufen werden soll.

3. Ergänzen Sie den Code in der Aufgabe #2, sodass der angehaltene (**und nicht der drehende!**) Würfel, mit den entsprechenden Tastaturtasten gesteuert, in 4 Richtungen drehen wird - *siehe Details in der **Vorlage,*** in Kommentarzeilen.

**Anlage**: [verwenden Sie den unten gegebenen Coderahmen für die Aufgabenlösung und beachten Sie dabei vor allem die Kommentare auf Deutsch – u.a. die Tabelle für die zugeordnete Tastenkontrolle]

```
package cg;

import static javax.swing.JFrame.EXIT_ON_CLOSE;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

import javax.swing.JFrame;

import com.jogamp.opengl.GL2;
import com.jogamp.opengl.GLAutoDrawable;
import com.jogamp.opengl.GLEventListener;
import com.jogamp.opengl.awt.GLCanvas;
import com.jogamp.opengl.glu.GLU;
import com.jogamp.opengl.util.FPSAnimator;

/*
Tastenkontrolle:

********** Kamera/Auge **********:

[c] - setzt die Kamera komplett (d.h. ihre Richtung und Position) zurück

[Pfeil oben]   - Bewegung vorwärts
[Pfeil unten]  - Bewegung rückwärts
[Pfeil links]  - Bewegung links
[Pfeil rechts] - Bewegung rechts
[Bild auf]     - Bewegung oben
[Bild ab]      - Bewegung unten

[a] - Rotation nach links
[d] - Rotation nach rechts
[w] - Rotation nach oben
[s] - Rotation nach unten

********** Objekte in der Szene **********:

[r] - sperrt die Rotation eines Objekts in der Szene

[f] - Rotation nach links (ein wenig schräg ist OK)
```

```
[h] - Rotation nach rechts (ein wenig schräg ist OK)
[t] - Rotation nach oben (ein wenig schräg ist OK)
[g] - Rotation nach unten (ein wenig schräg ist OK)
*/

public class Main implements GLEventListener, KeyListener  {

   // create an instance of GL Utility library and keep it
   final GLU glu = new GLU();

   // count the frames that were rendered
   int frames = 1;

   // für die Steuerung der Kameraposition (x, y, z Koordinaten)
   /**
    * Beachten Sie warum als statisch deklariert
    */
   static float camera_position[] = {0.0f, 0.0f, -10.0f};

   // für die Steuerung der Kamerarotation (um x, y, z Achsen)
   /**
    * Beachten Sie warum als statisch deklariert
    */
   static float camera_orientation[] = {0.0f, 0.0f, 0.0f};

   // Deklarierung einer Variable für die System.currentTimeMillis() Methode,
   // die die aktuelle Zeit in Millisekunden zurückgibt - falls nötig (für die Aufgabenlösung)
   static long timerstart=0;

   // speed for camera movement
   float speed = 0.2f;

   // camera angle increment in degrees
   float angle_incr = 1.0f;

   public static void main(String[] args) {
      // create OpenGL window
      GLCanvas canvas = new GLCanvas();
      canvas.addGLEventListener(new Main());
      canvas.addKeyListener(new Main());

      // animate the canvas with 30 frames per second
      FPSAnimator animator = new FPSAnimator(canvas, 30);
      animator.start();
```

```
    final int width = 800;
    final int height = 600;

    // create main window and insert the canvas into it
    JFrame frame = new JFrame("OpenGL Fenster");
    frame.setSize(width, height);
    frame.setDefaultCloseOperation(EXIT_ON_CLOSE);
    frame.getContentPane().add(canvas);

    // show the main window
    frame.setVisible(true);

    // set input focus to the canvas
    canvas.setFocusable(true);
    canvas.requestFocus();
}

public void init(GLAutoDrawable drawable) {
    GL2 gl = drawable.getGL().getGL2();

    // modify initial state for the GLCanvas

    // background color
    gl.glClearColor(0.0f, 0.749f, 1.0f, 1.0f);

    // enable backface culling (default is off)
    gl.glEnable(GL2.GL_CULL_FACE);

    // enable z-buffer (default is off)
    gl.glEnable(GL2.GL_DEPTH_TEST);
}

public void display(GLAutoDrawable drawable) {
    // obtain GL to draw to the framebuffer
    GL2 gl = drawable.getGL().getGL2();

    //gl.glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //when set, overrides the value set in init method

    frames++;

    // clear the framebuffer
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);

    gl.glLoadIdentity();
```

```
        /**
         * An dieser Stelle sind die Transformationen mit den oben deklarierten statischen Variablen zu
verwenden,
         * damit man mithilfe der KeyPress Methode (siehe unten) die Kamera (das Auge) in der Szene
         * mit den entsprechenden Tasten (siehe Kommentar-Tabelle oben) bewegt und rotiert.
         */

        // Transformationen und Zeichnen von Objekten
        // Objekt #1
        // Objekt #2
        // ...
        /**
         * Objekten könnten (und es wird sogar streng empfohlen) als separate Methoden
         * (zB mit dem Argument "GL2 gl") gezeichnet werden und in display-Methode
         * einfach an der passenden Stellen aufgerufen werden.
         */
    }
    /**
     * When the window changes its position or shape the projection transformation
     * and viewport dimensions have to be adjusted accordingly.
     */
    public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) {
        GL2 gl = drawable.getGL().getGL2();

        // set viewport to window dimensions
        gl.glViewport(0, 0, width, height);

        gl.glMatrixMode(GL2.GL_PROJECTION);
        gl.glLoadIdentity();
        // set perspective projection
        glu.gluPerspective(45.0f, (float) width / (float) height, 1.0f, 100.0f);

        gl.glMatrixMode(GL2.GL_MODELVIEW);
        gl.glLoadIdentity();
        /**
         * setzt die XYZ-Koordinaten von:
         * Kameraposition (erster Dreier),
         * Blickrichtung (nächster Dreier) und
         * Up-Vektor (letzter Dreier) - zB ...,0,-1,0) entspricht der "auf den Kopf gestellt" Lage
         */
        glu.gluLookAt(0.0f, 0.0f, 5.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);
    }
```

```
public void displayChanged(GLAutoDrawable arg0, boolean arg1, boolean arg2) {
   // TODO Auto-generated method stub
}

@Override
public void keyPressed(KeyEvent e) {
   int keyCode = e.getKeyCode();

   if (keyCode == KeyEvent.VK_UP) {
      //Code zum [Pfeil-oben] Tastendruck
   }

   if (keyCode == KeyEvent.VK_DOWN) {
      //Code zum [Pfeil-unten] Tastendruck
   }

   else if (keyCode == KeyEvent.VK_LEFT) {
      //Code zum [Pfeil-links] Tastendruck
   }

   else if (keyCode == KeyEvent.VK_RIGHT) {
      //Code zum [Pfeil-rechts] Tastendruck
   }

   else if (keyCode == KeyEvent.VK_PAGE_UP) {
      //Code zum [Bild auf] Tastendruck
   }

   else if (keyCode == KeyEvent.VK_PAGE_DOWN) {
      //Code zum [Bild ab] Tastendruck
   }

   else if (keyCode == KeyEvent.VK_A) {
      //Code zum [a] Tastendruck
   }

   // turn right
   else if (keyCode == KeyEvent.VK_D) {
      //Code zum [d] Tastendruck
   }

   // turn up
   else if (keyCode == KeyEvent.VK_W) {
      //Code zum [w] Tastendruck
```

```
    }

    // turn down
    else if (keyCode == KeyEvent.VK_S) {
      //Code zum [s] Tastendruck
    }

    else if (keyCode == KeyEvent.VK_C) {
      //Code zum [c] Tastendruck
    }

    else if (keyCode == KeyEvent.VK_R) {
      //Code zum [r] Tastendruck
    }

    else if (keyCode == KeyEvent.VK_T) {
      //Code zum [t] Tastendruck
    }

    else if (keyCode == KeyEvent.VK_H) {
      //Code zum [h] Tastendruck
    }

    else if (keyCode == KeyEvent.VK_F) {
      //Code zum [f] Tastendruck
    }

    else if (keyCode == KeyEvent.VK_G) {
      //Code zum [g] Tastendruck
    }

    else if (keyCode == KeyEvent.VK_SPACE) {
      //Code zum [Leertaste] Tastendruck
    }
  }

  @Override
  public void keyReleased(KeyEvent e) {
    // TODO Auto-generated method stub
  }

  @Override
  public void keyTyped(KeyEvent arg0) {
    // TODO Auto-generated method stub
```

```
    }

    @Override
    public void dispose(GLAutoDrawable drawable) {
        // TODO Auto-generated method stub
    }
}
```