WiSe 20/21

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Prof. Dr. Winfried Kurth
Alex Tavkhelidze

Praktikum Computergrafik

Folien zu #3

Eingabesteuerung, Wiedergabe von Multimediaformaten, komplexere Szene

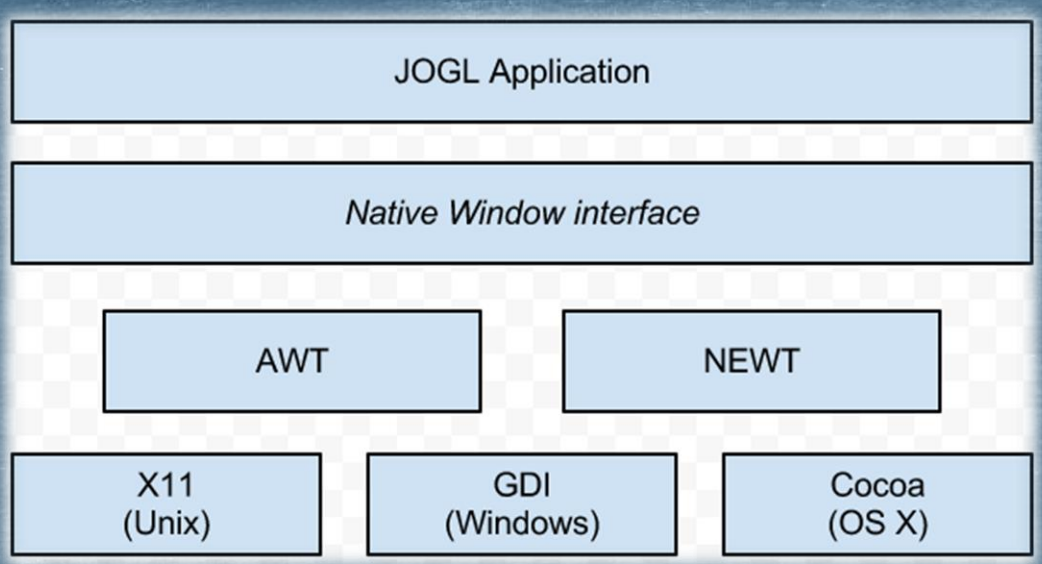
Tastatur

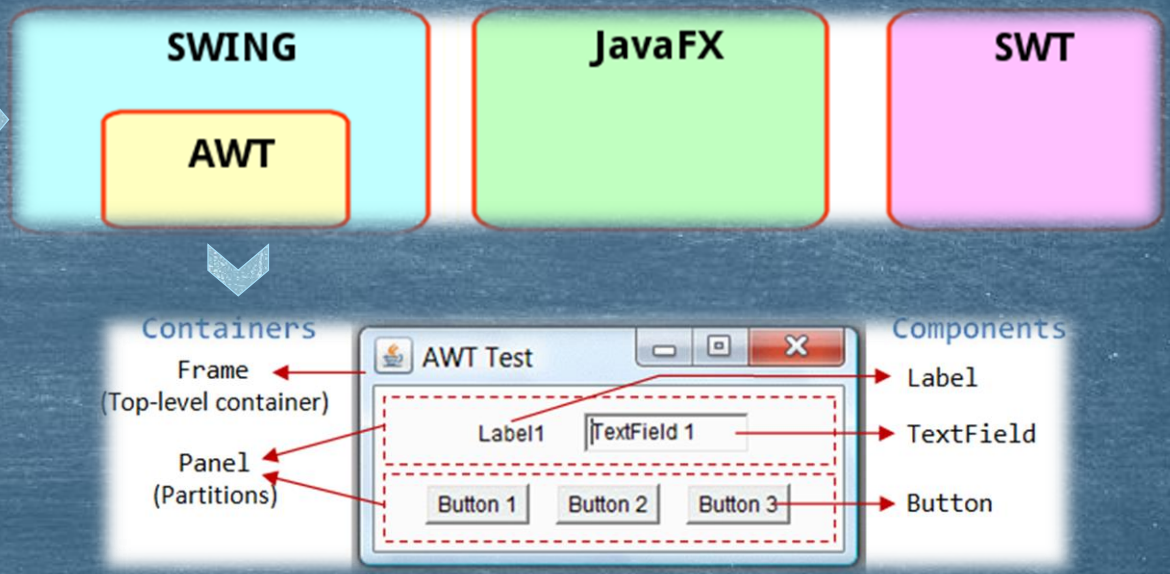
Maus

Audio (WAV)

V(**A**+**B**+**I**)**O**bjekte

2D TextRenderer





Java GUI Frameworks

Maus- & Tastatursteuerung

Bildquelle: http://quarkphysics.ca/ICS4U1/unit4-graphics/graphics1.html

Steuerung über Tastatur

- Die Liste aller Funktionen und Tastencodes der Klasse **KeyEvent** entnehmt ihr den **Oracle API-Spezifikationen** hier
- Die Methode keyPressed steuert die **Betätigung** (aka das **Drücken**) jeder Taste
- Die Methode keyReleased steuert das **Lösen** (aka das **Loslassen**) jeder Taste
- Die Methode key**Typed** steuert die **Betätigung** jeder **Zeichen**taste (d.h. Buchstaben, Symbole, Ziffern) – also sobald was gedruckt wird

```
package cg;
//-------------------------------------------------------
...
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
...
//-------------------------------------------------------
public class Main implements KeyListener {
        ...
        public static void main(String[] args) {
            ...
            canvas.addKeyListener(new Main());

            canvas.setFocusable(true);
            ...
        }
//-------------------------------------------------------
                public void keyTyped(KeyEvent e) {
                    int keyCo = e.getKeyCode();
                    if (keyCo == KeyEvent.VK_A) {...}
                    ...
                    }
//-------------------------------------------------------
                public void keyPressed(KeyEvent e) {...}
//-------------------------------------------------------
                public void keyReleased(KeyEvent e) {...}
//-------------------------------------------------------

}
```

Steuerung über Maus (1)

- Die Liste aller Funktionen und Tastencodes der Klasse **MouseEvent** entnehmt ihr den **Oracle API-Spezifikationen** hier
- Die Methode mouseClicked vereint in sich die Ereignisse „**Drücken**“ und „**Lösen**“ einer Maustaste (oder des Mausrades)
- Die Methoden mouseEntered bzw. mouseExited werden aufgerufen sobald der Mauszeiger die Fläche des „abgehörten“ **GUI-Elementes** betritt bzw. verlässt
- Fürs Drehen des Mausrads sind die extra Klassen MouseWheelListener und MouseWheelEvent zuständig

```
package cg;
//-------------------------------------------------------------------------
import javax.swing.JFrame;
//...
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
//...
//-------------------------------------------------------------------------
public class Main implements MouseListener {
	//...
	public static void main(String[] args) {

		JFrame frame = new JFrame("Fenster");
		final int width = 800;
		final int height = 600;
		frame.setSize(width, height);
		//...
		frame.addMouseListener(new Main());
		//...
		frame.setFocusable(true);
		//...
		frame.setVisible(true);
	}
//-------------------------------------------------------------------------
	public void mouseClicked(MouseEvent e) {/*...*/}
//-------------------------------------------------------------------------
	public void mousePressed(MouseEvent e) {System.out.println(e.getClickCount());}
//-------------------------------------------------------------------------
	public void mouseReleased(MouseEvent e) {/*...*/}
//-------------------------------------------------------------------------
	public void mouseEntered(MouseEvent e) {System.out.println(e.getXOnScreen()+ "/ " + e.getX());}
//-------------------------------------------------------------------------
	public void mouseExited(MouseEvent e) {System.out.println(e.getPoint());}
}
```

Steuerung über Maus (2)

- Die Methoden mouseDragged bzw. mouseMoved werden aufgerufen sobald der Mauszeiger über die Fläche des „abgehörten“ **GUI-Elementes** bei (einer bzw. keiner) gedrückten Maustaste bewegt wird

- MouseEvent.BUTTON**3**_MASK fängt idR jedes Drücken der **rechten** Maustaste (BUTTON**1**_MASK wäre für die **linke** Maustaste, BUTTON**2**_MASK – für das Maus**rad**)

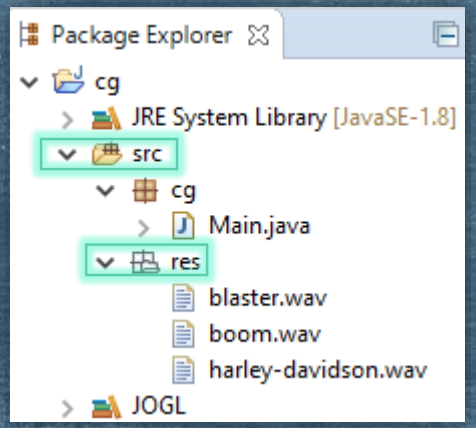
```
package cg;
//-------------------------------------------------------------------------------
import javax.swing.JFrame;
import com.jogamp.opengl.awt.GLCanvas;
//...
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
//...
//-------------------------------------------------------------------------------
public class Main implements MouseMotionListener {
	//...
	public static void main(String[] args) {

		JFrame frame = new JFrame("OpenGL Fenster");
		final int width = 800;
		final int height = 600;
		frame.setSize(width, height);
		//...
		GLCanvas canvas = new GLCanvas();
		frame.add(canvas);//alternativ: frame.getContentPane().add(canvas);
		canvas.addMouseMotionListener(new Main());
		//...
		//setzt den Tastaturfokus auf eine GUI-Komponente (hier: auf die GL-Leinwand)
		//canvas.setFocusable(true);//(bedingt) alternativ: frame.setFocusable(true);
		//...
		frame.setVisible(true);
	}
//-------------------------------------------------------------------------------
	public void mouseDragged(MouseEvent e) {
		if (e.getModifiers() == MouseEvent.BUTTON3_MASK) {System.out.println(e.getWhen());}
		}
//-------------------------------------------------------------------------------
	public void mouseMoved(MouseEvent e) {System.out.println(e.isShiftDown());}
}
```

alternative Eingabesteuerung (NEWT)

- NEWT weist idR eine höhere Leistung (im Vergleich zu AWT/Swing) auf
- NEWT kann mit AWT/Swing gekoppelt werden
- Oberster GUI-Container ist bereits ein „OpenGL-fähiges“ GLWindow-Fenster

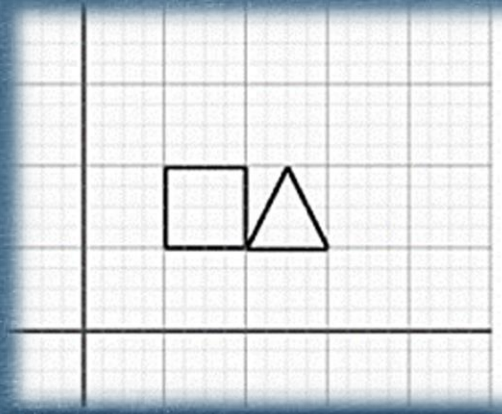
```
package cg;
//-----------------------------------------------------------------------------
import com.jogamp.opengl.GLProfile; import com.jogamp.opengl.GLCapabilities;
import com.jogamp.newt.opengl.GLWindow;
import com.jogamp.opengl.util.FPSAnimator;
import com.jogamp.newt.event.KeyEvent; import com.jogamp.newt.event.KeyListener;
import com.jogamp.newt.event.MouseEvent; import com.jogamp.newt.event.MouseListener;
//-----------------------------------------------------------------------------
public class Main implements KeyListener, MouseListener {
    //...
//-----------------------------------------------------------------------------
public static void main(String[] args) {
        //...
        GLProfile glp = GLProfile.getDefault();
        GLCapabilities caps = new GLCapabilities(glp);

        GLWindow window = GLWindow.create(caps);
        window.setSize(640, 480);
        window.setTitle("NEWT Fenster");
        window.setVisible(true);

        Main mn = new Main(); window.addKeyListener(mn); window.addMouseListener(mn);
        //...
        final FPSAnimator animator = new FPSAnimator(window, 60, true); animator.start();
    }
//-----------------------------------------------------------------------------
    public void keyPressed(KeyEvent e) {System.out.println(e.getKeyChar());}
    public void keyReleased(KeyEvent e) {/*...*/}
//-----------------------------------------------------------------------------
    public void mouseClicked(MouseEvent e) {System.out.println(e.getClickCount());}
    public void mouseEntered(MouseEvent e) {System.out.println(e.getWhen());}
    public void mouseExited(MouseEvent e) {System.out.println(e.getWhen());}
    public void mousePressed(MouseEvent e) {/*...*/}
    public void mouseReleased(MouseEvent e) {/*...*/}
    public void mouseMoved(MouseEvent e) {/*...*/}
    public void mouseDragged(MouseEvent e) {System.out.println(e.getEventType());}
    public void mouseWheelMoved(MouseEvent e) {System.out.println(e.getRotation()[1]);}
}
```

Sound abspielen & stoppen



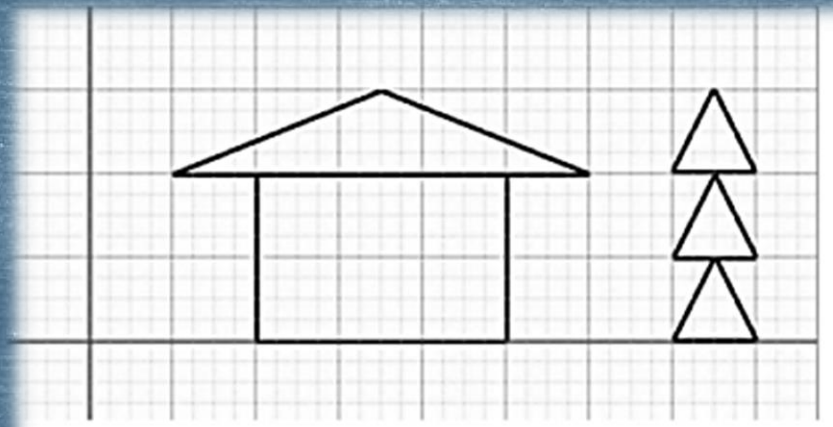
- Java Sound unterstützt die Tondateien mit folgenden Eigenschaften:
  - Dateiformat: AU, AIFF, WAV
    - unkomprimiertes Audio
  - Samplingtiefe: 8-16 Bits
  - Abtastrate: 8-48 kHz
- die Unterstützung von **3D-Sounds** (das heißt, mit Tonabschwächung, Doppler-Effekt und gerichteten Sounds) wird erst mit extra **OpenAL** Java-Paketen mitgeliefert

```
package cg;
//-------------------------------------------------------------------------
...
import java.io.File;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
...
//-------------------------------------------------------------------------
public class Main implements KeyListener {
     static Clip clip;
     static String[] datei = {"blaster.wav","boom.wav","harley-davidson.wav"};
//-------------------------------------------------------------------------
   public static void main(String[] args) {...}
//-------------------------------------------------------------------------
   public static void soundStart(String datei) {
        try {
              File file = new File("src/res/" + datei);
              AudioInputStream audioIn = AudioSystem.getAudioInputStream(file);
              clip = AudioSystem.getClip();
              clip.open(audioIn);
              clip.start();
         } catch (Exception e) {e.printStackTrace();}
     }
   public static void soundStop() {clip.stop();}
//-------------------------------------------------------------------------
     public void keyPressed(KeyEvent e) {
           int keyCode = e.getKeyCode();
           if(keyCode == KeyEvent.VK_A) {soundStart(datei[0]);}
           if(keyCode == KeyEvent.VK_S) {soundStart(datei[1]);}
           if(keyCode == KeyEvent.VK_D) {soundStart(datei[2]);}
           if(keyCode == KeyEvent.VK_SPACE) {soundStop();}
           }
     ...
}
```



Sample:
translate(1,1)
draw(square)
translate(1,0)
draw(triangle)

Bausteine der Szene



1. translate(2,0)
2. scale(3,2)
3. draw(square)
4. scale(1/3,1/2)
5. translate(5,0)
6. draw(triangle)
7. translate(0,1)
8. draw(triangle)
9. translate(0,1)
10. draw(triangle)
11. translate(-6,0)
12. scale(5,1)
13. draw(triangle)

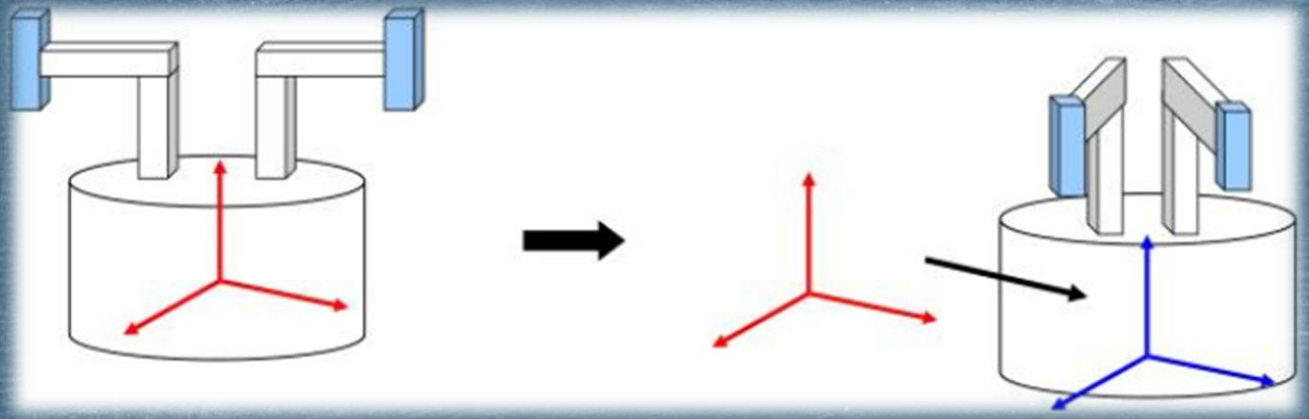
die Szene und deren Aufbau

komplexere Szene

hierarchische Modelle

Bildquelle: https://media.cheggcdn.com/media/e5c/e5cb7639-d176-4ce5-a0e5-d8617d81cb2b/phpVWgnJm.png

hierarchische Modelle



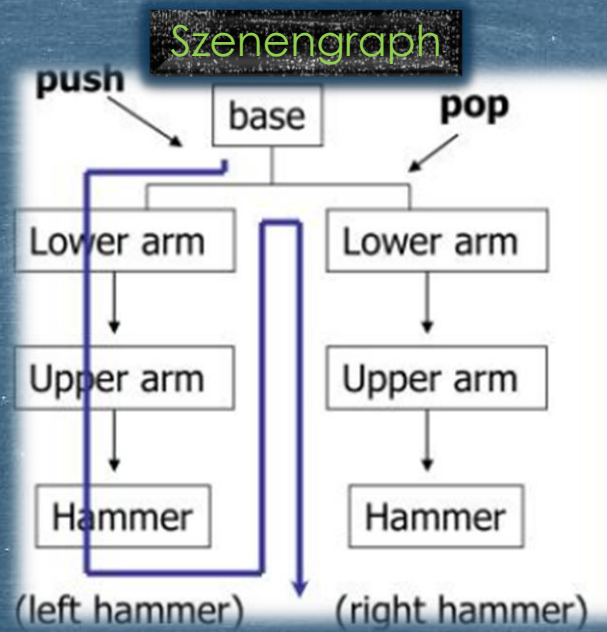
- Das ganze **Konstrukt** besteht aus:
  - einem zylinderförmigen Sockel
    - einem linken Arm – dieser besteht aus:
      - einem quaderförmigen Oberarm
      - einem quaderförmigen Unterarm
      - einem quaderförmigen Hammerkopf
    - einem rechten Arm – dieser besteht aus:
      - einem quaderförmigen Oberarm
      - einem quaderförmigen Unterarm
      - einem quaderförmigen Hammerkopf

Ziel #1:

- Das Konstrukt
  - zu **verschieben**
    - nach rechts
      - um 5 Einheiten

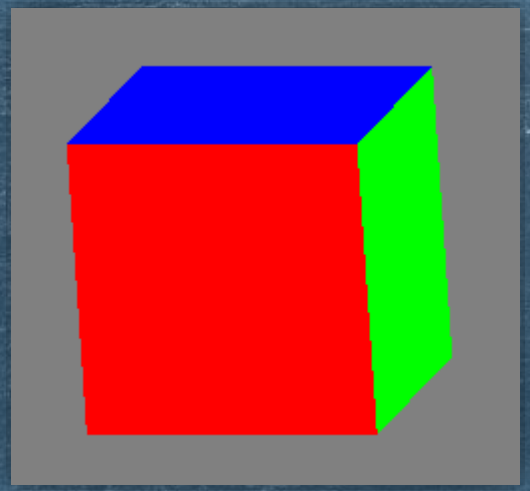
Ziel #2:

- Die Arme
  - zu **drehen**
    - um die Y-Achse
      - um 75 Grad
        - in unterschiedliche Richtungen




```
glTranslate(5,0,0)
Draw_base();
glPushMatrix();

glRotate(75, 0,1,0);
Draw_left_hammer();

glPopMatrix();
glRotate(-75, 0,1,0);
Draw_right_hammer();
```

Vertex Arrays: effizientes Zeichnen von Primitiven



```
package cg;
//-------------------------------------------------------------------------------
...
import com.jogamp.common.nio.Buffers;
import java.nio.FloatBuffer;
//-------------------------------------------------------------------------------
public class Main extends GLJPanel implements GLEventListener, KeyListener {
    ...
    //---------------------------------------------------------------------------
    public static void main(String[] args) {...}
    ...
    //---------------------------------------------------------------------------
                                    //     v6----- v5
                                    //    /|      /|
                                    //   v1------v0|
                                    //   | |     | |
                                    //   | v7----|-v4
                                    //   |/      |/
                                    //   v2------v3
    private float[] cubeCoords = {                      // äußere Seiten im Gegenuhrzeigersinn beschrieben
            1,1,1,    -1,1,1,   -1,-1,1,   1,-1,1,      // v00[v0]-v01[v1]-v02[v2]-v03[v3] - vordere Fläche
            1,1,1,     1,-1,1,   1,-1,-1,  1,1,-1,      // v04[v0]-v05[v3]-v06[v4]-v07[v5] - rechte Fläche
            1,1,1,     1,1,-1,  -1,1,-1,  -1,1,1,       // v08[v0]-v09[v5]-v10[v6]-v11[v1] - obere Fläche
            -1,-1,-1, -1,1,-1,   1,1,-1,   1,-1,-1,     // v12[v7]-v13[v6]-v14[v5]-v15[v4] - hintere Fläche
            -1,-1,-1, -1,-1,1,  -1,1,1,   -1,1,-1,      // v16[v7]-v17[v2]-v18[v1]-v19[v6] - linke Fläche
            -1,-1,-1,  1,-1,-1,  1,-1,1,   -1,-1,1      // v20[v7]-v21[v4]-v22[v3]-v23[v2] - untere Fläche
          };

    private float[] cubeFaceColors = {
            1,0,0,  1,0,0,  1,0,0,  1,0,0,      // rot
            0,1,0,  0,1,0,  0,1,0,  0,1,0,      // grün
            0,0,1,  0,0,1,  0,0,1,  0,0,1,      // blau
            1,1,0,  1,1,0,  1,1,0,  1,1,0,      // gelb
            0,1,1,  0,1,1,  0,1,1,  0,1,1,      // türkis
            1,0,1,  1,0,1,  1,0,1,  1,0,1       // pink
            };
    //---------------------------------------------------------------------------
```

Vertex Arrays: effizientes Zeichnen von Primitiven (2)

- Das Einbinden von Datenfeldern in die Java-Puffer **optimiert** die nachfolgende Übertragung der Inhalte (hier: Endpunkten und Farben) vom RAM in die Grafikkarte

- C/C++ benötigt keine Einbindung

```
private FloatBuffer cubeCoordBuffer = Buffers.newDirectFloatBuffer(cubeCoords);
private FloatBuffer cubeFaceColorBuffer = Buffers.newDirectFloatBuffer(cubeFaceColors);
//------------------------------------------------------------------------
public void display(GLAutoDrawable drawable) {
    ...
    //--------------------------------------------------------------------
    gl2.glVertexPointer(3, GL2.GL_FLOAT, 0, cubeCoordBuffer);
    gl2.glColorPointer(3,GL2. GL_FLOAT, 0, cubeFaceColorBuffer);

    gl2.glEnableClientState(GL2.GL_VERTEX_ARRAY);
    gl2.glEnableClientState(GL2.GL_COLOR_ARRAY);

    gl2.glDrawArrays(GL2.GL_QUADS, 0, 24);

    gl2.glDisableClientState(GL2.GL_VERTEX_ARRAY);
    gl2.glDisableClientState(GL2.GL_COLOR_ARRAY);
}
//------------------------------------------------------------------------
...
```

- Die Übertragung der Inhalte vom RAM in die Grafikkarte erfolgt erst unter der Methode display() - das heißt, per jedes Rendern-(aka Animations-)Frame

VBO: es geht noch effektiver ☺

- ein ganzzahliges Datenfeld wird eingeführt (aka deklariert) – es wird für die Speicherung der Verweise (aka Referenzen) auf die VBOe (aka Vertex-Puffer-Objekte) benötigt

```
package cg;
//-----------------------------------------------------------------------
import com.jogamp.common.nio.Buffers;
import java.nio.ShortBuffer;
import java.nio.FloatBuffer;
...
//-----------------------------------------------------------------------
public class Main implements GLEventListener, KeyListener {
    private int[] bufID = new int[3];
    ...
    //-------------------------------------------------------------------
    public static void main(String[] args) {...}
    //-------------------------------------------------------------------

                                                //     v4----- v7
                                                //   /|      /|
                                                //  v0------v3|
                                                //  | |     | |
                                                //  | v5----|-v6
                                                //  |/      |/
                                                //  v1------v2

    private float[] cubeCoords = {
                    -1.0f,  1.0f,  1.0f,            //v0
                    -1.0f, -1.0f,  1.0f,            //v1
                     1.0f, -1.0f,  1.0f,            //v2
                     1.0f,  1.0f,  1.0f,            //v3
                    -1.0f,  1.0f, -1.0f,            //v4
                    -1.0f, -1.0f, -1.0f,            //v5
                     1.0f, -1.0f, -1.0f,            //v6
                     1.0f,  1.0f, -1.0f,            //v7
                };
```

VBO: es geht noch effektiver ☺ (2)

- durch glBufferData() wird ein Verweis eines VBO für einen bestimmten Nutzungszweck gebunden
- **Achtung:** zu jeder Zeit darf nur ein einziges VBO zu einem konkreten Zweck gebunden sein – sonst wird die vorherige Bindung abgebrochen

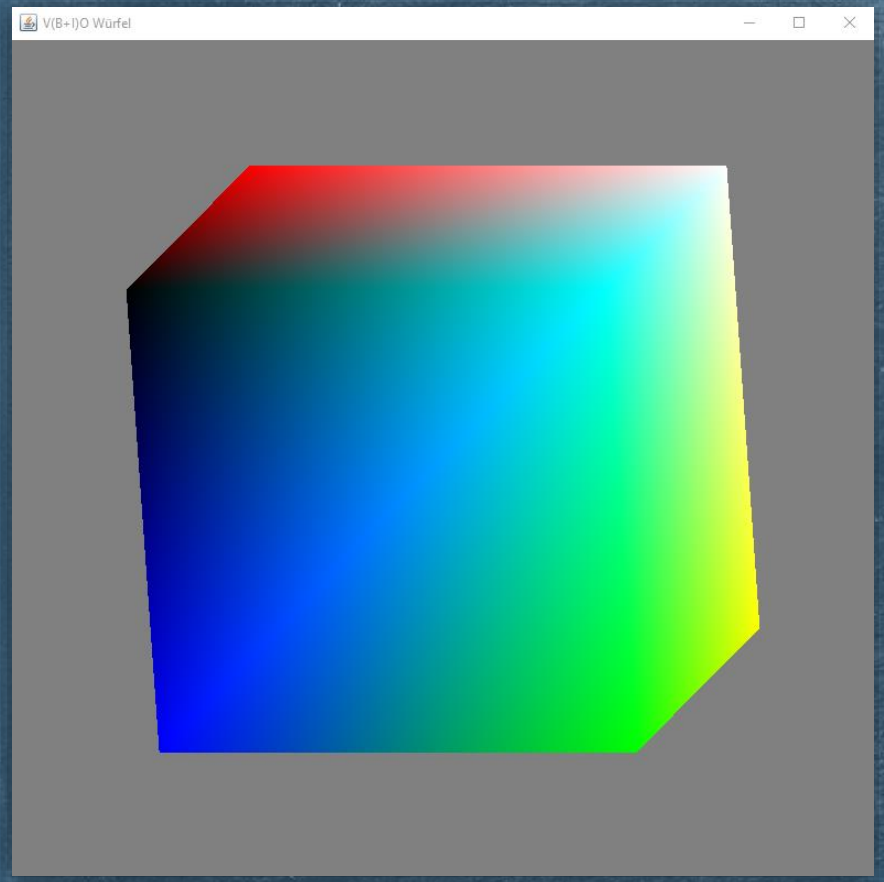
```
    private float[] cubeFaceCols = {
                        0.0f, 0.0f, 0.0f,                   //v0 - schwarz
                        0.0f, 0.0f, 1.0f,                   //v1 - blau
                        0.0f, 1.0f, 0.0f,                   //v2 - grün
                        0.0f, 1.0f, 1.0f,                   //v3 - türkis
                        1.0f, 0.0f, 0.0f,                   //v4 - rot
                        1.0f, 0.0f, 1.0f,                   //v5 - pink
                        1.0f, 1.0f, 0.0f,                   //v6 - gelb
                        1.0f, 1.0f, 1.0f,                   //v7 - weiß
                      };

    private short[] cubeFaceInds = {                  //äußere Seiten der Flächen sind im Gegenuhrzeigersinn
                0, 1, 2, 3,                                 //vordere Fläche
                0, 4, 5, 1,                                 //linke Fläche
                4, 7, 6, 5,                                 //hintere Fläche
                2, 6, 7, 3,                                 //rechte Fläche
                1, 5, 6, 2,                                 //untere Fläche
                3, 7, 4, 0,                                 //obere Fläche
                };

    //------------------------------------------------------------------------
    private FloatBuffer cubeCBuf = Buffers.newDirectFloatBuffer(cubeCoords);
    private FloatBuffer cubeFCBuf = Buffers.newDirectFloatBuffer(cubeFaceCols);
    private ShortBuffer cubeFIBuf = Buffers.newDirectShortBuffer(cubeFaceInds);
    //------------------------------------------------------------------------
     public void init(GLAutoDrawable drawable) {
            ...
            gl.glGenBuffers(3, bufID, 0);

            gl.glBindBuffer(GL2.GL_ARRAY_BUFFER, bufID[0]);
gl.glBufferData(GL2.GL_ARRAY_BUFFER, cubeCBuf.capacity()*Buffers.SIZEOF_FLOAT, cubeCBuf, GL2.GL_STATIC_DRAW);

            gl.glBindBuffer(GL2.GL_ARRAY_BUFFER, bufID[1]);
gl.glBufferData(GL2.GL_ARRAY_BUFFER, cubeFCBuf.capacity()*Buffers.SIZEOF_FLOAT, cubeFCBuf, GL2.GL_STATIC_DRAW);

            gl.glBindBuffer(GL2.GL_ELEMENT_ARRAY_BUFFER, bufID[2]);
gl.glBufferData(GL2.GL_ELEMENT_ARRAY_BUFFER, cubeFIBuf.capacity()*Buffers.SIZEOF_INT, cubeFIBuf, GL2.GL_STATIC_DRAW);
        }
```

- Verweise der 3 VBOe werden generiert und in bufID gelagert

- Die **Übertragung** der Inhalte vom RAM in die Grafikkarte erfolgt durch den Aufruf der Methode **glBufferData()** - das heißt **nur einmal** und **nicht** per jedes Rendern-Frame

- alternativ könnte man die Methode glBufferSubData() verwenden – diese wäre sogar flexibler (dadurch lassen sich die Inhalte jederzeit updaten).

VBO: es geht noch effektiver ☺ (3)




```
public void display(GLAutoDrawable drawable) {
    ...
    gl.glBindBuffer(GL2.GL_ARRAY_BUFFER,bufID[0]);
    gl.glVertexPointer(3, GL2.GL_FLOAT, 0, 0);
    gl.glEnableClientState(GL2.GL_VERTEX_ARRAY);

    gl.glBindBuffer(GL2.GL_ARRAY_BUFFER,bufID[1]);
    gl.glColorPointer(3, GL2.GL_FLOAT, 0, 0);
    gl.glEnableClientState(GL2.GL_COLOR_ARRAY);

    gl.glBindBuffer(GL2.GL_ELEMENT_ARRAY_BUFFER, bufID[2]);

    gl.glDrawElements(GL2.GL_QUADS, cubeFIBuf.capacity(), GL2.GL_UNSIGNED_SHORT, 0);

    gl.glDisableClientState(GL2.GL_VERTEX_ARRAY);
    gl.glDisableClientState(GL2.GL_COLOR_ARRAY);

    gl.glBindBuffer(GL2.GL_ARRAY_BUFFER, 0);
    gl.glBindBuffer(GL2.GL_ELEMENT_ARRAY_BUFFER, 0);
}
//-------------------------------------------------------------------------
...
```

VAO: VBO mit einfacherer Handhabung

- Ein VAO (aka Vertex-Array-Objekt) speichert keine Inhalte (wie Vertex-Daten oder die von Farben), sondern lediglich die Zustände der VBOe

- allgemeiner Hinweis: alternativ könnte man die Datenfelder (zB cube2Ver und cube2Col) vertexweise zusammensetzen (also ein vereintes Datenfeld erstellen) und die Argumenten stride [aka Schrittweite] sowie offset [aka Startindex] von _Pointer() Methoden [wie VertexPointer() und ColorPointer()] anpassen

```
package cg;
//-----------------------------------------------------------------------------------------------------------
import com.jogamp.common.nio.Buffers;
import java.nio.ShortBuffer;
import java.nio.FloatBuffer;
...
//-----------------------------------------------------------------------------------------------------------
public class Main implements GLEventListener, KeyListener {
    private int[] vaoID = new int[2];
    private int[] bufID = new int[5];
    ...
    //-------------------------------------------------------------------------------------------------------
    public static void main(String[] args) {...}
    //-------------------------------------------------------------------------------------------------------
    private float[] cubeCoords = {...};
    private float[] cubeFaceCols = {...};
    private short[] cubeFaceInds = {...};

    private float[] cube2Ver = {...};
    private float[] cube2Col = {...};
    //-------------------------------------------------------------------------------------------------------
    private FloatBuffer cubeCBuf = Buffers.newDirectFloatBuffer(cubeCoords);
    private FloatBuffer cubeFCBuf = Buffers.newDirectFloatBuffer(cubeFaceCols);
    private ShortBuffer cubeFIBuf = Buffers.newDirectShortBuffer(cubeFaceInds);

    private FloatBuffer cube2CBuf = Buffers.newDirectFloatBuffer(cube2Ver);
    private FloatBuffer cube2FCBuf = Buffers.newDirectFloatBuffer(cube2Col);
    //-------------------------------------------------------------------------------------------------------
```

VAO: VBO mit einfacherer Handhabung (2)

- zunächst muss der Name eines VAO generiert werden (hier: direkt 2 Namen, weil man 2 **VAO**e haben will) – es geschieht durch die Methode glGenVertexArrays()

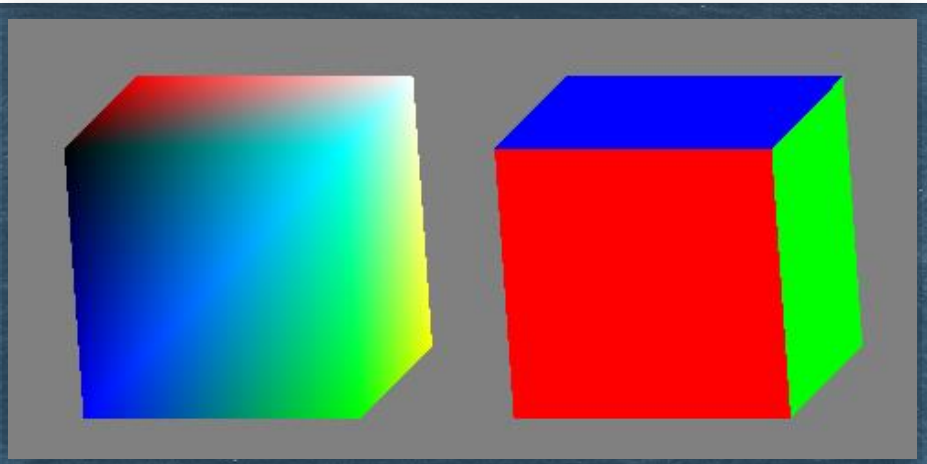
- um die Zustände der VBOe in einem VAO zu speichern, muss erst die Methode glBindVertexArray() aufgerufen werden und danach - die Methoden zur Einstellung der VBOe:
  - hier: glBindBuffer(), glEnableClientState(), glVertexPointer() und glColorPointer()
  - alternativ kann man glEnableVertexAttribArray() und glVertexAttribPointer() verwenden

```
    public void init(GLAutoDrawable drawable) {
            ...
            gl.glGenVertexArrays(2, vaoID, 0);
            gl.glGenBuffers(5, bufID, 0);
            //-------------------------------------------------------------------------
            gl.glBindVertexArray(vaoID[0]);

            gl.glBindBuffer(GL2.GL_ARRAY_BUFFER, bufID[0]);
gl.glBufferData(GL2.GL_ARRAY_BUFFER, cubeCBuf.capacity()*Buffers.SIZEOF_FLOAT, cubeCBuf, GL2.GL_STATIC_DRAW);
            gl.glVertexPointer(3, GL2.GL_FLOAT, 0, 0);
            gl.glEnableClientState(GL2.GL_VERTEX_ARRAY);

            gl.glBindBuffer(GL2.GL_ARRAY_BUFFER, bufID[1]);
gl.glBufferData(GL2.GL_ARRAY_BUFFER, cubeFCBuf.capacity()*Buffers.SIZEOF_FLOAT, cubeFCBuf, GL2.GL_STATIC_DRAW);
            gl.glColorPointer(3, GL2.GL_FLOAT, 0, 0);
            gl.glEnableClientState(GL2.GL_COLOR_ARRAY);

            gl.glBindBuffer(GL2.GL_ELEMENT_ARRAY_BUFFER, bufID[2]);
gl.glBufferData(GL2.GL_ELEMENT_ARRAY_BUFFER, cubeFIBuf.capacity()*Buffers.SIZEOF_INT, cubeFIBuf, GL2.GL_STATIC_DRAW);
            //
            gl.glBindVertexArray(vaoID[1]);

            gl.glBindBuffer(GL2.GL_ARRAY_BUFFER, bufID[3]);
gl.glBufferData(GL2.GL_ARRAY_BUFFER, cube2CBuf.capacity()*Buffers.SIZEOF_FLOAT, cube2CBuf, GL2.GL_STATIC_DRAW);
            gl.glVertexPointer(3, GL2.GL_FLOAT, 0, 0);
            gl.glEnableClientState(GL2.GL_VERTEX_ARRAY);

            gl.glBindBuffer(GL2.GL_ARRAY_BUFFER, bufID[4]);
gl.glBufferData(GL2.GL_ARRAY_BUFFER, cube2FCBuf.capacity()*Buffers.SIZEOF_FLOAT, cube2FCBuf, GL2.GL_STATIC_DRAW);
            gl.glColorPointer(3, GL2.GL_FLOAT, 0, 0);
            gl.glEnableClientState(GL2.GL_COLOR_ARRAY);
        }
    //-----------------------------------------------------------------------------------
```

VAO: VBO mit einfacherer Handhabung (3)

- nun werden die Szenenelemente nur in 2 Codezeilen zum Zeichnen gebracht:
  - Das **Binden** eines entsprechenden **VAO** durch den Aufruf der Methode glBindVertexArray()
  - Das eigentliche **Rendern** durch [entweder_oder]:
    - DrawElements() – etwas leistungsfähiger (nutzt Indizien)
    - DrawArrays()

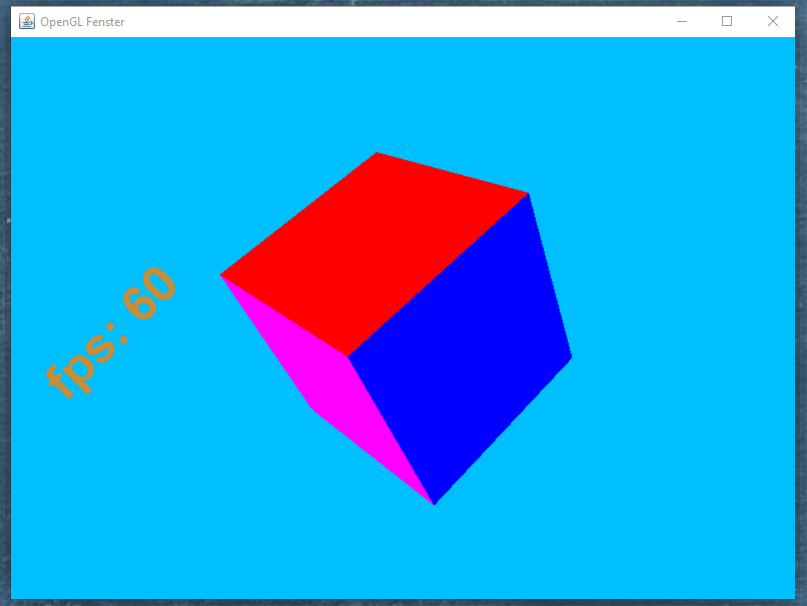
```
public void display(GLAutoDrawable drawable) {
   ...
   gl.glBindVertexArray(vaoID[0]);
   gl.glDrawElements(GL2.GL_QUADS, cubeFIBuf.capacity(), GL2.GL_UNSIGNED_SHORT, 0);
   //-------------------------------------------------------------------------
   ...
   gl.glBindVertexArray(vaoID[1]);
   gl.glDrawArrays(GL2.GL_QUADS, 0, cube2CBuf.capacity());
   //-------------------------------------------------------------------------
   gl.glBindVertexArray(0);
   gl.glDisableClientState(GL2.GL_VERTEX_ARRAY);
   gl.glDisableClientState(GL2.GL_COLOR_ARRAY);
   gl.glBindBuffer(GL2.GL_ARRAY_BUFFER, 0);
   gl.glBindBuffer(GL2.GL_ELEMENT_ARRAY_BUFFER, 0);
}
//-----------------------------------------------------------------------------
...
```



- **Achtung**: vor allem bei C/C++ sollte man die nicht mehr benötigte Puffer-Objekte entbinden und löschen – es geschieht durch das folgende **TOP-DOWN Schema**:

```
glDisableVertexAttribArray(0)
glBindBuffer(GL_ARRAY_BUFFER, 0)
glDeleteBuffers(bufID[x])
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0)
glDeleteBuffers(bufID[y])
glBindVertexArray(0)
glDeleteVertexArrays(vaoID[z])
```

TextRenderer (2D): Schreiben auf Zeichenfenstern




```
package cg;
//-----------------------------------------------------------------------------
...
import java.awt.Font;
import com.jogamp.opengl.util.awt.TextRenderer;
//-----------------------------------------------------------------------------
public class Main implements GLEventListener {
      //-----------------------------------------------------------------------
      ...
      TextRenderer renderer;
      //-----------------------------------------------------------------------
      public static void main(String[] args) {...}
      //-----------------------------------------------------------------------
      @Override
      public void init(GLAutoDrawable drawable) {
          ...
          renderer = new TextRenderer(new Font("SansSerif", Font.BOLD, 50));
      }
      //-----------------------------------------------------------------------
      @Override
      public void display(GLAutoDrawable drawable) {
            ...
            renderer.beginRendering(drawable.getSurfaceWidth(), drawable.getSurfaceHeight());
                gl.glMatrixMode(GL2.GL_MODELVIEW);
                gl.glRotatef(45.0f, 0.0f, 0.0f, 1.0f);

                renderer.setColor(1.0f, 0.5f, 0.0f, 0.8f);
                renderer.draw("  fps: " + fps, 150, 100);
            renderer.endRendering();
            ...
      }
      //-----------------------------------------------------------------------
      ...
}
```