WiSe 18/19

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Prof. Dr. Winfried Kurth
Alex Tavkhelidze

# Praktikum Computergrafik

*Folien für Extras*

Übung 1: grafische Primitive

Fenstereinstellungen

Exportieren in Eclipse

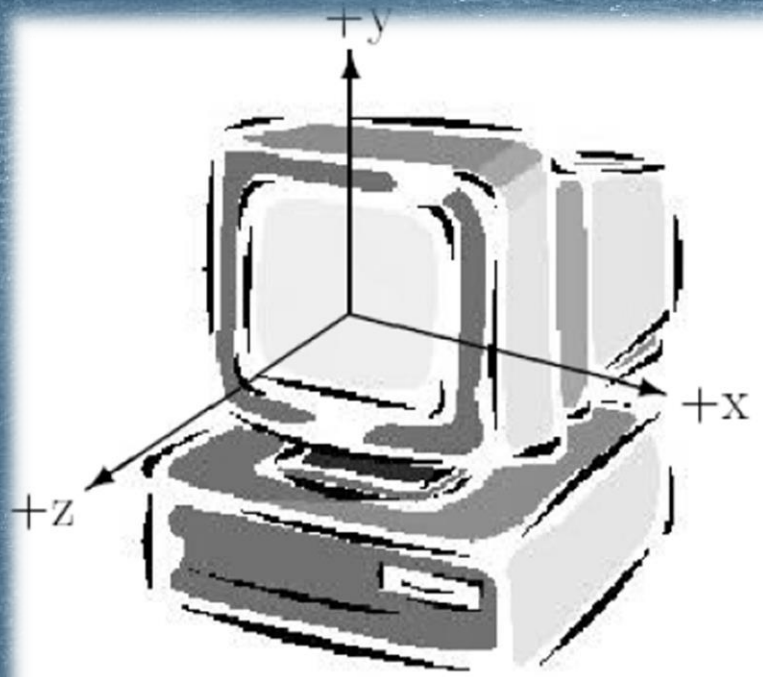
*Zum Abspielen klicken (Adobe Flash)*

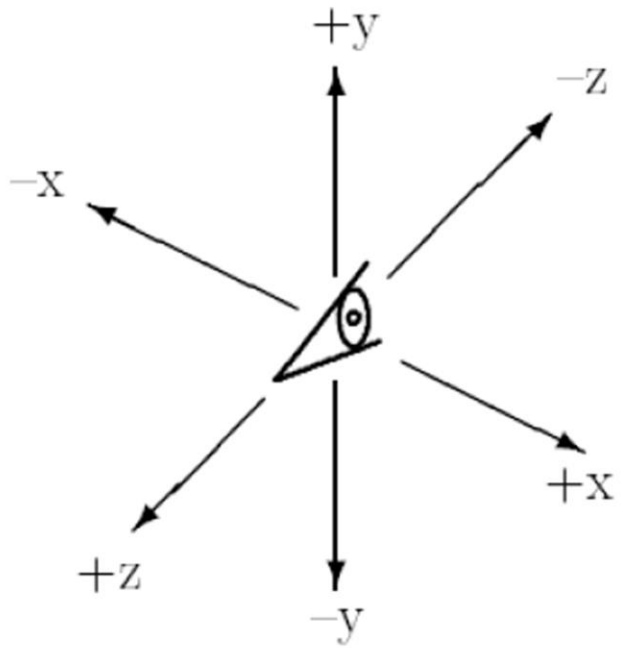
# Grafische Primitive in OpenGL

- Mit einfachen Bausteinen erstellt man (hoch)komplexe Strukturen
- Die Bausteine nennt man Primitive in OpenGL

*Originalquelle der Animation: https://upload.wikimedia.org/wikipedia/commons/b/b7/Beetle.gif*

# Koordinatensystem in OpenGL

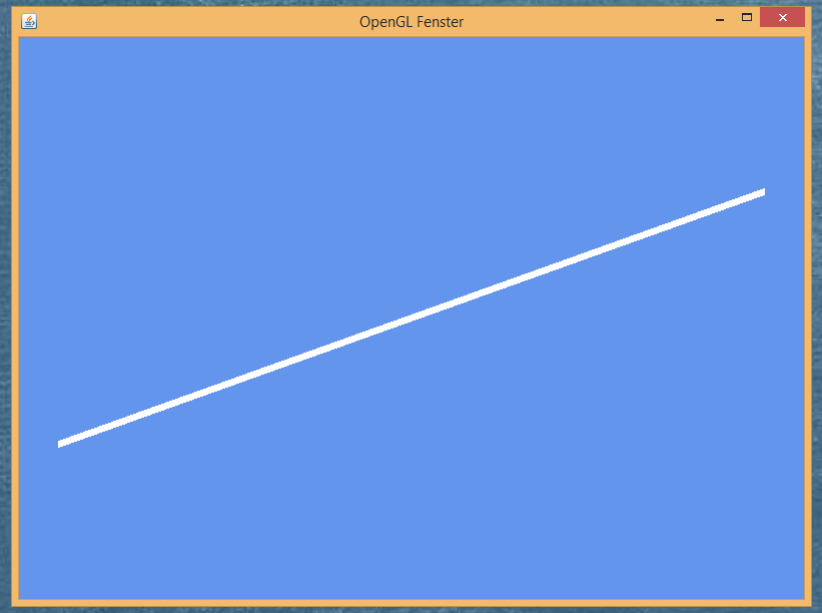



Bestimmung eines 3D-Knotens (Punktes) erfolgt durch den Befehl glVertex3f(float x, float y, float z)

(Nischwitz *et al.*, 2011)

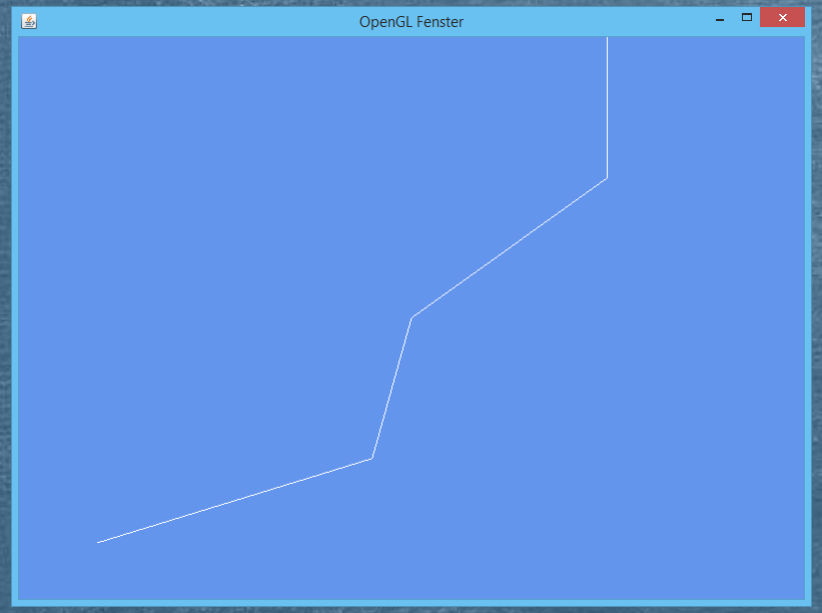
# Liniensegmente




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

    GL2 gl = drawable.getGL().getGL2();
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);

        final float w = 7.0f;

        gl.glLineWidth(w);

        gl.glBegin(GL2.GL_LINES);

            gl.glVertex3f(-0.9f, -0.45f, 0.0f);
            gl.glVertex3f( 0.9f, 0.45f, 0.0f);

        gl.glEnd();

    gl.glFlush();

}
```

- Der Parameter GL2.GL_LINES wandelt zwei Knoten (Vertices) in ihre Verbindungstrecke um
- Die Linienbreite setzt man mit der Methode glLineWidth(float width)

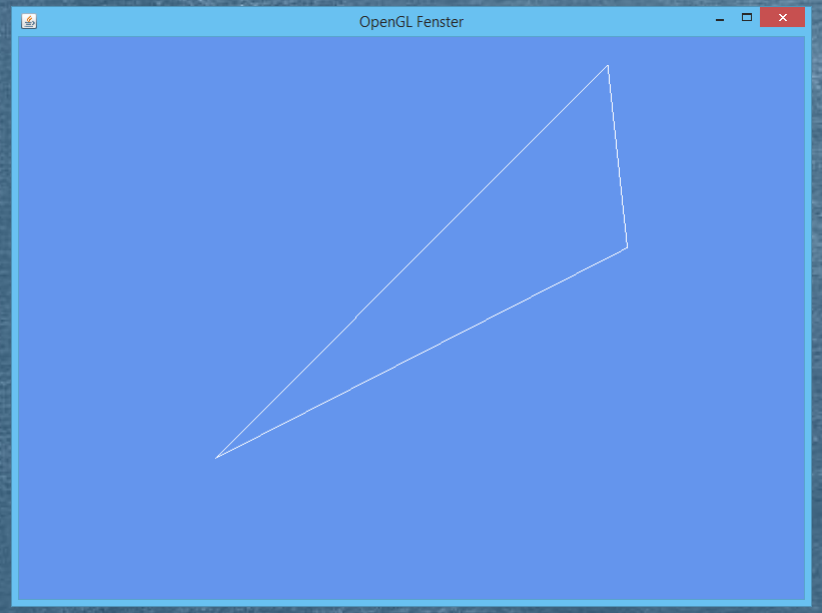
# Linienzüge




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

    GL2 gl = drawable.getGL().getGL2();
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);

    gl.glBegin(GL2.GL_LINE_STRIP);

        gl.glVertex3f(-0.8f,-0.8f, 0.0f);
        gl.glVertex3f(-0.1f,-0.5f, 0.0f);
        gl.glVertex3f( 0.0f, 0.0f, 0.0f);
        gl.glVertex3f(0.5f, 0.5f, 0.0f);
        gl.glVertex3f(0.5f, 1.0f, 0.0f);

    gl.glEnd();

    gl.glFlush();

}
```

- Der Parameter GL2.GL_LINE_STRIP interpretiert die Liste der Vertices als Endpunkte eines Linienzuges

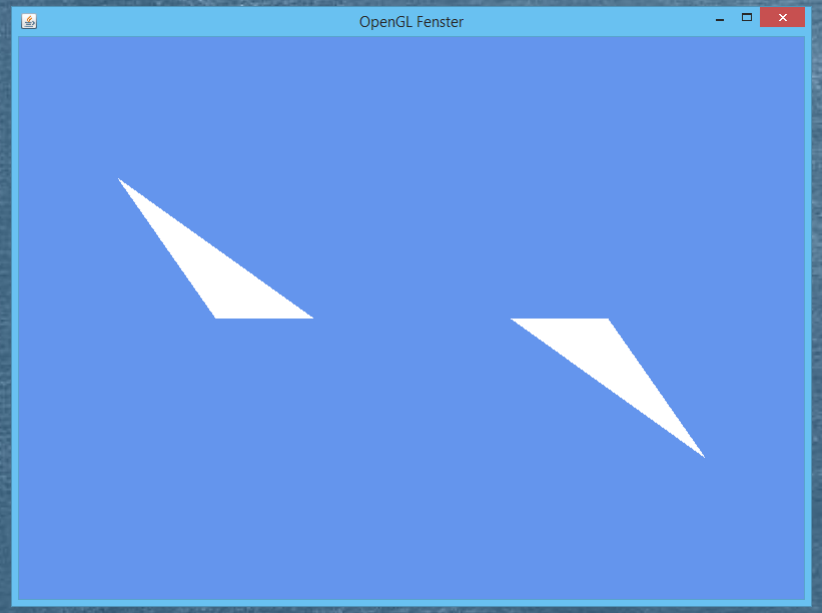
# geschlossene Linienzüge




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

    GL2 gl = drawable.getGL().getGL2();
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);

    gl.glBegin(GL2.GL_LINE_LOOP);

        gl.glVertex3f(-0.5f,-0.5f, 0.0f);
        gl.glVertex3f( 0.55f, 0.25f, 0.0f);
        gl.glVertex3f( 0.5f, 0.9f, 0.0f);

    gl.glEnd();

    gl.glFlush();

}
```

- Der Parameter GL2.GL_LINE_LOOP bildet aus der Liste der Vertices einen geschlossenen Linienzug

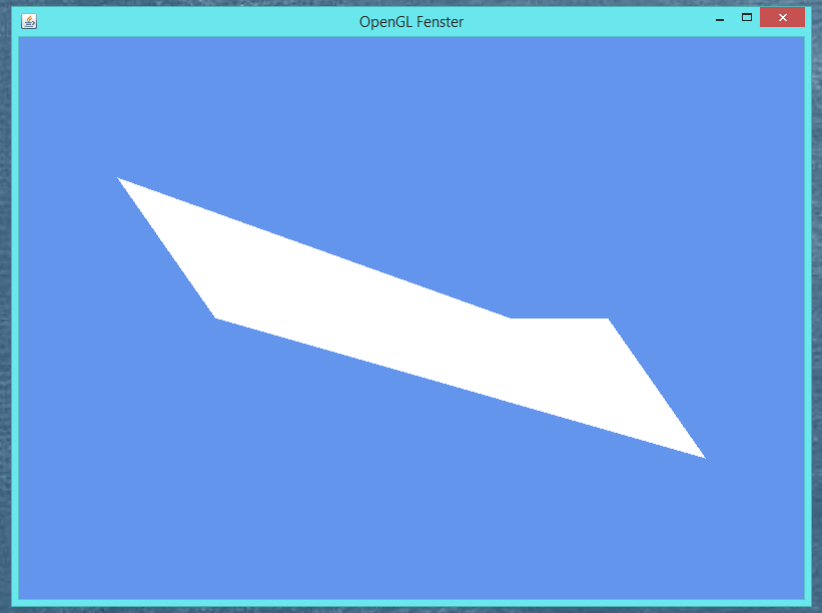
# Dreiecke




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

    GL2 gl = drawable.getGL().getGL2();
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);

    gl.glBegin(GL2.GL_TRIANGLES);

        gl.glVertex3f(0.50f, 0.0f, 0.0f);
        gl.glVertex3f(0.75f, -0.50f, 0.0f);
        gl.glVertex3f(0.25f, 0.0f, 0.0f);

        gl.glVertex3f(-0.50f, 0.0f, 0.0f);
        gl.glVertex3f(-0.75f, 0.50f, 0.0f);
        gl.glVertex3f(-0.25f, 0.0f, 0.0f);

    gl.glEnd();

    gl.glFlush();

}
```

- Der Parameter GL2.GL_TRIANGLES fasst die Dreier von Punkten als Dreiecke zusammen

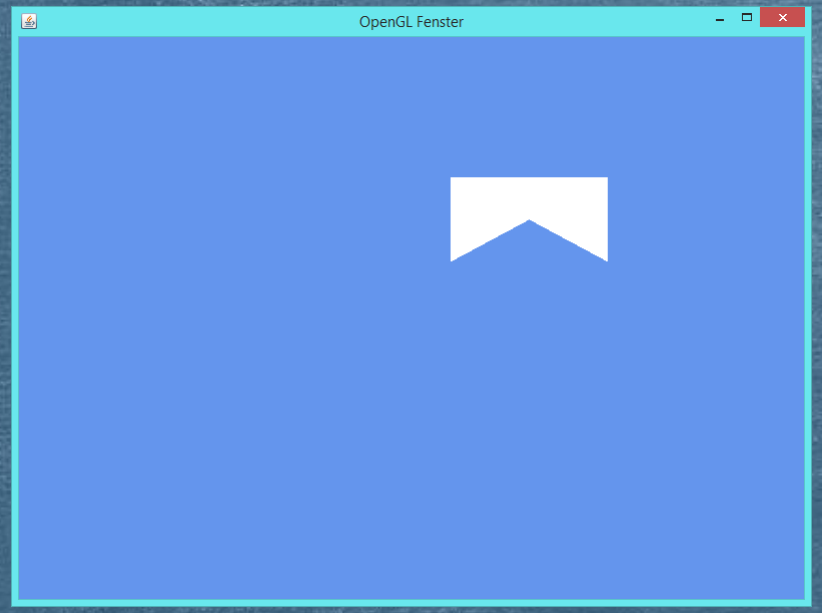
# Dreiecksstreifen (1)




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

    GL2 gl = drawable.getGL().getGL2();
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);

    gl.glBegin(GL2.GL_TRIANGLE_STRIP);

        gl.glVertex3f( 0.50f, 0.0f, 0.0f);
        gl.glVertex3f( 0.75f,-0.5f, 0.0f);
        gl.glVertex3f( 0.25f, 0.0f, 0.0f);

        gl.glVertex3f(-0.50f, 0.0f, 0.0f);
        gl.glVertex3f(-0.75f, 0.5f, 0.0f);
        gl.glVertex3f(-0.25f, 0.0f, 0.0f);

    gl.glEnd();;

    gl.glFlush();

}
```

- Der Parameter GL2.GL_TRIANGLE_STRIP interpretiert die Liste der Vertices als Gruppe verbundener Dreiecke

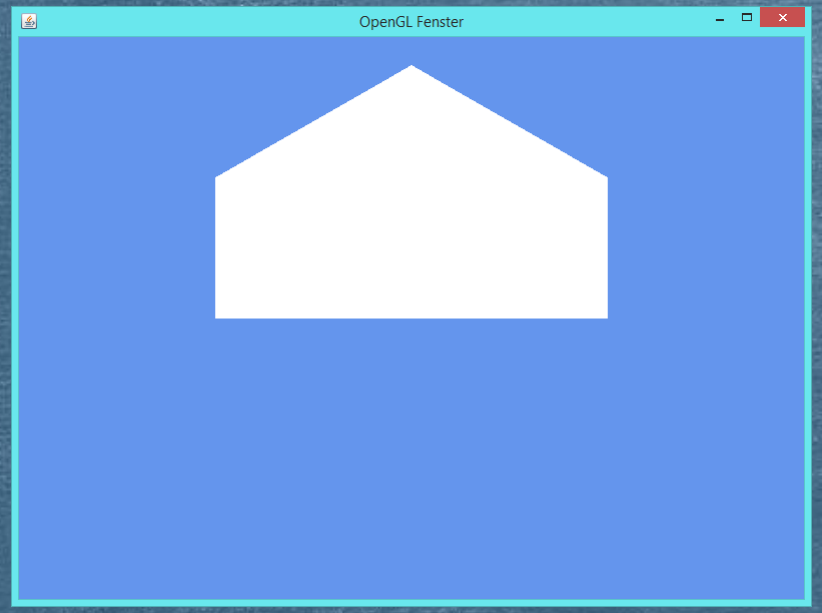
# Dreiecksstreifen (2)




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

    GL2 gl = drawable.getGL().getGL2();
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);

    gl.glBegin(GL2.GL_TRIANGLE_STRIP);

        gl.glVertex3f( 0.1f, 0.2f, 0.0f);
        gl.glVertex3f( 0.1f, 0.5f, 0.0f);
        gl.glVertex3f( 0.5f, 0.5f, 0.0f);
        gl.glVertex3f( 0.5f, 0.2f, 0.0f);

    gl.glEnd();;

    gl.glFlush();

}
```

- Welches Bauverfahren des Parameters GL2.GL_TRIANGLE_STRIP ergibt sich aus dem Vergleich beider Dreicksstreifen?

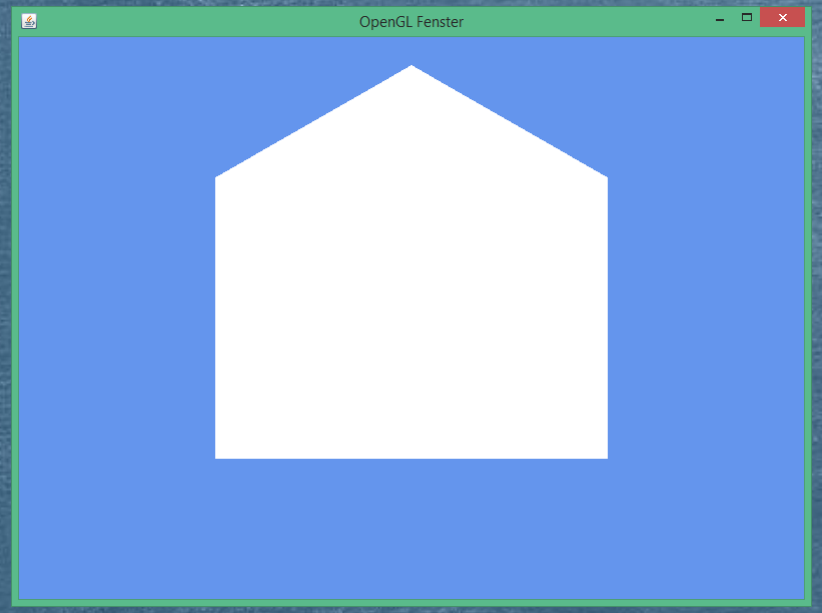
# Polygone




- Der Parameter GL2.GL_POLYGON nutzt die Liste der Knoten als Spitze eines Polygons

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

    GL2 gl = drawable.getGL().getGL2();
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);

    gl.glBegin(GL2.GL_POLYGON);

        gl.glVertex3f(-0.5f, 0.0f, 0.0f);
        gl.glVertex3f( 0.5f, 0.0f, 0.0f);
        gl.glVertex3f( 0.5f, 0.5f, 0.0f);
        gl.glVertex3f( 0.0f, 0.90f, 0.0f);
        gl.glVertex3f(-0.5f, 0.5f, 0.0f);

    gl.glEnd();

    gl.glFlush();

}
```

# einfachste Zusammensetzung von Bausteinen




- Erfolgt durch die nacheinanderfolgende Verwendung der gl.glBegin … gl.glEnd Code-Ausschnitte

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

    GL2 gl = drawable.getGL().getGL2();
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT | GL2.GL_DEPTH_BUFFER_BIT);

    gl.glBegin(GL2.GL_POLYGON);

        gl.glVertex3f(-0.5f, 0.0f, 0.0f);
        gl.glVertex3f( 0.5f, 0.0f, 0.0f);
        gl.glVertex3f( 0.5f, 0.5f, 0.0f);
        gl.glVertex3f( 0.0f, 0.9f, 0.0f);
        gl.glVertex3f(-0.5f, 0.5f, 0.0f);

    gl.glEnd();

    gl.glBegin(GL2.GL_POLYGON);

        gl.glVertex3f(-0.5f,-0.5f, 0.0f);
        gl.glVertex3f(-0.5f, 0.0f, 0.0f);
        gl.glVertex3f( 0.5f, 0.0f, 0.0f);
        gl.glVertex3f( 0.5f,-0.5f, 0.0f);

    gl.glEnd();

    gl.glFlush();

}
```

# Weitere Grafik-Primitive

| Befehl | Beschreibung |
|---|---|
| GL_POINTS | Zeichnet jeweils einen Punkt |
| GL_QUADS | Zeichnet mehrere 4-Ecke (aus jeweils 4 Vertices) |
| GL_QUAD_STRIP | Zeichnet einen Streifen von 4-Ecken |
| GL_TRIANGLE_FAN | Zeichnet eine Serie von 3-Ecken, die sich einen Mittelpunkt teilen: zB v0,v1,v2, dann v0,v2,v3, dann v0,v3,v4, usw. |

# Fenstereinstellungen
*Farbe*

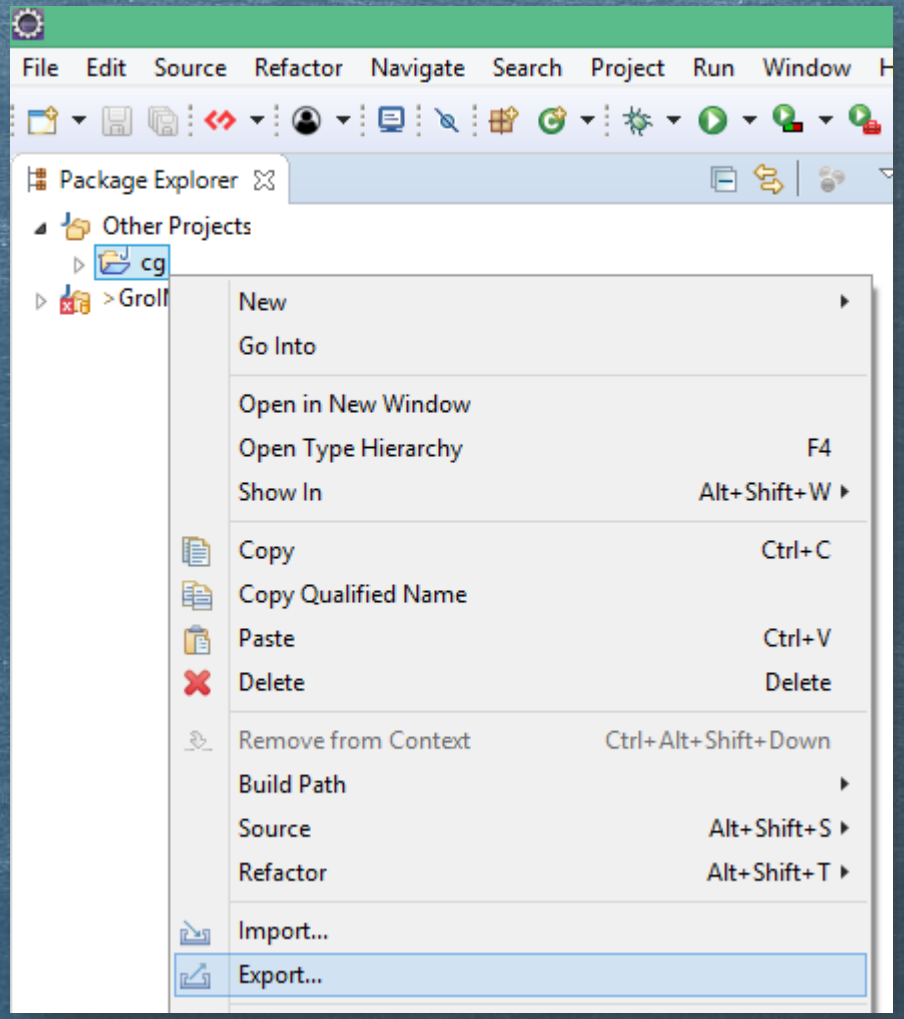

- Die Farbtabelle mit der von 0 bis 1 verlaufenden RGB-Werten ist beispielsweise unter diesem Link abrufbar:

http://prideout.net/archive/colors.php#Chart

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

    GL2 gl = drawable.getGL().getGL2();

2   /*Mit diesem Befehl setzt sich eine andere Farbe mit Gleitkommazahlen
     * als Parameter,die dem RGBA-Format entsprechen */
    gl.glClearColor(0.863f, 0.078f, 0.235f, 1.0f);

}

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

    GL2 gl = drawable.getGL().getGL2();

1   // Löschung des Farbpuffers (Schwarz setzt sich defaultmäßig)
    gl.glClear(GL2.GL_COLOR_BUFFER_BIT);

    // call your draw code here

    gl.glFlush();

}
```

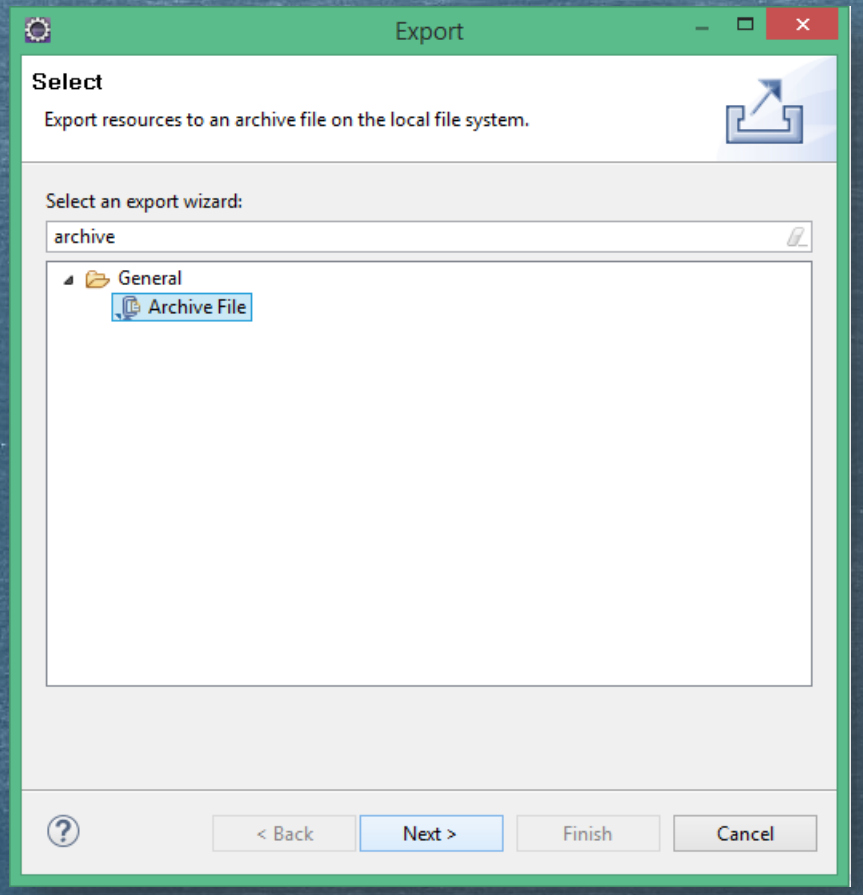
# Exportieren von Java-Projekten in Eclipse

*Schritt 1 von 4*



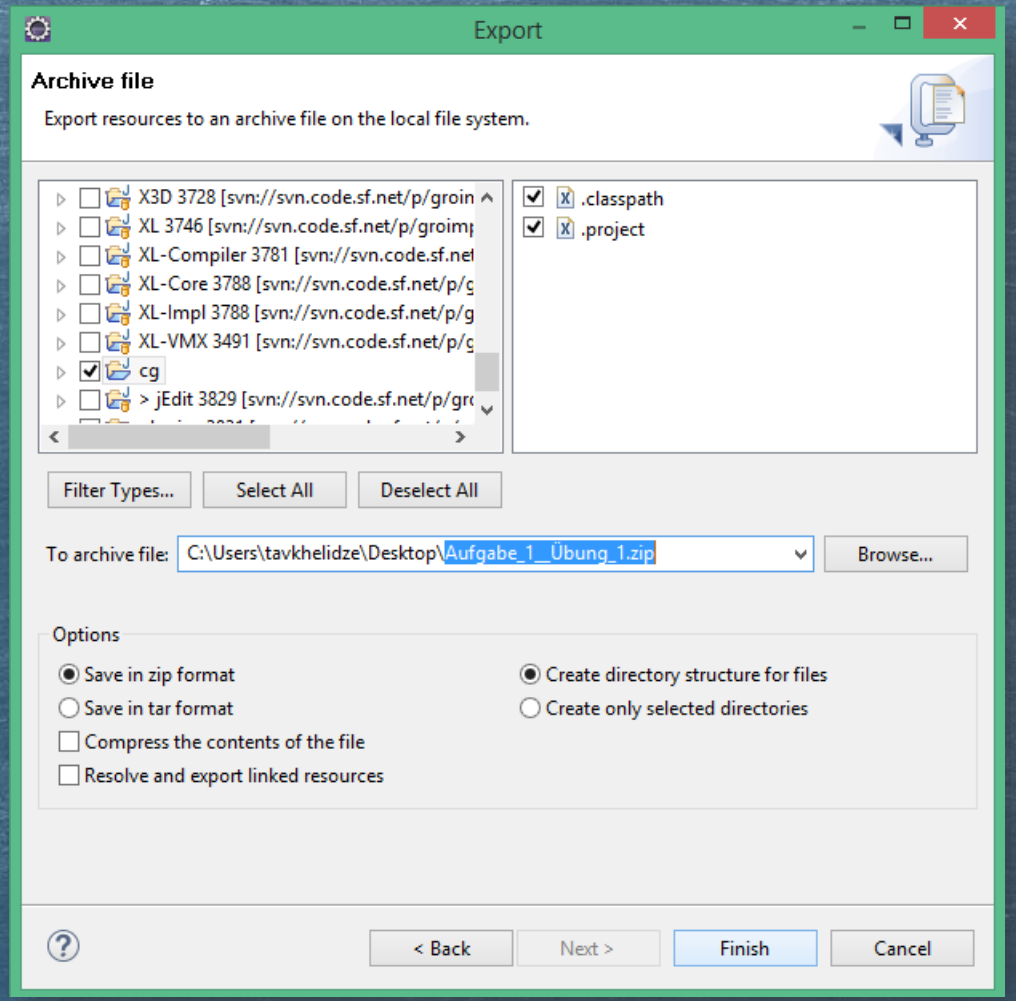
# Exportieren von Java-Projekten in Eclipse

*Schritt 2 von 4*



# Exportieren von Java-Projekten in Eclipse

*Schritt 3 von 4*



# Exportieren von Java-Projekten in Eclipse

*Schritt 4 von 4*