

Effizientes Zeichnen von Primitiven in OpenGL – `glDrawArrays()`

```
public class Main extends GLJPanel implements GLEventListener {
    public static void main(String[] args) {
    }
    public Main() {
    }
    //Koordinaten von allen Knoten eines Würfels
    private float[] cubeCoords = {
        1,1,1,  -1,-1,1,  -1,-1,-1,  1,-1,-1,    // Fläche #1
        1,1,1,  1,-1,1,  1,-1,-1,  1,1,-1,     // Fläche #2
        1,1,1,  1,1,-1,  -1,1,-1,  -1,1,1,     // Fläche #3
        -1,-1,-1, -1,1,-1,  1,1,-1,  1,-1,-1,   // Fläche #4
        -1,-1,-1, -1,-1,1, -1,1,1,  -1,1,-1,   // Fläche #5
        -1,-1,-1,  1,-1,-1,  1,1,1,  -1,-1,1  }; // Fläche #6

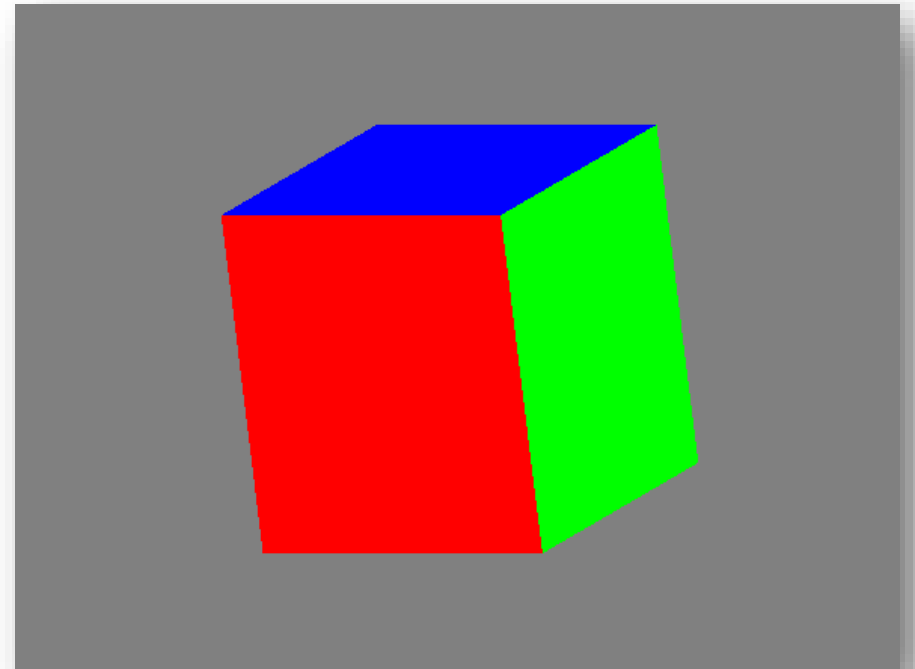
    //Farbwerte von allen Knoten eines Würfels
    private float[] cubeFaceColors = {
        1,0,0,  1,0,0,  1,0,0,  1,0,0,    // Fläche #1 is rot
        0,1,0,  0,1,0,  0,1,0,  0,1,0,    // Fläche #2 is grün
        0,0,1,  0,0,1,  0,0,1,  0,0,1,    // Fläche #3 is blau
        1,1,0,  1,1,0,  1,1,0,  1,1,0,    // Fläche #4 is gelb
        0,1,1,  0,1,1,  0,1,1,  0,1,1,    // Fläche #5 is ...
        1,0,1,  1,0,1,  1,0,1,  1,0,1,    }; // Fläche #6 is ...

    // Datenfelder werden in den entsprechenden Puffer eingepackt
    private FloatBuffer cubeCoordBuffer = Buffers.newDirectFloatBuffer(cubeCoords);
    private FloatBuffer cubeFaceColorBuffer = Buffers.newDirectFloatBuffer(cubeFaceColors);

    public void display(GLAutoDrawable drawable) {
        // Puffer werden zum Zeichnen übergeben (Vertex- und Farb-Arrays werden bestimmt)
        gl2.glVertexPointer( 3, GL2.GL_FLOAT, 0, cubeCoordBuffer );
        gl2.glColorPointer( 3, GL2.GL_FLOAT, 0, cubeFaceColorBuffer );

        // Schaltet Vertex- und Farb-Arrays ein
        gl2.glEnableClientState( GL2.GL_VERTEX_ARRAY );
        gl2.glEnableClientState( GL2.GL_COLOR_ARRAY );

        // der Würfel wird gezeichnet (durch Verwendung von der Konstante GL_QUADS über 8 Knoten mit je 3 Werten)
        gl2.glDrawArrays( GL2.GL_QUADS, 0, 24 );
    }
}
```



Einfaches Textrendering in OpenGL

```
import com.jogamp.opengl.util.FPSAnimator;
import com.jogamp.opengl.util.awt.TextRenderer;

public class Main implements GLEventListener {

    int fps;

    TextRenderer renderer;

    public static void main(String[] args) {

        final FPSAnimator animator = new FPSAnimator( glcanvas, 60,true );

        animator.start();

    }

    @Override
    public void init(GLAutoDrawable drawable) {

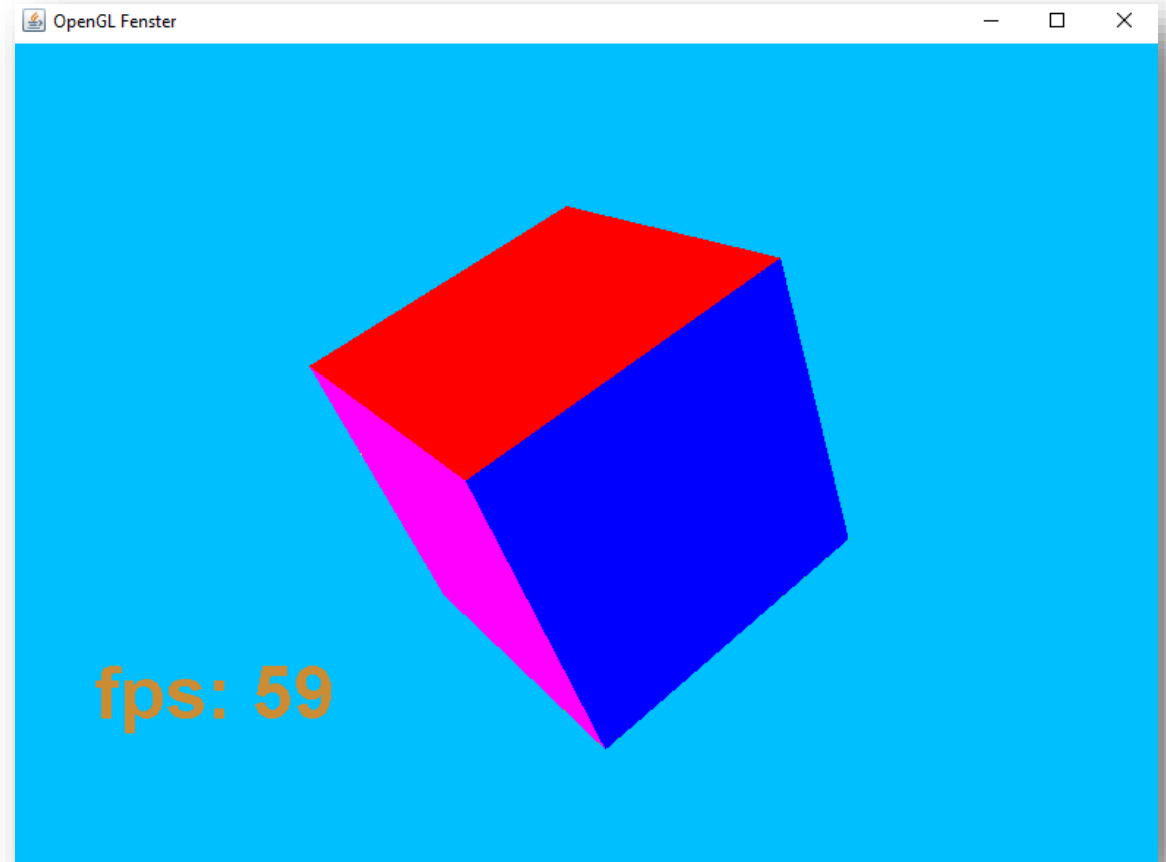
        renderer = new TextRenderer(new Font("SansSerif", Font.BOLD, 50));

    }

    @Override
    public void display(GLAutoDrawable drawable) {

        renderer.beginRendering(drawable.getSurfaceWidth(), drawable.getSurfaceHeight());
        renderer.setColor(1.0f, 0.5f, 0.0f, 0.8f);
        // 30 und 100 stehen für X- und Y-Koordinaten des Schaufensters (Viewport)
        renderer.draw(" fps: " + fps, 30, 100);
        renderer.endRendering();

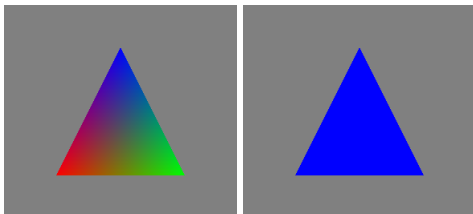
    }
}
```



Färbetechnik

- ▶ Die Art, wie die Primitive gefärbt werden (feste Farbe / Farbverlauf)
- ▶ **glShadeModel** mit Argument: GL_SMOOTH (Vorgabewert), GL_FLAT

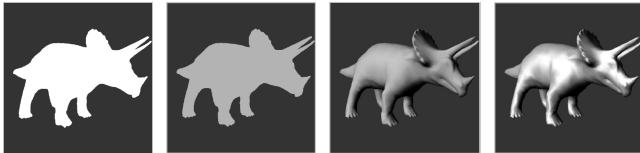
```
gl.glBegin(GL.GL_TRIANGLES);  
    gl.glColor3f(1.0f, 0.0f, 0.0f);  
    gl.glVertex3f(-1.0f, -1.0f, 0.0f);  
    gl.glColor3f(0.0f, 1.0f, 0.0f);  
    gl.glVertex3f(1.0f, -1.0f, 0.0f);  
    gl.glColor3f(0.0f, 0.0f, 1.0f);  
    gl.glVertex3f(0.0f, 1.0f, 0.0f);  
gl.glEnd();
```



OpenGL Beleuchtungsmodell (Phong-Beleuchtungsmodell)

- ▶ Licht kommt von mehreren Lichtquellen
- ▶ Verdeckung von Lichtquellen durch andere Objekte nicht berücksichtigt (kein Schattenwurf)
- ▶ Lichtapproximation: Rote, Grüne, Blaue Komponente
- ▶ 4 Beleuchtungskomponenten, die unabhängig voneinander berechnet werden

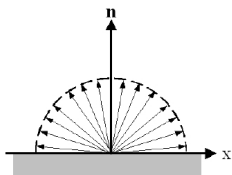
Vertexfarbe = emmisiv + ambient + diffus + spekulär



(Nischwitz *et al.*, 2011)

Emmisse Materialeigenschaft

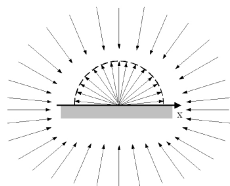
- ▶ Selbstleuchtende Oberfläche
- ▶ Licht gleichmäßig ausgesendet in alle Richtungen
- ▶ Allein durch die emmissiven Eigenschaften des Materials festgelegt



(Nischwitz *et al.*, 2011)

Ambienter Lichtanteil

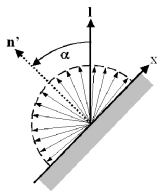
- ▶ Streulicht
- ▶ Kommt mit gleicher Intensität aus allen Richtungen, wird an der Oberfläche gleichmäßig in alle Richtungen gestreut
- ▶ Festgelegt durch die ambiente Eigenschaften der Lichtquelle und des Materials



(Nischwitz *et al.*, 2011)

Diffuser Lichtanteil

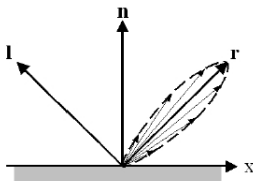
- ▶ Das gerichtete Licht, das von einer Punktquelle kommt
- ▶ Licht kommt aus einer Richtung, wird an der Oberfläche gleichmäßig in alle Richtungen gestreut
- ▶ Lichtintensität abhängig von der Orientierung der Oberfläche zur Lichtquelle (unabhängig von der Position des Beobachters), der Position und Farbe der Lichtquelle und Reflexionseigenschaften der Oberfläche



(Nischwitz *et al.*, 2011)

Spekularer Lichtanteil

- ▶ Das gerichtete Licht, das von einer Punktquelle kommt
- ▶ Licht kommt aus einer Richtung, wird an der Oberfläche überwiegend in eine Vorzugsrichtung (ideale Reflexionsrichtung) reflektiert
- ▶ Lichtintensität abhängig von der Position und der Farbe der Lichtquelle, der Orientierung und den Reflexionseigenschaften der Oberfläche, der Position des Beobachters

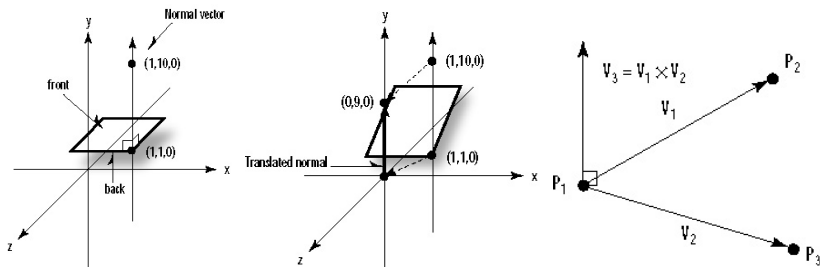


(Nischwitz *et al.*, 2011)

Lichtberechnung

- ▶ Normalenvektoren für jeden Vertex (Fläche) definieren
- ▶ Lichtquellen erstellen
- ▶ Beleuchtungsmodell festlegen
- ▶ Materialeigenschaften festlegen

Normalenvektoren



(Wright, Sweet, 2000)

- ▶ **glNormal***
- ▶ **glEnable, glDisable** mit
GL_NORMALIZE,
GL_RESCALE_NORMAL

```
gl.glBegin(GL.GL_QUADS);  
gl.glNormal3f(0.0f, 1.0f, 0.0f);  
gl.glVertex3f(-1.0f, 0.0f, 1.0f);  
gl.glVertex3f( 1.0f, 0.0f, 1.0f);  
gl.glVertex3f( 1.0f, 0.0f,-1.0f);  
gl.glVertex3f(-1.0f, 0.0f,-1.0f);  
gl.glEnd();
```

- ▶ Konvention: Normalen zeigen nach aussen der modellierten Oberfläche

Aktivierung von Beleuchtungsrechnung und Lichtquellen

- ▶ Beleuchtungsrechnung ein-, ausschalten:
glEnable(GL_LIGHTING), **glDisable**(GL_LIGHTING)
- ▶ Lichtquelle (Nr. 0) ein-, ausschalten:
glEnable(GL_LIGHT0), **glDisable**(GL_LIGHT0),
...

Richtungslichtquelle (d.h. $w=0$) - Beispiel

```
float light0_ambient[] = {0.3f, 0.3f, 0.3f, 1.0f};
float light0_diffuse[] = {0.7f, 0.7f, 0.7f, 1.0f};
float light0_specular[] = {0.7f, 0.7f, 0.7f, 1.0f};
float light0_position[] = {0.0f, 1.0f, 2.0f, 0.0f};

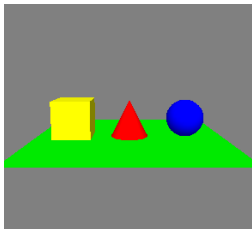
public void init(GLAutoDrawable drawable) {
    ...

    // enable lighting (default is off)
    gl.glEnable(GL.GL_LIGHTING);

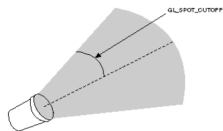
    // set up and enable light 0
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_AMBIENT, light0_ambient, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_DIFFUSE, light0_diffuse, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_SPECULAR, light0_specular, 0);
    gl.glLightfv(GL.GL_LIGHT0, GL.GL_POSITION, light0_position, 0);

    gl.glEnable(GL.GL_LIGHT0);
}

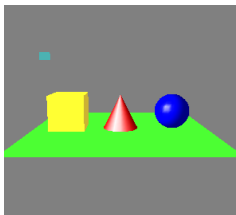
...
```



Spotlights (Scheinwerfer), Abschwächungsfaktoren



<http://www.glprogramming.com/red/chapter05.html>



Eigenschaft name	Standard-Werte param	Bedeutung
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	($\mathbf{k} = (k_x, k_y, k_z)^T$)-Richtung des Scheinwerferkegels
GL_SPOT_CUTOFF	180.0	halber Öffnungswinkel des Scheinwerferkegels ($Spot_{Cutoff}$)
GL_SPOT_EXPONENT	0.0	Abschwächungsfaktor senkrecht zur Strahlrichtung ($Spot_{Exp}$)
GL_CONSTANT_ATTENUATION	1.0	konstanter Abschwächungsfaktor f_c
GL_LINEAR_ATTENUATION	0.0	linearer Abschwächungsfaktor f_l
GL_QUADRATIC_ATTENUATION	0.0	quadratischer Abschwächungsfaktor f_q

(Nischwitz et al., 2011)

Spotlight (d.h. $w=1$) - Beispiel

```
float light1_ambient[] = {0.3f, 0.3f, 0.3f, 1.0f};
float light1_diffuse[] = {0.7f, 0.7f, 0.7f, 1.0f};
float light1_specular[] = {0.7f, 0.7f, 0.7f, 1.0f};
float light1_position[] = {0.0f, 1.0f, 2.0f, 1.0f};
float light1_direction[] = {0.0f, 0.0f, -1.0f};
float light1_cutoff = 12.0f;
float light1_exponent = 2.0f;

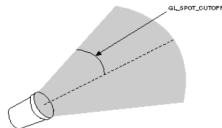
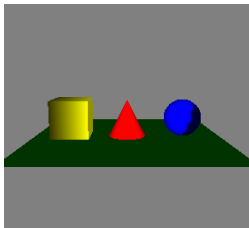
public void init(GLAutoDrawable drawable) {
    ...
    // enable lighting (default is off)
    gl.glEnable(GL.GL_LIGHTING);

    // set up and enable light 1
    gl.glLightfv(GL.GL_LIGHT1, GL.GL_AMBIENT, light1_ambient, 0);
    gl.glLightfv(GL.GL_LIGHT1, GL.GL_DIFFUSE, light1_diffuse, 0);
    gl.glLightfv(GL.GL_LIGHT1, GL.GL_SPECULAR, light1_specular, 0);
    gl.glLightfv(GL.GL_LIGHT1, GL.GL_POSITION, light1_position, 0);

    gl.glLightf(GL.GL_LIGHT1, GL.GL_SPOT_CUTOFF, light1_cutoff);
    gl.glLightf(GL.GL_LIGHT1, GL.GL_SPOT_EXPONENT, light1_exponent);
    gl.glLightfv(GL.GL_LIGHT1, GL.GL_SPOT_DIRECTION, light1_direction, 0);

    //gl.glLightf(GL.GL_LIGHT1, GL.GL_CONSTANT_ATTENUATION, 1.5f);
    //gl.glLightf(GL.GL_LIGHT1, GL.GL_LINEAR_ATTENUATION, 0.1f); //
    gl.glLightf(GL.GL_LIGHT1, GL.GL_QUADRATIC_ATTENUATION, 0.0f);

    gl.glEnable(GL.GL_LIGHT1);
}
```



Lighting model

- ▶ **glLightModel{if}**(int *pname*, TYPE *param*)
- ▶ **glLightModel{if}v**(int *pname*, TYPE[] *params*, int *params_offset*)

Eigenschaft name	Standard-Werte von param	Bedeutung
GL_LIGHT_MODEL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	RGBA-Intensitäten für den globalen ambienten Lichtanteil (\mathbf{a}_{global})
GL_LIGHT_MODEL_LOCAL_VIEWER	0.0 oder GL_FALSE	unendlich entfernter oder lokaler Augenspunkt, relevant für die Berechnung des spekularen Lichtanteils
GL_LIGHT_MODEL_TWO_SIDE	0.0 oder GL_FALSE	Beleuchtungsrechnung nur bezogen auf die Vorderseiten (einseitig), oder getrennt für Vorder- und Rückseite (zweiseitig)
GL_LIGHT_MODEL_COLOR_CONTROL	GL_SINGLE_COLOR	Alle Beleuchtungskomponenten in einer Farbe oder separate Berechnung des spekularen Farbanteils

(Nischwitz *et al.*, 2011)

Materialeigenschaften (1) - beim `glDisable(GL_COLOR_MATERIAL)`

- ▶ `glMaterial{if}(int pname, int face, TYPE param)`
- ▶ `glMaterial{if}v(int pname, int face, TYPE[] params, int params_offset) face - GL_FRONT, GL_BACK, GL_FRONT_AND_BACK`

Eigenschaft name	Standard-Werte von param	Bedeutung
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	RGBA-Werte für die ambiente Reflexion. Standardwert dunkelgrau
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	RGBA-Werte für die diffuse Reflexion. Standardwert hellgrau
GL_AMBIENT_AND_DIFFUSE		Die ambienten und diffusen Reflexionskoeffizienten werden auf die gleichen RGBA-Werte gesetzt
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	RGBA-Werte für die spekulare Reflexion. Standardwert schwarz
GL_SHININESS	0.0	Spiegelungsexponent S
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	Emissive Farbe des Materials. Standardwert schwarz

(Nischwitz et al., 2011)

Materialeigenschaften (2)

- ▶ **glEnable**(GL_COLOR_MATERIAL)
glColorMaterial(int *face*, int *mode*)
face - GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
mode - GL_AMBIENT, GL_DIFFUSE, GL_AMBIENT_AND_DIFFUSE,
GL_SPECULAR, GL_EMISSION
- ▶ Die aktuelle Farbe wird mit **glColor** festgelegt