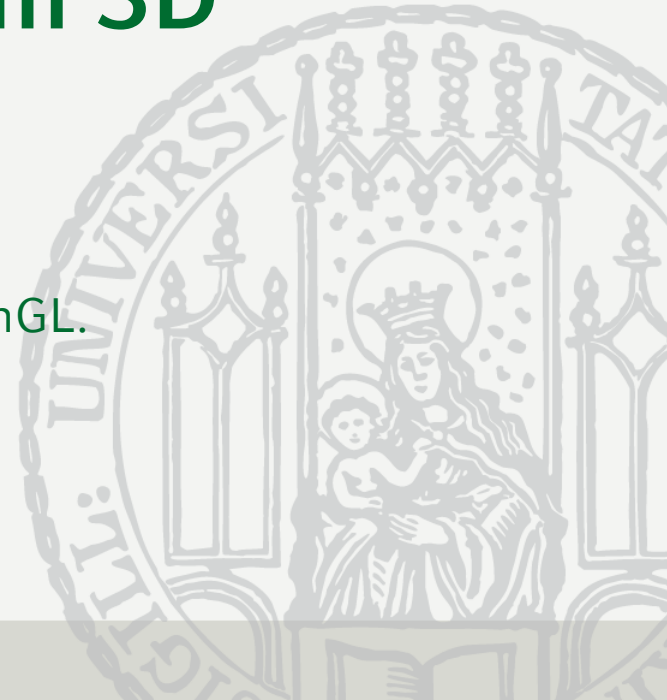


Prof. Andreas Butz | Dipl.Inf. Otmar Hilliges

# Programmierpraktikum 3D Computer Grafik

Grundlagen der Computergrafik:

Affine Transformationen, Beleuchtung in OpenGL.



- Bearbeitungszeitraum für aktuelles Übungsblatt sind 14 Tage! (Abgabe 29.Mai)
- Erster Teil mit Wissen aus dieser Stunde machbar.
- Für zweiten Teil ist nächste Woche nützlich (aber nicht notwendig).
- Ab Ende Mai beginnt die Projektphase!
  - Teambildung sollte nächste Woche erledigt sein.
  - Überlegungen für Projektideen dürfen angestellt werden ;-)
- Präsentation der Projekte am 30. Mai
- Am 13. Juni muss erste Version des Projektes bereits laufen.

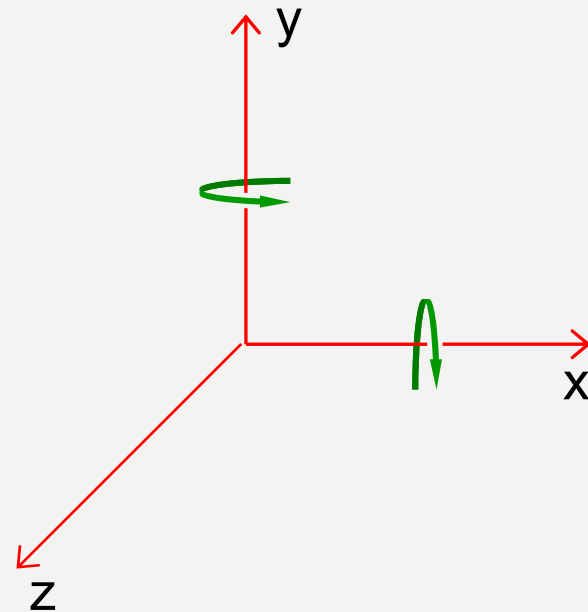
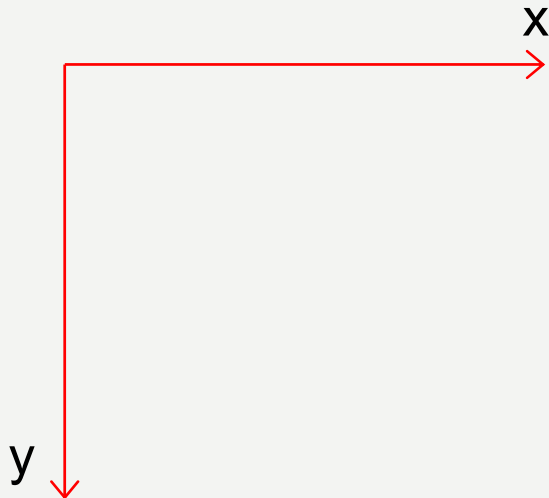


- Geometrische Transformationen in 3D
  - Koordinatensysteme
  - Translation, Rotation und Skalierung
  - Homogene Koordinaten und Transformationsmatrix
  - Projection- und Modelview-Matrix und Matrixoperationen in OpenGL
- Farben und Materialeigenschaften
- Beleuchtung
- Blending

## ■ Koordinatensysteme

- Im allgemeinen frei wählbar
- Meistens *rechtsdrehend* und *rechtwinklig*

## ■ Beispiele aus der Computergraphik:





■ Koordinatensysteme werden aufgespannt durch Ursprung und Basis

$$K = B, (b_1, b_2)$$

$$K' = B', (b_1', b_2')$$

y

Beispiel:

$$B = (0,0)^{T,K} = \left(\frac{3}{2}, \frac{3}{4}\right)^{T,K'} \dots B' = (0,0)^{T,K'}$$

$$b_1 = (1,0)^{T,K} = \left(\frac{1}{2}, \frac{1}{8}\right)^{T,K'} \dots b_1' = (1,0)^{T,K'}$$

$$b_2 = (0,1)^{T,K} = \left(\frac{1}{4}, 1\right)^{T,K'} \dots b_2' = (0,1)^{T,K'}$$

Gegeben:

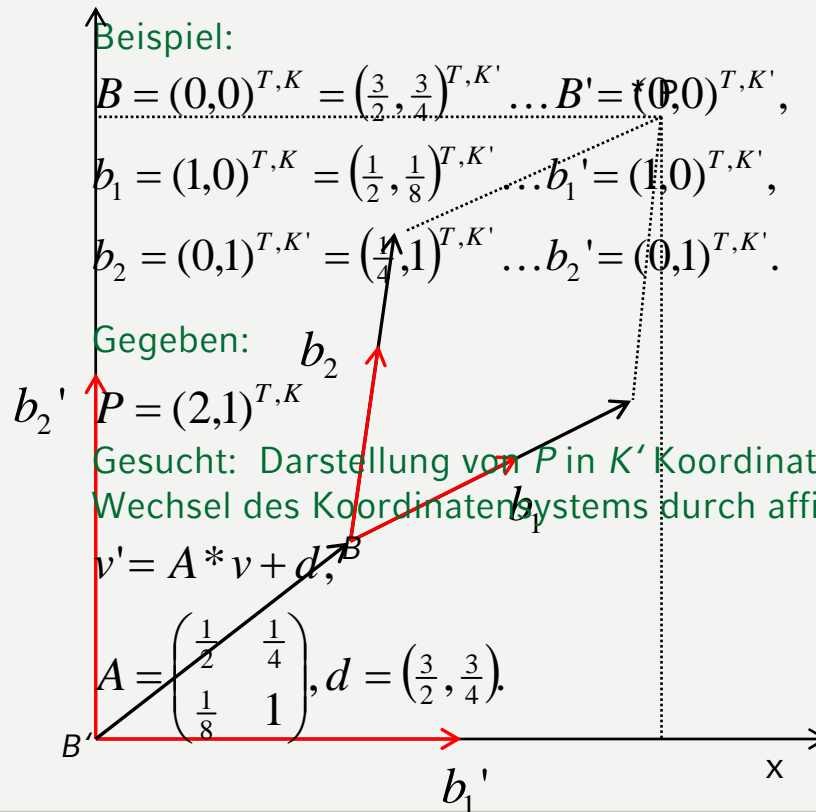
$$P = (2,1)^{T,K}$$

Gesucht: Darstellung von P in K' Koordinaten?

Wechsel des Koordinatensystems durch affine Transformation:

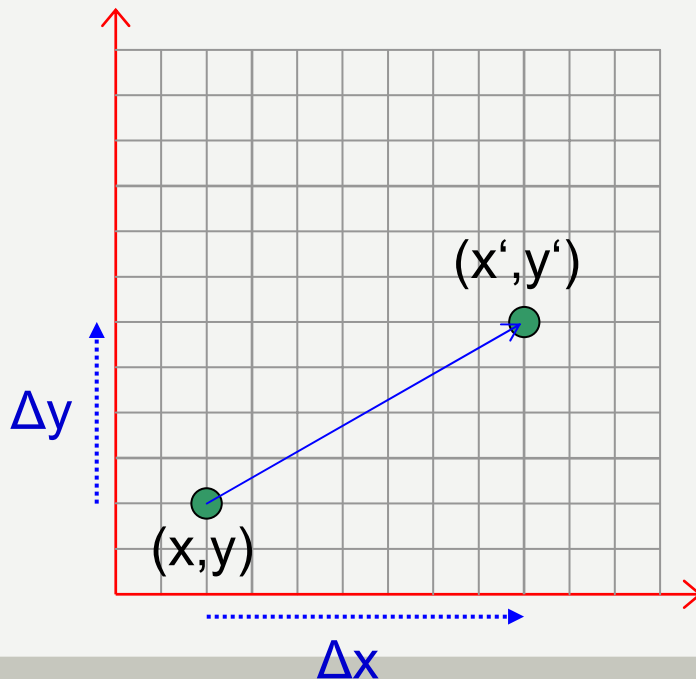
$$v' = A * v + d,$$

$$A = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} \\ \frac{1}{8} & 1 \end{pmatrix}, d = \left(\frac{3}{2}, \frac{3}{4}\right).$$



## Translation:

- Verschiebung eines Punktes auf einen anderen
- Beschreibung der Translation:

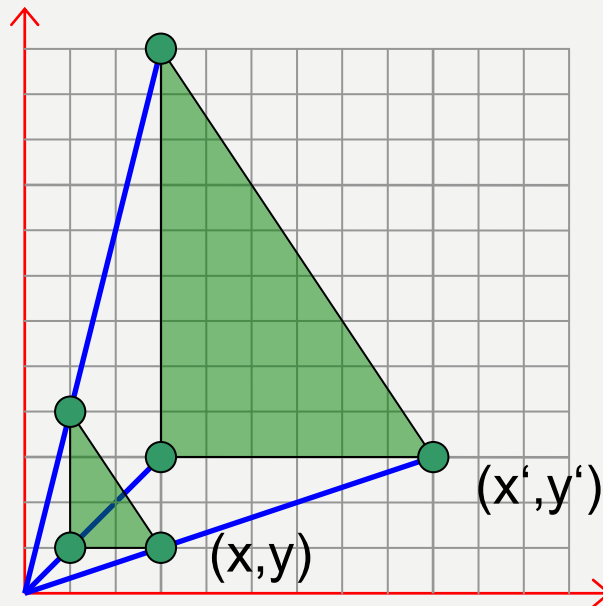


$$A = Id,$$

$$v' = v + d$$

## Skalierung (uniform):

- Streckung in alle Richtungen mit dem Ursprung als Zentrum um  $\alpha$
- Beschreibung der Skalierung:



$$A = \begin{pmatrix} a_{1,1} & 0 \\ 0 & a_{1,1} \end{pmatrix},$$

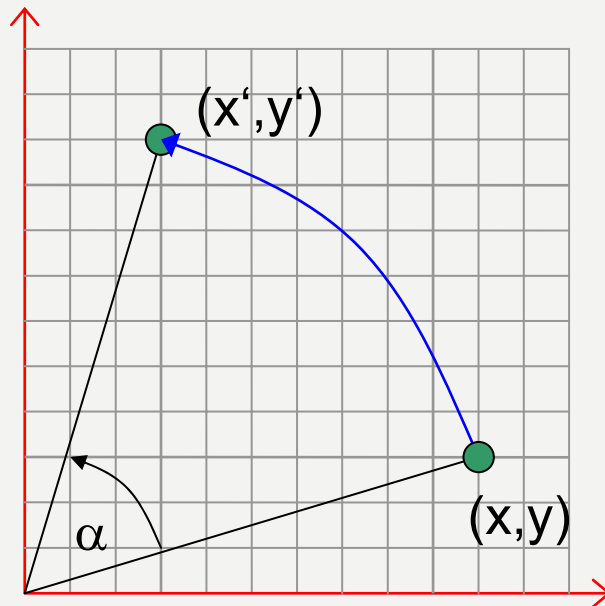
$$d = 0.$$

Spezialfälle:

- Streckung:  $a_{1,1}=a_{2,2} > 0 \Rightarrow$  Zoom
- Spiegelung  $a_{1,1}=a_{2,2} = -1$

## Rotation:

- Drehung eines Punktes um den Ursprung mit dem Winkel  $\alpha$  ( $\alpha > 0 \rightarrow$  CCW)
- Beschreibung der Rotation (2D):



$$A = \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix},$$
$$d = 0$$





## Bisher:

- Translation: Addition eines Vektors
- Skalierung, Rotation : Multiplikation des Faktors

## Problem:

- Keine einheitliche Behandlung
- Zusammengesetzte Transformationen nur schwer zu realisieren
- Umkehrung von verketteten Transformationen nur schwer möglich.



- Jeder Punkt  $P$  ist eindeutig Darstellbar durch:

$$P = (\beta_1 * b_1 + \beta_2 * b_2 + 1 * B),$$

bzw.

$$P' = (\beta_1' * b_1' + \beta_2' * b_2' + 1 * B')$$

- Das Tripel  $(\beta_1, \beta_2, 1)$  ist eine Darstellung von  $P$  in homogenen Koordinaten.
- Die Affine Transformation lässt sich dann so darstellen:

$$\begin{pmatrix} v' \\ 1 \end{pmatrix} = \begin{pmatrix} A & d \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} v \\ 1 \end{pmatrix}$$



- Hintereinanderausführen von affinen Transformationen ergibt wieder affine Transformation:

$$v' = A_1 * v + d_1,$$

$$v'' = A_2 * v' + d_2$$

$$= A_2 * A_1 * v + (A_2 * d_1 + d_2)$$

- In homogenen Koordinaten:

$$\begin{pmatrix} v' \\ 1 \end{pmatrix} = \begin{pmatrix} A_1 & d_1 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} v \\ 1 \end{pmatrix},$$

$$\begin{pmatrix} v'' \\ 1 \end{pmatrix} = \begin{pmatrix} A_2 & d_2 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} v' \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} A_2 & d_2 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} A_1 & d_1 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} v \\ 1 \end{pmatrix}$$

- Komplizierte Transformationen lassen sich zusammensetzen
  - Drehung um *beliebigen* Punkt P
  - Skalierung mit Bezug auf *beliebigen* Punkt P
- Transformation wird akkumuliert
  - Transformationsmatrix wird im vornherein berechnet
  - Anwendung auf alle Punkte auf einmal
- Affinität erlaubt das nur die Punkte transformiert werden müssen
  - Geraden bleiben gerade
  - Parallele Kanten bleiben parallel
  - Linien und Flächen sind durch Punkte eindeutig bestimmt
  - Transformierte Objekte sind isometrisch (ähnlich) zum Original



Überführung eines Koordinatensystems in ein  
homogenes Koordinatensystem durch Hinzufügen  
einer weiteren Dimension (Projektion des  $\mathbb{R}^3$  in  $\mathbb{R}^4$ )

- Repräsentation eines 3D-Punktes in homogenen  
Koordinaten:

$$p = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \cong \begin{pmatrix} \lambda \cdot x \\ \lambda \cdot y \\ \lambda \cdot z \\ \lambda \end{pmatrix}, \lambda \neq 0$$



- Rotation (in 3D):
- Drei verschiedene Matrizen für die Drehung eines Punktes um eine Achse

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Drehung um  
die x-Achse

$$\begin{pmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Drehung um  
die y-Achse

$$\begin{pmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Drehung um  
die z-Achse



## Die allgemeine Transformationsmatrix:

$$T = \begin{pmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \end{pmatrix}$$

- Skalierung
- Rotation
- Translation



## OpenGL verwendet zwei Matrizen:

- Modelview-Matrix:
  - Transformiert die Objektkoordinaten
  - Erzeugt Kamerakoordinaten
  - Beschreibt die Beziehung zwischen Objekt und Kamera (als Transformationsmatrix)
- Projection-Matrix:
  - Transformiert die Kamerakoordinaten
  - Erzeugt Schnittkoordinaten
- Schnittkoordinaten werden mit einem Faktor letztlich auf den Bildschirm übertragen





## Matrixoperationen in OpenGL:

- Die Wahl der Matrix:

```
void glMatrixMode(GLenum mode);  
// mode = GL_MODELVIEW, GL_PROJECTION  
oder GL_TEXTURE
```

- Laden der Einheitsmatrix:

- Setzt den Ursprung an die Position des Betrachters (also der Kamera)

```
void glLoadIdentity();
```



## Matrixoperationen in OpenGL:

- Laden einer Matrix:
  - Matrizen sind 1D repräsentiert, d.h. bei einer 4x4-Matrix handelt es sich um ein Feld der Länge 16
  - Matrizen werden "row-major" behandelt, d.h. die ersten vier Elemente bilden die erste Spalte, usw.

```
void glLoadMatrixf(GLfloat* mp);
```

- Multiplikation der aktuellen Matrix:

```
void glMultMatrixf(GLfloat* mp);
```



## Matrixoperationen in OpenGL:

- Der Matrix-Stack:
  - OpenGL besitzt für jede Matrix einen Stack
  - Matrizen können “gesichert” werden, um sie nach weiteren Transformationen wieder laden zu können
  - Ohne Stack müssten alle Transformationen rückgängig gemacht werden (→ hoher Aufwand)
  - Der Modelview-Stack kann mindestens 32, die beiden anderen können mindestens 2 Matrizen aufnehmen

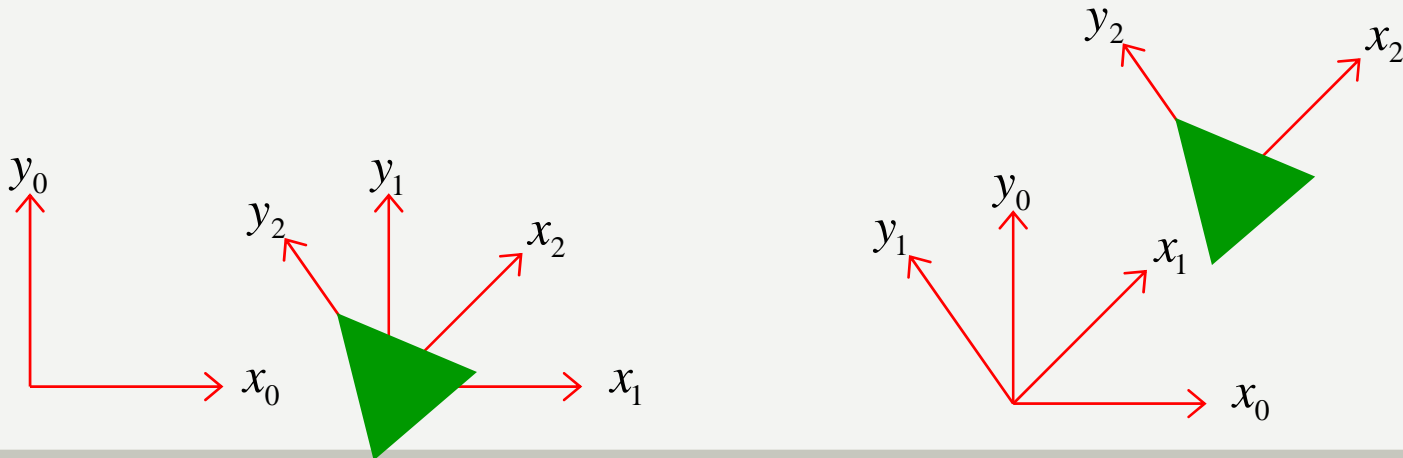


## Matrixoperationen in OpenGL:

- Der Matrix-Stack:
  - Speichern einer Matrix auf dem Stack:  
`void glPushMatrix();`
  - Laden einer Matrix vom Stack:  
`void glPopMatrix();`
  - Push- und Pop-Operationen müssen sich **nicht** abwechseln, da die Stacks eine gewisse Tiefe zur Verfügung stellen

## Matrixoperationen in OpenGL:

- Der Matrix-Stack:
  - Die Aufrufe zu `glTranslatef(...)`, `glRotatef(...)` und `glScalef(...)` führen lediglich eine Multiplikation der aktuellen Transformationsmatrix durch
  - Die Reihenfolge der Multiplikationen wird durch den Programmierer festgelegt
  - **Achtung:** Es ist entscheidend, ob erst eine Translation gefolgt von einer Rotation durchgeführt wird, oder umgekehrt!





## Beispiel:

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glTranslatef(0.0f, 0.0f, -10.0f); // pos(1)  
glPushMatrix(); // speichern von pos(1)  
glTranslatef(2.0f, 0.0f, 0.0f);  
glRotatef(90.0f, 1.0f, 0.0f, 0.0f);  
// zeichne etwas  
glPopMatrix(); // zurück zu pos(1)  
glTranslatef(-2.0f, 0.0f, 0.0f);  
glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);  
// zeichne etwas
```



## Weitere Funktionen für Matrizen:

- Laden der aktuellen Matrix:

```
void glGetFloatv(GLenum pname,  
                 GLfloat* params);  
  
pname = GL_MODELVIEW_MATRIX,  
        GL_PROJECTION_MATRIX,  
        GL_TEXTURE_MATRIX  
  
params = Feld (16 Elemente, "row-major")
```



# Materialeigenschaften bedeutet (noch nicht) Texturen

## Eigenschaften eines Materials:

- Ambiente Farbe
- Diffuse Farbe
- Spiegelungsfarbe
- Stärke des Glanzpunktes
- Aussendende Farbe





## Setzen von (single-value) Materialeigenschaften in OpenGL:

```
void glMaterialf(GLenum face,
                GLenum pname,
                GLfloat param);

// face = GL_FRONT, GL_BACK oder
           GL_FRONT_AND_BACK

// pname = GL_SHININESS

// param = Wert für GL_SHININESS
```



## Setzen von (multi-value) Materialeigenschaften in OpenGL:

```
void glMaterialfv(GLenum face,
                 GLenum pname,
                 GLfloat* param);

// face = wie vorher
// pname = GL_AMBIENT, GL_DIFFUSE,
          GL_SPECULAR, GL_EMISSION,
          GL_COLOR_INDEXES
// param = Wert für pname (4D)
```

## Übersicht der Werte (+ Default-Werte):

Parameter	Default	Bedeutung
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	Ambiente Farbe (allgemeine Farbe)
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	Diffuse Farbe (Farbe durch Licht)
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	Spiegelungsfarbe (abhängig von Pos.)
GL_SHININESS	0.0	Glanzpunktstärke (je höher, desto heller)
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	Farbe des ausgesandten Lichts
GL_COLOR_INDEXES	(0, 1, 1)	Ambient-, Diffus-, Spiegelungsindex

## Beispiele für Materialien

Material	$r_{ar}, r_{ag}, r_{ab}$	$r_{dr}, r_{dg}, r_{db}$	$r_{sr}, r_{sg}, r_{sb}$	n
Plastik	0.0, 0.0, 0.0	0.01, 0.01, 0.01	0.50, 0.50, 0.50	32
Messing	0.33, 0.22, 0.03	0.78, 0.57, 0.11	0.99, 0.94, 0.81	28
Bronze	0.21, 0.13, 0.05	0.71, 0.43, 0.18	0.39, 0.27, 0.17	26
Kupfer	0.19, 0.07, 0.02	0.70, 0.27, 0.08	0.26, 0.14, 0.09	13
Gold	0.25, 0.20, 0.07	0.75, 0.61, 0.23	0.63, 0.56, 0.37	51
Silber	0.19, 0.19, 0.19	0.51, 0.51, 0.51	0.51, 0.51, 0.51	51



## Verschiedene Lichtquellen:

- *Punktlichtquelle*: nur Position (strahlt in alle Richtungen gleichmäßig)
- *Gerichtetes Licht*: nur Vektor (unendlich weit entfernte Lichtquelle (z.B. Sonne))
- *Spotlight*: Position, Richtungsvektor, Öffnungswinkel und Intensitätsabfall (Punktlichtquelle mit bestimmtem Öffnungskegel)



## Eigenschaften von Lichtquellen:

- Position
- Richtungsvektor (für Spotlights)
- Öffnungswinkel (für Spotlights)
- Ambiente Farbe
- Diffuse Farbe
- Spiegelungsfarbe
- Intensitätsabfall
- Radialer Intensitätsabfall (für Spotlights)



## Setzen von (single-value) Lichteigenschaften in OpenGL:

```
void glLightf(GLenum light,  
             GLenum pname,  
             GLfloat param);  
  
// light = GL_LIGHTi  
// pname = GL_SPOT_EXPONENT, GL_SPOT_CUTOFF,  
          GL_CONSTANT_ATTENUATION,  
          GL_LINEAR_ATTENUATION,  
          GL_QUADRATIC_ATTENUATION  
// param = Wert für pname
```



## Setzen von (multi-value) Lichteigenschaften in OpenGL:

```
void glLightfv(GLenum light,  
              GLenum pname,  
              GLfloat* param);  
  
// light = GL_LIGHTi  
// pname = GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR,  
          GL_POSITION, GL_SPOT_DIRECTION  
// Achtung: vierter Wert = 1 für Position und 0 für Richtung (GL_POSITION vs.  
          GL_SPOT_DIRECTION)  
// param = Wert für pname (4D)
```



## Übersicht der Werte (+ Default-Werte):

Parameter	Default	Bedeutung
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	Ambiente Farbe des Lichts
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	Diffuse Farbe des Lichts
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	Spiegelungsfarbe
GL_POSITION	(0.0, 0.0, 1.0, 1.0)	Position
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0, 0.0)	Richtung (Spotlight)
GL_SPOT_EXPONENT	0.0	Radiale Intensität (Spotlight)

## Übersicht der Werte (+ Default-Werte):

Parameter	Default	Bedeutung
GL_SPOT_CUTOFF	180.0	Halber Öffnungswinkel (Spotlight)
GL_CONSTANT_ATTENUATION	1.0	Konstanter Abschwächungsfaktor
GL_LINEAR_ATTENUATION	0.0	Linearer Abschwächungsfaktor
GL_QUADRATIC_ATTENUATION	0.0	Quadratischer Abschwächungsfaktor



## Ein- und Ausschalten der Lichtquellen:

- Globales (De-)Aktivieren:

```
void glEnable(GL_LIGHTING);
```

```
void glDisable(GL_LIGHTING);
```

- Danach: Einzelne Lichter (de-)aktivieren:

```
void glEnable(GL_LIGHTi);
```

```
void glDisable(GL_LIGHTi);
```

- Einzelne Lichter können während dem Aufbau der Szene an- und ausgeschaltet werden (Beleuchtung einzelner Objekte verhindern)



Möglichkeit, Objekte (semi-)transparent darstellen zu können

Zunächst muss eine Blend-Funktion ausgewählt werden:

```
void glBlendFunc(GLenum src, GLenum dst);  
src, dst = GL_ZERO, GL_ONE, GL_DST_COLOR,  
           GL_ONE_MINUS_DST_COLOR,  
           GL_SRC_ALPHA,  
           GL_ONE_MINUS_SRC_COLOR,  
           GL_DST_ALPHA,  
           GL_ONE_MINUS_DST_ALPHA,  
           GL_SRC_ALPHA_SATURATE
```



## Aktivieren des Blendings:

- Deaktivieren des Tiefentests (und evtl. Beleuchtung):

```
glDisable(GL_DEPTH_TEST);
```

- Aktivieren des Blendings:

```
glEnable(GL_BLEND);
```

- Setzen einer Vertexfarbe mit 4 Komponenten:

```
glColor4f(0.0f, 1.0f, 0.0f, 0.5f);
```



- Wiederherstellen der Szenen-eigenschaften durch Deaktivierung des Blendings und Aktivierung des Tiefentests (und evtl. der Beleuchtung)
- Blending kann für einzelne Objekte während dem Aufbau der Szene (de-) aktiviert werden (analog zur Beleuchtung)