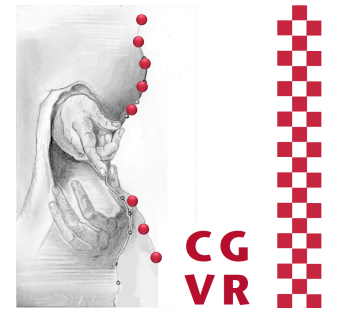
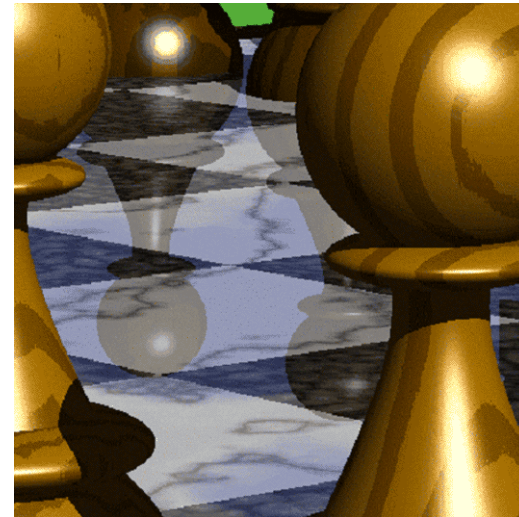


Bremen



# Computer-Graphik 1

## Lighting & Shading



G. Zachmann

University of Bremen, Germany

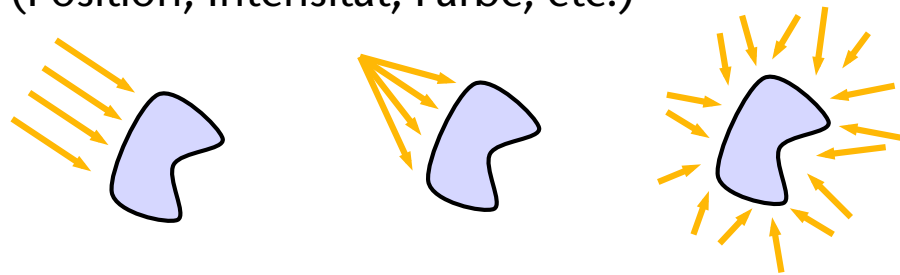
[cgvr.informatik.uni-bremen.de](http://cgvr.informatik.uni-bremen.de)



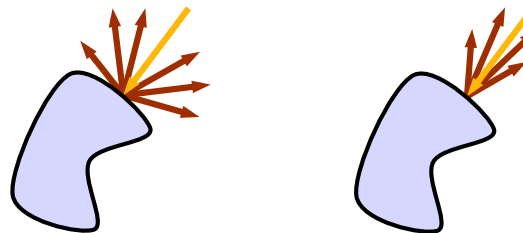
# Beleuchtungsmodelle (*Lighting Models*)

- Definition "**Beleuchtungsmodell**": Vorschrift zur Berechnung der **Farb- und Helligkeitswerte** an Punkten auf der Oberfläche von Objekten
  - Grundlage sind physikalische Gesetze
  - Modelliert werden Einflüsse von:

1. **Lichtquellen** (Position, Intensität, Farbe, etc.)

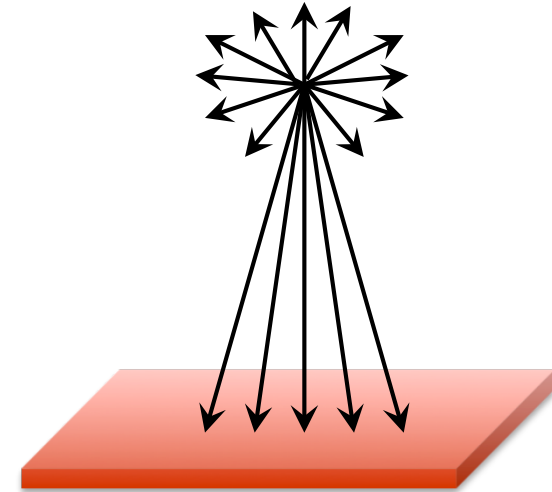


2. **Objektoberfläche** (Geometrie, Reflexionseigenschaften)

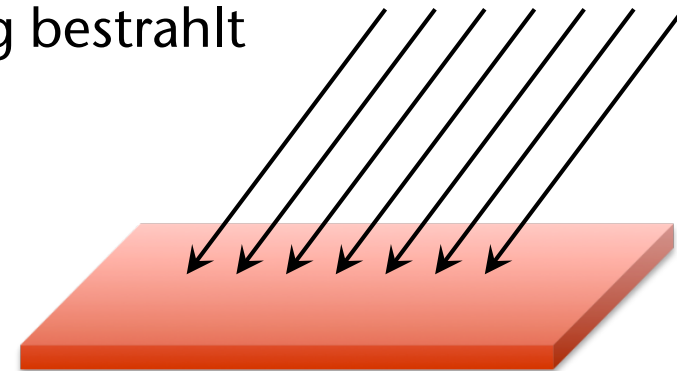


- Für Echtzeitanwendungen verwendet man sehr einfache Modelle

- **Punktlichtquelle** (*point light*): strahlt in alle Richtungen gleichmäßig ab
  - Wird eindeutig charakterisiert durch
    1. **Position** &
    2.  $I(\lambda)$  = abgestrahltes Spektrum  
 = Intensität abhängig von der Wellenlänge

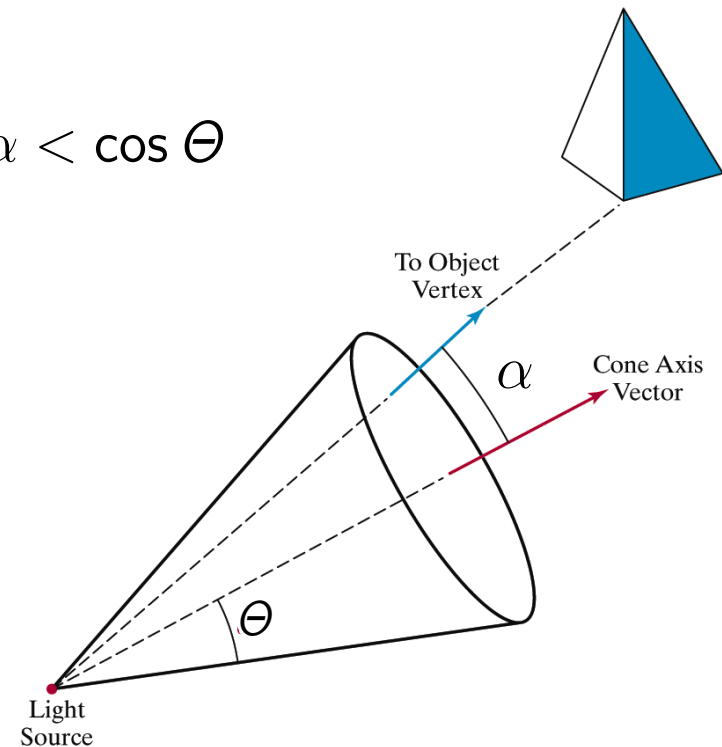
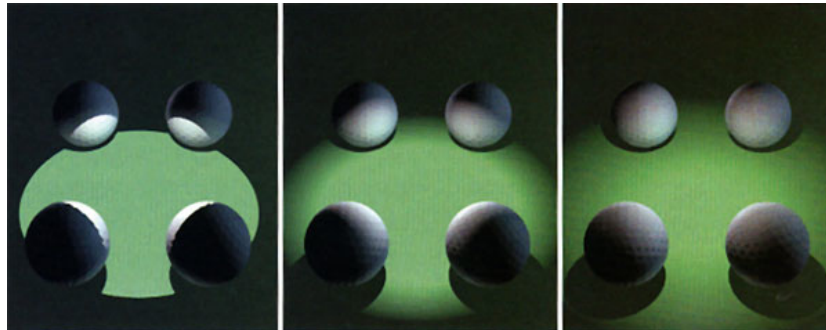


- **Richtungslichtquelle** (*directional light*): jeder Punkt im Raum wird aus derselben Richtung bestrahlt
  - Charakterisiert durch **Richtung** &  $I(\lambda)$
  - Beispiel: Sonne



- Strahler (spot light):** Lichtausbreitung wird auf einen bestimmten Raumwinkel (Lichtkegel) beschränkt. Der Abfall der Lichtstärke von der Kegelachse zum Rand wird durch folgendes Gesetz bestimmt:

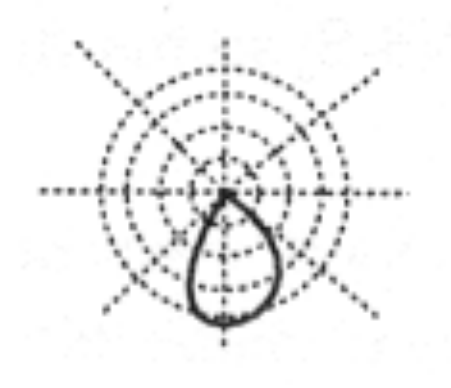
$$I(\lambda) = \begin{cases} 0 & \text{falls } \cos \alpha < \cos \Theta \\ I_0(\lambda) \cos^n \alpha & \text{sonst} \end{cases}$$





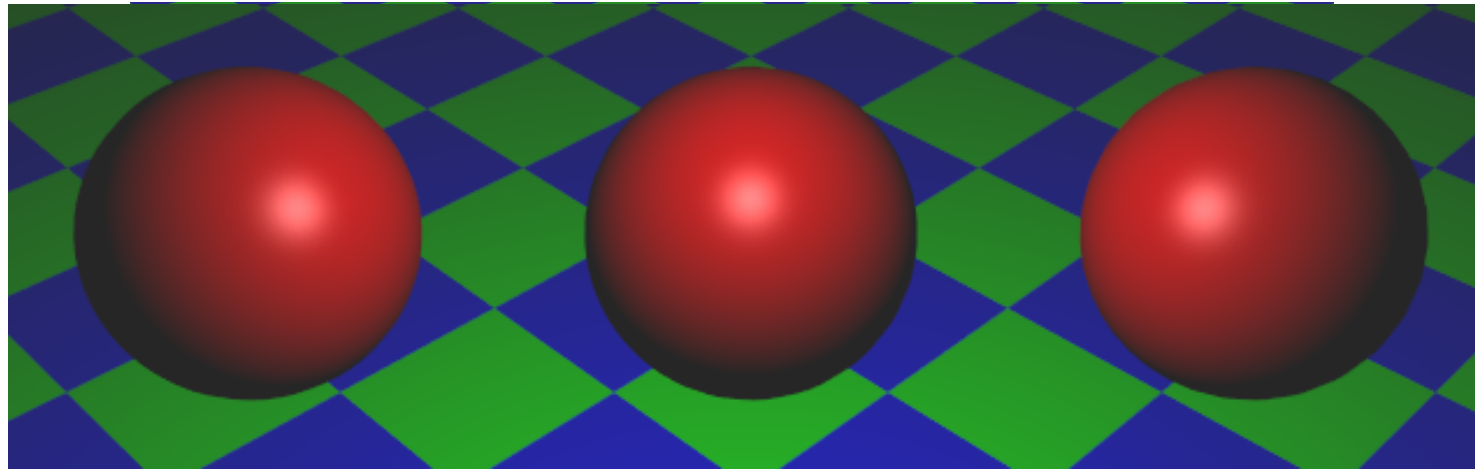
- **Goniometrische Lichtquelle:** Abstrahlcharakteristik wird per Tabelle beschrieben. Zur Ermittlung von  $I(\lambda)$  muß evtl. zwischen Tabelleneinträgen interpoliert werden

- **Area light source:**

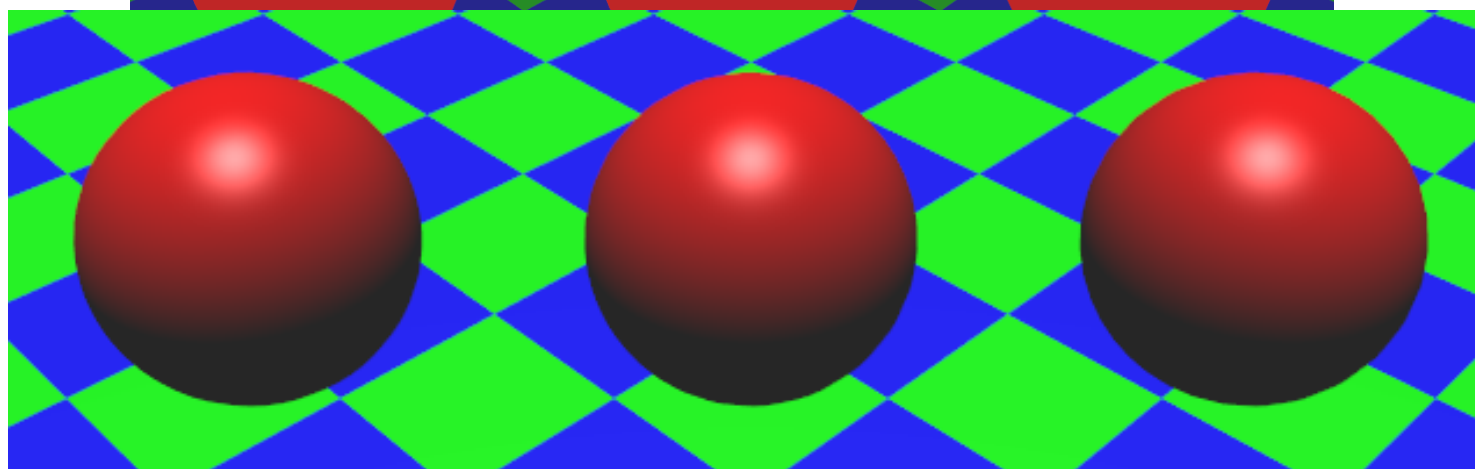


# Unterschiedlicher Effekt zwischen *point light* und *directional light*

- Wie erkennt man anhand der Beleuchtung einer Kugel, von welcher Art die Lichtquelle ist? (point oder directional?)



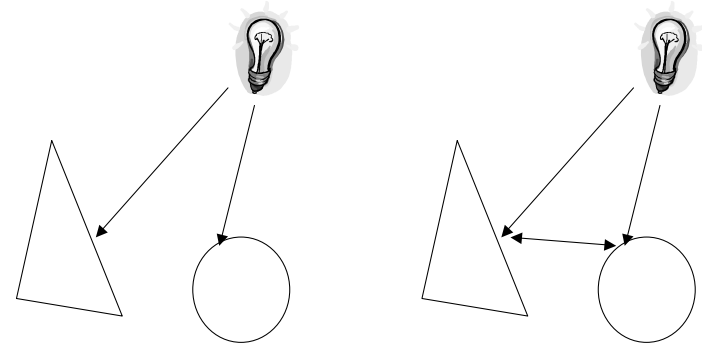
Point light source



Directional light source

# Lokale Beleuchtungsmodelle

- Vereinfachung: berücksichtige bei der Berechnung der Beleuchtung eines Punktes **keine sekundären Effekte** (Strahlungs-austausch zwischen Objekten), **nur primären** Austausch zwischen Lichtquelle und Objekt → **lokales Beleuchtungsmodell**



- Superpositionsprinzip**: betrachte Licht als Teilchen →

$$I(\lambda) = \sum_j I_j(\lambda)$$

- Vereinfachung der Notation: wir lassen im Folgenden  $\lambda$  überall weg, und merken uns, daß alle photometrischen Größen eigtl. von  $\lambda$  abhängen!

# Warum ist Lighting & Shading so wichtig?

- Die "Shutterbug"-Szene ganz ohne Shading ...



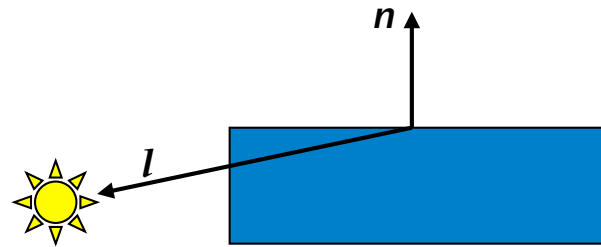
Pixar "Shutterbug"



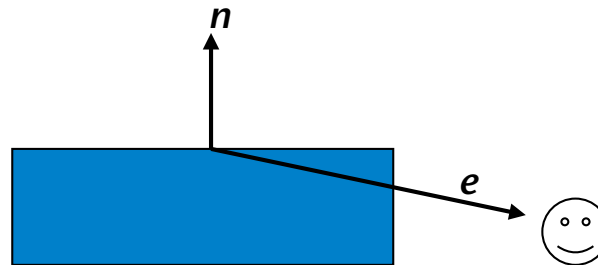
Impossible Rooftops, by Kokichi Sugihara

# Eine Konvention bzgl. negativer Skalarprodukte

- Falls  $\mathbf{n} \cdot \mathbf{l} < 0$ , dann befindet sich das Licht hinter der Fläche



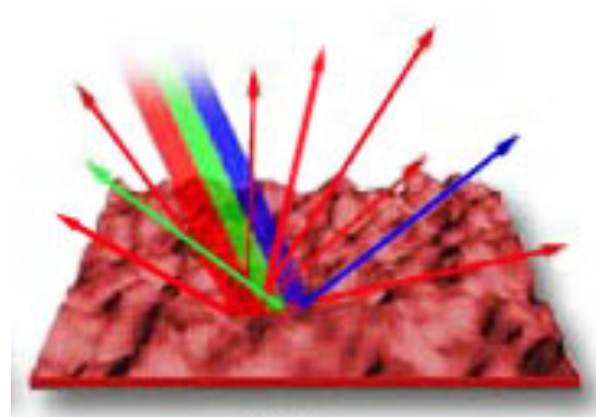
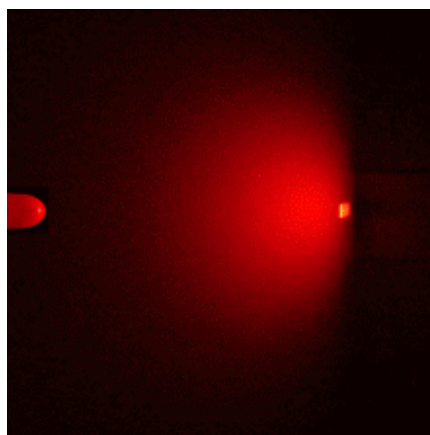
- Falls  $\mathbf{n} \cdot \mathbf{e} < 0$ , so befindet sich der Viewpoint auf der Rückseite der Fläche



- Wir definieren im Folgenden (der Einfachheit halber) prinzipiell:

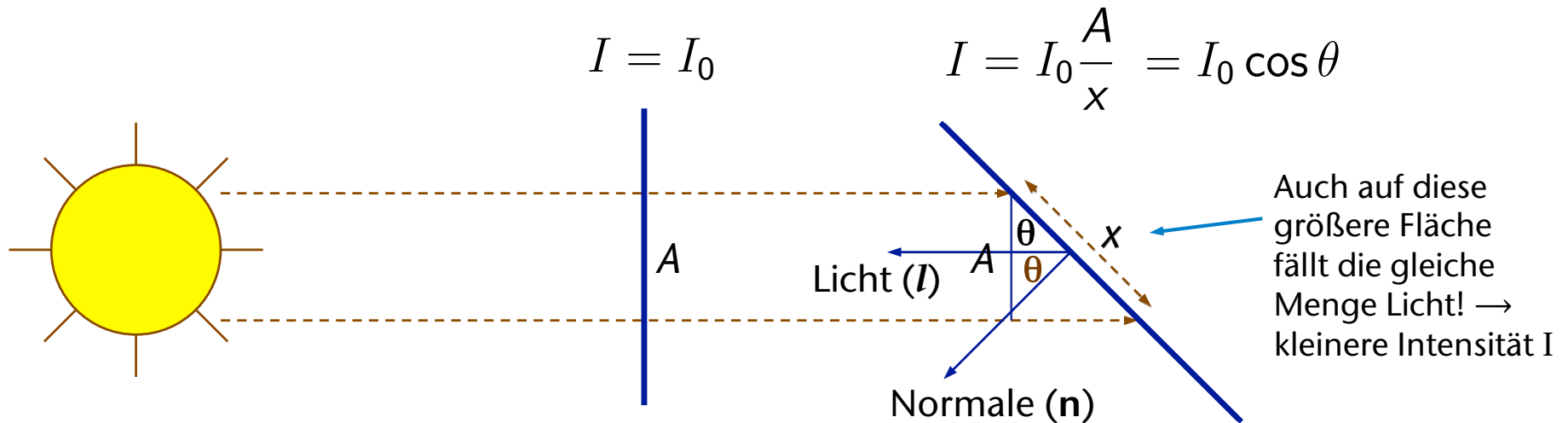
$$\mathbf{n} \cdot \mathbf{l} := \max(0, \mathbf{n} \cdot \mathbf{l})$$

- Licht wird von der Objekt Oberfläche gleichmäßig in alle Richtungen reflektiert
- Folge: Helligkeit ist unabhängig vom Viewpoint!
- Beispiele: Stück Papier, Tafel, unbearbeitetes Holz
- **Diffuse/Matte** Objekte werden auch **Lambert'sche** Objekte bezeichnet



- Das Lambert'schen Kosinus-Gesetz:

$$I = I_0 \cos \theta = I_0 \cdot \mathbf{n} \cdot \mathbf{l} \quad \text{Annahme: } \mathbf{n} \text{ und } \mathbf{l} \text{ sind Einheitsvektoren}$$



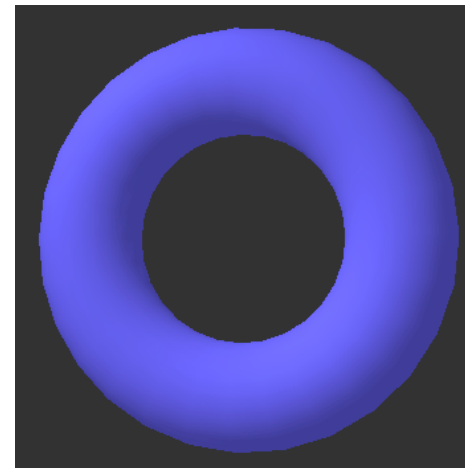


- Das Lambert'sche Modell erzeugt schwarze Farbe für Oberflächen, die nicht zur Lichtquelle zeigen
- In der Realität trifft Licht aus allen Richtungen ein (dieses wurde von anderen Objekten, evtl. mehrfach, reflektiert)
- Füge für alle Objekte einen **ambienten Beleuchtungsterm** ein:

$$I = I_a + I_0 \cdot \mathbf{n} \cdot \mathbf{l}$$



Nur ambiente Beleuchtung

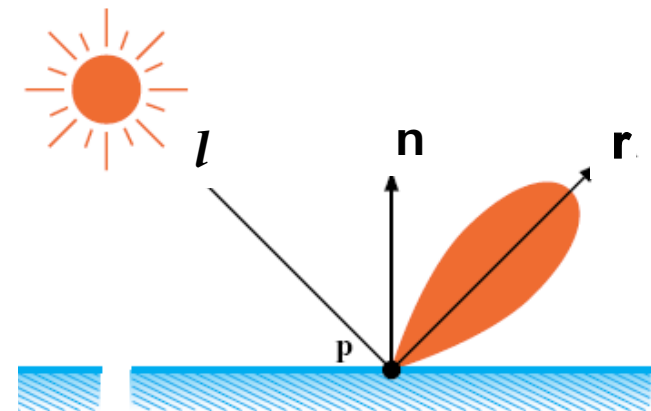
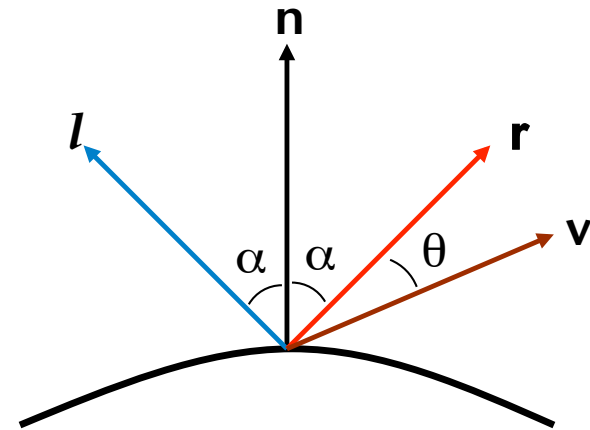


Diffuse + ambiente Beleuchtung

# Spiegelnde (spekulare) Reflexion

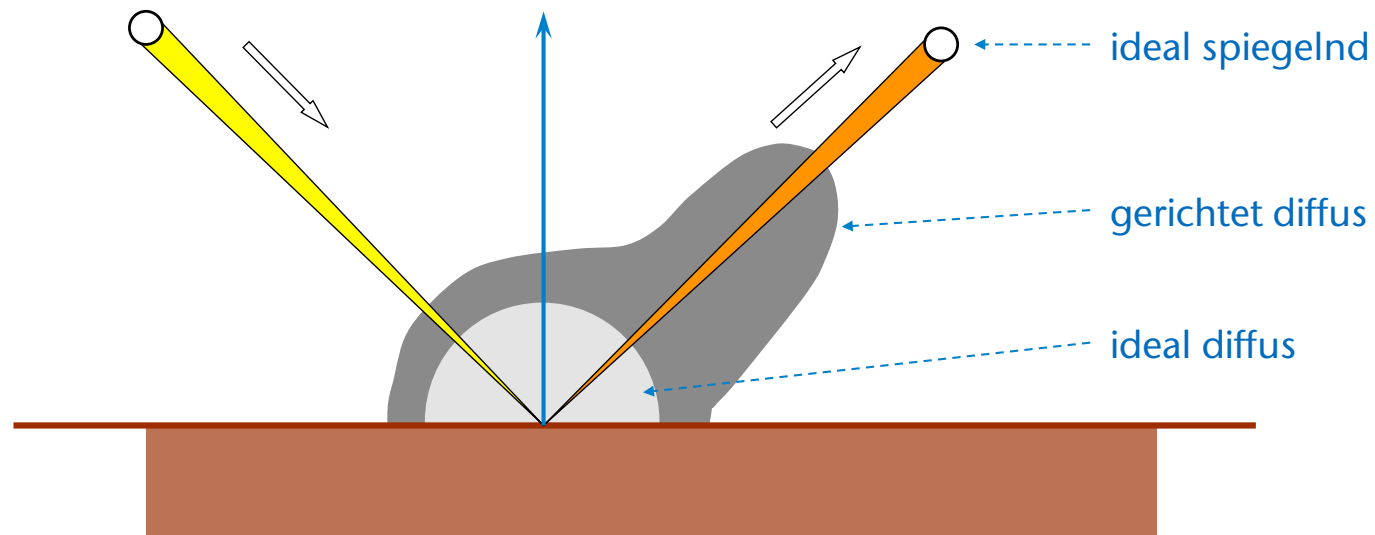
- Stellt Glanzpunkte auf glänzenden Oberflächen dar
- Oberflächenreflexion ist abhängig von
  - Richtung der Lichtquelle,  $l$
  - Oberflächennormale,  $n$
  - Richtung zum Betrachter,  $v$
- Bei idealer spiegelnder Reflexion sieht man nur dann Licht von der Lichtquelle, wenn  $r = v$
- Bei glänzenden Oberflächen sieht man auch "nahe" bei  $r$  ein Highlight; das erreicht man z.B. mit

$$I = I_0 (\cos \Theta)^p$$



# Gerichtet diffuse Reflexion

- Ideal diffuse und ideal spiegelnde Reflexion sind in der Realität selten; meist eine Mischung
- **Gerichtet diffuse Reflexion** (*glossy / specular / directional diffuse*): die abgestrahlte Intensität hat (oft) ein Maximum in Richtung der idealen Reflexion



- Zusammensetzung:

$$I = I_{\text{amb}} + I_{\text{diff}} + I_{\text{spec}}$$

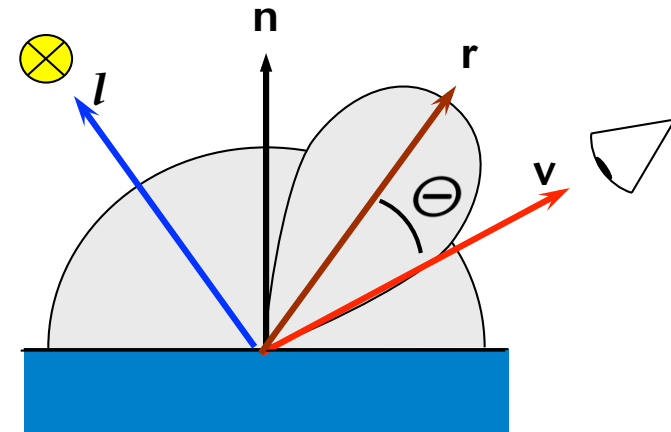
- Aufgrund des Superpositionsprinzips erhält man für  $n$  Lichtquellen:

$$I_{\text{out}} = r_d \cdot I_a + \sum_{j=1}^n (r_d \cos \phi_j + r_s \cos^p \Theta_j) \cdot I_j$$

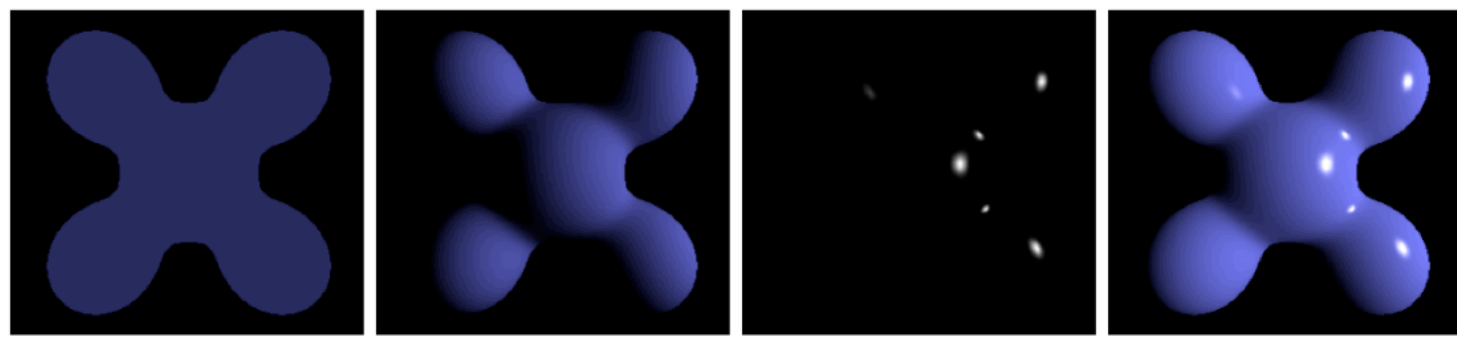
$r_d = r_d(\lambda)$  = diffuser Reflexionskoeffizient (spiegelnde Materialfarbe)


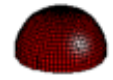



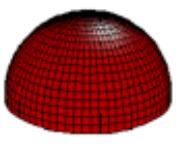

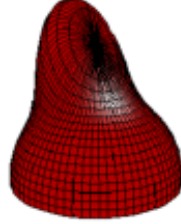



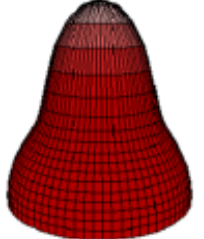
$r_s = r_s(\lambda)$  = spekularer Reflexionskoeffizient (diffuse Materialfarbe)

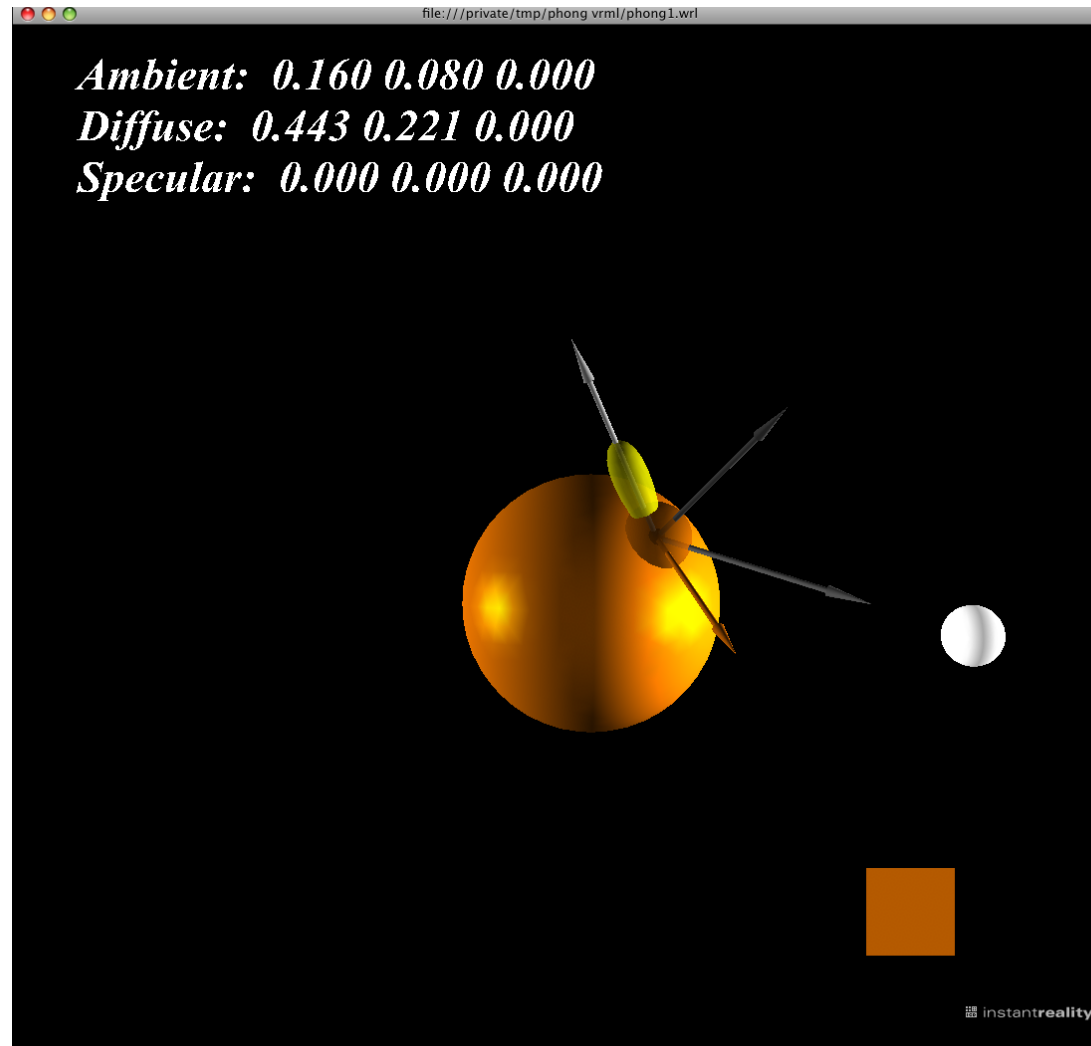
$p$  = "Glanz Zahl" (*shininess*), hat keine Einheit (hat keine physikalische Interpretation)



ambient + diffus + spekulär = Phong



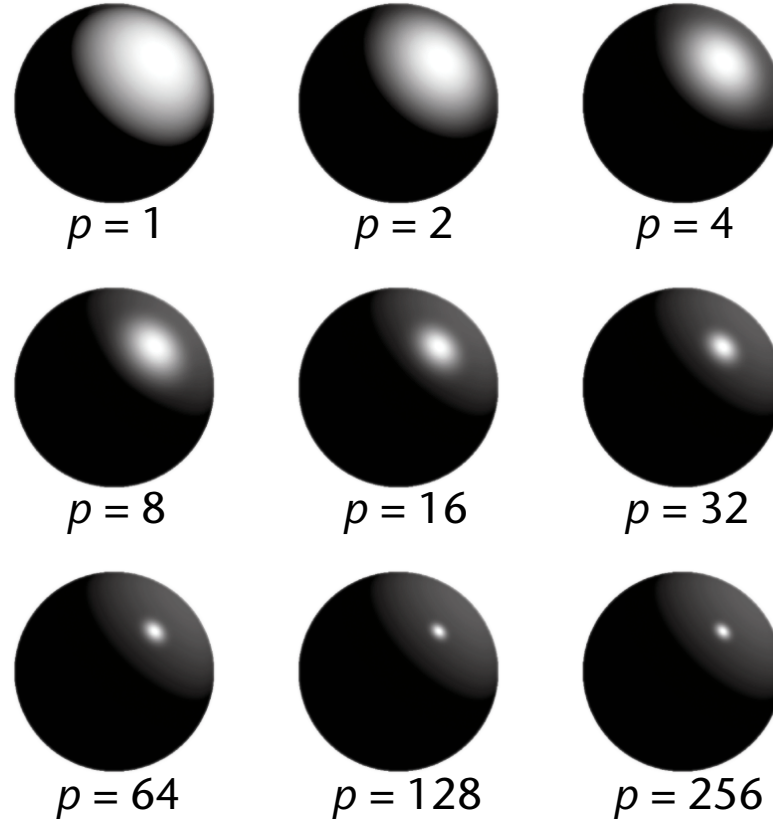
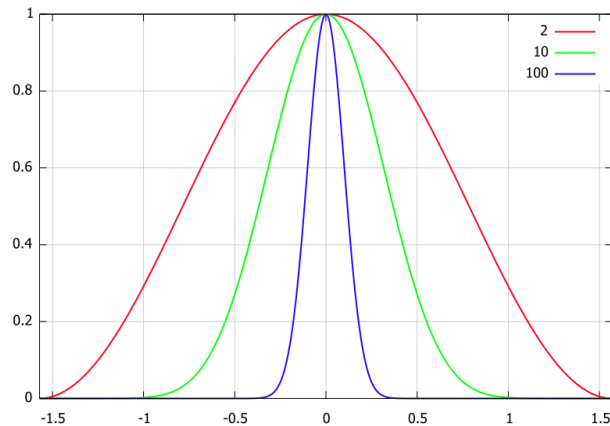
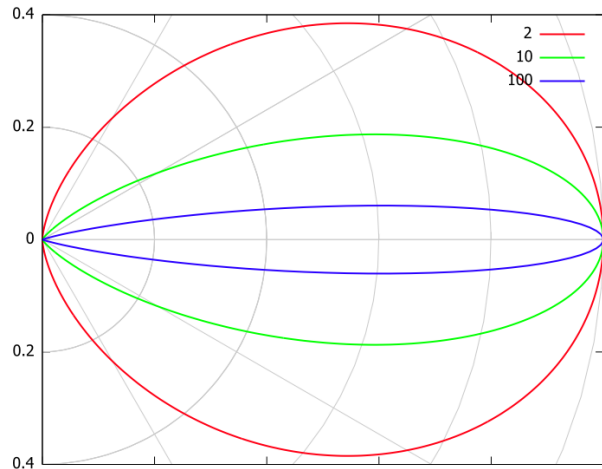
Phong	$\rho_{\text{ambient}}$	$\rho_{\text{diffuse}}$	$\rho_{\text{specular}}$	$\rho_{\text{total}}$
$\phi_i = 60^\circ$				
$\phi_i = 25^\circ$				
$\phi_i = 0^\circ$				



(Quelle: <http://www.avl.iu.edu/%7Eewernert/gviz/phong/> )

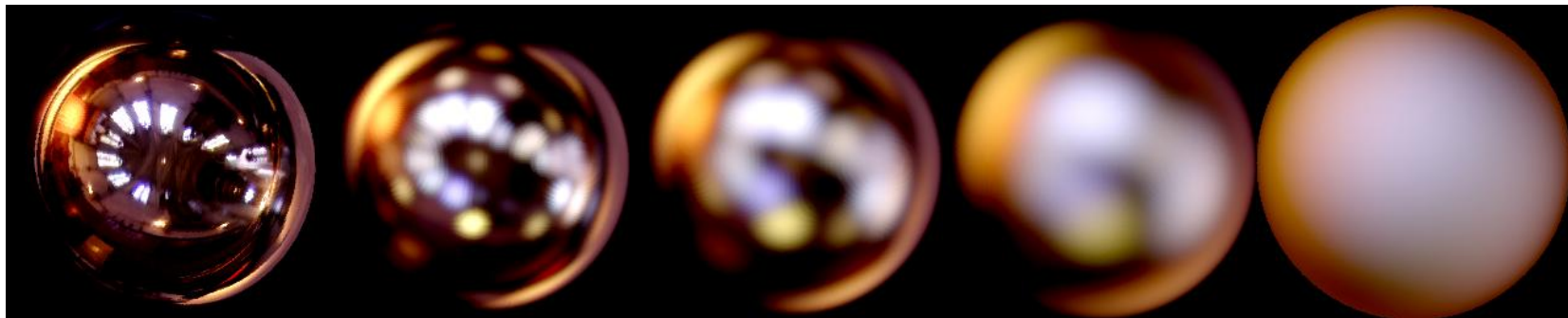
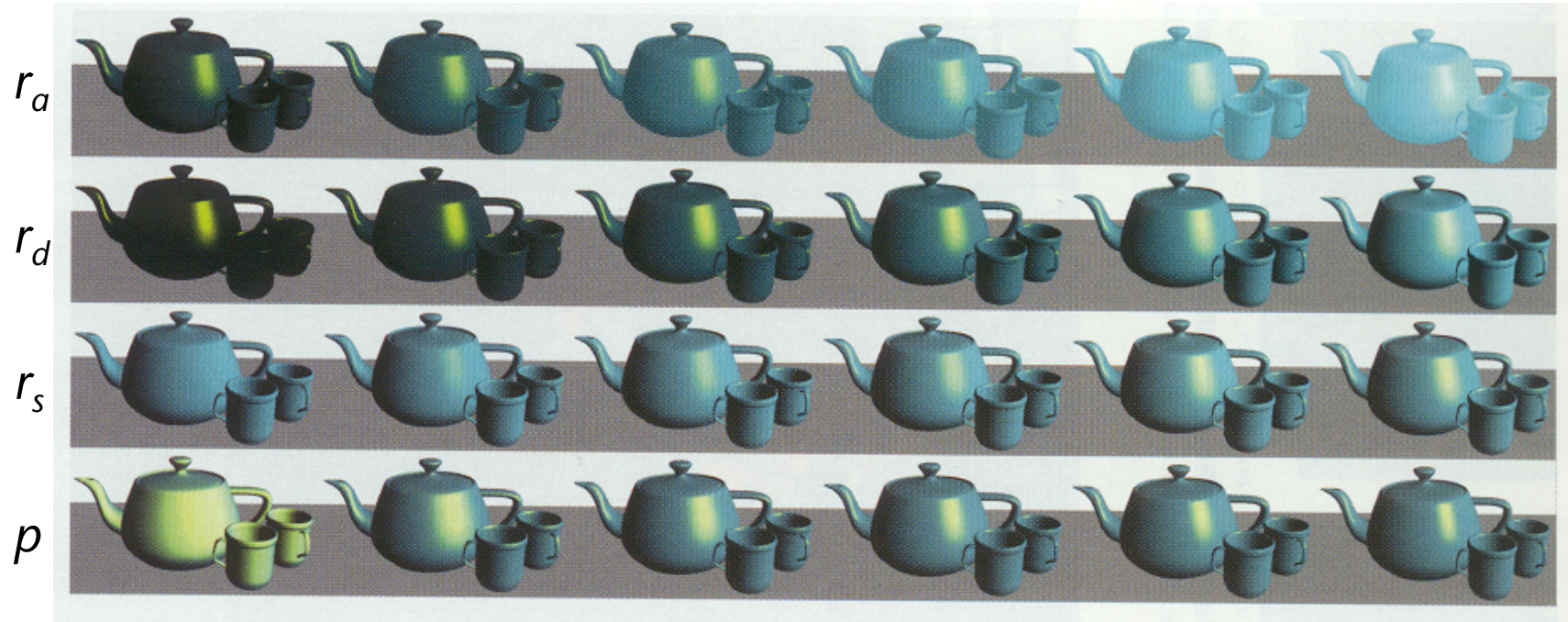
# Effekt der Parameter im Phong-Modell

- Der Exponent  $p$  steuert die "Schärfe" des Highlights:



$$I_{\text{spec}} = r_s I_0 \cos^p \Theta = r_s I_0 (\mathbf{r} \cdot \mathbf{e})^p$$





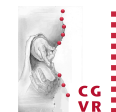
→ "Oberflächenkörnigkeit" ( $p$ )



- Zusätzliche Freiheit in OpenGL: diffuse und spekulare (= spiegelnde) Materialfarbe können verschieden sein
- Problem: Werte  $> 1$  können entstehen!
  - Abhilfe: Clamping
  - Besser wäre: Erhalten von Farbton und Sättigung ( $\rightarrow$  *Tone Mapping*)



# Highlights can Reveal Poorly Faked Fotos



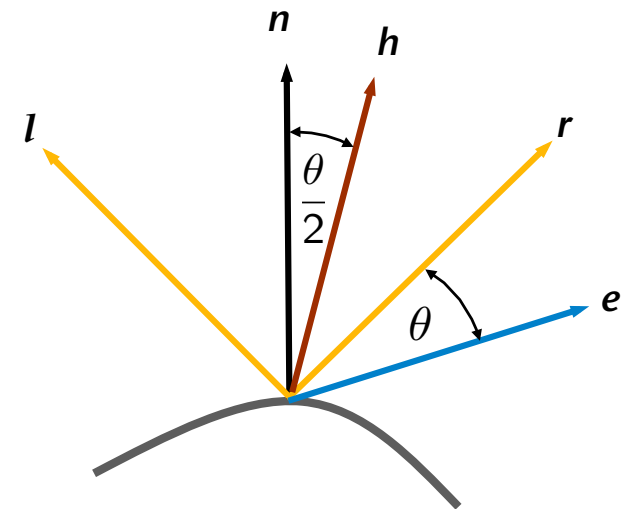
Pro 7, Galileo, Fake Fotos, 30. 5. 2013

- Problem des Phong-Modells: man muß für jeden Punkt den Reflexionsvektor bestimmen
- Idee: verwende Winkelhalbierende  $h$  ("half-vector") und  $n$ , statt  $r$  und  $e$ :

$$\mathbf{h} = \frac{\mathbf{l} + \mathbf{e}}{|\mathbf{l} + \mathbf{e}|}$$

- Setze:

$$I'_{\text{spec}} = r_s I_0 \cos^q \frac{\theta}{2} = r_s I_0 (\mathbf{h} \cdot \mathbf{n})^q$$



- Es gilt:  $I'$  ist max  $\Leftrightarrow I$  ist max
- Frage: ist es dasselbe Modell?  $\rightarrow$  fast
- Vorteil dieser Methode: wenn Auge und Lichtquelle unendlich weit entfernt sind, dann ist  $h$  (für eine bestimmte Lichtquelle) konstant! (Kann man also am Beginn eines Frames vorberechnen)

# Reflectance Models in General

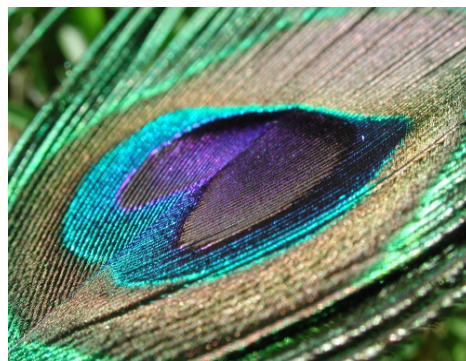
- **Reflectance** :=  $\frac{\text{Outgoing radiance}}{\text{Incoming radiance}}$ 
  - **Radiance** = physikalischer Terminus für Lichtintensität
- Das Lambert'sche Gesetz und das Phong-Modell sind schon einfache Reflectance-Modelle:

$$I_o = \underbrace{r_s (\mathbf{h} \cdot \mathbf{n})^q}_{\text{reflectance}} I_i$$

$$I_o = \underbrace{(\mathbf{n} \cdot \mathbf{l})}_{\text{reflectance}} \cdot I_i$$

- Das sind sehr einfache Modelle, um die Materialeigenschaft "Reflectance" mathematisch zu fassen

- Echte Materialien sind i.A. viel komplizierter!

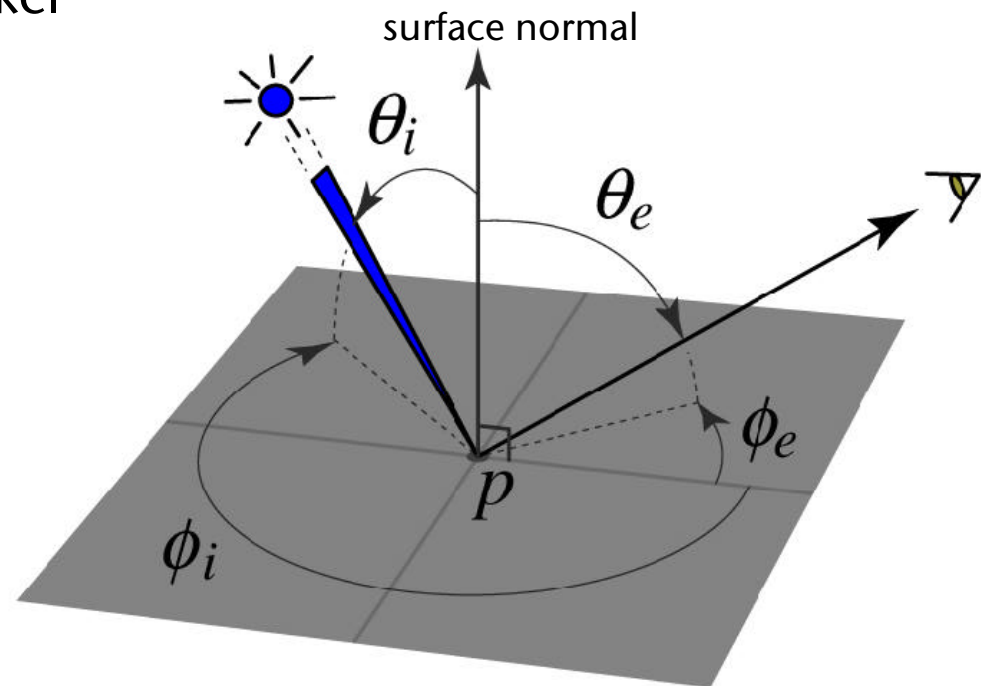


# BRDFs als Verallgemeinerung

- **BRDF (Bidirectional Reflectance Distribution Function)** = Funktion, die zu jedem Einfallswinkel und Ausfallswinkel die Reflectance liefert:

$$\rho(\theta_i, \phi_i; \theta_o, \phi_o)$$

dabei sind  $(\theta_i, \phi_i)$  die Einfallswinkel,  
und  $(\theta_o, \phi_o)$  die Ausfallswinkel

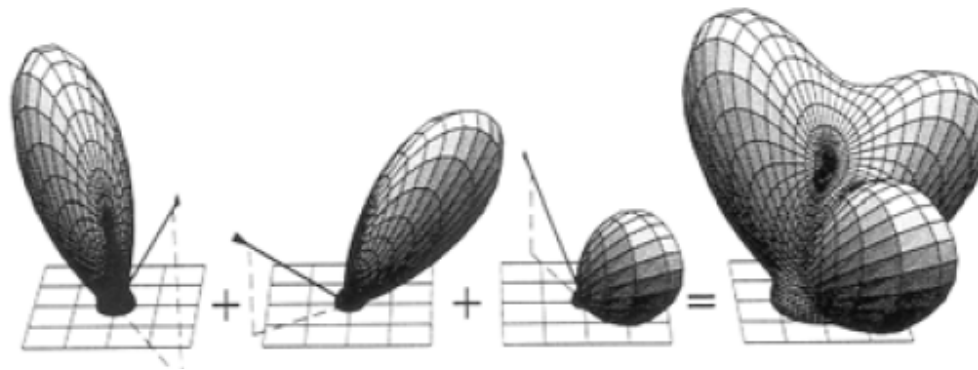




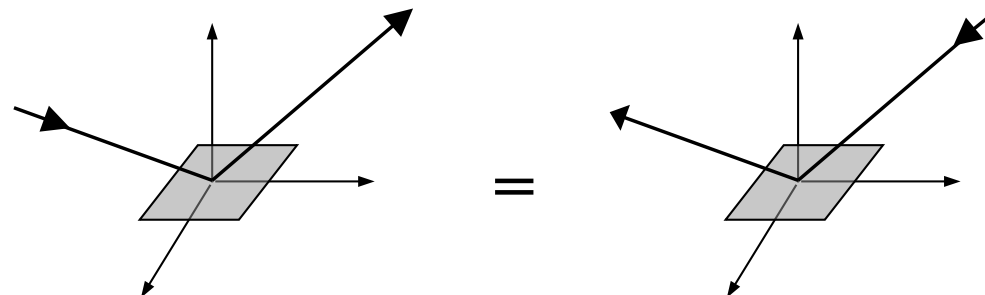
# Generelle Eigenschaften von BRDFs

- Sind eigentlich Annahmen, die zu Vereinfachungen führen
- Manche werden nicht gemacht / gelten für manche Materialien nicht

## 1. Linearität:

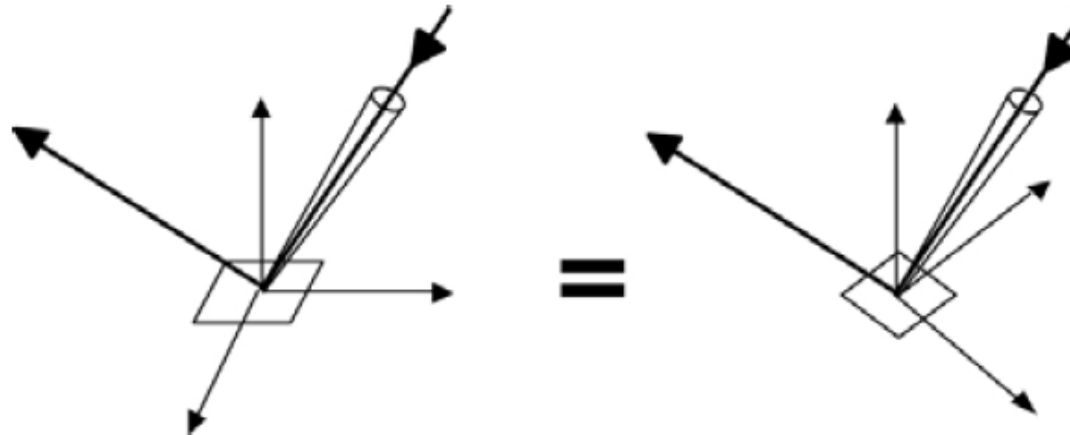


## 2. Reciprozität (reciprocity):



### 3. Isotropie (isotropy):

BRDF ist invariant bzgl. Rotation des Materials um die Normale



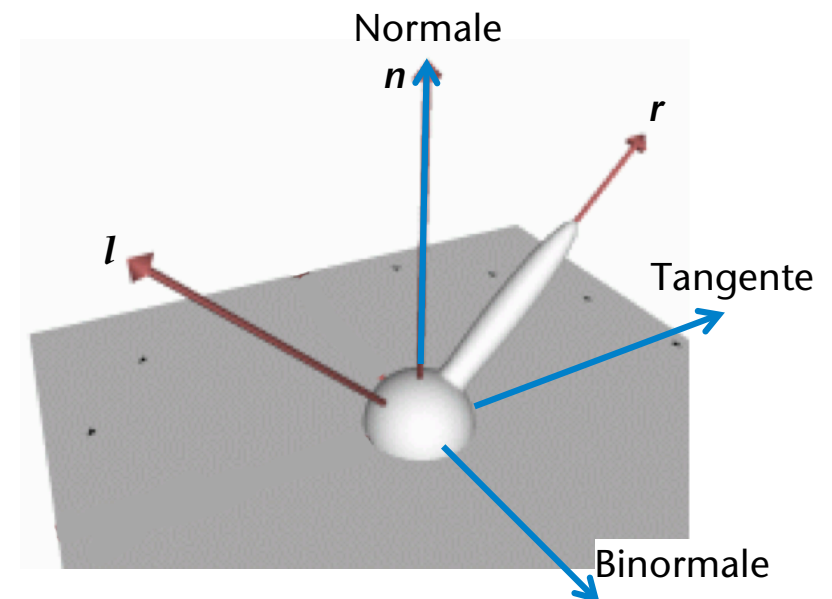
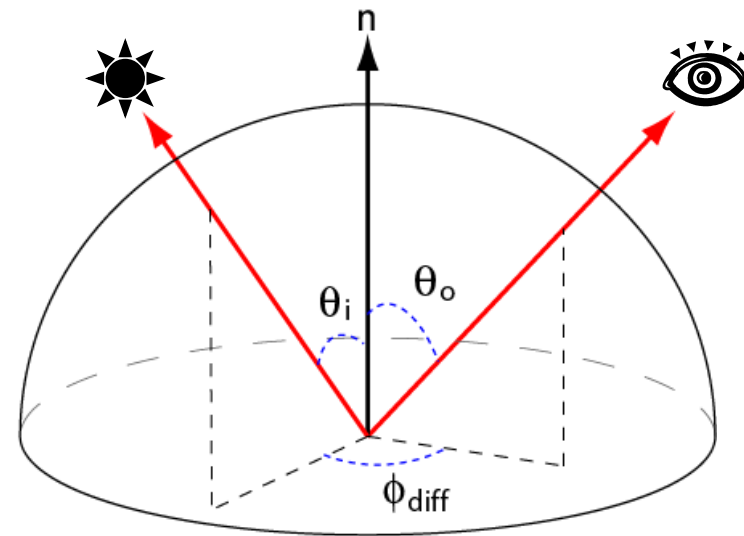
### 4. Energieerhaltung (conservation of energy):

total quantity of outgoing light  $\leq$  total quantity of incoming light  
(m.a.W.: Integral über die BRDF  $\leq 1$ )

### 5. Die BRDF eines Materials hängt *nicht* von dem jeweiligen Punkt auf der Oberfläche ab!

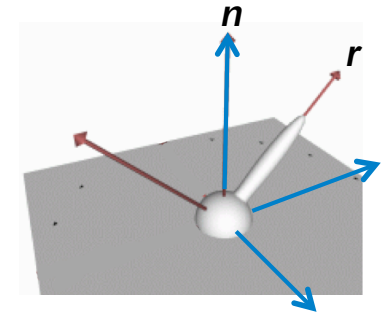


- Konsequenz: die BRDF ist eigentlich nur eine 3D-Funktion
  
- Wähle im folgenden dieses spezielle Koordinatensystem zur Auswertung einer BRDF in einem Punkt



- Das Phong-Modell in diesem Koordinatensystem ist

$$\rho(\mathbf{l}, \mathbf{e}) = (\mathbf{e} \cdot \mathbf{r})^p = \left( \mathbf{e} \cdot \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \mathbf{l} \right)^p$$

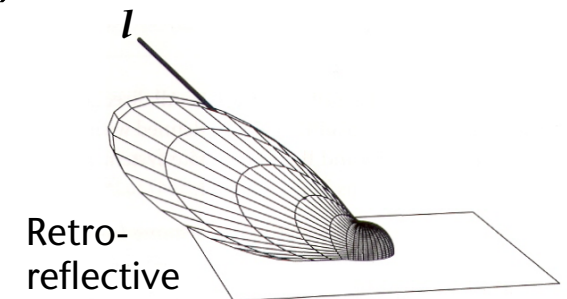


- Das Lafortune-Modell:

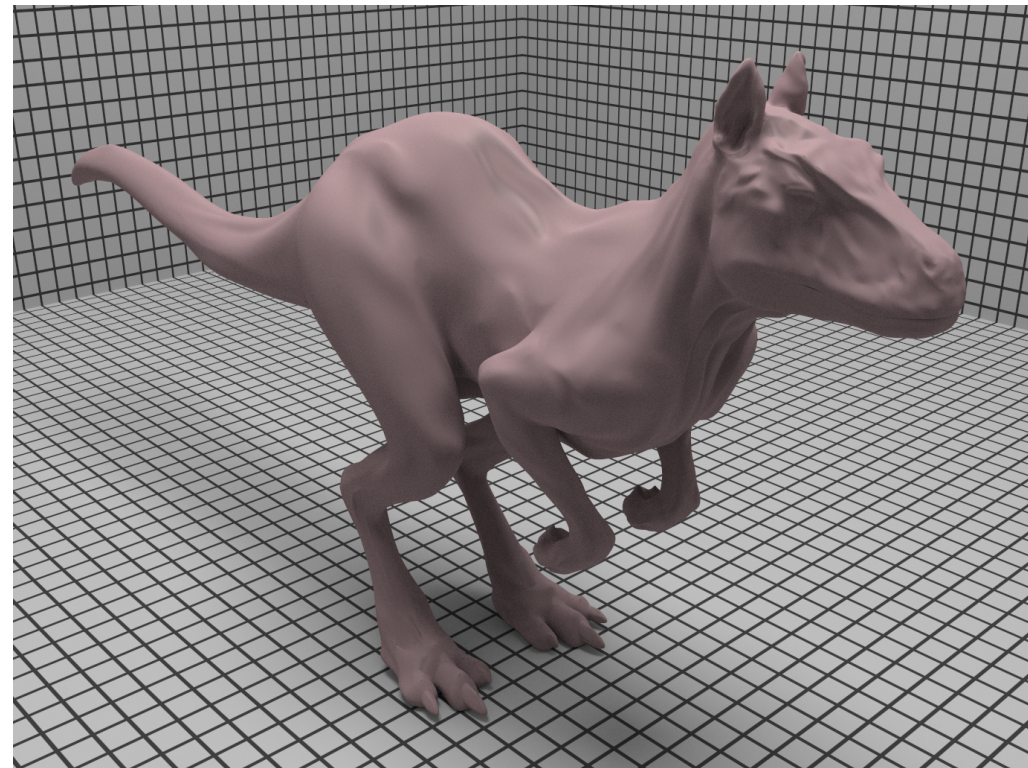
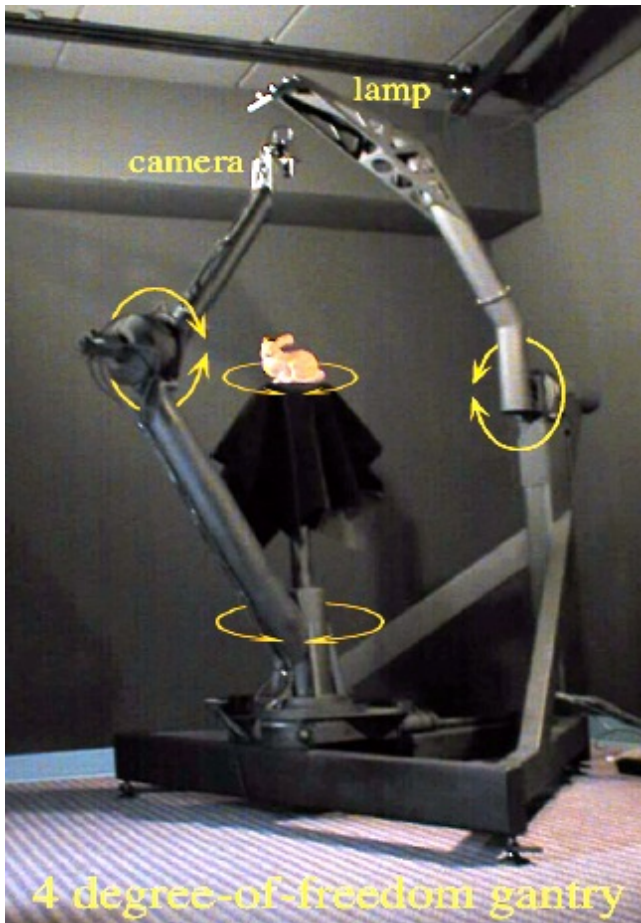
- Ersetze die spezielle Matrix durch eine beliebige Matrix C
- Erlaube beliebig viele "Lobes" (= Keulen, Lappen)
- Zusammen:

$$\rho(\mathbf{l}, \mathbf{e}) = r_d(\mathbf{n} \cdot \mathbf{l}) + \sum_{i=1}^n r_{s,i} (\mathbf{e} \cdot \mathbf{C}_i \cdot \mathbf{l})^{p_i}$$

wobe  $n$  = Anzahl lobes



- In der Praxis misst man Materialien aus und versucht, das Lafortune-Modell darauf zu "fitten" (=  $1 + 11n$  Parameter)



Mit ausgemessenem Ton

# Spectral Lighting vs RGB-Lighting

- Erinnerung: alle photometrischen Größen sind eigtl. **Funktionen in  $\lambda$ !** beschreiben also ein Spektrum ...
- In der Praxis (z.B. OpenGL): führe alle Berechnungen jeweils für die 3 Primärvalenzen (z.B.  $r, g, b$ ) durch
- Aber: dadurch erhält man nicht 100% korrekte Bilder!
- Denn:

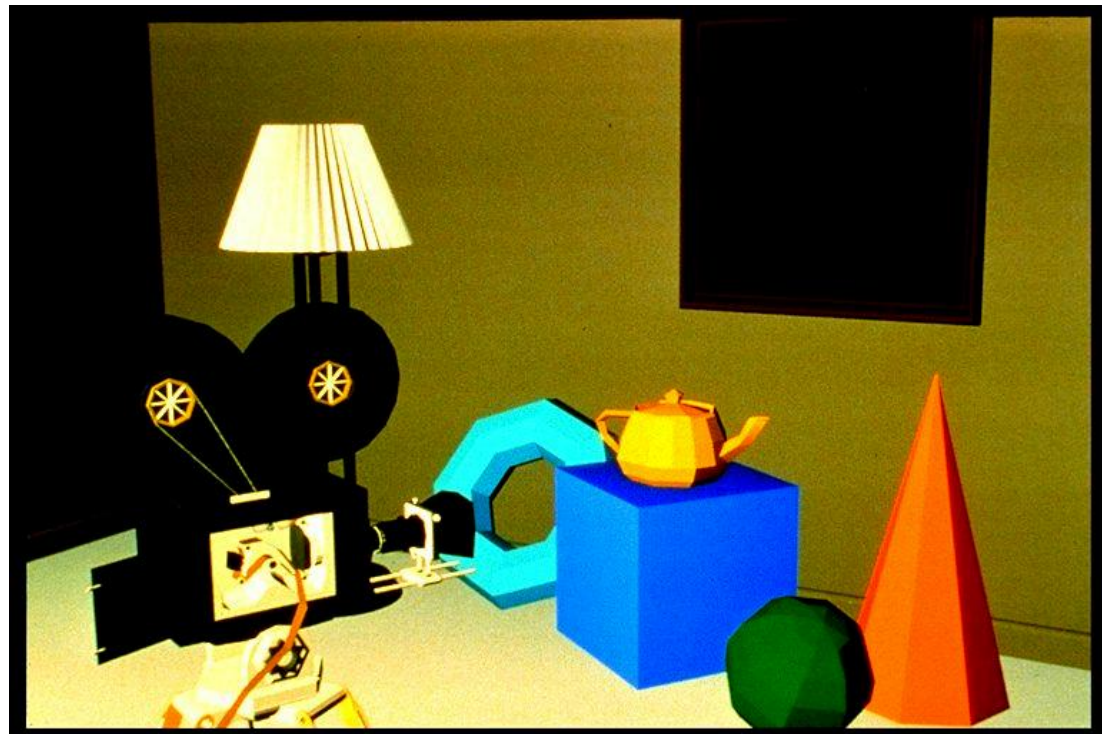
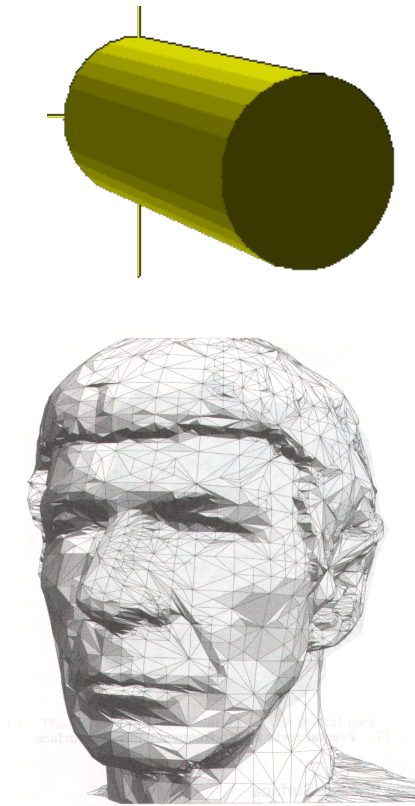
$$\text{RGB}(r(\lambda) \cdot I(\lambda)) \neq \text{RGB}(r(\lambda)) \cdot \text{RGB}(I(\lambda))$$

- Achtung: unterscheide zwischen **Beleuchtungsmodell** (*lighting model*) und **Beleuchtungsalgorithmus** (*shading algorithm*)!
  - **Beleuchtungsmodell** beschreibt Zusammenhang zwischen Lichtquellen und Oberflächen zur Berechnung der Intensität in jedem Punkt
  - **Beleuchtungsalgorithmus** berechnet aus der Intensität/Farbe einiger Punkte die Farbe **aller** Bildpunkte
  - Leider: große Begriffsverwirrung! ;-(
    - "lighting algorithm", "shading model", ...
- 3 Möglichkeiten, das Beleuchtungsmodell auszuwerten:
  - 1x pro Polygon
  - 1x pro Vertex
  - 1x pro Pixel



# Flat Shading (Konstante Beleuchtung)

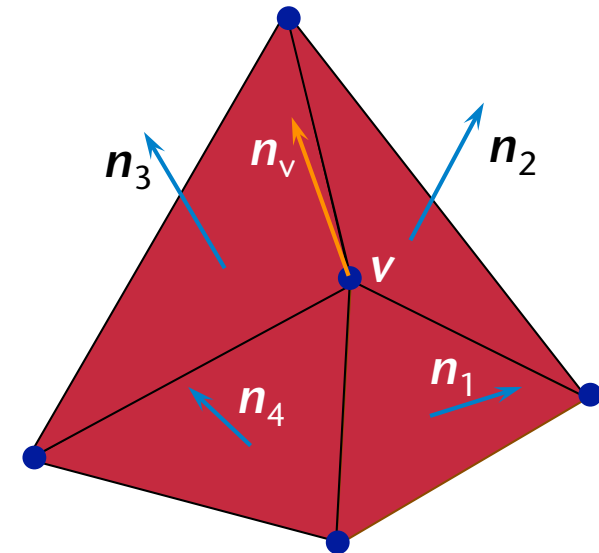
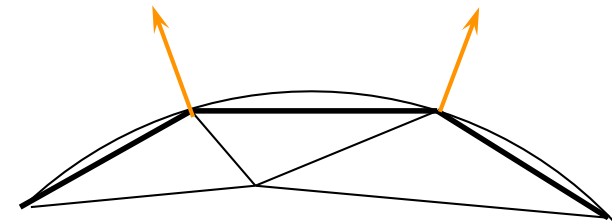
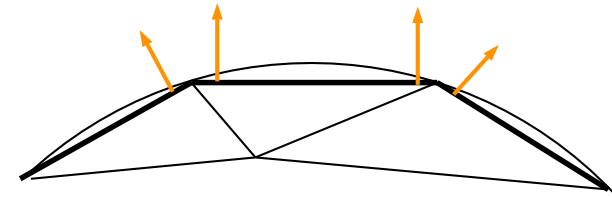
- Simplester Shading-Algo: jedes Polygon erhält einen einheitlichen Farbwert
  - Werte dazu das Beleuchtungsmodell an irgend einer Ecke des Polygons aus



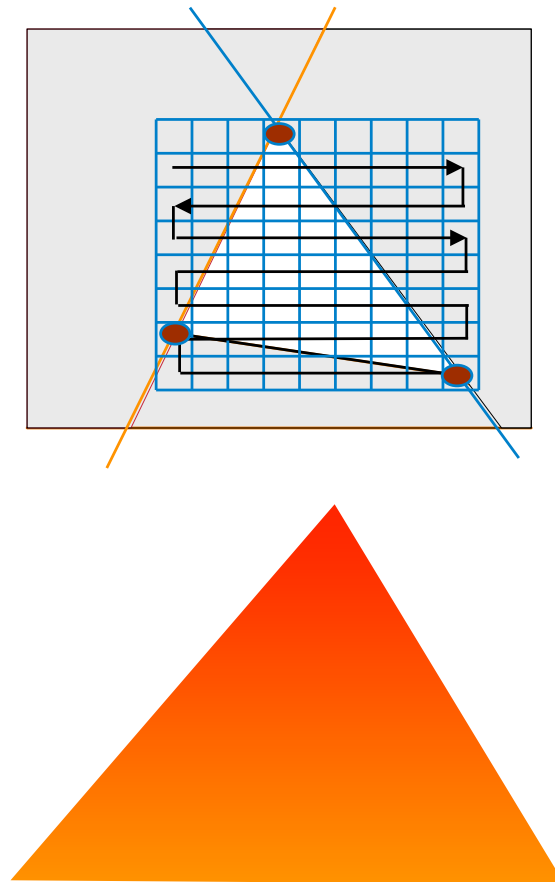
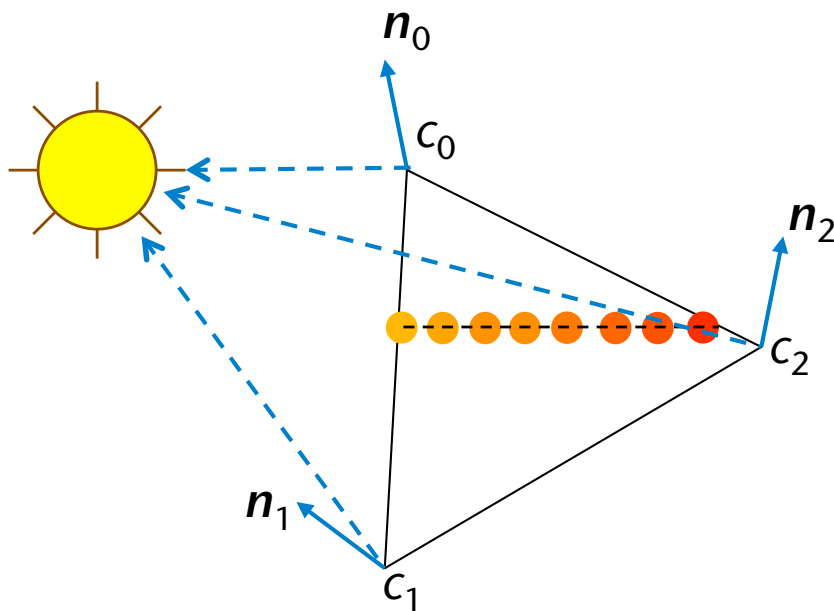
Pixar "Shutterbug"

# Berechnen der Normalen der Eckpunkte

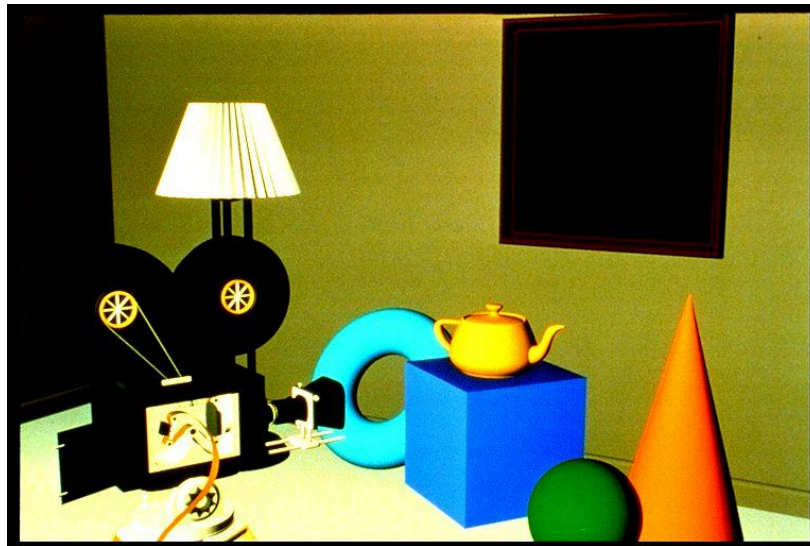
- Die Dreiecke bilden nur eine **Annäherung** an die wirkliche Oberfläche eines Objektes
- An den Vertices hätte man gerne die Normale der Fläche, nicht der Dreiecke!
- Algorithmus:
  - Zu Beginn berechne eine Normale für jedes Polygon
  - Bestimme für jeden Vertex, welche Polygone diesen enthalten
  - Bestimme den "Mittelwert" der Normalen dieser angrenzenden Polygone
    - Einfach aufsummieren, dann normieren



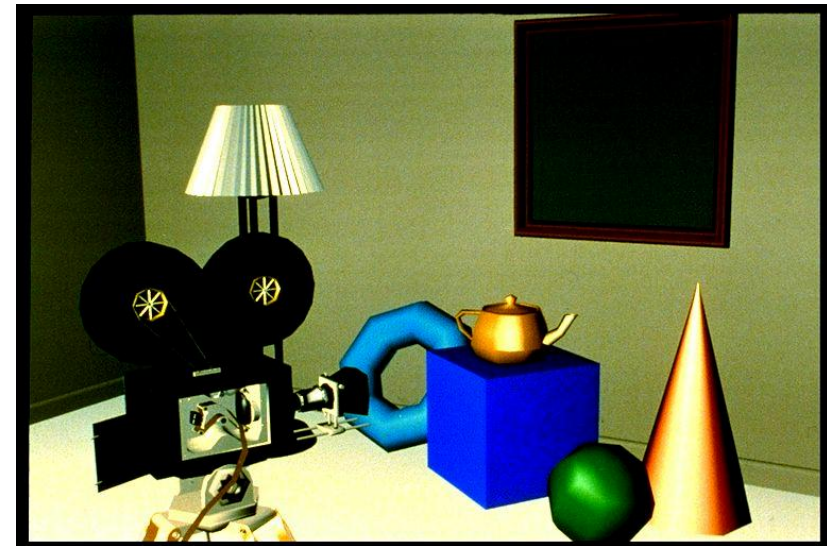
- Werte das Beleuchtungsmodell an allen 3 Ecken des Dreiecks aus, interpoliere linear dazwischen während der Scanline-Konvertierung



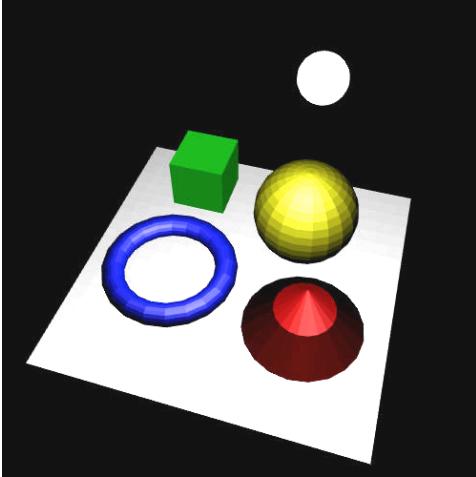




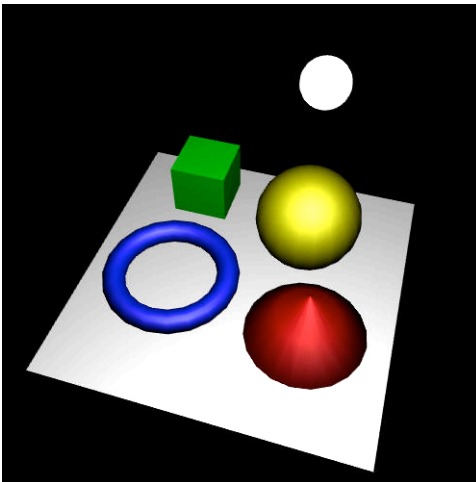
Gouraud-Shading mit  
rein diffusem Beleuchtungsmodell



Gouraud-Shading mit  
Phong-Modell



Flat-Shading mit  
Phong-Lighting

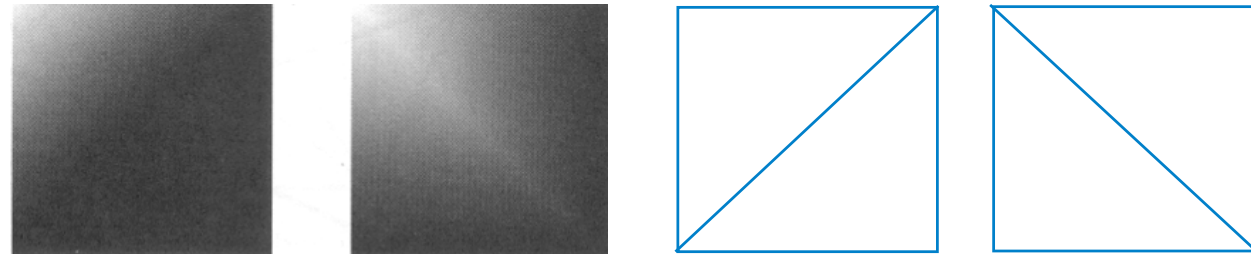


Gouraud-Shading mit  
Phong-Lighting

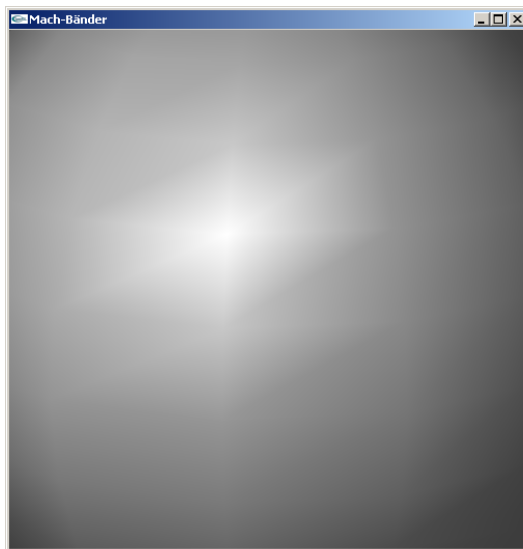


# Einfluß der Triangulierung

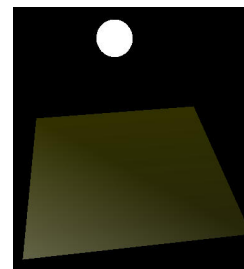
- Orientierung:



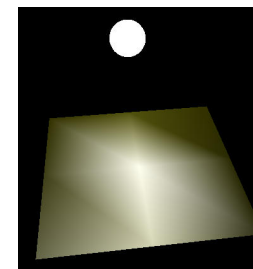
- Hier gilt: "size matters"



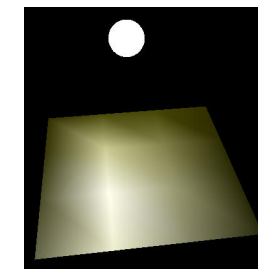
Vergrößerung



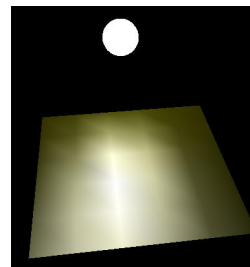
1 x 1 Flächen



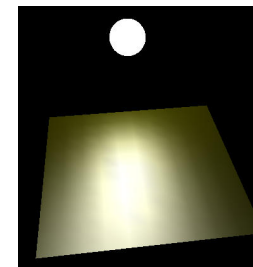
2 x 2 Flächen



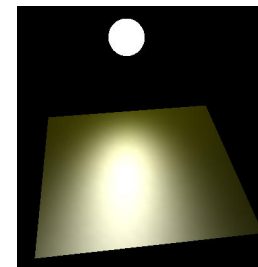
3 x 3 Flächen



5 x 5 Flächen



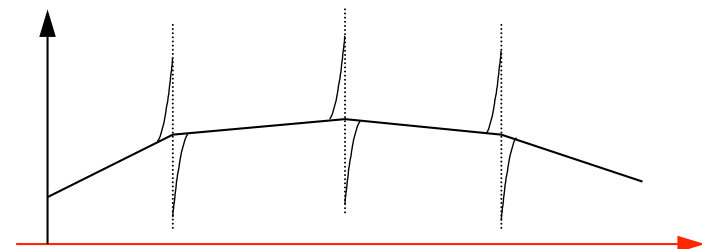
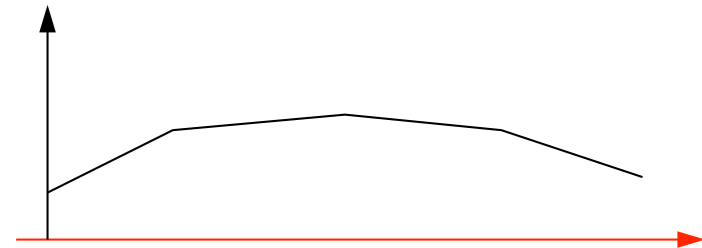
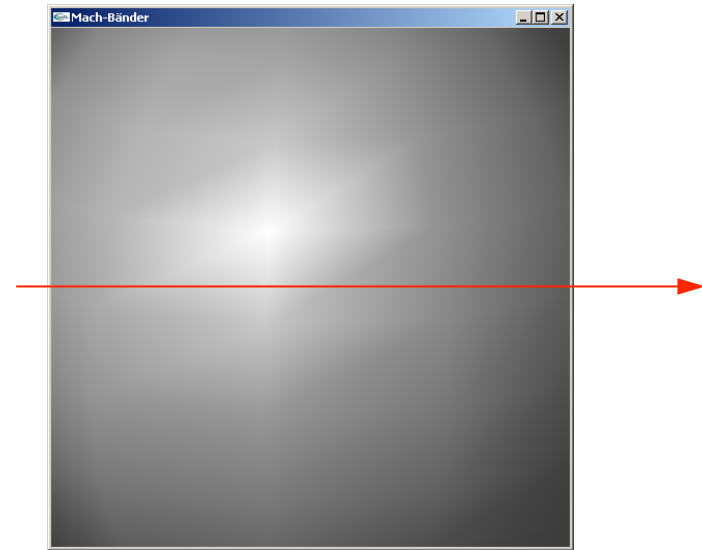
10 x 10 Flächen



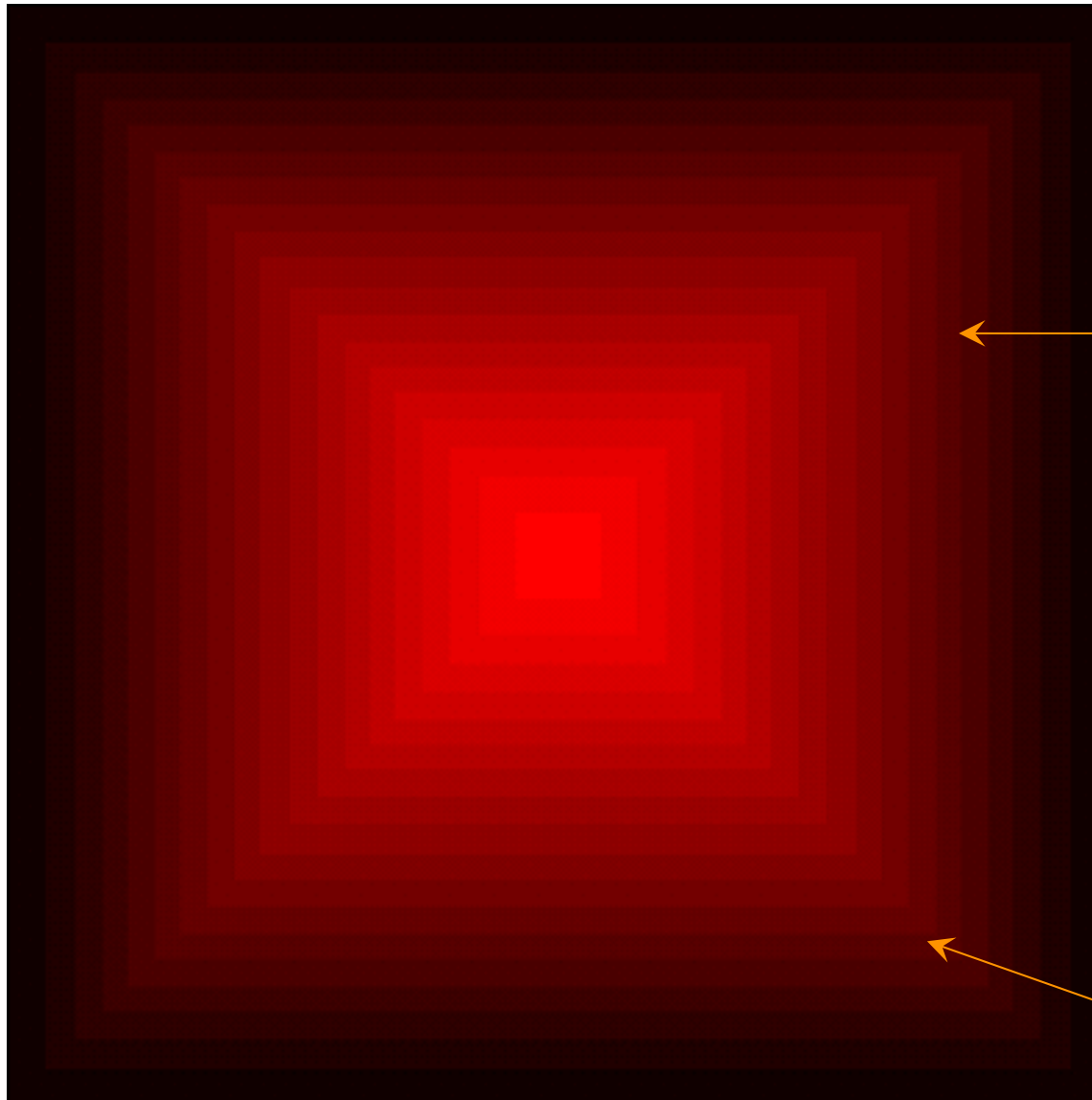
40 x 40 Flächen

- Frage: Woher kommen die „Streifen“?

- Problem: die lineare Interpolation der "Lichtwerte"
  - Diese Interpolation ist  $C^0$ -stetig, aber nicht  $C^1$ -stetig!
- Das menschliche Auge hat einen eingebauten "Kantendetektor" (erkennt genau diese "Knicke")
  - Es gibt Neuronen, die die räuml. Ableitung bilden (jew. für ein Retina-"Pixel")
  - Wahrgenommene Intensität = physik. Intensität + Ableitung
  - Resultat: Mach-Bänder bei linearer Interpolation
- Abhilfe: Hardware müsste höherwertig interpolieren...



# Extremes Beispiel

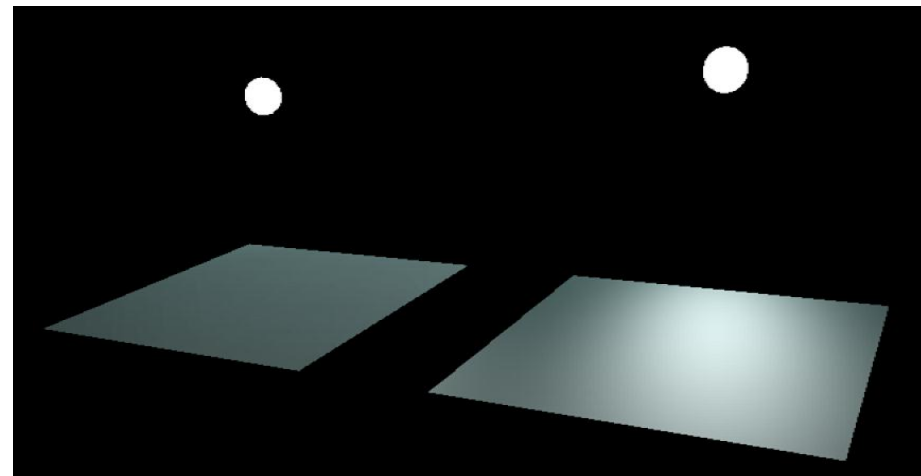


← Die Intensität innerhalb eines Quadrates ist konstant!  
Ein Farbverlauf von innen nach außen ist eine Illusion.

← Die hellen Linien bei  $45^\circ$  (und  $135^\circ$ ) sind eine Illusion!

# Weiteres Problem beim Gouraud-Shading

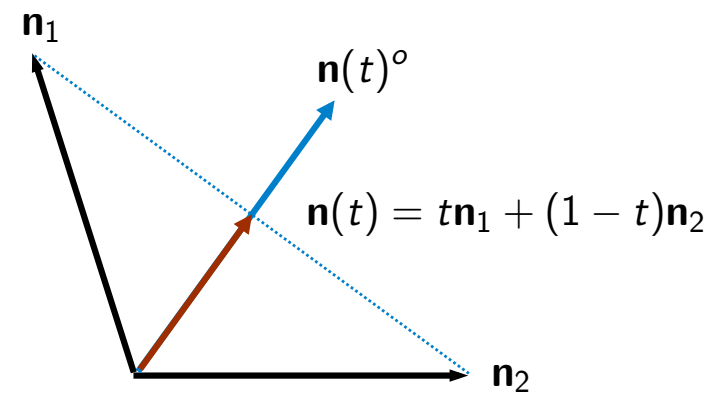
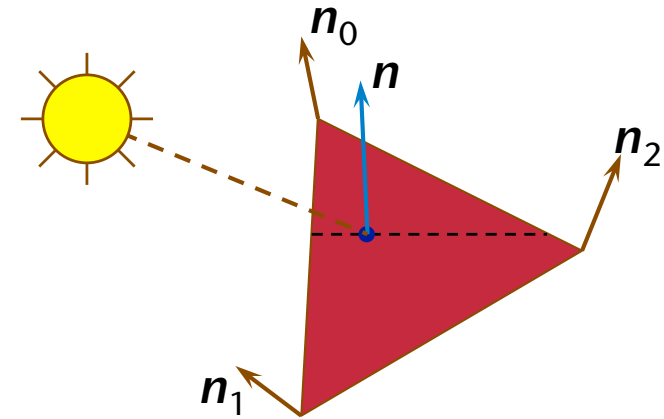
- Evtl. "verpasst" man Highlights im Inneren eines Polygons:



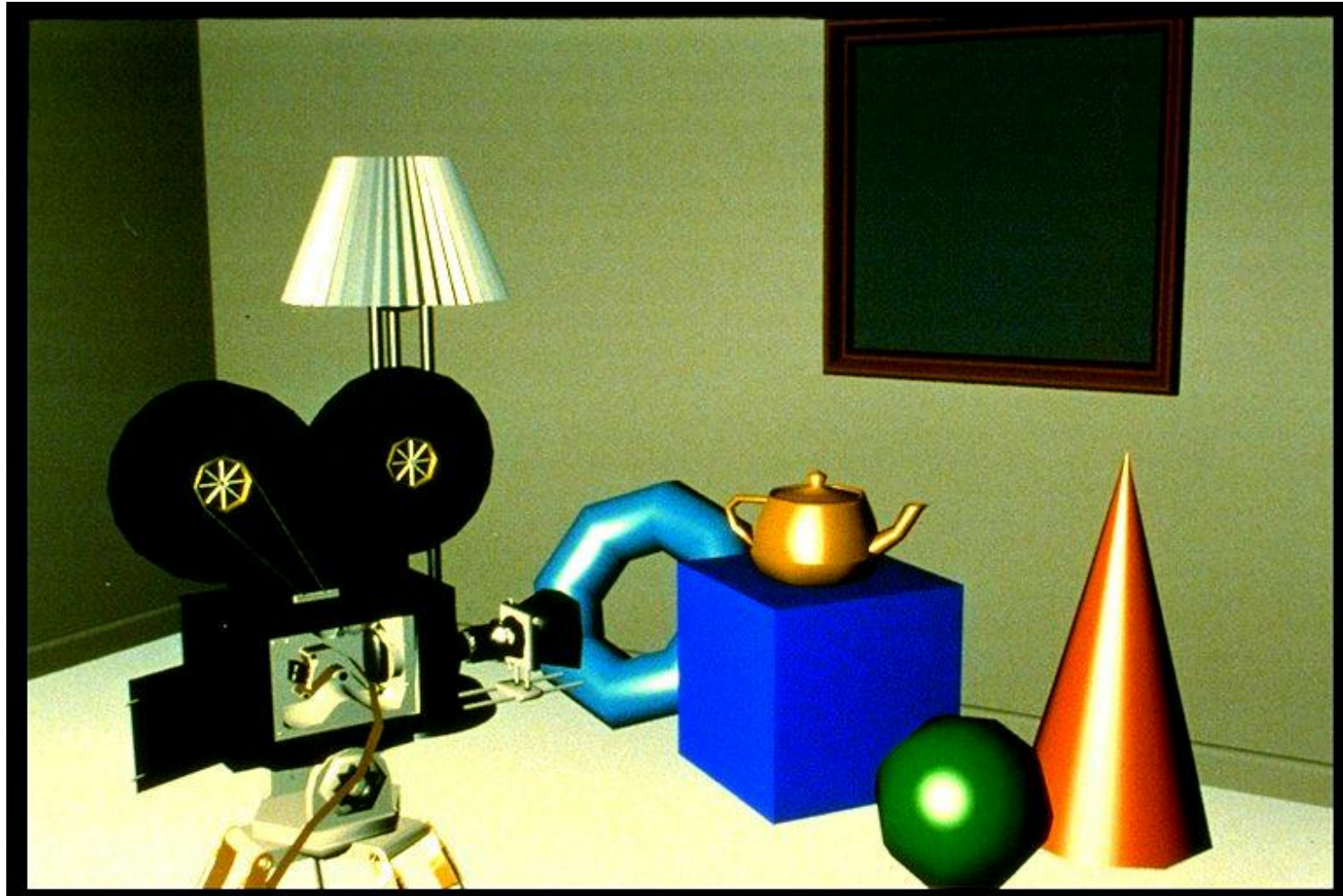
Gouraud-Shading

Phong-Shading

- Idee: interpoliere die **Normale** während der Scanline-Konvertierung und werte das Beleuchtungsmodell in **jedem Pixel** aus
- Wie interpoliert man Normalen?
  - Typischerweise: linear mit anschließender Normierung
  - **Achtung: ohne Normierung** bekäme man nur (sehr umständliches) **Gouraud-Shading!**
  - War früher sehr teuer, daher wurden viele Alternativen vorgeschlagen
    - Inkrementell, Taylor-Reihe + LUT, ...









# Beleuchtung in OpenGL

- **Phong-** oder Blinn-Phong-Modell mit **Gouraud**-Shading; plus ein paar zusätzlichen Freiheitsgrade
- Jede **Lichtquelle**  $L_j$  in OpenGL besteht aus 3 Teilen: **ambienter** ( $I_{j,a}$ ), **diffuser** ( $I_{j,d}$ ), und **spekularer** ( $I_{j,s}$ ) Anteil
- Es gibt eine zusätzliche globale ambiente Lichtfarbe  $I_a$
- **Materialien** in OpenGL bestehen aus: Emissionsfarbe ( $E$ ), und ambiente ( $r_a$ ), diffuse ( $r_d$ ), spiegelnde ( $r_s$ ) Reflexionskoeffizienten
  - $r_a, r_d, r_s$  sind RGB-Vektoren

- Insgesamt:

$$I = I_a + \sum_{j=1}^n (E_j + r_a \cdot I_{j,a} + r_d I_{j,d} \cos \phi_j + r_s \cdot I_{j,s} \cos^n \Theta_j)$$

- Werte größer 1 werden auf 1 "geclamt"



# Lichtquellendefinition



```
GLfloat ambient[] = { 0.0, 0.0, 0.0, 1.0 };  
GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };  
GLfloat specular[] = { 1.0, 1.0, 1.0, 1.0 };  
GLfloat position[] = { 0.0, -0.5, 0.5, 1.0 };
```

```
glShadeModel( GL_SMOOTH );
```

Shading-Algo  
(Gouraud)  
einschalten

```
glLightfv( GL_LIGHT0, GL_AMBIENT, ambient );  
glLightfv( GL_LIGHT0, GL_DIFFUSE, diffuse );  
glLightfv( GL_LIGHT0, GL_SPECULAR, specular );  
glLightfv( GL_LIGHT0, GL_POSITION, position );
```

Lichtquelle  
"Nr. 0"  
definieren

```
glEnable( GL_LIGHTING );
```

Beleuchtung einschalten

```
glEnable( GL_LIGHT0 );
```

Lichtquelle „Nr. 0“ einschalten



# Materialdefinition



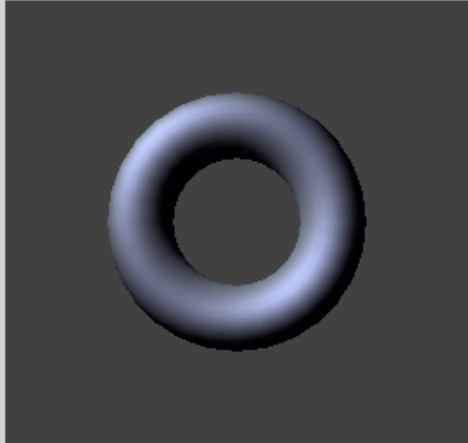
```
GLfloat mat_emission[] = {0.0, 0.0, 0.0, 0.0};
GLfloat mat_ambient[] = { 0.25, 0.20, 0.07, 1.0 };
GLfloat mat_diffuse[] = { 0.75, 0.61, 0.23, 1.0 };
GLfloat mat_specular[] = { 0.63, 0.56, 0.37, 1.0 };
GLfloat shininess[] = { 51.0 };

glMaterialfv( GL_FRONT, GL_EMISSION, mat_emission );
glMaterialfv( GL_FRONT, GL_AMBIENT, mat_ambient );
glMaterialfv( GL_FRONT, GL_DIFFUSE, mat_diffuse );
glMaterialfv( GL_FRONT, GL_SPECULAR, mat_specular );
glMaterialfv( GL_FRONT, GL_SHININESS, shininess );

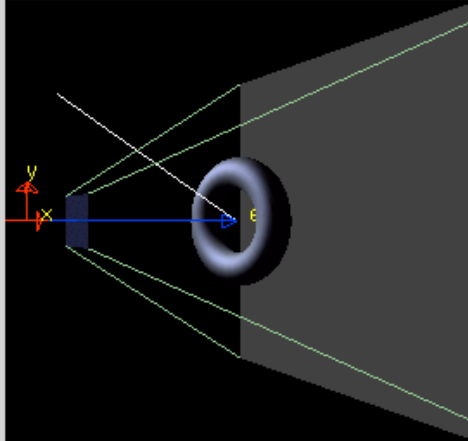
DrawSphere (...);
```

○ ● ● **Light & Material**

Screen-space view



World-space view



Command manipulation window

```

GLfloat light_pos[] = { -2.00 , 2.00 , 2.00 , 1.00 };
GLfloat light_Ka[] = { 0.00 , 0.00 , 0.00 , 1.00 };
GLfloat light_Kd[] = { 1.00 , 1.00 , 1.00 , 1.00 };
GLfloat light_Ks[] = { 1.00 , 1.00 , 1.00 , 1.00 };

glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_Ka);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_Kd);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_Ks);

GLfloat material_Ka[] = { 0.11 , 0.06 , 0.11 , 1.00 };
GLfloat material_Kd[] = { 0.43 , 0.47 , 0.54 , 1.00 };
GLfloat material_Ks[] = { 0.33 , 0.33 , 0.52 , 1.00 };
GLfloat material_Ke[] = { 0.00 , 0.00 , 0.00 , 0.00 };
GLfloat material_Se = 10 ;

glMaterialfv(GL_FRONT, GL_AMBIENT, material_Ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, material_Kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, material_Ks);
glMaterialfv(GL_FRONT, GL_EMISSION, material_Ke);
glMaterialfv(GL_FRONT, GL_SHININESS, material_Se);

```

Click on the arguments and move the mouse to modify values.

<http://www.xmission.com/~nate/>

- Gouraud-Shading:

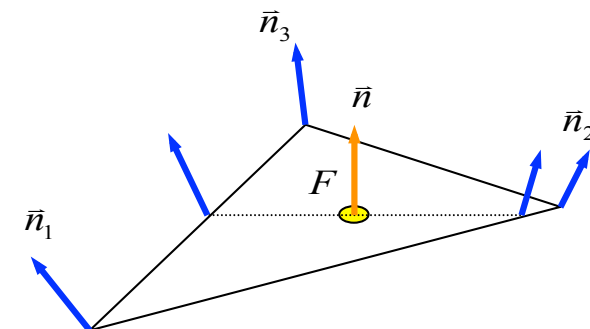
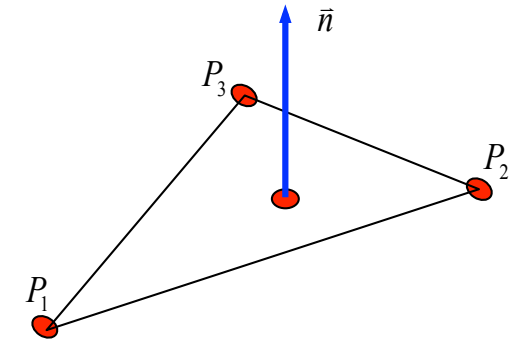
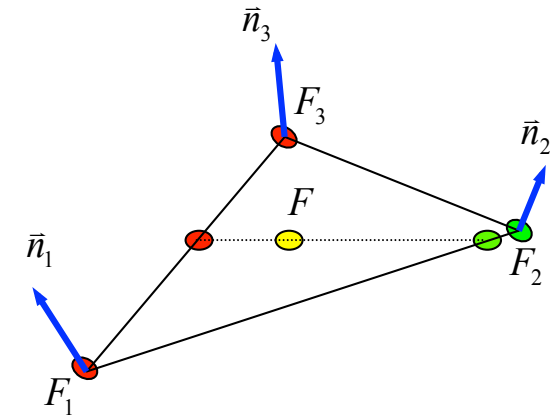
```
glShadeModel( GL_SMOOTH );
// Normale pro Eckpunkt
glBegin( GL_TRIANGLES )
    glNormal3f(...);
    glVertex3f(...);
    ...
glEnd();
```

- Flat-Shading:

```
glShadeModel( GL_FLAT );
// konstante Flächennormale
glNormal3f(...);
glBegin( GL_TRIANGLES )
    glVertex3f(...);
    ...
glEnd();
```

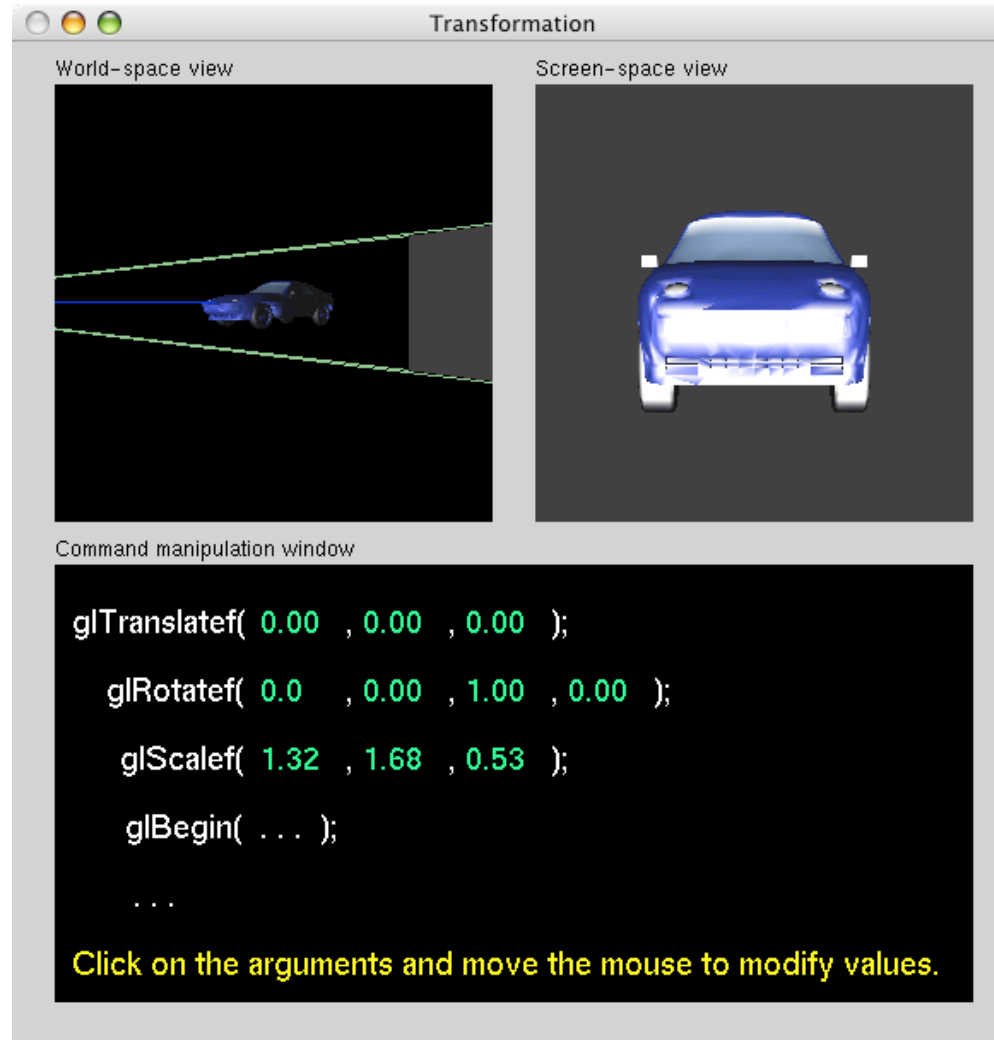
- Phong-Shading:

```
// Nur mit Shadern möglich ...
```





- Die Berechnung setzt voraus, daß die Normalen normiert sind!
  - Wenn das nicht der Fall ist, sind die Objekte zu hell oder zu dunkel (s. Demo)
- Erinnerung: Normalen werden mit der transponierten Inversen der `GL_MODELVIEW`-Matrix transformiert
  - Problem: danach sind die Normalen evtl nicht mehr normiert!
- Falls man nur Rotation und Translation verwendet, genügt es, normierte Normalen an OpenGL zu übergeben (warum?)
- Sonst:
  - **`glEnable( GL_NORMALIZE )`**
    - normiert die Normalen vor jeder Beleuchtungsberechnung
    - Vorteil: Funktioniert immer, man muß die Normalen nicht selbst normieren
    - Nachteil: teuer
  - **`glEnable( GL_RESCALE_NORMAL )`**
    - Skaliert die Normale mit der inversen Skalierung, die aus der `GL_MODELVIEW`-Matrix ermittelt wird
- Konkret: Wenn nur Rotation, Translation, uniforme Skalierung verwendet werden und alle Normalen normiert übergeben werden, genügt **`GL_RESCALE_NORMAL`**
  - (manche OpenGL-Implementierungen haben `GL_RESCALE_NORMAL` durch `GL_NORMALIZE` implementiert :-())



<http://www.xmission.com/~nate/>

- glColor hat – sobald GL\_LIGHTING eingeschaltet ist – (per default) keinen Einfluss mehr auf die Objektfarbe
  - Die Farbe wird nur noch durch die Materialeigenschaften und die Farbe der Lichtquelle(n) bestimmt
- Lichtquellen haben keine Geometrie, sind also nicht sichtbar
  - Rendere extra Geometrie, falls sie doch "sichtbar" sein sollen
- Lichtquellen werden nur berücksichtigt, solange sie eingeschaltet sind
  - Somit kann man für verschiedene Objekte verschiedene Lichtquellen aktivieren

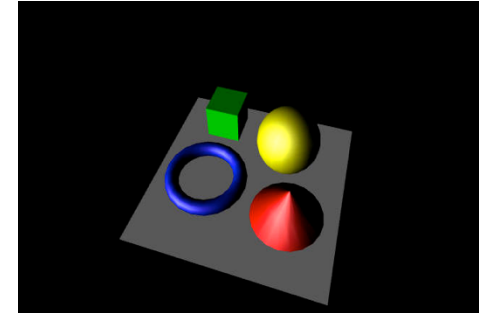
- Die Position (und Richtung) der Lichtquelle wird genauso transformiert wie ein Geometrieprimitiv
  - Transformation mit **GL\_MODELVIEW**
- Lichtquelle in Weltkoordinaten („fest am Objekt“):

```
gluLookAt( ... );  
glLightfv( GL_LIGHT0, GL_POSITION, pos );  
drawObject(...);
```

- Lichtquelle in Kamerakoordinaten („fest an Kamera“):

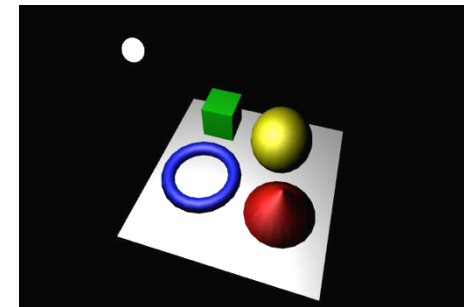
```
glLightfv( GL_LIGHT0, GL_POSITION, pos );  
gluLookAt( ... );  
drawObject(...);
```

- Unterscheidung zwischen **Position** und **Richtung**
- Gerichtete Lichtquellen (*directional lights*):
  - Richtung  $\rightarrow$  homogener Koordinate = 0



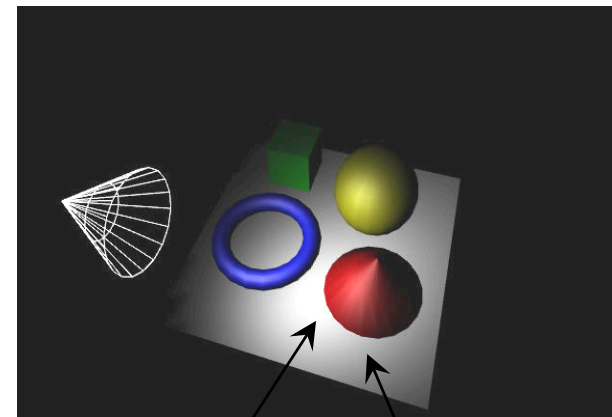
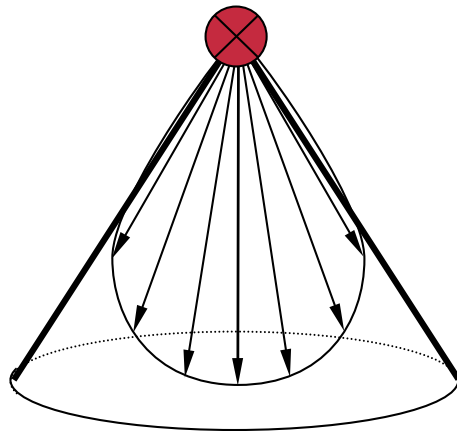
```
GLfloat direction[] = { 0.0, -0.5, 0.5, 0.0 };
glLightfv( GL_LIGHT0, GL_POSITION, direction );
```

- Punktlichtquelle (*point lights*):
  - Position  $\rightarrow$  homogene Koordinate = 1



```
GLfloat position[] = { 0.0, -0.5, 0.5, 1.0 };
glLightfv( GL_LIGHT0, GL_POSITION, position );
```

- Nur um einen bestimmten Winkel um eine angegebene Richtung wird Licht ausgestrahlt
- Je weiter von der Richtung weg, desto schwächer wird das Licht ( $\cos^n$ -Verteilung)
- Parameter: Position, Richtung, Exponent, Farbe
- Details: Siehe “Red Book”



Hotspot

Fall-off

# Weitere Parameter des Beleuchtungsmodells

- Über

```
glLightModeli( GLenum name, value )
```

- GL\_LIGHT\_MODEL\_LOCAL\_VIEWER (Bool):

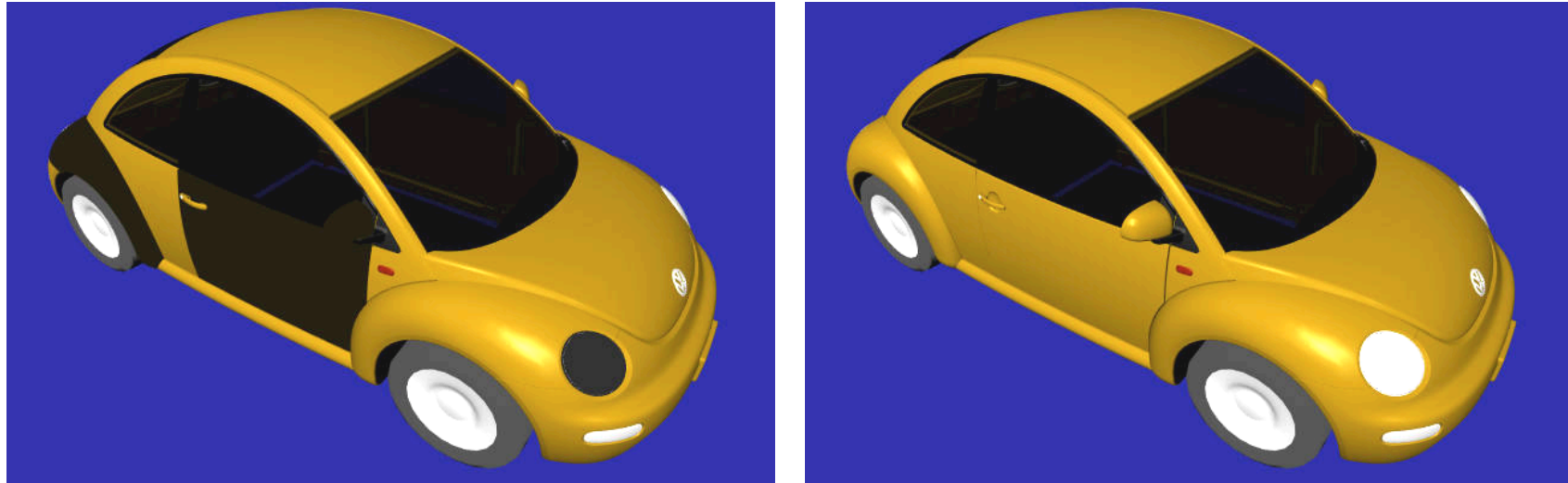
- GL\_FALSE (default): der Augpunkt wird als unendlich weit weg angenommen, d.h., der Half-Vector = const. (ergibt schnelleres Rendering für *directional lights*)
- GL\_TRUE: der Augpunkt liegt bei (0,0,0) und der reflektierte Lichtstrahl bzw. Half-Vector wird für jeden Vertex neu berechnet

- GL\_LIGHT\_MODEL\_TWO\_SIDE (Bool):

- Abgewandte Normalen (*back-facing polygons*) werden per Default ignoriert
- Mit "two-sided lighting" werden Normalen von back-facing Polygonen umgedreht, d.h., Beleuchtung ist von vorne wie von hinten gleich



# Vergleich zwischen *single-sided* und *two-sided lighting*



- Achtung: es hängt vom konkreten Fall ab, welche Option sinnvoll ist!
- Es ist tatsächlich nicht immer trivial, die Normalen "richtig" herum zu drehen, wenn die Geometrie vorgegeben ist ...
- Der zusätzliche Test bei *two-sided lighting* kostet bis zu 20% Performance!

# Abschwächung (*attenuation*)

- Punkt- oder Spotlichtquellen können ihre Stärke abhängig von der Entfernung  $d$  zur Oberfläche abschwächen:

$$I = M_e + M_a I_a + \sum_{j=1}^n a_j \cdot (M_a L_{j,a} + M_d \cos \Phi_j L_{j,d} + M_s \cos^n \Theta_j L_{j,s})$$

- Eigentlich ist Abschwächung  $\sim 1/d^2$ ; hat aber keine schönen Effekte
- Daher verwendet OpenGL ein Modell mit mehr Parametern:

$$a = \frac{1}{k_c + k_l \cdot d + k_a \cdot d^2}$$

- $d$ : Abstand zwischen Lichtquelle und dem Eckpunkt

```
glLightf( GL_LIGHT0, GL_CONSTANT_ATTENUATION, kc );
glLightf( GL_LIGHT0, GL_LINEAR_ATTENUATION, kl );
glLightf( GL_LIGHT0, GL_QUADRATIC_ATTENUATION, kq );
```

# Atmosphärische Dämpfung (Fog)

- Lineare Abnahme der Intensität beim Mischen mit Partikeln (z.B. Nebel)

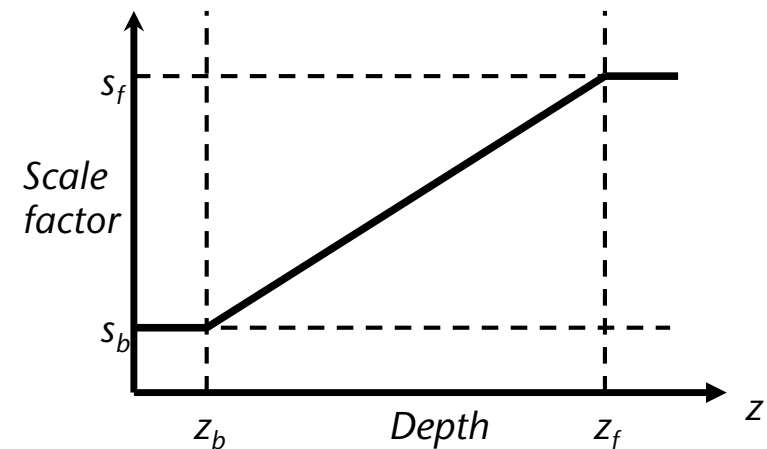
$$I = s(z)I_{obj} + (1 - s(z))I_{fog}$$

$$s(z) = s_b + \frac{z - z_b}{z_f - z_b} \cdot (s_f - s_b)$$

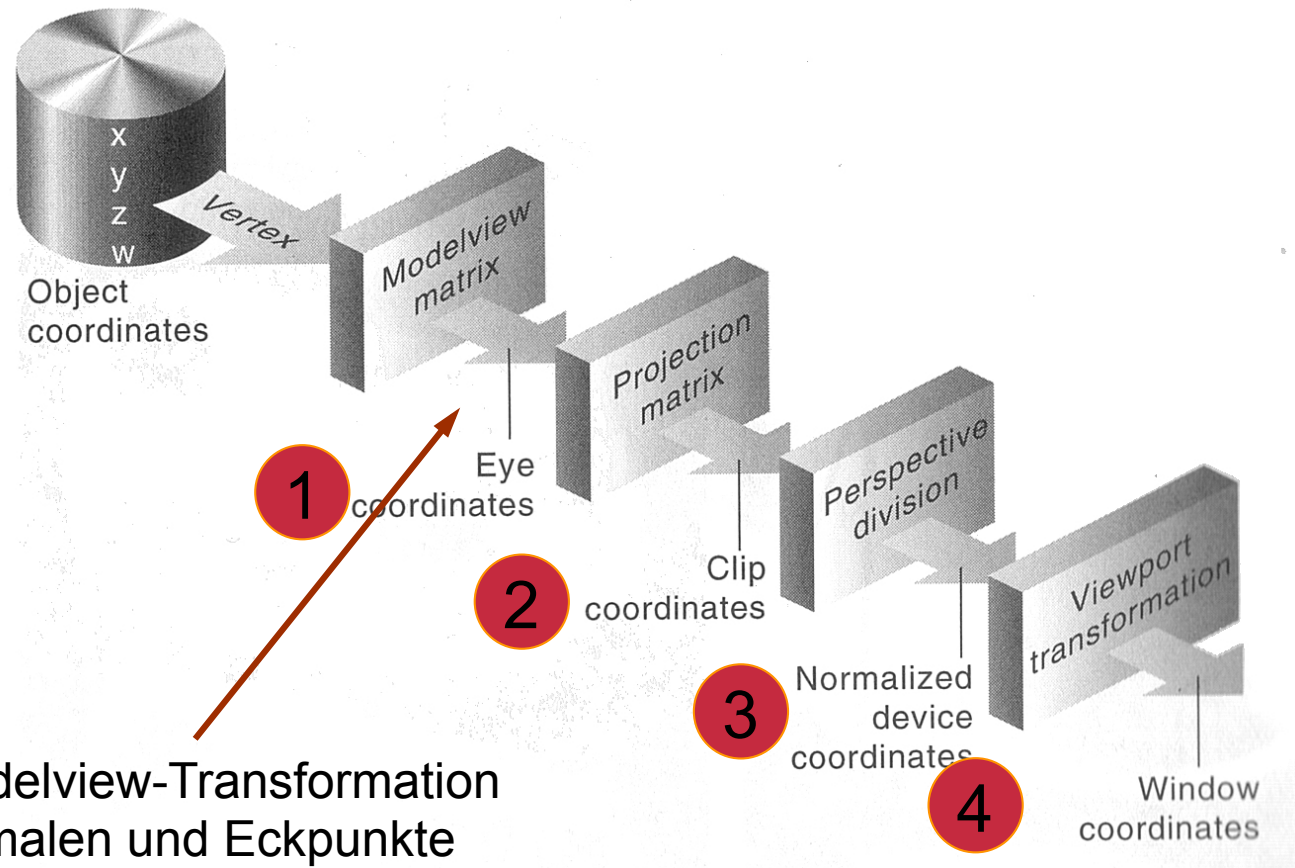
mit  $z_b \leq z \leq z_f$



- Wird von OpenGL unterstützt (s. "Red Book")



# Wo wird in der Graphik-Pipeline beleuchtet?



Nach der Modelview-Transformation sind alle Normalen und Eckpunkte im (gemeinsamen) Kamerakoordinatensystem. Diese Werte werden für die Beleuchtung verwendet.