Reflectance Models in General

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

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

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

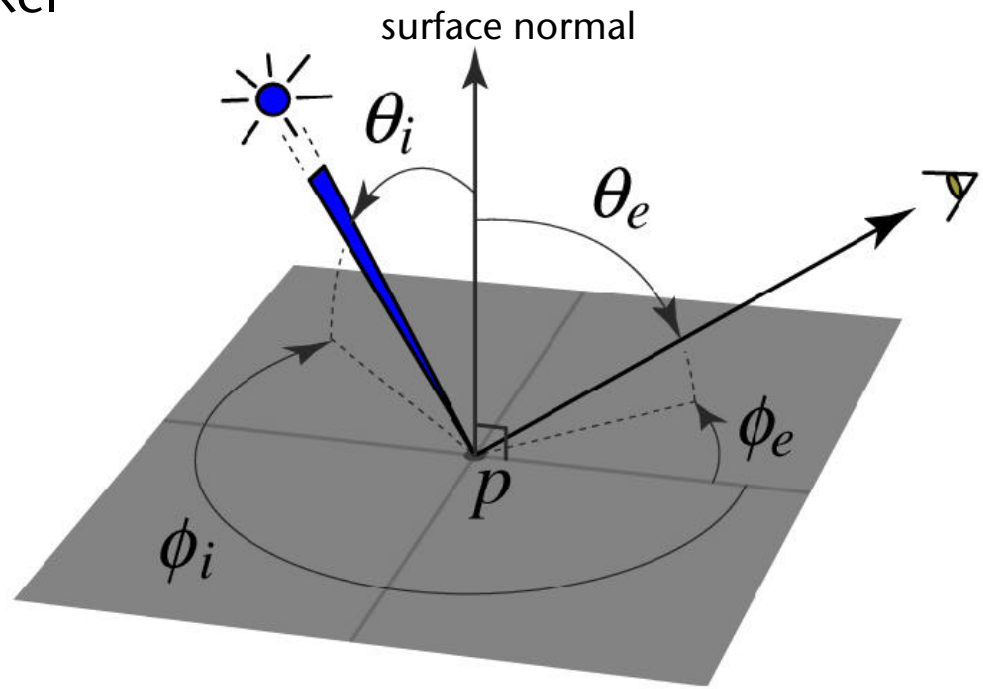
- Echte Matierialien sind i.A. viel komplizierter!

BRDFs als Verallgemeinerung

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

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

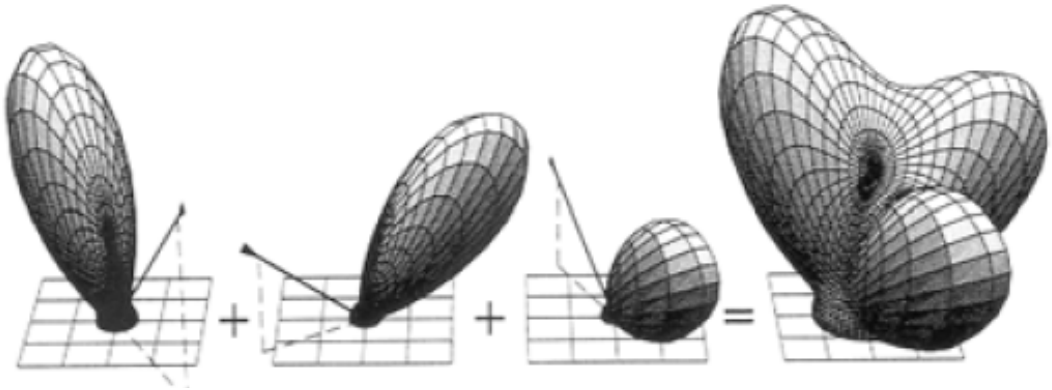
dabei sind (θ_i, ϕ_i) die Einfallswinkel,
und (θ_o, ϕ_o) die Ausfallswinkel



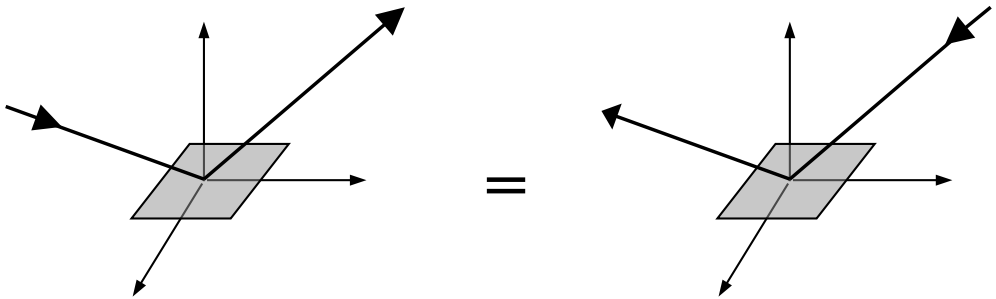
Generelle Eigenschaften von BRDFs

- Sind eigentlich Annahmen, die zu Vereinfachungen führen

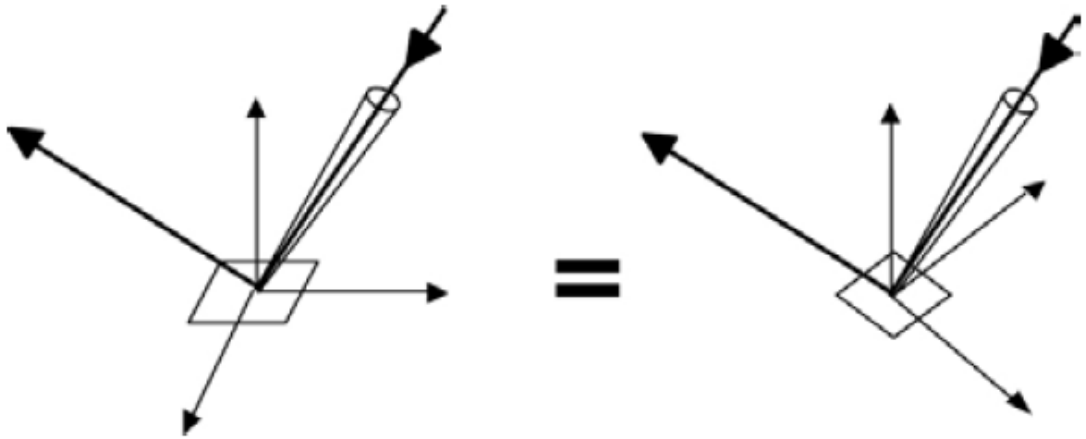
1. Linearität:



2. Reciprozität (reciprocity):

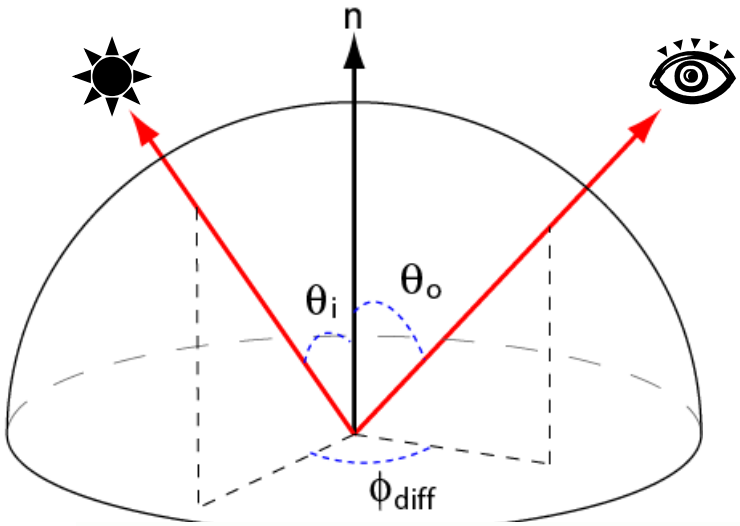


1. Isotropie (isotropy):
   BRDF ist invariant bzgl Rotation des Materials um die Normale

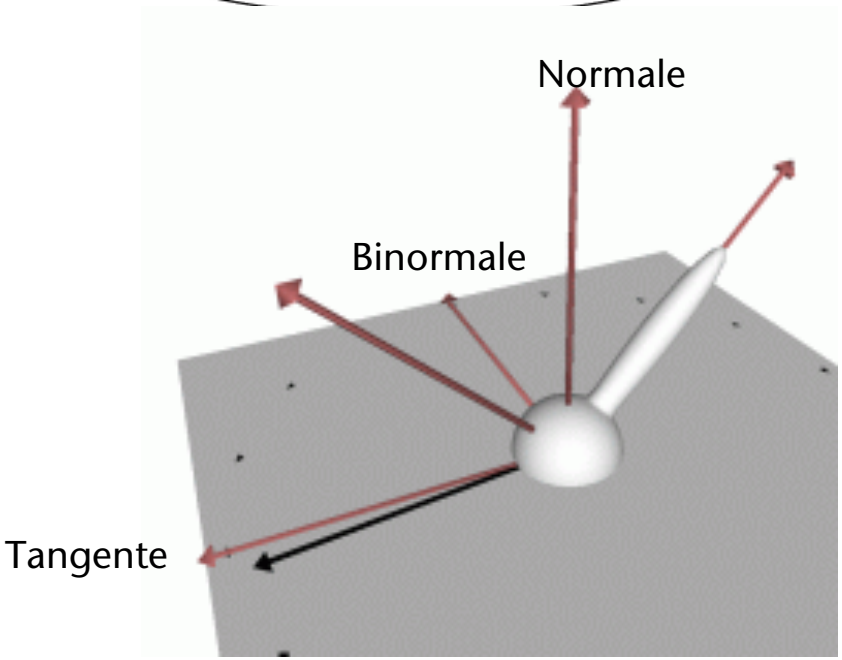


2. Energieerhaltung (conservation of energy):
   quantity of outgoing light ≤ quantity of incoming light
   (m.a.W.: Integral über die BRDF ≤ 1)
3. Die BRDF eines Materials hängt *nicht* von dem jeweiligen Punkt auf der Oberfläche ab!

- Konsequenz: die BRDF ist eigentlich nur eine 3D-Funktion



- Wähle im folgenden dieses spezielle Koordinatensystem zur Auswertung einer BRDF in einem Punkt



Das Lafortune-Modell

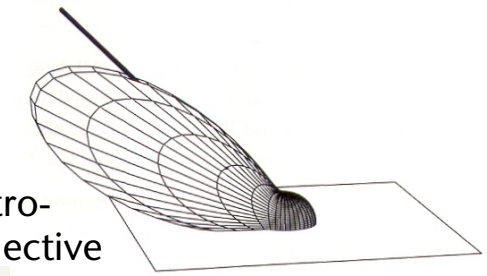
- Das Phong-Modell in diesem Koordinatensystem ist

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

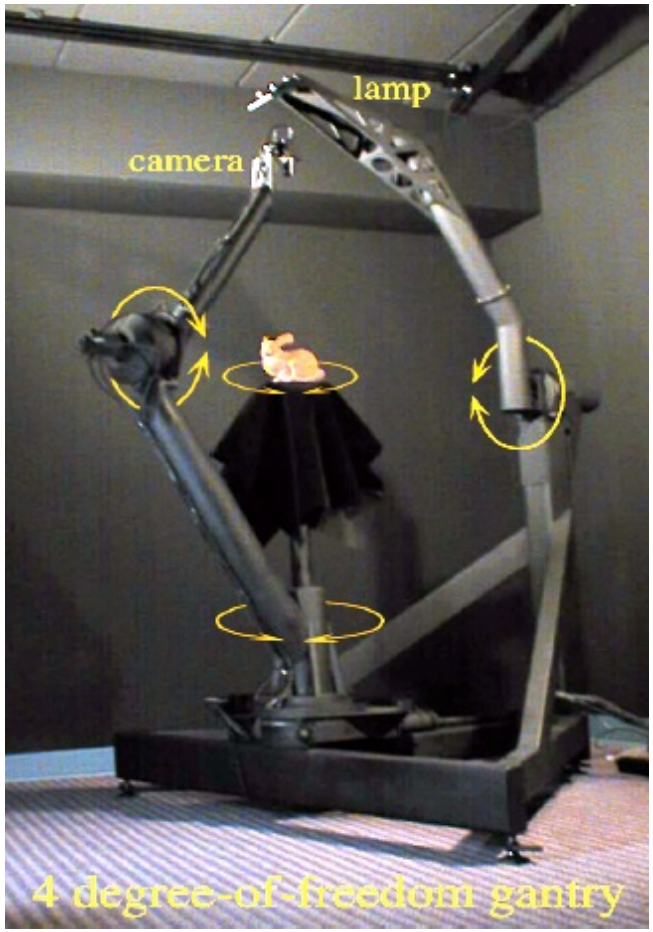
- Das Lafortune-Modell:
  - Ersetze die spezielle Matrix duch eine beliebige Matrix C
  - Erlaube beliebig viele "*Lobes*" (= Keulen, Lappen)
  - Zusammen:

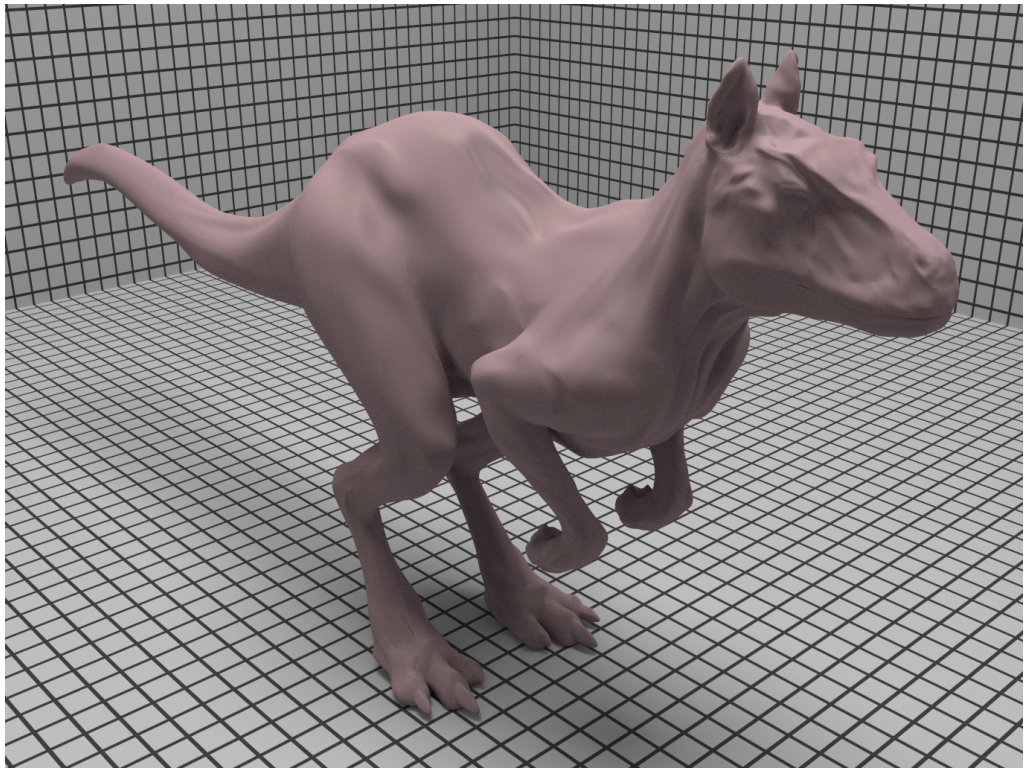
$$\rho(\boldsymbol{l}, \mathbf{e}) = r_d(\mathbf{n} \cdot \boldsymbol{l}) + \sum_{i=1}^{n} r_{s,i}(\mathbf{e} \cdot C_i \cdot \boldsymbol{l})^{p_i}$$

wobe n = Anzahl *lobes*



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





Mit ausgemessenem Ton

Spectral Lighting vs RGB-Lighting

- Erinnerung: alle photometrischen Größen sind eigtl.

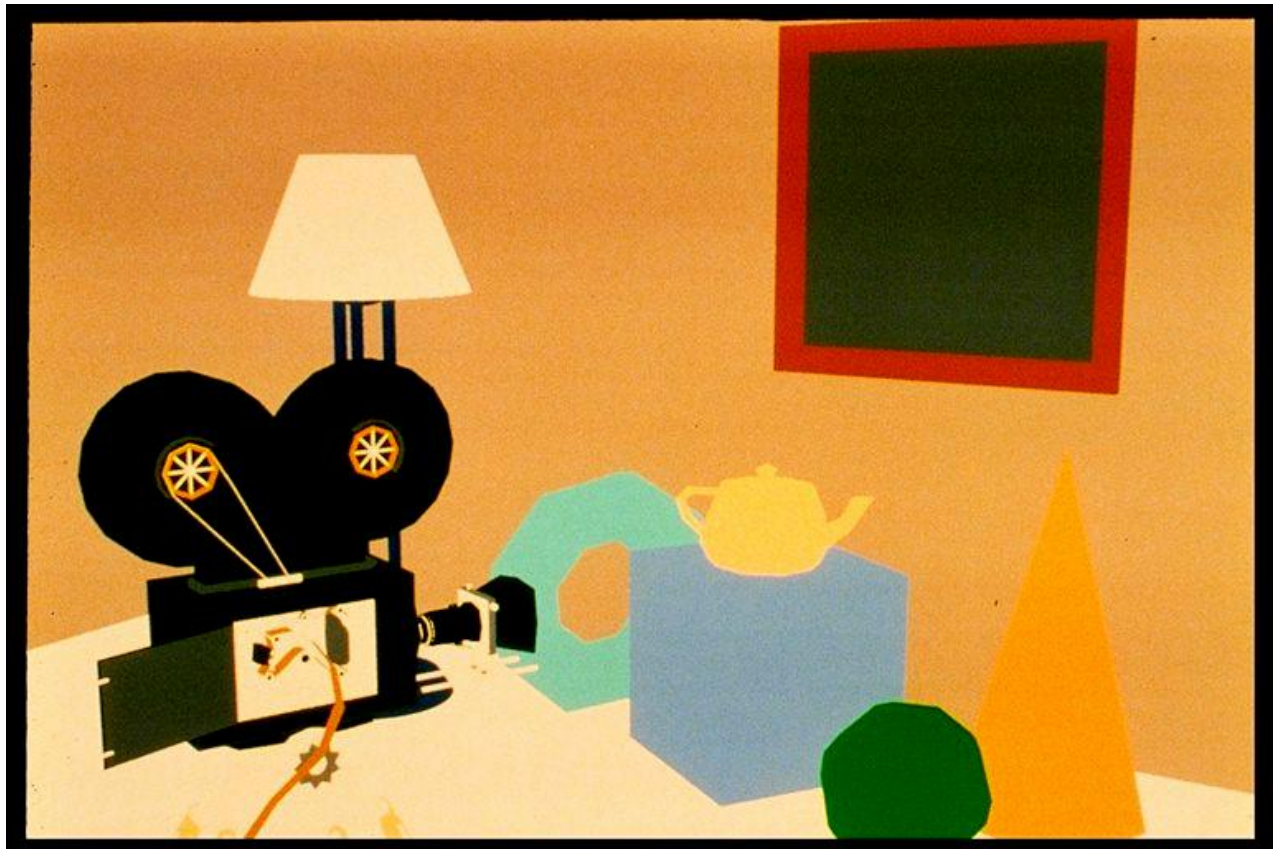
  Funktionen in λ! beschreiben also ein Spektrum ...
- In der Praxis (z.B. OpenGL): führe alle Berechnungen jeweils für die 3 Primärvalenzen (z.B. *r, g, b*) durch
- Aber: dadurch erhält man nicht 100% korrekte Bilder!
- Denn:

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

Shading-Algorithmen

- Achtung: unterscheide zwischen Beleuchtungs**modell** (*lighting model)* und Beleuchtungs**algorithmus** (*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

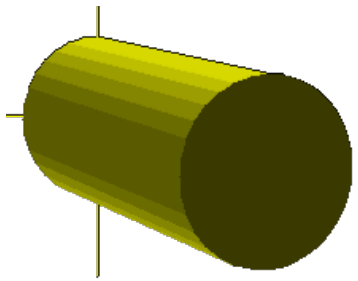
- Die Szene ganz ohne Shading ...

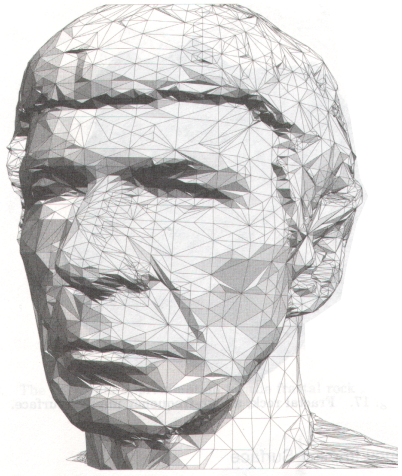


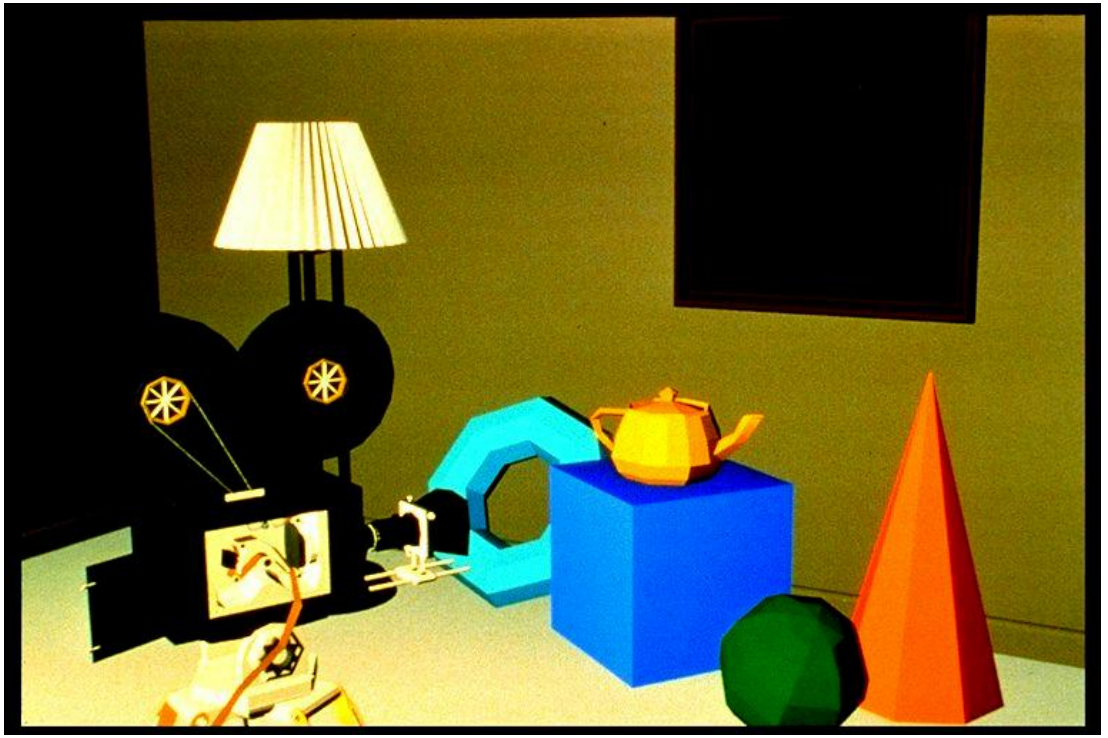
Pixar "Shutterbug"

Flat Shading (Konstante Beleuchtung)

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



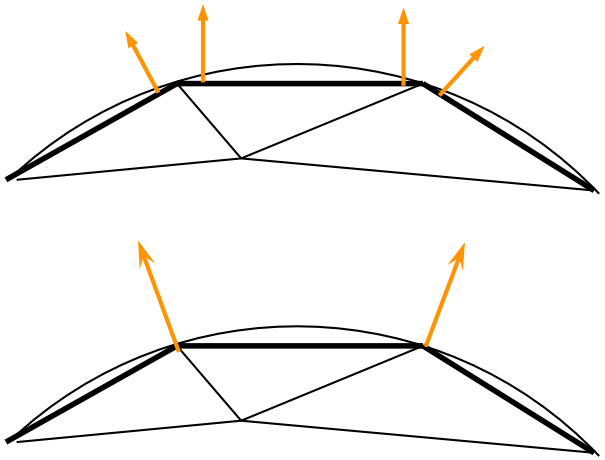


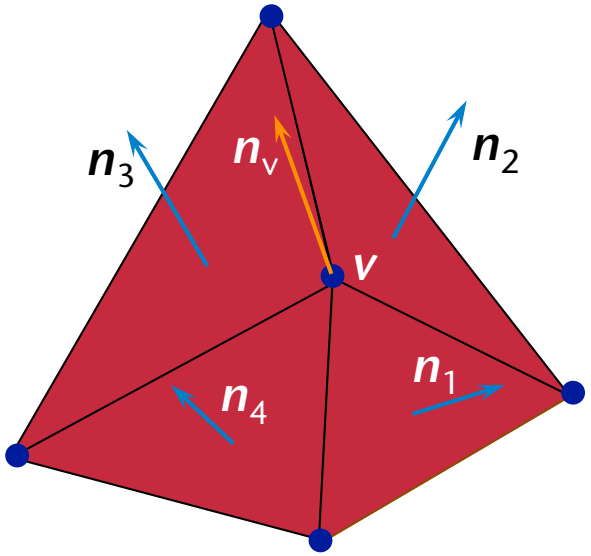


Pixar "Shutterbug"

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

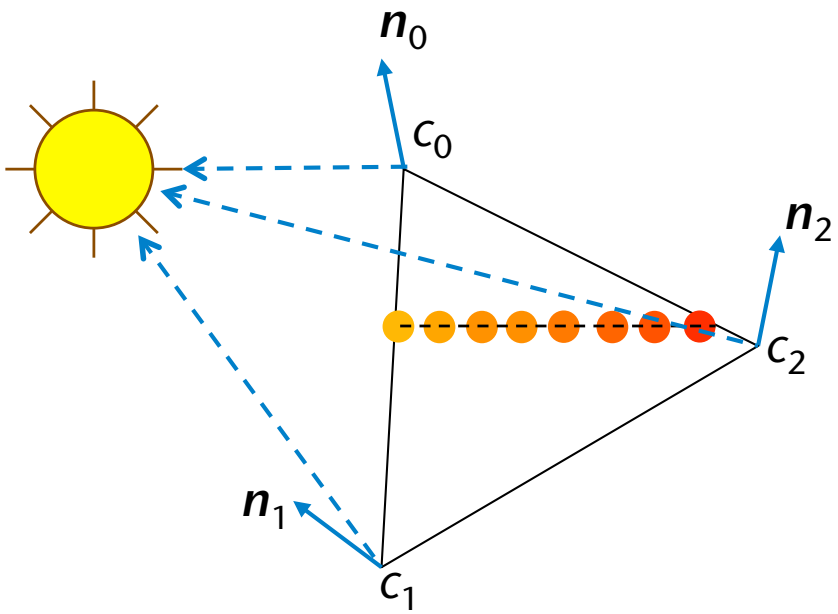


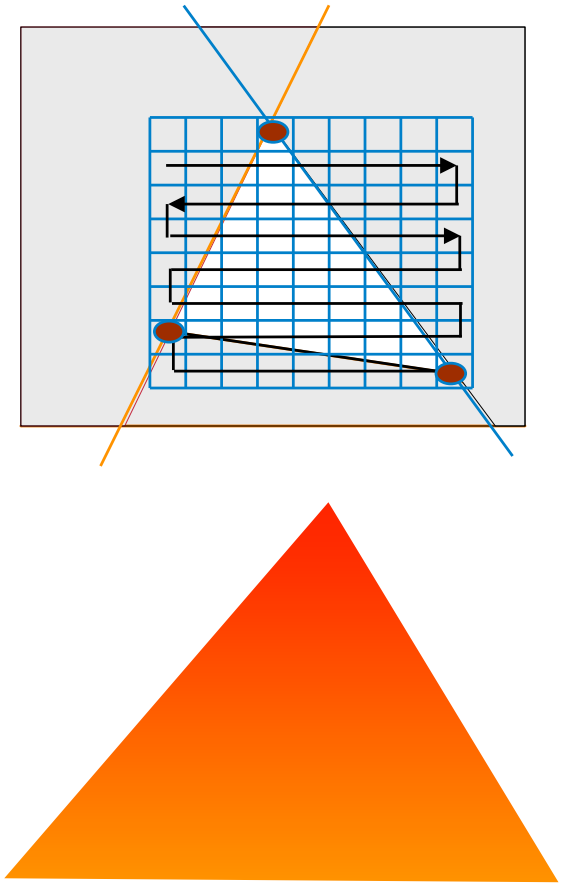


Gouraud-Shading

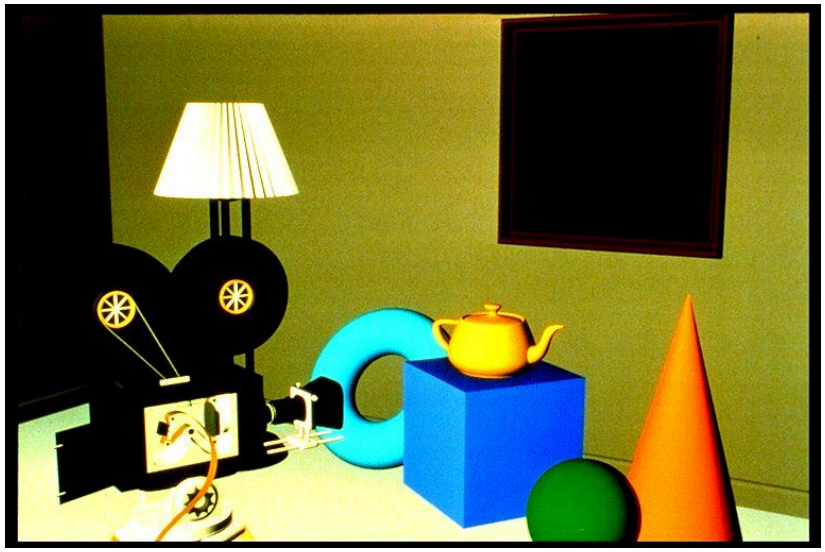
[1971]

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

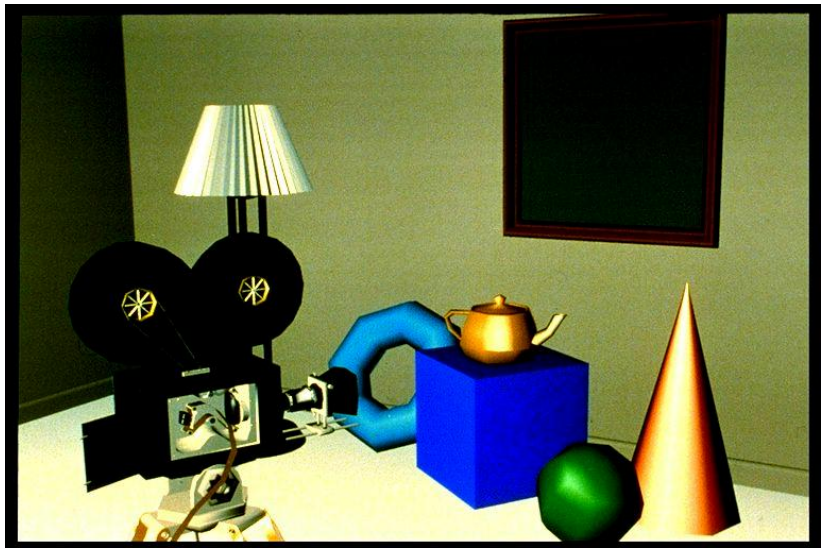




Vergleiche



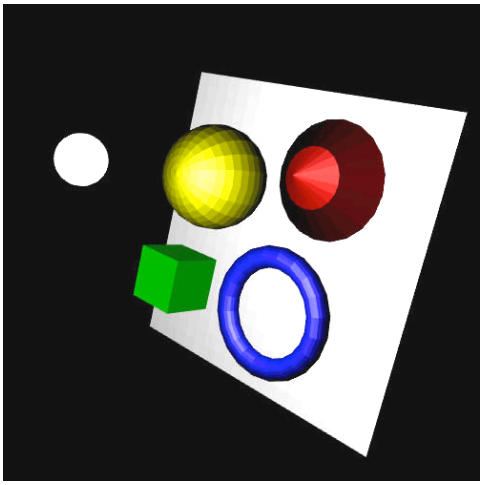
Gouraud-Shading mit
rein diffusem Beleuchtungsmodell

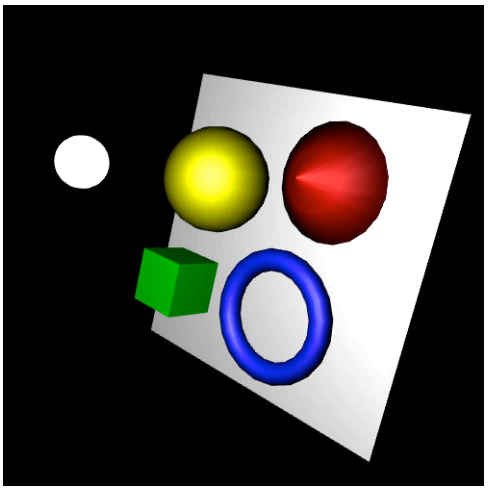


Gouraud-Shading mit
Phong-Modell

Flat-Shading

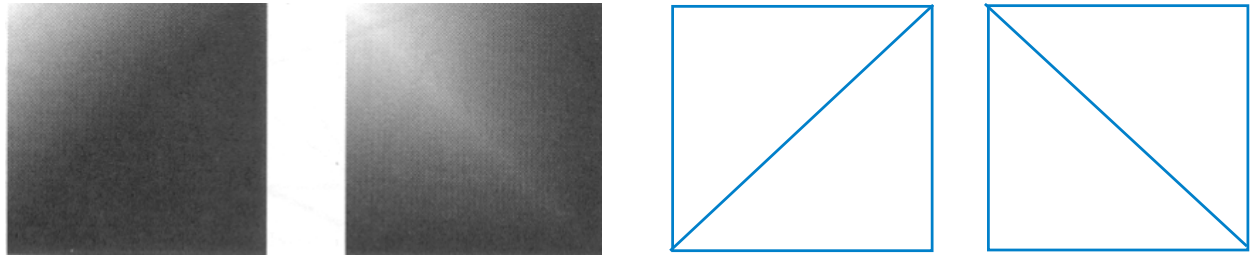
Gouraud-Shading mit
Phong-Lighting





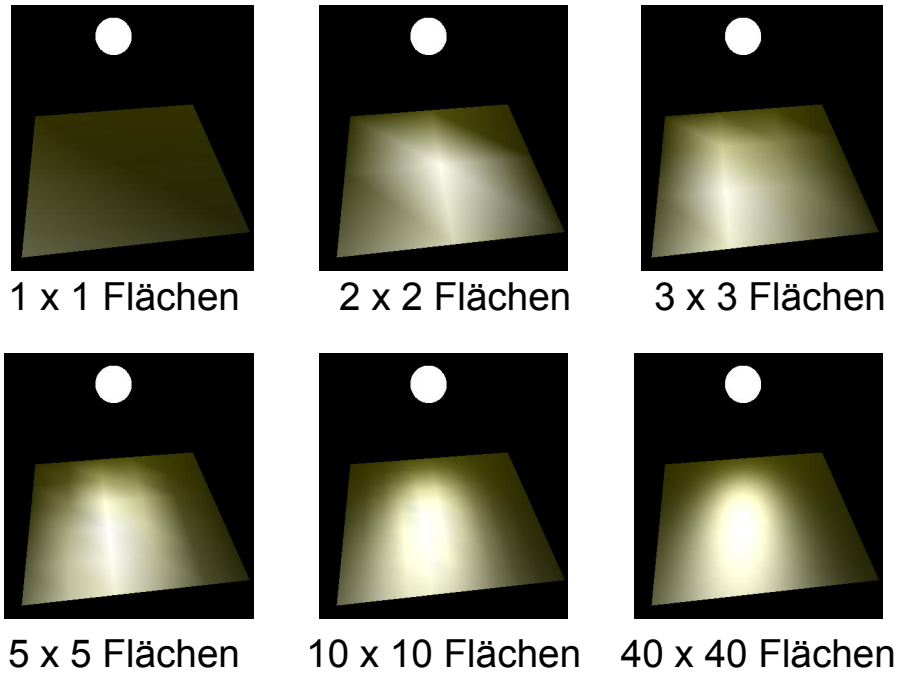
Einfluß der Triangulierung

- Orientierung:



- Hier gilt: "size matters"

Vergrößerung



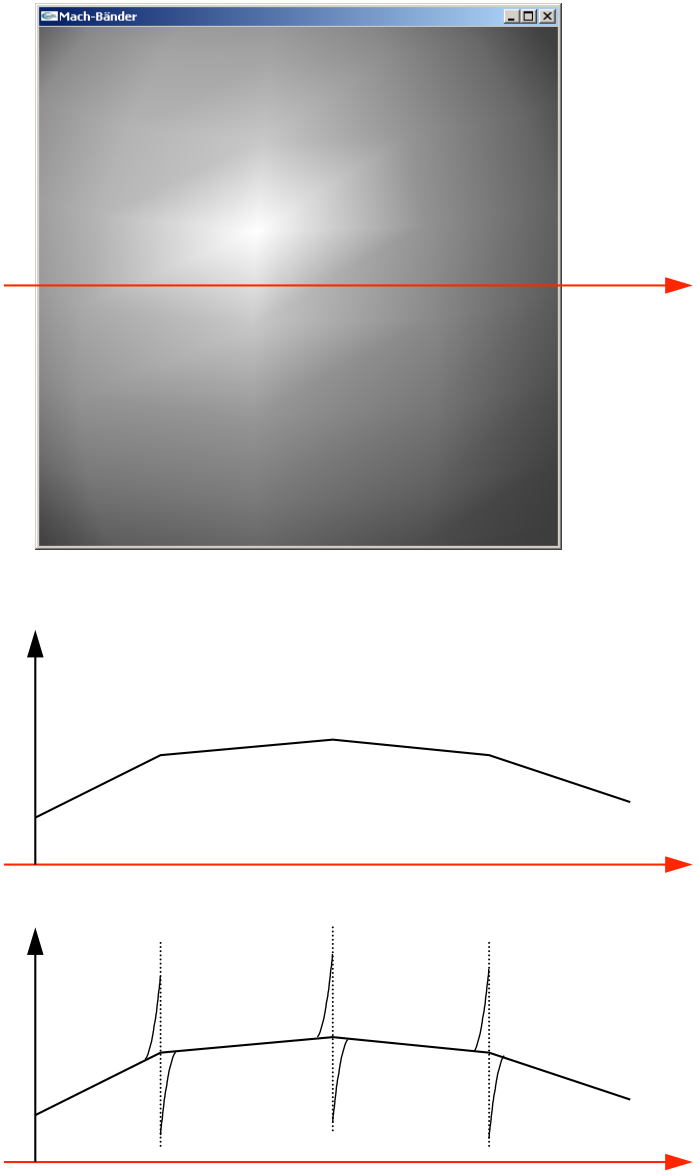
1 x 1 Flächen | 2 x 2 Flächen | 3 x 3 Flächen

5 x 5 Flächen | 10 x 10 Flächen | 40 x 40 Flächen

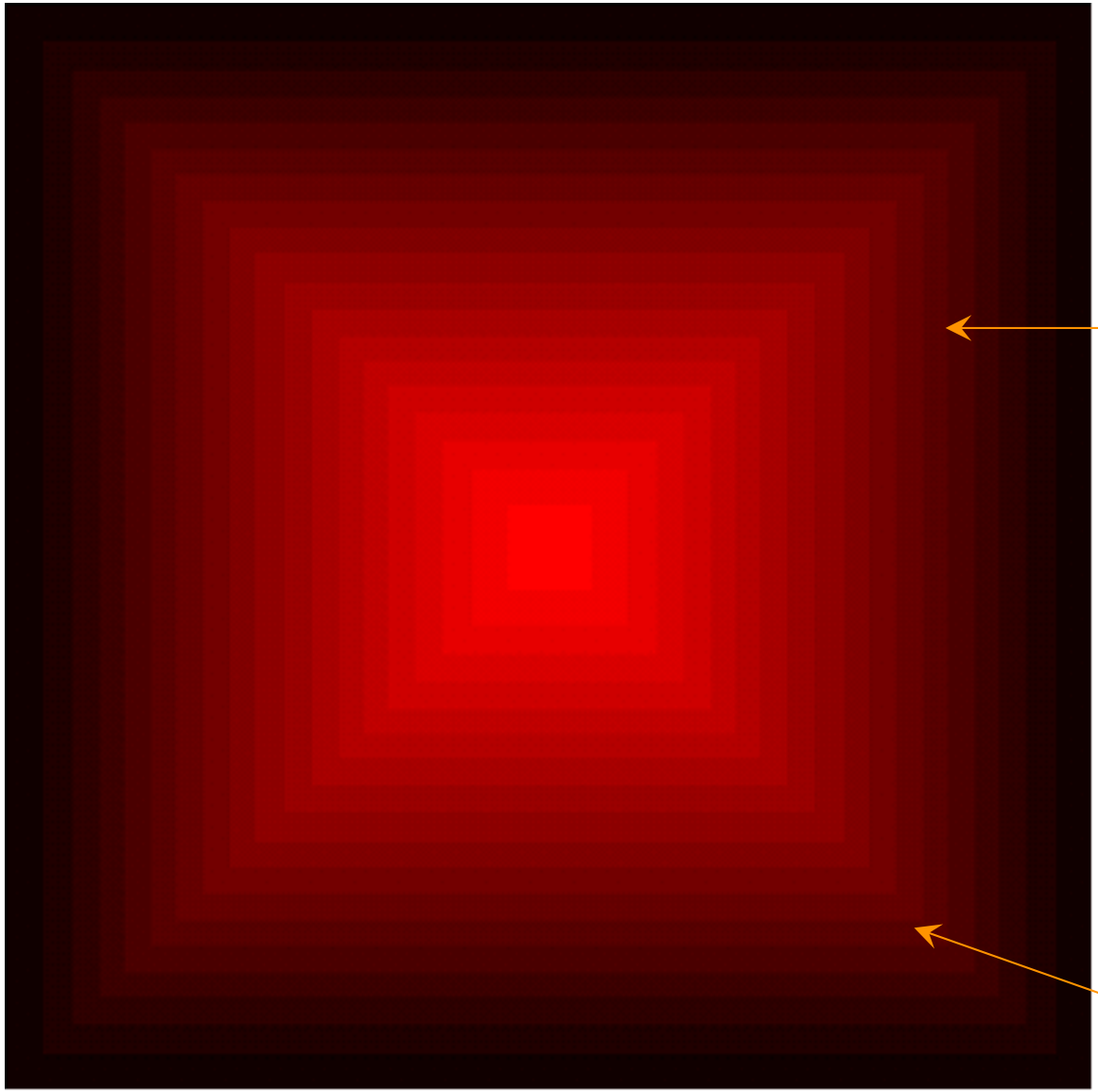
- Frage: Woher kommen die „Streifen“?

Mach-Bänder

- 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

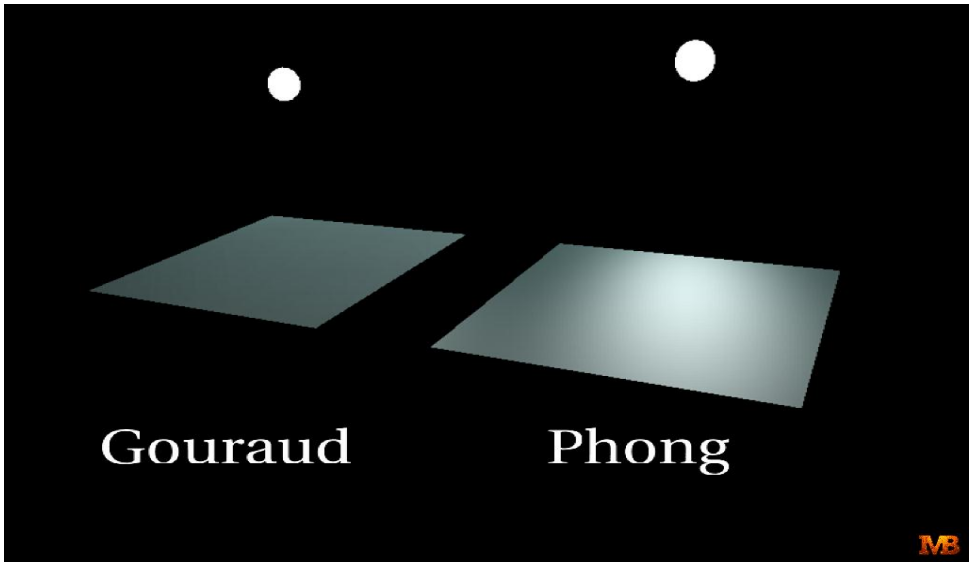


Die Intensität innerhalb eines Quadrates ist konstant!
Ein Farbverlauf von innen nach außen ist eine Illusion.

Die hellen Linien bei 45° (und 135°) sind eine Illusion!

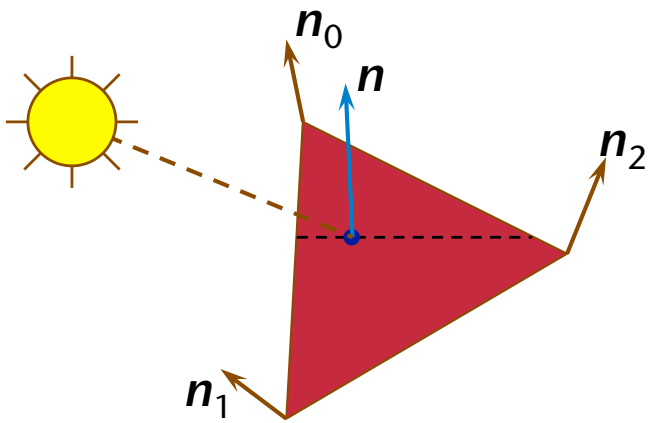
Weiteres Problem beim Gouraud-Shading

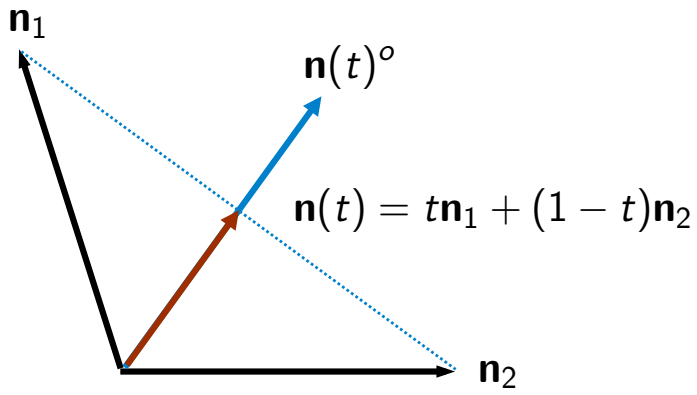
- Evtl. "verpasst" man Highlights im Inneren eines Polygons:

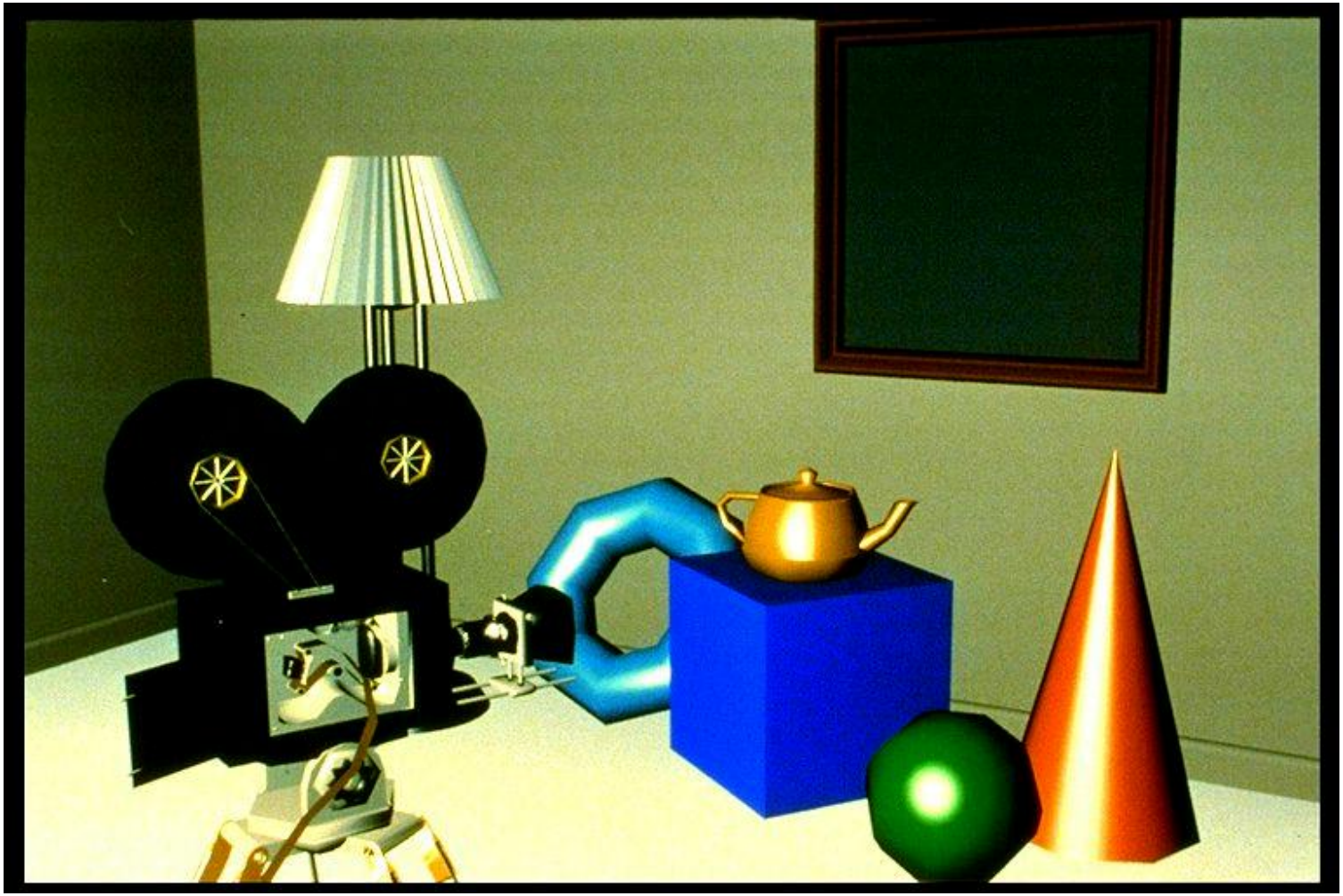


Phong-Shading

- 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 Freiheitsgraden
- Jede Lichtquelle L_i in OpenGL besteht aus 3 Teilen: ambienter ($I_{i,a}$), diffuser ($I_{i,d}$), und spekularer ($I_{i,s}$) 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
- Insgesamt:

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

- Werte größer 1 werden auf 1 "geclampt"

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 );

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

glEnable( GL_LIGHTING );
glEnable( GL_LIGHT0 );
```

Shading-Algo (Gouraud) einschalten

Lichtquelle "Nr. 0" definieren

Beleuchtung einschalten

Lichtquelle „Nr. 0“ einschalten

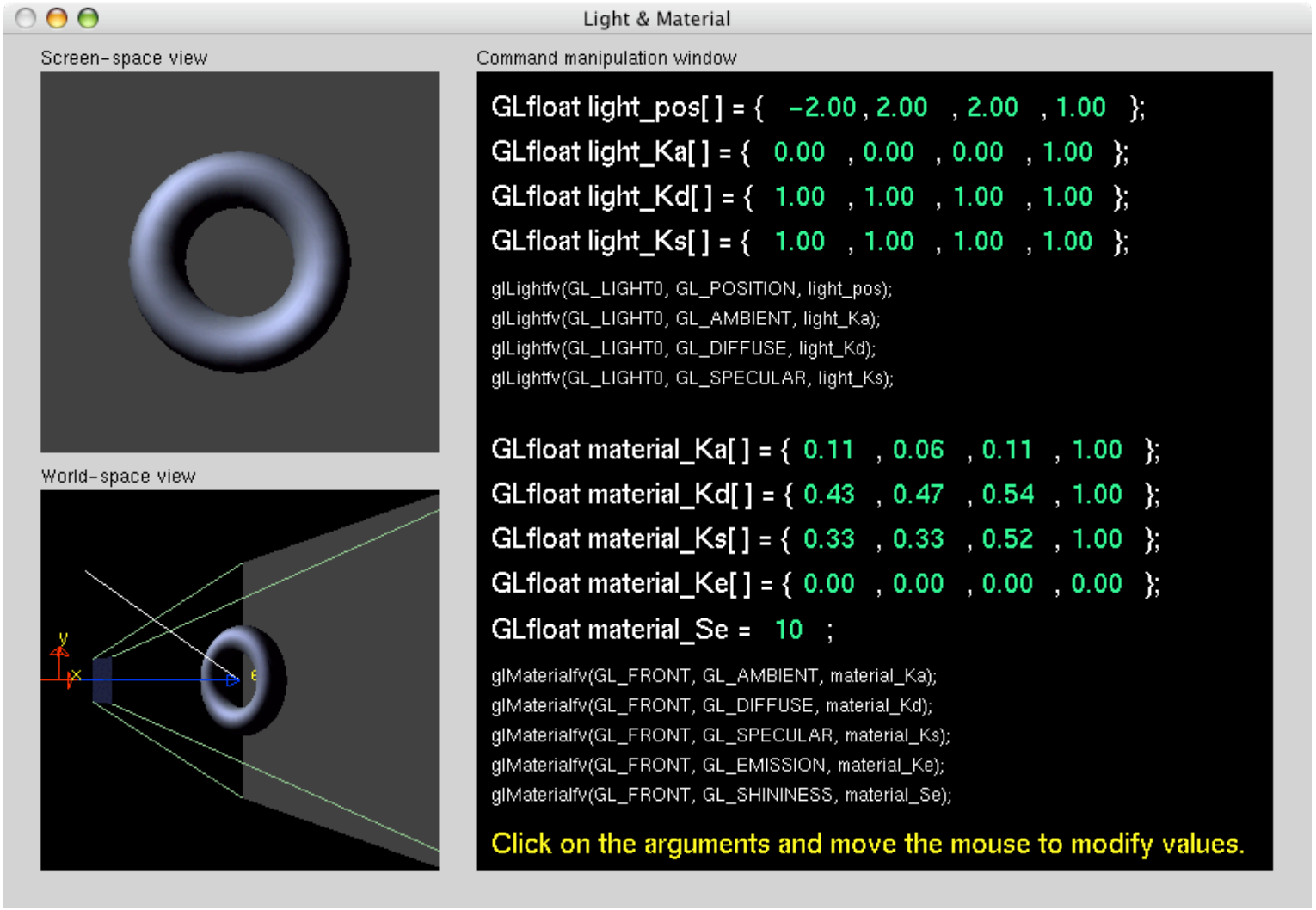
Materialdefinition

```
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(…);
```

Demo

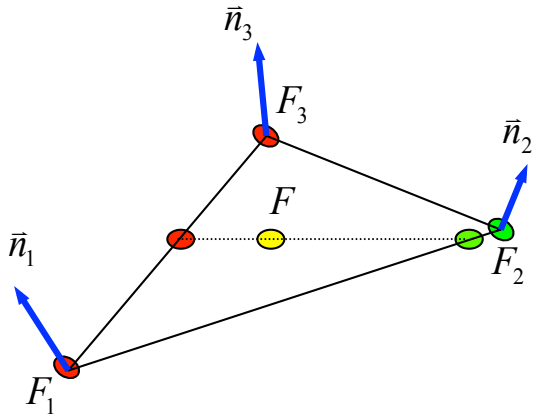


http://www.xmission.com/~nate/

Flat- vs. Gouraud-Shading

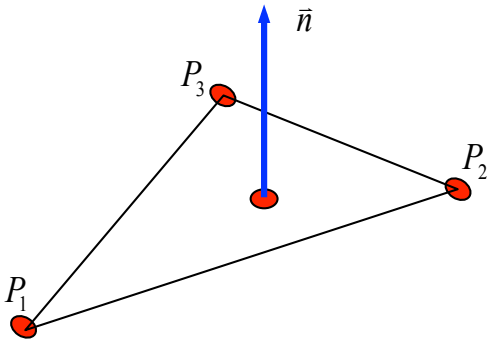
- Gouraud-Shading:

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



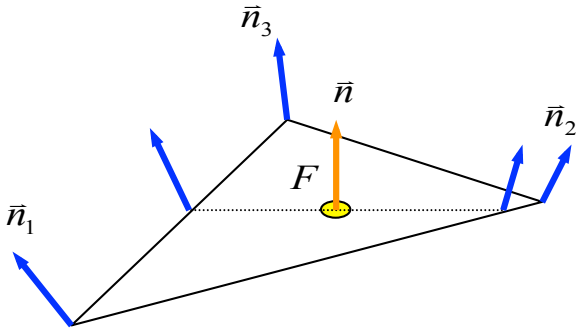
- Flat-Shading:

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



- Phong-Shading:

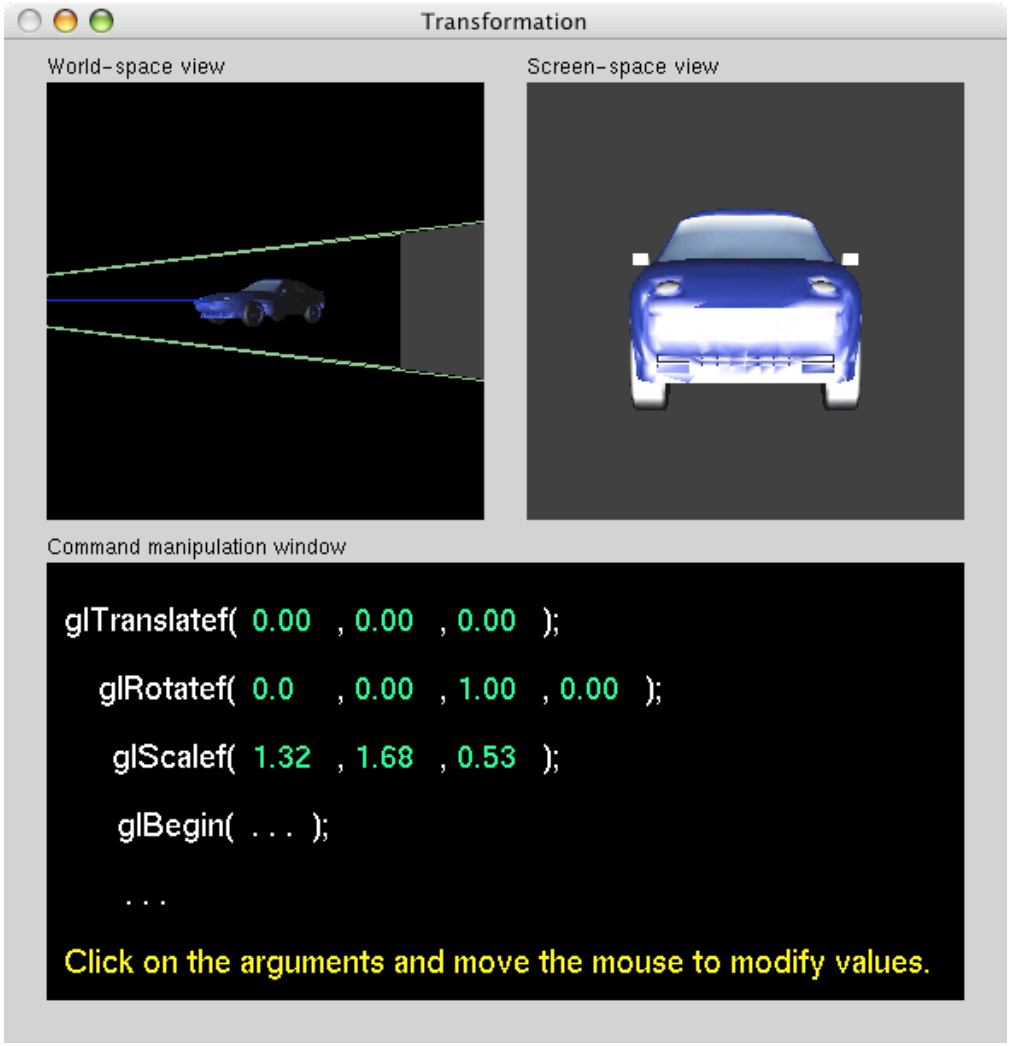
```
// Nur mit Shadern möglich …
```



Normalen

- 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 :-( )

Effekt von nicht-normierten Normalen in der Beleuchtung



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

Anmerkungen

- 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

Position der Lichtquelle

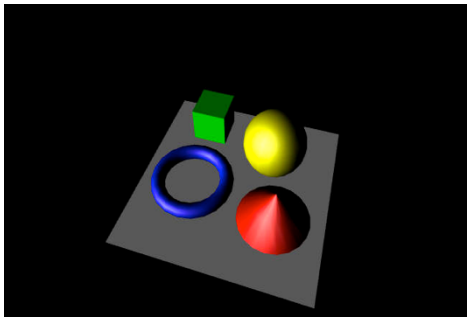
- Die Position (und Richtung) der Lichtquelle wird genauso transformiert wie ein Geometrieprimitiv
  - Transformation mit **GL_MODELVIEW**
- Lichtquelle in Weltkoordinaten („fest am Objekt“):

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

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

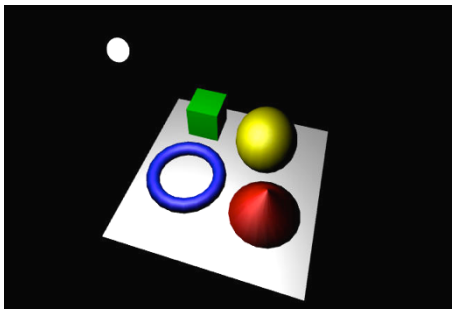
```
glLightfv( GL_LIGHT0, GL_POSITION, pos );
gluLookAt( …);
drawObject(…);
```

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



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

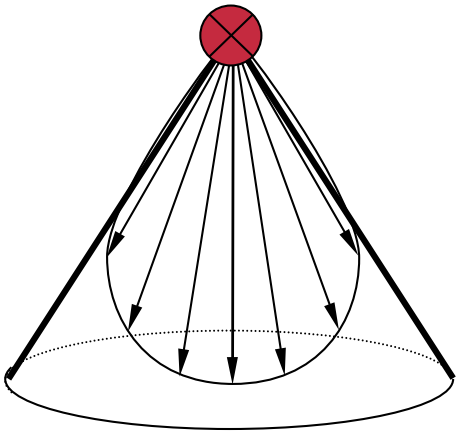
- Punktlichtquelle (*point lights*):
  - Position → homogene Koordinate = 1

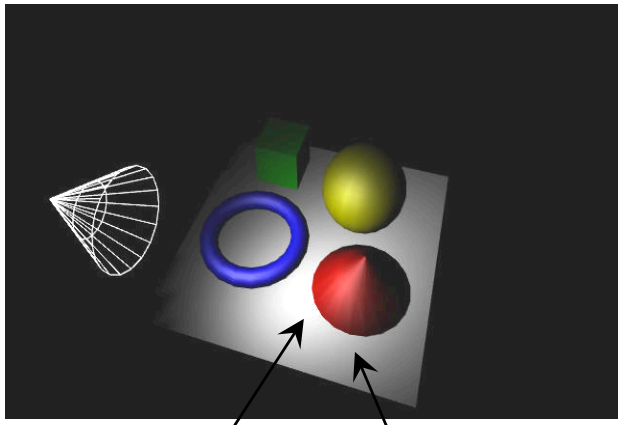


```
GLfloat position[] = { 0.0, -0.5, 0.5, 1.0 };
glLightfv( GL_LIGHT0, GL_POSITION, position );
```

Spotlight

- 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”





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 ~ $1/d^2$; 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)

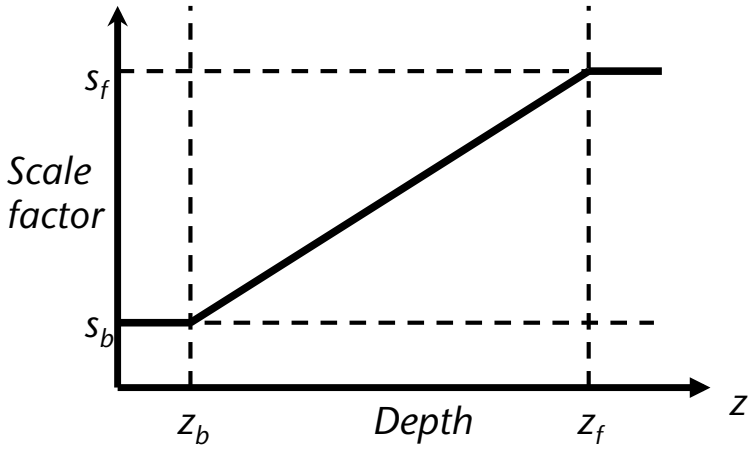
- Lineare Abnahme der Intensität beim Mischen mit Partikeln (z.B. Nebel)

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

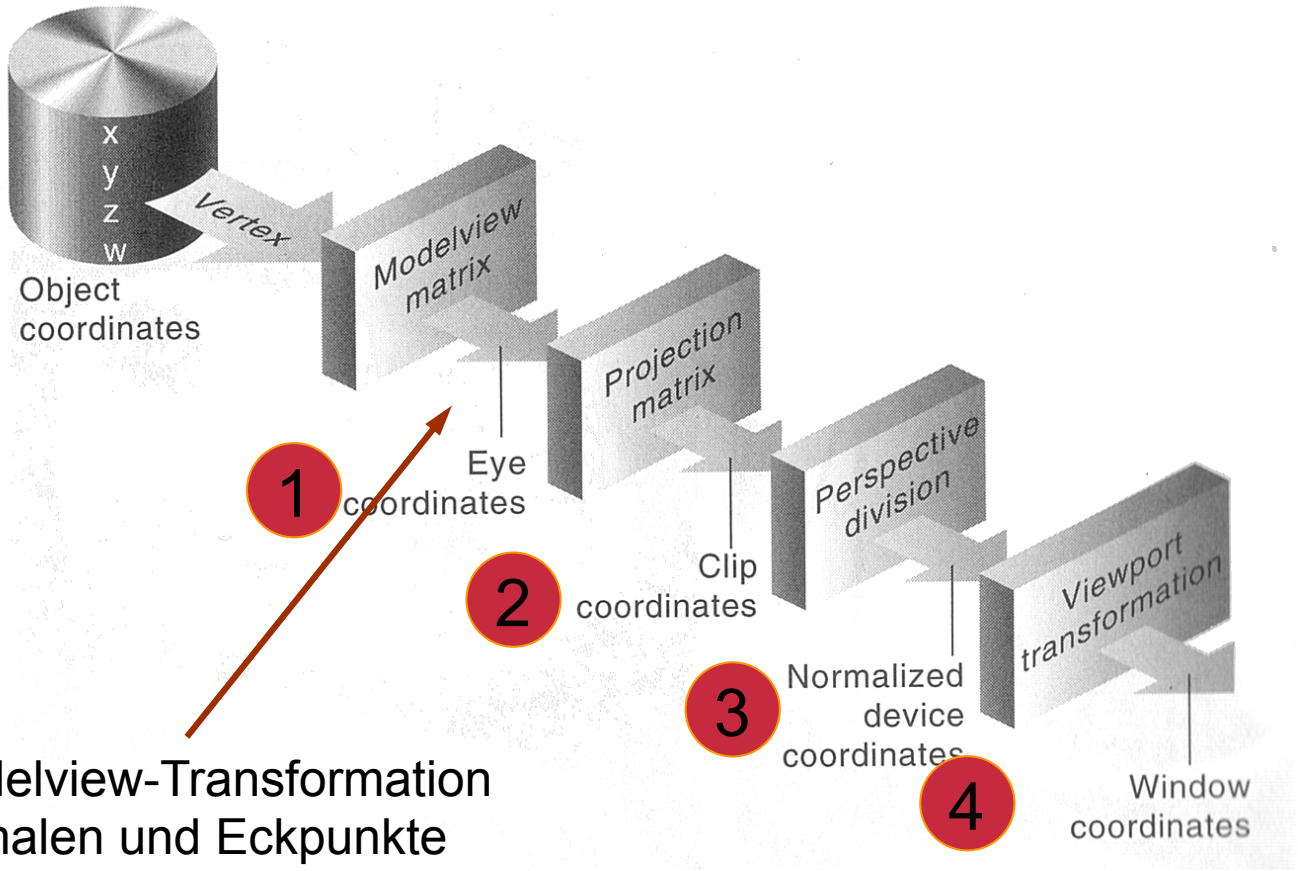
$$s(z) = s_b + \frac{z - z_b}{z_f - z_b} \cdot (s_f - s_b)$$

$$\text{mit } z_b \leq z \leq z_f$$

- 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) Kamera-koordinatensystem. Diese Werte werden für die Beleuchtung verwendet.