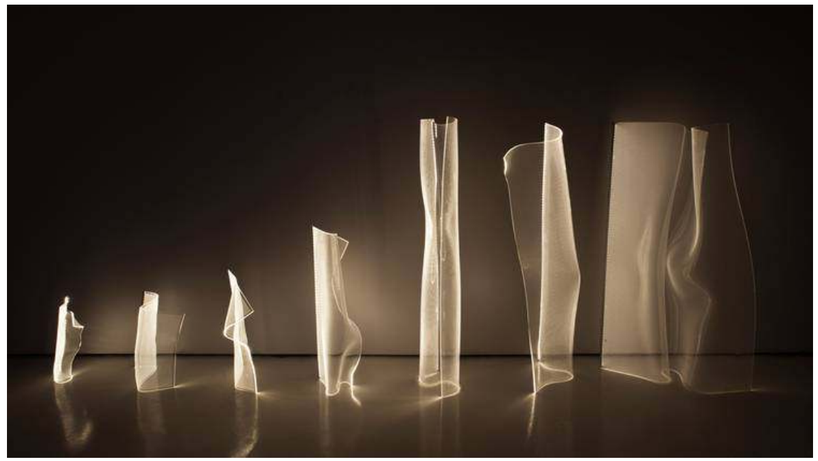
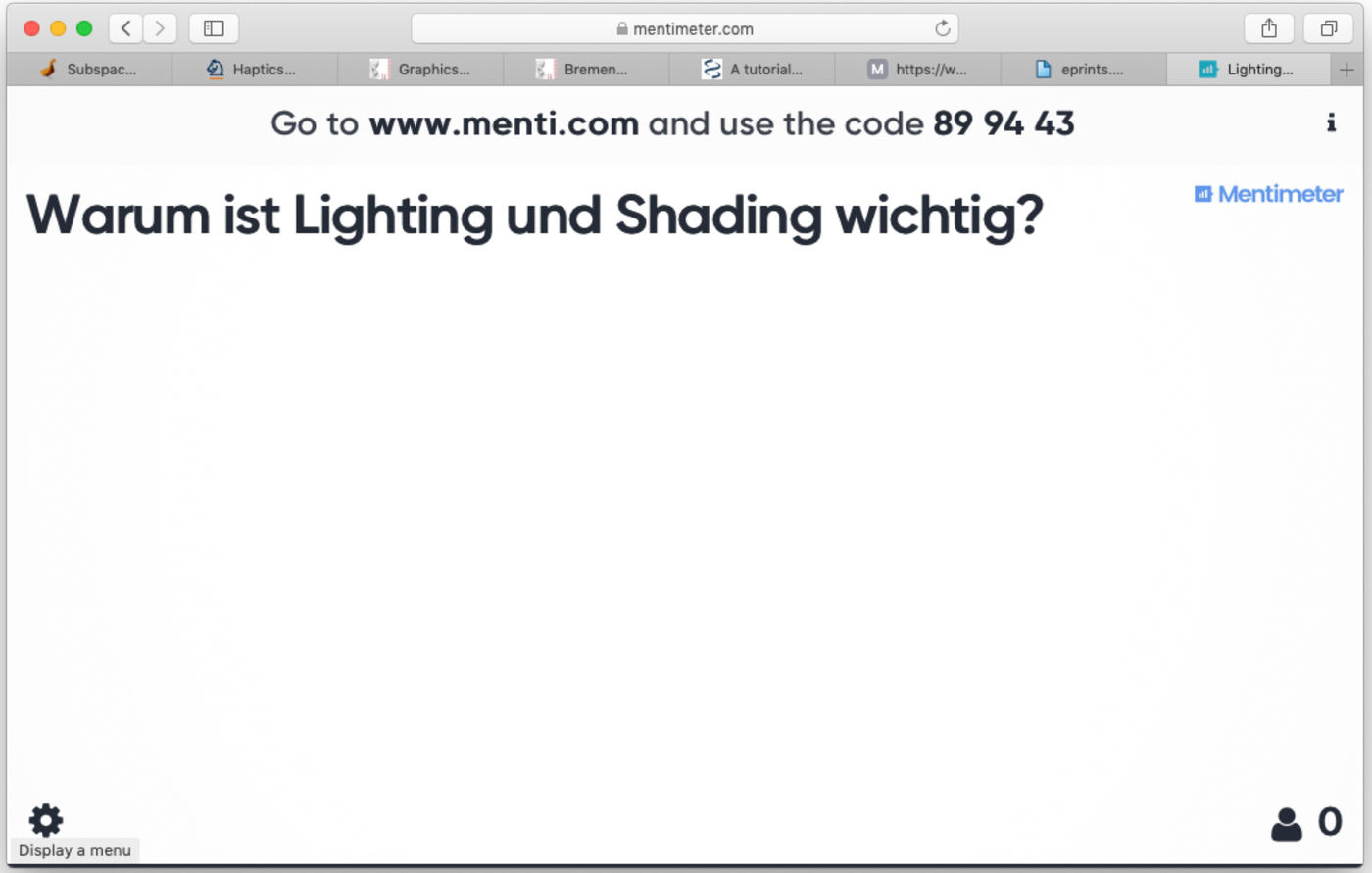
Computer-Graphik 1
Lighting & Shading



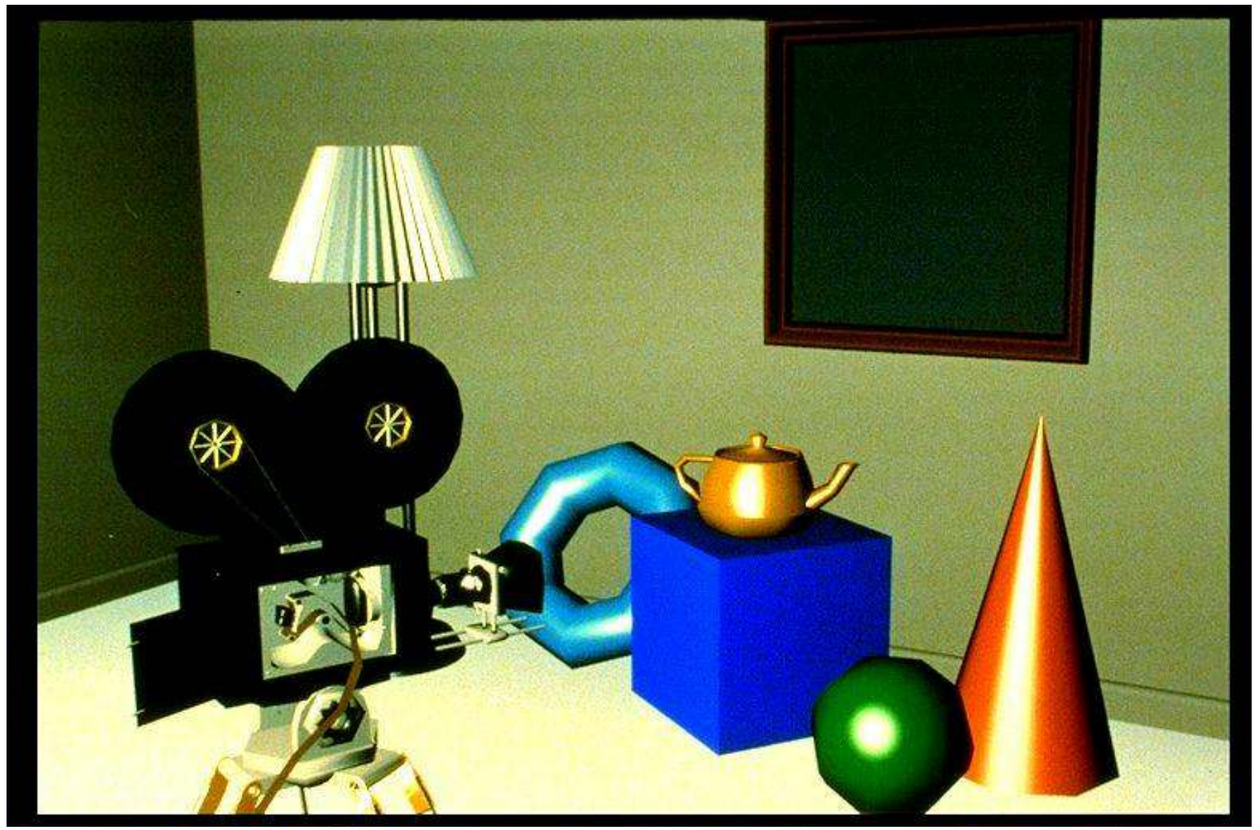
G. Zachmann
University of Bremen, Germany
cgvr.cs.uni-bremen.de

Warum ist Lighting & Shading so wichtig?



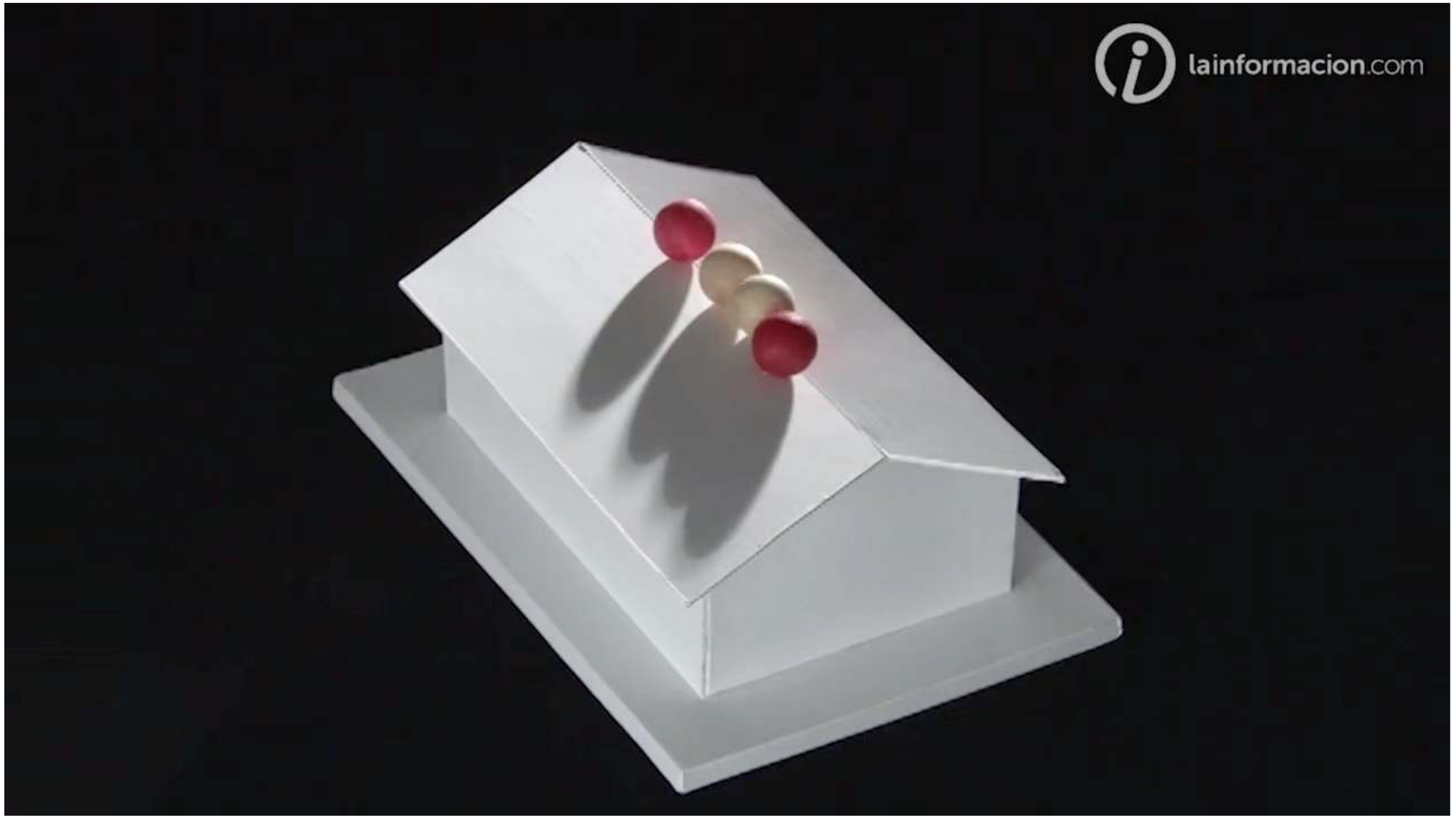
Warum ist Lighting & Shading so wichtig?

- Die “Shutterbug"-Szene: ganz ohne Shading, bis zum "maximalen" Shading



Pixar's "Shutterbug"

Das hilft natürlich nicht immer ...

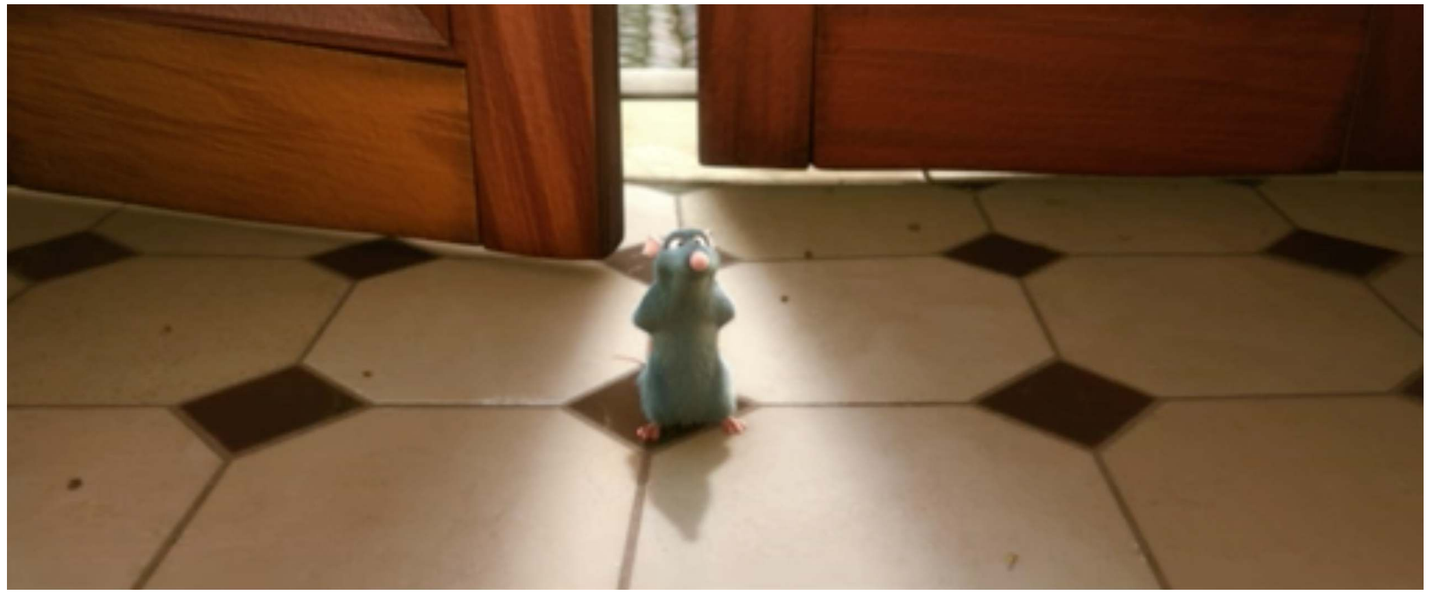


Impossible Rooftops, by Kokichi Sugihara

The Importance of Lighting from an Artistic Perspective

Creates a mood

Sets time of day



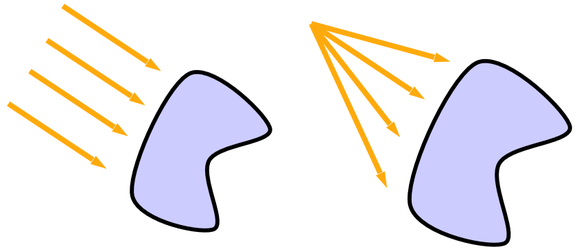
Guides the audience's eye

Makes the character stand out

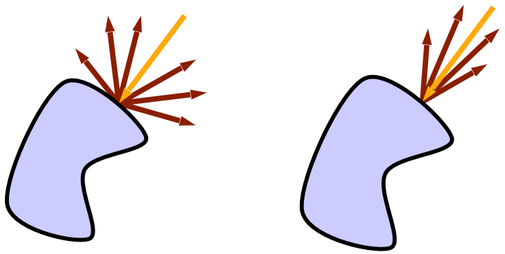
Beleuchtungsmodelle *(Lighting Models)*

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

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



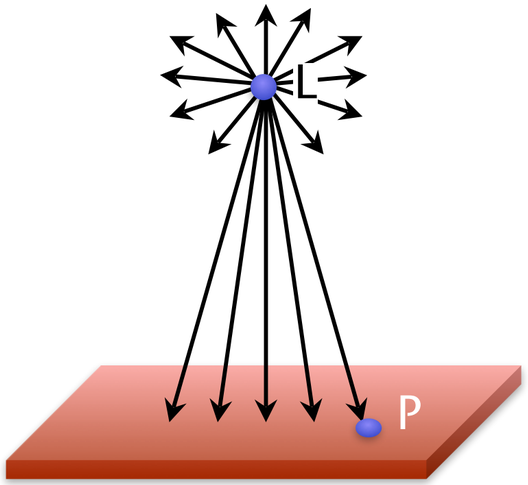
2. Objektoberfläche (Geometrie, Reflexionseigenschaften)



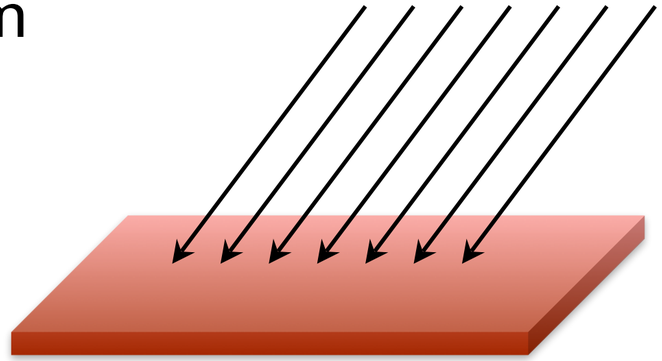
- Für Echtzeitanwendungen verwendet man sehr einfache Modelle

Einfache Lichtquellen-Modelle

- Punktlichtquelle (*point light*): strahlt in alle Richtungen gleichmäßig ab; wird beschrieben durch
  1. $I_0(\lambda) =$ abgestrahltes Spektrum (Intensität abhängig von λ)
  2. Position
  3. Intensität, die in Punkt P ankommt: $I(P) = \frac{1}{(L-P)^2} I_0$

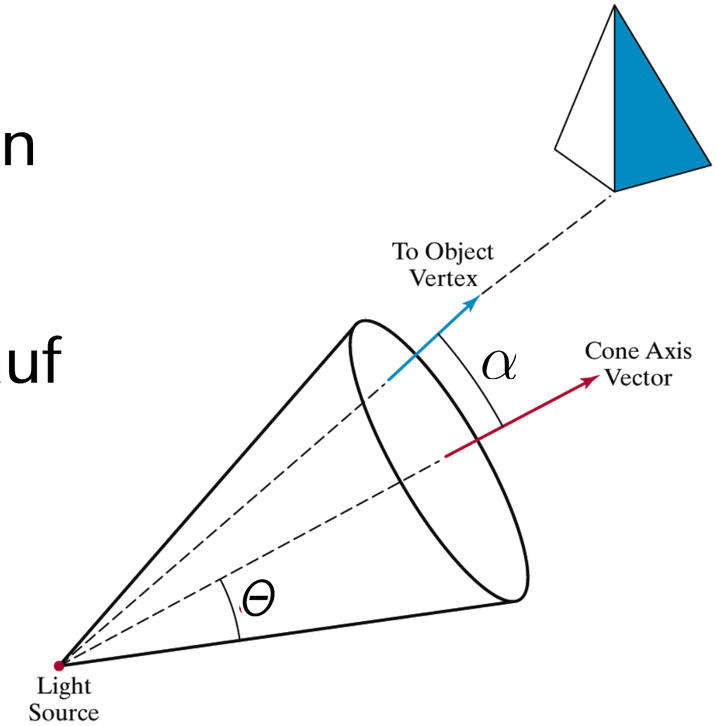


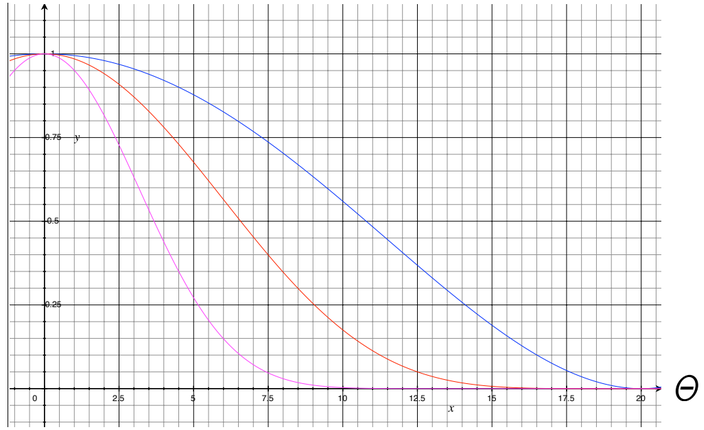
- Richtungslichtquelle *(directional light)*: jeder Punkt im Raum wird aus derselben Richtung bestrahlt
  - Charakterisiert durch Richtung & $I(\lambda)$
  - Beispiel: Sonne

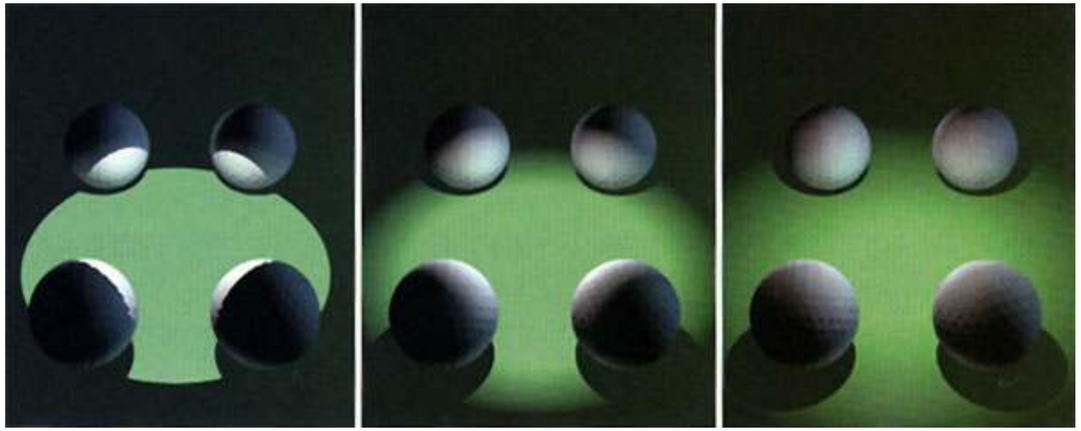


- Strahler (*spot light*): Lichtausbreitung wird auf einen bestimmten Raumwinkel (Lichtkegel) beschränkt
- Gesucht: Funktion, die $\cos(\alpha) \in [1, \cos(\theta)]$ abbildet auf [1,0] mit "schönem" Fall-off
- Diese tut's:

$$I = \begin{cases} 0 & \text{falls } \cos\alpha < \cos\theta \\ \frac{\cos^n \alpha - \cos\theta}{1 - \cos\theta} I_0 & \text{sonst} \end{cases}$$

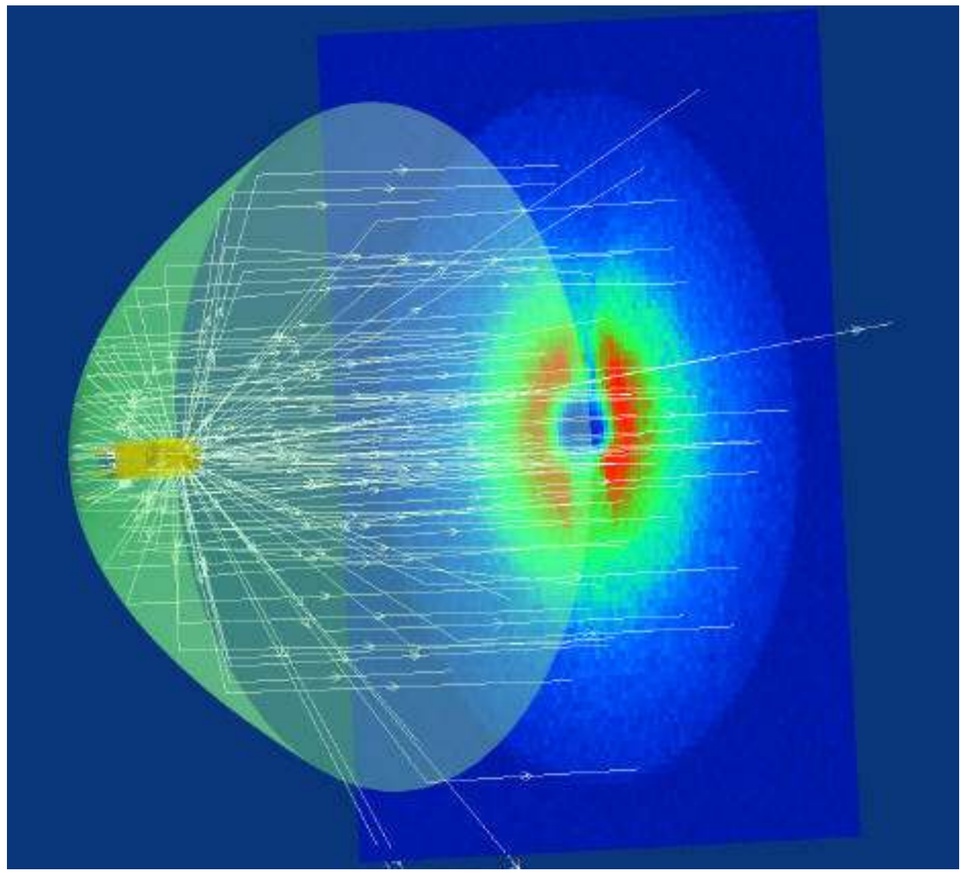






- Goniometrische Lichtquelle:
  Abstrahlcharakteristik wird per Tabelle beschrieben. Zur Ermittlung von $I_0(\lambda)$ muß evtl. zwischen Tabelleneinträgen interpoliert werden

- Area light source:

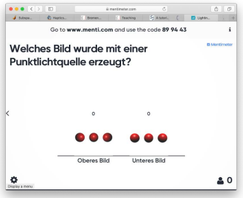


Unterschiedlicher Effekt zwischen *point light* und *directional light*

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

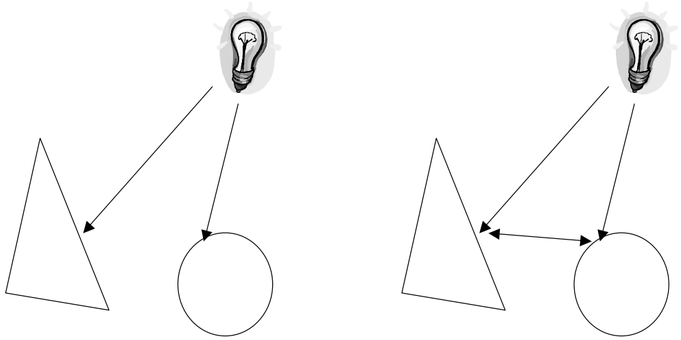
Point light source

Directional light source



Lokale Beleuchtungsmodelle

- Vereinfachung: berücksichtige bei der Berechnung der Beleuchtung eines Punktes keine sekundären Effekte (kein Strahlungsaustausch zwischen Objekten), nur primären Austausch



von Lichtquelle zu Objekt zu Auge ⟶ lokales Beleuchtungsmodell

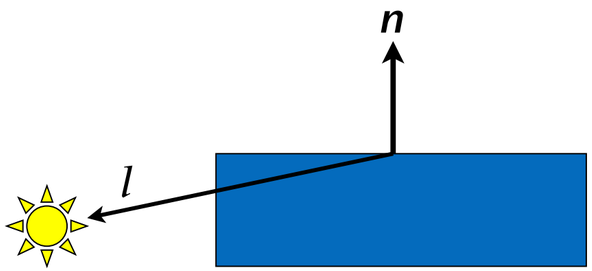
- Superpositionsprinzip: betrachte Licht als Teilchen ⟶

$$I = \sum_j I_j$$

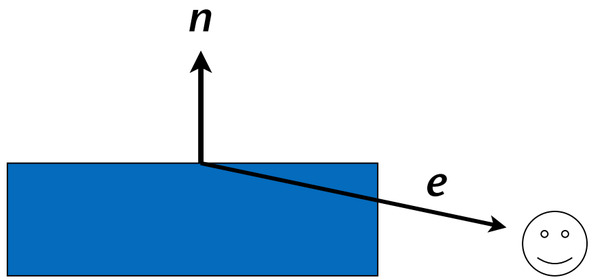
- Vereinfachung der Notation: wir lassen im Folgenden λ überall weg, und merken uns, daß alle photometrischen Größen eigtl. von λ abhängen!

Eine Konvention bzgl. negativer Skalarprodukte

- Falls $\mathbf{n} \cdot \boldsymbol{l} < 0$, 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

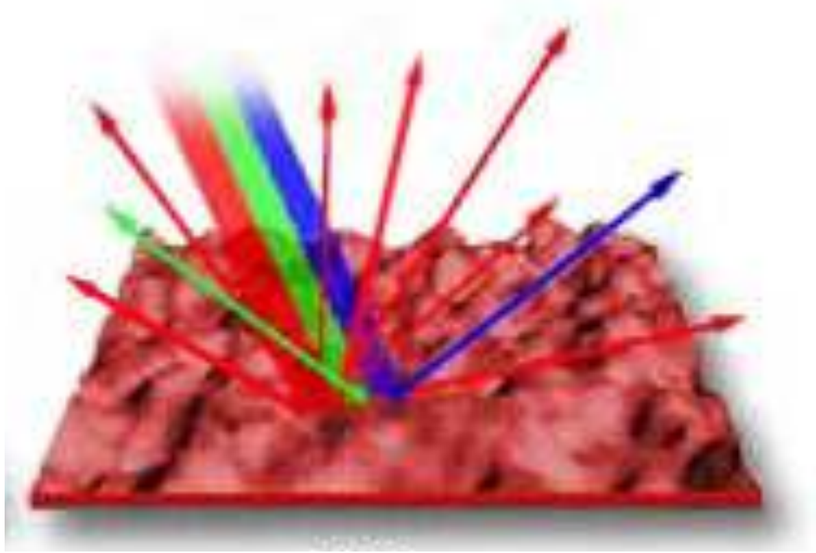


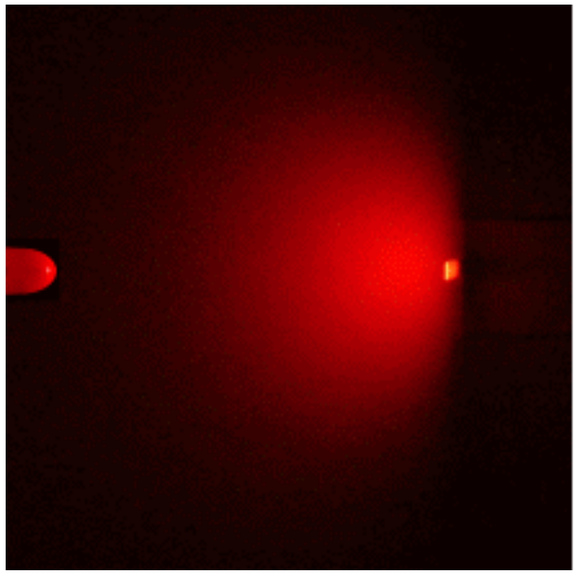
- Wir definieren im Folgenden (der Einfachheit halber) prinzipiell:

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

Diffuse Reflexion

- Mikro-Facetten-Modell: Oberfläche besteht aus sehr vielen, winzigen, planaren Facetten; alle Normalen kommen gleich häufig vor
- Licht wird von der Objektoberfläche gleichmäßig in alle Richtungen reflektiert
- Folge: Helligkeit ist unabhängig vom Viewpoint!
- Beispiele: Stück Papier, Tafel, unbearbeitetes Holz
- Diffuse/Matte Objekte werden auch als Lambert'sche Objekte bezeichnet





Lambert'sche Oberflächen

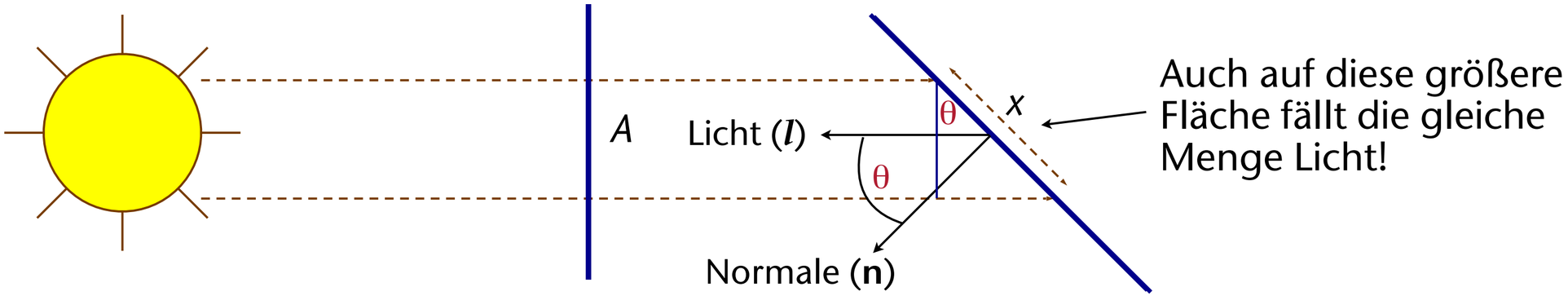
- Das Lambert'sche Kosinus-Gesetz:

$$I_{\text{diff}} = I_0 \cos\theta = I_0 \cdot \mathbf{n} \cdot \boldsymbol{l}$$

Hier: **n** und **l** sind Einheitsvektoren

- "Beweis":

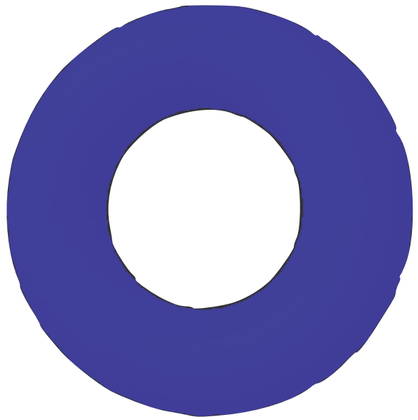
$$I = I_0 \qquad I = I_0 \frac{A}{x} = I_0 \cos\theta$$



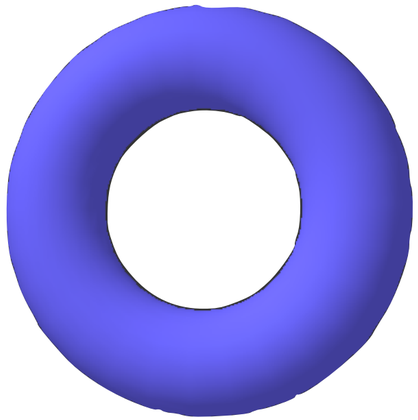
Ambientes Licht

- 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)
- Hack: füge für alle Objekte einen ambienten Beleuchtungsterm ein:

$$I = I_{\text{amb}} + I_0 \cdot \mathbf{n} \cdot l$$



Nur ambiente Beleuchtung

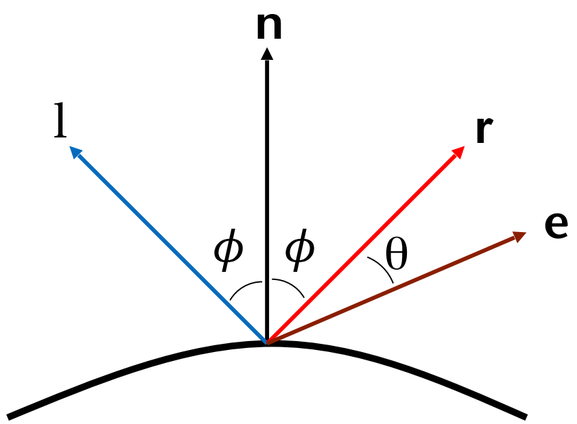


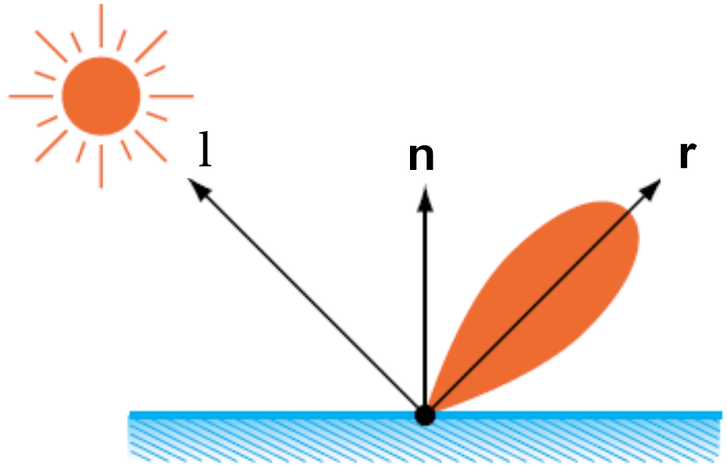
Diffuse + ambiente Beleuchtung

Spiegelnde (spekulare, glossy) Reflexion

- Stellt Highlights auf glänzenden Oberflächen dar
- Oberflächenreflexion ist abhängig von
  - Richtung der Lichtquelle: $\mathbf{l}$
  - Oberflächennormale: **n**
  - Richtung zum Betrachter: **e**
- Bei idealer spiegelnder Reflexion sieht man nur dann Licht von der Lichtquelle, wenn **r** = **e**
- Bei glänzenden Oberflächen sieht man auch "nahe" bei **r** ein Highlight; das erreicht man mit

$$I_{\text{spec}} = I_0 \left(\cos\theta\right)^p$$





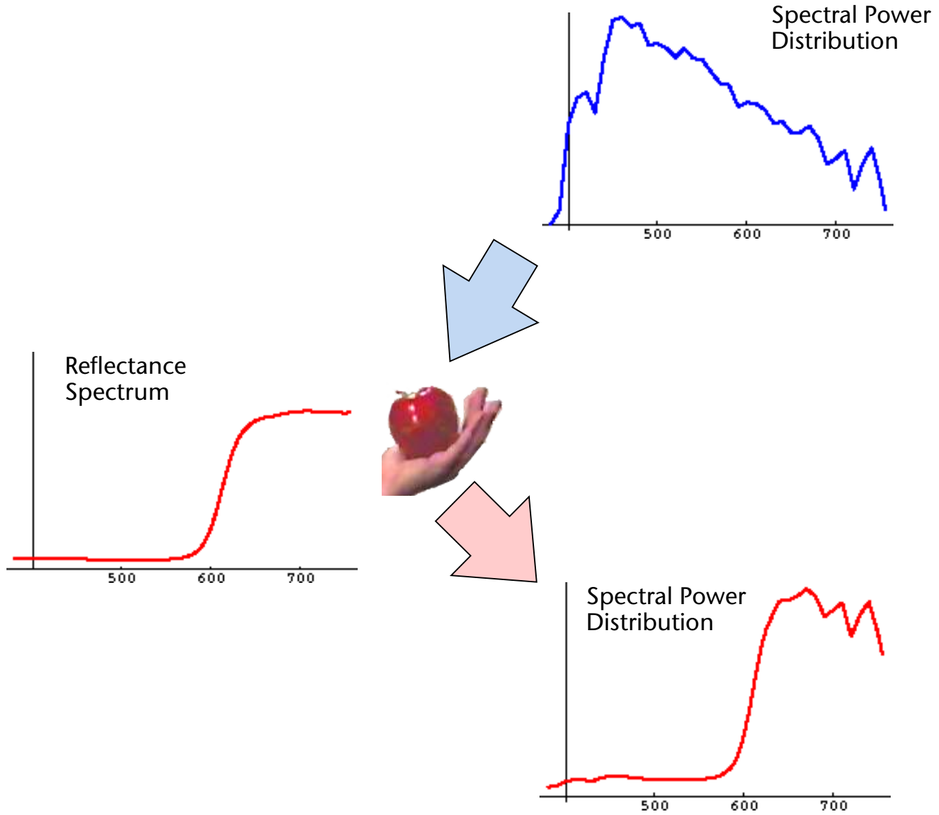
Highlights can Reveal Poorly Faked Fotos

Pro 7, Galileo: "Fake Fotos"

Reflexionskoeffizienten

- Bislang galt: $I_{\text{out}} = \rho \cdot I_{\text{in}} \ , \quad \rho \in [0,1]$

- Fehlt noch: Beschreibung des Reflexionsverhaltens des *Materials* der Objekte (z.B. rotes Objekt reflektiert mehr roten Anteil des Spektrums)



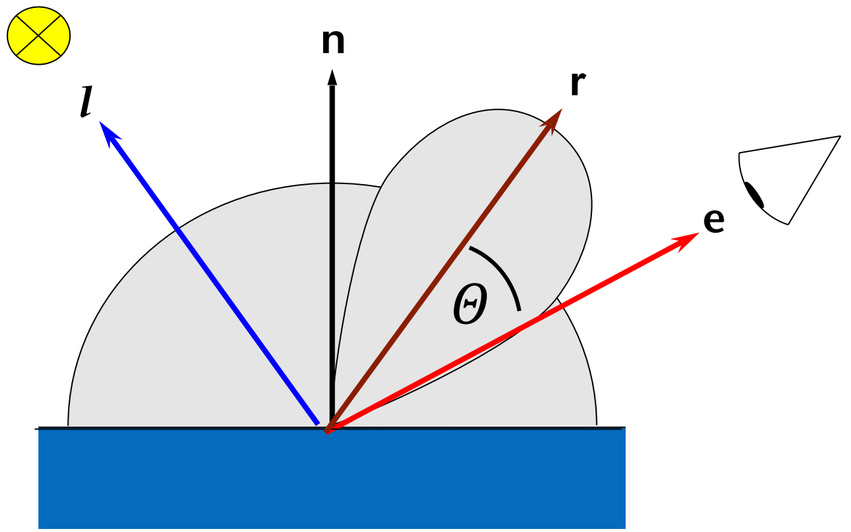
- Mit Reflexionskoeffizient *k*:

$$I_{\text{out}}(\lambda) = k(\lambda) \cdot \rho \cdot I_{\text{in}}(\lambda) \ , \quad \rho \in [0,1]$$

Das Phong-Beleuchtungsmodell

- Zusammensetzung:

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

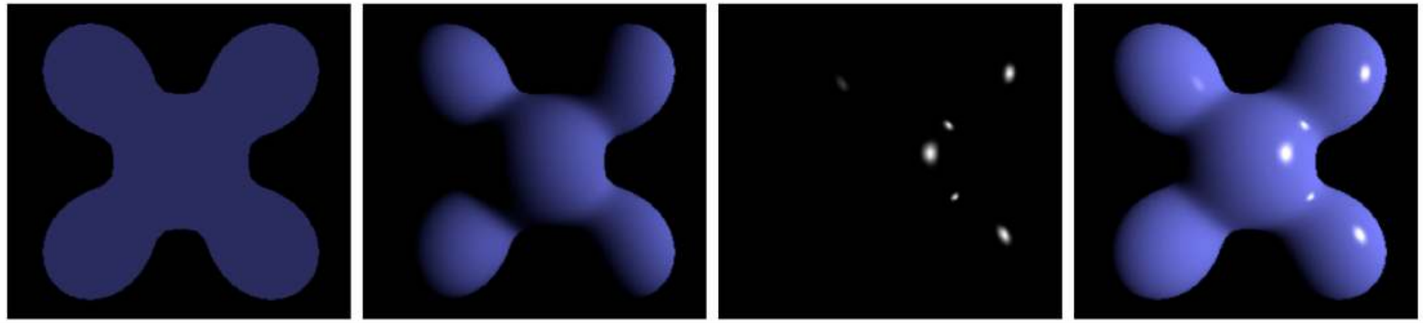


- Aufgrund des Superpositionsprinzips erhält man für *n* Lichtquellen:

$$I_{\text{out}} = k_d \cdot I_a + \sum_{j=1}^{n} (k_d \cos \Phi_j + k_s \cos^p \Theta_j) \cdot I_j$$

$k_d = k_d(\lambda)$ = diffuser Reflexionskoeffizient (diffuse Materialfarbe)
$k_s = k_s(\lambda)$ = spekularer Reflexionskoeffizient (spiegelnde Materialfarbe)
p = "Glanzzahl" (*shininess*), hat keine Einheit (hat keine physikalische Interpretation)

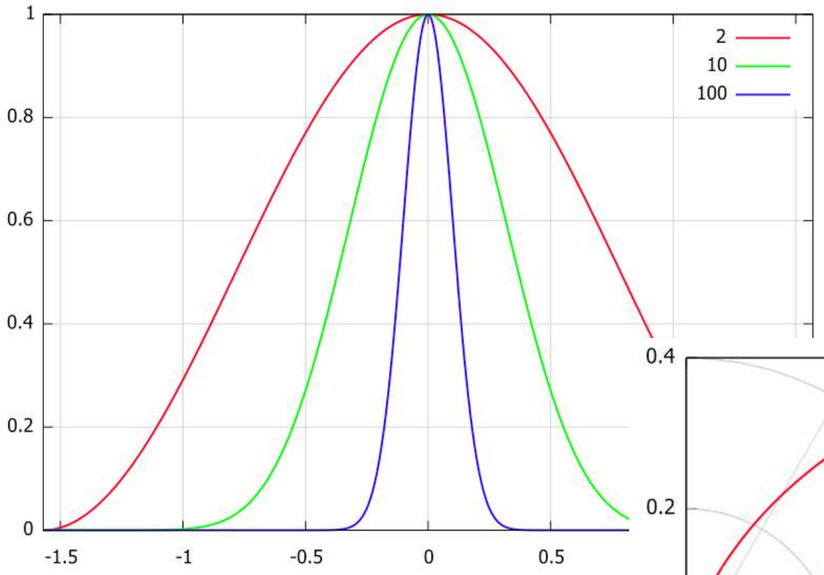
ambient + diffus + spekular = Phong



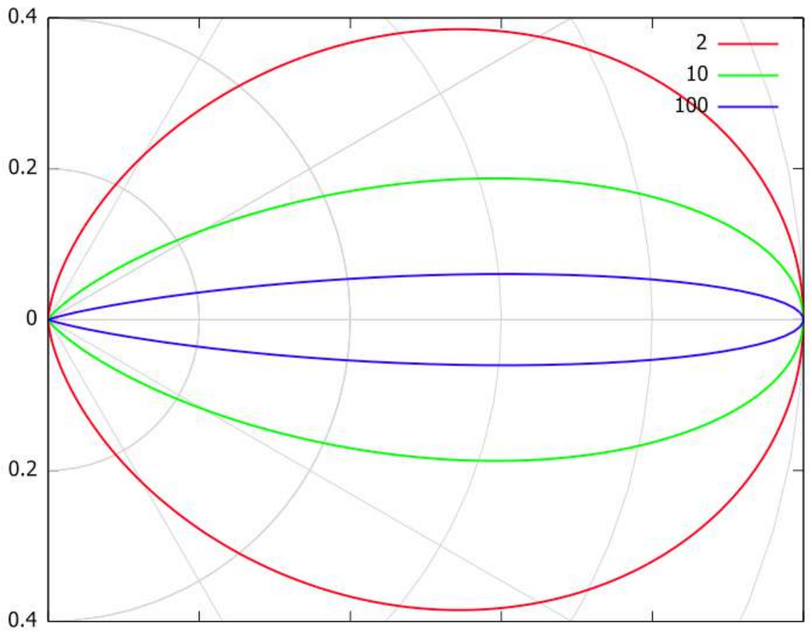
Phong	$\rho_{ambient}$	$\rho_{diffuse}$	$\rho_{specular}$	ρ_{total}
$\phi_i = 60^\circ$				
$\phi_i = 25^\circ$				
$\phi_i = 0^\circ$				

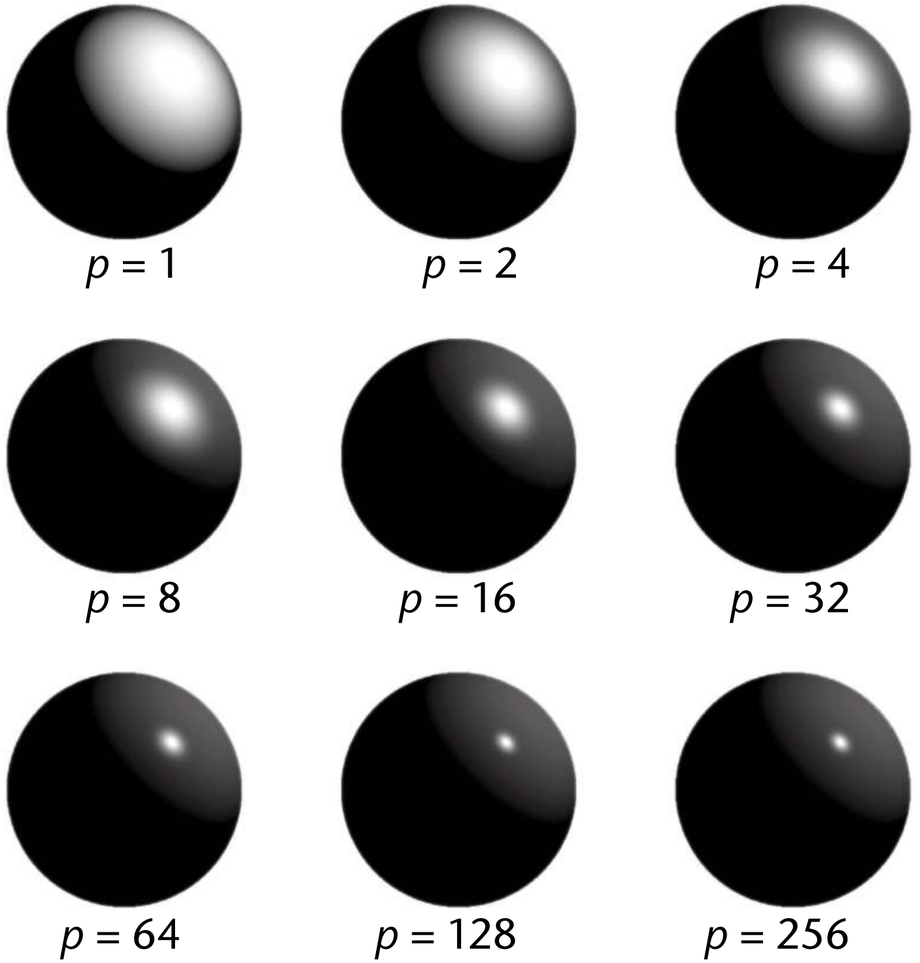
Effekt der Parameter im Phong-Modell

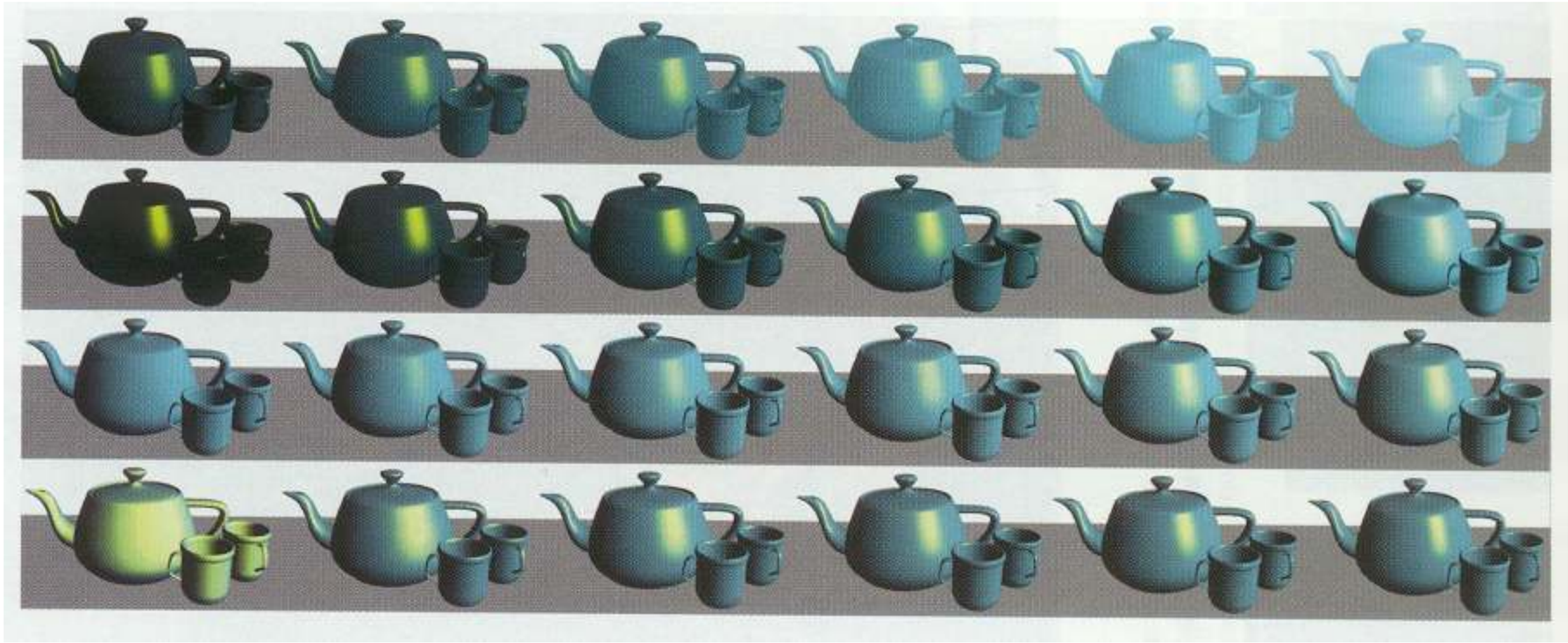
- Die Shininess p steuert die "Schärfe" des Highlights:

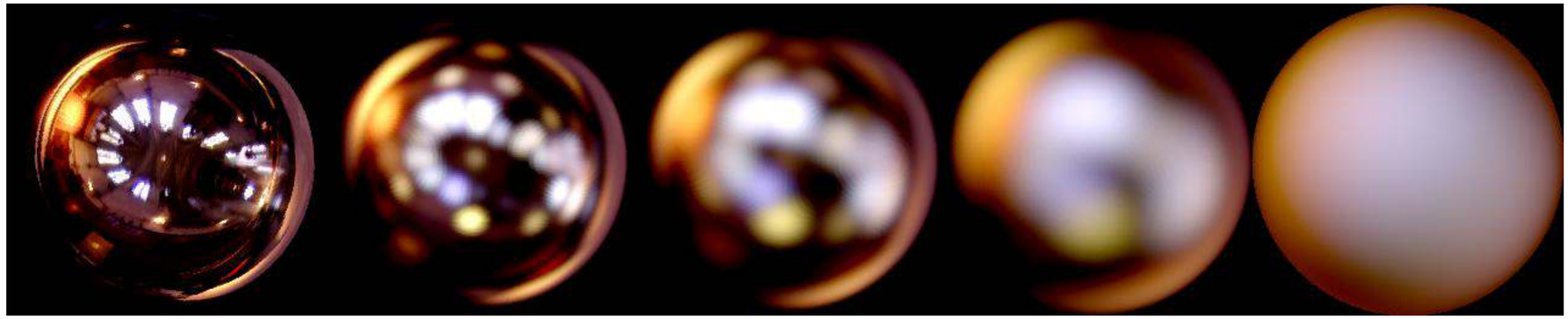


$\cos^p \theta$









"Oberflächenkörnung" (p)

Spectral Lighting

- Erinnerung: alle photometrischen Größen sind eigtl. Funktionen in λ! beschreiben also ein Spektrum ...
- In der Praxis (z.B. OpenGL) führt man alle Berechnungen jeweils für die 3 Primärvalenzen (z.B. *r, g, b*) durch
  - Das Produkt $k_d \cdot I_{\text{in}}$ ist in Wahrheit ein merkwürdiges Produkt von Vektoren:

$$\begin{pmatrix} r_{\text{diff}} \\ g_{\text{diff}} \\ b_{\text{diff}} \end{pmatrix} \cdot \begin{pmatrix} I_r \\ I_g \\ I_b \end{pmatrix} = \begin{pmatrix} r_{\text{diff}} I_r \\ g_{\text{diff}} I_g \\ b_{\text{diff}} I_b \end{pmatrix}$$

- Praktisch, aber: dadurch erhält man nicht 100% korrekte Bilder!
- Denn: $\text{RGB}(k(\lambda) \cdot I(\lambda)) \neq \text{RGB}(k(\lambda)) \cdot \text{RGB}(I(\lambda))$

Spezifikation der Materialfarbe in OpenGL und VRML

FYI

- OpenGL:

Beachte: diffuse und spekulare Materialfarbe können verschieden sein!

```
float[] mat_ambient = {0.7f, 0.7f, 0.7f, 1.0f};
float[] mat_diffuse = {0.1f, 0.5f, 0.8f, 1.0f};
float[] mat_specular = {1.0f, 1.0f, 1.0f, 1.0f};
float low_shininess = 5.0f;
float[] mat_emission = {0.3f, 0.2f, 0.2f, 0.0f};
glMaterialfv( GL_FRONT, GL_AMBIENT, no_mat);
glMaterialfv( GL_FRONT, GL_DIFFUSE, mat_diffuse );
glMaterialfv( GL_FRONT, GL_SPECULAR, mat_specular );
glMaterialf( GL_FRONT, GL_SHININESS, low_shininess );
glMaterialfv( GL_FRONT, GL_EMISSION, mat_emission );
```

- VRML/X3D:

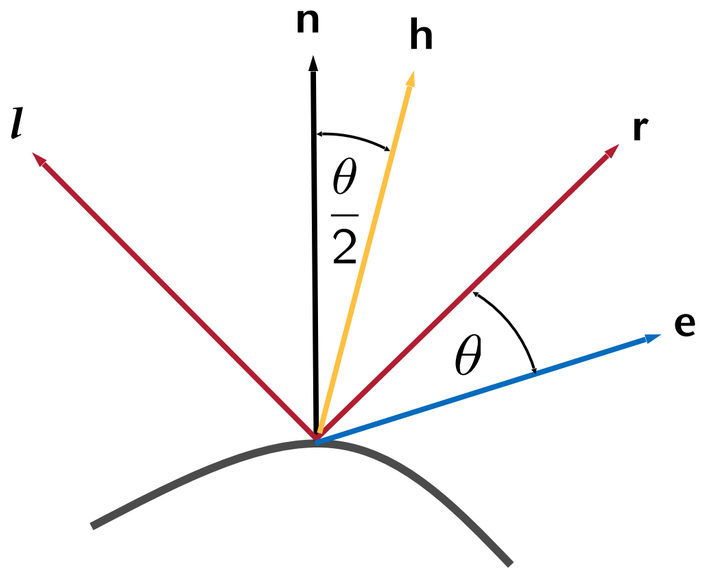
```
Material {
   SFFloat ambientIntensity 0.2
   SFColor diffuseColor     0.8 0.8 0.8
   SFColor specularColor    0 0 0
   SFFloat shininess        0.2
   SFColor emissiveColor    0 0 0
   SFFloat transparency     0
}
```

Bemerkungen

- Problem: Werte > 1 können entstehen!
  - Abhilfe: Clamping
  - Besser wäre: Erhalten von Farbton und Sättigung (⟶ *Tone Mapping*)

Das Blinn-Phong Modell

- Problem des Phong-Modells: man muß für jeden Vertex den Reflexionsvektor bestimmen
- Idee: verwende Winkelhalbierende **h** ("*half-vector*") und **n**, statt **r** und **e**:
  - Setze: $\mathbf{h} = \dfrac{\boldsymbol{l} + \mathbf{e}}{|\boldsymbol{l} + \mathbf{e}|}$
  - Berechne damit $I'_{\text{spec}} = k_s I_{\text{in}} \cos^q \frac{\theta}{2} = k_s I_{\text{in}} (\mathbf{h} \cdot \mathbf{n})^q$



- Frage: ist es dasselbe Modell? ⟶ 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)

Echte Materialien sind i.A. viel komplizierter

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:

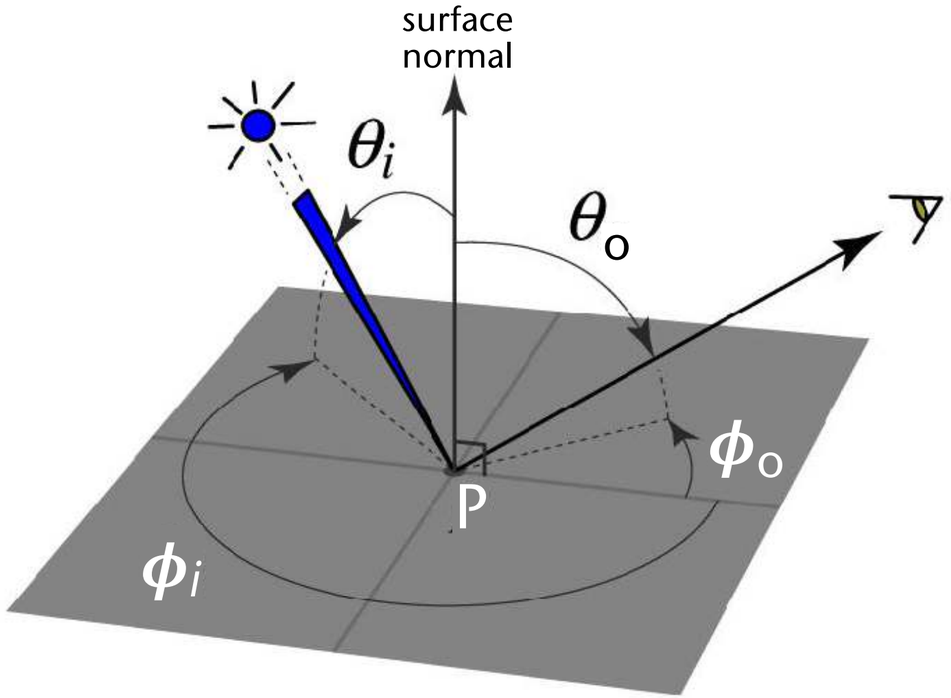
$$I_{\text{out}} = \underbrace{k_d(\mathbf{n}\cdot\mathbf{l})}_{\text{reflectance}} \cdot I_{\text{in}} \qquad I_{\text{out}} = \underbrace{k_s(\mathbf{h}\cdot\mathbf{n})^q}_{\text{reflectance}} I_{\text{in}}$$

BRDFs als Verallgemeinerung

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

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

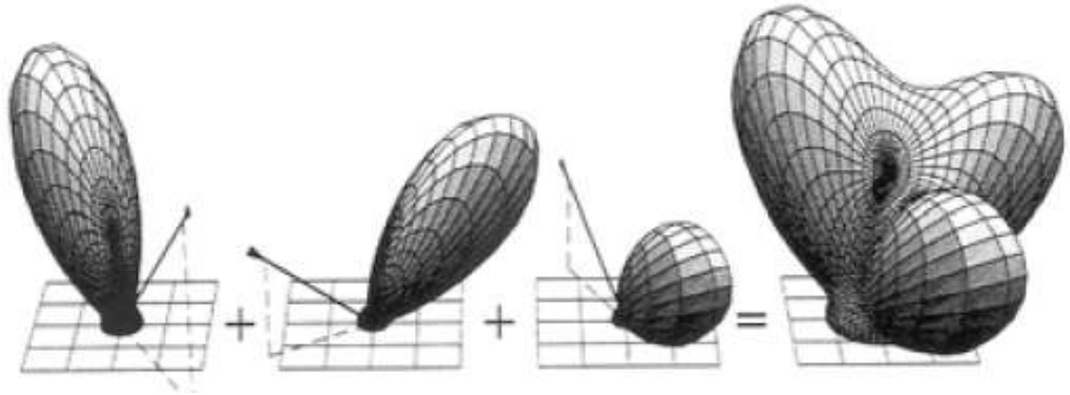
dabei sind (θ_i, ϕ_i) die Einfallswinkel,
und (θ_o, ϕ_o) die Ausfallswinkel



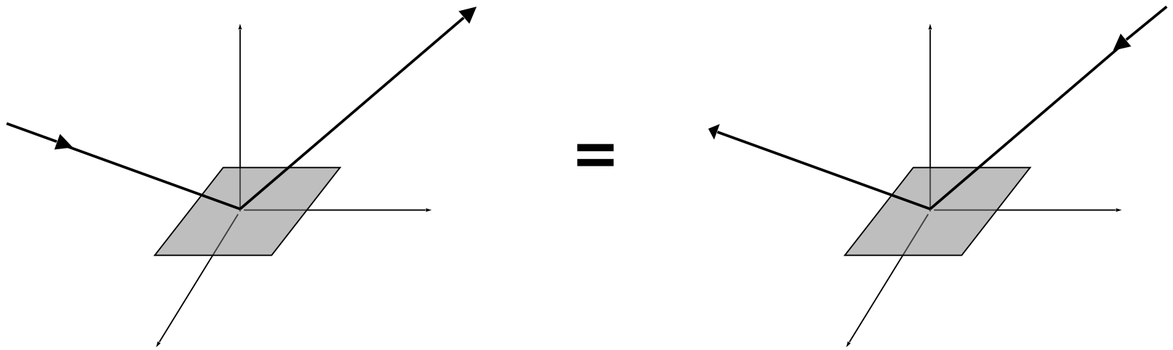
Generelle Eigenschaften von BRDFs

- Folgende Annahmen/Eigenschaften werden (meist) postuliert (je nach Material gelten mehr oder weniger)

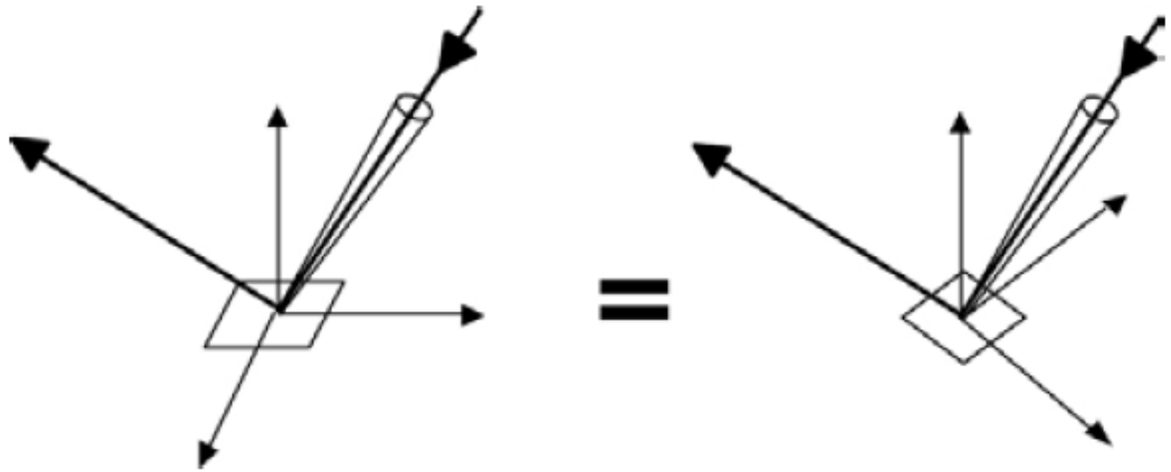
1. Linearität:



2. Reciprozität (reciprocity):

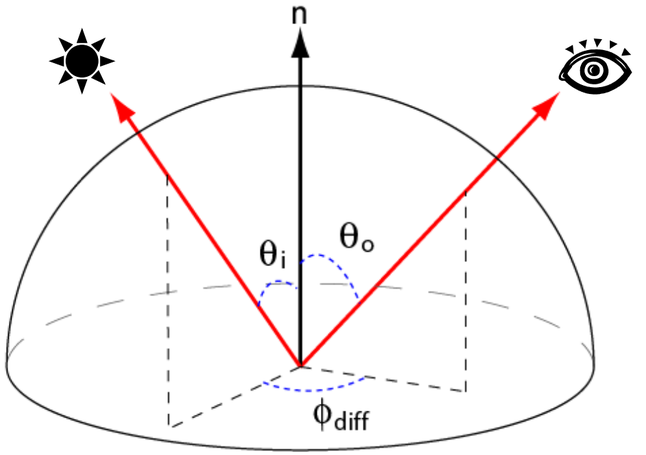


3. Isotropie (isotropy):
   BRDF ist invariant bzgl. Rotation des Materials um die Normale

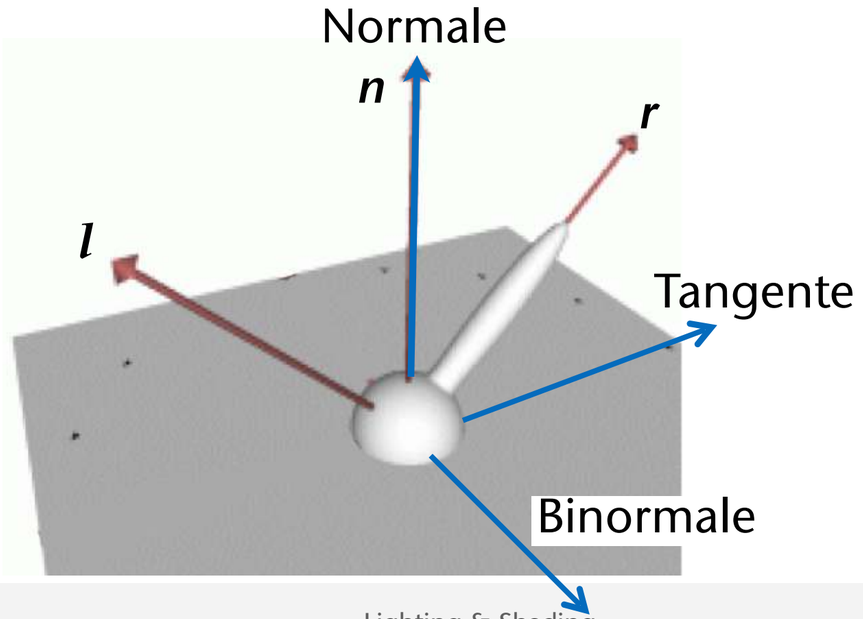


4. Energieerhaltung (conservation of energy):
   total quantity of outgoing light ≤ total quantity of incoming light
   (m.a.W.: Integral über die BRDF ≤ 1)
5. Die BRDF eines Materials hängt *nicht* von dem jeweiligen Punkt auf der Oberfläche ab!

- Konsequenz aus Reziprozität und Isotropie: die BRDF ist eigentlich nur eine 3D-Funktion



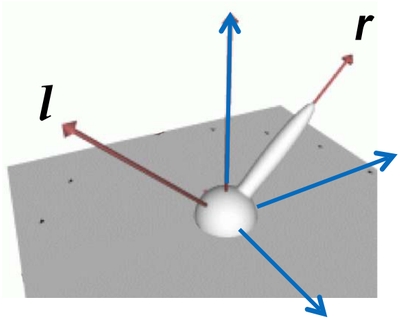
- Wähle im folgenden dieses spezielle Koordinatensystem zur Auswertung einer BRDF in einem Punkt auf der Oberfläche eines Objektes



Beispiel-BRDF: das Lafortune-Modell

- Der spekulare Anteil im Phong-Modell *in diesem* Koordinatensystem ist

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

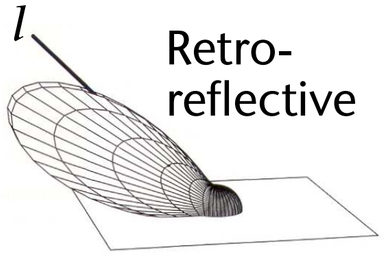


- Das Lafortune-Modell:
  - Ersetze die spezielle Matrix durch eine beliebige Matrix *C*
  - Erlaube beliebig viele "*Lobes*" (= Keulen, Lappen)
  - Zusammen:

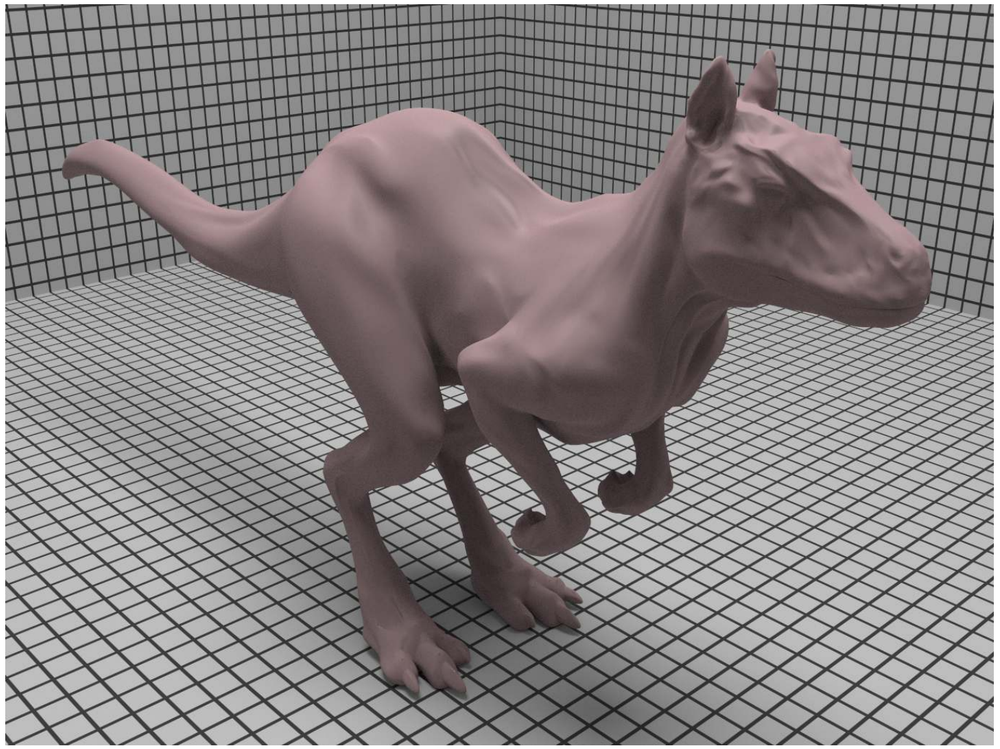
$$\rho(\boldsymbol{l},\mathbf{e}) = k_d(\mathbf{n}\cdot\boldsymbol{l}) + \sum_{i=1}^{n} k_{s,i}(\mathbf{e}\cdot C_i\cdot\boldsymbol{l})^{p_i}$$

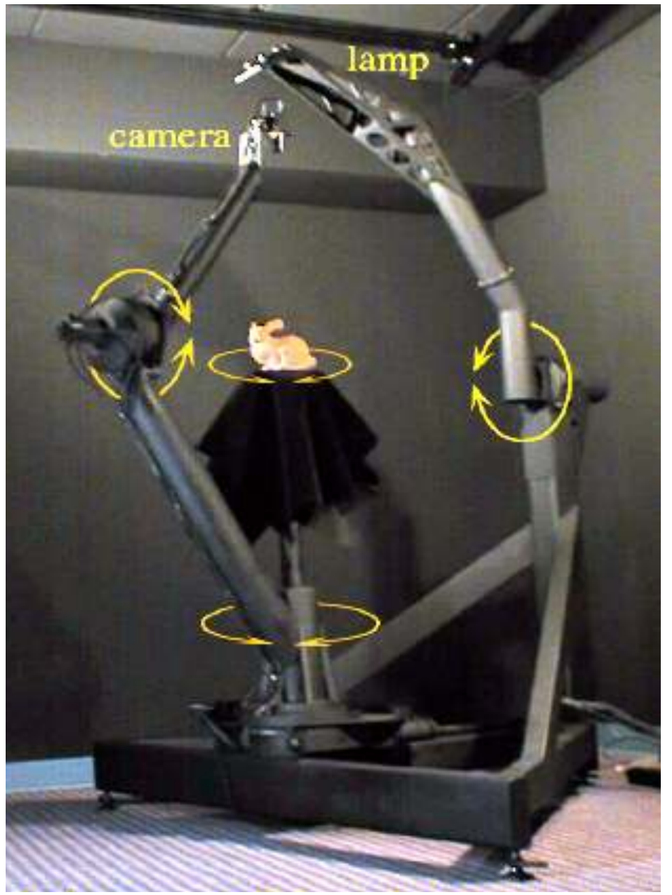
    wobei *n* = Anzahl *lobes*

- Frage: welche Matrix *C* braucht man hier?



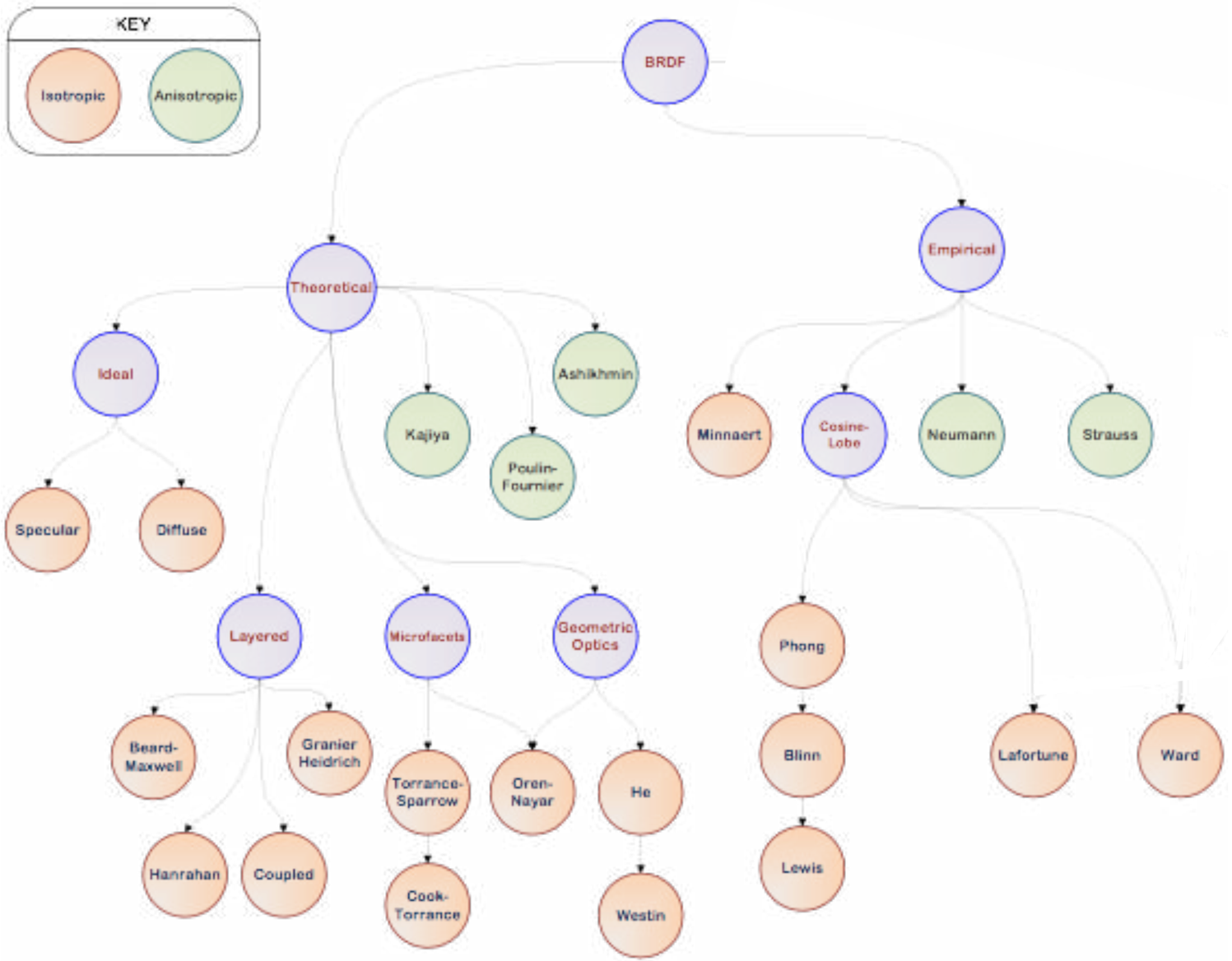
- In der Praxis misst man Materialien aus und versucht, das Lafortune-Modell darauf zu "fitten", mit Hilfe eines numerischen Optimierungsverfahrens (= 3 + (3+9+1)*n* Parameter!)





Gerendert mit Lafortune-Modell, das vermessenen Ton modelliert

Overview of Some of the Popular BRDFs

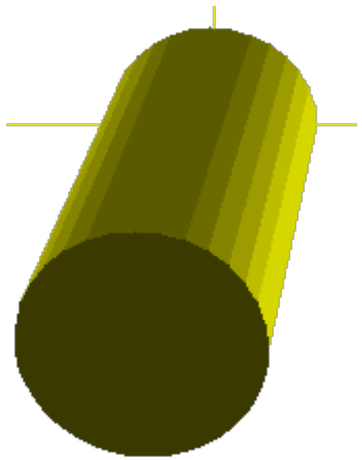


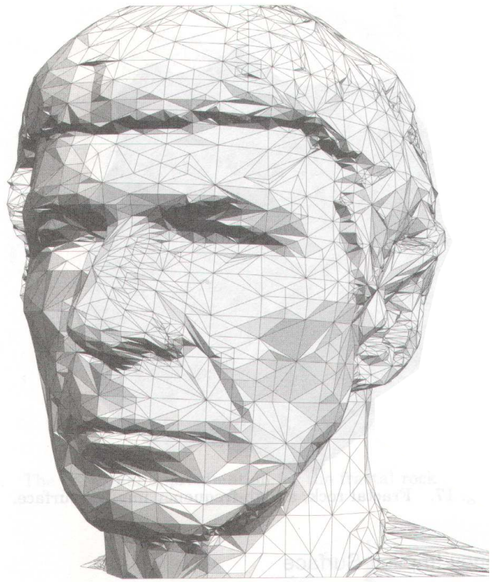
Shading-Algorithmen

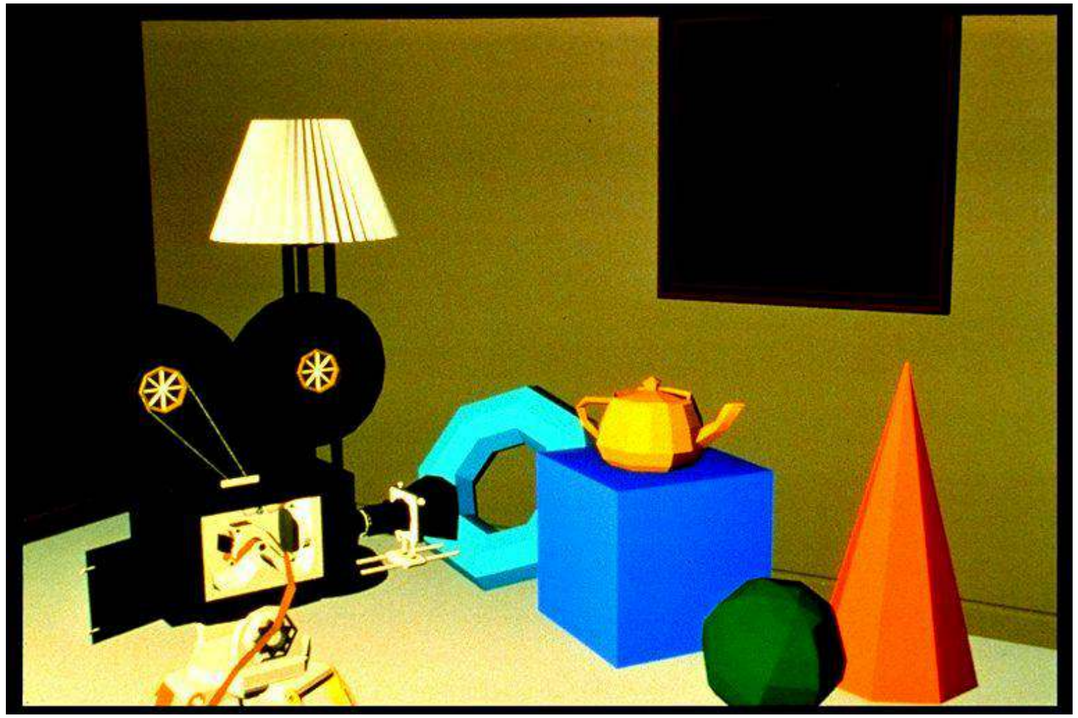
- Achtung: unterscheide zwischen Beleuchtungs**modell** (*lighting model)* und Schattierungs**algorithmus** (*shading algorithm*)!
  - Beleuchtungsmodell beschreibt Zusammenhang zwischen Lichtquellen und Oberflächen zur Berechnung der Intensität/Farbe in jedem Punkt
  - Schattierungsalgorithmus berechnet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

Flat Shading (Konstante Beleuchtung)

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



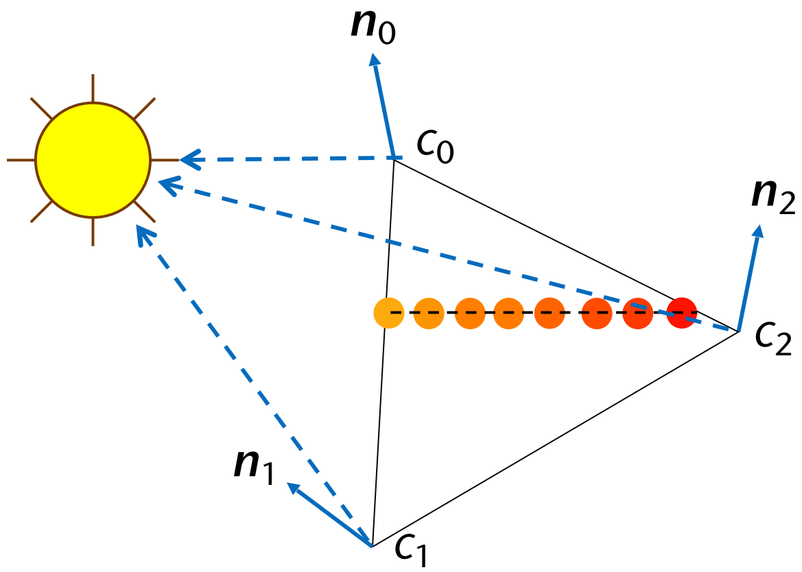




Pixar "Shutterbug"

Gouraud-Shading

- Werte das Beleuchtungsmodell (Phong, Lafortune, ...) an allen 3 Vertices des Dreiecks aus, interpoliere linear dazwischen während der Scanline-Konvertierung



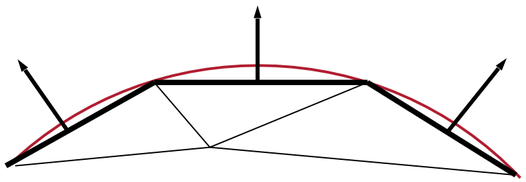
- Achtung: i.A. macht dies nur mit Vertex-Normalen Sinn

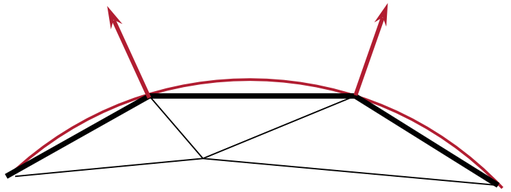
First Gouraud-shaded images, of Sylvie Gouraud

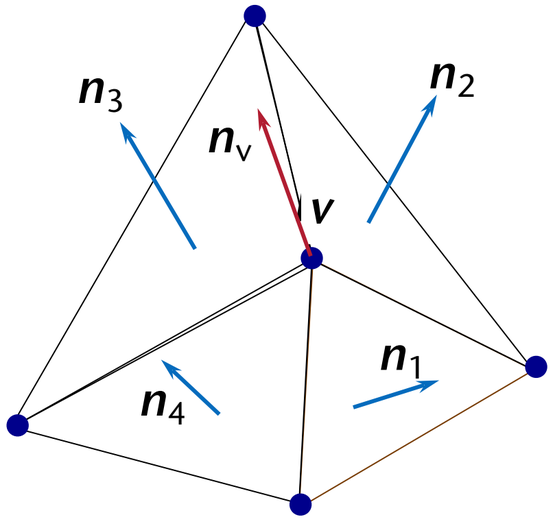
Berechnung der Normalen der Eckpunkte

- 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:

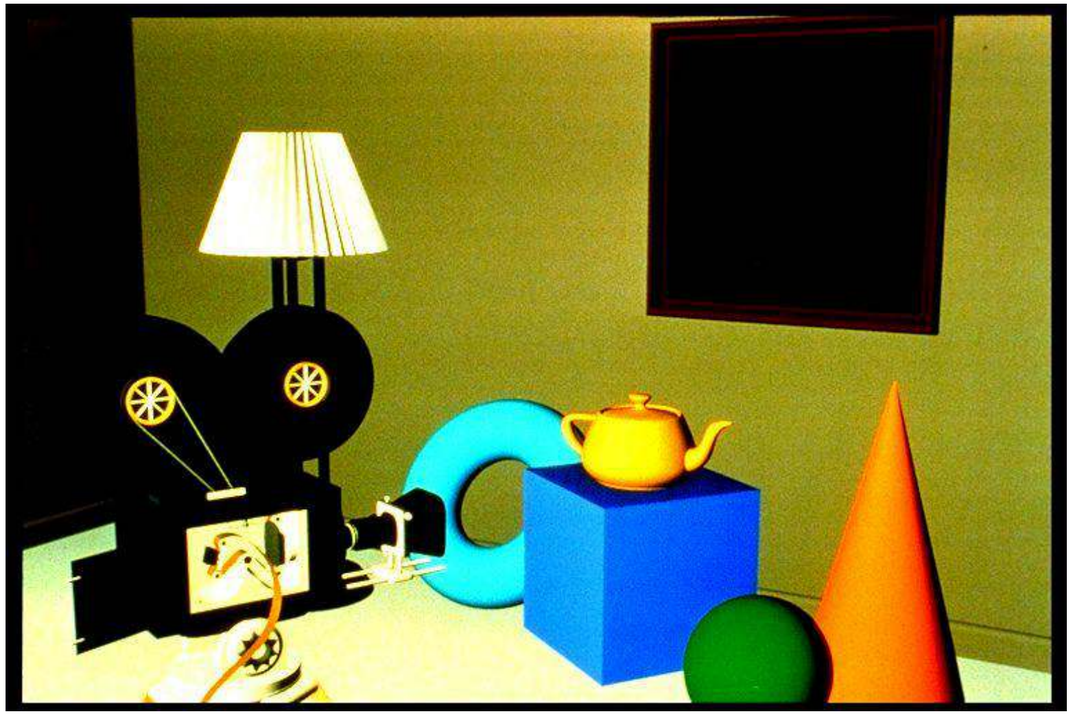
1. Zu Beginn berechne eine Normale für jedes Polygon
2. Bestimme für jeden Vertex, welche Polygone diesen enthalten (Inzidenz)
3. Bestimme den Mittelwert der Normalen dieser angrenzenden Polygone
   - Einfach aufsummieren, dann normieren



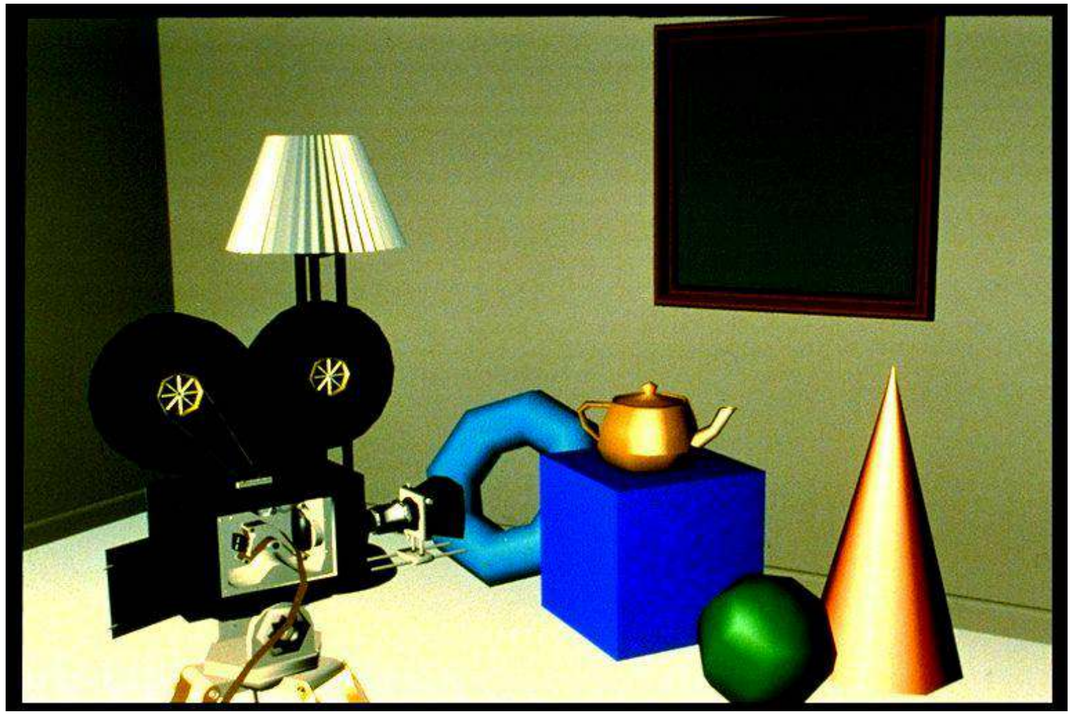




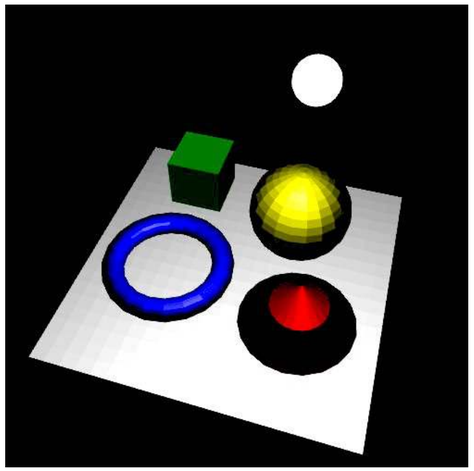
Vergleiche

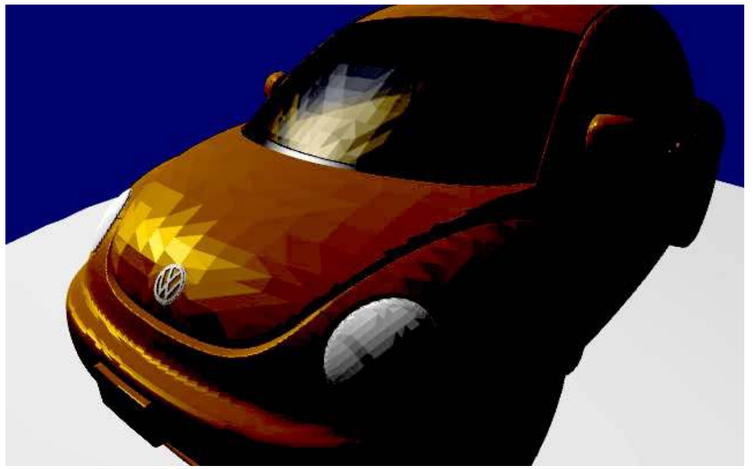


Gouraud-Shading mit
rein diffusem Beleuchtungsmodell

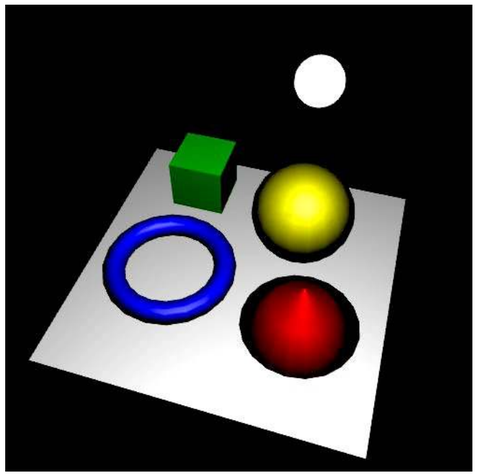


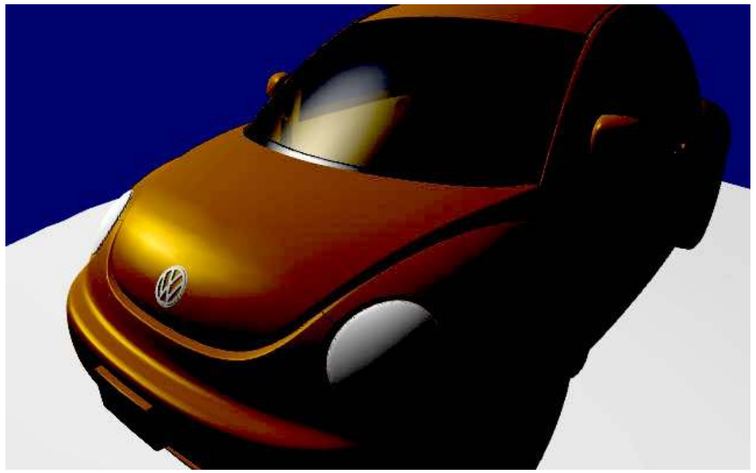
Gouraud-Shading mit Phong-Modell





Flat-Shading mit
Phong-Modell

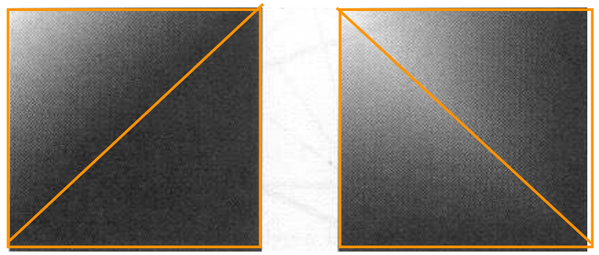




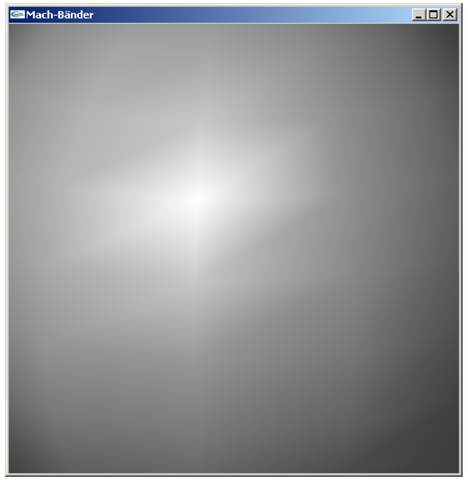
Gouraud-Shading mit
Phong-Modell

Einfluß der Triangulierung

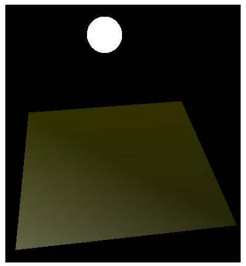
- Orientierung:



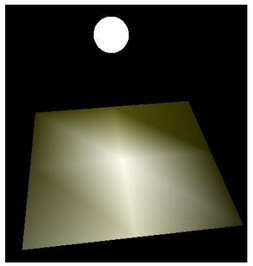
- Hier gilt: "size matters"



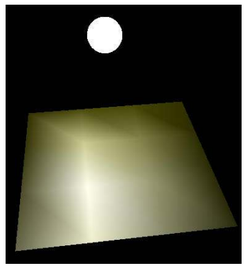
Vergrößerung



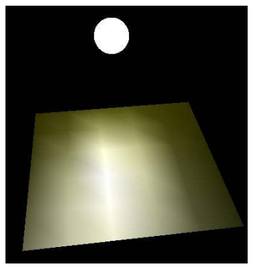
1 x 1 Flächen



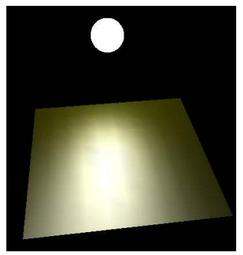
2 x 2 Flächen



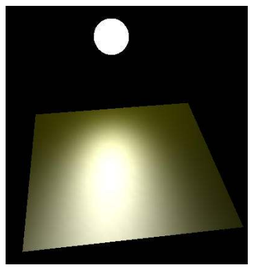
3 x 3 Flächen



5 x 5 Flächen



10 x 10 Flächen

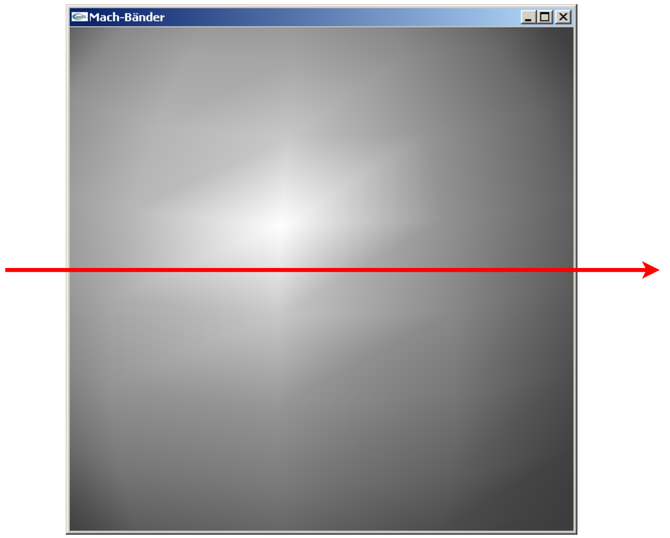


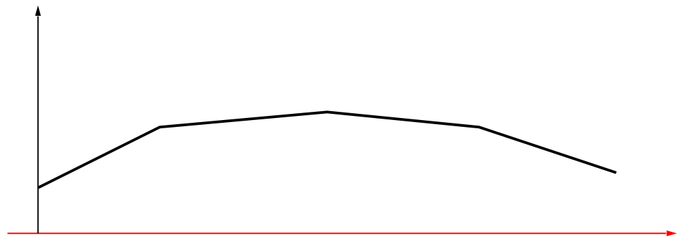
40 x 40 Flächen

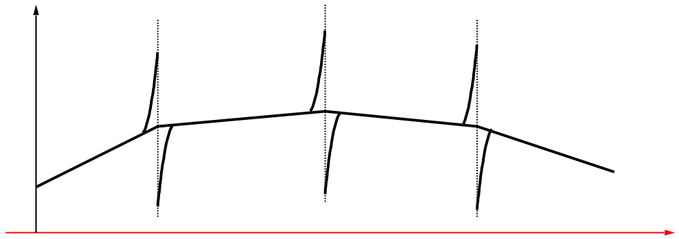
- Frage: Woher kommen die „Streifen“?

Mach-Bänder

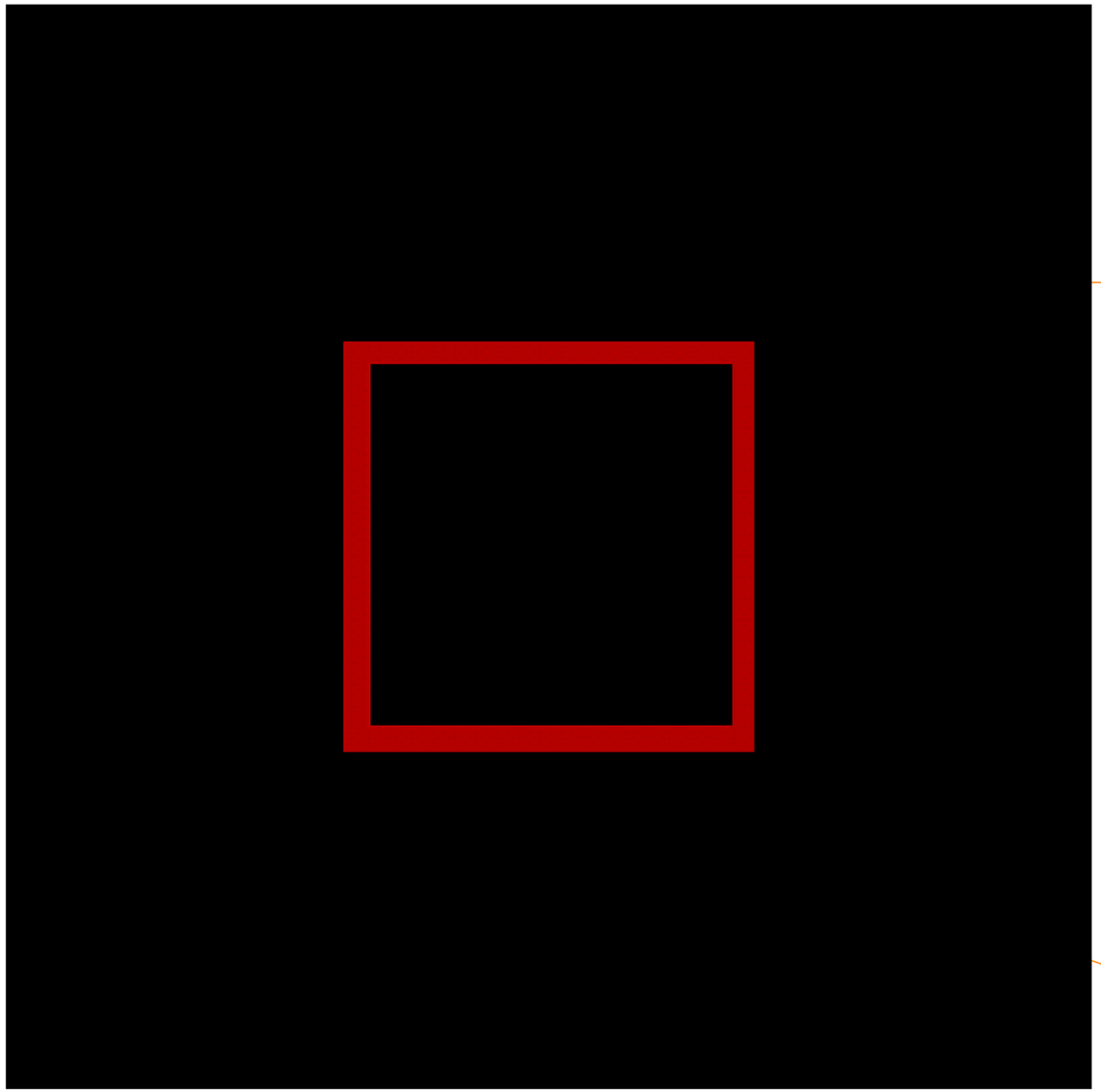
- Problem: die lineare Interpolation der "Lichtwerte"
  - Diese Interpolation ist C^0-stetig, aber *nicht* C^1-stetig!
- Das menschliche Auge hat einen eingebauten "Kantendetektor" (erkennt genau diese "Knicke")
  - Es gibt Neuronen, die die räumliche Ableitung bilden (jeweils für ein Retina-"Pixel")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







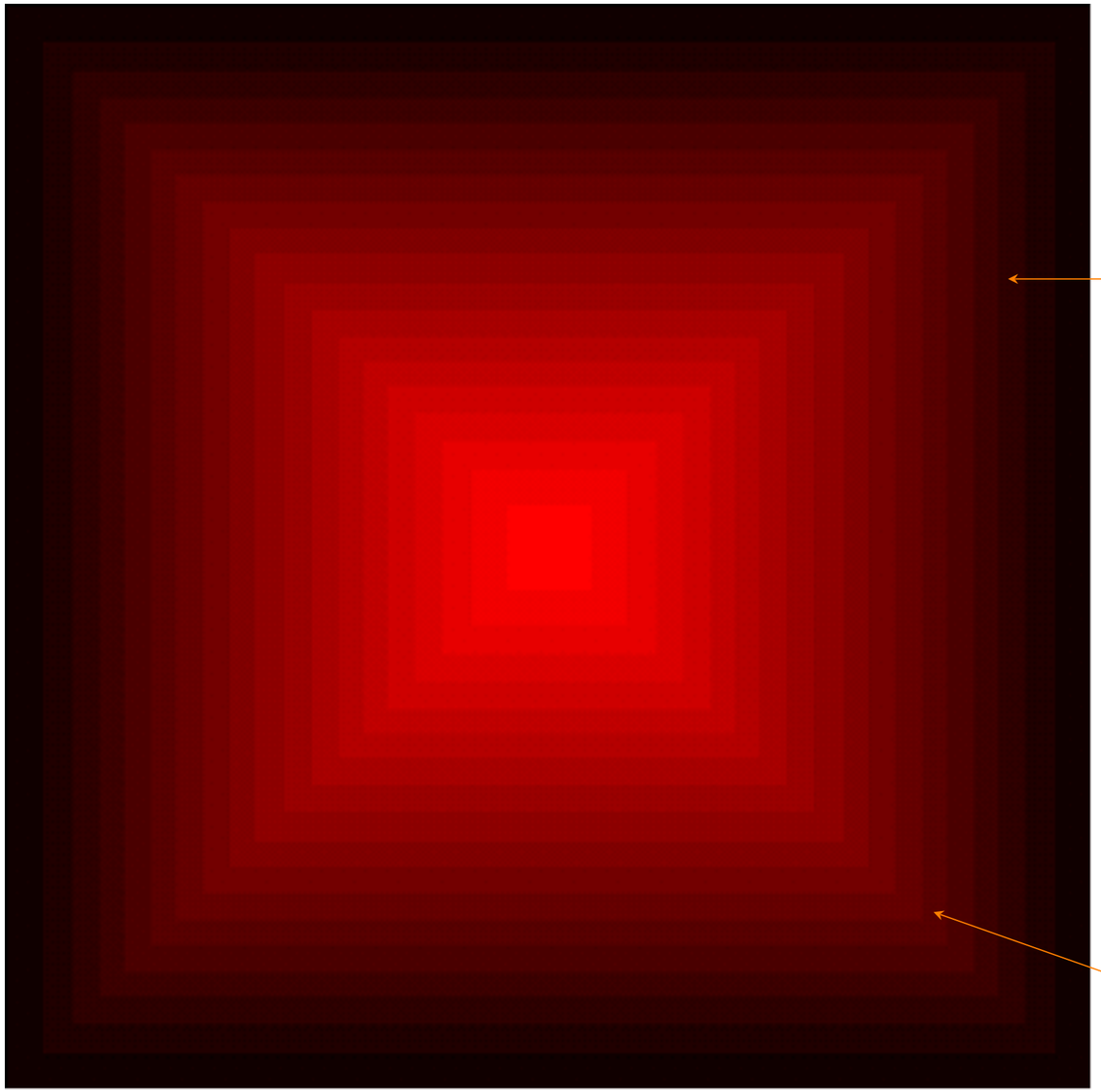
Extremes Beispiel



Die Intensität innerhalb jedes der quadratischen Rahmen ist konstant!
Ein Helligkeitsverlauf von innen nach außen ist eine Illusion.

Die hellere Eindruck bei 45° (und 135°) sind eine Illusion!

Extremes Beispiel



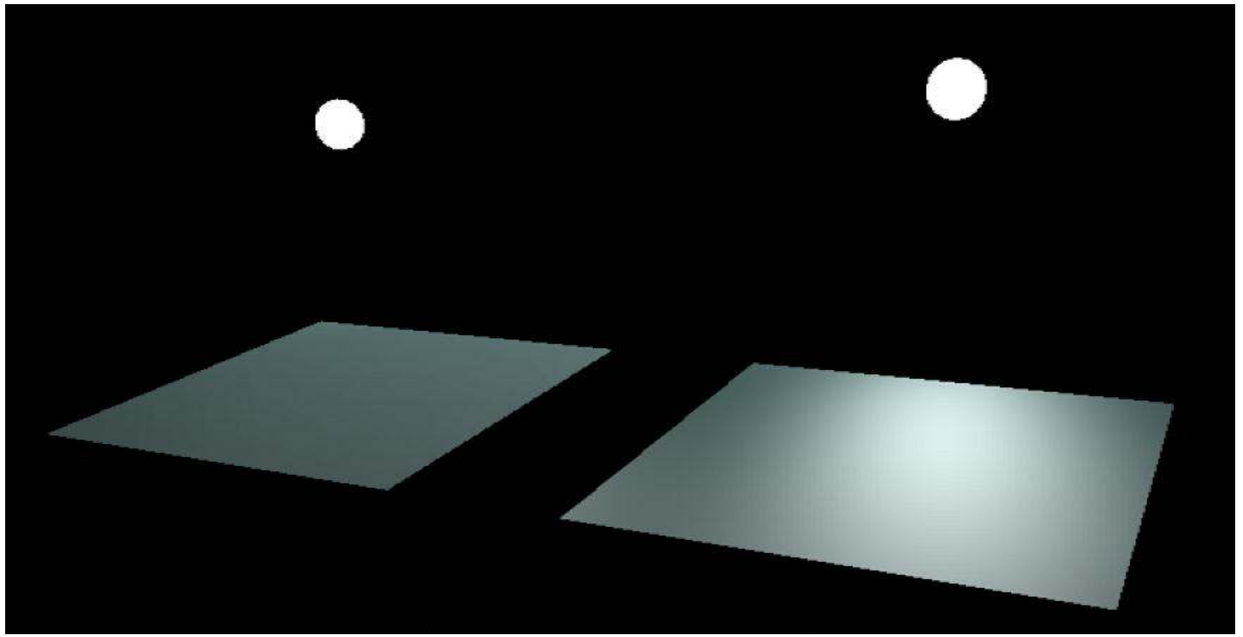
Die Intensität innerhalb jedes der quadratischen Rahmen ist konstant!
Ein Helligkeitsverlauf von innen nach außen ist eine Illusion.

Die hellere Eindruck bei 45° (und 135°) sind eine Illusion!

Real-World Example

Weiteres Problem beim Gouraud-Shading

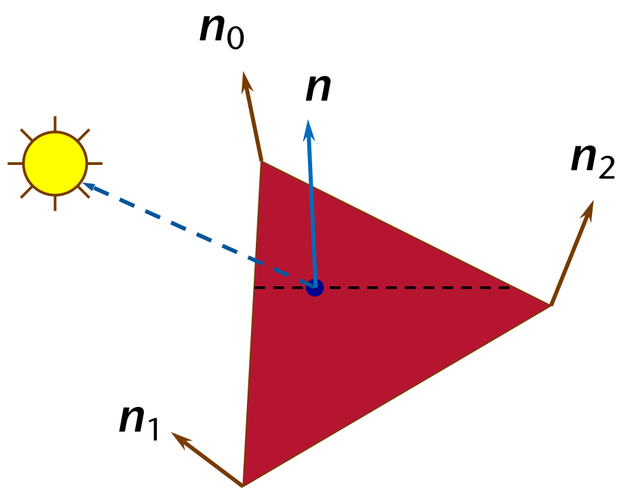
- Evtl. "verpasst" man Highlights im Inneren eines Polygons:

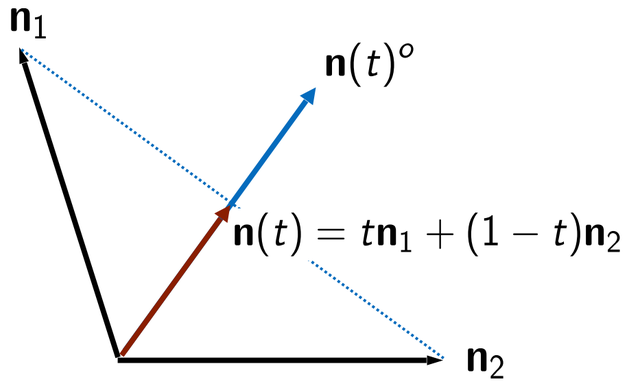


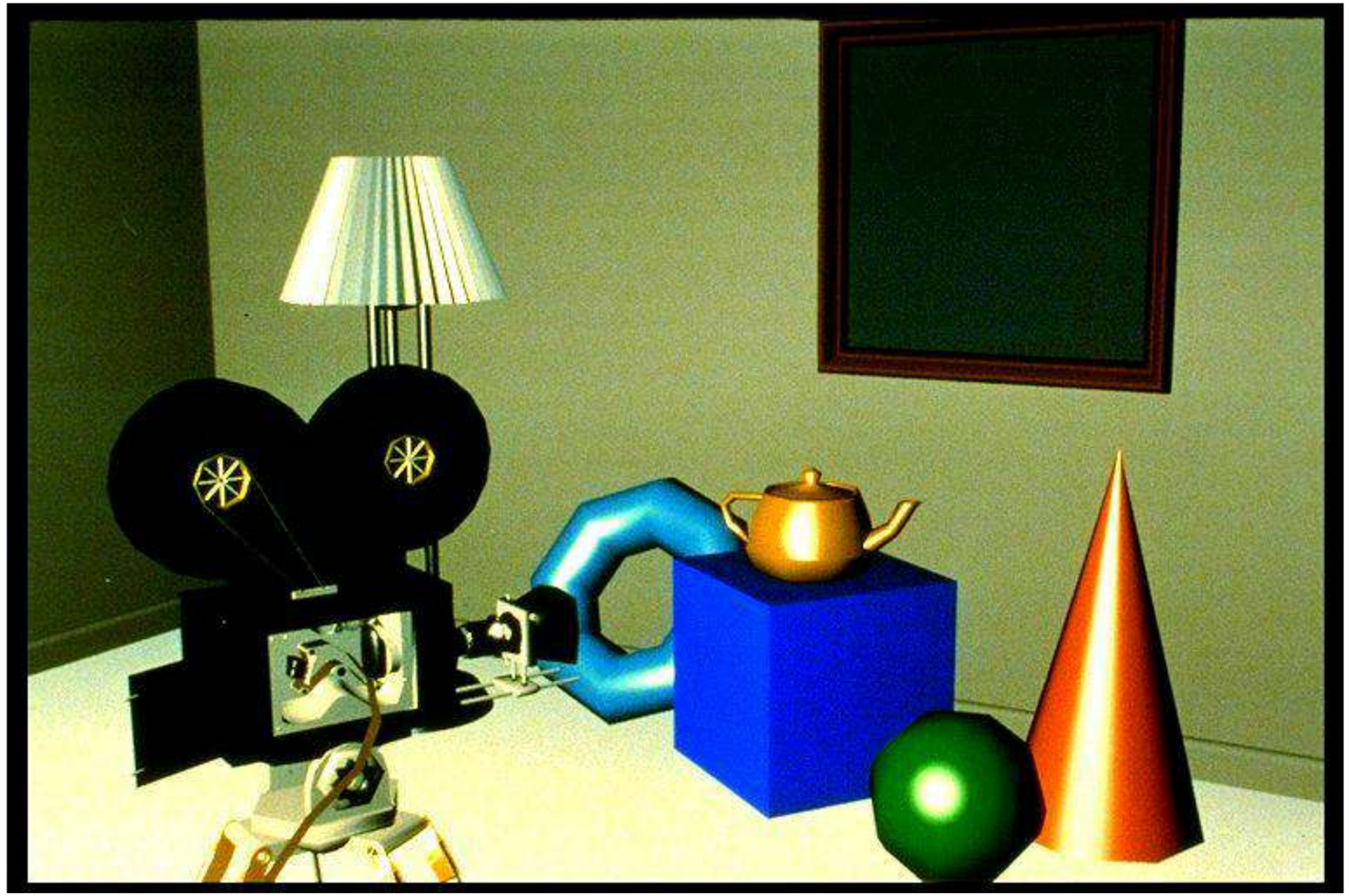
Gouraud-Shading Ideales Shading

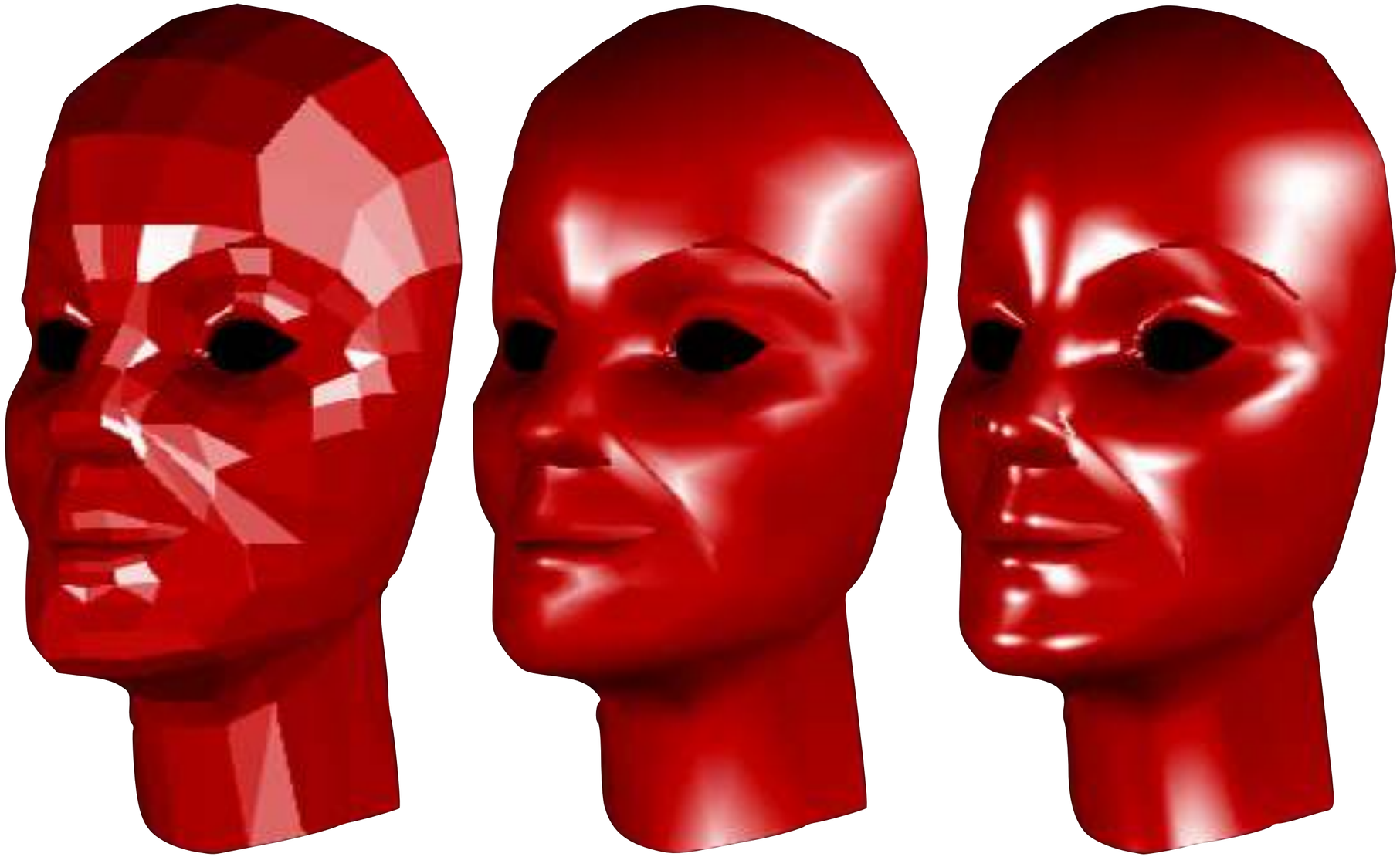
Per-Pixel-Shading (a.k.a. Phong-Shading)

- Idee: werte das Beleuchtungsmodell in jedem Pixel aus
  - Interpoliere dazu die Normale während der Scanline-Konvertierung
- 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_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 (k_a), diffuse (k_d), spiegelnde (k_s) Reflexionskoeffizienten
  - k_a, k_d, k_s sind RGB-Vektoren
- Insgesamt:

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

  - 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

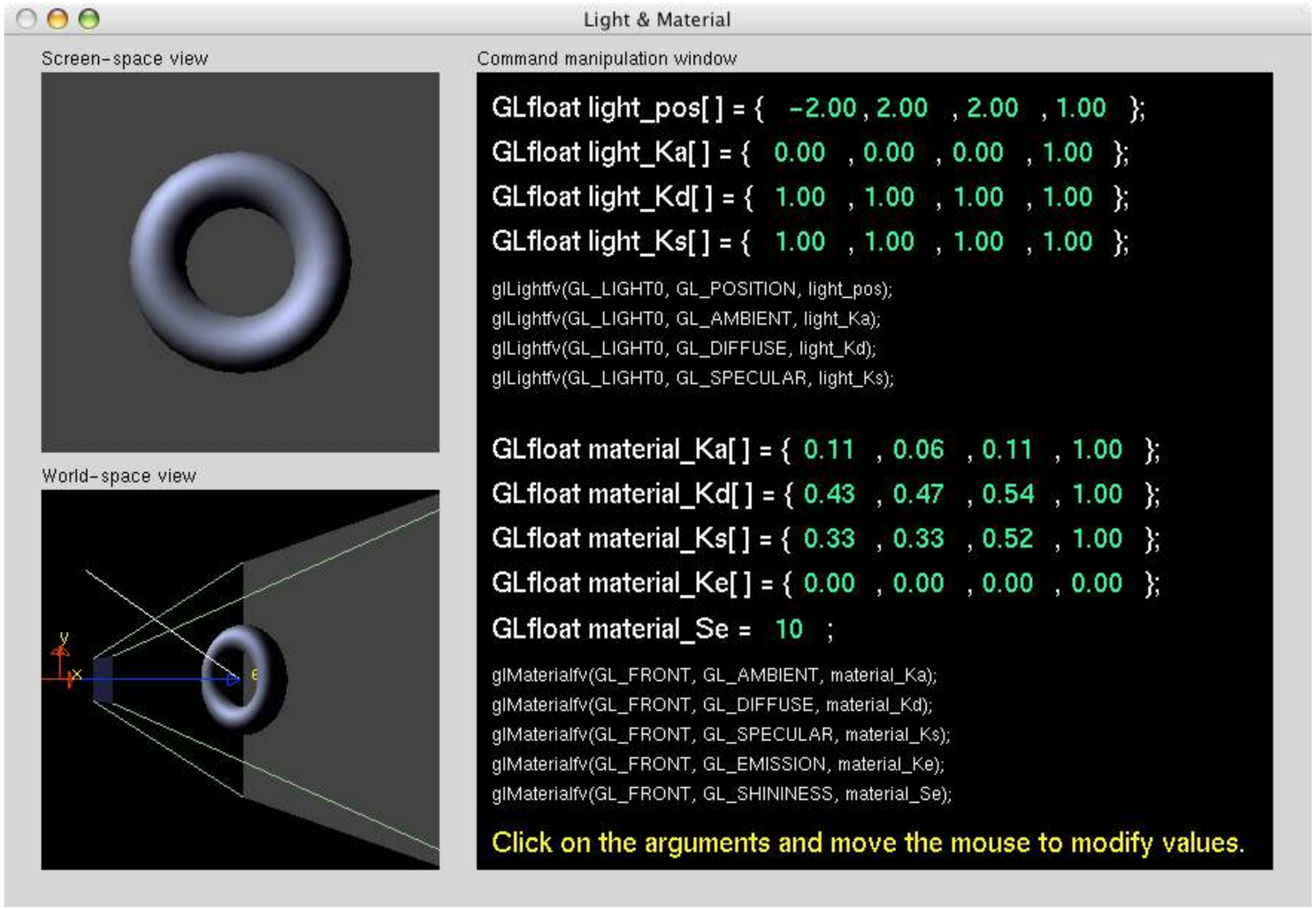
Materialdefinition

```
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

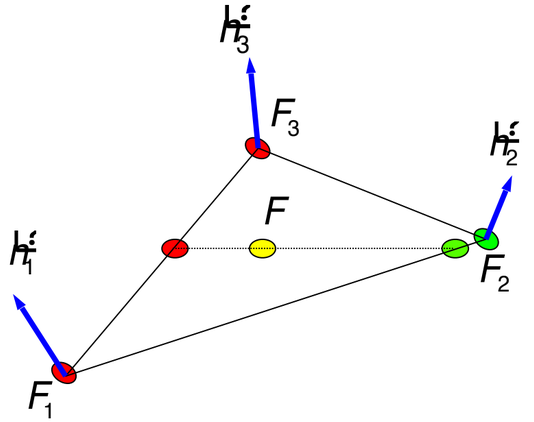


http://www.xmission.com/~nate/

Flat- vs. Gouraud-Shading

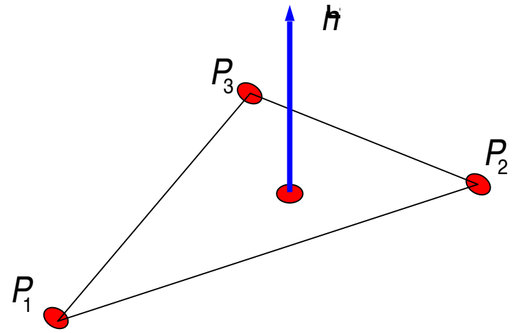
- Gouraud-Shading:

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



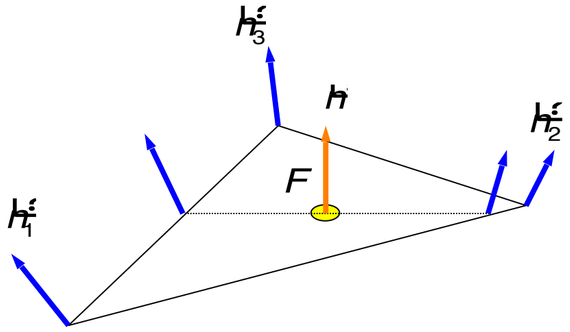
- Flat-Shading:

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



- Phong-Shading:

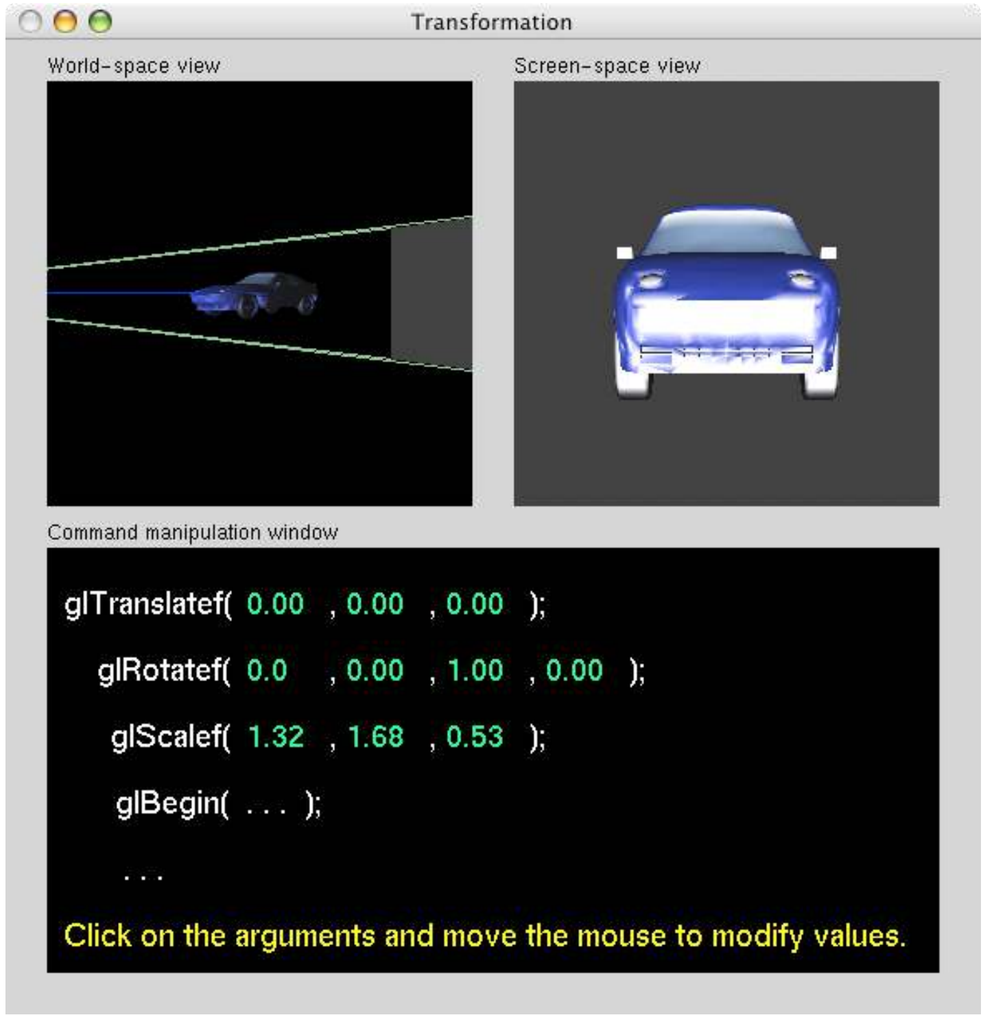
```
// Nur mit Shadern möglich
…
```



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

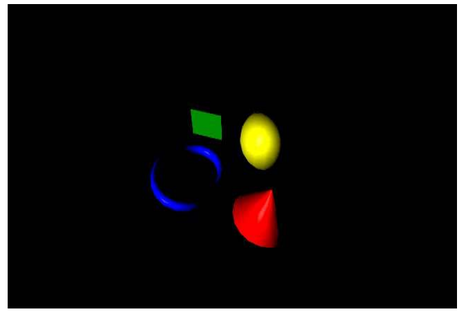
- Die Position (und Richtung) der Lichtquelle wird genauso transformiert wie die Geometrie
  - Transformation mit **GL_MODELVIEW**
- Lichtquelle in Weltkoordinaten („fest am Objekt“):

```
gluLookAt( …);
... set up obj transformation
glLightfv( GL_LIGHT0, GL_POSITION, pos );
drawObject(…);
```

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

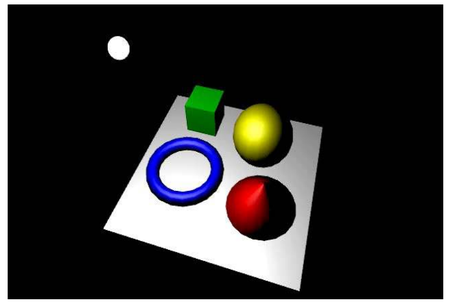
```
glLightfv( GL_LIGHT0, GL_POSITION, pos );
gluLookAt( …);
... set up obj transformation
drawObject(…);
```

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



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

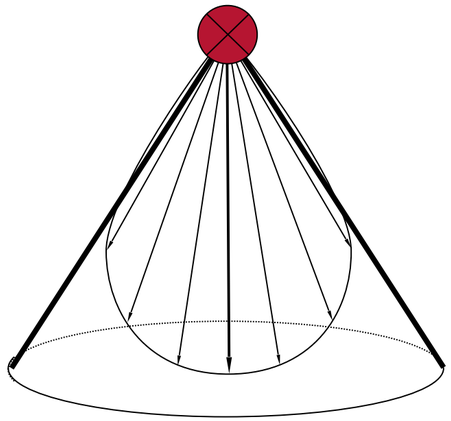
- Punktlichtquellen (*point lights*):
  - Position → homogene Koordinate = 1

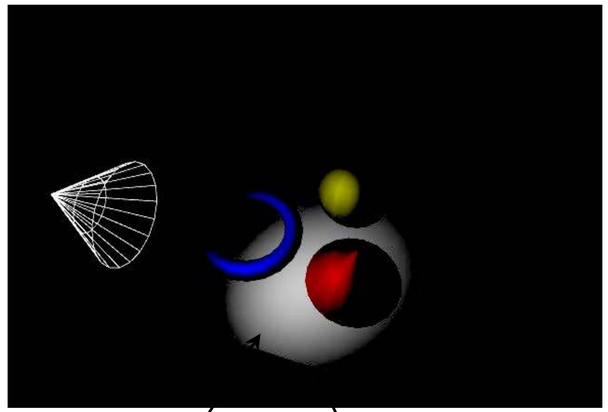


```
GLfloat position[] = { 0.0, -0.5, 0.5, 1.0 };
glLightfv( GL_LIGHT0, GL_POSITION, position );
```

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”





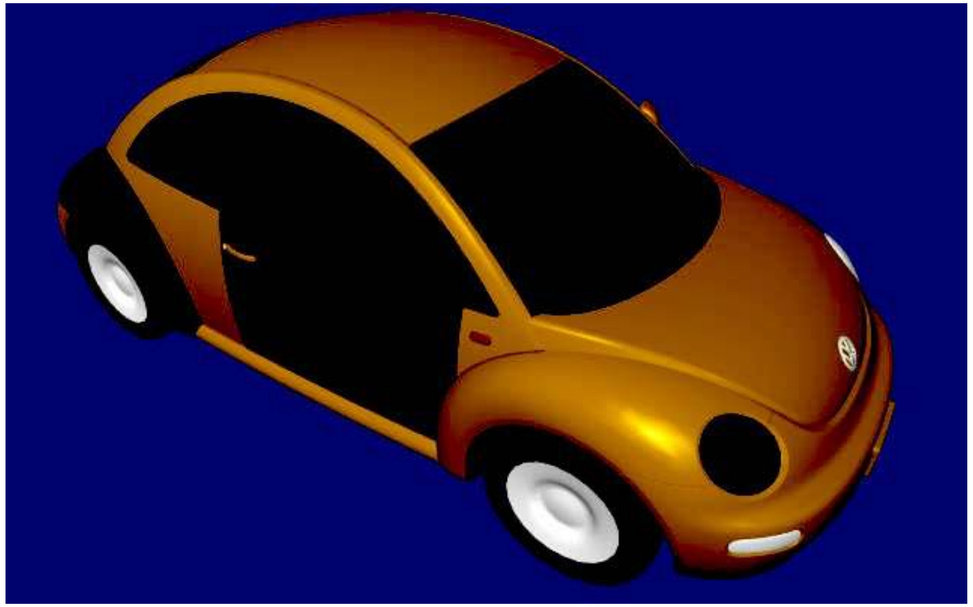
Weitere Parameter des Beleuchtungsmodells

- 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:

$$I = I_a + \sum_{j=1}^{n} a_j \cdot \left(k_a \cdot I_{j,a} + k_d \cdot I_{j,d} \cos \Phi_j + k_s \cdot I_{j,s} \cos^n \Theta_j \right)$$

- Eigentlich ist Abschwächung ~ 1/d²; 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 );
```

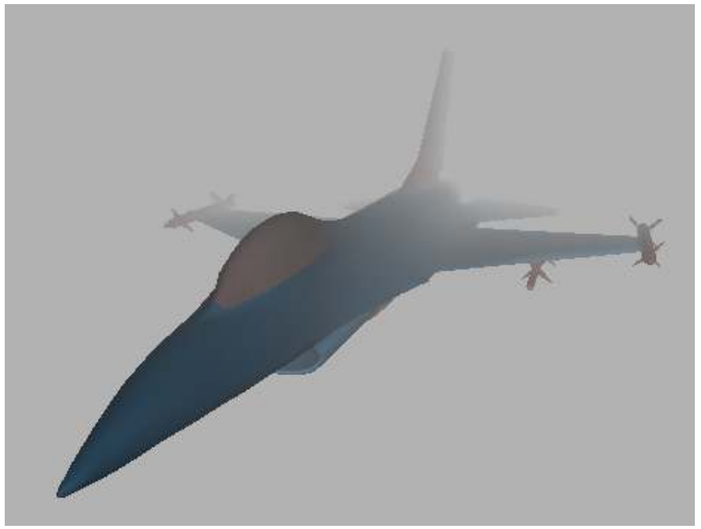
Atmosphärische Dämpfung (Fog)

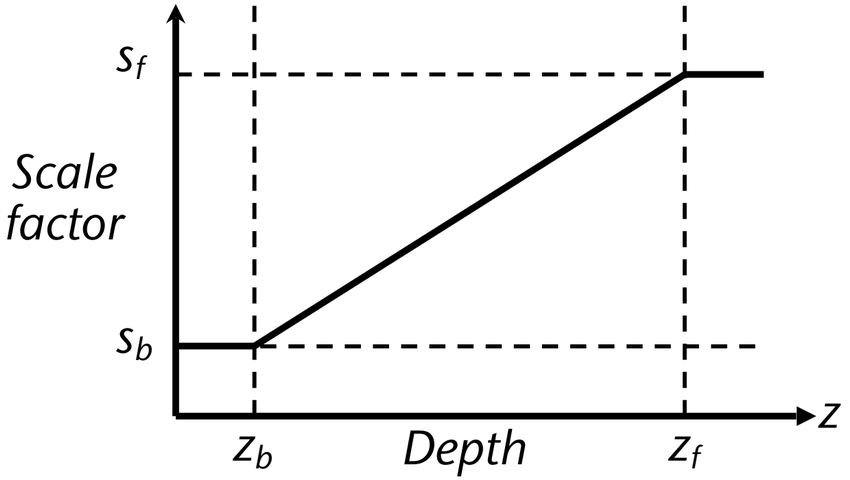
- Lineare Abnahme der Intensität beim Lichtweg durch Medien mit feinen Partikeln(z.B. Nebel)

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

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

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





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

Wo wird in der Graphik-Pipeline beleuchtet?

- In welchem Koordinatensystem wird das Beleuchtungsmodell ausgewertet?
  - Model-, World-, Camera-, Screen-Space-Koordinaten?

- In welcher Pipeline-Processing-Stufe?