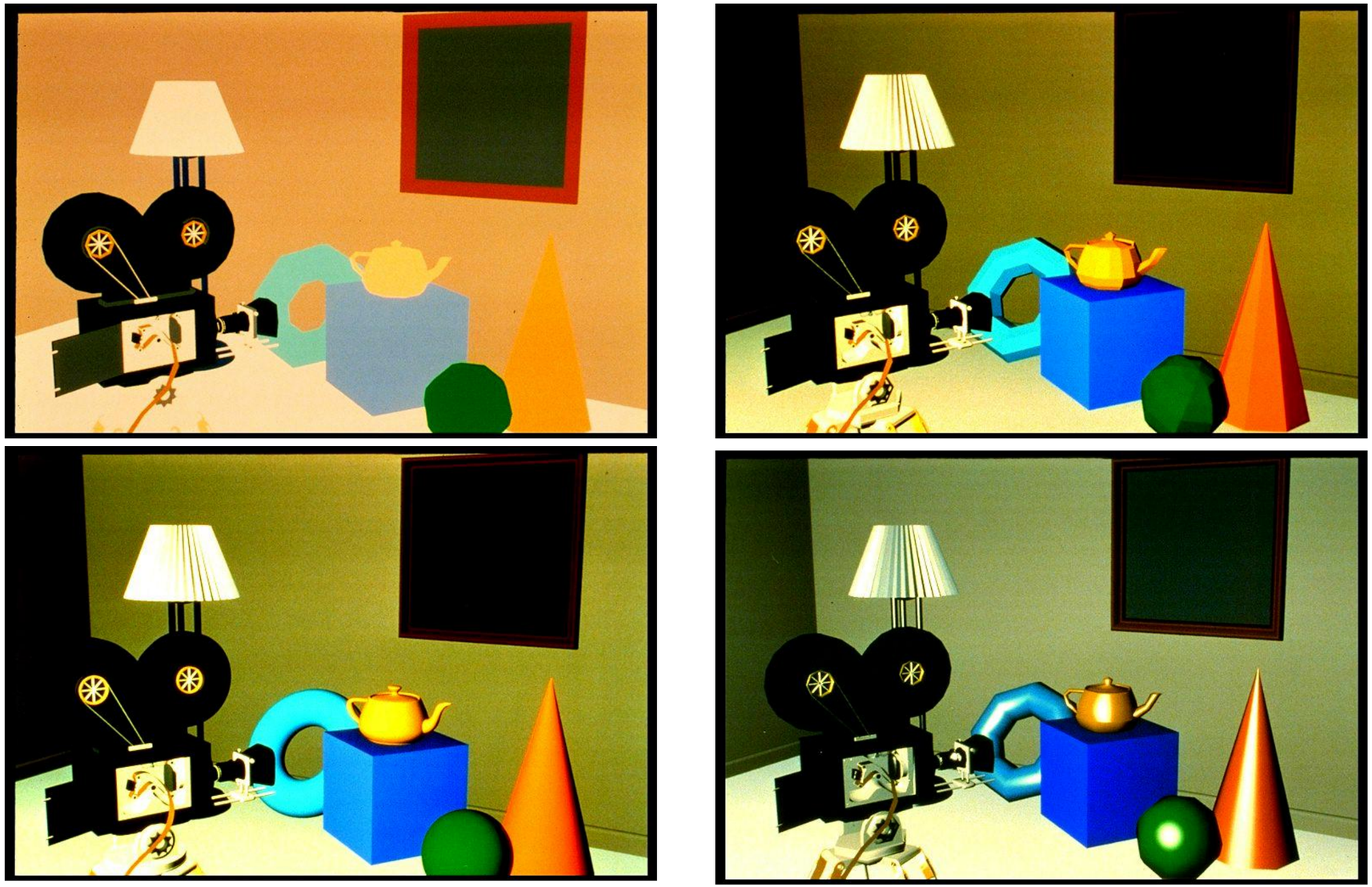
Computer-Graphik 1
Lighting & Shading

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

Warum ist Lighting & Shading so wichtig?

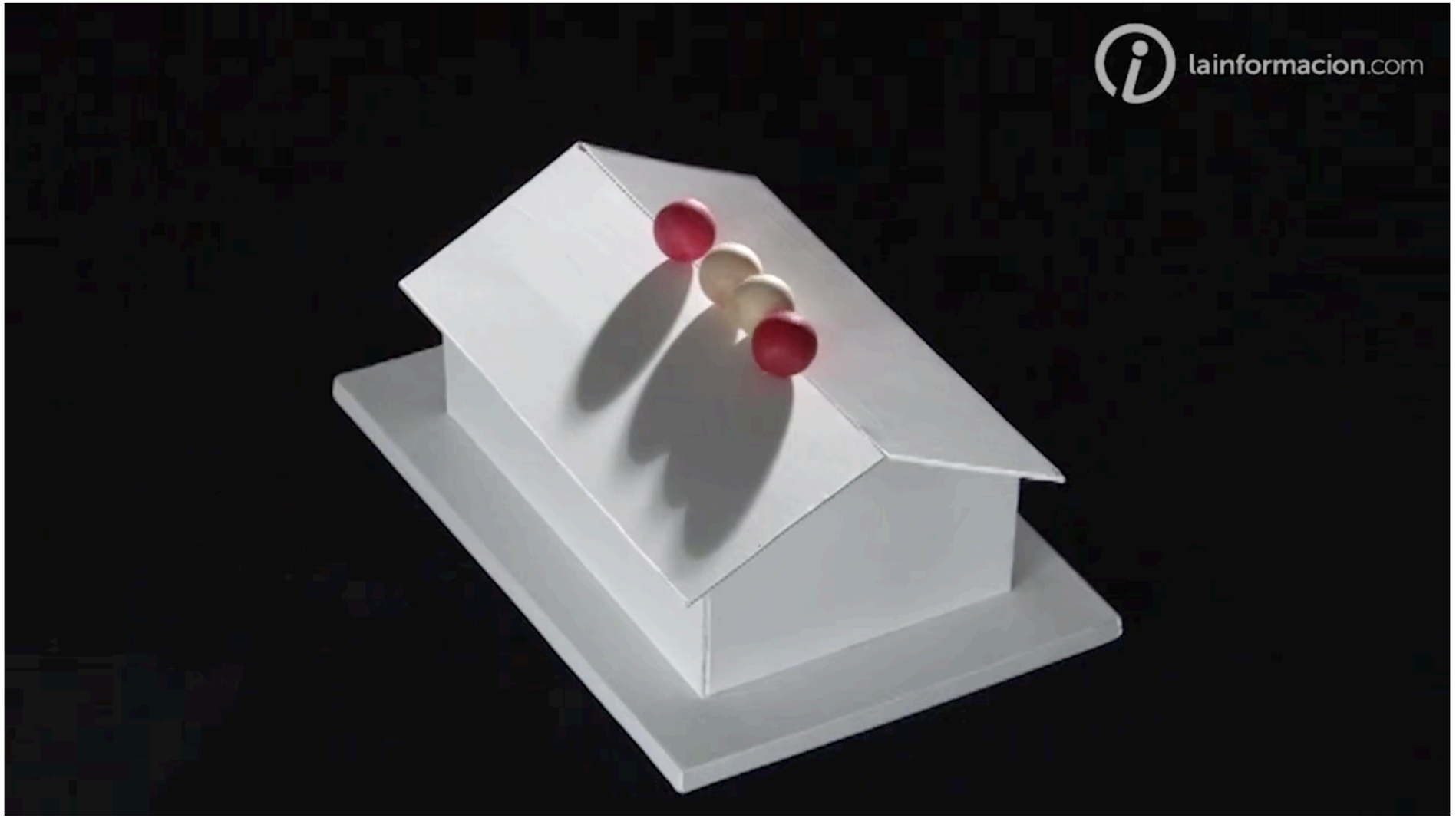
https://www.menti.com/rs2rt7mgjb

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

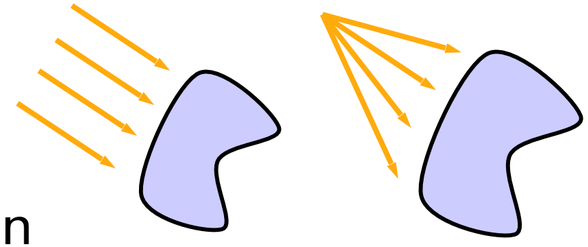
Guides the audience's eye

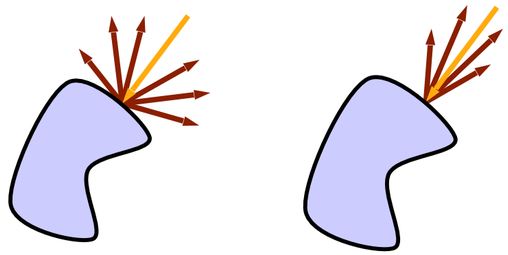
Makes the character stand out

Definition 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 des Materials)

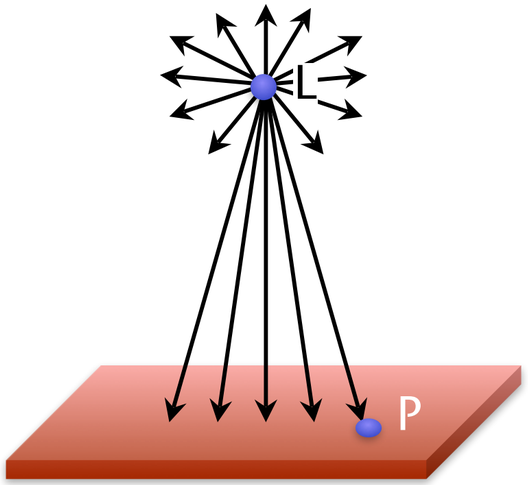




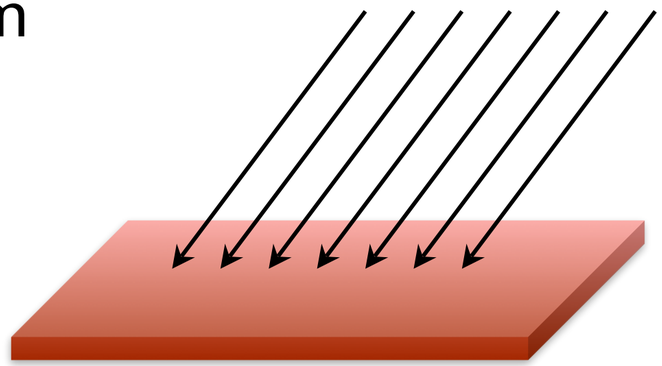
- Für Echtzeitanwendungen verwendet man sehr einfache Modelle

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: $I(P) = \frac{1}{(L-P)^2} I_0$

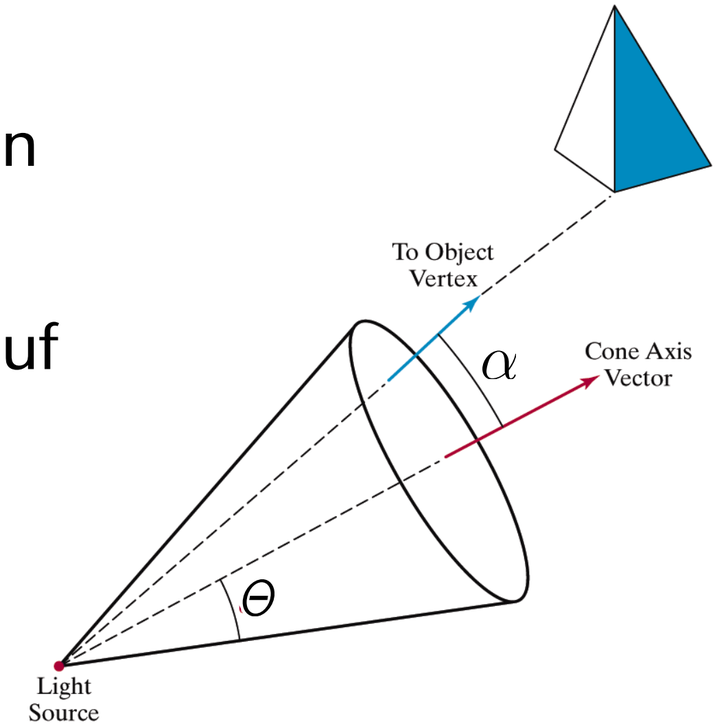


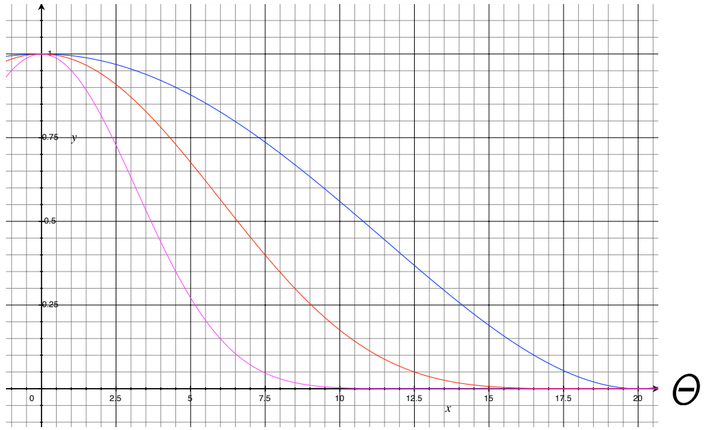
- Richtungslichtquelle *(directional light)*: jeder Punkt im Raum wird aus derselben Richtung bestrahlt
  - Charakterisiert durch Richtung & $I(\lambda)$
  - Beispiel: Sonne

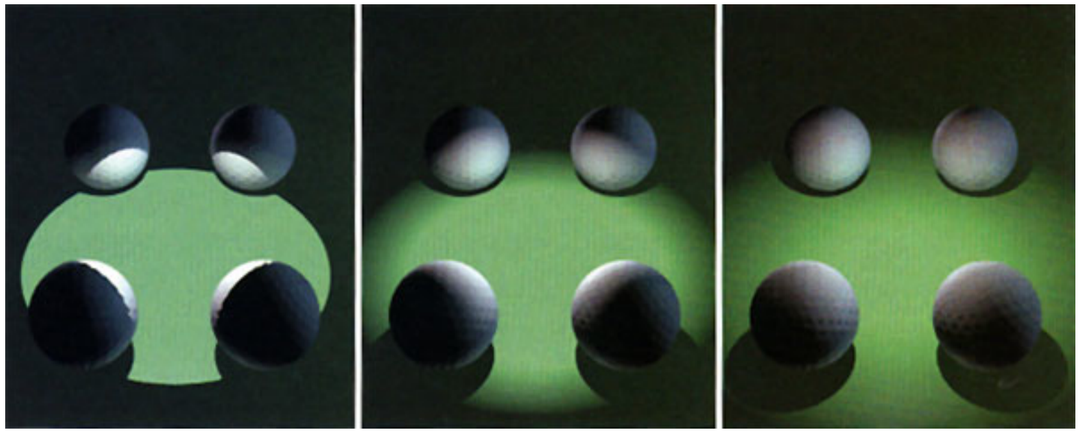


- Strahler (*spot light*): Lichtausbreitung wird auf einen bestimmten Raumwinkel (Lichtkegel) beschränkt
- Gesucht: Funktion, die $\cos(\alpha) \in [1, \cos(\theta)]$ abbildet auf [1,0] mit "schönem" Fall-off
- Diese tut's:

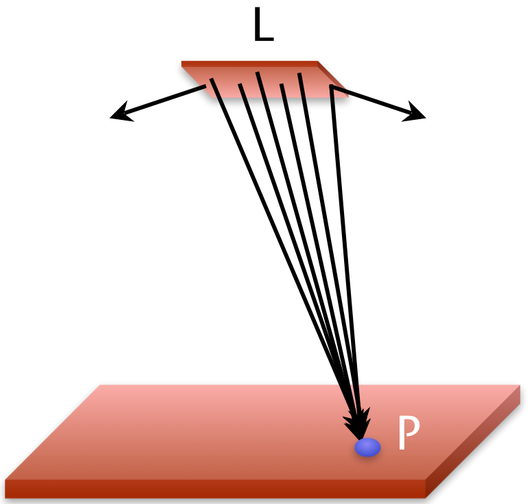
$$I = \begin{cases} 0 & \text{falls } \cos\alpha < \cos\theta \\ \frac{\cos^n \alpha - \cos\theta}{1-\cos\theta} I_0 & \text{sonst} \end{cases}$$



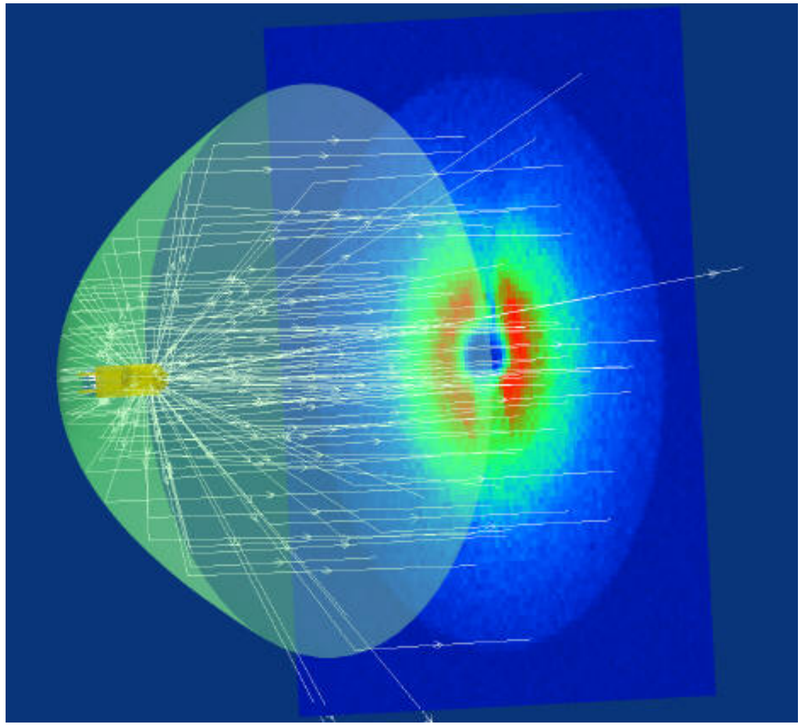




- Area light source:
  Wie Punktlichtquelle, aber mit Fläche statt Punkt als Ausgangsort

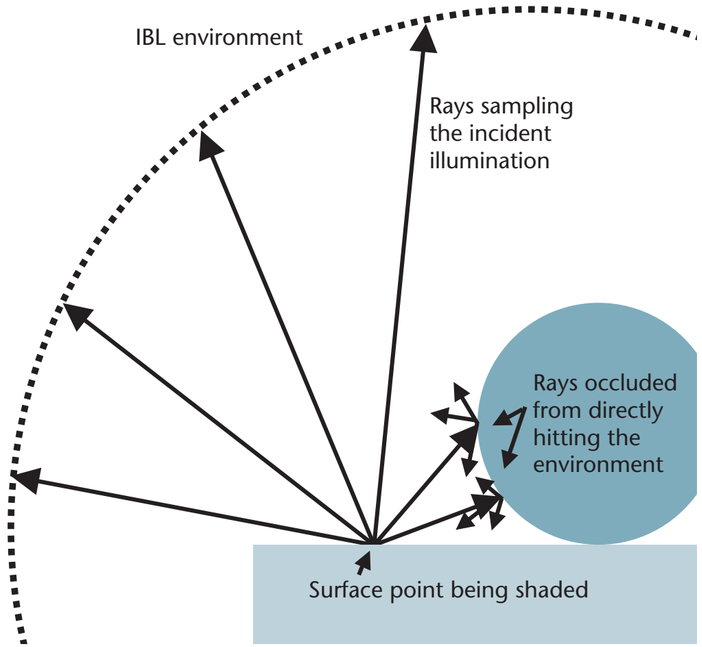


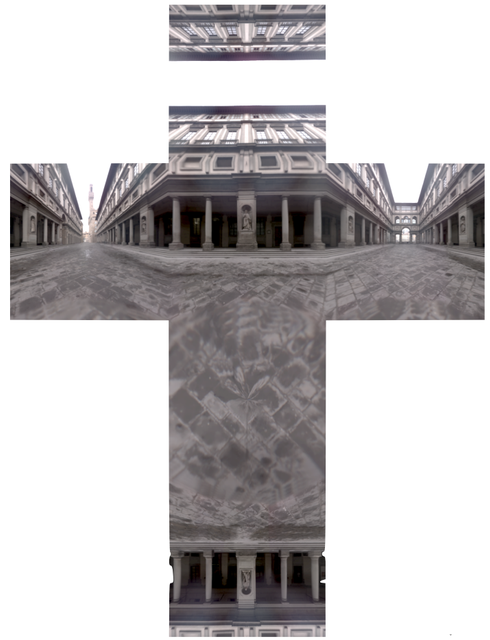
- Goniometrische Lichtquelle:
  Abstrahlcharakteristik wird per Tabelle beschrieben. Zur Ermittlung von $I_0(\lambda)$ muß evtl. zwischen Tabelleneinträgen interpoliert werden



- Image-based lighting:
  - Light probe := omnidirectional, high dynamic range image that records the incident illumination conditions at a particular point in space
  - On object's surface:
    shoot rays in all directions
    and accumulate the
    incident illumination

Check out the video "Fiat Lux" on the course home page!

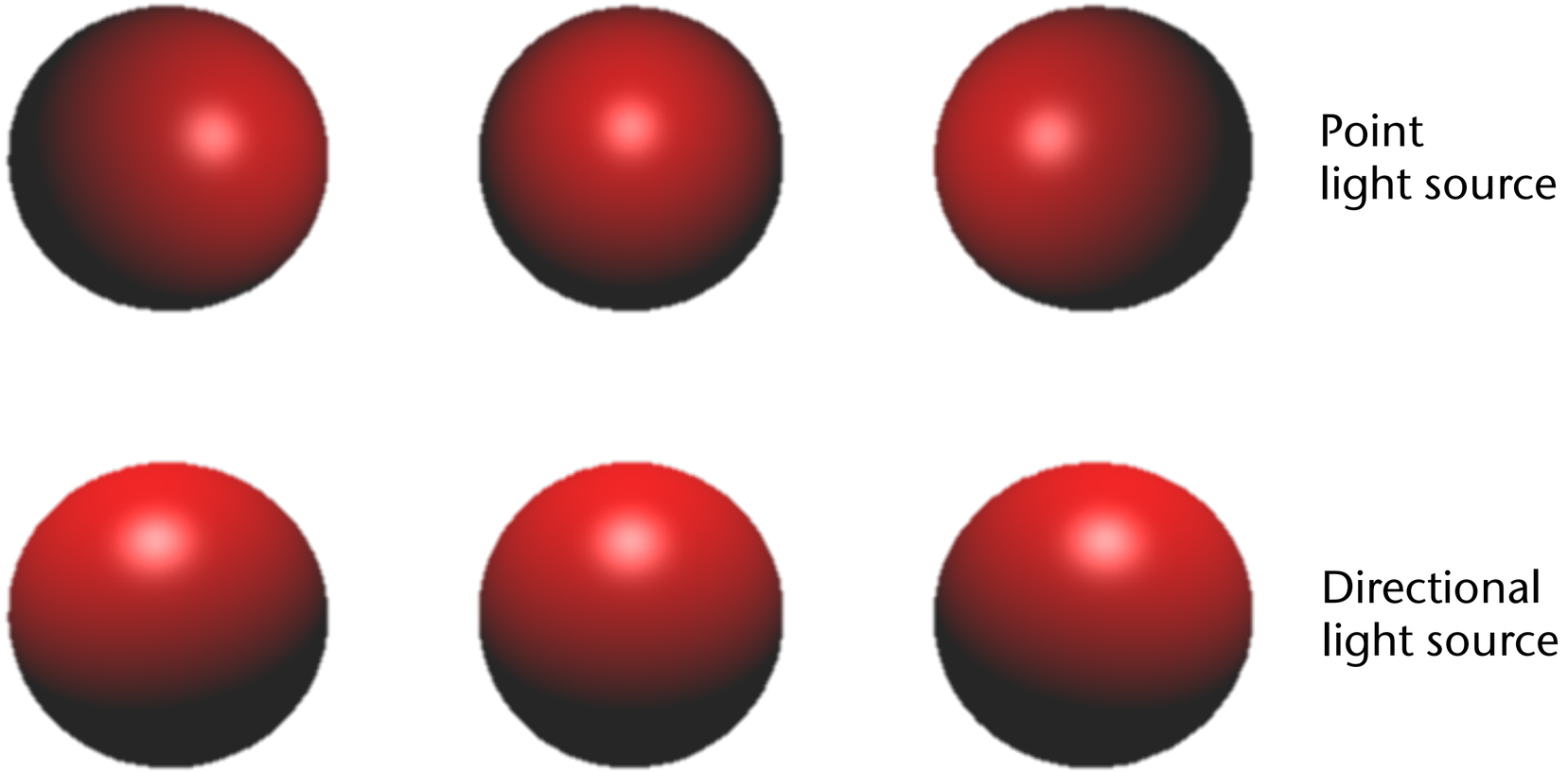




by Paul Debevec

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?)



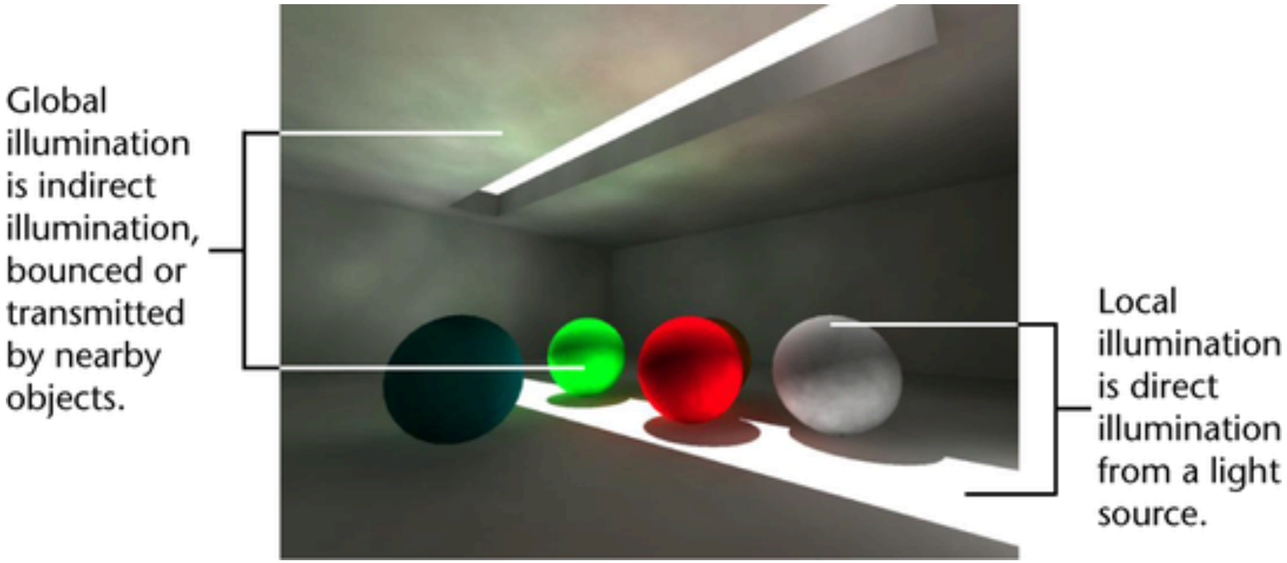
Point
light source

Directional
light source

https://www.menti.com/rs2rt7mgjb

Definition lokales Beleuchtungsmodell

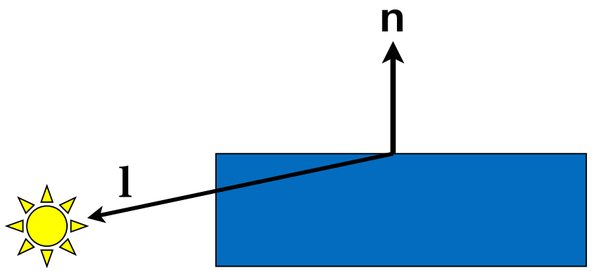
- Lokales Beleuchtungsmodell: berücksichtige bei der Berechnung der Beleuchtung eines Objs keine indirekten Effekte (kein Austausch zwischen Objekten), nur direkten Austausch von Lichtquelle zu Objekt zu Auge



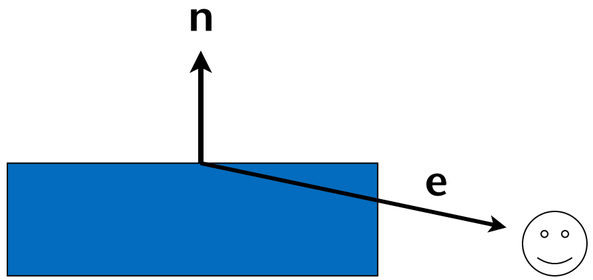
- Superpositionsprinzip: $I = \sum_j I_j$ (keine Interferenz oder Beugung)
- Vereinfachung der Notation: wir lassen im Folgenden λ überall weg, und merken uns, daß alle photometrischen Größen eigtl. von λ abhängen!

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

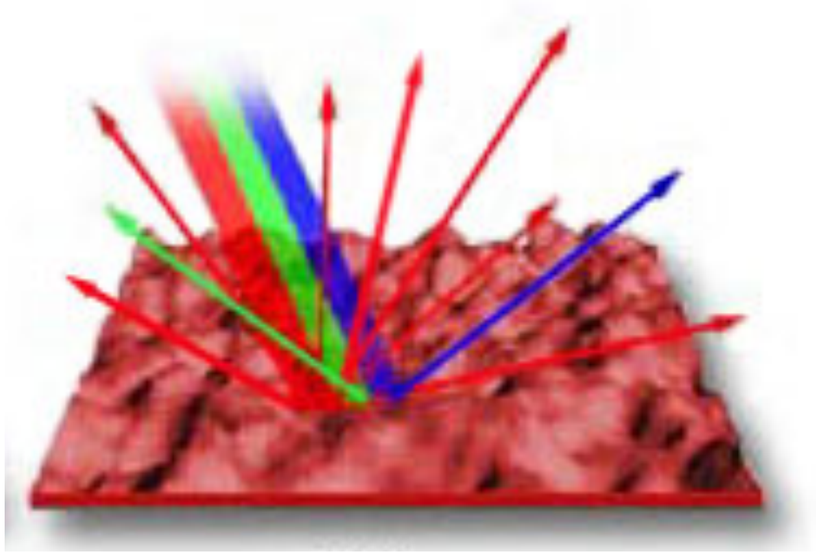


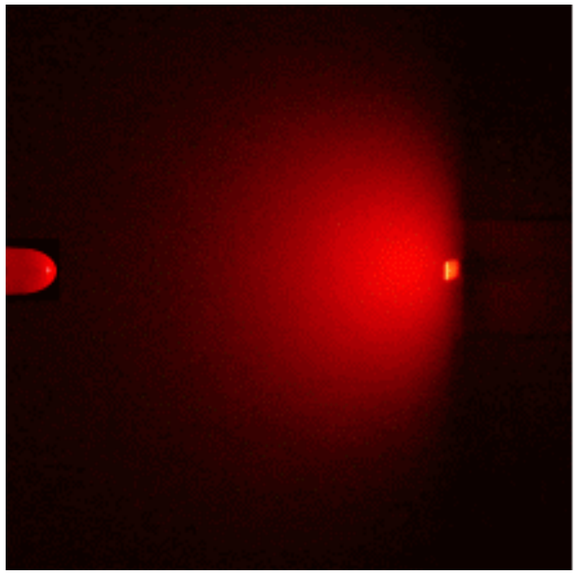
- Wir definieren im Folgenden (der Einfachheit halber) prinzipiell:

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

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

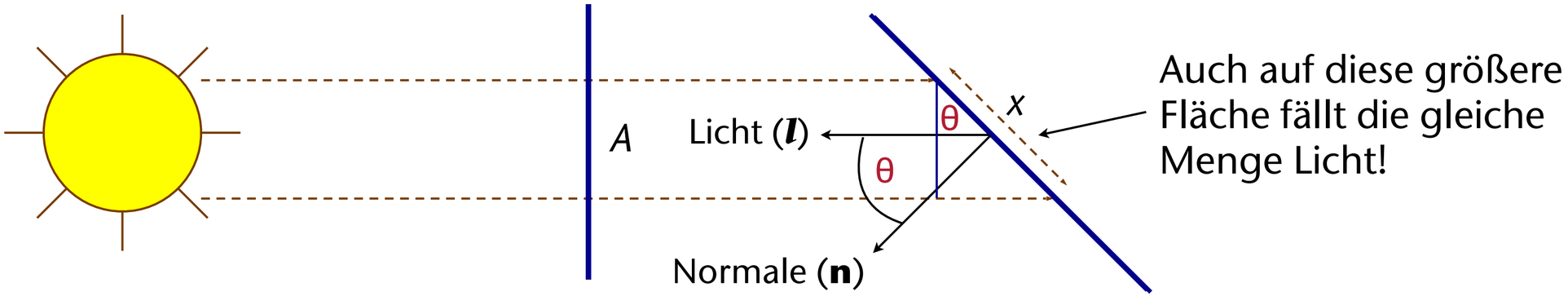
- Das Lambert'sche Kosinus-Gesetz:

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

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

- "Beweis":

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

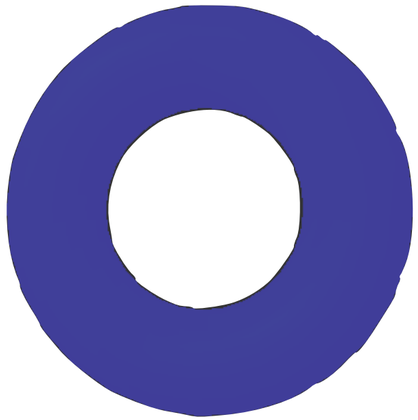


Ambientes Licht

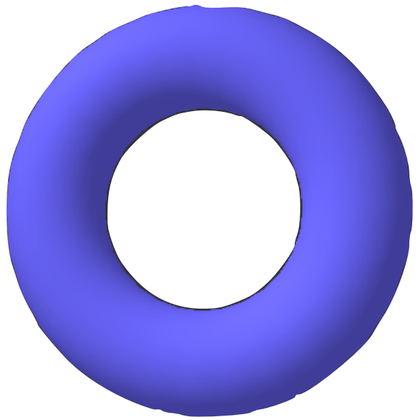
- Das Lambert'sche Modell erzeugt schwarze Farbe für Oberflächen, die nicht zur Lichtquelle zeigen
- In der Realität trifft Licht aus allen Richtungen ein (dieses wurde von anderen Objekten, evtl. mehrfach, reflektiert)
- Hack: füge für alle Objekte einen ambienten Beleuchtungsterm ein

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

Nur diffuse Beleuchtung



Nur ambiente Beleuchtung

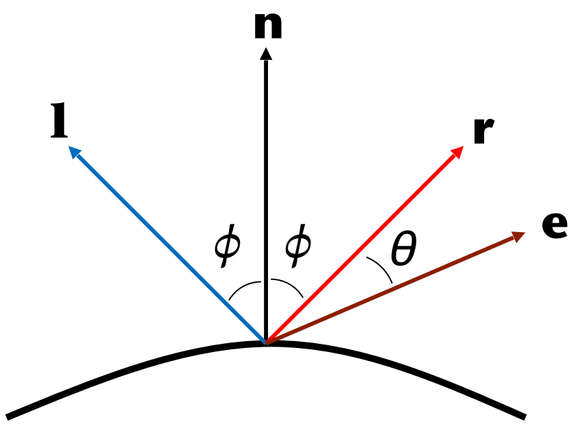


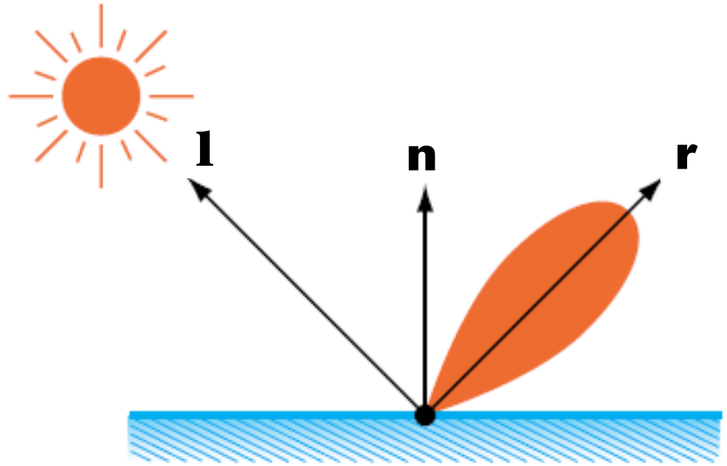
Diffuse + ambiente Beleuchtung

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

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





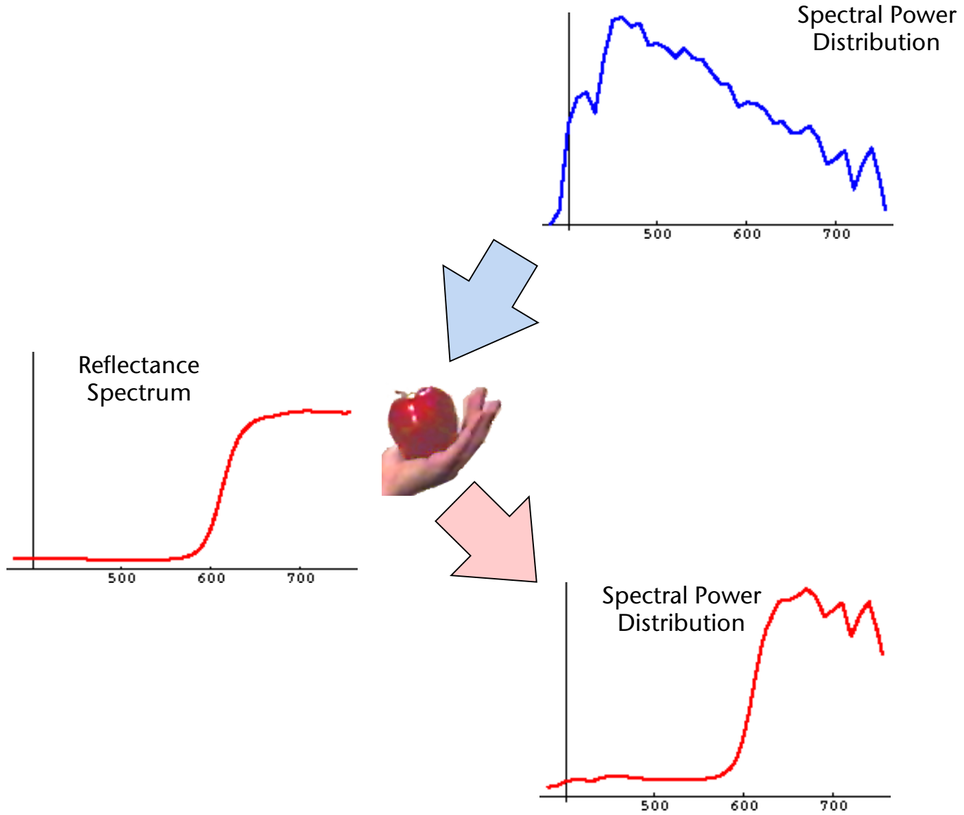
Highlights can Reveal Poorly Faked Fotos

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)



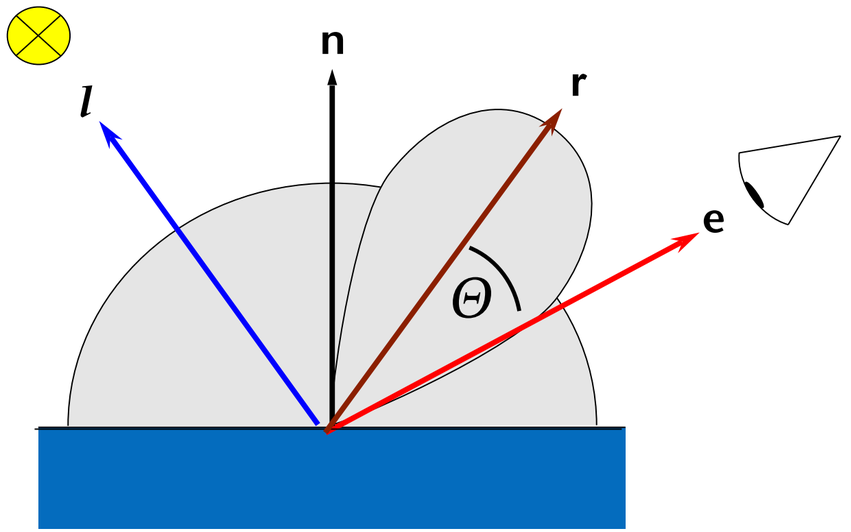
- Mit Reflexionskoeffizient *k*:

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

Das Phong-Beleuchtungsmodell

- Zusammensetzung:

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

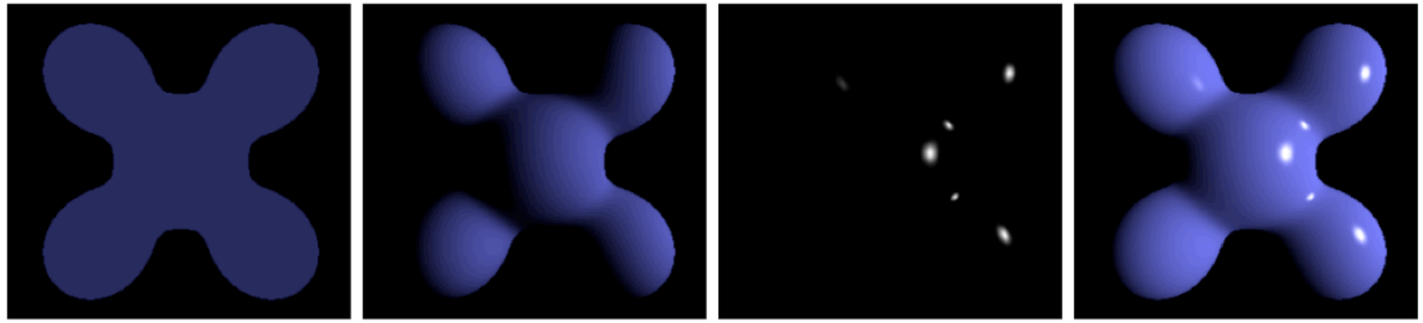


- Aufgrund des Superpositionsprinzips erhält man für *n* Lichtquellen:

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

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

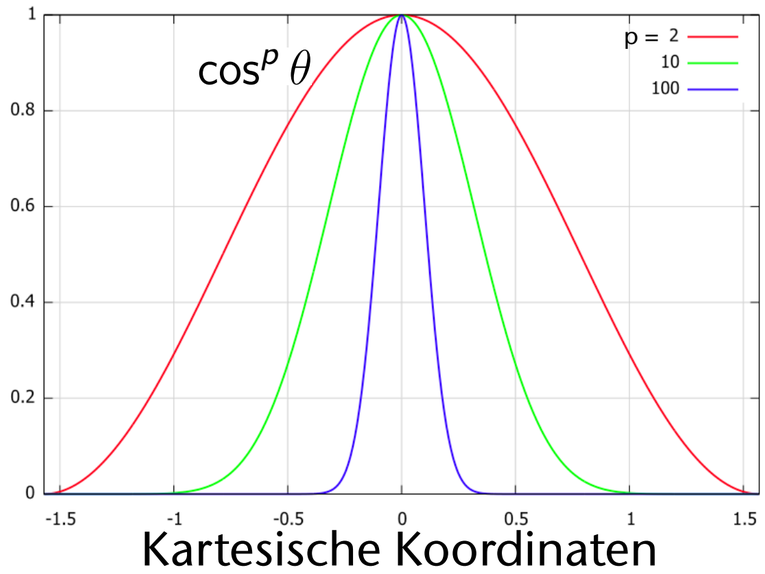
ambient + diffus + spekular = Phong



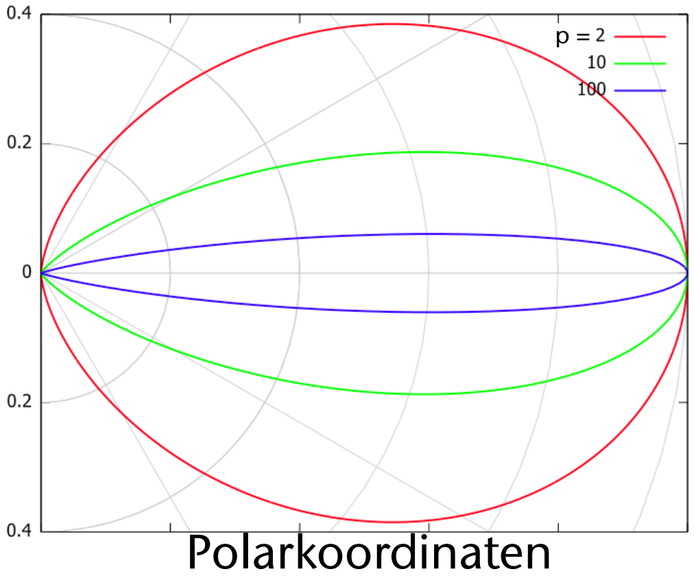
Phong	$\rho_{ambient}$	$\rho_{diffuse}$	$\rho_{specular}$	ρ_{total}
$\phi_i = 60^\circ$				
$\phi_i = 25^\circ$				
$\phi_i = 0^\circ$				

Effekt der Parameter im Phong-Modell

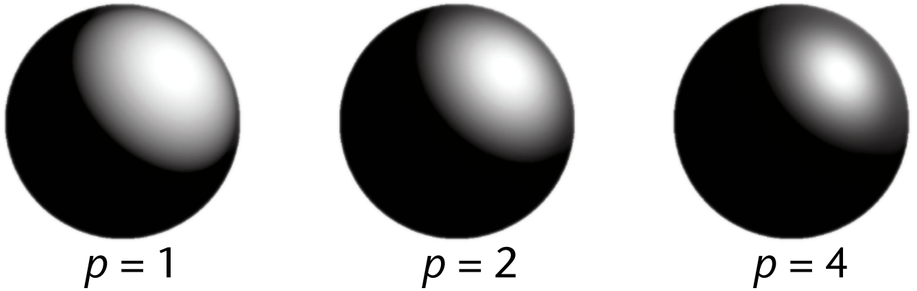
- Die Shininess *p* steuert die "Schärfe" des Highlights:

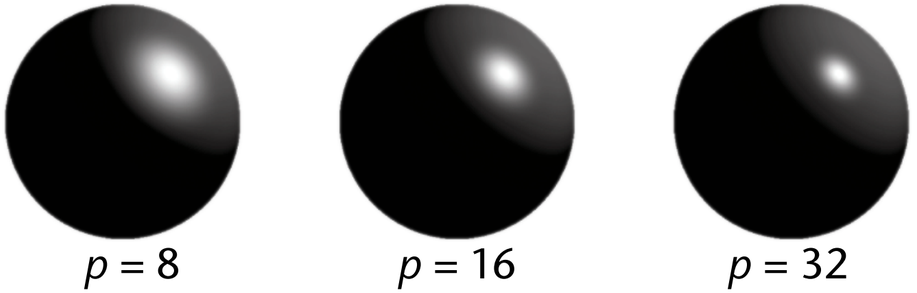


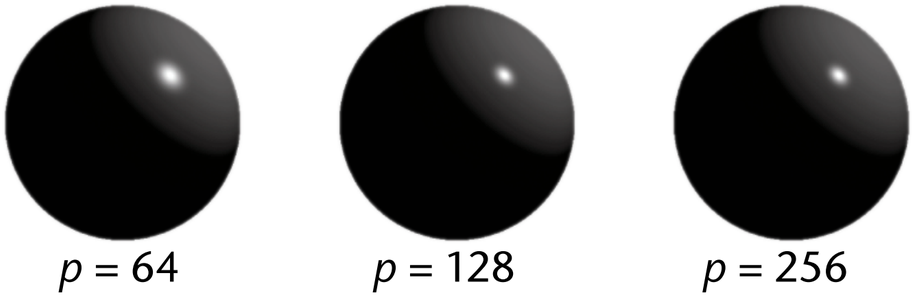
Kartesische Koordinaten



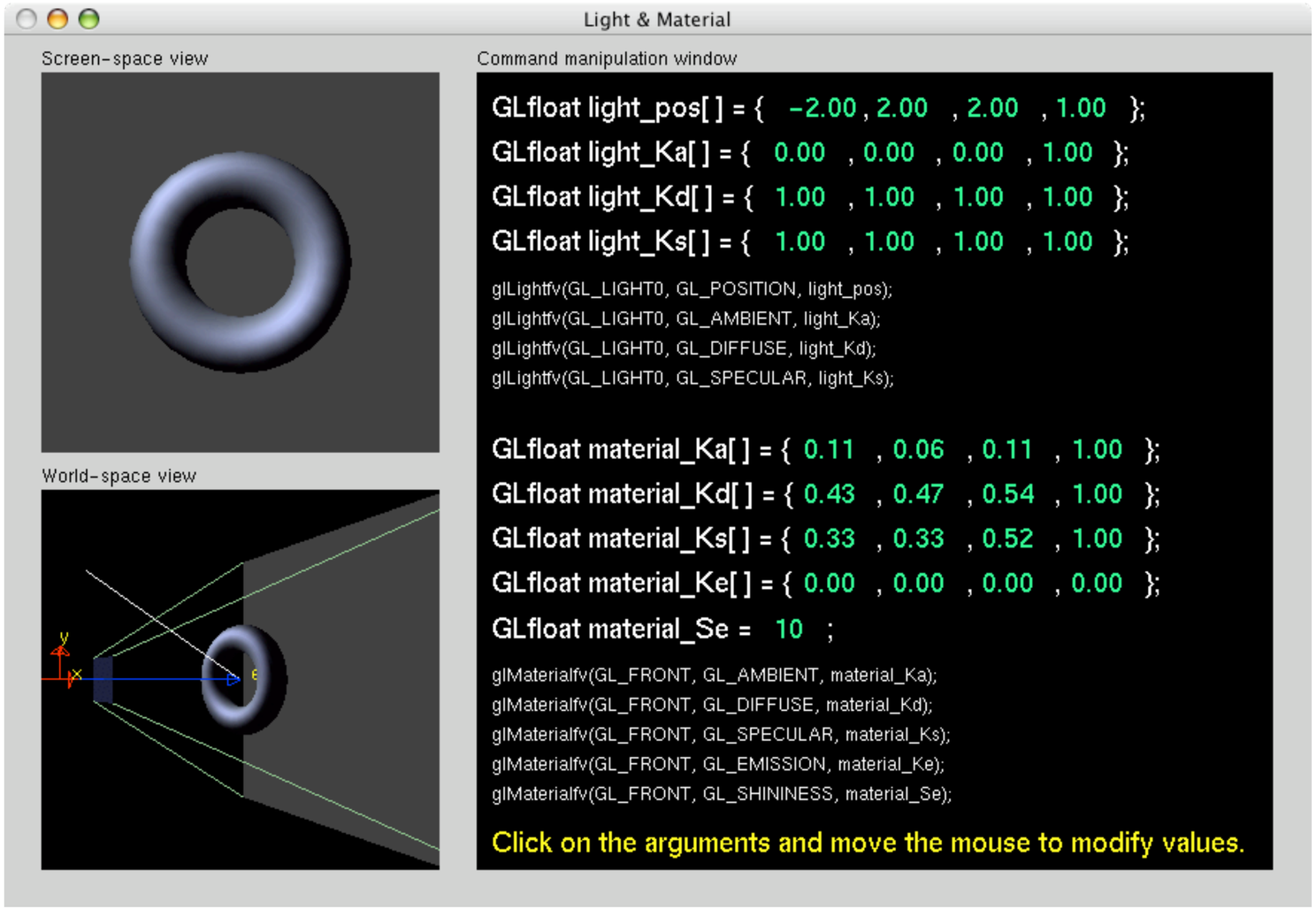
Polarkoordinaten


p = 1 *p* = 2 *p* = 4

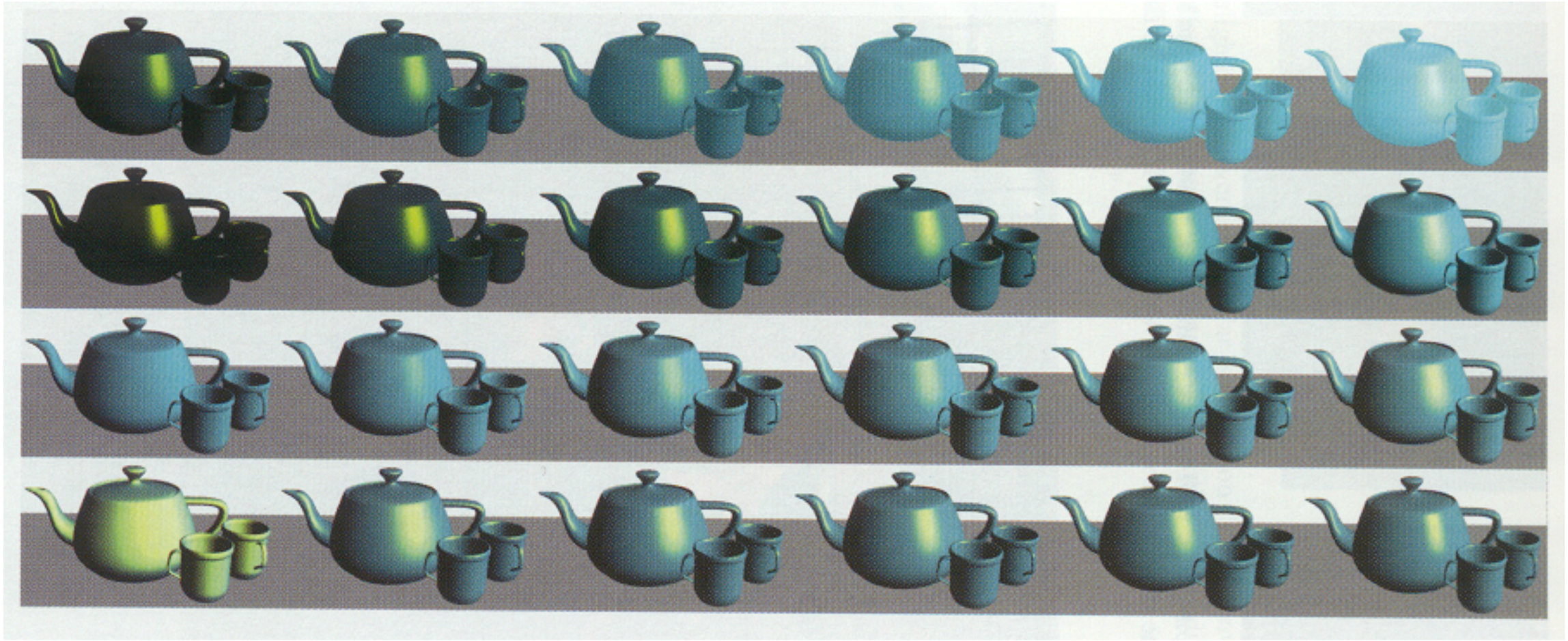

p = 8 *p* = 16 *p* = 32


p = 64 *p* = 128 *p* = 256

Demo



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

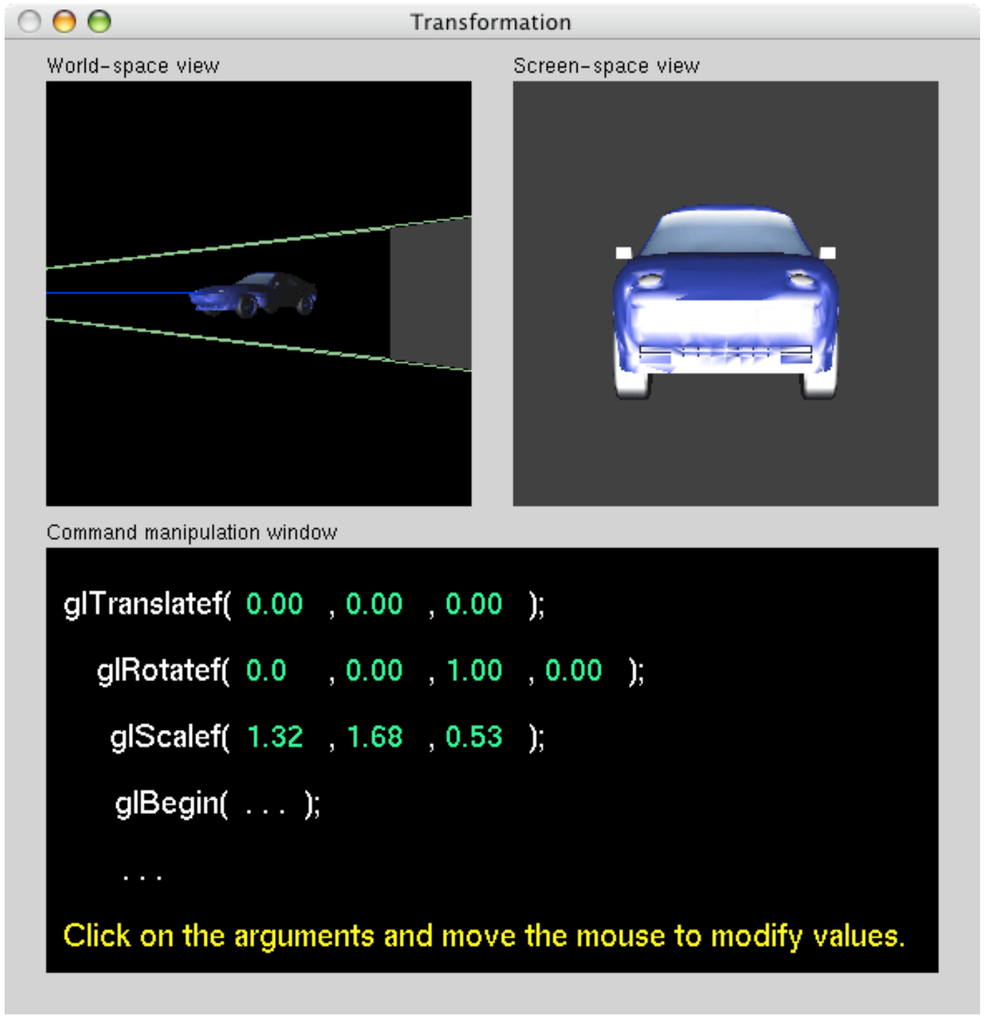

k_a
k_d
k_s
p
Größere Werte

Effekt von nicht-normierten Normalen in der Beleuchtung

Achtung:

Alle Beleuchtungsmodelle setzen voraus, daß die Normalen **normiert** sind!

Wenn das nicht der Fall ist, sind die Objekte zu hell oder zu dunkel (s. Demo)



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

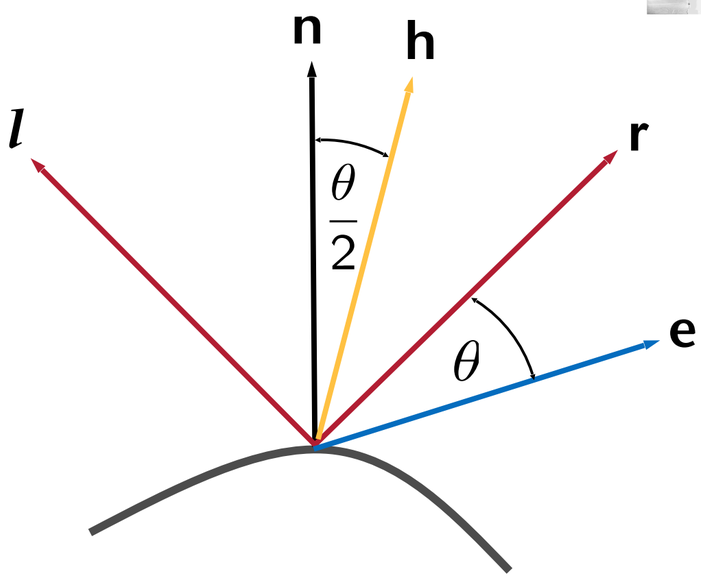
Spectral Lighting

- 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 somit 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: $\text{RGB}(k(\lambda) \cdot I(\lambda)) \neq \text{RGB}(k(\lambda)) \cdot \text{RGB}(I(\lambda))$

Das Blinn-Phong Modell



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

Fragen zum Phong-Modell

https://www.menti.com/rs2rt7mgjb

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
}
```

Manche (die meisten) Materialien sind viel komplizierter

Lighting Models in General

- Reflectance := $\frac{\text{Gesamte eintreffende Lichtenergie}}{\text{Gesamte ausgehende Lichtenergie}}$ (auf ein kleines Stück der Oberfläche)

- 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 eine Art "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

- Oft fasst man die Winkel zusammen:

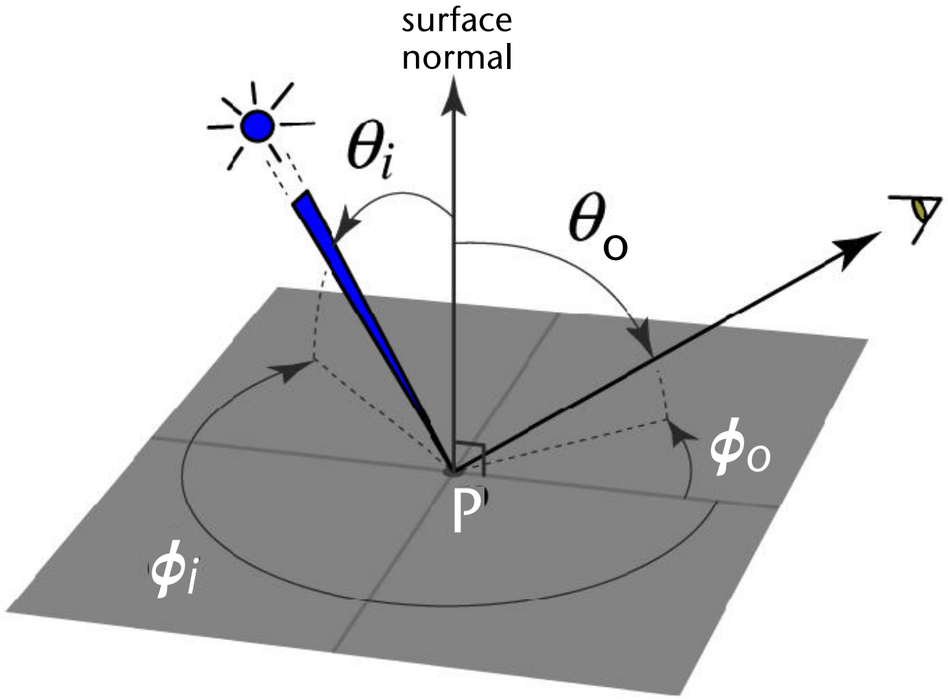
$$\omega = (\theta, \phi)$$

und man schreibt dann

$$\rho = \rho(\omega_i, \omega_o)$$

oder sogar mit Vektoren

$$\rho = \rho(\boldsymbol{l}, \mathbf{v})$$



FYI

- Die physikalisch korrekte Definition der BRDF ist etwas komplexer
- Basiert auf der Radiance L_i = *Lichtintensität pro auftreffender Fläche pro Raumwinkel, aus dem das Licht "eingesammelt" wird*
  (analog für ausgehende Radiance L_o)
- Definition:

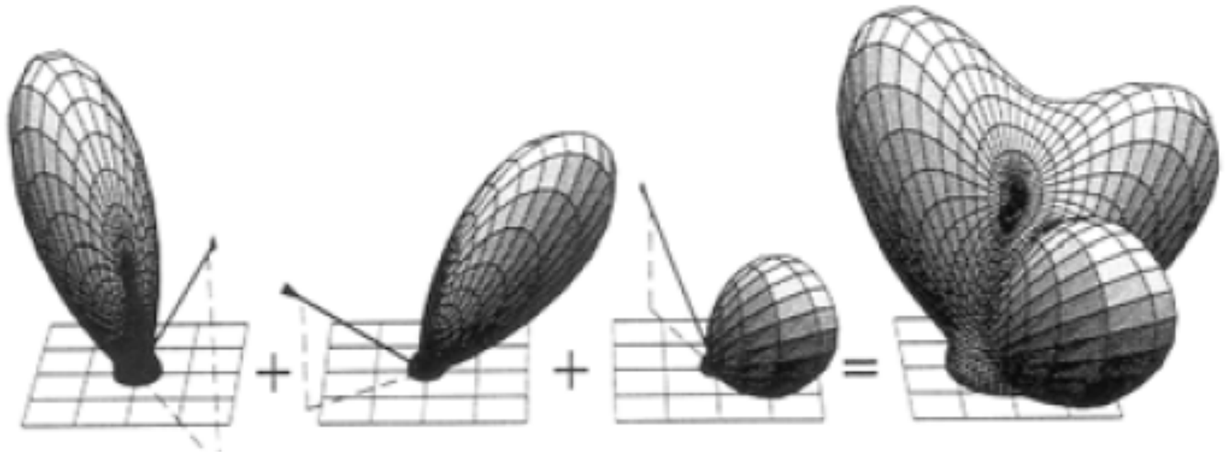
$$\rho(\omega_o, \omega_i) = \frac{\mathrm{d}L_o(\omega_o)}{L_i(\omega_i)\cos\theta_i \,\mathrm{d}\omega_i}$$

  - Mehr dazu in "Advanced Computer Graphics"

Wünschenswerte Eigenschaften von BRDFs

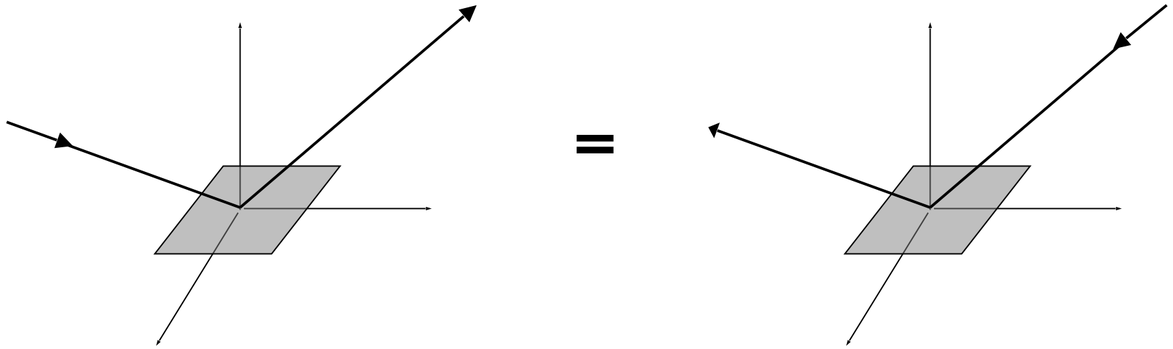
- Folgende Annahmen/Eigenschaften werden (meist) postuliert (für die meisten Materialien gelten sie tatsächlich)

1. Die BRDF eines Materials hängt nicht von dem jeweiligen Punkt auf der Oberfläche ab!
2. Linearität: BRDFs können einfach mittels Addition akkumuliert werden

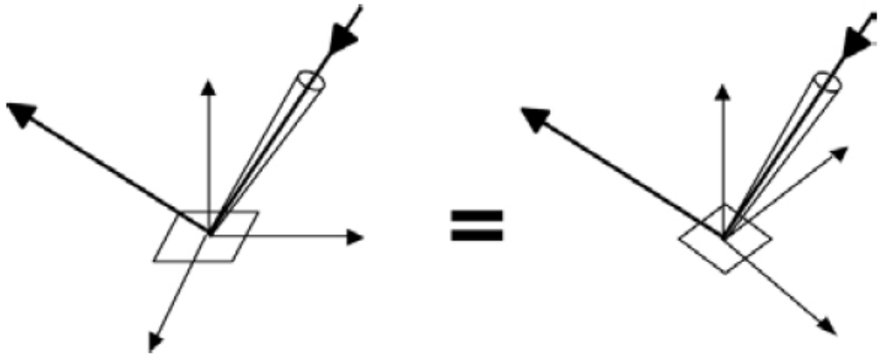


3. Reciprozität (reciprocity):

$$\rho(\omega_i, \omega_o) = \rho(\omega_o, \omega_i)$$

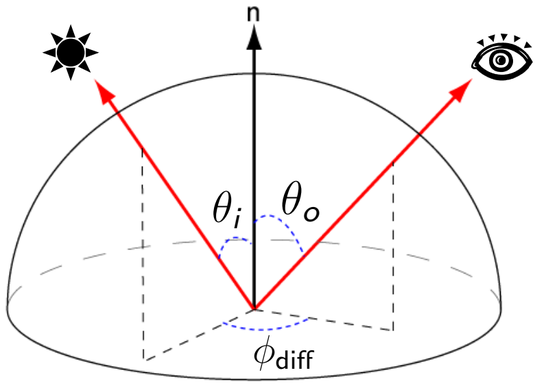


4. Isotropie (isotropy): BRDF ist invariant bzgl. Rotation des Materials um die Normale



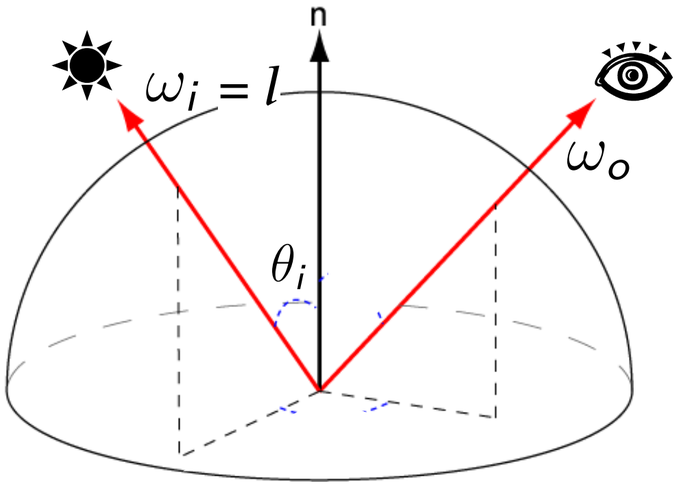
- Konsequenz aus Reciprocity und Isotropie (falls gegeben): man benötigt nur 3 Parameter

$$\rho = \rho(\theta_i, \theta_o, \phi_{\text{diff}})$$



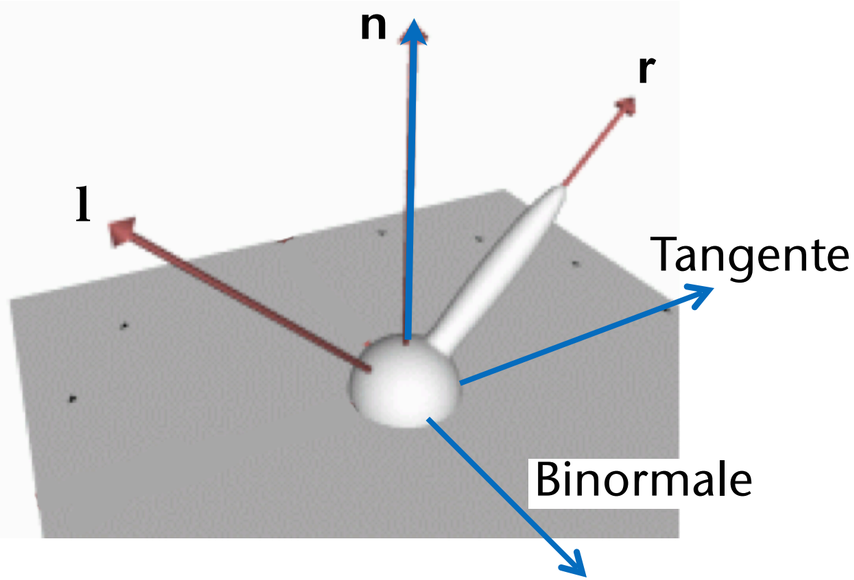
5. Energieerhaltung (conservation of energy): total quantity of outgoing light ≤ total quantity of incoming light, or for a given l

$$\forall \omega_i : \int_\Omega \rho(\omega_i, \omega_o) \cos\theta_i \, d\omega_o \leq 1$$



6. Positivität: $\rho \geq 0$ überall

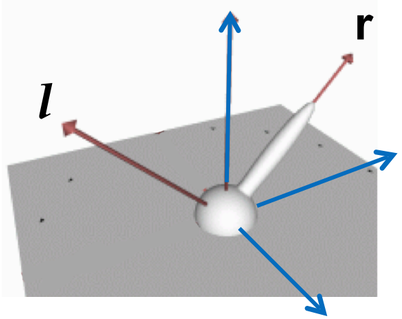
- Wähle im folgenden dieses spezielle Koordinatensystem zur Auswertung einer BRDF in einem Punkt auf der Oberfläche eines Objektes



Beispiel-BRDF: das Lafortune-Modell

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

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

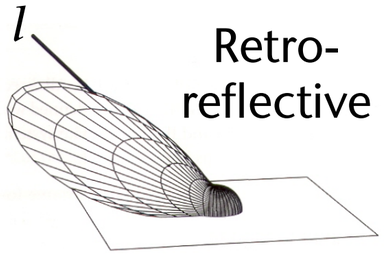


- Das Lafortune-Modell:
  - Ersetze die spezielle Matrix durch eine beliebige Matrix *C*
  - Erlaube beliebig viele "*Lobes*" (= Keulen, Lappen)
  - Zusammen:

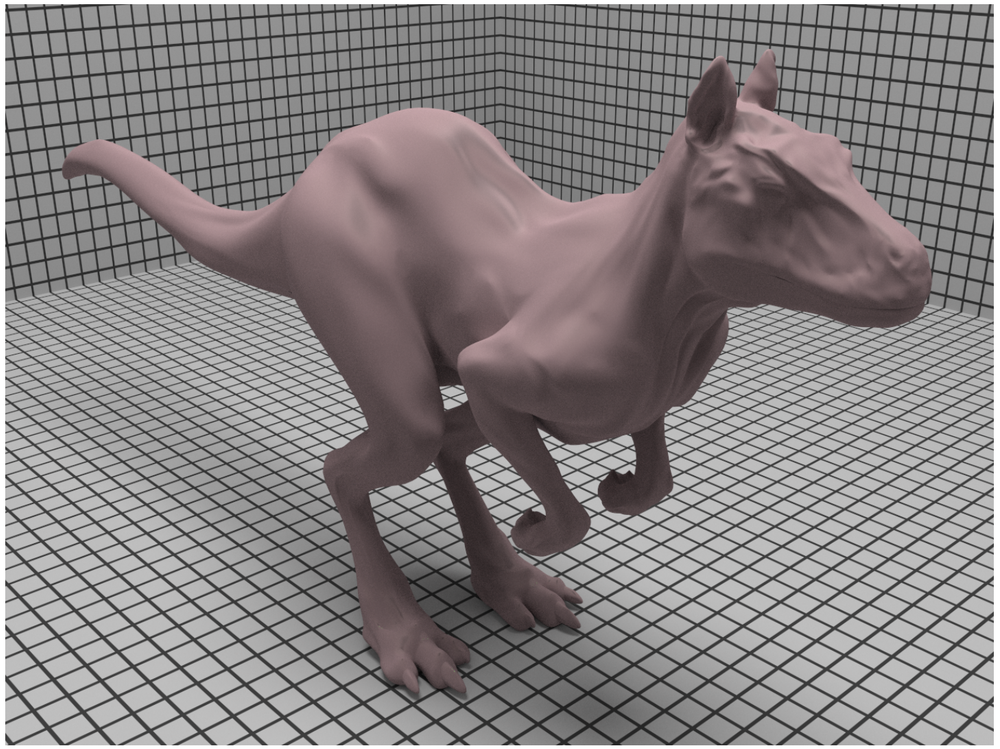
$$\rho(\boldsymbol{l}, \mathbf{e}) = k_d(\mathbf{n}\cdot\boldsymbol{l}) + \sum_{i=1}^{n} k_{s,i}(\mathbf{e}\cdot C_i\cdot\boldsymbol{l})^{p_i}$$

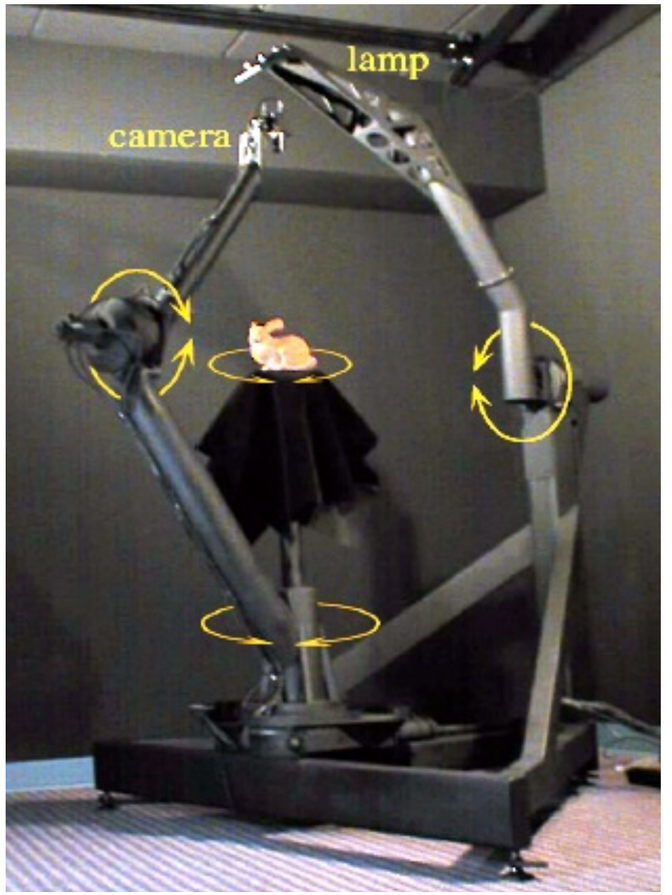
    wobei *n* = Anzahl *lobes*

- 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!)





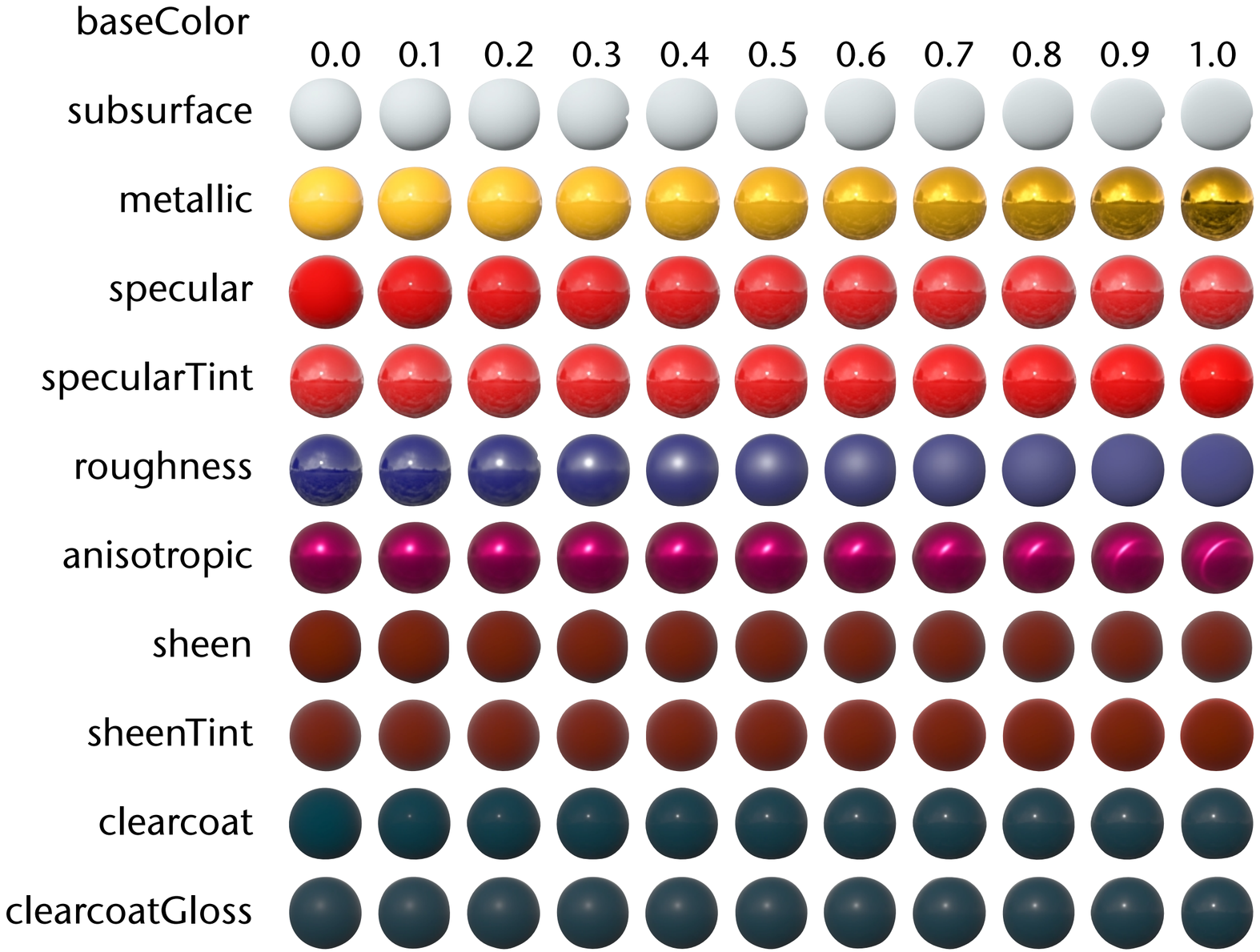
Gerendert mit Lafortune-Modell, das vermessenen Ton modelliert

Ein aktueller Standard ist die "Disney BRDF"

- Häufige alternative Bezeichnung: *metal/roughness workflow*
- Verfolgt einige Design-Prinzipien, um dadurch die Arbeit für das "Art Department" einfacher und intuitiver zu machen:
  1. Möglichst physikalisch korrekt, aber Intuitivität der Parameter hat Priorität über physikalischer Korrektheit
  2. So wenig Parameter wie möglich
  3. Möglichst alle Parameter in [0,1]
  4. Alle Kombinationen sollten möglich sein und (wenigstens) plausible Ergebnisse liefern (keine "komischen" Effekte bei Kombination bestimmter Parameterwerte)

- Im Folgenden ein paar Aspekte davon, und nur etwas oberflächlich

Die Parameter



baseColor = the surface color, usually supplied by texture maps

subsurface scattering approximation

metallic-ness (linear blend between two different models

incident specular amount

a concession for artistic control that tints incident specular towards the base color

surface roughness, controls both diffuse and specular response

degree of anisotropy, controls the aspect ratio of the specular highlight

additional grazing component

amount to tint sheen towards base color

second, special-purpose specular lobe (für lackierte Oberflächen)

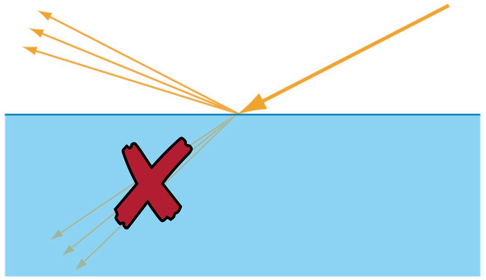
0 = "satin" appearance,
1 = "gloss" appearance

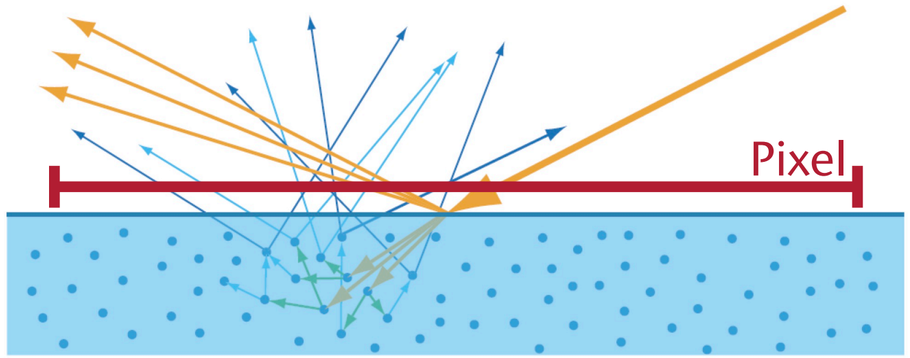
Zwei Material-Kategorien

- Alle Materialien reflektieren einen Teil des Lichtes

1. Metalle: gebrochenes Licht wird sofort vollständig absorbiert; kein diff. Term!
2. Nicht-Metalle (dielectrics): gebrochenes Licht wird (mehrfach) gestreut, teils absorbiert, teils tritt es wieder *diffus* aus
   - Bei vielen Materialien: Distanz zwischen Ein-/Austritt ≪ Pixel-Größe
   - Vorteil: man kann alles lokal am Auftreffpunkt berechnen

- Ist eigtl eine binäre Entscheidung, aber das Disney-Modell betrachtet *metallic* ∈ [0,1]

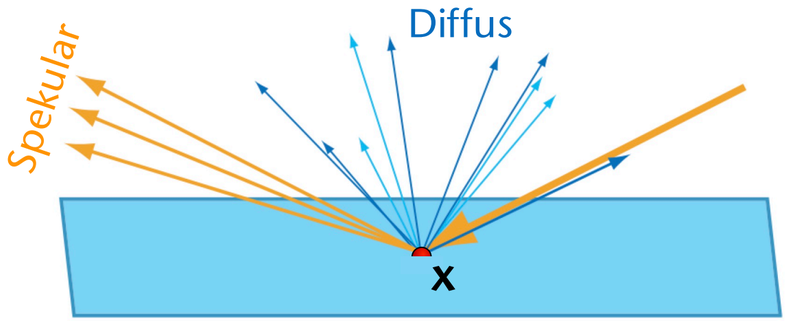




Zwei wesentliche Kategorien bei der Reflexion von Licht

- Diffuse Reflexion und spekulare Reflexion
- Die BRDF setzt sich also zusammen:

$$\rho = \rho_{\text{diff}} + \rho_{\text{spec}}$$



- Damit kann man die Reflectance Equation aufstellen:

$$L_o(\mathbf{v}) = \int_\Omega \rho(\boldsymbol{l}, \mathbf{v}) \cdot L_i(\boldsymbol{l})(\mathbf{n} \cdot \boldsymbol{l}) \, \mathrm{d}\omega_i$$

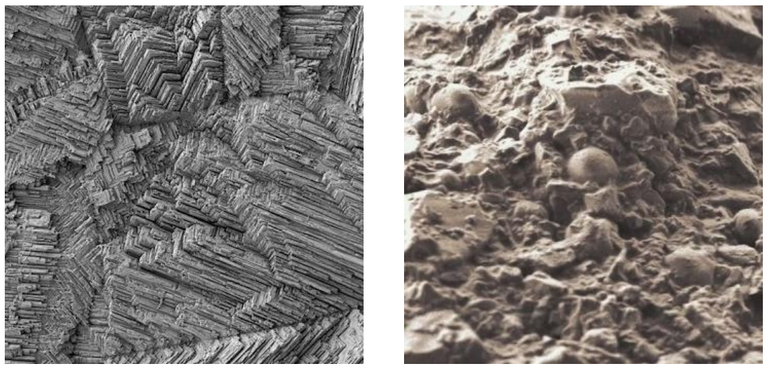
- Diffuse Reflexion ist einfach das Lambert'sche Modell:

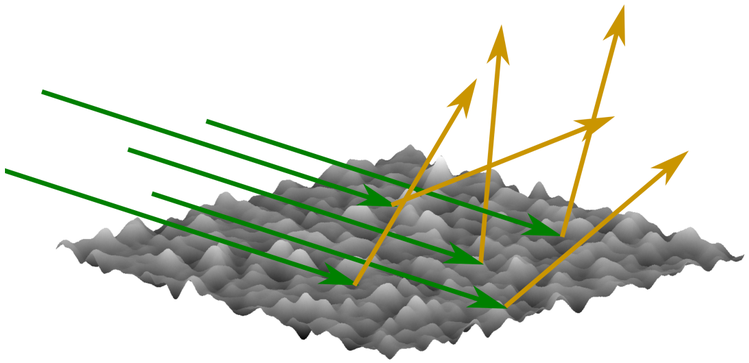
$$\rho_{\text{diff}}(\boldsymbol{l}, \mathbf{v}) = \frac{1}{\pi} C_{\text{base}}$$

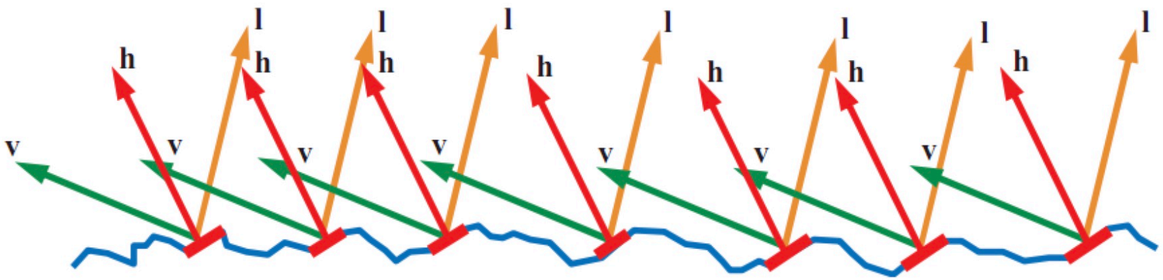
  - Beachte: $(\boldsymbol{l} \cdot \mathbf{n})$ ist nicht Teil der Lambert-BRDF! (steht schon separat in der Reflectance Equation) (π ist ein Normierungsfaktor)

Der spekulare Term (Mikrofacetten-Theorie)

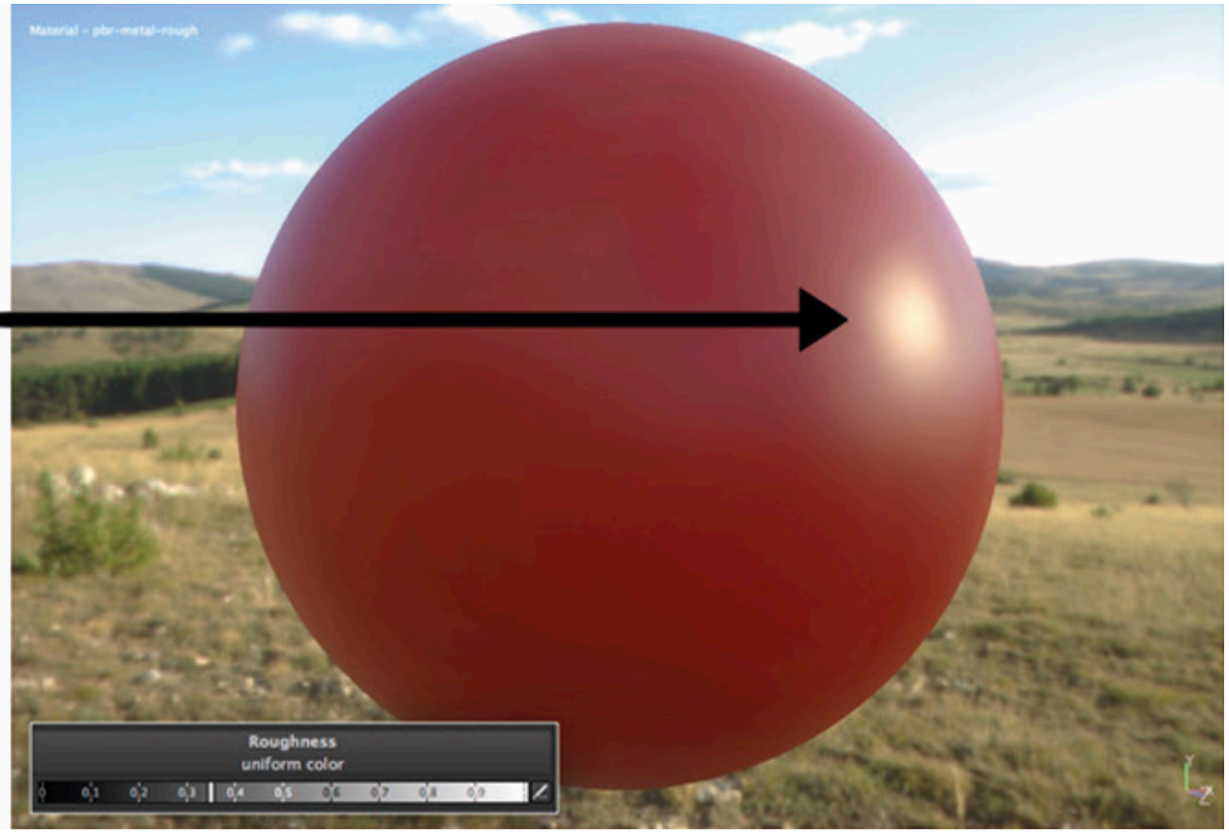
- Gängiges Modell:
  - Die allermeisten Oberflächen sind in Wahrheit nicht perfekt glatt → Mikrofacetten
  - Jede Mikrofacette ist wie perfekter Spiegel
  - Varianz der Mikrofacettennormalen, **m**, um die Normale herum hängt von Rauhigkeit (*roughness*) der Oberfläche ab
- Beachte: zu gegebenem **l** und **v** interessieren uns nur die Mikrofacetten mit **m** = **h**

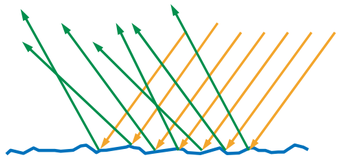


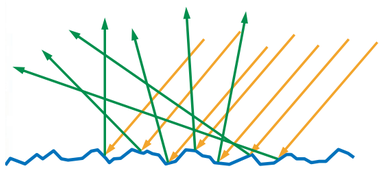




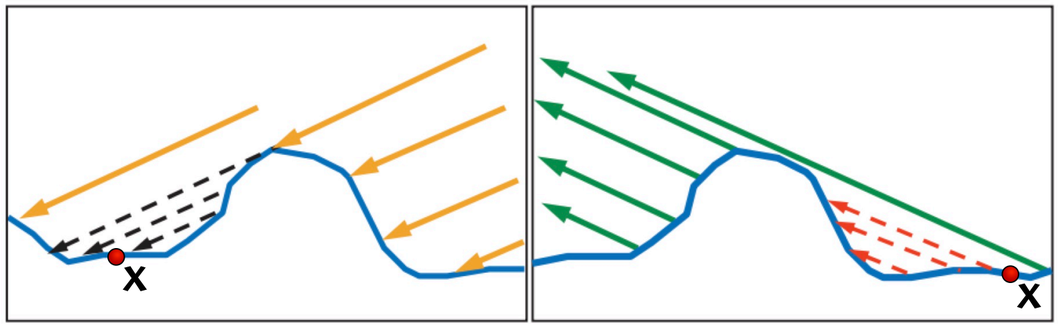
Effekt der NDF (Rauhigkeit)

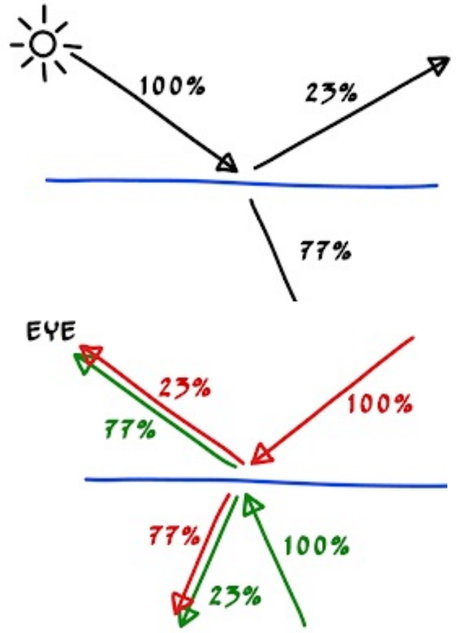






- Geometr. Effekte: Shadowing und Masking
  - Eigtl interessieren nur Facetten mit **m** = **h**
  - *Andere* Facetten blockieren eintreffendes Licht im Punkt **x** → Shadowing
  - Punkt **x** ist vom Viewpoint aus evtl durch *andere* Facetten verdeckt → Masking
  - Modelliere diese mit einer Geometry Function
- Der Fresnel-Effekt:
  - Die Anteile gebrochenen und reflektierten Lichts hängen vom Winkel ab
  - Modelliere dies mit einer Fresnel-Gleichung





M. C. Escher

Die Fresnel-Gleichung

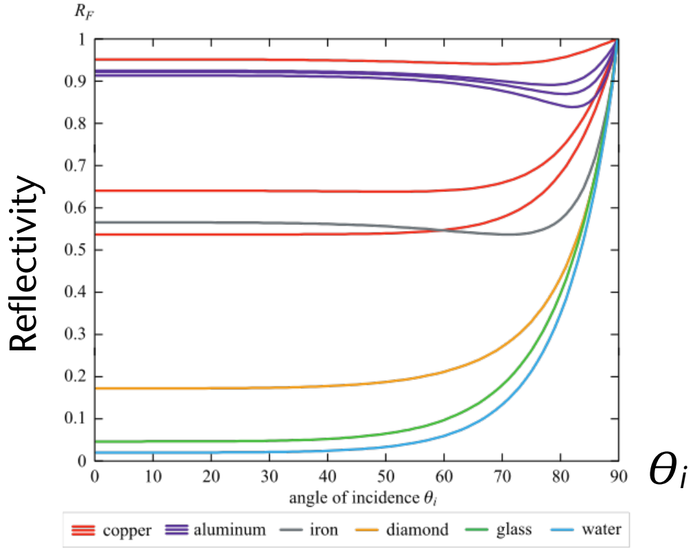
- Ist eigtl extrem komplex (details omitted)
- Eine gute Approximation von Schlick:

$$F(\boldsymbol{l}, \mathbf{n}) = F_0 + (1 - F_0)(1 - \boldsymbol{l} \cdot \mathbf{n})^5$$

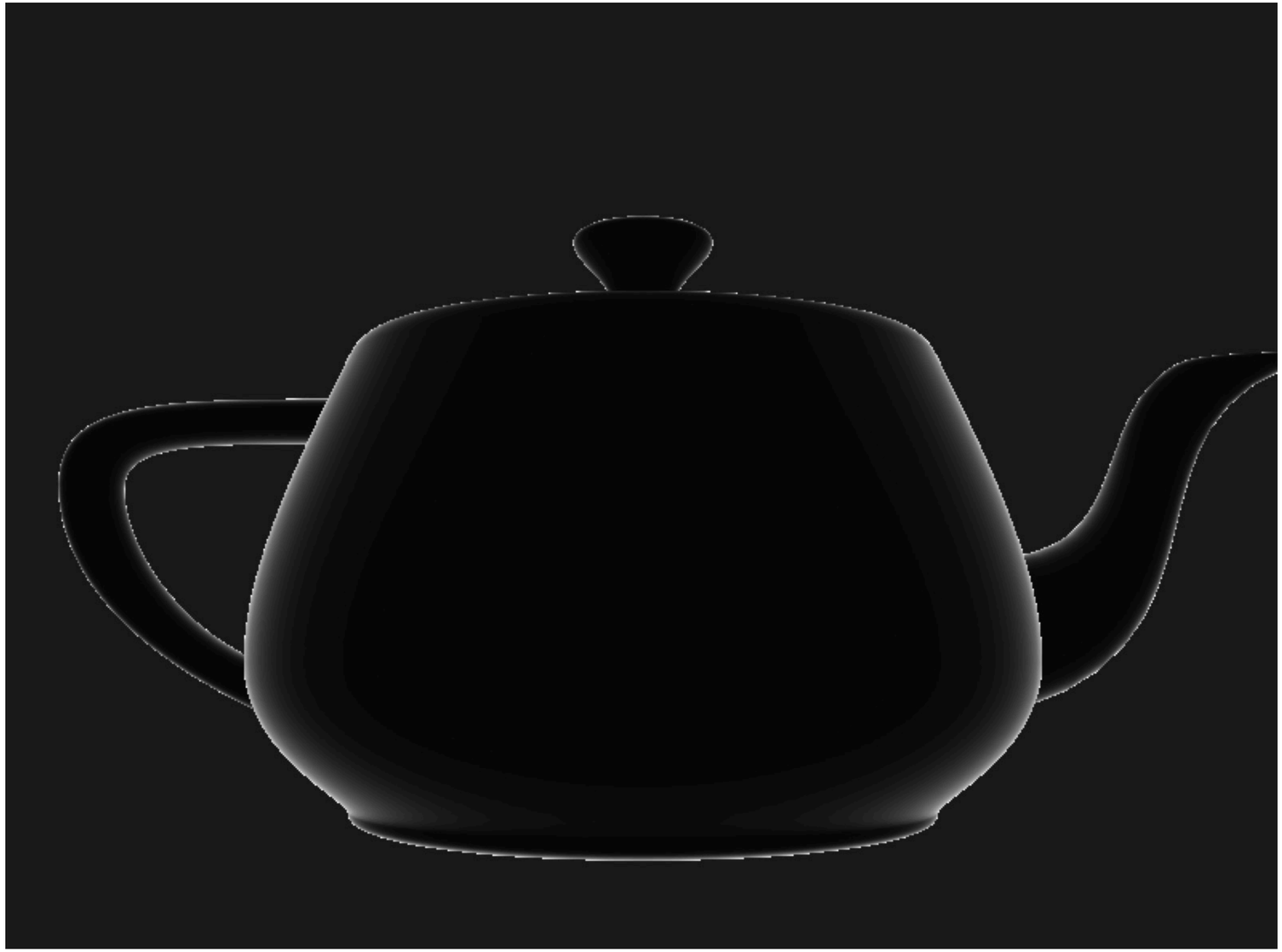
- Im Falle des Mikrofacetten-Modells interessieren uns nur Reflexionen an Mikrofacetten,
  somit muss **n** = **m** und **m** = **h**, also verwenden wir

$$F(\boldsymbol{l}, \mathbf{h}) = F_0 + (1 - F_0)(1 - \boldsymbol{l} \cdot \mathbf{h})^5$$

  - Bei Nichtmetallen ist F_0 ein Skalar, bei Metallen eine Farbe!

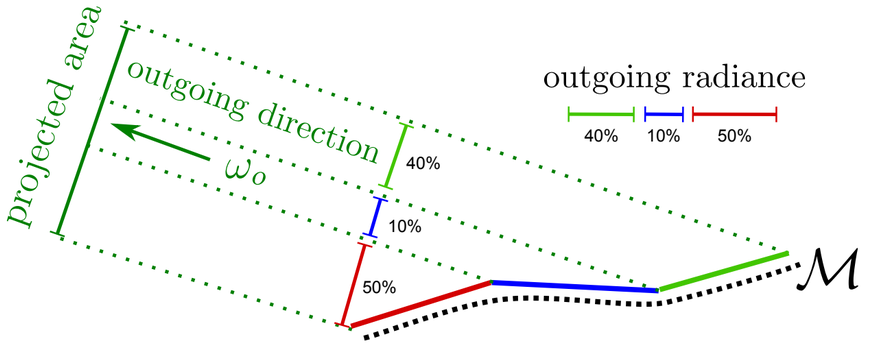


Visualisierung der Fresnel-Funktion

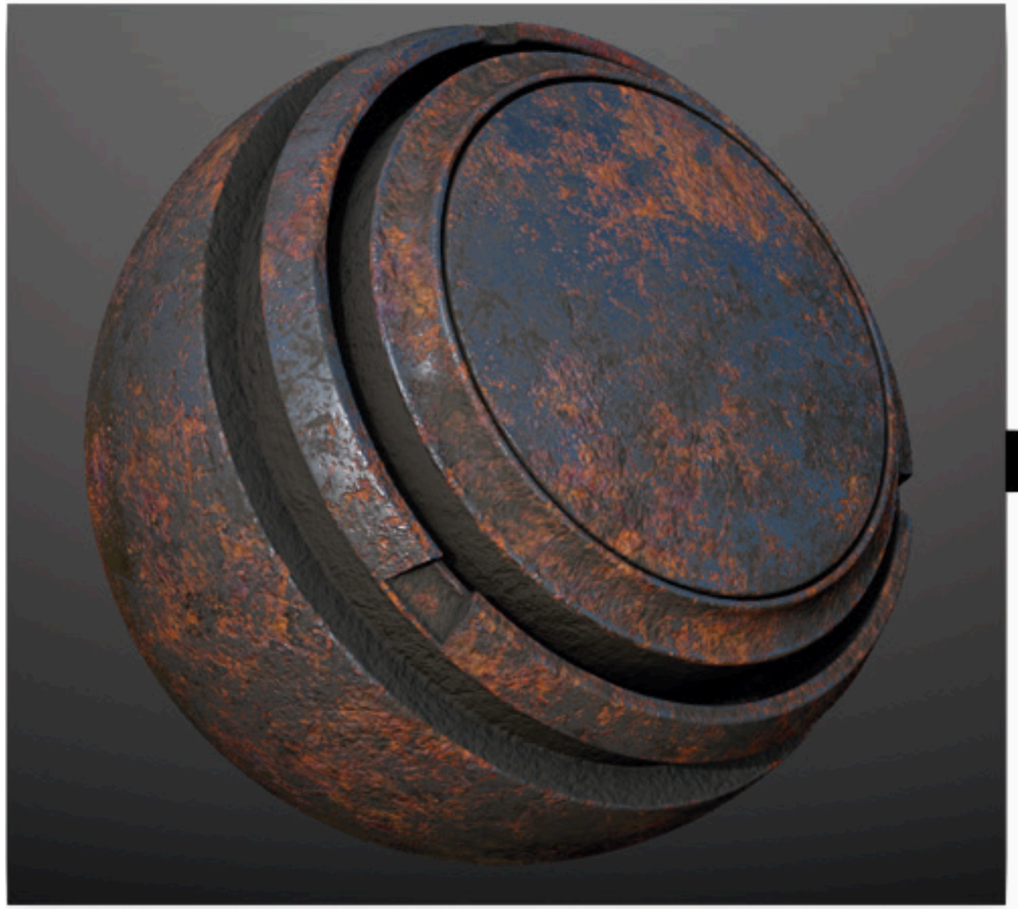


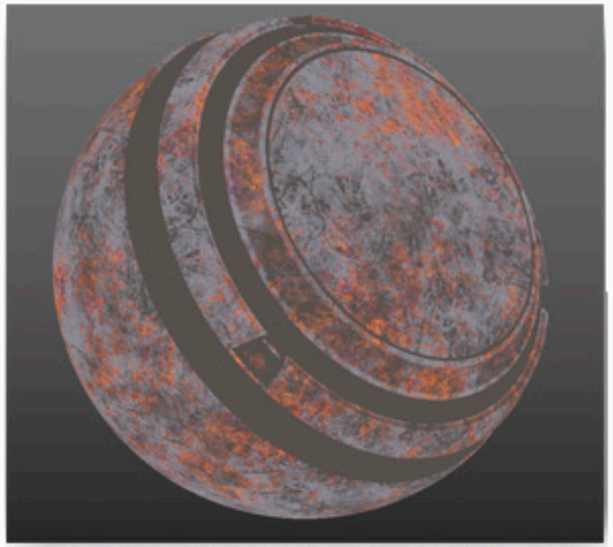
Die Normal Distribution Function (NDF)

- Streuung der Mikrofacetten-Normalen, **m**, um die Makro-Normale, **n**, hängt von Rauheit der (Makro-)Oberfläche ab
- Modelliere dies durch eine Normal Distribution Function (NDF), $D(\mathbf{m})$, die den "Anteil" der Oberfläche angibt, deren Mikrofacetten **m**=**h** haben



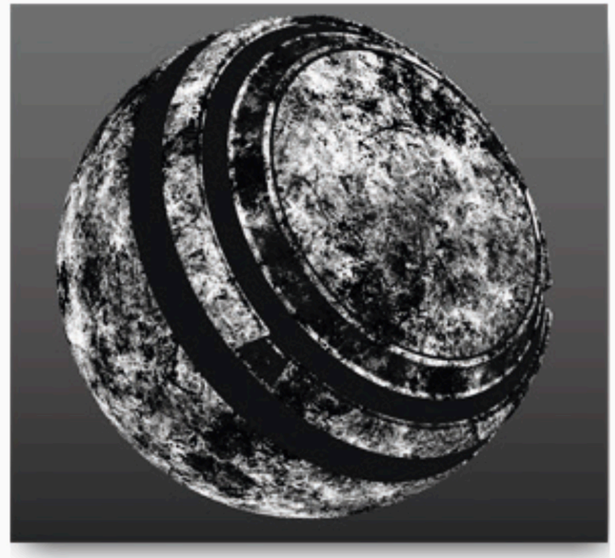
Beispiel für eine Roughness-, Farb-, und "Metallness"-Textur





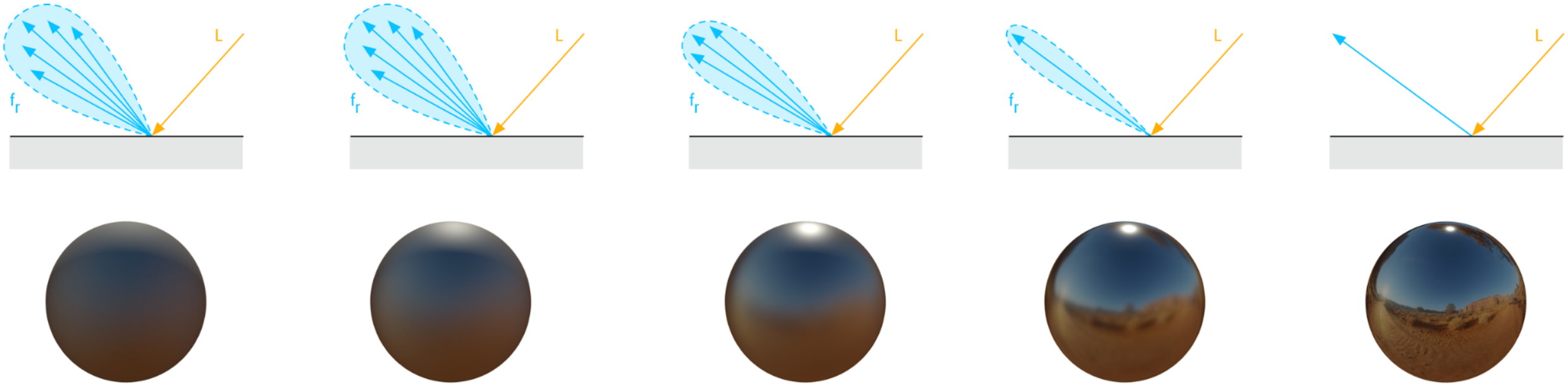
base color - RGB - interpreted as sRGB

roughness - Grayscale - interpreted as linear



metallic- Grayscale - interpreted as linear

Visualisierung des Effektes der Roughness (NDF)



Die Microfacet BRDF als Ganzes

- Besteht im Wesentlichen aus folgenden 3 Komponenten:
  1. Normal distribution function
  2. Geometrie-Term
  3. Fresnel-Term
- Im Prinzip multipliziert man alle 3 Terme und normiert das Ganze:

$$\rho_{\text{spec}}(l, \mathbf{v}) = \frac{F(l, \mathbf{h}) D(\mathbf{h}) G(l, \mathbf{v})}{4(\mathbf{n} \cdot l)(\mathbf{n} \cdot \mathbf{v})}$$

- Achtung: stelle sicher, dass Nenner ≥ 0!
  - In der Praxis: Skalarprodukte auf 0 clampen, und kleines Epsilon addieren
- Es gibt unzählige Varianten!

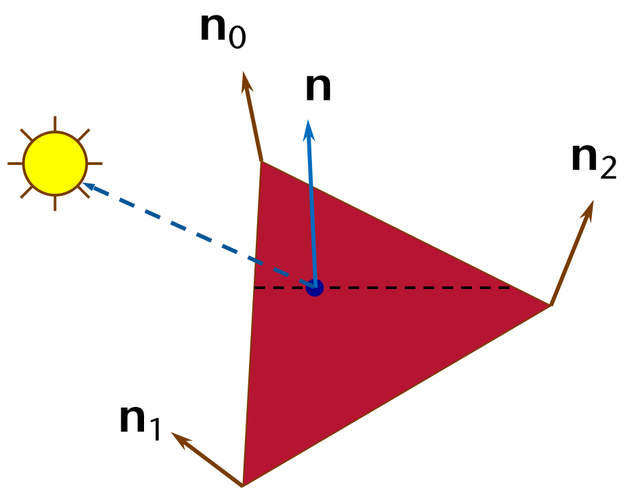
Einige Materialien funktionieren mit solchen BRDF's nicht

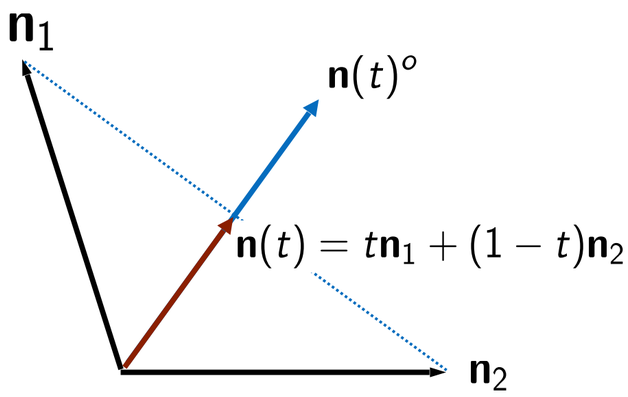
Shading-Algorithmen

- 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 auf der Oberfläche
  - Schattierungsalgorithmus berechnet aus der Intensität/Farbe an den Vertices die Farbe aller Fragmente
  - Leider: große Begriffsverwirrung! ;-(("lighting algorithm", "shading model", ...)
- Drei Möglichkeiten, wie häufig das Beleuchtungsmodell ausgewertet wird:
  - 1x pro Polygon (flat shading)
  - 1x pro Vertex (Gouraud shading)
  - 1x pro Fragment (per-pixel shading)

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

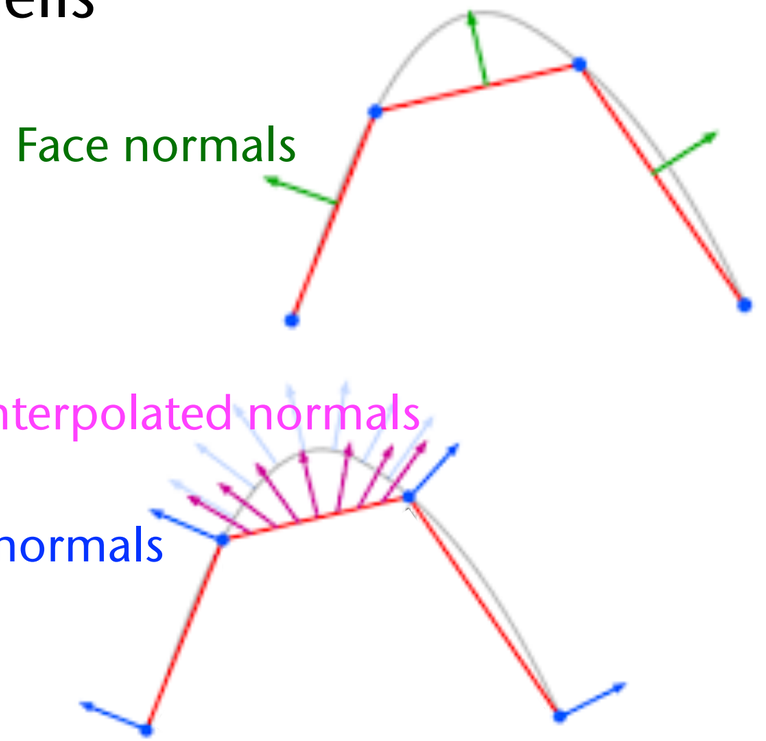
- Werte das Beleuchtungsmodell in jedem Fragment aus (ist heute Standard)
  - Interpoliere dazu die Vertex-Normalen während der Scanline-Konvertierung, wie alle anderen Attribute
- Wie interpoliert man Normalen?
  - Typischerweise: linear mit anschließender Normierung
  - Achtung: ohne Normierung bekäme man nur (sehr umständliches) Gouraud-Shading!
  - War früher sehr teuer, daher wurden viele Alternativen vorgeschlagen
  - Interpolation übernimmt heute der Rasterizer gleich mit

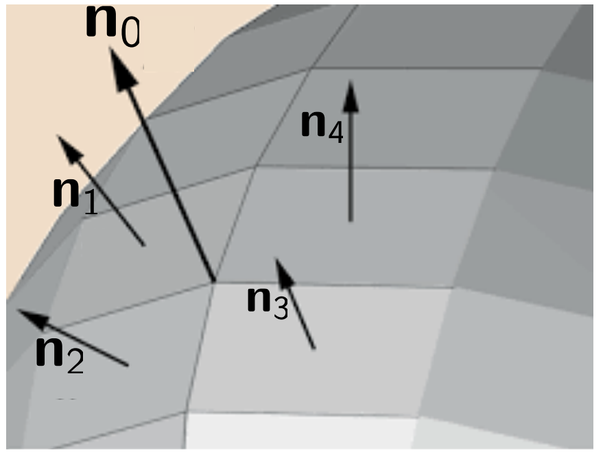




Berechnung der Normalen der Eckpunkte eines 3D-Modells

- Die Polygone bilden nur eine Annäherung an die wirkliche (unbekannte) Oberfläche eines Objektes
- An den Vertices hätte man gerne die Normale der ursprünglichen 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

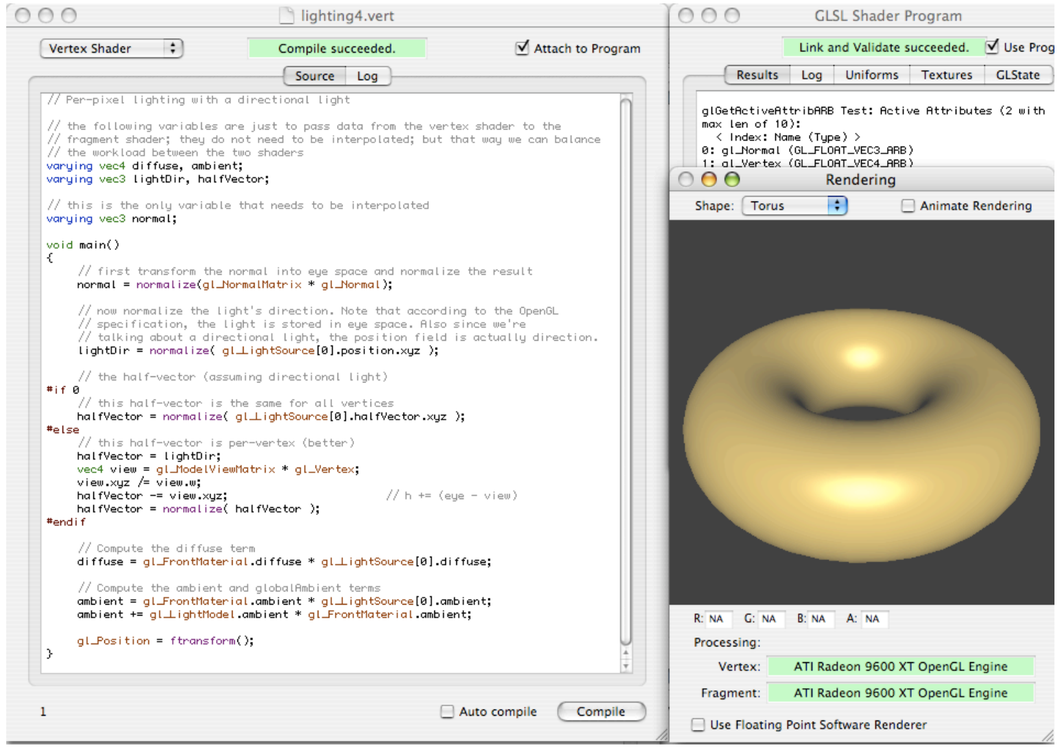




Implementierung als GLSL-Shader

FYI

1. Diffuse lighting per-vertex
2. Mit ambientem Licht
3. Mit spekularem Lichtanteil (note the Mach bands!)
4. Per-Pixel Shading



`lighting[1-3].*`

Der Vertex Shader

FYI

```
// Parameters pre-defined on host side (as uniforms)
uniform mat4 viewMatrix;      // model-to-world-to-camera matrix of the model
uniform mat4 projection;      // projection matrix (depends on aspect ratio of window)

// Attributes per vertex
in vec3 vPosition;            // xyz position of the vertex
in vec3 vNormal;              // xyz normal of the vertex

// Attributes to be calculated by this shader, for the following fragment shader
out vec3 half_vector;         // half vector (to be interpolated by rasterizer)
out vec3 normal;              // will be the normal per fragment
flat out vec3 light_dir;      // pass-through; we can use 'flat', because it's a direction


void main()
{
    vec3 light_direction = vec3(-1, 1, 1);       // we define the direction directly in camera coord

    // Transform the normal into eye space and normalize the result
    // We can use the viewing matrix here, because ShaderMaker uses only rigid transformations
    normal = ( viewMatrix * vec4(vNormal, 0.0) ).xyz;

    // transform the vertex into camera space
    vec4 vpos_cam_space = viewMatrix * vec4(vPosition, 1.0);

    // compute the half vector   (note: view vector = eye - vertex = 0 - vpos_cam_space)
    vec3 view_vector = normalize( vpos_cam_space.xyz / vpos_cam_space.w );
    light_direction = normalize( light_direction );
    half_vector = light_direction - view_vector;
    light_dir = light_direction;

    gl_Position = projection * vpos_cam_space;
}
```

Der Fragment Shader

FYI

```
// Attributes per fragment, coming in from rasterizer
in vec3 half_vector;           // half vector (interpolated by rasterizer)
in vec3 normal;                // normal per fragment
flat in vec3 light_dir;        // we can use 'flat', because it's a direction

// Output of fragment shader
out vec3 fragColor;            // fragment color (the sole output of the fragment shader)


void main()
{
    // define light source and material (normally, this would be done via uniforms)
    vec3 light_color = vec3(1, 1, 1);                              // white
    vec3 kd = vec3( 0.7, 0.7, 0 );                                 // reflection coefficients
    vec3 ks = vec3( 0, 0, 0.7 );
    float shininess = 50;

    vec3 color = vec3( 0.2, 0.2, 0.2 );                            // init color with ambient light

    // do the normalizations here, becasue the rasterizer doesn't do it
    vec3 unit_light_dir = normalize( light_dir );
    vec3 unit_normal = normalize( normal );
    vec3 unit_half_vector = normalize( half_vector );              // very necessary!

    // Compute the diffuse term
    float NdotL = max( dot(unit_normal, unit_light_dir), 0.0 );
    color +=  NdotL * kd * light_color;

    // Compute the specular term
    // Note: blue highlight with yellow base color will yield white
    float NdotH = max( dot(unit_normal, unit_half_vector), 0.0 );
    color += pow(NdotH, shininess) * ks * light_color;

    // Note: we "should" take care about RGB values > 1!

    fragColor = color;
}
```

Caveat bei Subtraktion homogener Punkte (und GLSL-Trick)

- Betrachte homogener Punkte **`v = vec4(v.xyz, v.w)`**
  - 3D-Äquivalent = **`v.xyz/v.w`**
- Subtraktion zweier Punkte **v** und **e** :
  - Naive Subtraktion der homogenen Punkte, **v - e** , ist falsch!
  - Korrekte Rechnung in 3D:

$$\frac{\mathbf{v}.xyz}{\mathbf{v}.w} - \frac{\mathbf{e}.xyz}{\mathbf{e}.w} = \frac{\mathbf{v}.xyz \cdot \mathbf{e}.w - \mathbf{e}.xyz \cdot \mathbf{v}.w}{\mathbf{v}.w \cdot \mathbf{e}.w}$$

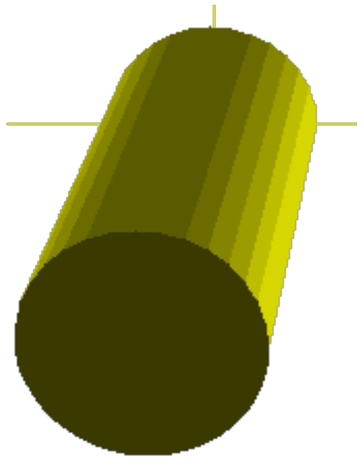
- Rechentrick: Normierung "absorbiert" Skalierung, denn $\left(\frac{\mathbf{v}}{a}\right)^0 = \mathbf{v}^0$
- Zusammen in GLSL :

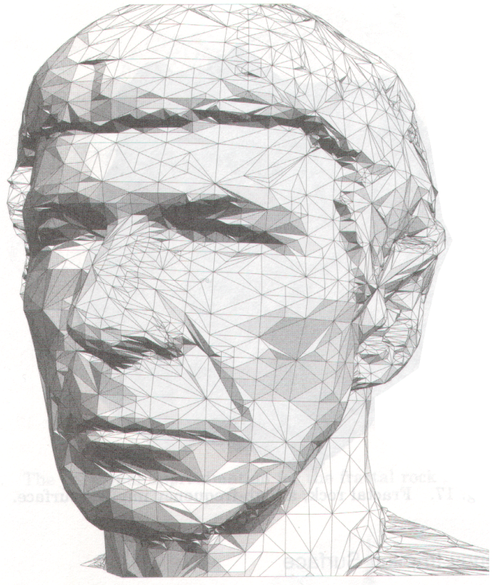
  Vektor von Pkt **e** zu Pkt **v** ≡ **`normalize(v.xyz*e.w - e.xyz*v.w)`**

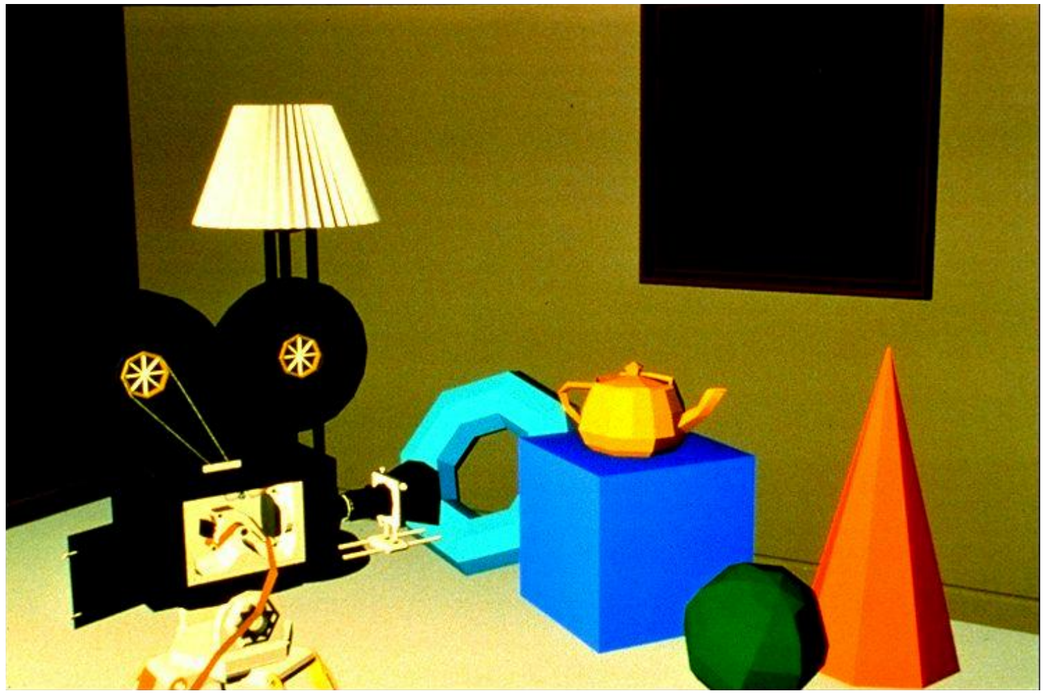
- Vorteil: kein Division-by-Zero-Fehler, falls v.w oder e.w = 0 !

Flat Shading (Konstante Beleuchtung) FYI

- Simpelster Shading-Algo: jedes Polygon erhält einen einheitlichen Farbwert für alle dessen Fragmente
  - Werte dazu das Beleuchtungsmodell (Phong, Lafortune, ...) an irgend einem Vertex des Polygons aus



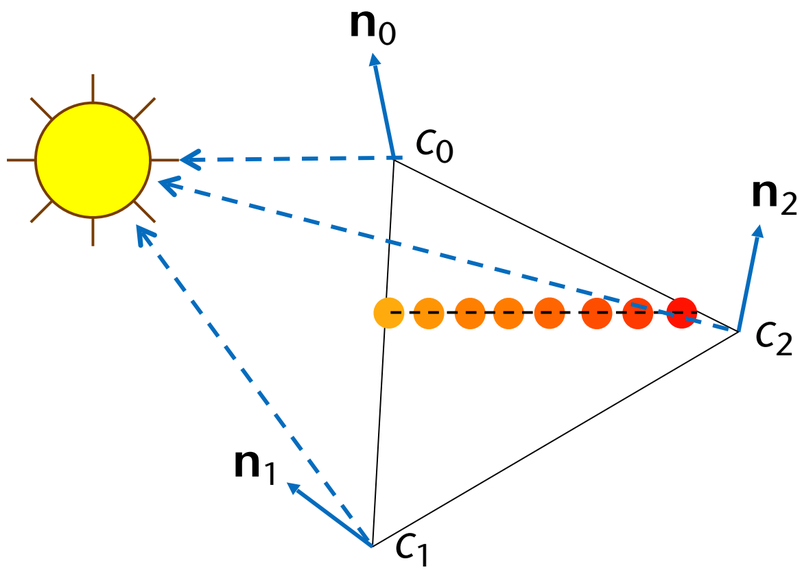




Pixar "Shutterbug"

Gouraud-Shading

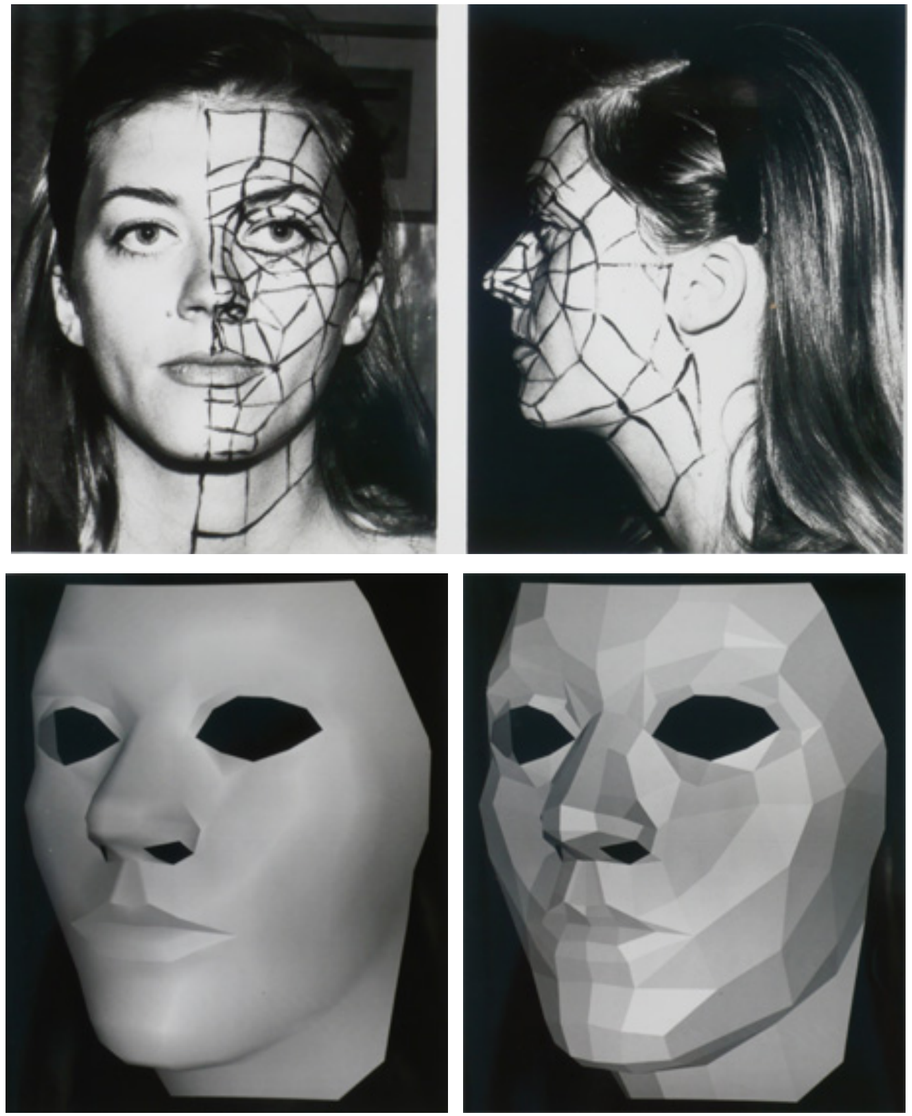
- Werte das Beleuchtungsmodell (Phong, Lafortune, ...) an allen 3 Vertices des Dreiecks aus, interpoliere linear dazwischen während der Scanline-Konvertierung



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

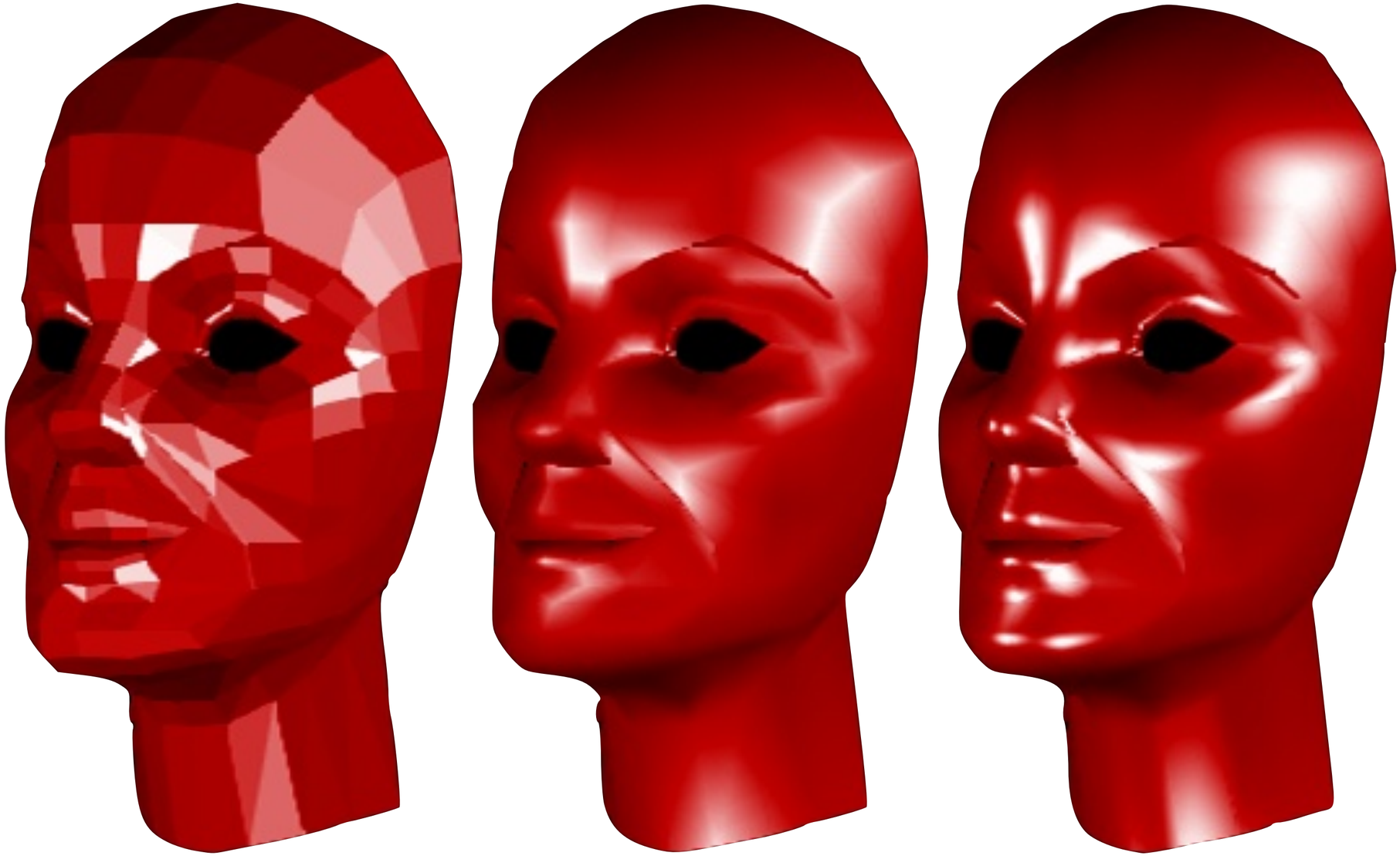
FYI

First Gouraud-shaded images, of Sylvie Gouraud



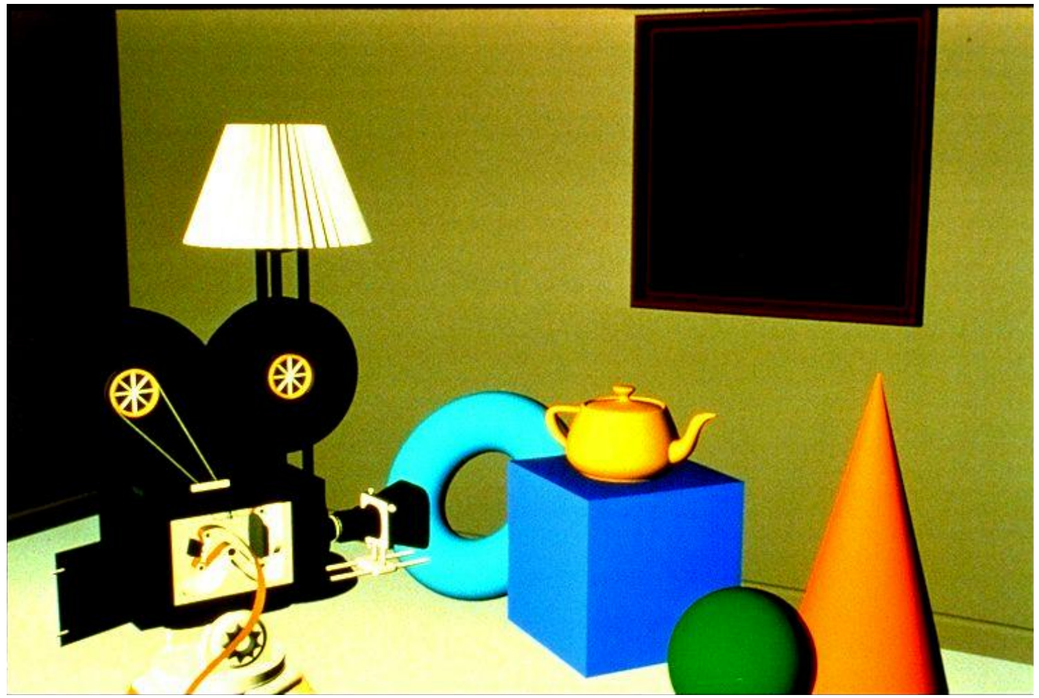
Vergleich zwischen Flat-, Gouraud- und Per-Pixel-Shading

FYI

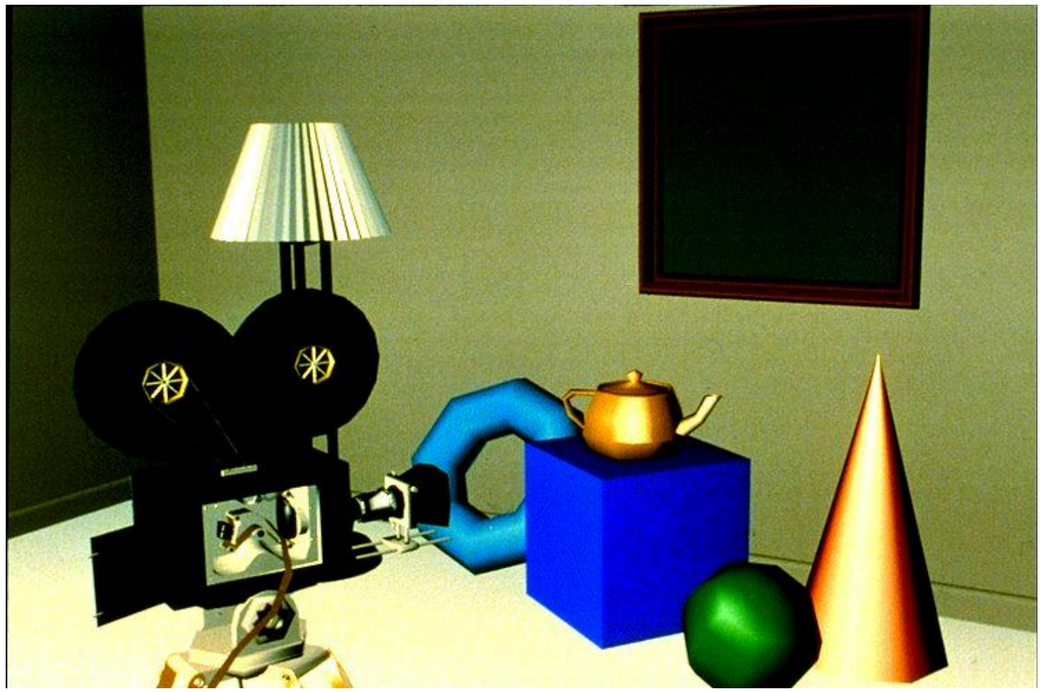


Vergleiche

FYI

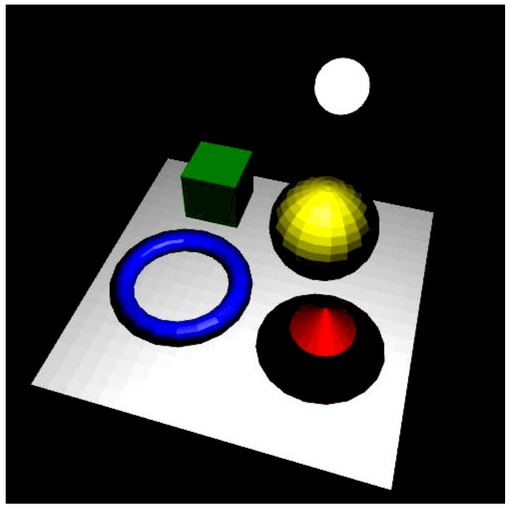


Gouraud-Shading mit
rein diffusem Beleuchtungsmodell

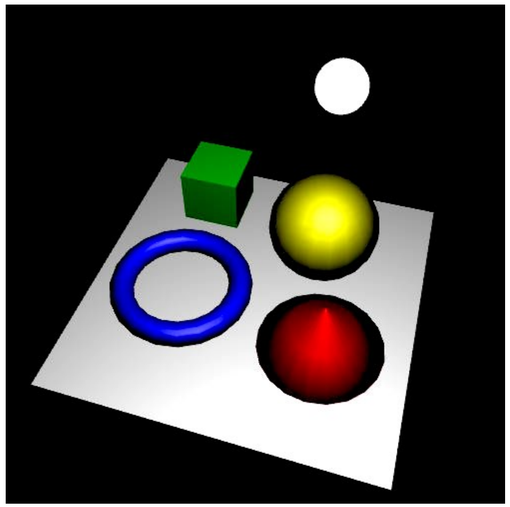


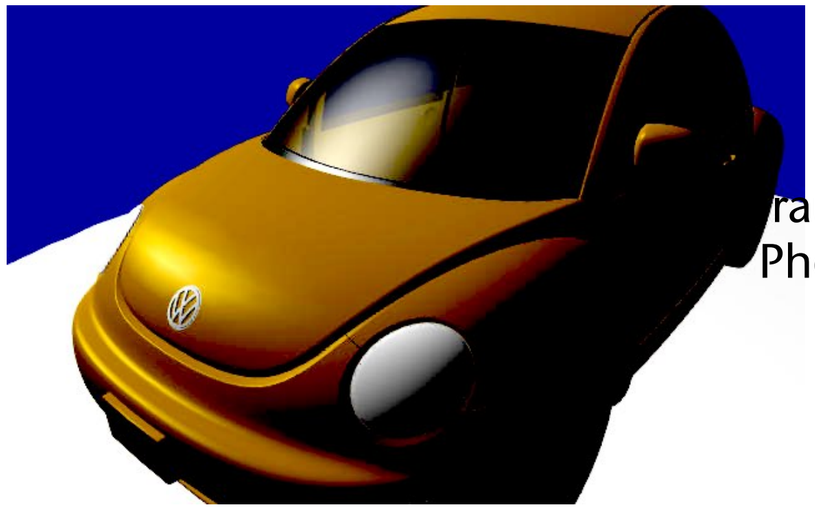
Gouraud-Shading mit Phong-Modell

FYI



at-Shading mit
Phong-Modell



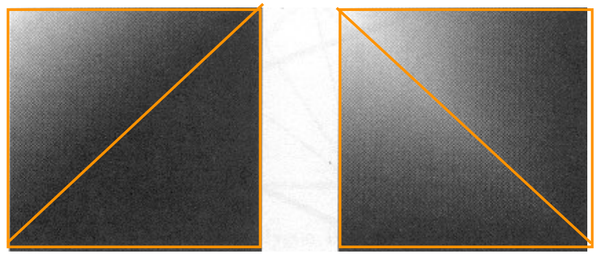


raud-Shading mit
Phong-Modell

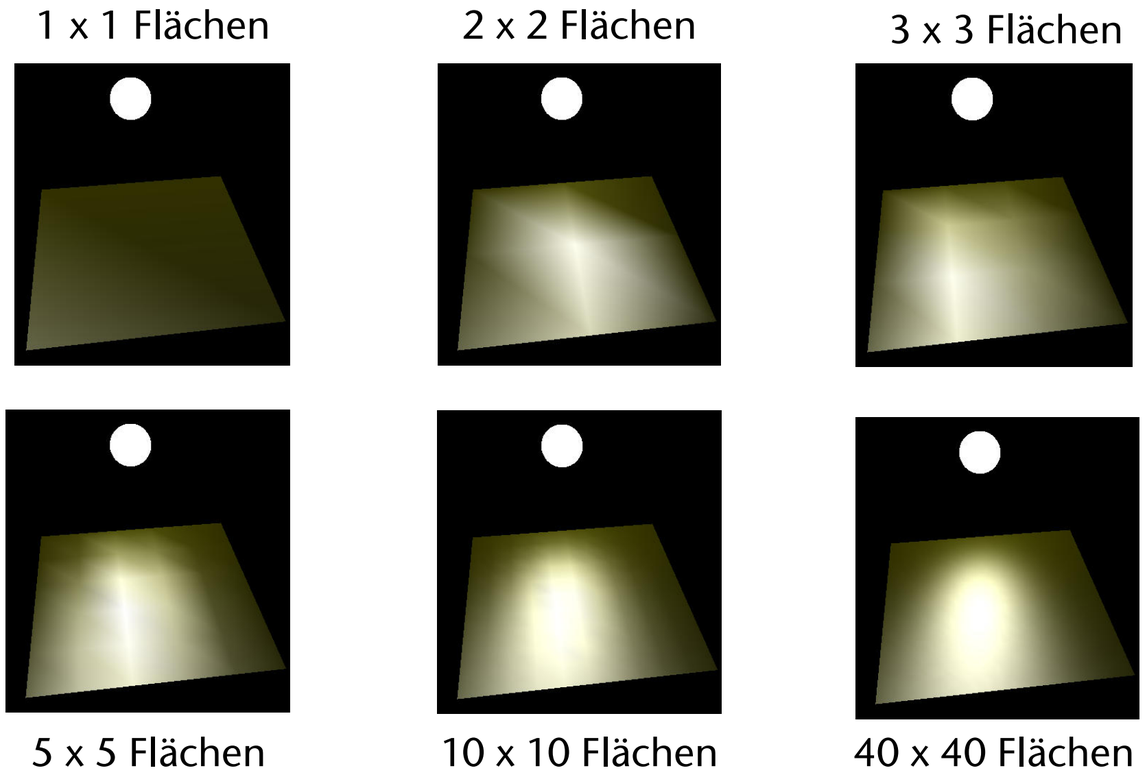
Einfluß der Triangulierung

FYI

1. Orientierung:

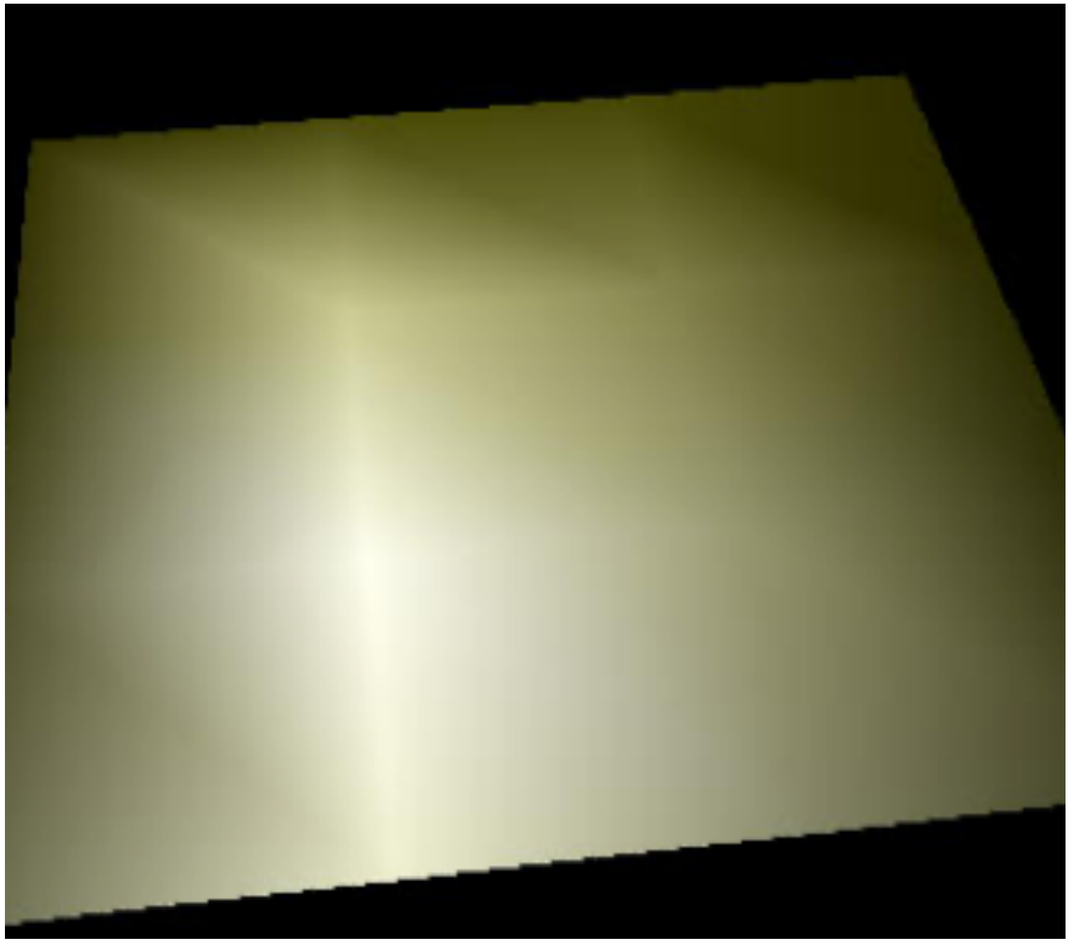


3. Hier gilt: "size matters"



Was fällt (sonst noch) auf?

FYI

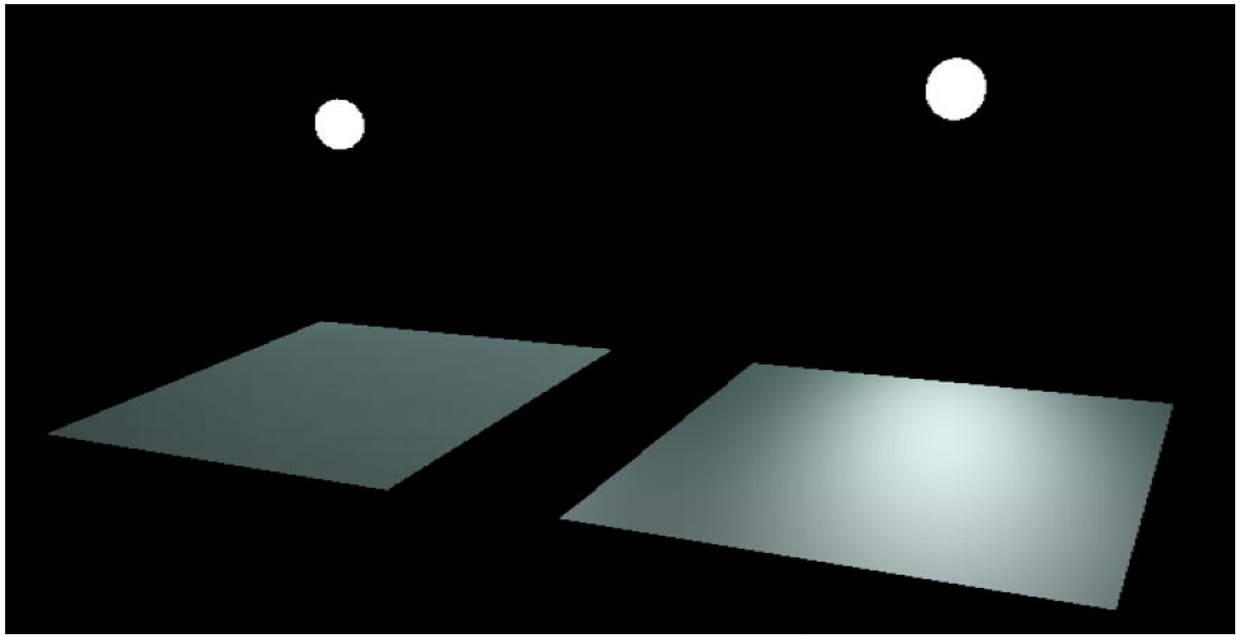


https://www.menti.com/rs2rt7mgjb

Weiteres Problem beim Gouraud-Shading

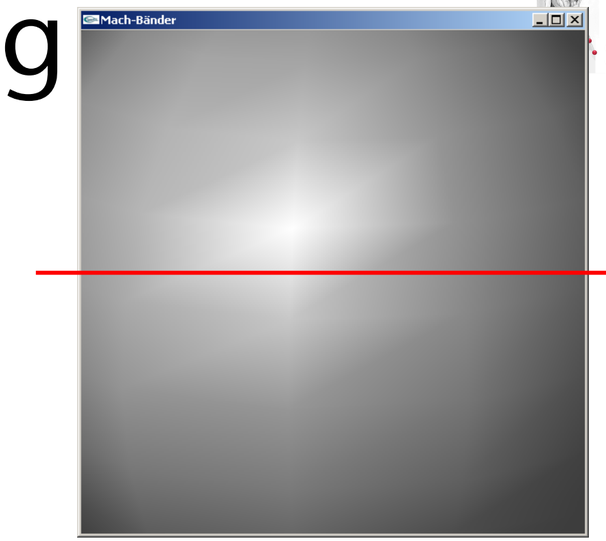
FYI

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

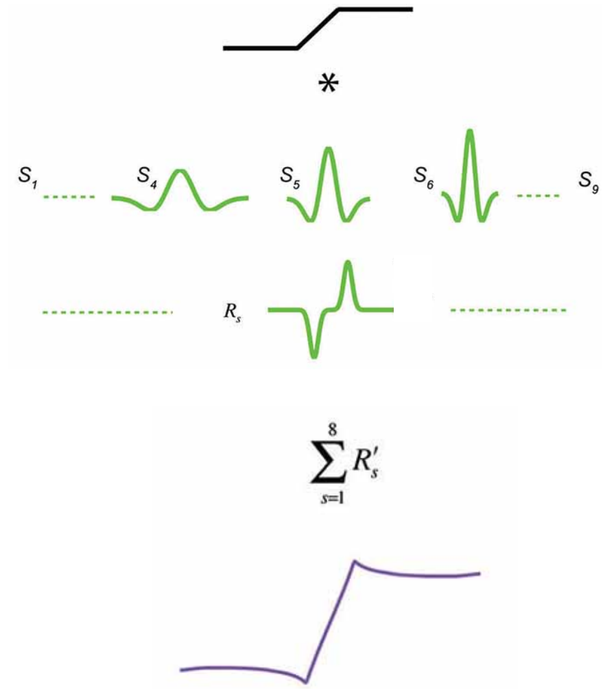


Gouraud-Shading Ideales Shading

Mach-Bänder beim Gouraud-Shading

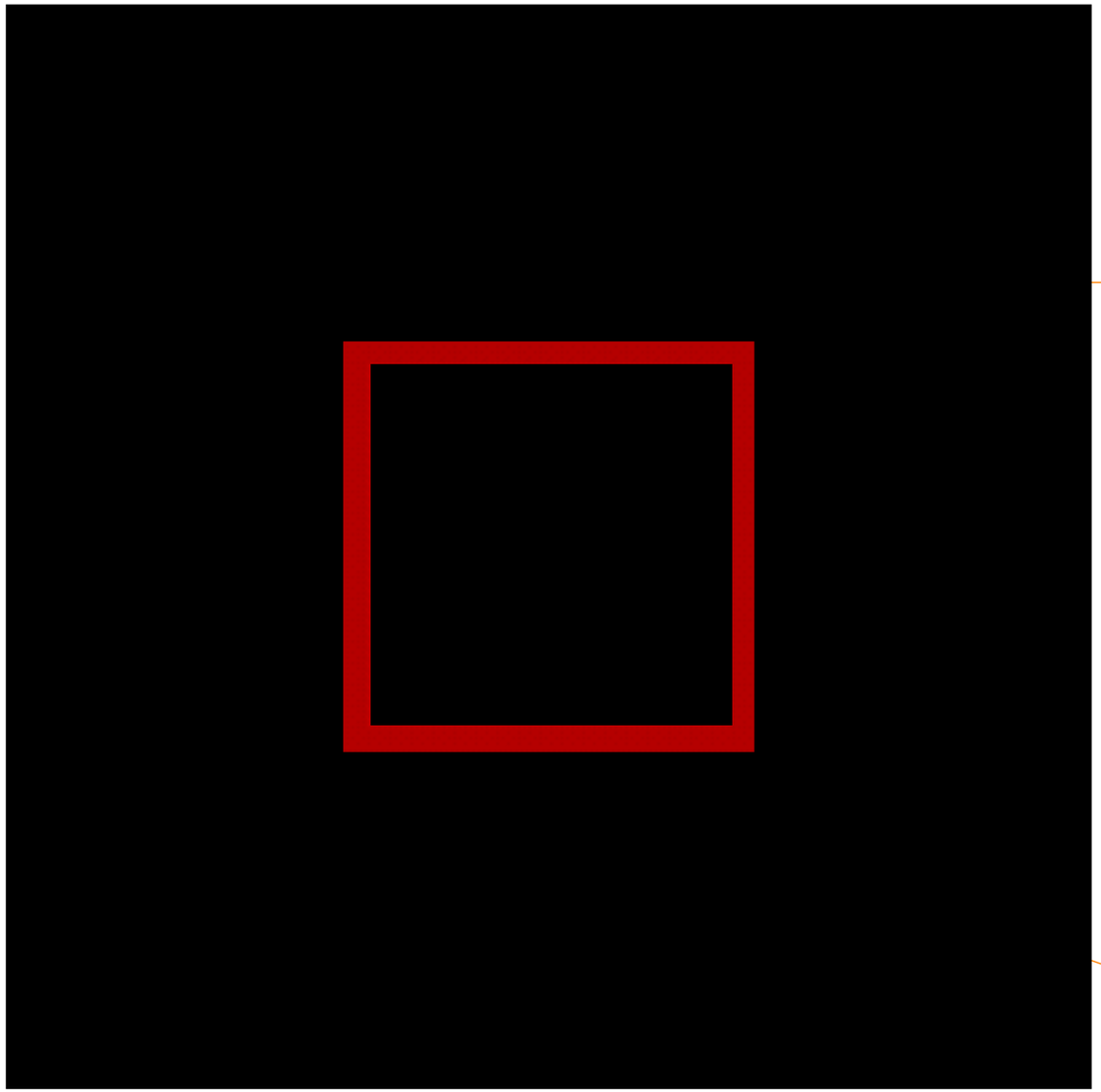


- Problem: die lineare Interpolation der "Lichtwerte"
  - Diese Interpolation ist C^0-stetig, aber *nicht* C^1-stetig!
- Eine aktuelle Hypothese [F. Kingdom, 2014]:
  - Es gibt Neuronen, die Kantendetektoren verschiedener "Breite" implementieren
  - Wahrgenommene Intensität = physikalische Intensität + Kantensignale
  - Resultat: Mach-Bänder bei linearer Interpolation



Extremes Beispiel (Chevreul Illusion)

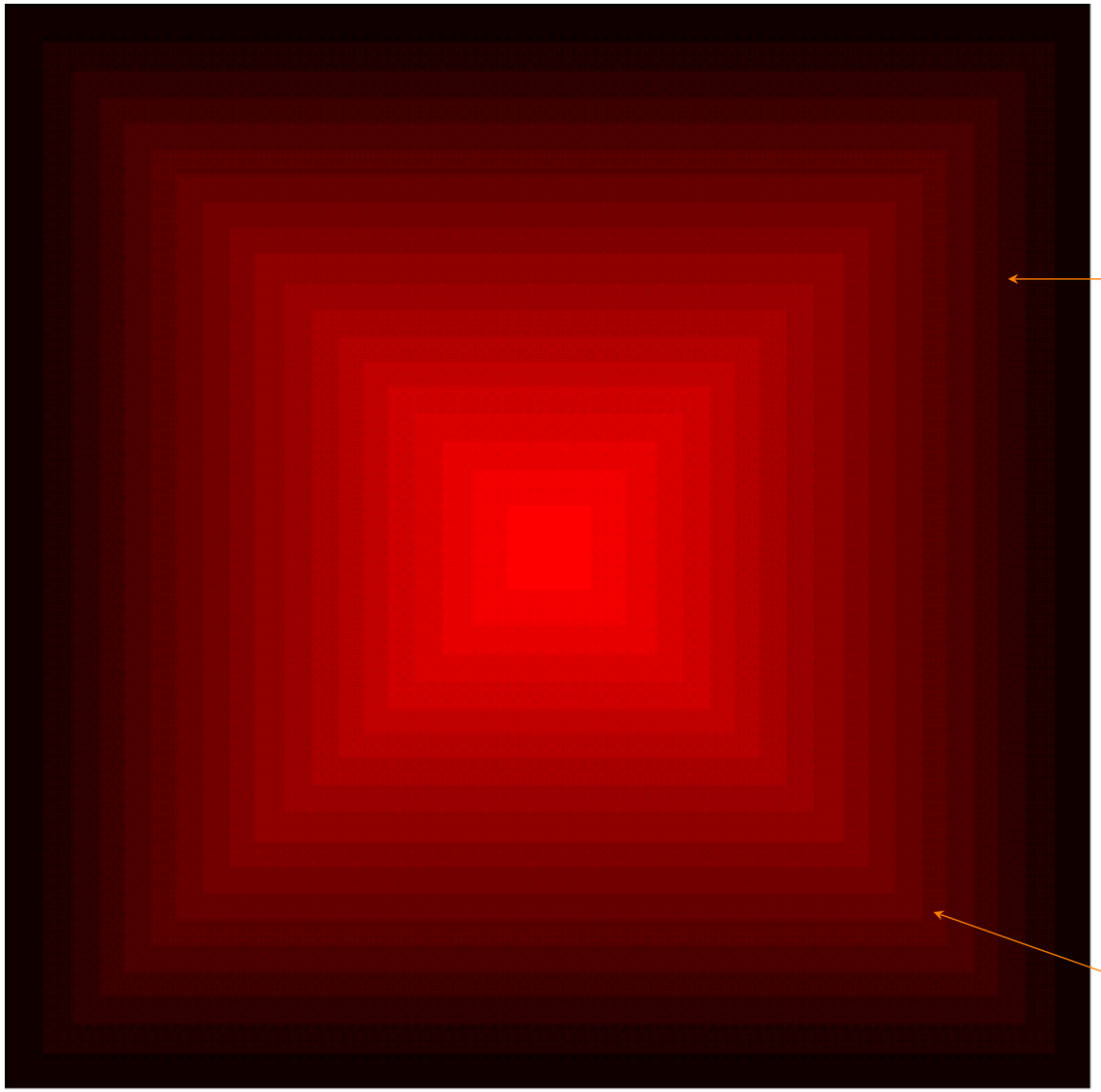
Die meisten Hypothesen zur Erklärung der Mach-Bänder erklären auch diese Illusion



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

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

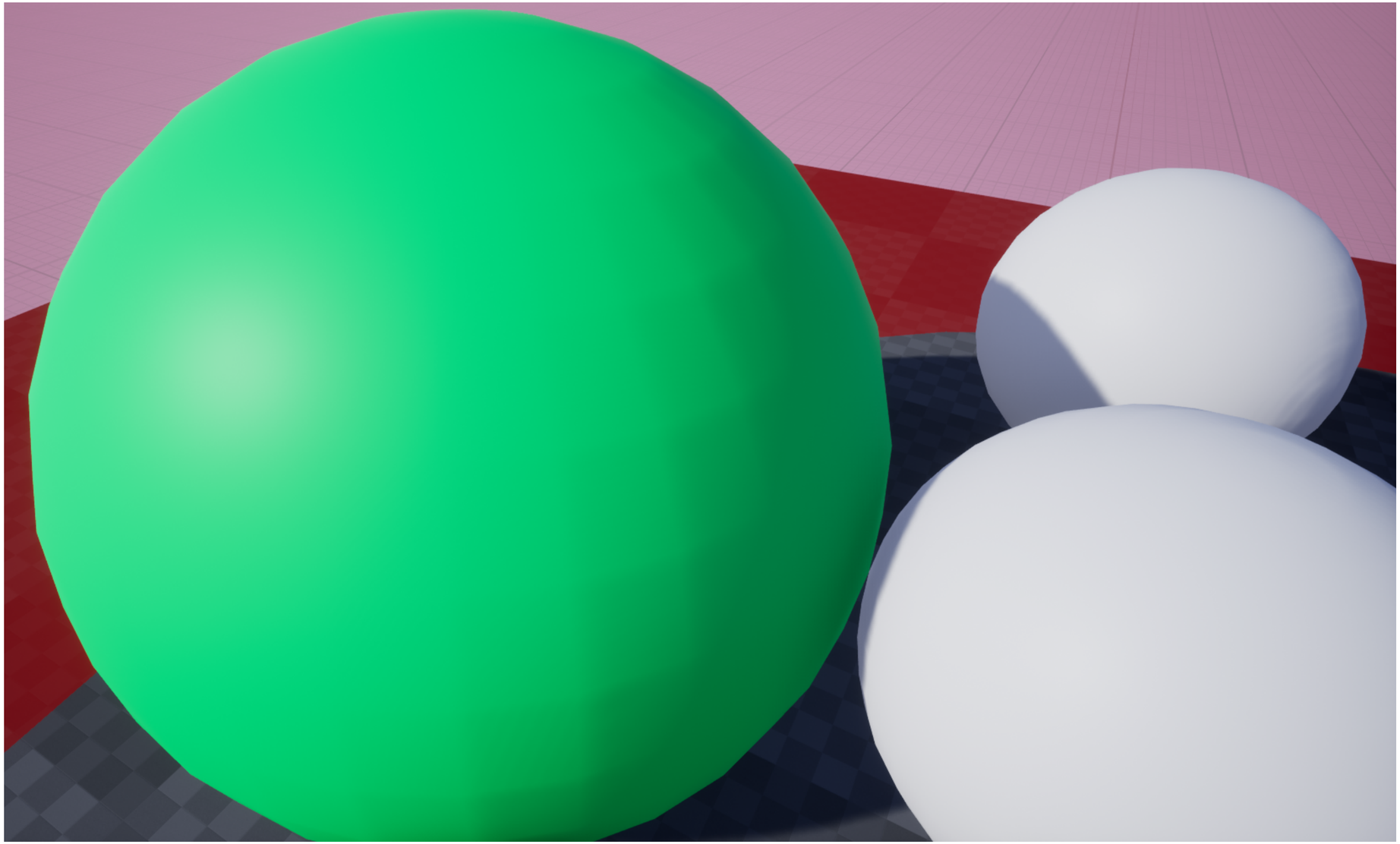
Extremes Beispiel



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 für die Artefakte, die durch Mach-Banding entstehen



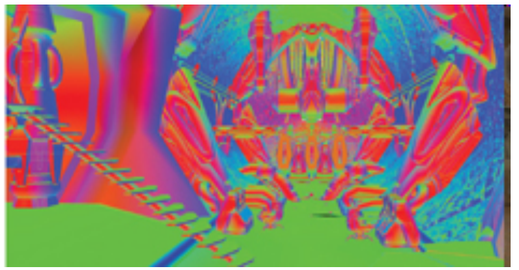
Deferred Shading / Deferred Lighting

- 2-Pass-Rendering

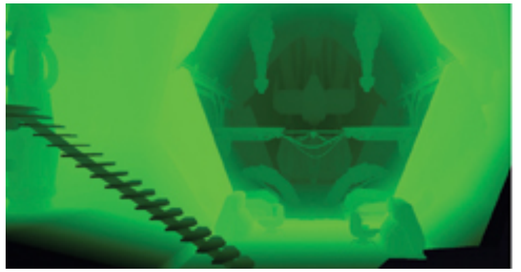
1. Geometry-only: rendere Geometrie, ohne Lighting/ Shading, speichere statt dessen alle für das Lighting notwendigen Attribute in einem "G-Buffer" (= Satz von user-defined "Frame Buffers" für die notwendigen Attribute/Daten)
2. Lighting-only:
    1. Setze Lichtquellen
    2. Rendere 1 großes Quad (um pro Pixel den Fragment-Shader 1x zu aktivieren)
    3. Lese im Fragment-Shader alle Werte aus dem G-Buffer
    4. Werte im Fragment-Shader das Lighting-Modell aus & schreibe in Color-Buffer

Fragment Colors

Normals

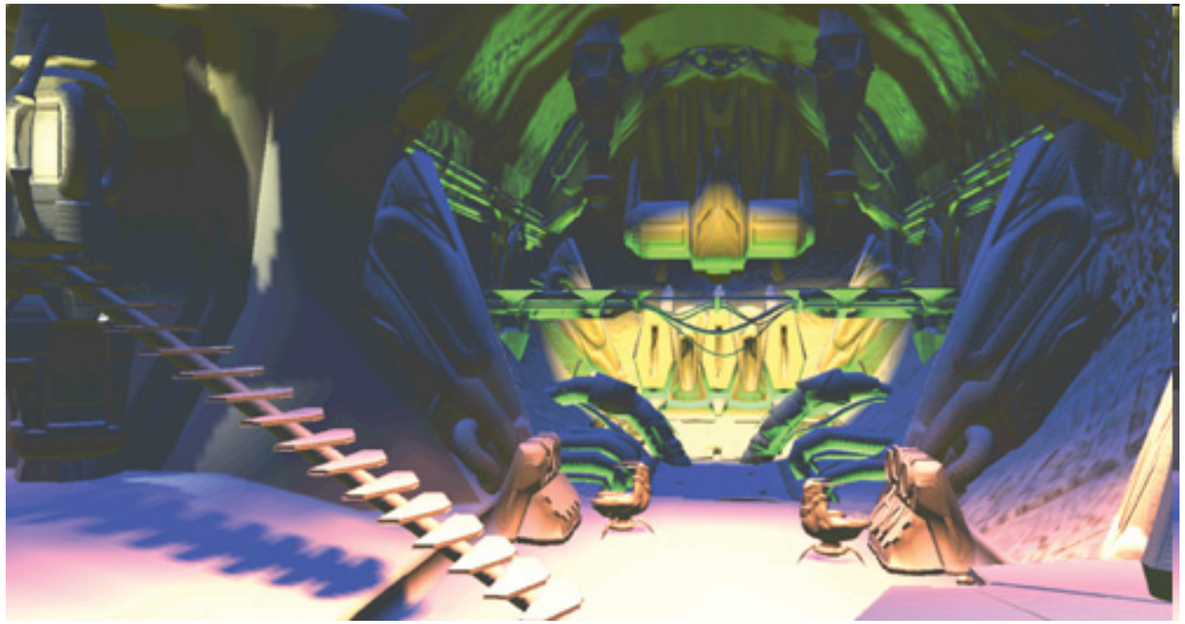


Depth



Edge Weight

- Fertige Szene:



- Vorteile: vermeidet aufwendiges Shading von Fragmenten, die durch Overdraw wieder überschrieben werden
- Nachteil: benötigt mehr Framebuffer-Speicher
- Frage: Was ist mit der Bandbreite?

[GPU Gems 3, Kapitel 19, http://developer.nvidia.com/object/gpu-gems-3.html]

Weiteres Beispiel, wo diese Technik angewendet wurde

S.T.A.L.K.E.R: Clear Skies

- Weitere Vorteile:
  - Einige Rendering-Techniken benötigen sowieso einen ersten Z-Only-Pass (z.B. Shadow Volumes), also kann man gleich etwas mehr bei diesem Pass im Buffer speichern
  - Die Kosten für Lighting sind unabhängig von der Objekt-Komplexität (= Anzahl Vertices)

Single-sided und *two-sided lighting*

FYI

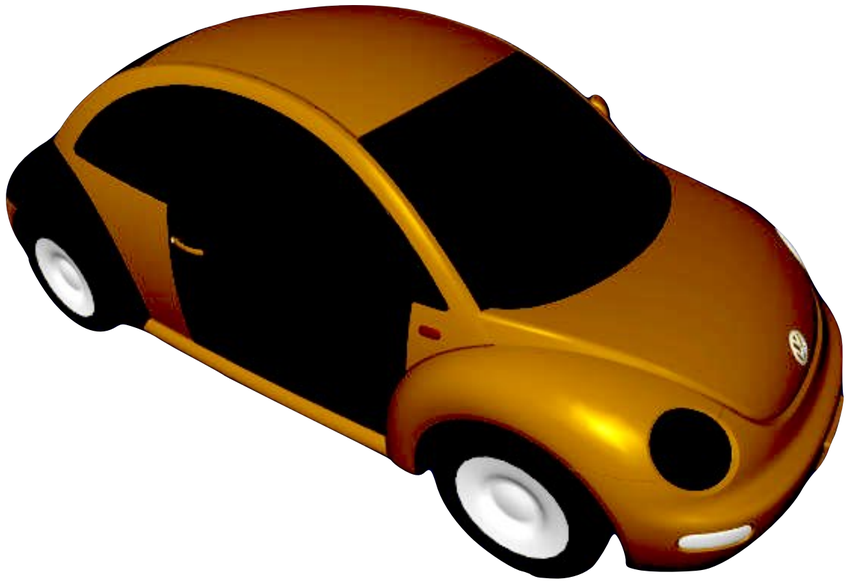
- Normalerweise haben wir definiert (single-sided lighting):

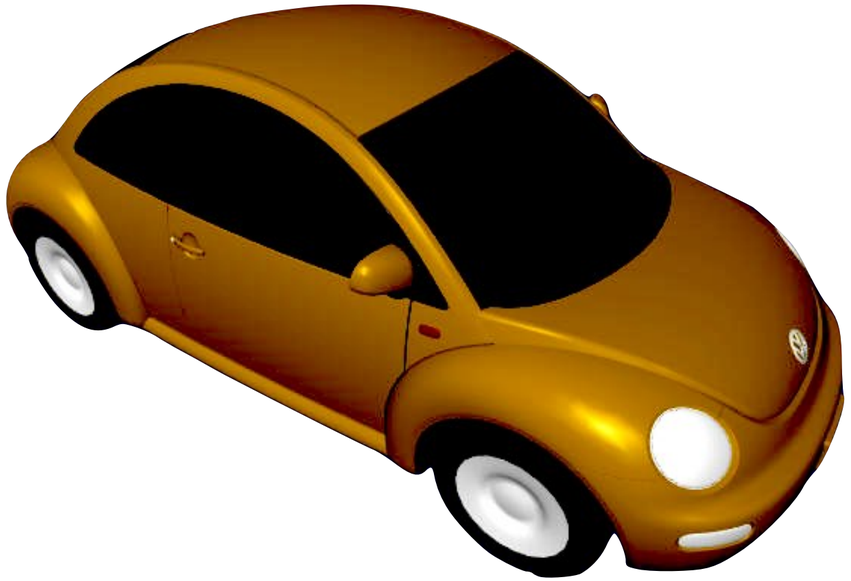
$$\mathbf{n} \cdot \boldsymbol{l} := \max(0, \mathbf{n} \cdot \boldsymbol{l})$$

- Manchmal zeigen die Normalen in die "falsche" Richtung
- Lösung: two-sided lighting, d.h., wir definieren

$$\mathbf{n} \cdot \boldsymbol{l} := \begin{cases} n \cdot \boldsymbol{l} & \text{, falls Skalarprod. positiv} \\ -n \cdot \boldsymbol{l} & \text{, falls Skalarprod. negativ} \end{cases}$$

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





Atmosphärische Dämpfung (Fog)

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

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

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

mit $z_b \leq z \leq z_f$