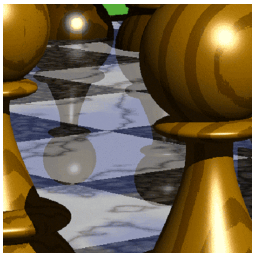
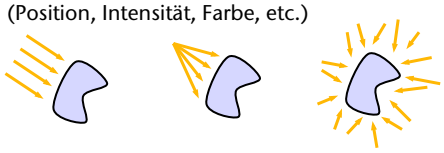
Computer-Graphik 1
Lighting & Shading



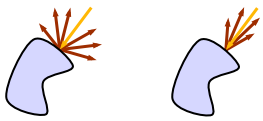
G. Zachmann
Clausthal University, Germany
cg.in.tu-clausthal.de

Beleuchtungsmodelle *(lighting models)*

- Definition "Beleuchtungsmodell": Vorschrift zur Berechnung der Farb- und Helligkeitswerte von Punkten auf der Oberfläche von Objekten
  - Grundlage sind physikalische Gesetze
  - Modelliert werden Einflüsse von:
    1. Lichtquellen (Position, Intensität, Farbe, etc.)



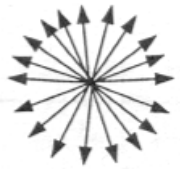
    2. Objektoberfläche (Geometrie, Reflexionseigenschaften)



- Für Echtzeitanwendungen verwendet man sehr einfache Modelle

Lichtquellen

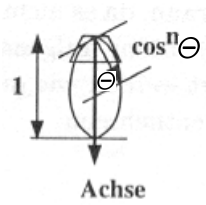
- Punktlichtquelle (*point light*): strahlt in alle Richtungen gleichmäßig ab
  - Wird eindeutig charakterisiert durch
    1. Position &
    2. $I(\lambda)$ = abgestrahltes Spektrum
       = Intensität abhänig von der Wellenlänge



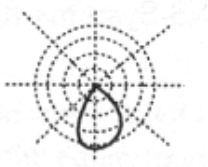
- Richtungslichtquelle *(directional light)*: jeder Punkt im Raum wird aus derselben Richtung bestrahlt
  - Charakterisiert durch Richtung & $I(\lambda)$
  - Beispiel: Sonne

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

$$I(\lambda) = I_0(\lambda)\cos^n \Theta$$



  - Charakterisierung durch: Position, Richtung (Kegelachse), Exponent (Öffnungswinkel), $I(\lambda)$
- Goniometrische Lichtquelle: Abstrahlcharakteristik wird per Tabelle beschrieben. Zur Ermittlung von $I(\lambda)$ muß evtl. zwischen Tabelleneinträgen interpoliert werden



Lokale Beleuchtungsmodelle

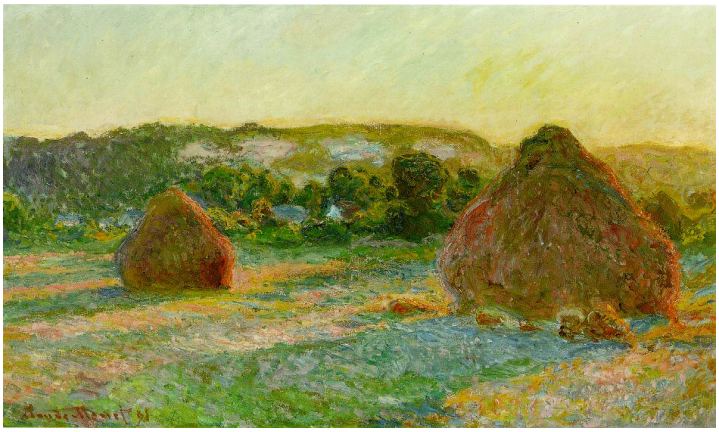
- Vereinfachung: berücksichtige bei der Berechnung der Beleuchtung eines Punktes keine sekundären Effekte (Strahlungs-austausch zwischen Objekten), nur primären Austausch zwischen Lichtquelle und Objekt → lokales Beleuchtungsmodell

- Superpositionsprinzip: betrachte Licht als Teilchen →

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

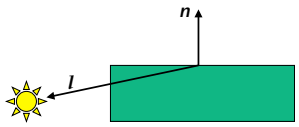
- Vereinfachung der Notation: wir lassen im Folgenden λ überall weg, und merken uns, daß alle photometrischen Größen eigtl. von λ abhängen!

- Im 19. Jahrhundert haben sich viele Maler für globale Beleuchtungseffekte interessiert:

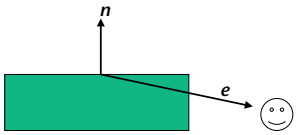


Konvention bzgl. negativer Skalarprodukte

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



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

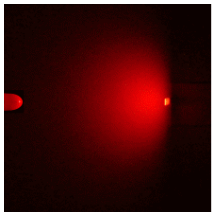


- Wir definieren im Folgenden (der Einfachheit halber) prinzipiell:

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

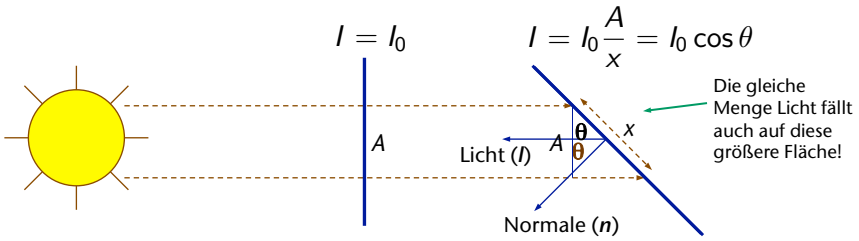
Diffuse Reflexion

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



Lambert'sche Oberflächen

- Folge des Lambert'schen Kosinus-Gesetzes: die Farbintensität I einer Oberfläche ist proportional zum Kosinus des Winkels zwischen der Oberflächennormalen und der Richtung zur Lichtquelle



- Fazit:

$$I = I_0 \cos\theta = I_0 \cdot \mathbf{n} \cdot \mathbf{l}$$

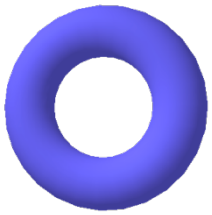
Annahme: ***n*** und ***l*** seien Einheitsvektoren

Der ambiente Term

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

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

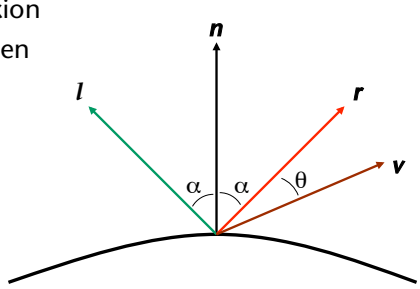
Nur ambiente Beleuchtung



Diffuse + ambiente Beleuchtung

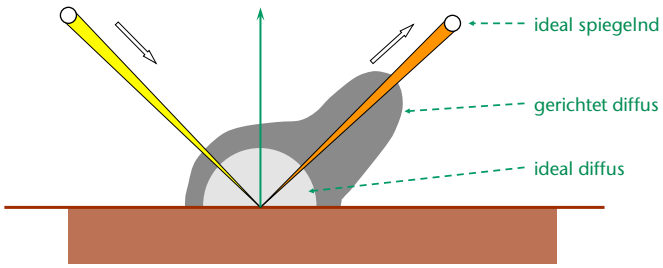
Spiegelnde Reflexion

- Oberflächenreflexion ist abhängig von
  - Richtung der Lichtquelle, ***l***
  - Oberflächennormale, ***n***
  - Richtung zum Betrachter, ***v***
- Stellt Glanzpunkte auf glänzenden Oberflächen dar
- Bei idealer spiegelnder Reflexion sieht man nur dann etwas (den hellen Punkt der Punktlicht-quelle), wenn ***r*** = ***v***.



Gerichtet diffuse Reflexion

- Ideal diffuse und ideal spiegelnde Reflexion ...



- ... sind in der Realität selten; meist eine Mischung

→ Gerichtet diffuse Reflexion: die abgestrahlte Intensität hat (oft) ein Maximum in Richtung der idealen Reflexion

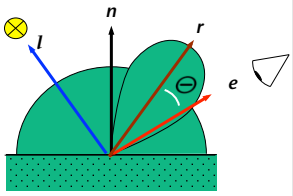
- Lösung: Kombination von "point light" und "spot light"

Das Phong-Beleuchtungsmodell [1975]

- Zusammensetzung:

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

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



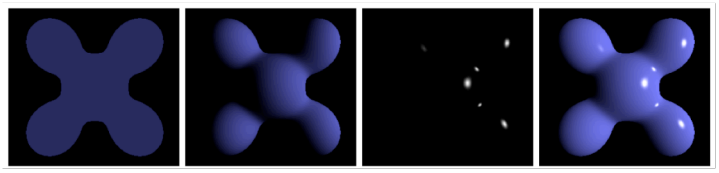
$\mathbf{r}$ = reflektierter "Lichtstrahl"
$\mathbf{e}$ = Augestrahl (Sehstrahl)
$r_s = r_s(\lambda)$ = Reflexionscharakteristik der spekularen Reflexion
p = "Glanzzahl" (*shininess*), hat keine Einheit (keine physikal. "Bedeutung")

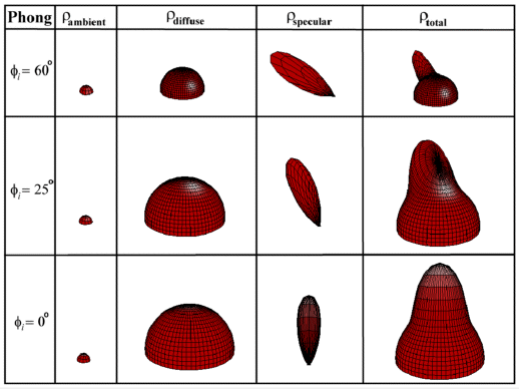
- Aufgrund des Superpositionsprinzips erhält man für n Lichtquellen:

$$I = r_d \cdot I_a + \sum_{j=1}^{n} (r_d \cos \Phi_j + r_s \cos^p \Theta_j) \cdot I_j$$

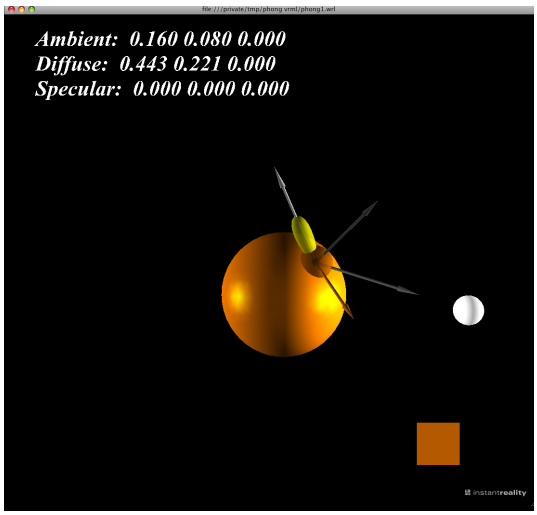
r_d = diffuser Reflexionskoeffizient (spiegelnde Materialfarbe)
r_s = spekularer Reflexionskoeffizient (diffuse Materialfarbe)

ambient + diffus + spekular = Phong





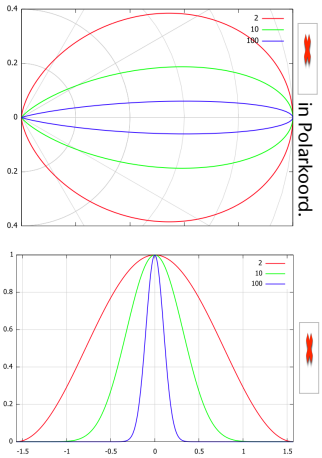
Demo

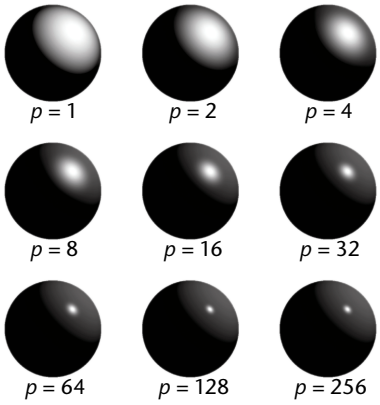


demos/phong vrml/phong.wrl
(Quelle: http://www.avl.iu.edu/%7Eewernert/gviz/phong/)

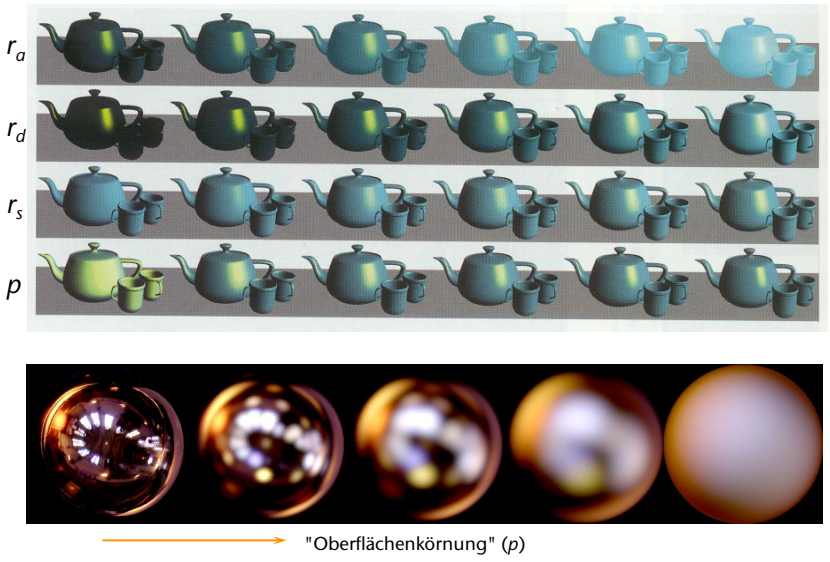
Effekt der Parameter im Phong-Modell

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





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

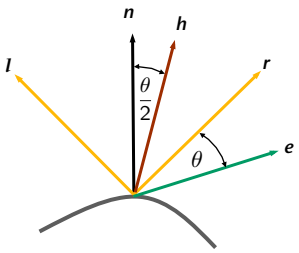


- Zusätzliche Freiheit: diffuse und spekulare (= spiegelnde) Materialfarbe können verschieden sein.
- Problem: Werte > 1 können entstehen!
  - Abhilfe: Clamping
  - Besser wäre: Erhalten von Farbton und Sättigung

Das Blinn-Phong Modell [1978]

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

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



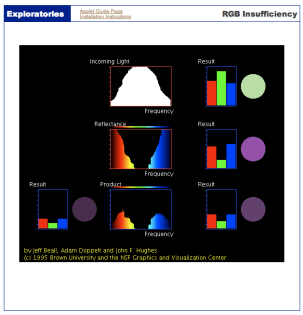
  - Setze:

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

  - Es gilt: I' ist max ↔ I ist max
- 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)

Spectral Lighting vs RGB-Lighting

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

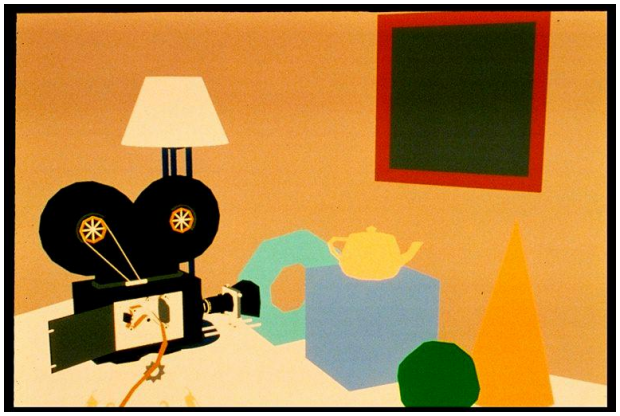


http://www.cs.brown.edu/exploratories/

Shading-Algorithmen

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

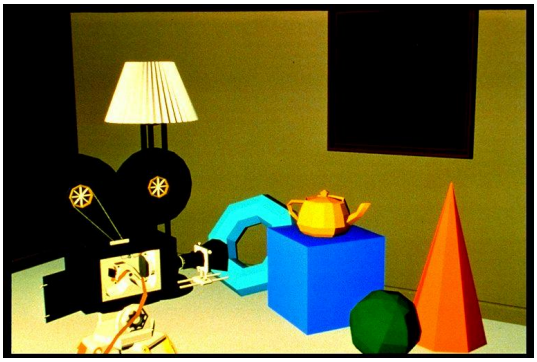
- Die Szene ganz ohne Shading ...



Pixar "Shutterbug"

Flat Shading (Konstante Beleuchtung)

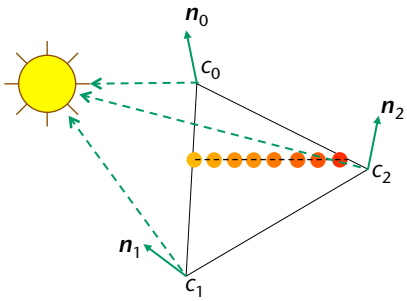
- Simpelster Shading-Algo: jedes Polygon erhält einen einheitlichen Farbwert
  - Werte dazu (im Prinzip irgend ein) Beleuchtungsmodell an irgend einer Ecke des Polygons aus

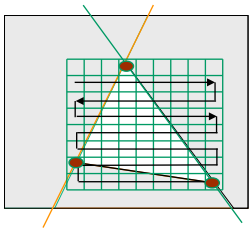


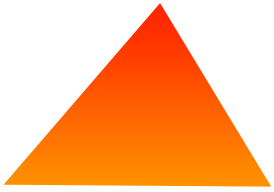
Pixar "Shutterbug"

Gouraud-Shading [1971]

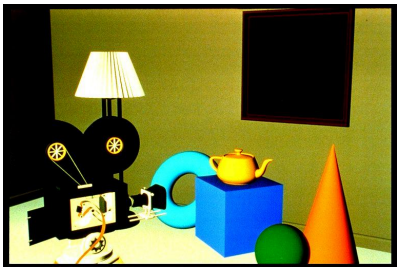
- Werte das Beleuchtungsmodell an allen 3 Ecken des Dreiecks aus, interpoliere linear dazwischen während der Scanline-Konvertierung



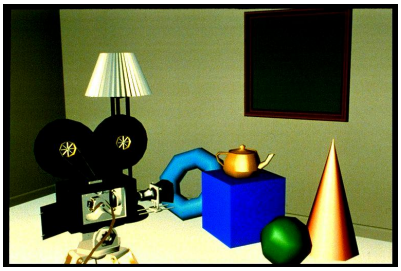




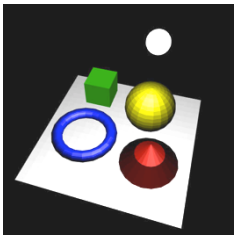
Vergleiche



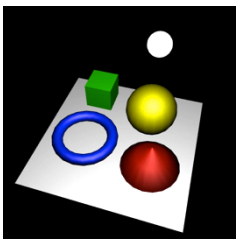
Gouraud-Shading mit
rein diffusem Beleuchtungsmodell



Gouraud-Shading mit
Phong-Lighting



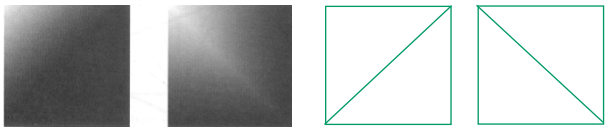
Flat-Shading



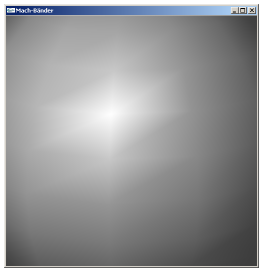
Gouraud-Shading mit
Phong-Lighting

Einfluß der Triangulierung

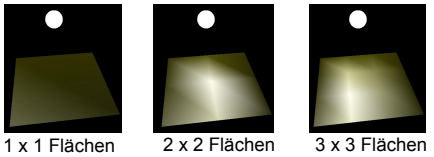
- Orientierung:



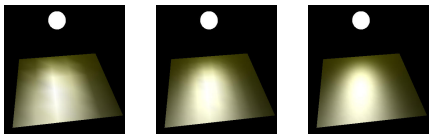
- Auflösung:



Vergrößerung



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

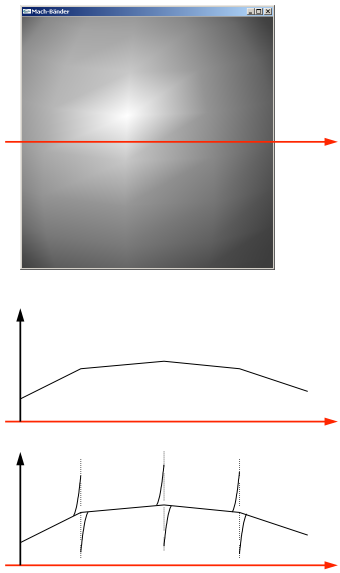


5 x 5 Flächen | 10 x 10 Flächen | 40 x 40 Flächen

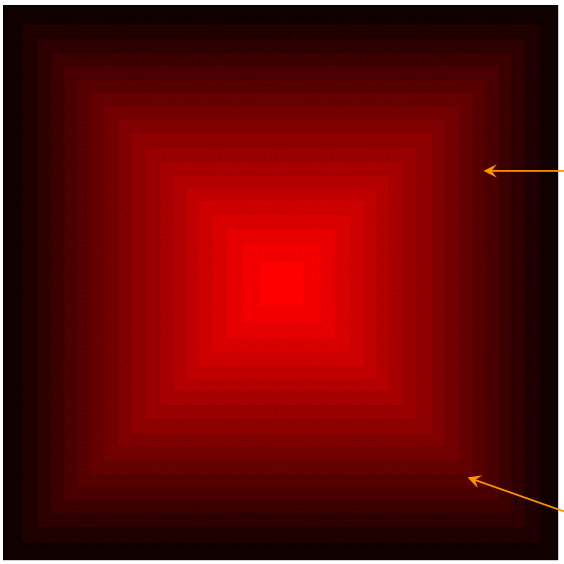
- Frage: Woher kommen die „Streifen“?

Mach-Bänder

- Problem: lineare Interpolation der „Lichtwerte“
  - Diese Interpolation ist C^0-stetig aber nicht C^1-stetig
- Das menschliche Auge „sucht“ Kanten (genau diese Knicke)
  - Es gibt Neuronen, die die Ableitung bilden (jew. für ein Retina-"Pixel")
  - Wahrgenommene Intensität = Intensität + Ableitung
  - Resultat: Machbänder bei linearer Interpolation
- Abhilfe: Hardware müsste höherwertig interpolieren...



Extremes Beispiel

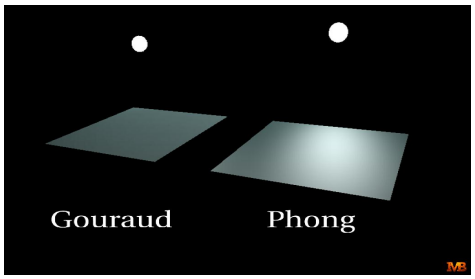


Die Intensität innerhalb eines Quadrates ist konstant!
Ein Farbverlauf von innen nach außen ist eine Illusion.

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

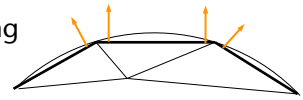
Weiteres Problem beim Gouraud-Shading

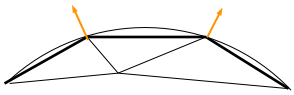
- Evtl. "verpasst" man Highlights im Inneren eines Polygons:

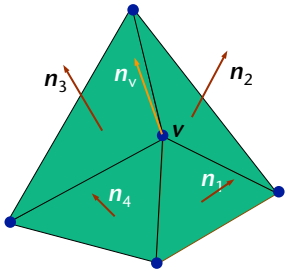


Berechnen der Normalen der Eckpunkte

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

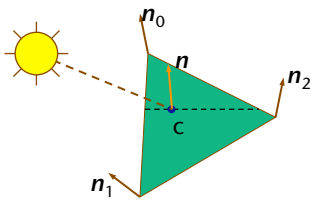


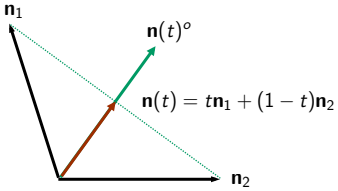


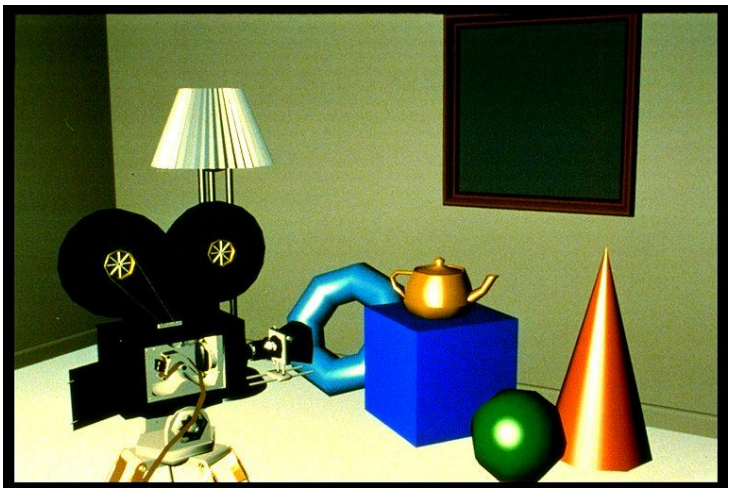


Phong-Shading

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







Beleuchtung in OpenGL

- Phong- oder Blinn-Phong-Modell mit Gouraud-Shading; mit ein paar zusätzlichen Freiheitsgraden
- Jede Lichtquelle L_i in OpenGL besteht aus 3 Teilen: ambienter ($I_{i,a}$), diffuser ($I_{i,d}$), und spekularer ($I_{i,s}$) Anteil
- Es gibt eine zusätzliche globale ambiente Lichtfarbe
- Materialien (M) bestehen aus: Emissionsfarbe (M_e), und ambiente (M_a), diffuse (M_d), spiegelnde (M_s) Reflexionskoeffizienten
- Werte größer 1 werden auf 1 "geclampt"
- Insgesamt:

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

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

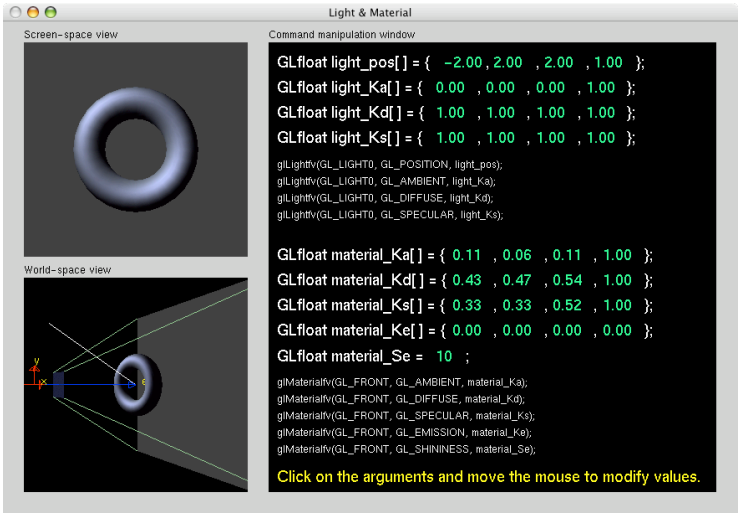
Materialdefinition

```
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

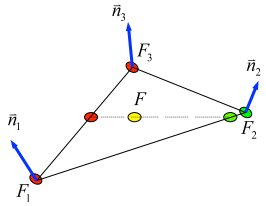


http://www.xmission.com/~nate/

Flat- vs. Gouraud-Shading

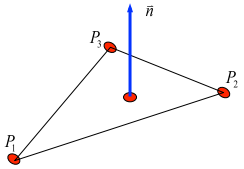
- Gouraud-Shading:

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



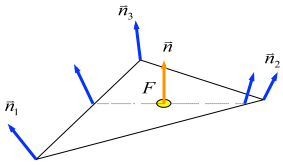
- Flat-Shading:

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



- Phong-Shading:

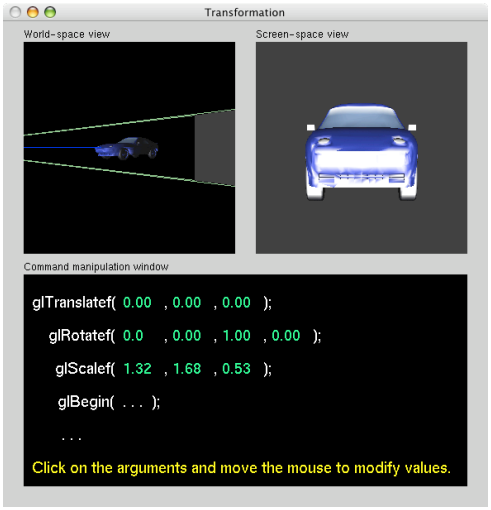
```
// In OpenGL nicht direkt
möglich ...
```



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
- Lichtquellen werden nur berücksichtigt, solange sie eingeschaltet sind
  - Somit kann man für verschiedene Objekte verschiedene Lichtquellen aktivieren

Position der Lichtquelle

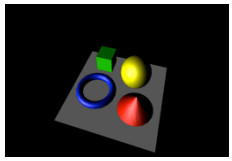
- Die Position (und Richtung) der Lichtquelle wird genauso transformiert wie ein Geometrieprimitiv
  - Transformation mit **GL_MODELVIEW**
- Lichtquelle in Weltkoordinaten („fest am Objekt“):

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

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

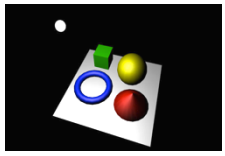
```
glLightfv( GL_LIGHT0, GL_POSITION, pos );
gluLookAt( …);
drawObject(…);
```


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



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

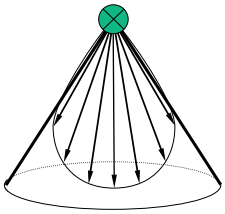
- Punktlichtquelle (*point lights*):
  - homogene Koordinate der Position == 1

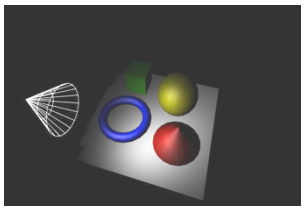


```
GLfloat position[] = { 0.0, -0.5, 0.5, 1.0 };
glLightfv( GL_LIGHT0, GL_POSITION, position );
```

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”





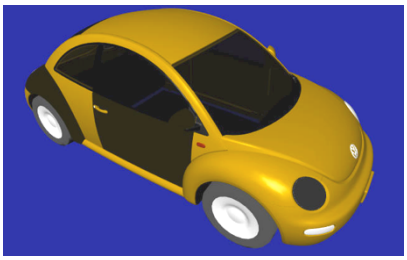
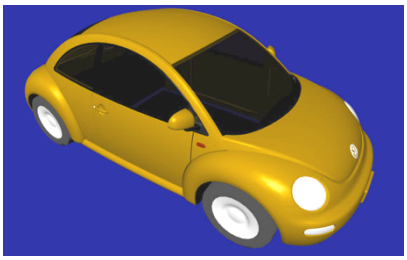
Weitere Parameter des Beleuchtungsmodells

- 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. (für *directional lights* schneller, aber etwas schlechter)
  - 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 werden per Default ignoriert
  - Mit "two-sided lighting" werden Normalen von back-facing Polygonen umgedreht, d.h., Beleuchtung ist von vorne wie von hinten gleich

Vergleich zwischen single-sided und two-sided lighting

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

Abschwächung *(attenuation)*

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

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

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

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

  - d: Abstand zwischen Lichtquelle und dem Eckpunkt

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

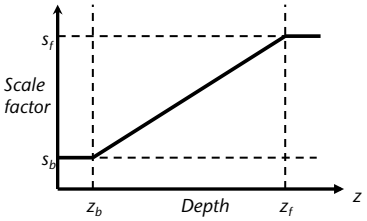
Atmosphärische Dämpfung (Fog)

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

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

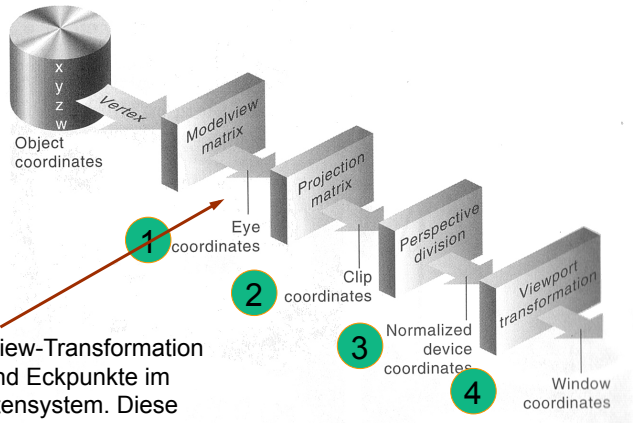
$$s(z) = s_b + \frac{z - z_b}{z_f - z_b} \cdot (s_f - s_b)$$

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



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

Wo wird beleuchtet?



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