

# Computergrafik

7

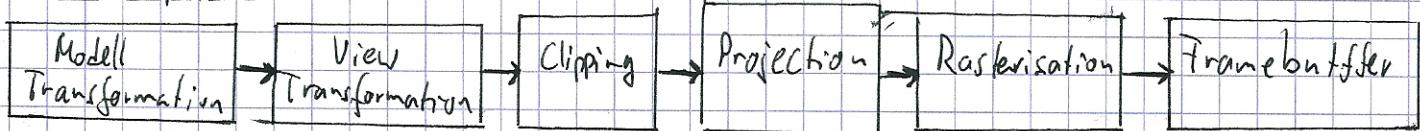
- Lichtsimulation  $L(x, \omega) = \underbrace{L_e(x, \omega)}_{\text{emit}} + \int f_r(x, \omega', \omega) L_i(x, \omega') (-\omega' \cdot n) d\omega'$

$\omega$  - Beobachtungsrichtung

$\omega'$  - Einfallen des Lichts

$x$  - Punkt, der beobachtet wird

- Grafikpipeline



input

- Objektbesch.
- Kamera
- Beleuchtung
- Viewport

+ unabhängige Verarbeitung

+ Parallel Ausführung

+ Hohe Geschwindigkeit

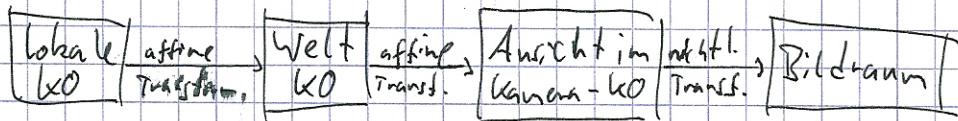
- Keine Beziehung zw. Objekten

- Keine Volumeninformationen

- Einschränkung durch gp. Betrachtungsweise

Output:

Farbwert für jeden Pixel



- Fragmentbearbeitung

Pixel: (Picture Element): Position( $x, y$ ) + Farbe( $r, g, b$ )

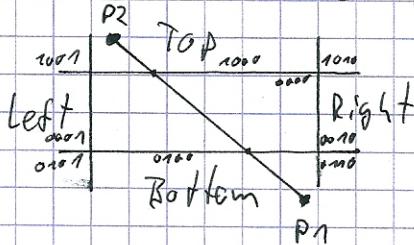
Fragment: Position( $x, y, z$ ), Normale, Tiefe, Farbe, Alpha

- Clipping: Abschneiden der Teile, die nicht sichtbar sind.

- bei der Geometrieverarbeitung (analytisch)

- nach dem Rastern (scissoring) → sinnvoll bei gestrichenen Objekten (Schriften)

- Cohen-Sutherland Algorithm



Einer Punkt wird Bitcode zugewiesen aufgrund Lage

links: 0001

top: 1000

rechts: 0000

bottom: 0100

- Clipping von Polygone: schwierig bei nicht konvexen Polygone

⇒ Tetrisation: Umwandlung in Dreiecke

Sutherland-Hodgeman Algorithmus (Polygone mit Polygone clippen)

⇒ Polygone als Menge von Linien betrachten



Beschleunigung: Bounding-Box



- effizienter: Cyrus-Beck

• Culling: Entfernen nicht benötigter Geometrie im Sichtbereich

- Backface Culling: Entfernen von Objekten "mit dem Rücken zur Kamera"

- Occlusion Culling: "die von anderen verdeckt werden"

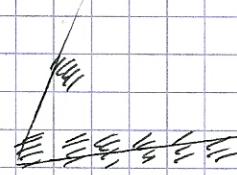
- Viewfrustum Culling: "innerhalb des Sichtvolumens"  
(→ Clipping)

- benötigt Szeneinformationen (nicht vorhanden in Grafikhardware)

N<sub>p</sub>: Normale der Fläche, N: Sichtlinie

⇒ Flächen mit  $N_p \cdot N > 0$  entfernen

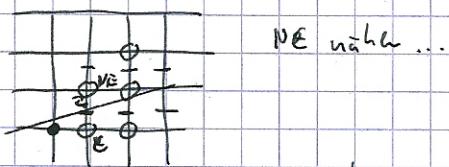
• Rasterung: Abbilden auf diskretes Raster



- Probleme bei zu steilen Steigungen

- Bresenham/Midpoint-Algorithmus

Prototyp viele inkrementelle Verfahren



- Kreise: Algo zeichnet  $\frac{1}{8}$  des Kreises ⇒ spiegeln alle Punkte

- Scanline: Rasterung von Polygone

• Füllen einer horiz Linie, Inner-/Außenfest für Pixel auf Linie

• Schnittpunkte/doppelte Endpunkte → Punkte werden weg gelassen

- ~~Flood-Fill~~: Rasterung des Polygons von innen

• Startpunkt im Polygon

• rekursives Füllen der 4er/8er Nachbarschaft

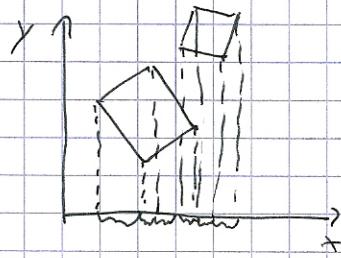
\* Sichtbarkeitsproblem Welche Teile eines 3D-Objekts nach der Projektion auf Bildebene sichtbar

⇒ Lösen der Verdeckungsproblematik

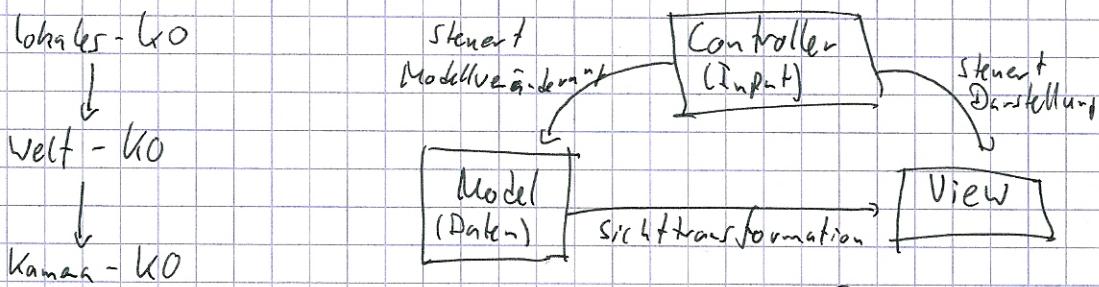
- Objektnormalalgorithmen ⇒ Weltkoordinaten des Modells  $\approx \mathcal{O}(n^2)$   
→ hohe Detailauflösung (bspw. Plotterdarstellungen)  
+ Analytische Angabe der Resultate
- komplexe Verfahren  
- schwer parallelisierbar, schwer in Hardware darzustellen
- Bildrahmalgorithmen ⇒ in Bildschirm koordinaten  $\mathcal{O}(n \cdot p)$   
+ einfache Verfahren  
+ parallelisierbar, in Hardware zu implementieren  
- beschränkte Genauigkeit  
- hohes Speicher-/Datenvolumen  
→ Genauigkeit auf Bildschirmauflösung abgestimmt  
→ heute meist eingesetzt ⇒ in Hardware ( $\rightarrow$  Tiefebuffer)  
→ Leger Strahl durch Pixel

- Optimierungen:
- Kohärenzeigenschaften (Kanten-, Flächen-, Tiefe; Framekohärenz)
  - Tests über Hülleobjekte (Würfel, Zylinder, Kugel)
  - Entfernen von Rückseiten (Halbiierung des Aufwands)
  - verring. Unterteilung (Zellmasken, BSP/oct/Quadratras, Divide & Conquer)  
→ Änderung der asympt. Aufwände
  - Algorithmen mit Priorität ⇒ zuerst ~~fernre~~ ferne Objekte zeichnen  
→ Painter's Algorithm: Fehl bei Überlappung  
→ Newell, Newell & Sancha
  - Tiefepräferenz-Algo: nur zeichne, wenn Tiefe geringer ist. (eigtl. Painter's algo)
  - gut, wenn Transparenz vorhanden

# Spannung Scan-line: Mischung aus Scanline & Tiefenpuffern (2)



- Programmierung: Model - View - Controller (Trennung von Daten und Sicht)



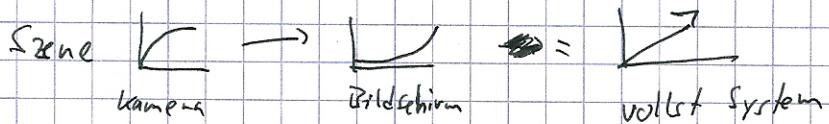
- Farbe Farbmodelle
    - RGB (Bildschirm)
    - CMYK (Drucker)
    - YIQ (Fernseher)
    - HSV, HLS
    - CIE
    - theor. Modelle
- Ein farbmodellübergreifendes Dreieckdiagramm zeigt die Beziehungen zwischen den Farben:
- RGB-Raum: blau, cyan, rot
  - CMYK: magenta, cyan, gelb, schwarz
  - YIQ: blau, grün, rot
  - HSV: hue, saturation, value
  - CIE: weiß, grün, blau
  - theor. Modelle: schwarz, cyan, magenta, gelb, rot
- Umsetzung:
- Farbtöne (hue) blau, rot, grün, ...
  - Sättigung (saturation/purity) Entfernung der Farbe von Grau
  - Helligkeit (brightness) Gesamtenergie des aufgenommenen Lichts
  - Komplementärfarben: Mischung erzeugt weißes Licht (rot+cyan, grün+magenta, ...)
  - Primärfarben: Grundfarben des Modells (z. ausreichend)
  - Farb-Gamut: Menge der Farben, die durch Primärfarben erzeugbar sind
  - RGB / CMYK nicht exakt umrechenbar:
    - gemeinsame Röhre sRGB adobeRGB
    - übergreifender Raum CIE

CIE: additive + subtraktive Farbmischung erzeugt alle Farben sichtbar 3

→ Linearkombination von RGB

→ Negativanteile auf Bildschirm nicht darstellbar

- Gamma-Korrektur: nichtlineare Verzerrung durch Kamera/Monitor



## • Geometrie

- Punkt: Koordinaten bez. KO-Ursprung

- Vektor: Verschiebungsvektor

- Vektorraum: koordinationslos, keine Distanzen

⇒ affine Räume: • Punkte  $\Rightarrow$  erlauben Geometrie  $\rightarrow$  CG

$$1. (\forall p, q \in A^n) \exists v \in V^n \rightarrow \exists v = (pq)$$

$$2. (\forall p \in A^n \wedge \forall v \in V^n) \rightarrow \exists v = (pq)$$

$$3. (\forall v = (pq) \wedge w = (qr)) \rightarrow v + w = (pr)$$

- projektive Räume ( $R^4$ )

~~$i: R^3 \rightarrow P(R^4)$~~

$i: P(R^4) \rightarrow R^3$

$(x_1, y_1, z_1) \mapsto [x_1, y_1, z_1, 1]$

$[x_1, y_1, z_1, 1] \mapsto \left( \frac{x_1}{1}, \frac{y_1}{1}, \frac{z_1}{1} \right)$

- projektive Abbildungen

injektiv

$\text{Translation: } \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$

4x4 - Matrix

$\text{Rotation } R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$\text{Skalierung: } \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$R_y = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Kombination von Transf.

→ Matrix Mult.

$R_z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

$p \rightarrow A \rightarrow B \rightarrow C \rightarrow q$

$q = C(B(Ap))$

## • Projektionen:

- orthografisch: Projektionsstrahlen treffen im rechten Winkel auf (Parallelprojektion/Schnittsichten)
- di-/tri-/isometrisch: Möglichkeit zum Maßnehmen
- Perspektive: Ein-/Zwei-/Dreipunktsperspektive am häufigsten
- Model-View-Controller Prinzip trennt Objekt / Kamera  
→ Projektionsmatrix und Modellviewmatrix
- Multiprojektionen: Verschiebung der Flachtpunkte im versch. Gegenenden  
→ kann bspw bei Winkelverzerrung helfen  
→ Problem mit Sichtbarkeit/Verdeckung

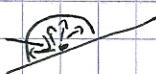
## • Betrachtung

Variablen: Position  $U_x, U_y, U_z$  }  
 Richtung  $\phi, \Theta$  } 6-dimensionale Funktionen  
 Zeit +

- Fotografie Position, Zeit fest, Ausschnitt von Richtung
- Film/Video  $(U_x, U_y, U_z) = f(t)$ , Ausschnitt von Richtung
- Holografie + fest, Ausschnitt von Position/Richtung
- CG → simuliert Foto und Film
- Beleuchtung: Berechnung der Beleuchtungsgleichung
- physikalische Grundlagen: - Emission, Reflexion/Refraction, Absorption  
- Wellenmodell / Quantenmodell
- Licht wird: absorbiert, reflectiert, gestreut (scattered), gebrochen (refracted)  
verteilt (transmittiert)

$$P = \int \int_{\text{Area}} L(x \rightarrow \Theta) \cdot \cos \Theta \cdot d\omega_\Theta \cdot dA = L \cdot \text{Area} \cdot \pi$$

diffraktiv (Lambertian)



spiegelnd (specular)



direktional (glänzend) (glossy, specular)



## Rendering - Gleichung

$$L(\rightarrow \oplus) = L_e(x \rightarrow \oplus) + \int_{\text{emission}}^{} \int_{\text{hemisphere}}^{} \int_{\text{light+}}^{} L(x \leftarrow \psi) f_r(x, \psi \rightarrow \oplus) \cos(N_x, \psi) d\omega_\psi \text{ reflectiveness direction}$$

## Beleuchtung in OpenGL

- Punktlichtquellen

- nur direkte Beleuchtung

- Materialien: diffus, glänzend, transparent

## Schatten Kernschatten (umbra), Halbschatten (Penumbra)

OpenGL: Punktlichtquellen  $\Rightarrow$  nur Kernschatten

nicht automatisch  $\rightarrow$  Shadowmaps

$\rightarrow$  Volumenschatten

## Phong'scher Beleuchtungsmodell (ver einfacht)

- Licht (ambient, diffus, specular)

- Reflexion (ambient, diffus, specular)

- alle Parameter für rot, grün, blau getrennt

$$I = k_a I_a + \sum_i I_i (k_d (N \cdot L) + k_s (R \cdot V)^k)$$

$k \rightarrow$  Größe des Lichtstrahls

$k \rightarrow$  Reflektivität

## Shading: Phong-Shading (1975) gute Ergebnisse, geringe Kosten

aber Raytracing und Radiosity besser

## lokales vs globales Beleuchtungsmodell

flat-Shading: ein Helligkeitswert pro Polygon

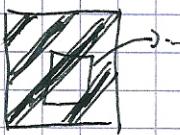
gouraud-shading: Normale in den Ecken  $\rightarrow$  Berechnung der Fläche

## Texturen (= Bearbeitung von Pixel(daten)) $\rightarrow$ eigene Pipeline

Stencil  $\rightarrow$  Accumulation  $\rightarrow$  Overlay planes  $\rightarrow$  Auxiliary  $\mathbb{P} \rightarrow$  Color index  $\rightarrow$  depth  $\mathbb{P}$

$\rightarrow$  Back  $\mathbb{P} \rightarrow$  Front  $\mathbb{P}$

- Texture-Mapping: Aliasingprobleme bei ungenauer Abtastung  
(Moiré)



Vergroßern  $\rightarrow$  Interpolation  $\rightarrow$  undersampling  
verkleinern  $\rightarrow$  Mittelwert  $\rightarrow$  undersampling filtering



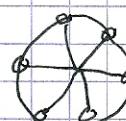
- linearer Texture-Mapping

④

- 2D/3D Mapping : Bilder  $\neq$  Volumenteratur
- planare Textur "Projektion von einer Seite" gut bei Würfel
- zylindrische Textur "Projektion von runderum" gut bei Säulen
- sphärische Textur
- kubische Textur  $\rightarrow$  planare Textur von mehreren Seiten
  - vorfertig Textur, sechs Einzelbilder
- environment-mapping: Objekt wird auf Fläche projiziert
- reflection-mapping ähnlich  $\rightarrow$  Objekt als Textur vorhanden
- uv-mapping: Textur wird auf Objekt "gelegt"
  - $\rightarrow$  natürlicher Aussehen
  - $\rightarrow$  lineare und nicht-lineare Abbildung möglich
- 3D-Mapping: Erzeugung der Volumenteratur Bsp. Projektion, Funktionen + Rauschen (lattice noise, gradient noise, fractal Rauschen)
- Bump-Mapping: Oberflächennormalen werden geändert  
Objekt sieht "3D" auf Oberfläche aus  $\rightarrow$  nichts an Kanten
- Displacement-Mapping: Form wird verändert
- Antialiasing:
  - Flächenintegral (uv-Mapping) (Prefilter) sehr gut, langsam
  - Mehrere Punkte (Randpunkte) (Supersampling) schnell
  - Mipmaps: kleinere Textur erzeugen (mehrere)
  - Kombination zw Supersampling & Mipmaps
- Farbwahrnehmung



L-type



n-Harmonic



i-type



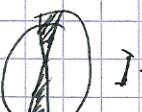
v-type



f-type



y-type



j-type



x-type

- harmonische Farben: • nahe zusammen liegende Farben

• bunt & grau

• sattiert & unsattoriert

• Kontrast (komplementär, warm-kalt)

(S)

- Hue-Diagramm berechnen,

Färbungen zu harmonischen Typ verschieben

• Schprozess: Auge verarbeitet: Bild & das wir sehen ist Kombination mehrerer Bilder ( $\rightarrow$  HDR)

- Blauproblem: Wenig Rezeptoren für blau im Auge  
 $\rightarrow$  blau - rot Kontraste vermeiden

- Kontrastverstärkung: sorgt dafür, dass man auch auf Bildschirmen (1:500) was sehen kann (Natur 1:100 000)

- "Leuchtkräfte" durch wenige weiße Stellen

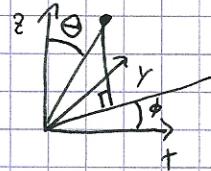
- Präattentive Wahrnehmung (Textur, Figure-Ground-Segregation)

- Attentive Wahrnehmung (2D Konstruktion, 3D Konstruktion, Attributierung)

- Assoziation mit vorhandenem Wissen (Fehlendes Wissen wird ergänzt)

### UV-Mapping

$$(u, v) \in [0, 1]^2 \rightarrow (\Theta, \phi) \in [\theta, \pi] \times [-\pi, \pi]$$



$$\left. \begin{array}{l} \Theta = \arccos(z) = \pi(p, \mathbf{v}) \\ \phi = \arctan(\frac{y}{x}) = 2\pi u \end{array} \right\} (\mathbf{r}, \Theta, \phi) \rightarrow \text{Punkt auf der Oberfläche der Einheitskugel mit Mittelpunkt } (\Theta, \phi, \Theta)$$

$$u = \frac{\phi}{2\pi} = \frac{\arctan(\frac{y}{x})}{2\pi}$$

$$v = \frac{\pi - \Theta}{\pi} = \cancel{\frac{\pi - \arccos(z)}{\pi}}$$

$\text{atan2}(y, x)$  übernimmt Fallunterscheidung

$$\text{atan2}(y, x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{für } x \geq 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{für } x < 0, y \geq 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{für } x < 0, y < 0 \\ \frac{\pi}{2} & \text{für } x = 0, y > 0 \\ -\frac{\pi}{2} & \text{für } x = 0, y < 0 \\ 0 & \text{für } x = 0, y = 0 \end{cases}$$