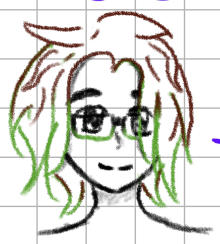
Disclaimer

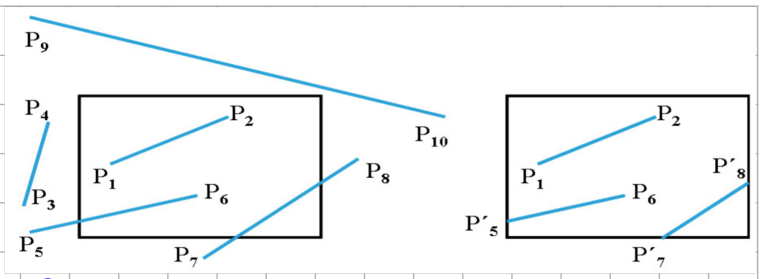
Das ist nur eine Zusammenfassung und kein Ersatz für das Skriptum / die VOs. Keine Garantie darauf, dass alles so stimmt, wie es hier steht. Das ist nur meine Interpretation der Inhalte. Falls etwas unklar sein sollte, bitte im Skriptum nachschauen. Einige Unterkapitel (wie z.B. "Andere Objektrepräsentationen") könnten fehlen, da ich sie als unwichtig / nicht prüfungsrelevant erachtet hab. Jegliche Bilder gehören den Urhebern des Skriptums.



Viel Spaß beim Lernen!

Inhaltsverzeichnis

Linien-Clipping



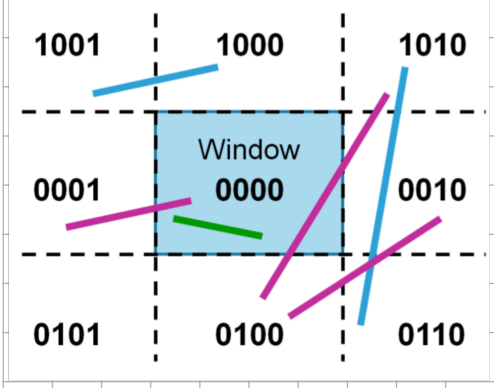
je früher in der Graphikpipeline, desto weniger unnötige Umformungen

Clipping in Weltkoordinaten: frühstmögliche, analytische Berechnung

Clipping in Clipkoordinaten: analytische Berechnung am Bildrand

Clipping in der Rasterkonversion: in der Umformung von Primitiven zu Pkt.

Cohen-Sutherland-Verfahren



⇒ Endpunkte bzgl. Lage in 4 Bit codieren

1. OR = 0000 ⇒ Linie ganz sichtbar
2. AND ≠ 0000 ⇒ Linie ganz unsichtbar
3. else: mit relevanten Fensterkante schneiden

& weggeschnittenen Punkt durch Schnittpunkt ersetzen

Schnittpunktberechnung: vertikal – $y = y_0 + m(x_w - x_0)$, für $w \in \{w_{min}, w_{max}\}$

horizontal – $x = x_0 + (y_w - y_0)/m$, für $w \in \{w_{min}, w_{max}\}$

⇒ Punkte auf der Kante = innerhalb

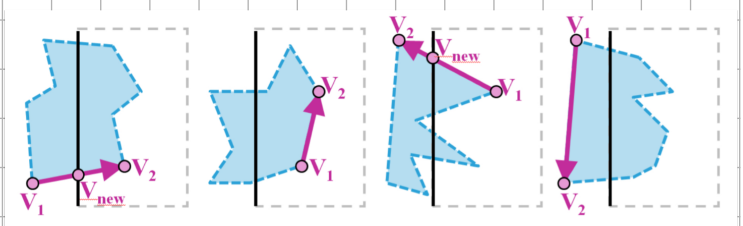
Polygon-Clipping

⇒ Füllung problematisch bei Linienclipping von Polygonen, daher:

Sutherland-Hodgman-Verfahren

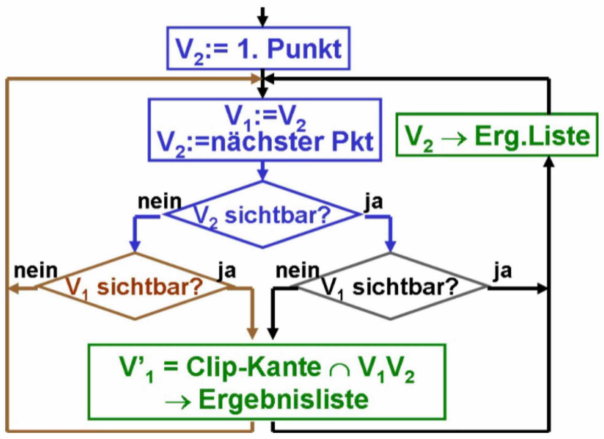
4 verschiedene Fälle:

⇒ sequenzielle Eckpunktbearbeitung



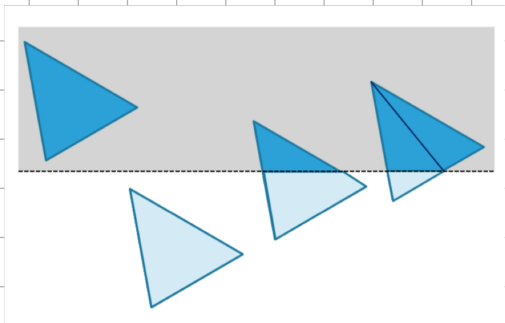
1. out→in (Output: V_{new}, V_2) 2. in→in (Output: V_2) 3. in→out (Output: V_{new}) 4. out→out (kein Output)

Der Algorithmus *für eine Fensterkante* läuft dann so ab:



- 3 Zwischenergebnisse weniger, wenn rekursiver Aufruf
- Schlussendlich: Erstellung von Kanten entlang des Clippingfensters

Clipping von Dreiecken



⇒ Resultat muss wieder Dreieck sein, erneut 4 Fälle vorhanden (s. Abb.)

Clipping in Clipkoordinaten

⇒ nur Vergleich zweier Zahlen nötig, da alle Ränder = achsenparallel ($x = \pm 1$, $y = \pm 1$, $z = \pm 1$).

⇒ Clipping der Ebenen $x = \pm h$, $y \pm h$, $z \pm h$ vor Homogenisierung: keine Projektion der Punkte hinter Kamerapunkt & ersparung der Homogenisierungsdivision

Aliasing & Antialiasing

Aliasing-Effekte: Fehler bei der Diskretisierung analoger Daten

Nyquist-Limit: Grenze des Nyquist-Shannon-Theorems

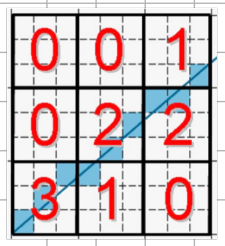
⇒ führt zu Aliasing, wenn Abtastrate zu grob

Antialiasing: Methoden zur Reduktion dieser Artefakte, u.a. durch Vorfiltern (durch Oversampling) oder durch Nachbearbeiten

Antialiasing von Linien

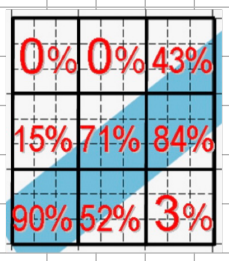
Ziel: stark durchkreuzte Pixel stark, gestreifte Pixel leicht gefärbt

1. Pixel in Subpixel unterteilen (manchmal in der Mitte stärker gewichtet als am Rand – "weighted oversampling")
2. Zählen, wie viele Subpixel auf der Linie liegen
3. Intensität proportional zur Anzahl wählen



Falls Linie breiter:

2. Überdeckungsprozentsatz berechnen
3. Intensität der Linienfarbe wählen



Antialiasing von Polygonen

zwei Möglichkeiten: 1) Supersampling/Oversampling

2) Berechnung, wie sehr das Polygon das Pixel überdeckt

⇒ erfolgt mit Rasterkonversion (= Randerzeugung & Füllung), daher sehr einfache Berechnung durch Scanlinien-Füllung

Backface Detection / Backface Culling (Objektraum-Verfahren)

Entfernen aller Polygone, deren Oberflächennormale vom Betrachter wegzeigt (im Durchschnitt 50%)

Berechnung bei:

orthographischer Projektion: $V_{view} \cdot N > 0 \rightarrow$ unsichtbar (V_{view} = Blickrichtungsvektor, N = Oberflächennormale)

perspektivischer Projektion: $Ax + By + Cz + D < 0 \rightarrow$ unsichtbar ($Ax+By+\ldots$ = Ebenengl., x, y, z = Blickpunkt)

Z-Puffer-Verfahren (Bildraum-Verfahren)

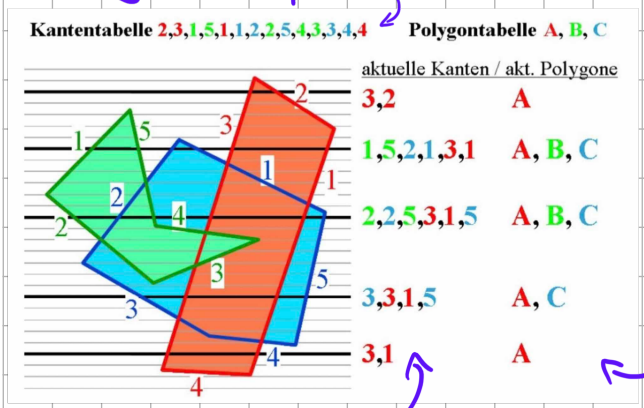
geschieht in zusätzlichem Puffer

1. Farbinformation & Position des Objekt speichern für alle Pixel
   ⇒ z-Wert reicht, da z-Richtung = Blickrichtung, daher "z-Puffer"
2. Objekte in beliebiger Reihenfolge zeichnen
3. z-Werte des nächsten Objekts/Polygons berechnen
4. Falls neuer z-Wert größer/näher zum Betrachter:
   Objekt über alten Bildwert zeichnen & z-Wert im Puffer ersetzen
   Sonst: neues Objekt = verdeckt/unsichtbar

Vorteil: Objekte müssen nicht sortiert werden

Scanline-Methode (Bildraum-Verfahren)

nach größtem y-Wert sortiert



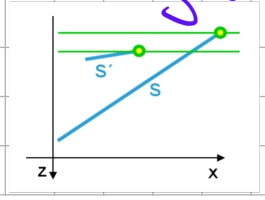
⇒ Berechnung zeilenweise, da unwesentliche Sichtbarkeitsunterschiede zwischen den Scanlines

Schlussendlich: nächstes Polygon zum Betrachter wird gezeichnet

Sortierung nach x-Wert, sobald berechnet

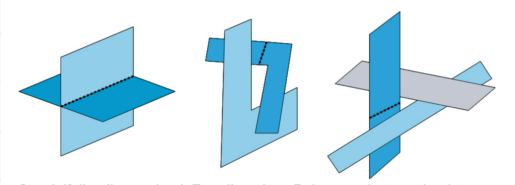
Depth-Sorting-Methode (Objektraum-Verfahren)

1. Polygone nach kleinstem z-Wert (größte Tiefe) sortieren
2. Vergleiche Polygon S mit anderem Polygon S'; Sortierung korrekt, falls:
   - max. z-Wert von S < min. z-Wert von S'
   - x- & y-Intervalle schneiden sich nicht



- alle Ecken von S hinter S' Ebene liegen
- alle Ecken von S' vor S' Ebene liegen
- Projektionen von S & S' auf die xy-Ebene sich nicht schneiden

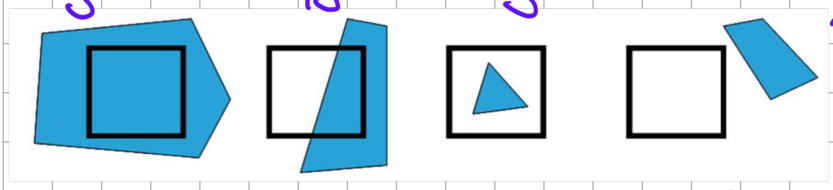
3. Falls Sortierung falsch: Vertauschen & neu kontrollieren, falls erneut falsch: Spezialfall ⇒

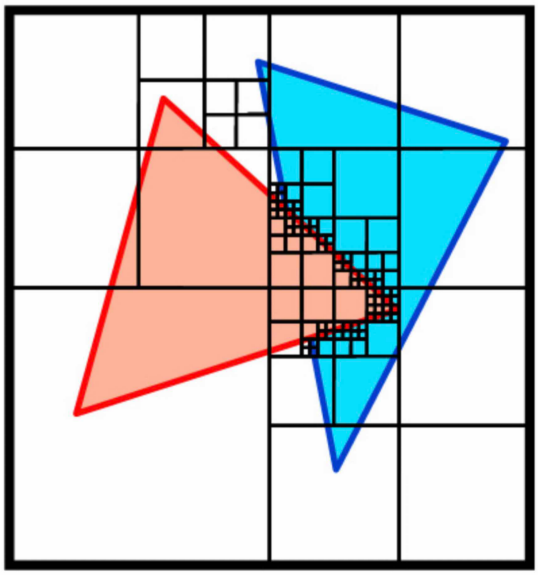


Spezialfälle, die nur durch Zerteilen eines Polgons gelöst werden können

Area-Subdivision-Methode (Bildraum-Verf.)

Lagen des Polygons bezüglich des Fensters





Quadrat = leer: nix tun

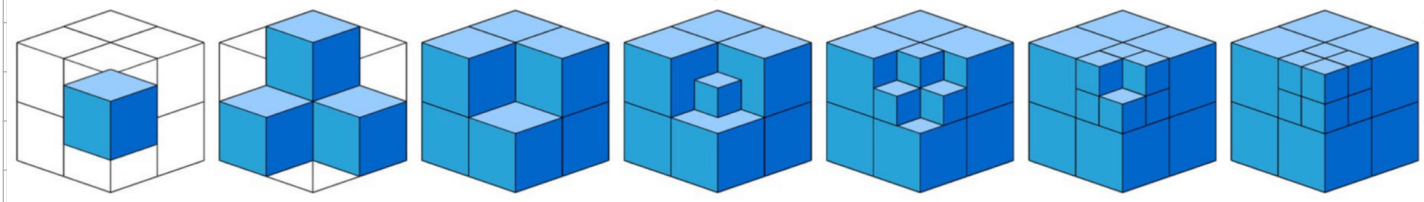
ein Polygon im Quadrat: dieses zeichnen

Basisfälle

zwei (nicht-überlappende) Polygone im Quadrat: Quadrat unterteilen

Rekursion

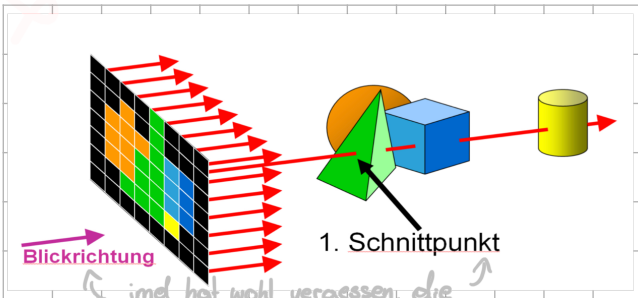
Octree-Methode (Objektraum-Verfahren)



Vorteil: implizietes Wissen, was weiter vorne und was weiter hinten ist

Ray-Casting (Bildraum-Verfahren)

Ray-Casting = für jedes Pixel der Darstellungsfläche:
- erzeuge eine Gerade durch das Pixel in Blickrichtung („Blickstrahl")
- schneide den Blickstrahl mit allen Objekten
- wähle aus der Schnittpunktliste den zum Betrachter nächsten Punkt
- färbe das Pixel mit der Farbe der Oberfläche dieses Punktes



jmd hat wohl vergessen, die Autokorrektursprache zu ändern lmao

Vorteil: Leichtes Rendern jeglicher Oberflächen

Nachteil: großer Aufwand, da für jedes Pixel ein Schnitt benötigt wird

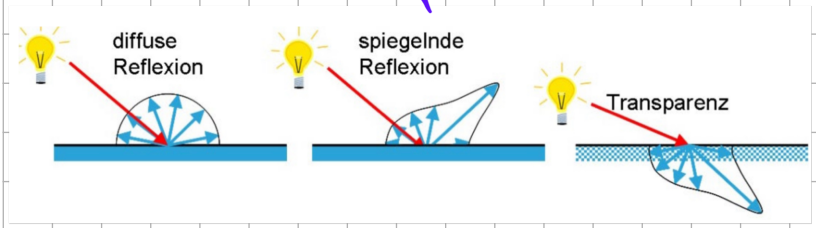
Lichtquellen und Oberflächen

Merkmale von Lichtquellen:

a. Form (Lichtrichtung, (gerichtete) Punktlichtquellen, flächige Lichtquellen)

b. Eigenschaften (Helligkeit, Farbe, Entfernung)

Merkmale von Oberflächen:



Reale Oberflächen: Mischung von diesen

Hintergrundlicht

Wegen Lichtreflexion: nie ganz dunkel, wenn kein direktes Licht da ist

⇒ Inkludierung eines konstanten Wertes I_a zum Beleuchtungsmodell

Lambert'sches Gesetz

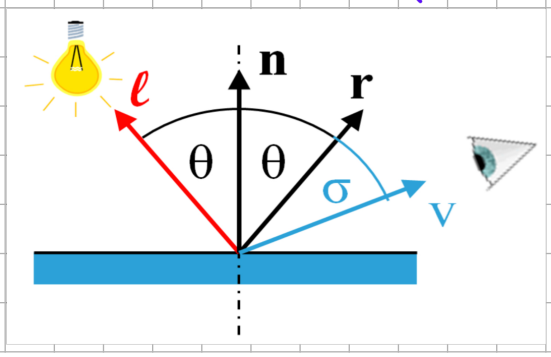
flacheres Licht = dünklere Oberfläche,

daher: $L =$

$= k_d \cdot I \cdot \cos(\theta)$

$= k_d \cdot I \cdot n \cdot \ell$

Skalarprodukt



I = Helligkeit der Lichtquelle

k_d = diffuser Reflexionskoeffizient ($0 \leq k_d \leq 1$)

n = Oberflächennormale

ℓ = Lichtrichtung

L = resultierende Pixelfarbe

Glanzpunkte

exakte spiegelnde Reflexion schwierig zu berechnen, daher verwendet man einfachere Funktion $\cos^p$ ⇒ größeres p = kleinerer Glanzpunkt

Phong-Beleuchtungsmodell:

$I_{spec} = k_s \cdot I \cdot \cos^p(\sigma) = k_s \cdot I \cdot (r \cdot v)^p$

k_s = spiegelnder Reflexionskoeffizent

r = exakter Reflexionsstrahl

h = Halbierende

Fernell'sche Grenze $W(\Theta)$ akkurater als k_s, aber meistens fast konstant

$r = (2n \cdot \ell)n - \ell$ ⇒ vereinfacht: $r \cdot v = n \cdot h$, daher

Blinn-Phong-Beleuchtungsmodell: $I_{spec} = k_s \cdot I \cdot (n \cdot h_i)^p$

Ein einfaches Beleuchtungsmodell

Zusammensetzung: $L = k_a \cdot I_a + \sum_{i=1}^{N} (k_d \cdot I_i \cdot (n \cdot \ell) + k_s \cdot I_i \cdot (n \cdot h_i)^p)$

Flat-Shading

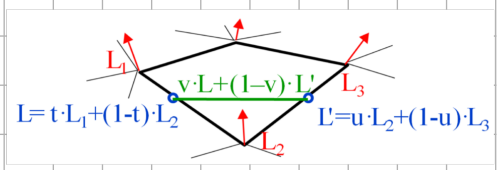
unterschiedliche Färbung basierend auf Oberflächennormale

Problem: Grenzen zwischen Polygonen werden erkennbar

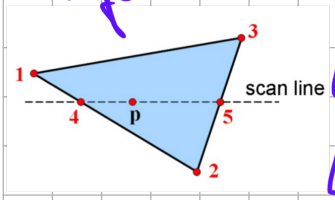
Lösung: Interpolation der Schattierung zwischen Polygonen

Gourand-Schattierung

1. Berechnung von Mittelwerten der Oberflächennormalen
2. Berechnung der jeweiligen Helligkeitswerte
3. Lineare Interpolation entlang der Kanten
4. Erneute lineare Interpolation entlang der Scanlines



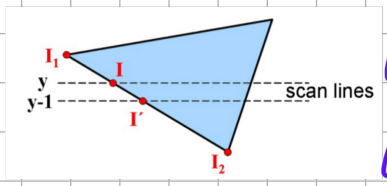
Einfache lineare Interpolation



$$L_4 = \frac{y_4 - y_2}{y_1 - y_2} L_1 + \frac{y_1 - y_4}{y_1 - y_2} L_2$$

$$L_p = \frac{x_5 - x_p}{x_5 - x_4} L_4 + \frac{x_p - x_4}{x_5 - x_4} L_5$$

Inkrementelle Interpolation



$$L = \frac{y - y_2}{y_1 - y_2} L_1 + \frac{y_1 - y}{y_1 - y_2} L_2$$

$$L' = L + \frac{L_2 - L_1}{y_1 - y_2}$$

Problem: Silhouette bleibt unverändert, zufällige Ergebnisse bei Glanzpkt.

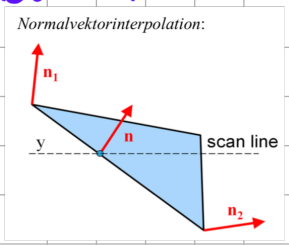
Phong-Shattierung

(Reihenfolge der Schritte anders)

Vorgehensweise wie bei Gouraud-Schattierung, aber: 1., 3., 4., 2.

Vorteil: schönere Ergebnisse

Nachteil: höherer Aufwand



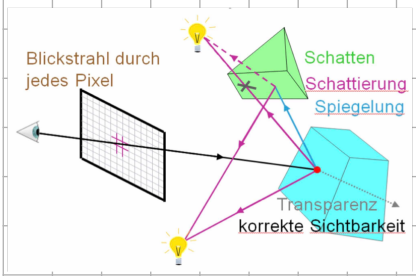
$$n = n_1 \cdot \frac{y - y_2}{y_1 - y_2} + n_2 \cdot \frac{y_1 - y}{y_1 - y_2}$$

Korrekte Sichtbarkeit und Schattierung (Ray-Tracing)

1. Blickstrahl durch jedes Pixel legen & mit allen Objekten schneiden
2. nächsten Schnittpunkt zum Bild wählen
3. Schattierung an dem Pkt. zur Pixelfarbe machen

Ergebnis: korrekte Sichtbarkeit

Schatten



Schattenfühler: Lichtstrahl, der Objekt & Quelle verbindet. Falls er etwas schneidet, wird die Lichtquelle nicht berücksichtigt

Spiegelbilder

Falls Blickstrahl auf einen Spiegel trifft: gespiegelten Reflexions-

strahl mit allen Objekten schneiden & nächstes auswählen

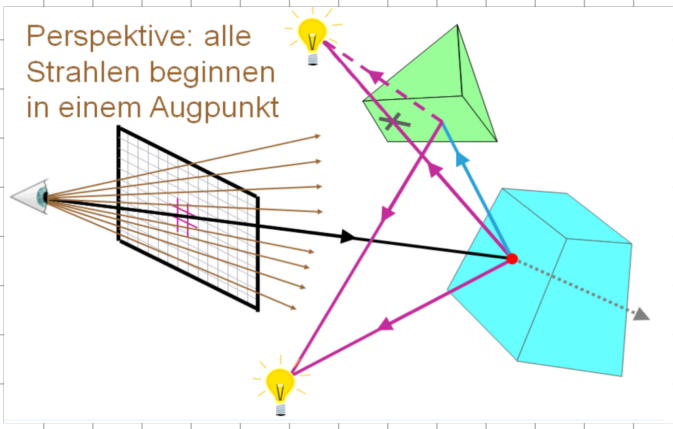
Transparenz

Transparenzstrahl durchsenden ⇒ nächster Schnittpunkt nach transparentem Objekt = sichtbar

Rekursion

Gleichwertige Behandlung aller Strahlen, außer Schattenfühlern

Perspektive



orthonormale Parallelprojektion:
Verwendung von parallelen Strahlen, die normal zur Bildebene sind

Perspektivische Projektion: siehe Abb.

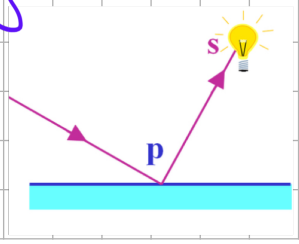
Implementierung

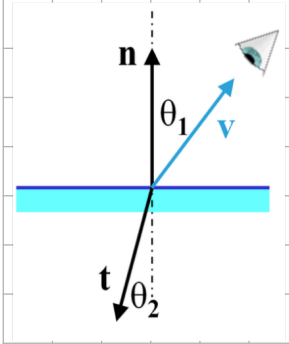
```
FOR alle Pixel p0 DO
 1. lege Blickstrahl vom Auge e aus durch p0
     schneide mit allen Objekten und wähle den nähesten Schnittpunkt p
 2.FOR alle Lichtquellen s DO
     schneide Schattenfühler p -> s mit allen Objekten
     IF kein Schnittpunkt zwischen p, s THEN Schattierung += Einfluss von s
 3.IF Oberfläche von p ist spiegelnd
   THEN verfolge Sekundärstrahl; Schattierung += Einfluss der Reflexion
 4.IF Oberfläche von p ist transparent
   THEN verfolge Sekundärstrahl; Schattierung += Einfluss d. Transparenz
```

parametrisierte Strahlenform: $p(t) = p_0 + t \cdot d$; p_0 = Startpunkt, d = Richtungsvektor | t = Parameter

Primärstrahlen: $e + t \cdot (p_0 - e)$; p_0 = Pixel, e = Augpunkt

Sekundärstrahlen: $p + t \cdot (s - p)$; p = Oberflächenpunkt

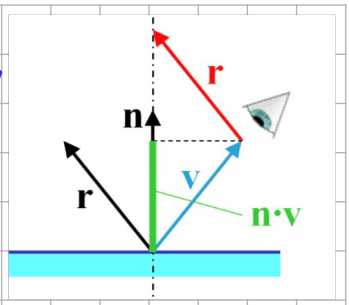




Reflexionsstrahlen: $p+t\cdot r$; $r=(2n\cdot v)n-v$

Transparenzstrahlen: $p+t\cdot t$

$t=-\frac{n_1}{n_2}v-\left(\cos(\theta_2)-\frac{n_1}{n_2}\cos(\theta_1)\right)n$

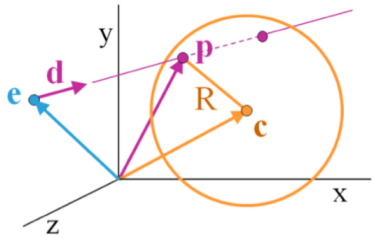


Schnitte zwischen Strahlen und Objekten

notwendige Kriterien für Ray-Tracing:

- Schnittpunkt muss mit Gerade berechenbar sein
- Oberflächennormale muss bekannt sein
- Materialeigenschaften müssen vorhanden sein

Schnitt Strahl-Kugel



Kugelgleichung: $|\mathbf{p}-\mathbf{c}|^2-R^2=0$

In diese setzt man den Strahl ein: $|(\mathbf{e}+t\mathbf{d})-\mathbf{c}|^2-R^2=0$

Dann wird zur besseren Lesbarkeit $\Delta\mathbf{p}$ eingeführt: $\Delta\mathbf{p}=\mathbf{c}-\mathbf{e}$

Und man erhält eine quadratische Gleichung in t: $t^2-2(\mathbf{d}\cdot\Delta\mathbf{p})t+(|\Delta\mathbf{p}|^2-R^2)=0$ $(\mathbf{d}^2=1)$

Die 2 Lösungen entsprechen den beiden Schnittpunkten mit der Kugel: $t=\mathbf{d}\cdot\Delta\mathbf{p}\pm\sqrt{(\mathbf{d}\cdot\Delta\mathbf{p})^2-|\Delta\mathbf{p}|^2+R^2}$

Umgehung von Rundungsfehlern: $t=d\cdot\Delta p\pm\sqrt{R^2-|\Delta p-(d\cdot\Delta p)d|^2}$

Schnitt Strahl-Polygon

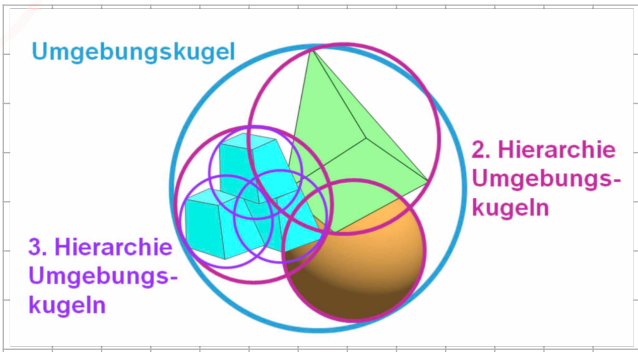
Ebenengleichung: $Ax+By+Cz+D=0$ (mit $n=(A,B,C)$)

Strahl einsetzen: $n\cdot(p_0+t\cdot d)=-D$

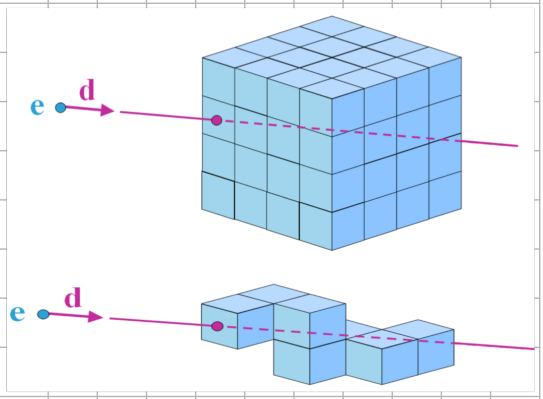
Daraus folgt: $t=-\frac{D+n\cdot p_0}{n\cdot d}$

Objektumgebungen

Überprüfen, ob der Strahl in die Nähe des Objekts kommt durch



Umgebungskugeln
Verringerung des Aufwands von $O(n)$ zu $O(\log(n))$



Raumteilungsmethode

Unterteilen des Raumes in Raster. Objekt wird nur geschnitten, wenn der Teilraum durchquert wird.

Radiosity (Globale Beleuchtungen und Texturen)

beleuchtete Gegenstände => Sekundäre Lichtquellen

Berechnung der Lichtausbreitung ohne Beobachter, da einfachere Darstellung aus versch. Richtungen

Patches = ebene Polygone P_i mit homogener, perfekt diffuser Oberfläche

Radiosity B_i = gesamte abgestrahlte Energie, E_i = Eigenstrahlung

vereinfachte Formel: $B_i = E_i + P_i \cdot \sum_{j=1}^{n} B_j \cdot F_{ij}$, mit F_{ij} = Formfaktoren

B_j = Radiosity aller anderen Patches, n = Anzahl Patches der Szene

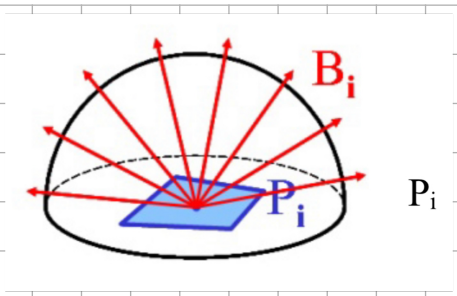
$E_i = B_i - P_i \sum_{j \neq i} B_j F_{ij}$ => mit Gleichungssystem iterativ lösbar

Berechnung der Formfaktoren (sry für die fehlenden Erklärungen)

Formfaktor = Anteil der von P_i nach P_j gehenden Strahlungsenergie

$F_{ij} = \frac{\cos(\phi_i)\cos(\phi_j)A_j}{\pi r^2}$, $A_i \cdot F_{ij} = F_{ji} \cdot A_j$

Fortschreitende Verfeinerung



Verteilung der Energie des hellsten Patches auf die anderen: $B_{(j \text{ von } B_i)} = \rho_j B_i F_{ij} \frac{A_i}{A_j}$

Vereinfacht sieht ein Iterationsschritt daher so aus:

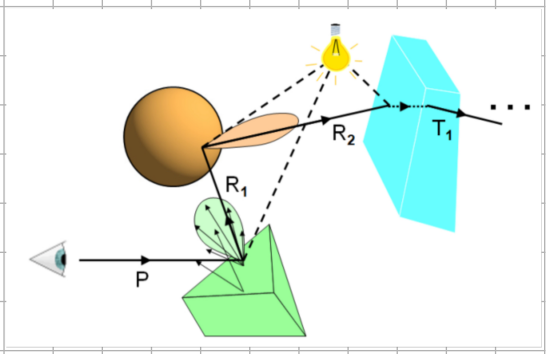
```
select patch i with highest Ai*ΔBi
FOR selected patch i {set up hemicube
                      calculate form factors Fij   }

FOR each patch j { Δrad := ρj*ΔBi*Fij*Ai/Aj
                   ΔBj  := ΔBj + Δrad
                   Bj   := Bj + Δrad     }
ΔBi := 0
```

Aspekte von Radiosity

Radiosity = blickpunktunabhängige Methode zur Helligkeitsberechnung

Discontinuity-Meshing = Methode, um passende Patchgröße zu wählen

Path Tracing



Erweiterung von Ray-Tracing, bei dem die Richtung der Sekundärstrahlen zufällig gewählt wird ⇒ Inkludiert Fälle wie diffuse Reflexion und ausgedehnte Lichtquellen

Photon Mapping

Verfolgung der Lichtstrahlen von Lichtquellen aus in Vorwärtsrichtung

Vorteil: ermöglicht korrekte Lichtberechnung in schwierigen Situationen

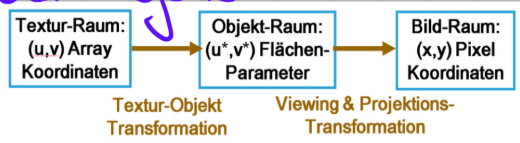
Environment Mapping

1. Umgebung von zentralem Punkt aus in Bild umwandeln mit der

Annahme, dass die Umgebung verhältnisweise groß ist
2. Annahme, dass das Objekt ungefähr im Mittelpunkt liegt

Texture Mapping

1. Festlegung der Orientierung & Skalierung der Objekte
2. Textur auf das Objektabbild rendern



Erzeugung einer Textur

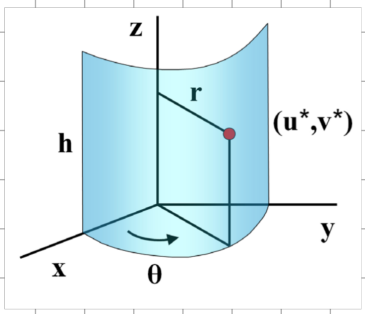
"Procedural Texturing": Texturen aus mathematischen Funktionen erstellen

Textur-Objekt-Transformation

bilineare Funktion zum Aufbringen einer Textur: $u^* = u^*(u,v) = a_u u + b_u v + c_u$

$v^* = v^*(u,v) = a_v u + b_v v + c_v$

Viewing und Projekttransformation



- arbeiten in umgekehrter Richtung
- für jeden Punkt bestimmen, was gezeichnet wird
- daraus welches Texturpixel gültig ist

Anti-Aliasing für Texturen

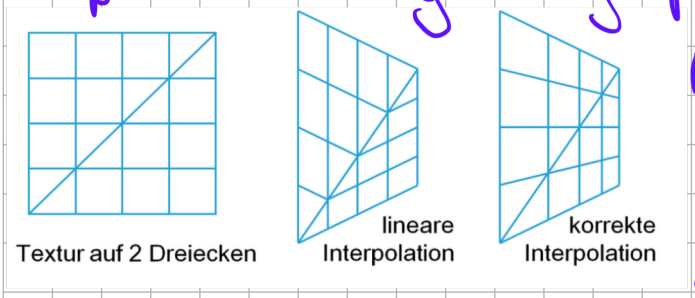
Mip-Mapping: Vorberechnung in versch. Größen, danach Interpolation

Summed-Area-Table Methode: Ermittlung der Durchschnittwerte rechteckiger Bereiche durch Differenzbildung

Texturen auf perspektivisch verzerrten Dreiecken

Verwendung baryzentrischer Koordinaten ⇒ lineare Interpolation

Interpolation vor Homogenisierung ⇒ perspektivisch korrekte Interpolation



Berechnung des Texturwertes:

$d = h_1 h_2 + h_2 \beta (h_0 - h_1) + h_1 \gamma (h_0 - h_2)$

$u = \alpha_w u_0 + \beta_w u_1 + \gamma_w u_2$; $\beta_w = \frac{h_0 h_2}{d}$, $\gamma_w = \frac{h_0 h_1 \gamma}{d}$

$v = \alpha_w v_0 + \beta_w v_1 + \beta_w v_2$; $\alpha_w = 1 - \beta_w - \gamma_w$

$color(x, y) = t(u, v)$

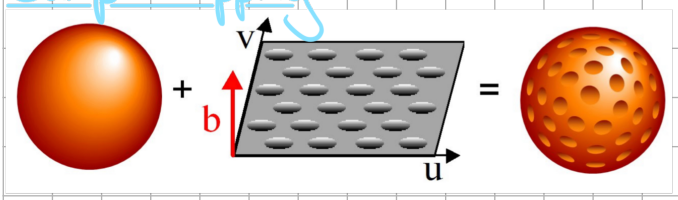
Solid Texturing

Input: Textur in Form eines 3D-Volumens, entweder durche eine mathematische Funktion oder Volumendaten

Vorteil: es kann keine Zusammenfügungsprobleme geben

Abildung ist wesentlich leichter zu handhaben

Bump Mapping



Ziel: Eindruck von Unebenheiten erwecken

hier meistens $b'(u,v)$ nötig

Formel: $n' = n + b_v (p_u \times n) + b_u (n \times p_v)$, $b(u,v)$ = Verschiebung

$p(u,v)$ = Punkt, der verschoben wird, $n = \frac{n^*}{|n^*|}$ mit $n^* = p_u \times p_v$

Fehler:
- starke Strukturverzerrung bei kleinen Winkeln
- Silhouette bleibt glatt (keine "echten" Bumps vorhanden)
- Objekt wirft Schatten mit falschen Rändern
- ev. falsche Lichtverteilung

Displacement-Mapping

= tatsächliches Erhöhen der Bumphöhe ⇒ korrekte Silhouette

Nachteil: aufwendigere Implementierung

Multitexturing

= Kombination mehrerer Mappings

Quadratische Flächen

implizite Repräsentation: alle Punkte, die eine Formel erfüllen, liegen auf der Oberfläche

explizite Repräsentation: Koordinatenwert wird abhängig von den anderen dargestellt

parametrische Repräsentation: Erzeugung eines Punktes der Oberfläche für jede Kombi von Parameterwerten

Kurven

⇒ angegeben durch Formel oder Menge Stützpunkten/Kontrollpunkten

charakteristische Eigenschaften verschiedener Kurventypen:

- interpolierend vs. approximierend (Kurve schneidet Stützpunkte (nicht))
- Stetigkeitsgrad an Verbindungen
- globaler vs. lokaler Einfluss (welchen Bereich die Punkte beeinflussen)
- achsenabhängige vs. -unabhängige Darstellung (Drehung des Koordinatensystems verändert die Kurve (nicht))

Kubische Spline-Interpolation

kubischer Spline = Interpolationskurve, die zwischen zwei Stützpunkten aus einem kubischen Polynom besteht

Kurve $p_k(u) = a_k u^3 + b_k u^2 + c_k u + d_k$ mit $k = 0, 1, 2, 3, \dots, n-1$; $0 \le u \le 1$

natürliche kubische Splines: zweimal stetig differenzierbare k. Splines

Nachteil: jeder Spline hat globalen Einfluss

Hermite-Interpolation

spezielle Form der kubischen Splines

4 Bestimmungsstücke: $p_k(0) = p_k$, $p_k(1) = p_{k+1}$, $p_k'(0) = Dp_k$, $p_k'(1) = Dp_{k+1}$

$p_k(u) = a_k u^3 + b_k u^2 + c_k u + d_k \Rightarrow p_k'(u) = 3a_k u^2 + 2b_k u + c_k$

$$p_k(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot \begin{bmatrix} a_k \\ b_k \\ c_k \\ d_k \end{bmatrix}, \quad p_k'(u) = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_k \\ b_k \\ c_k \\ d_k \end{bmatrix}$$

$$\begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_k \\ b_k \\ c_k \\ d_k \end{bmatrix} \Rightarrow \begin{bmatrix} a_k \\ b_k \\ c_k \\ d_k \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix}$$

$$P_k = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot M_H \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ Dp_k \\ Dp_{k+1} \end{bmatrix}$$

Bézier-Kurven

Kurvenpunkt = gewichtetes Mittel aller Kontrollpunkte: $p(u) = \sum_{k=0}^{n} p_k b_{k,n}(u)$

$b_{k,n}(u) = \binom{n}{k} u^k (1-u)^{n-k}$; n = Anzahl der Stützpunkte, k = betrachteter St.pkt.

$b_{k,n}(u) > 0$ außer für $u = 0$ und $u = 1$

Eigenschaften:
- bei n+1 Stützpunkten ist $p(u)$ vom Grad n
- Kontrollpunkte ziehen die Kurve mit
- globaler Einfluss
- p_0 und p_n liegen auf der Kurve
- Tangenten verbinden p_0 und p_1 & p_n und p_{n-1}
- Kurve liegt in konvexer Hülle der Kontrollpunkte

B-Spline-Kurven

approximierende Kurven, die B-Spline-Polynome $B_{k,d}$ verwenden

Berechnung komplexer & rekursiv, jedoch gilt: $\sum_{k=0}^{n} B_{k,d}(u) = 1$

Falls $d = n+1$ (d = Anzahl der Kontrollpunkte) ⇒ Bréziekurven

Hauptunterschiede dazu:
- lokaler Einfluss der Kontrollpunkte
- linearer Aufwand zur Anzahl von diesen

wichtigste Erweiterung: Non-Uniform Rational B-Splines (NURBS)

⇒ erlauben konsistente Repräsentation geometrischer Formen

Brézier- und B-Spline-Flächen

Freiformfläche $p(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} p_{j,k}\, b_{j,m}(v)\, b_{k,n}(u)$

Randkurven = Brézier-/B-Spline-Kurven

Übertragung der Eigenschaften auf die Fläche

zeichnen durch Ray-Casting oder B-Reps möglich