Verdeckungsrechnung

Vom CAD-Modell zum Bild

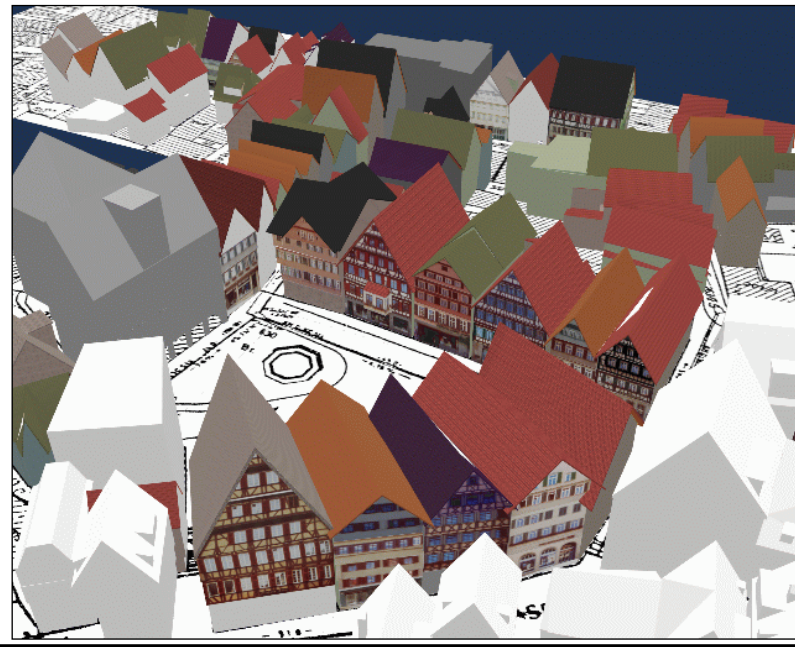
Bisher:

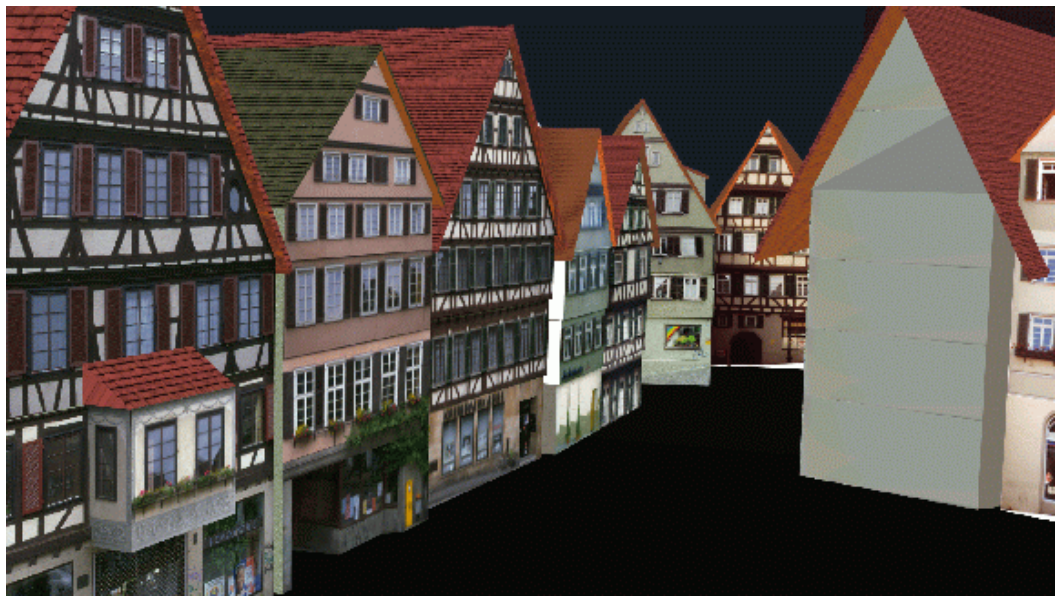
- Rigide Transformation
- Perspektivische Projektion
- Beleuchtungsmodell
- -> zeichne Dreiecke in Pixelraster

Zu berechnen bleiben:

- Verdeckungen
- Schlagschatten

Beispiel: Virtual Tübingen Projekt

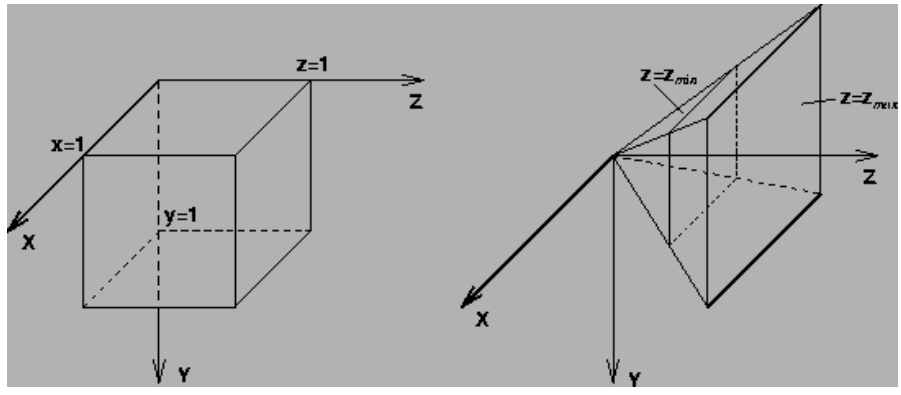




Vorauswahl sichtbarer Flächen

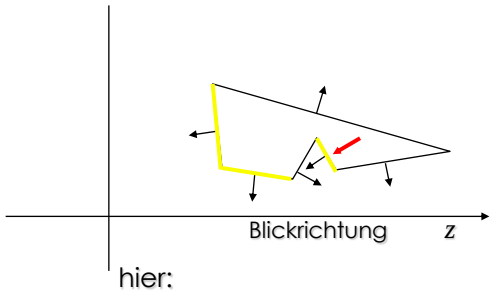
Einschränken des Sichtvolumens

Für Parallel- und Perspektivische Projektionen wird das Sichtvolumen auf Minimale und Maximale Werte (cilpping planes) beschränkt.



Eliminierung der Rückseiten (***back-face culling***)

Bei nicht durchsichtigen Objekten ist die Rückseite nicht sichtbar!



Vorzeichen des Skalarprodukts zwischen $< Normalen , Sehstrahl >$ unterscheidet Vorder- und Rückseite.

hier:

Für $< z , n > \geq 0$ handelt es sich um eine Rückseite

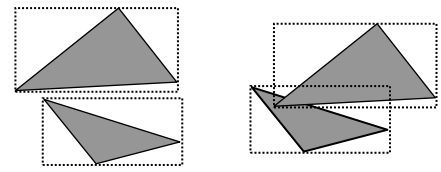
Nicht alle Vorderseiten sich sichtbar,

sie können auch von anderen Vorderseiten verdeckt sein!

Min / Max-Test

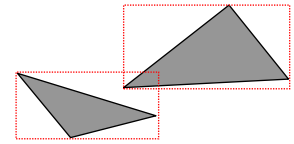
Verdeckungen zwischen zwei Objekten können nur dann auftreten, wenn sie sich in der x,y-Bildebene überlappen.

Mit einem ‚Bounding- Box' -Test lassen sich viele nicht überlappende Objekte erkennen.



Einfach, weil nur jeweils für x,y getrennt ein eindimensionaler Test der Minima und Maxima notwendig ist.

Die Methode ist nur eine Vorauswahl, da aus dem Überlappen der Bounding-Box nicht immer ein Überlappen der Objekte folgt.

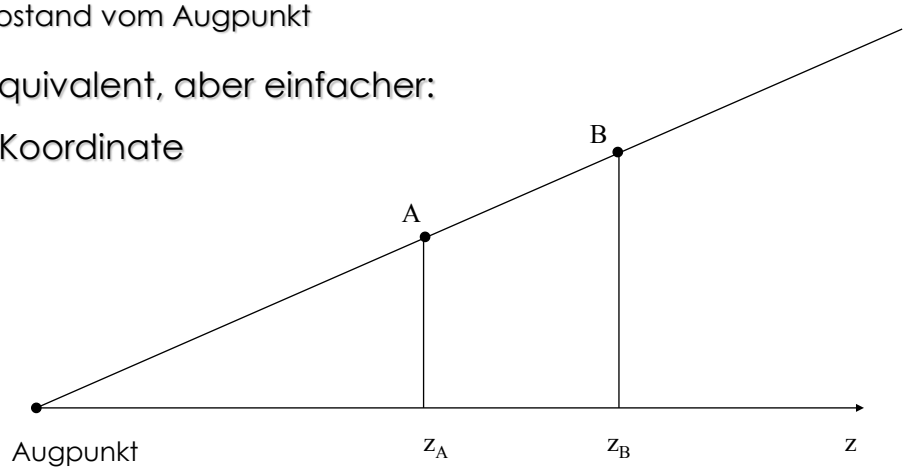


Verdeckungskriterium

Abstand vom Augpunkt

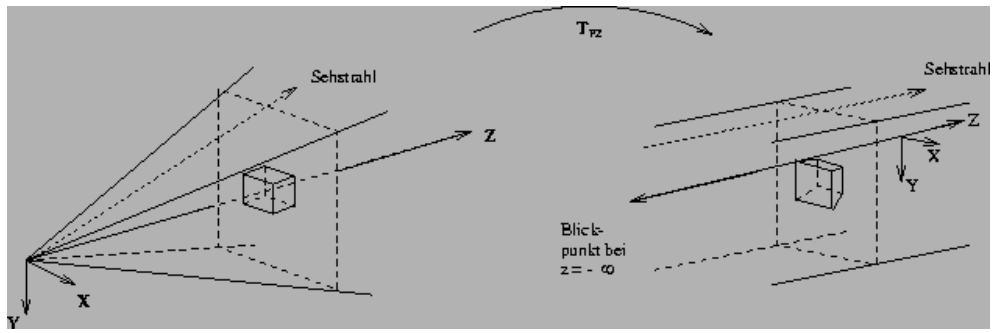
Äquivalent, aber einfacher:

z-Koordinate



Perspektivische Transformation

Für alle Verfahren liefert die Perspektivische Transformation als Vorverarbeitung die Bildkoordinaten und die Tiefenwerte z.



=> dadurch reduziert sich das Problem auf einen reinen Tiefenvergleich.

Objekt versus Bild bezogener Tiefentest

Eine Szene bestehe aus n Objekten (Dreiecke).

A)

für alle Objekte n {
  bestimme alle sichtbaren Teile;
  zeichne die sichtbaren Teile;
}

etwa $n \cdot n$ Operationen

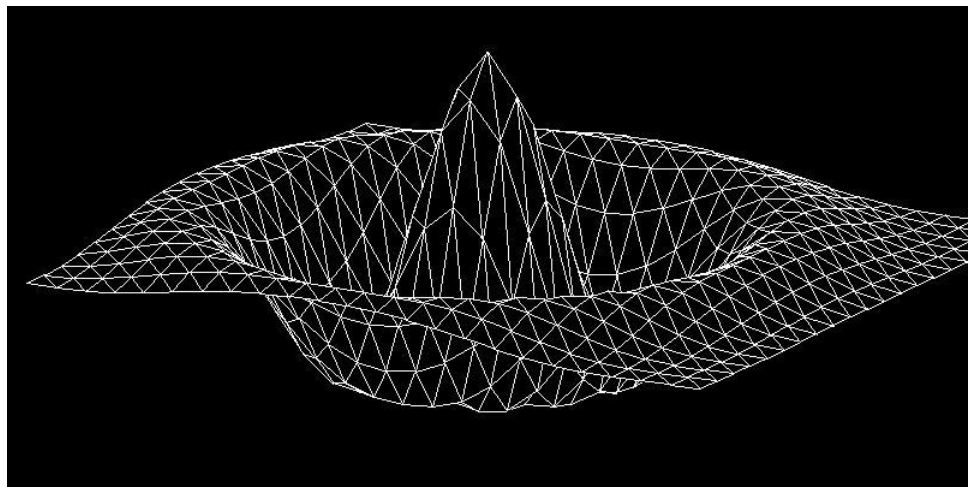
B)

für jeden Bildpunkt p {
  bestimme das zum Beobachter nächste Objekt;
  zeichne den Bildpunkt;
}

etwa $n \cdot p$ Operationen

Objektbezogene Verfahren

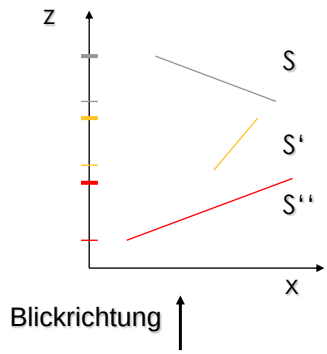
A) Tiefensortierung bei Parametrisierten Flächen



Einfaches Verfahren, wenn die Daten schon in geeigneter Form vorliegen wie z.B. beim plotten von Funktionen. Hier wird von **vorn nach hinten** gezeichnet und für jede Bildspalte x ein y_{min} und y_{max} ein Wert gespeichert.

B) Tiefensortierung mit „*Painter's algorithm*"

1. Sortiere Objekte nach ihrem maximalen z-Wert.

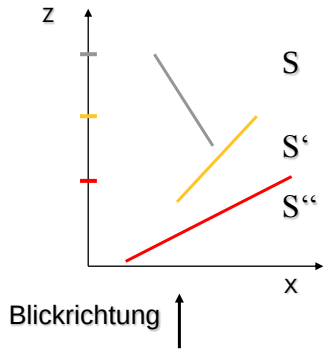


Falls keine Überlappung in der Tiefe auftritt

=> Zeichne alle Objekte von **hinten nach vorne**.

Tiefensortierung (*Painter's algorithm*)

Für allgemeine Objekte:
Sortiere Polygone nach maximalem z-Wert,
also dem am weitesten entfernten Punkt.



Painter's algorithm (2)

Gehe die Polygone von hinten nach vorne durch.

Vergleiche S mit allen anderen Flächen S‘.

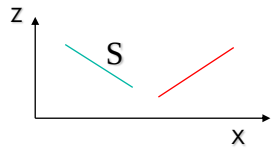
S		S´					

4 weitere Tests die ausschließen, dass das Polygon ein nachfolgendes verdeckt.

1.) Sobald ein Kriterium erfüllt ist:	Zeichne S
2.) Alle Kriterien verletzt:	Umsortieren erforderlich.

Painter's algorithm (3)

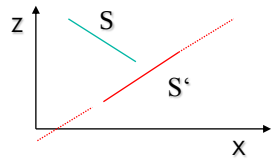
Test 1.) Min / Max -Test in der Bildebene durchführen.



Nebeneinander:

=> S kann S' nicht verdecken.

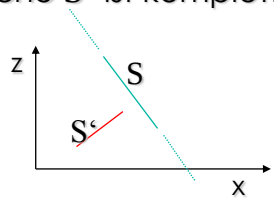
Test 2.) die Fläche S ist komplett <u>hinter</u> S' mit der sie überlappt.



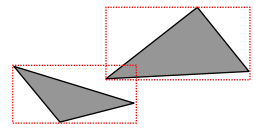
Alle Eckpunkte von S werden in die Ebenengleichung von S' eingesetzt und dann das Vorzeichen ausgewertet.

Painter's algorithm (4)

Test 3.) die Fläche S' ist komplett <u>vor</u> S mit der sie überlappt.



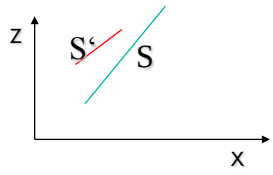
Test 4.) Die Projektionen der Flächen überlappen nicht in der Bildebene.



Was bleibt ------- >

Painter's algorithm (5)

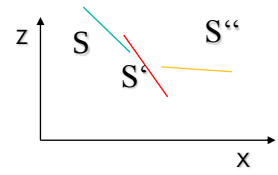
Einige einfache Situationen:



S verdeckt also S'.

Vertausche die Reihenfolge.

oder



Tausche S mit S'.

Genügt nicht!

Tests für S' :

Tausche S' mit S''.

Zeichne erst S'', dann S', dann S.

Was bleibt ------- >

Painter's algorithm - The End

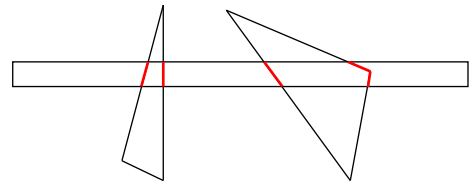
Keine einfache globale Lösung ---

lokale Methoden notwendig!

C) Scan-line algorithm (Watkins 1970)

Das Gesamtbild ergibt sich als Summe der Zeilen.

⇒ Verdeckungsrechnung wird für einzelne Zeilen gelöst, unter Nutzung von nach *X,Y* geordneter Kantenlisten.



1. Erstellen einer Kantenliste.
2. Erstelle für jede Rasterlinie die aktive Kantenliste.
3. Für jede einzelne Zeile die Überdeckungen der einzelnen Segmente bestimmen und dann die Bereiche ohne Visibilitäts-änderung zusammensetzen.

Bildpunktbezogene Verfahren

Tiefenspeicher (*z-buffer algorithm 1974*)

Für jeden Rasterpunkt x,y wird zusätzlich, außer dem Bildspeicher $B(x,y)$ der die Farbintensitäten i speichert, noch ein Tiefenspeicher $T(x,y)$ für $z(x,y)$ angelegt (16 - 32 bit).

Wolfgang Strasser, in seiner Habilitationschrift (1974) erster Hinweis auf Z-buffer Methodik.

1.) Initialisieren: für alle x,y

$B(x,y)$ = Hintergrundfarbe

$T(x,y) = Zmax$ (maximal vom Beobachter entfernt).

2.) Alle Objekte *rendern* (rastern, scanline-konvertieren)

$z(x,y)$ berechnen;

nur falls $z(x,y) < T(x,y)$

==> $T(x,y) = z(x,y)$ und $B(x,y) = i\,(x,y)$

z-buffer algorithm (2)

Vorteil: Bildpunkte können in beliebiger Reihenfolge erzeugt werden.

==> Nachteil: pro Bildpunkt wird nur der z-Wert für ein Objekt gespeichert. Somit sind u.a. Antialiasing und Transparenz nicht realisierbar.

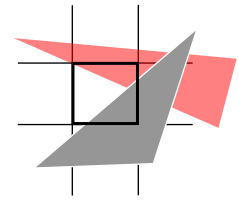
weitere Vorteile:

1.) Beliebig komplexe Szenen lassen sich darstellen.

2.) Neue Objekte lassen sich in fertige Szenen integrieren.

Der **A**ccumulation-Buffer (das Konzept)

Immer wenn Objektstrukturen in der Größenordnung der Rasterung sind, kommt es zu Fehlern.



Ideal wäre: Farben je nach ihrem Flächenanteil zu mischen.

$$I = I_1 \frac{A_1}{A_{gesamt}} + I_2 \frac{A_2}{A_{gesamt}}$$

$$= I_1\alpha_1 + I_2\alpha_2 \quad \text{mit} \quad \alpha_1 + \alpha_2 = 1$$

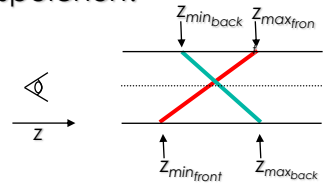
oder allgemein $I = \sum_{i=1}^{n} I_i\alpha_i$ mit $\sum_{i=1}^{n} \alpha_i = 1$ α-Blending

A-Buffer (eine Approximation)

Zum Vergleich, beim z-Buffer wird für den gesamten Pixel die Farbe des vordersten Objekts übernommen, wobei die Position nur in der Mitte des Pixels berücksichtigt wird.

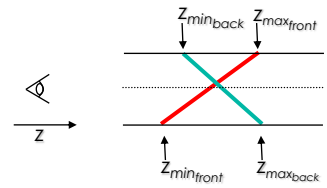
A-Buffer Implementierung:

Für jeden Pixel wird eine Liste der Fragmente erstellt und für jedes Fragment z_{min} und z_{max} gespeichert.



A-Buffer (eine Approximation 2 historisch)

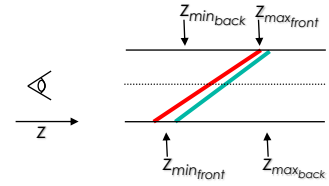
Wie groß ist der Flächenanteil des vorderen Segments?



Approximation der sichtbaren Fläche

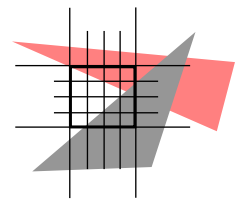
$$==> \quad A_{vis_{front}} = \frac{z_{max_{back}} - z_{min_{front}}}{(z_{max} - z_{min})_{front} + (z_{max} - z_{min})_{back}}$$

ABER: Falsches Ergebnis bei:



A-Buffer (der Ausweg ?)

Für eine höhere Genauigkeit wird mehr Information über die Form der einzelnen Fragmente innerhalb des Pixels benötigt !



Es muß in subpixel Genauigkeit gerechnet werden.

Hierzu kann dann einer der bekannten Algorithmen wie Z-Buffer oder eine Scan-Line-Methode verwendet werden.

Schattenwurf

Schatten

Schlagschatten:
Lichtquelle durch anderes Flächenelement verdeckt.

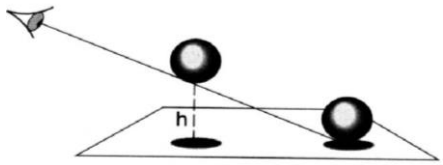
Nicht lokal!

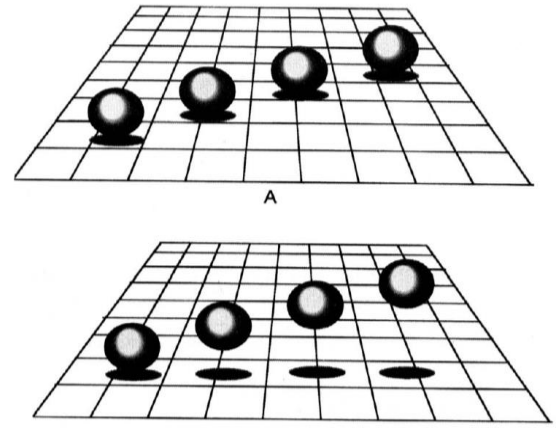
Selbstschatten:
Normale weist von der Lichtquelle weg.

Lokales Kriterium.

Schatten als Hinweis zur Tiefenwahrnehmung

"Depth Cue"





Weiche Schatten

verursacht durch ausgedehnte Lichtquellen

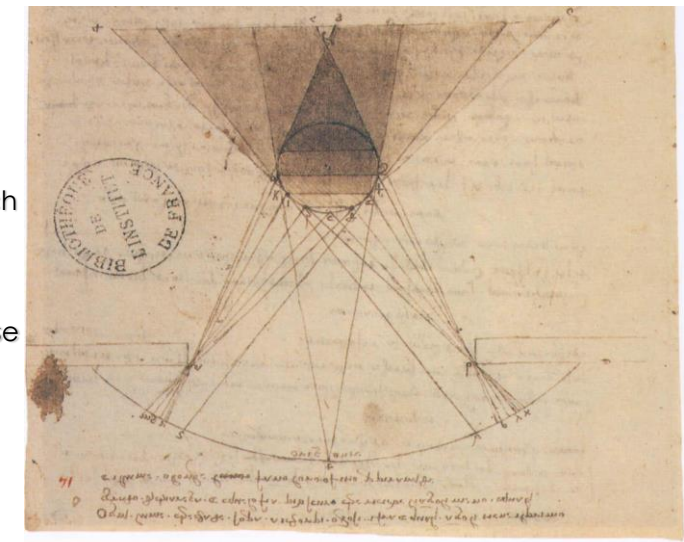
<u>Umbra</u>

- Lichtquelle gänzlich verdeckt

<u>Penumbra</u>

- Lichtquelle teilweise verdeckt

<u>Gänzlich beleuchtet</u>



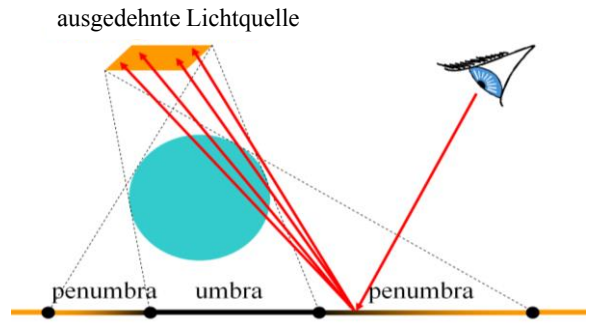
XVI. Léonard de Vinci (1452-1519). Lumière d'une fenêtre sur une sphère ombreuse avec (en partant du haut) ombre intermédiaire, primitive, dérivée et (sur la surface, en bas) portée. Plume et lavis sur pointe de métal sur papier, 24 x 38 cm. Paris, Bibliothèque de l'Institut de France (ms. 2185 ; B.N. 2038, f° 14 r°).

Schatten durch Ray Tracing

Für jeden sichtbaren Punkt integriere alle Strahlen über die gesamte Lichtquelle unter Berücksichtigung der Verdeckung der Lichtquelle.

Optimierung?

- Abbrechen sobald ein erstes verdeckendes Objekt (occluder) gefunden ist.
- Kohärenz : speichere den letzten "occluder" und teste dieses Objekt zuerst.



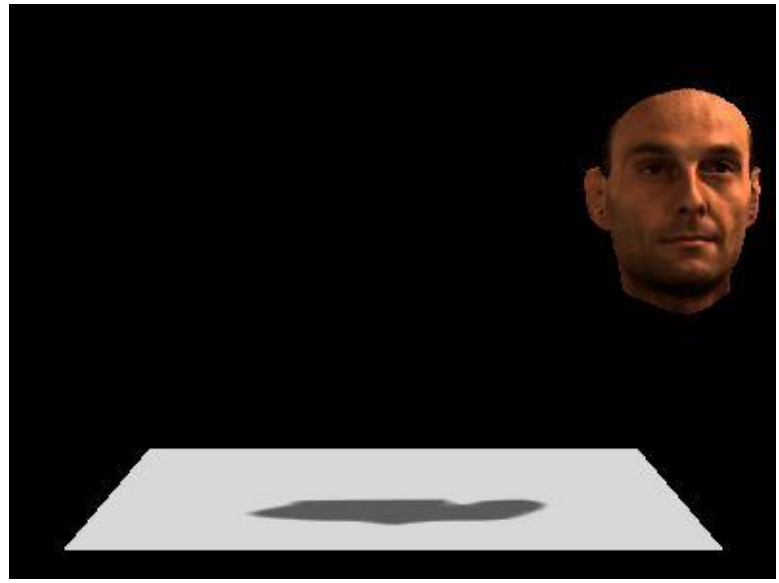
Schlagschatten auf einer Ebene

Erzeuge eine Kopie des 3D-Objektes,

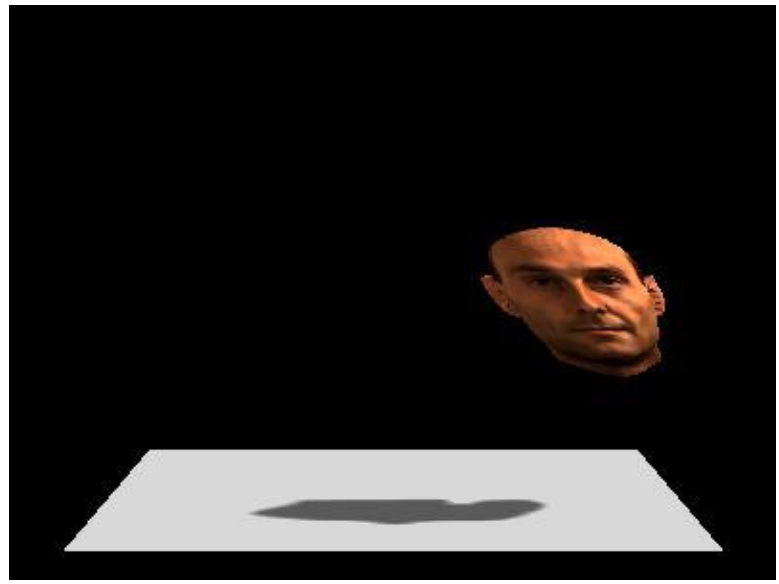
projiziere Punkte der Kopie auf die Ebene,

zeichne die Polygone dort in der Farbe des Schattens.

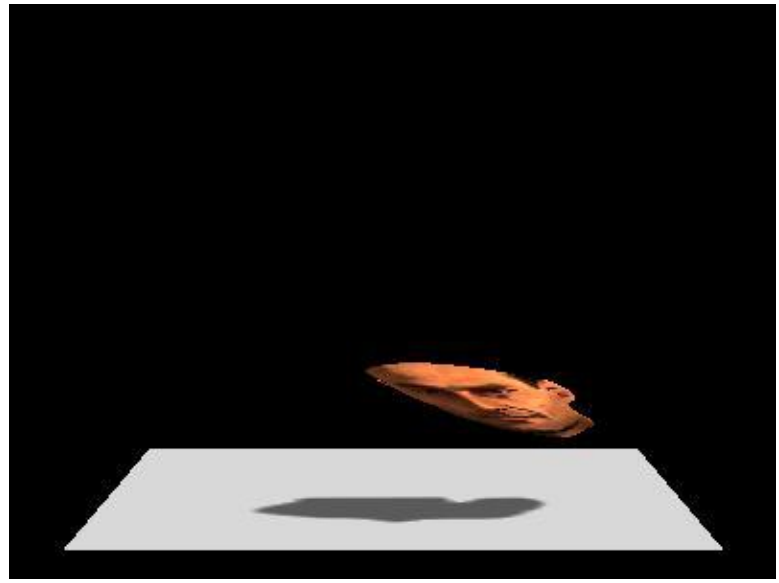
Schlagschatten auf einer Ebene



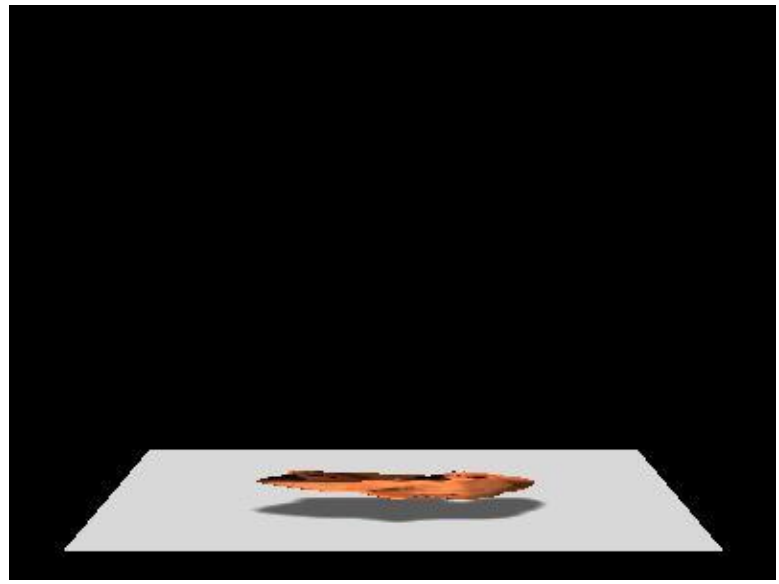
Schlagschatten auf einer Ebene



Schlagschatten auf einer Ebene



Schlagschatten auf einer Ebene



Schlagschatten auf einer Ebene

Ebene:	$\langle n,v \rangle + b = 0$
Betrachte Punkt	p
Beleuchtungsrichtung	L
Geradengleichung	$v = p + t\,L$
Eingesetzt:	$\langle n, p + t\,L \rangle + b = 0$
Berechne	t
	$\rightarrow \quad v = p + t\,L$

Kann als affine Transformation geschrieben werden,

Matrixmultiplikation in homogenen Koordinaten.

Schlagschatten auf gekrümmten Flächen

"Sieht" die Lichtquelle einen Bildpunkt ?

Punktlichtquelle:

Perspektivische Projektion

Paralleles Licht:

Orthographische Projektion

Zweifach z-Buffer Algorithmus (Williams 1978)

(1) Berechne z-Buffer aus der Sicht der Lichtquelle: „Shadow Map“.

(2) Zeichne Bild aus der Sicht des Betrachters:
Prüfe Sichtbarkeit mittels Standard z-Buffer.

-> Wenn sichtbar:
I) Setze z-Buffer
II) Prüfe: Liegt der Punkt im Schatten?

Transformation in Perspektive der Lichtquelle, dann Abfragen der Shadow-Map.

-> Wenn im Schatten: nur ambientes Licht,
kein Lambertscher Strahler,
kein Glanz.

Shadow Mapping

"Texture mapping" mit Tiefeninformation

2x durch die Pipeline

- Berechne ShadowMap (Tiefenkarte aus Sicht der Lichtquelle).
- Zeichne Bild aus der Sicht des Betrachters.

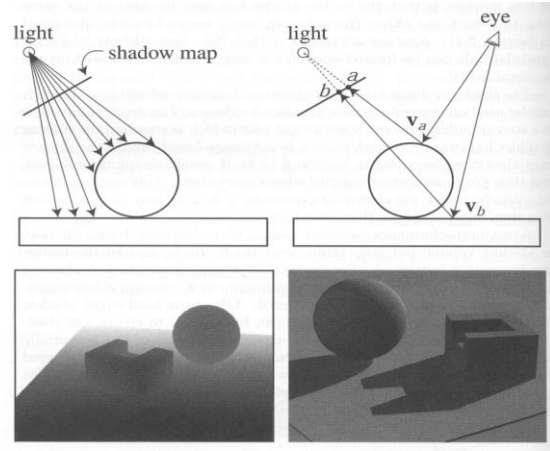


Bild aus Foley et al.
“Computer Graphics Principles and Practice”

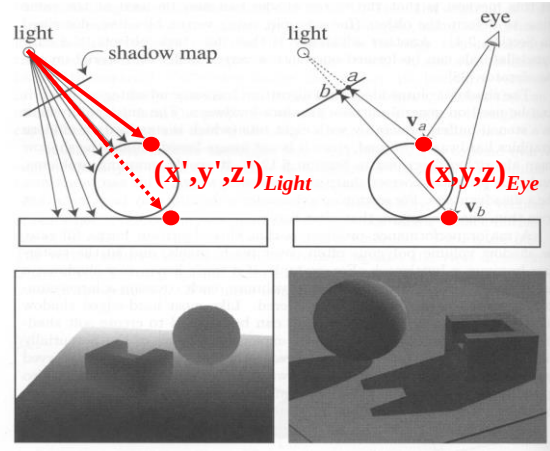
Shadow Map Look Up

Wir berechnen 3D Punkt $(x,y,z)_{Auge}$

Wir bestimmen Tiefe aus der "shadow map"?

Nutze 4x4 perspektivische Projektions-matrix der Lichtquelle
--> $(x',y',z')_{Licht}$

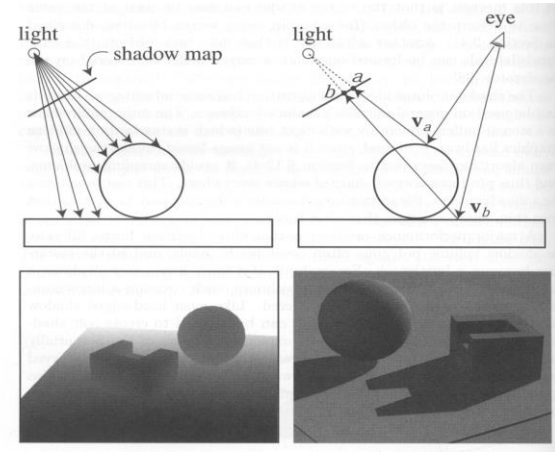
ShadowMap(x',y') < z'?



Foley et al. "Computer Graphics Principles and Practice"

Grenzen der "Shadow Maps"

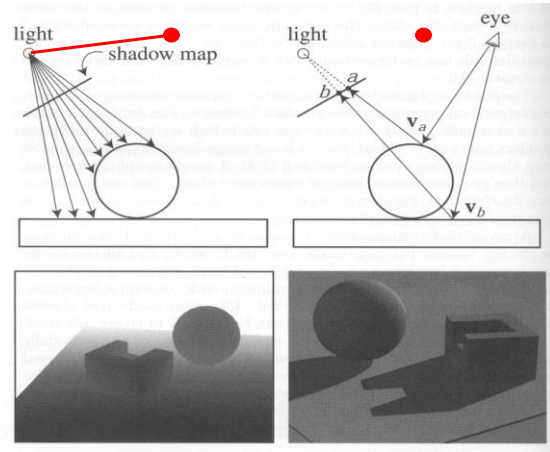
1. Sehfeld
2. Bias (Epsilon)
3. Aliasing



1. Sehfeldproblem

Was passiert wenn der Punkt nicht im Sehfeld der Lichtquelle liegt?

- Nutze eine kubische, rundum `shadow map´ . (welche Auflösung?)



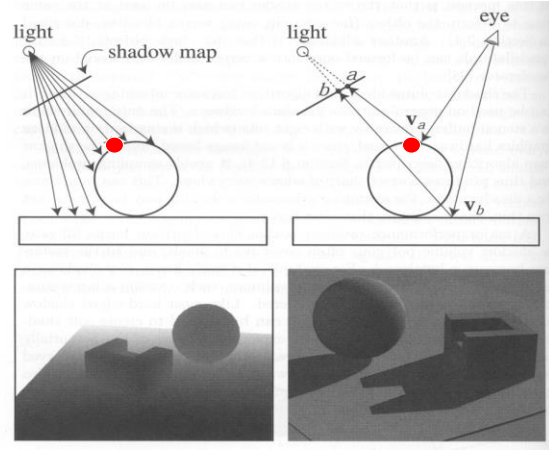
2. Das Bias (Epsilon) Drama

Für einen von der Lichtquelle aus sichtbaren Punkt berechen wir x',y',z' und vergleichen die ShadowMap(x',y') ≈ z' .

Der untersuchte Punkt liegt jedoch selten im Zentrum eines ShadowMapPixels --> Ungenauigkeit.

Wie kann man eine fehlerhafte Eigenbeschattung vermeiden?

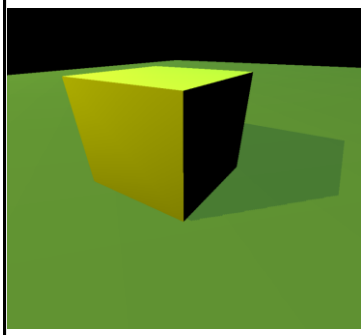
- Addiere einen Bias (epsilon) .



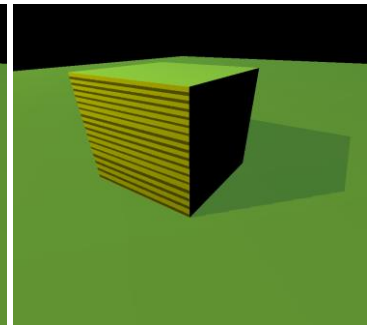
2. Bias (Epsilon) für `Shadow Maps´

ShadowMap(x',y') + bias < z'

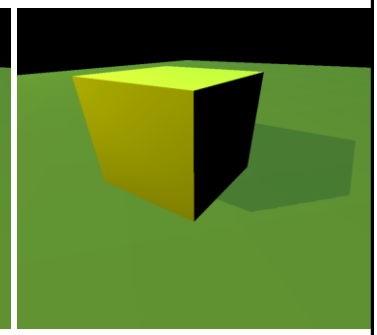
Die richtige Wahl des Bias ist oft schwierig!



das korrekte Bild



zu wenig Bias



viel zu viel Bias

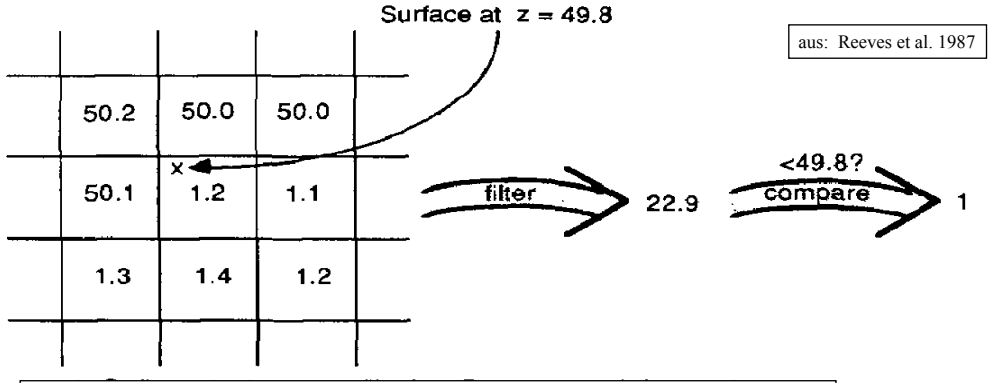
3. Shadow Map Aliasing

Unterabtastung der `shadow map´

Shadow Map Filtering

Sollen die Tiefenwerte gefiltert werden?
(gewichteter Mittelwert benachbarter Tiefenwerte)

Nein... Filtern der Tiefenwerte ist unsinnig!



aus: Reeves et al. 1987

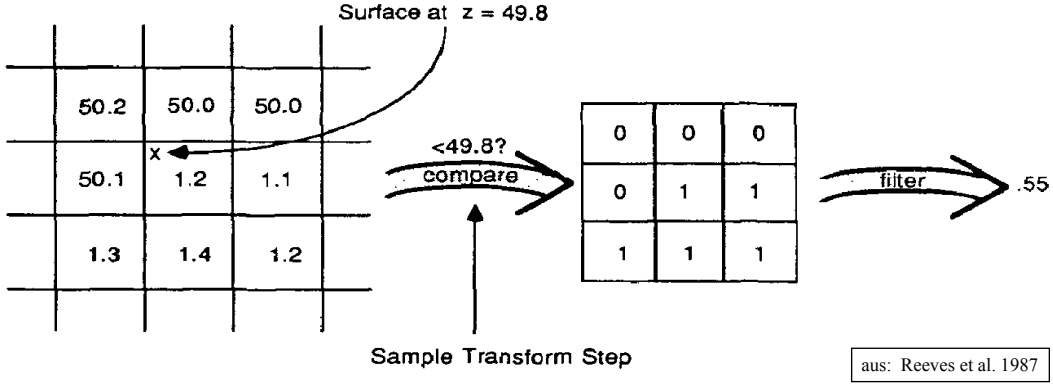
a) Gewöhnliches Filtern der Shadow Map ist KEINE Lösung .

Shadow Map: Percentage Closer Filtering

(Reeves et al. 1987)

Man filtert das Resultat des Schattentests
(gewichteter Mittelwert der Vergleichswerte)

Bias wird noch etwas komplizierter!



aus: Reeves et al. 1987

Shadow Map: Percentage Closer Filtering
(Reeves et al. 1987)

5x5 Abtastwerte

Antialiased Schatten

grössere Filter erzeugen
weiche Schatten

Bias ?

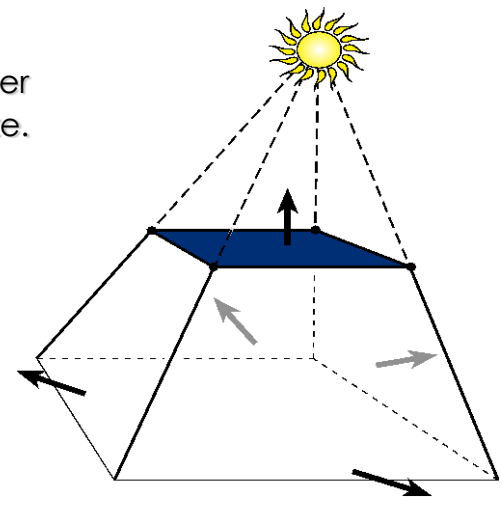
Schatten Volumen "Shadow Volumes"

Explizite Repräsentation des Raumvolumens im Schatten.

Für jedes Ploygon

- Pyramide (Tetraeder) mit der Punktlichtquelle in der Spitze.
- Verwende das Polygon für den Schnitt.

Schattentest vergleichbar mit "clipping„.

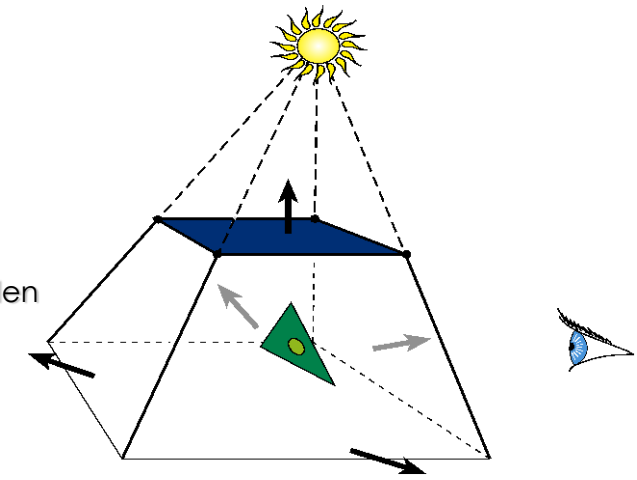


Schattenvolumen

Liegt ein Punkt innerhalb des
Schattenvolumens einer Lichtquelle,
dann erhält der Punkt
von dieser kein Licht.

Naive Implementierung:

#Polygone * #Lichtquellen



Schattenvolumen

Schiesse einen Strahl vom Auge zu
den sichtbaren Punkt

Erhöhe / Erniedrige einen Zähler
jedes Mal wenn ein das
Schattenvolumen
begrenzendes Polygon
geschnitten wird *(z buffer test)*

Ist der Zähler ≠ 0,
liegt der Punkt im Schatten