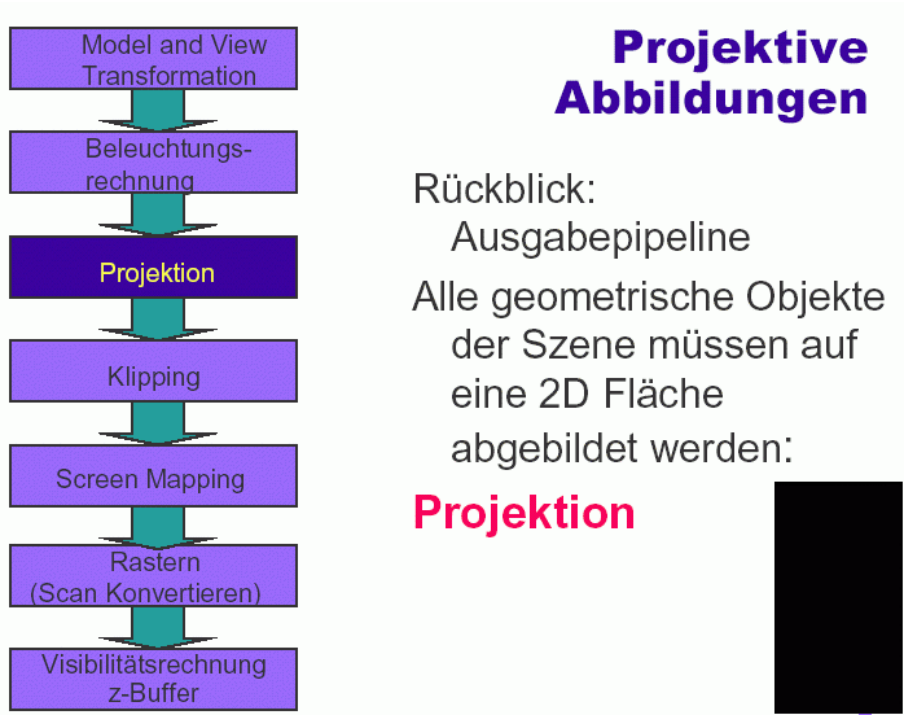
7. Projektionen und Sichtbarkeit

Projektionen: transformieren 3D-Objekte in 2D-Bilder
(mathematisch: lineare Abb., aber nicht bijektiv
$\Rightarrow$ zugehörige Matrix singulär, d.h. Determinante = 0)

Projektion ist Grundaufgabe in der Grafik

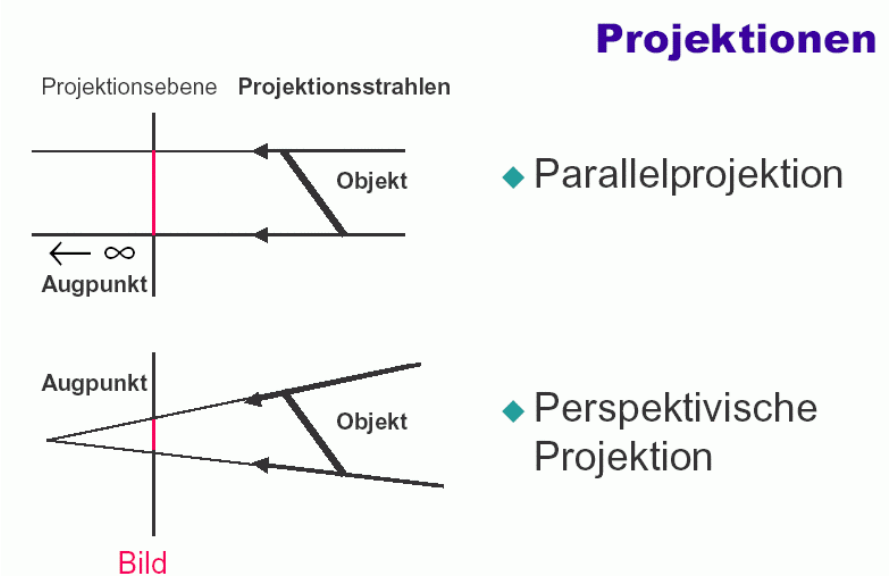


Zeichnen als Projektion:

(Ausschnitt aus Karl Friedrich Schinkel, Die Erfindung des Zeichnens, 1830)

Wir beschränken uns (zunächst) auf *planare* Projektionen:

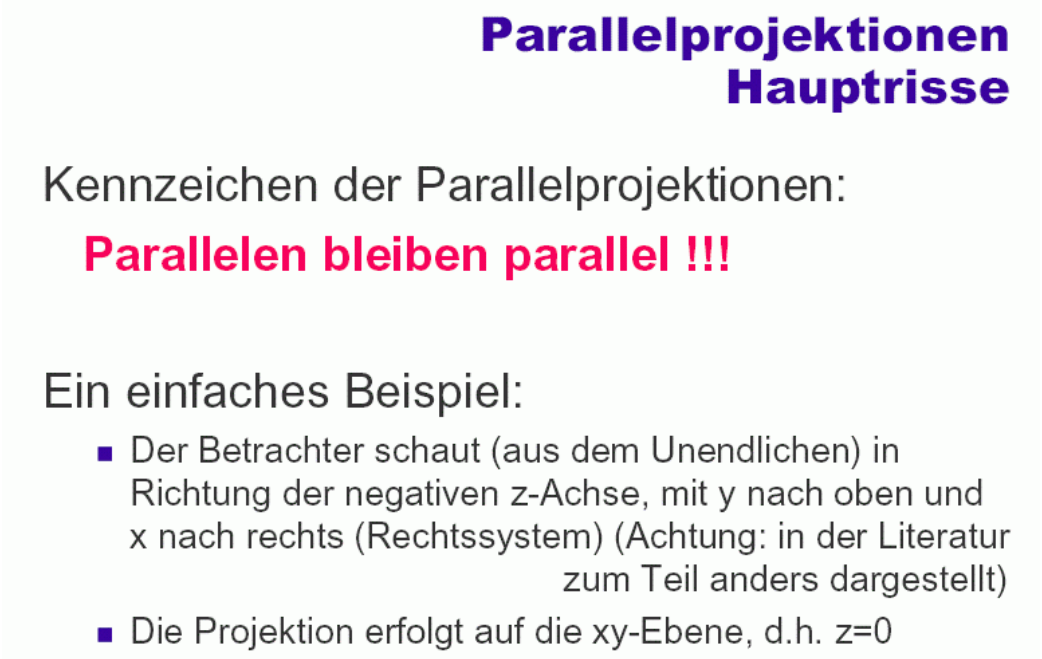
- die Projektionsstrahlen sind Geraden
- die Projektionsfläche ist eine Ebene



Haupttypen planarer Projektionen:

Parallelprojektion:

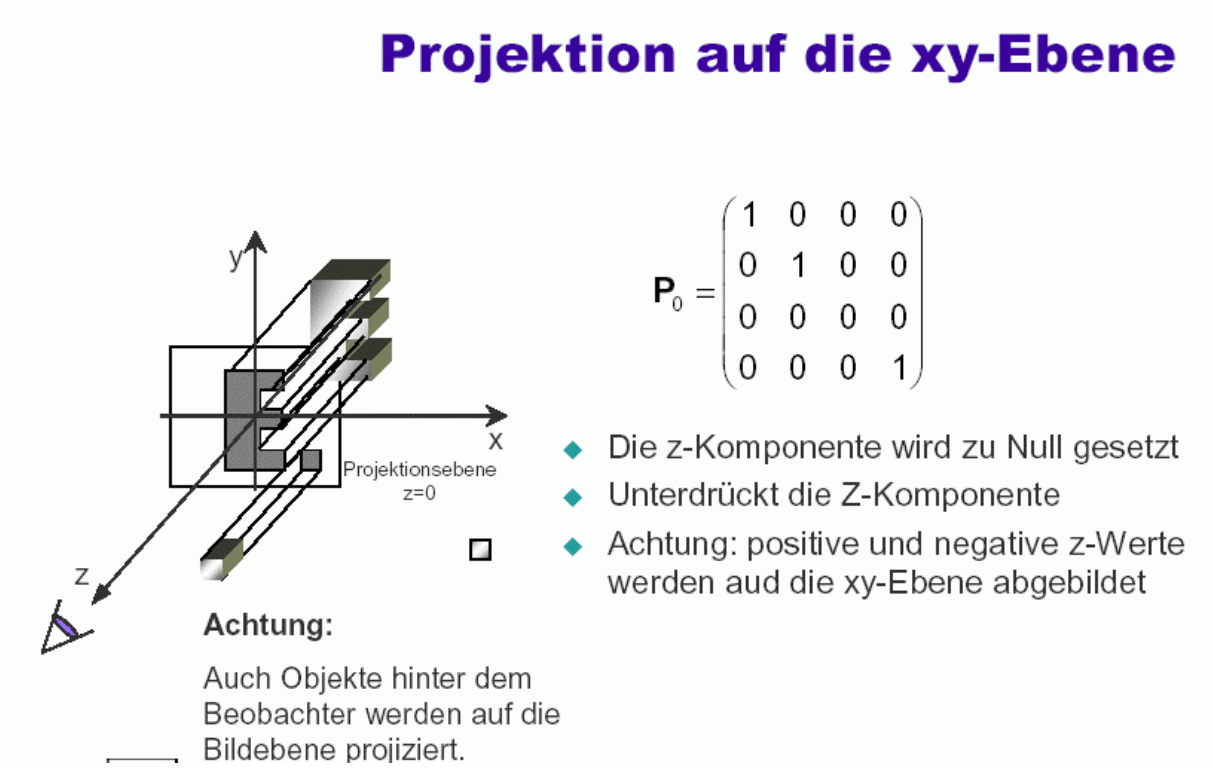
- Projektionszentrum (Center of Projection, COP) liegt im Unendlichen ($\Rightarrow$ ist zu ersetzen durch "Direction of Projection", DOP)
- Projektionsstrahlen verlaufen parallel zueinander
- keine perspektivische Verkürzung



Verwendung der Parallelprojektion:
Ingenieurwesen und Architektur
– kann für Messungen verwendet werden: Längenverhältnisse bleiben erhalten

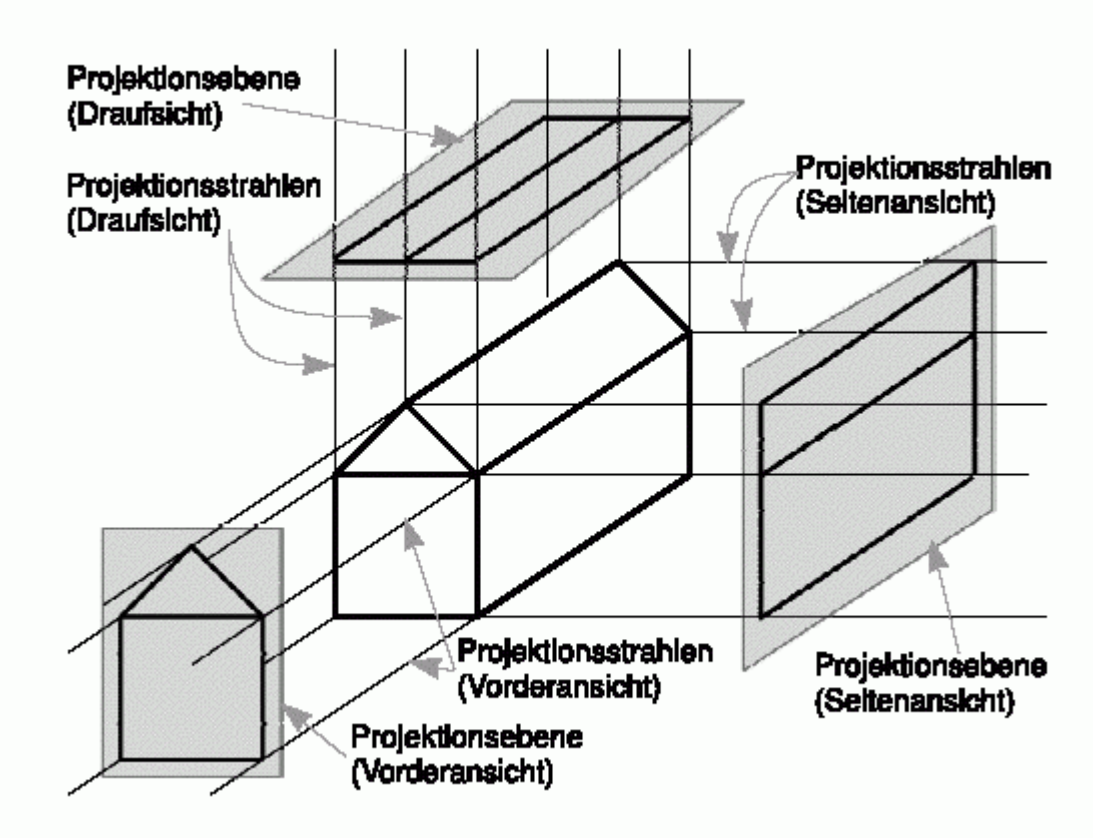
Perspektivische Projektion: imitiert unser Auge oder eine Kamera ⇒ natürlichere Wirkung
Verwendung, wenn das Ziel Fotorealismus ist, u. generell in Kunst u. Design

Parallelprojektion mathematisch:



(Darstellung hier ebenfalls mit homogenen Koordinaten!)

analog für die anderen Koordinatenebenen:
"*Hauptrisse*"
in Grafik- und Konstruktions-Software oft in 3 Fenstern dargestellt (4. Fenster: perspektivische Ansicht des Objekts)
(AutoCAD, Maya, LightWave, 3D Studio Max, Photoshop...)
"*Multiview-Darstellung*".
"Top", "Front", "Side": *Draufsicht*, *Vorderansicht*, *Seitenansicht*.



Hauptrisse: Stimmt die Projektionsrichtung mit einer der Koordinatenrichtungen überein, so erhält man je nach Wahl der Koordinatenrichtung und des Vorzeichens einen der sechs Hauptrisse eines Objekts.
Matrixdarstellung der Projektion auf die Ebene $z = z_0$ (häufig wird $z_0 = 0$ gewählt):

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(zuerst *z*-Koordinate zu 0 machen, dann konstanten Translationsanteil (0; 0; z_0) addieren.)
Analog für die Projektionen auf $x = x_0$ bzw. $y = y_0$.

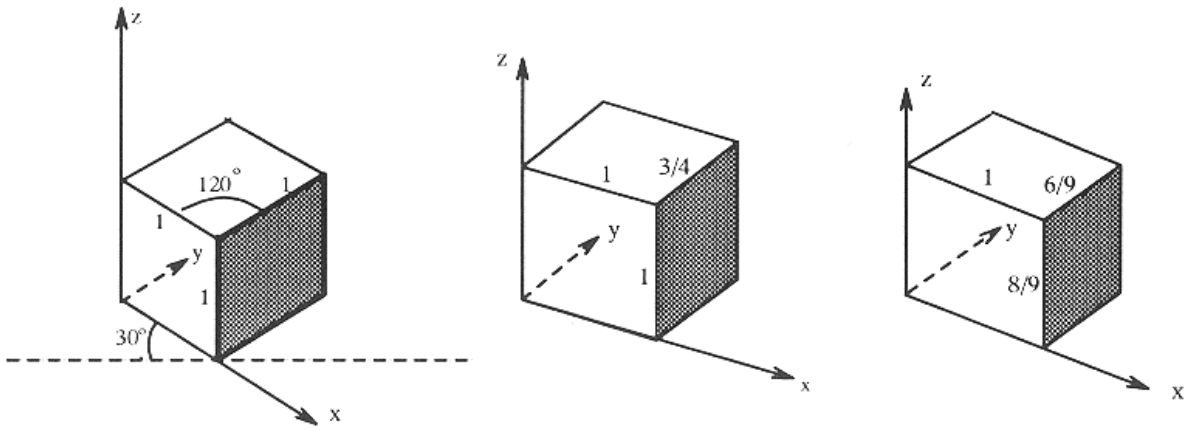
Die Hauptrisse gehören zu den *rechtwinkligen* (auch: *orthografischen*) Parallelprojektionen: die Projektionsstrahlen treffen *senkrecht* auf die Projektionsebene.

rechtwinklige Parallelprojektionen, die keine Hauptrisse sind, heißen "*axonometrisch*".

Kennzeichen:
- Projektionsebene nicht parallel zu einer der Koordinaten-ebenen
- uniforme Verkürzung (unabhängig von der Entfernung)
- Parallelität bleibt erhalten, Winkel nicht

Man unterteilt weiter:

- *isometrische Projektion*: selbe Verkürzung entlang aller drei Achsen
- *dimetrische Projektion*: selbe Verkürzung auf 2 der Achsen, anderer Faktor gilt für die dritte
- *trimetrische Projektion*: unterschiedliche Verkürzungsfaktoren für alle 3 Koordinatenachsen



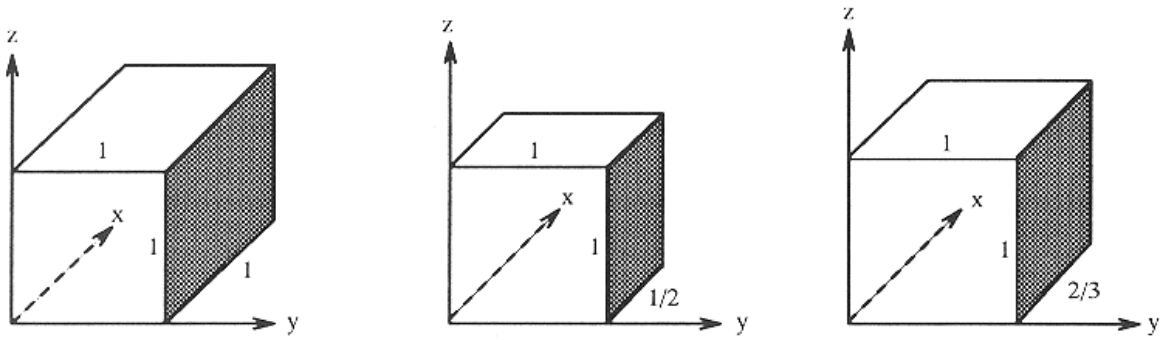
links: isometrische Projektion eines Würfels, Mitte: dimetrische Projektion (selber Verkürzungsfaktor entlang der *x*- und der *z*-Achse), rechts: trimetrische Projektion. Aus Rauber (1993).

Nicht-rechtwinklige Parallelprojektionen heißen *schiefwinklig* (*oblique*): Projektionsstrahlen treffen nicht im rechten Winkel auf die Projektionsfläche auf.

Häufig gebrauchte Spezialfälle:

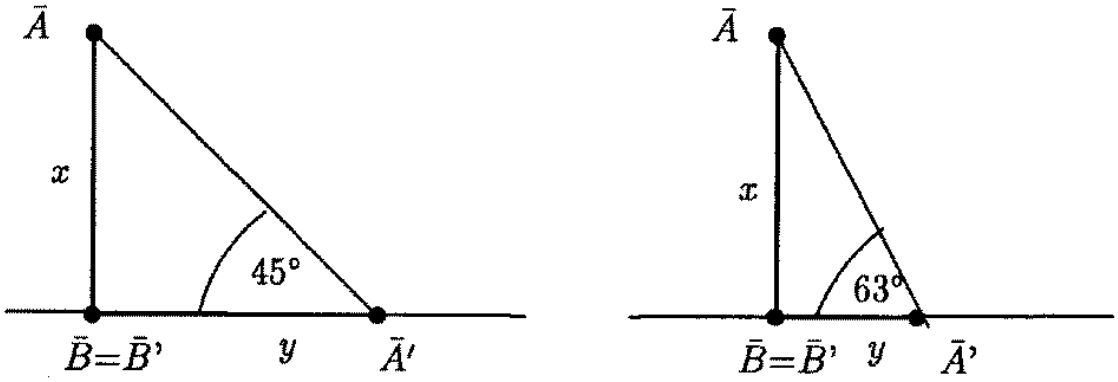
- Kavalierperspektive: Winkel zwischen Projektionsrichtung und Projektionsebene ist 45°. Senkrecht zur Projektionsebene verlaufende Geraden werden ohne Verkürzung wiedergegeben.

- Kabinettperspektive: Winkel 63,4° = arctan(2), Verkürzung senkrechter Geradensegmente um Faktor 1/2.



Darstellung eines Einheitswürfels mit schiefwinkliger Parallelprojektion. Links: Kavalierprojektion, Mitte: Kabinettprojektion, rechts: Variante mit 56°-Winkel der Projektionsstrahlen, entsprechend einer Verkürzung von 2/3 (liefert oft natürlicheres Bild).

Verkürzung der zur Projektionsebene senkrechten Achse:



links: Kavalierprojektion, rechts: Kabinettprojektion.

Matrixdarstellung einer schiefwinkligen Parallelprojektion mit der *xy*-Ebene als Projektionsebene und Projektionsstrahlen mit Richtungsvektor (*rx*, *ry*, *rz*):

$$P_{obl} = \begin{pmatrix} 1 & 0 & -rx/rz & 0 \\ 0 & 1 & -ry/rz & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Perspektivische Projektionen

Vorteile gegenüber Parallelprojektion:
- realistischerer Eindruck der Dreidimensionalität
- bessere (keine perfekte!) Entsprechung zum Foto

Nachteil:
- Längenverhältnisse und Winkel bleiben nicht erhalten (außer bei Objekt-Teilen, die parallel zur Projektionsebene liegen)

Verwendung:
- Präsentationszeichnungen, Werbezeichnungen, Architektur, Kunst, Design

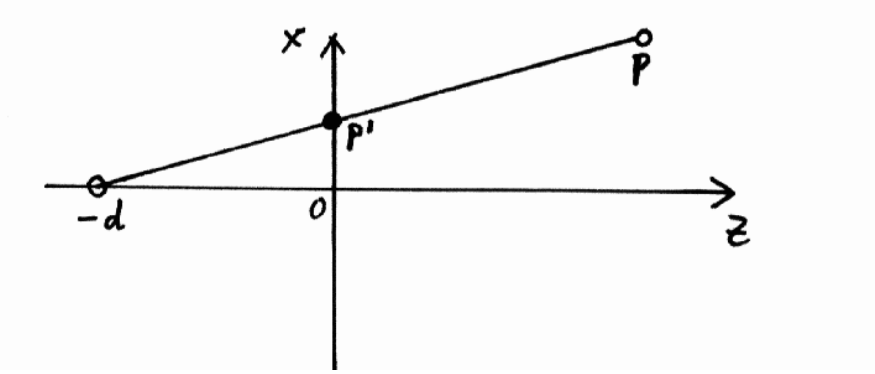
Unterschiede zur Parallelprojektion:
- parallele Geraden, die nicht parallel zur Projektionsebene sind, konvergieren
- Größe von Objekten verringert sich mit dem Abstand
- perspektivische Verkürzung ist nicht einheitlich für alle Abstände

Fluchtpunkte:

je nach Lage des abzubildenden Objekts relativ zur Projektionsebene hat man 1, 2 oder 3 Fluchtpunkte (vanishing points) $\Rightarrow$ 3 Typen von Perspektive

Matrixdarstellung der Zentralprojektion

Urbildraum mit Koordinaten *x*, *y*, *z*
Bildebene: *xy*-Ebene
Projektionszentrum bei $z = -d < 0$ („hinter der Bildebene“)



Bildpunkt $P' = \begin{pmatrix} u \\ v \\ 0 \end{pmatrix}$

$$P = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -d \end{pmatrix} + \alpha \cdot \left(\begin{pmatrix} u \\ v \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ -d \end{pmatrix} \right)$$

$$\Rightarrow \quad z = -d + \alpha d = d(\alpha - 1) \quad \Rightarrow \quad \alpha = \frac{z}{d} + 1 = (z+d)/d, \quad \alpha^{-1} = \frac{d}{z+d}$$

somit (aus den ersten beiden Komponenten der Vektorgleichung):

$$x = \alpha u \quad \Rightarrow \quad u = \frac{x}{\alpha} = \frac{xd}{z+d}$$

$$y = \alpha v \quad \Rightarrow \quad v = \frac{y}{\alpha} = \frac{yd}{z+d}$$

Matrixdarstellung:

$$P_h' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \\ (z+d)/d \end{pmatrix} \equiv \begin{pmatrix} \frac{xd}{z+d} \\ \frac{yd}{z+d} \\ 0 \\ 1 \end{pmatrix}$$

äquivalent ist die Darstellung:

$$P_h' = \begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} xd \\ yd \\ 0 \\ z+d \end{pmatrix} \equiv \begin{pmatrix} \frac{xd}{z+d} \\ \frac{yd}{z+d} \\ 0 \\ 1 \end{pmatrix}$$

Beachte: Um auf kartesische Koordinaten zu kommen, ist im letzten Schritt jeweils eine *Normalisierung* der homogenen Koordinaten durchzuführen (Division durch die 4. Komponente, um dort eine 1 zu erzeugen).

Für $d \to \infty$ nähert man sich der orthogonalen Parallelprojektion an:

$\lim_{d\to\infty} \frac{xd}{z+d} = \lim_{d\to\infty} \frac{x}{1} = x$ (Regel von de l'Hospital), analog für *y*, also

$$P_h' \to \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix}$$ (vgl. Parallelprojektion).

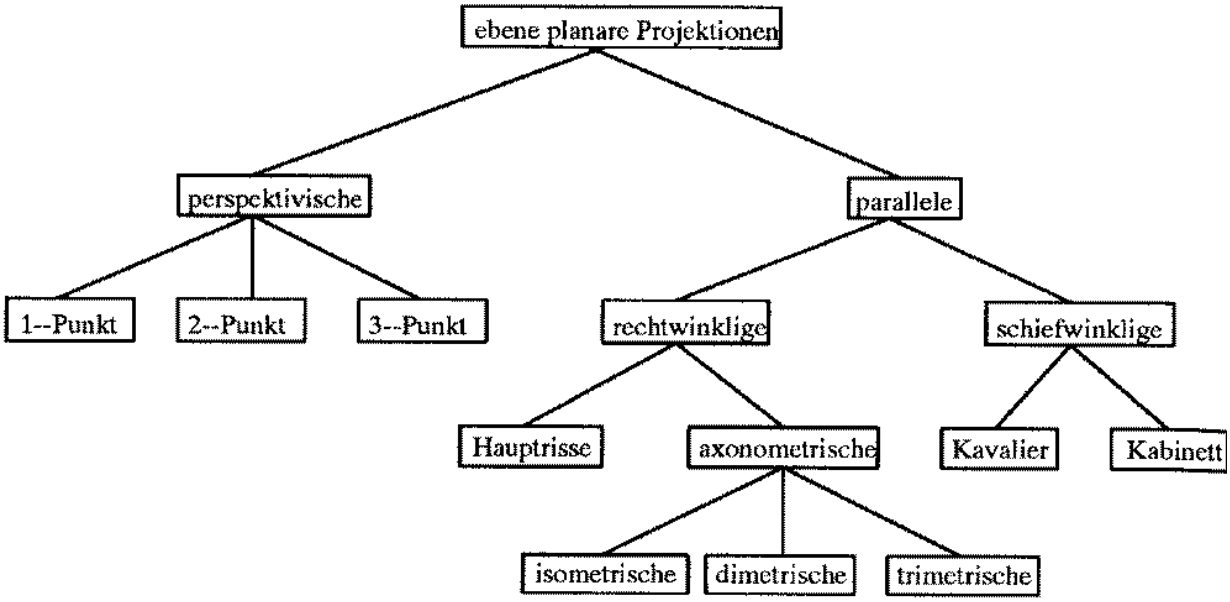
Beispiel:
Das Projektionszentrum liege im Abstand *d*=12 hinter der Projektionsebene, der *xy*-Ebene. Was ist das Bild des Objektpunktes (1; 1; 5)?

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/12 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 17/12 \end{pmatrix} \equiv \begin{pmatrix} 12/17 \\ 12/17 \\ 0 \\ 1 \end{pmatrix}$$

$\Rightarrow$ Der Bildpunkt hat die kartesischen Koordinaten $x = y = 12/17$, $z = 0$.

Projektionen aus beliebigen Richtungen und mit beliebigem Projektionszentrum (Blickpunkt) können durch Translationen und Drehungen in die obige Form überführt werden.

Übersicht:
Typen von Projektionen



Betrachtungstransformationen (*Viewing transformations*)

In der Regel wird bei der Implementierung der Projektion in die Bildschirmebene auch das Clipping vorbereitet.

Anstelle der Projektion auf eine Ebene und einem Clippen in dieser Ebene transformiert man in ein *kanonisches Sicht-volumen* (Sichtkörper, i.allg. ein Quader) und clippt an dessen Seitenflächen.
Dann: Projektion in Ebene auf einheitliche Weise.

Im Folgenden Darstellung nach van Dam (2001) und Schlechtweg (2001).

Der Sichtkörper

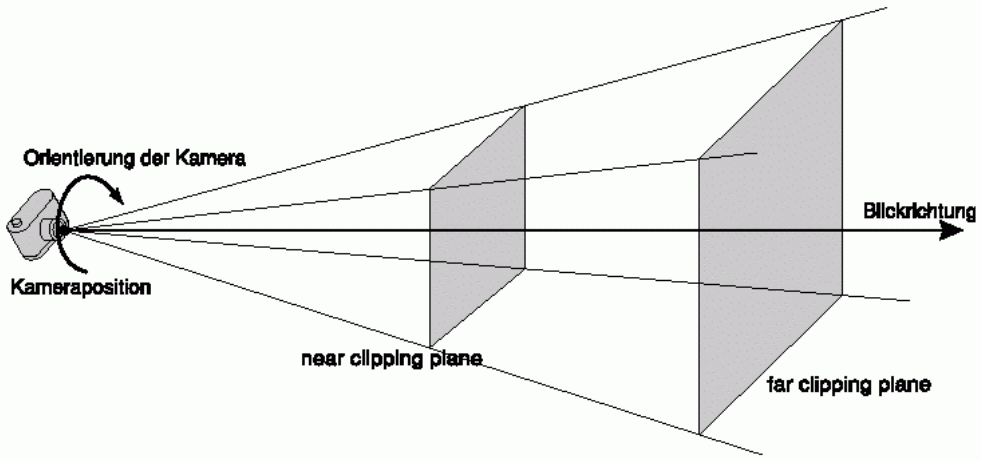
- engl.: view volume
- Sichtkörper enthält alles, was vom COP (oder DOP) aus sichtbar ist (=was die Kamera sieht)
- normalerweise ein Kreiskegel → teure Berechnungen, insbesondere beim Clipping von Objekten gegen eine Kegelfläche
- Approximation durch Pyramide(nstumpf)
  - die Ausgabe ist sowieso rechteckig
  - Clipping durch Lösen eines linearen Gleichungssystems
- Pyramide(nstumpf) = view frustum

Vereinfachtes Kameramodell

Sechs Parameter zur Spezifikation der Kamera

1. Position der Kamera
2. (Blick-)Richtung der Kamera
3. Orientierung der Kamera
4. Seitenverhältnis des zu erzeugenden Bildes
5. Öffnungswinkel
6. near clipping plane und far clipping plane
7. optional: Brennweite

Sichtkörper für dieses Modell



Kameraposition

- analog zur Entscheidung eines Photographen, wo die Kamera aufgestellt wird
- Spezifikation durch einen Punkt in 3D

Orientierung der Kamera

Spezifikation durch zwei Angaben

- Blickrichtung
  - Angabe eines Zielpunktes,
    Berechnung der Richtung aus Zielpunkt – Position
  - Richtung, in die die Kamera „schaut“
  - Spezifikation durch einen Vektor in 3D
- up vector
  - bestimmt die Rotation der Kamera um die Blickrichtung
  - Projektion des up vectors muß in der Ebene senkrecht zur Blickrichtung liegen, d. h. Blickrichtung und up vector dürfen nicht kollinear sein

Seitenverhältnis

- engl.: aspect ratio
- analog zum Seitenverhältnis des Films in der Photographie
- Verhältnis von Breite zu Höhe
- für quadratische Bilder: $1:1$
- Kinofilme: $2:1$
- Fernsehen: $4:3$
- HDTV: $16:9$

Öffnungswinkel

- analog zur Auswahl des Objektivs beim Photographieren
- bestimmt die Stärke der perspektivischen Verzerrung
  - keine Verzerrung – Paralleprojektion
  - sehr starke Verzerrung – Weitwinkellinsen
- eigentlich zwei Öffnungswinkel im view frustum: horizontal und vertikal
- sollten übereinstimmen (Spezifikation eines Winkels, der andere kann aus Seitenverhältnis berechnet werden)

Clipping planes

- Raum zwischen near clipping plane und far clipping plane bestimmt, was die Kamera sieht
- Position der Ebenen durch Entfernungen entlang der Blickrichtung festgelegt
- Objekte außerhalb dieser Ebenen werden nicht gezeichnet
- diese Ebenen schneidende Objekte müssen geclippt werden

Gründe für die Verwendung der Clipping planes

- near clipping plane
  - Objekte zu nah an der Kamera sollen nicht dargestellt werden, da:
    - Sicht auf Rest der Szene blockiert würde
    - zu starke Verzerrung
  - Objekte hinter der Kamera sollen nicht dargestellt werden
- far clipping plane
  - Objekte zu weit von der Kamera entfernt sollen nicht dargestellt werden, da sie visuell nicht signifikant (kaum erkennbar) sind, aber in die Berechnungen einbezogen werden müssen

Kameramodell . . .

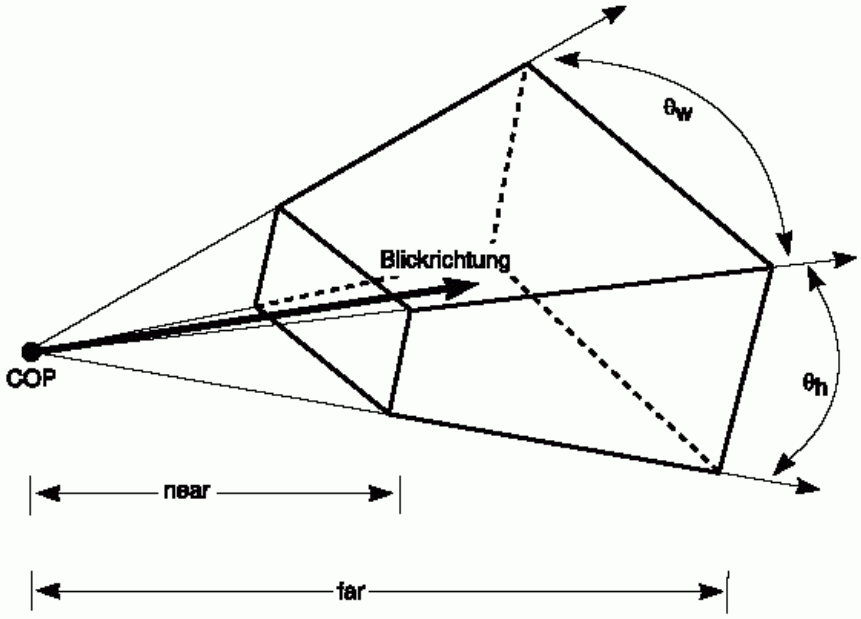
- . . . kann Sichtkörper für folgende Projektionen erstellen:
  - perspektivische Projektion mit positivem Öffnungswinkel
  - Parallelprojektion (Öffnungswinkel ist Null)
- . . . kann nicht für schräge Projektionen verwendet werden

Welche Transformation ist rechnerisch anzuwenden, um die Betrachtungstransformation (Abbildung in den Bildschirmausschnitt) durchzuführen?

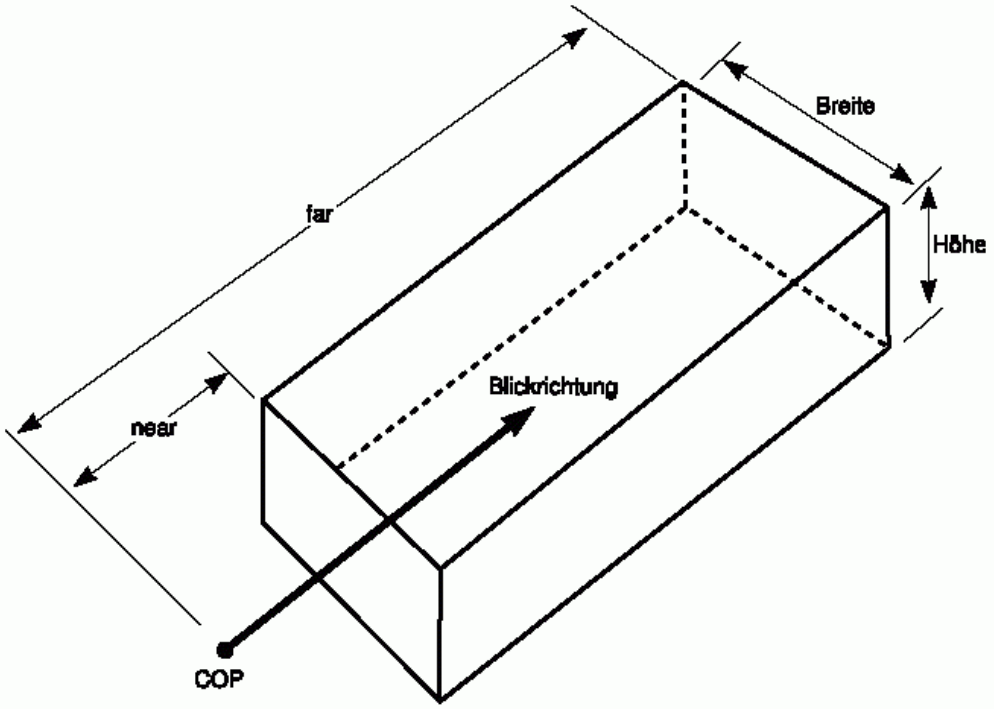
Ziel: Herleitung einer Transformationsmatrix (für Objektdarstellung in homogenen Koordinaten)

- Vorgehensweise:
  - Der kanonische Sichtkörper
  - Spezifikation des Sichtkörpers (bereits behandelt)
  - Transformation des spezifizierten Sichtkörpers in den kanonischen Sichtkörper
  - Projektion und Clipping an diesem kanonischen Sichtkörper

Sichtkörper für perspektivische Projektionen



Sichtkörper für Parallelprojektionen

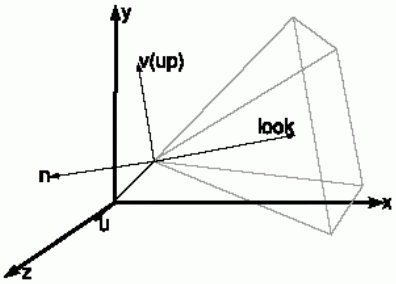


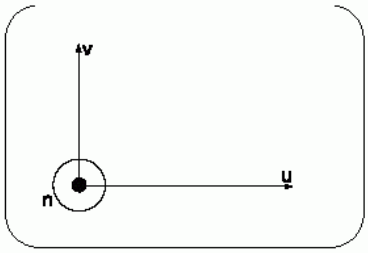
Spezfifkation einer beliebigen 3D-Ansicht

- Plazierung des Sichtkörpers spezifiziert durch:
  - Position (COP)
  - Orientierung (Blickrichtung, up vector)
- Form des Sichtkörpers spezifiziert durch
  - Öffnungswinkel
  - near clipping plane, far clipping plane
- perspektivische Projektion: Projektionsstrahlen schneiden sich im COP
- Parallelprojektion: Projektionsstrahlen parallel zur DOP, schneiden sich nie

Koordinatensysteme

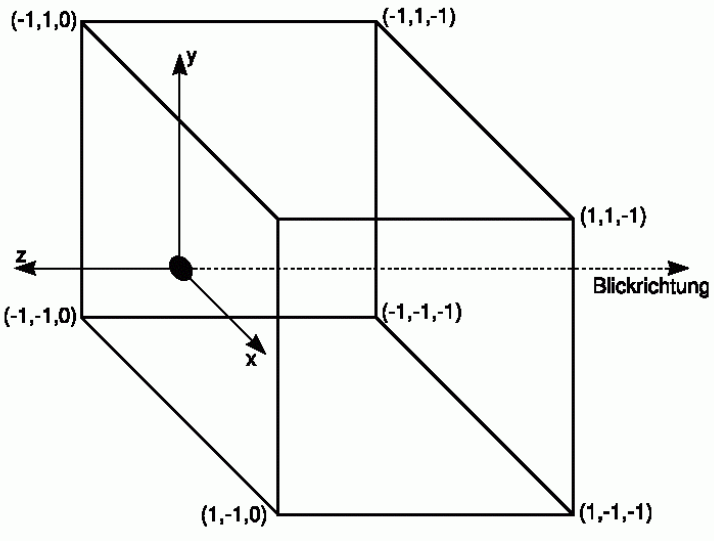
- Weltkoordinaten – Definition des Modells, der Kameraparameter
- View-Referenz-Koordinaten (Kamerakoordinaten) – Ursprung im COP, Achsen rotiert je nach Orientierung der Kamera





Der kanonische Sichtkörper

- beliebiger Sichtkörper zu komplex (Clipping und Projektion)
- Bilderzeugung einfacher vom kanonischen Sichtkörper aus:
  - Parallelprojektion
  - Kamera im Ursprung (Position: $(0, 0, 0)$)
  - Sicht entlang der negativen z-Achse (look vector: $(0, 0, -1)$)
  - aufrecht orientiert (up vector: $(0, 1, 0)$)
  - Ausdehnung der Bildebene zwischen -1 und 1 in x und y
- für perspektivische Projektion ein zusätzlicher Schritt: Unwandlung des perspektivischen Sichtkörpers in den der Parallelprojektion



Vorgehen in einzelnen Schritten – wir lassen die Details weg

Endergebnis

- endgültige Transformation:

$$p' = DS_{xyz}\left(\frac{1}{\text{far}}\right) S_{xy} M^T T(-\text{COP}) p$$

- Die Matrizen $D, S_{xyz}\left(\frac{1}{\text{far}}\right), S_{xy}, M^T, T(-\text{COP})$ können alle berechnet werden, wenn die Kameraparameter bekannt sind.

- Erzeugen einer zusammengesetzten Transformationsmatrix, mit der alle Szenenkoordinaten multipliziert werden, um diese von Weltkoordinaten in den Standard-Sichtkörper für Parallelprojektionen zu überführen.

Clipping

- Transformierte Szene paßt in einen Quader nahe des Koordinatenursprungs
- noch notwendig: Clipping der Szene gegen die Seiten des Sichtkörpers
- normalisierter Sichtkörper mit folgender Ausdehnung:

$$\begin{aligned} -1 \leq\ & x\ \leq 1 \\ -1 \leq\ & y\ \leq 1 \\ 0 \leq\ & z\ \leq 1 \end{aligned}$$

- Clipping ist jetzt sehr einfach!!! x- und y-Komponenten gegen ± 1 testen, z-Komponenten gegen 0 und 1

es kann eine Variante des Cohen-Sutherland-Algorithmus angewandt werden:

6-Bit-Binärcode anstatt 4-Bit im planaren Fall
Bit 5: gesetzt für Punkte (*x*, *y*, *z*) oberhalb des Sichtvolumens
Bit 4: unterhalb
Bit 3: rechts
Bit 2: links
Bit 1: hinter dem Sichtvolumen
Bit 0: vor dem Sichtvolumen

damit dann analoge Oder- und Und-Abfrage wie im planaren Cohen-Sutherland-Algorithmus. (Vgl. Rauber 1993, S. 147 f.)

Projektion

- Projektion in die xy-Ebene durch „Weglassen“ der z-Koordinate
- dabei Mapping auf (Pixel-)Koordinaten des Bildschirms
- Seien (x, y, z) Koordinaten nach Normalisierungstransformation und Clipping, dann ergeben sich die Screen-Koordinaten (x', y') aus:

$$x' = \left\lfloor \frac{W}{2}(x+1) \right\rfloor$$
$$y' = \left\lfloor \frac{H}{2}(y+1) \right\rfloor$$

mit W als Bildschirmbreite und H als Bildschirmhöhe

Zusammenfassung

- Reduktion des gesamten Problems auf Multiplikation der Koordinaten mit einer zusammengesetzten Matrix, die aus der Spezifikation der Kameraparameter gewonnen werden kann
- Zusätzlich dazu noch die Modelltransformationen
- $\rightarrow$ Transformation lokaler Koordinaten des Modells p in Bildschirmkoordinaten p' ist Matrixmultiplikation:

$$p' = NMp$$

mit der Normalisierungstransformation N und der zusammengesetzten Modell-Trannsformationsmatrix M

Visibilitätsrechnung

"Hidden Surface Removal" (HSR)
"Visible Surface Determination" (VSD)

Problem der Verdeckung (Unsichtbarkeit von Teilen einer Szene) entsteht dadurch, dass Projektionen nicht injektiv sind

$\Rightarrow$ mehrere Objektpunkte haben denselben Bildpunkt ("Projektionsäquivalenz"), aber nur einer davon ist "sichtbar" (= bestimmt Farbe u. Intensität des betr. Pixels).

Im Folgenden:
- Problemdefinition
- Konservative Sichtbarkeitstests für die Vorverarbeitung
- Algorithmen (hier nur zwei, es gibt sehr viel mehr!)

Problemdefinition:

Gegeben sei eine Menge von 3D-Objekten und die Spezifikation einer Ansicht dieser Szene (Kameramodell).
Problem: Bestimme, welche Linien oder Flächen der gegebenen Objekte unter der gegebenen Ansicht sichtbar sind.

Drei Klassen von Algorithmen:

1. Bildraumalgorithmen ($\rightarrow$ *image precision*)
- Sichtbarkeit wird für diskrete Bildpunkte bestimmt
- Beispiel: *z*-Buffer
2. Objektraumalgorithmen ($\rightarrow$ *object precision*)
- exakte Sichtbarkeitsbestimmung im Modellraum
- Beispiele: Clipping von Polygonen an Polygonen, 3D-Tiefensortierung, BSP-Bäume
3. Hybride Algorithmen
- arbeiten sowohl im Objekt- als auch im Bildraum

Zusätzlich: "konservative Sichtbarkeitstests" (Vorverarbeitung): Aussondern von mehr oder weniger "trivialen" Fällen.

Back-face culling

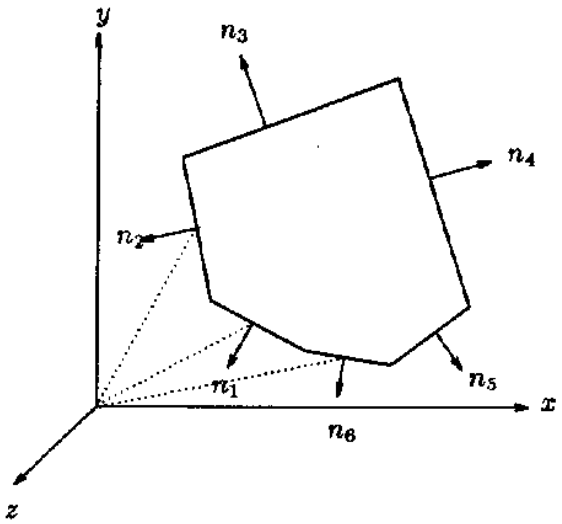
Entfernen von nicht sichtbaren Rückseiten, d.h. von allen Polygonen, die vom Betrachter weg zeigen
(diese Rückseiten machen etwa die Hälfte aller Flächen aus!)

setzt voraus:
- Objekte sind als Polygonnetze modelliert, der Betrachterstandpunkt liegt außerhalb
- Polygone sind orientiert: Festlegung von "innen" und "außen" hinsichtlich des modellierten Körpers.

Testgrundlage: nach außen zeigende Normalen der Polygone.

Zwei Möglichkeiten der Betrachtung:
- Line-of-Sight-Interpretation
- Halbraum-Interpretation

Line of Sight (LOS):
Strahl vom Betrachterstandpunkt (COP) zu einem beliebigen Punkt des Polygons
(bei Parallelprojektion ist die LOS identisch mit der DOP)



Zeigt die Normale zur selben Seite wie die LOS (Winkel < 90°), dann handelt es sich bei dem Polygon um eine zu entfernende Rückseite.

Rechnerisch: Skalarprodukt prüfen
LOS · Normale > 0 $\Rightarrow$ unsichtbar
LOS · Normale $\leq$ 0 $\Rightarrow$ möglicherweise sichtbar

Halbraum-Interpretation:
Normalenform der Ebenengleichung für die Polygon-Ebene:
(Normale · x) + D = 0
Die Ebene teilt den übrigen Raum in zwei Halbräume:
(Normale · x) + D > 0: x liegt im positiven Halbraum
(Normale · x) + D < 0: x liegt im negativen Halbraum

Polygon zeigt vom Betrachter weg, wenn sich der Betrachterstandpunkt (COP) im negativen Halbraum befindet:
unsichtbar, wenn (Normale · COP) + D < 0

beide Interpretationen sind letztlich äquivalent (LOS lässt sich aus COP und Polygonpunkt berechnen).

Algorithmen zur Visible Surface Determination

Z-Buffer-Algorithmus

häufig in der Hardware implementiert
z-Buffer (z-Puffer, Tiefenpuffer): pixelbezogener Speicher für die z-Werte

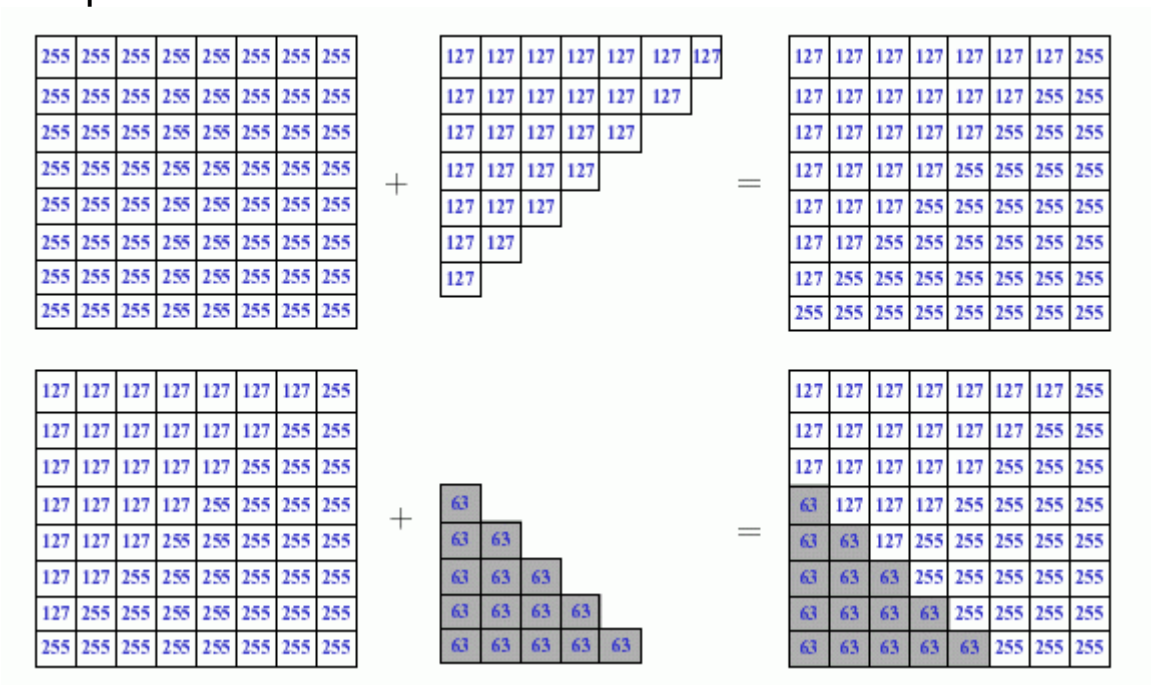
- schon 1974 vorgeschlagen, aber damals nicht realisiert
- heute dominierender Algorithmus für Rastergeräte

Vorgehensweise:

- zusätzlich zum Framebuffer (Bild) noch eine zweite Bitmap, die die z-Werte speichert
- z-Buffer initialisiert mit Hintergrundwert (Tiefe der am weitesten entfernten Ebene des Sichtkörpers, $z = 1$
- z-Wert jedes Pixels wird mit dem Wert im z-Buffer verglichen
  - z aus Ebenengleichung des aktuellen Polygons berechnen
  - besser: z an den Eckpunkten berechnen und interpolieren
- ist aktueller z-Wert kleiner als der an dieser Position gespeicherte: aktuellen Wert in den z-Buffer speichern und Pixel zeichnen
- sonst keine Änderungen

die Polygone können in jeder beliebigen Reihenfolge gezeichnet werden

Beispiel:



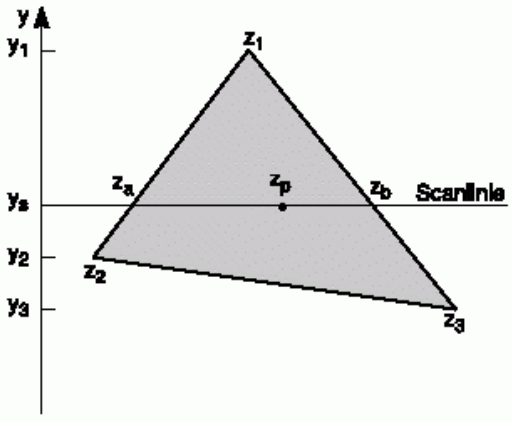
(hier *z*-Werte nicht zwischen 0 und 1, sondern zwischen 0 und 255; implementationsabhängig)

Algorithmus:

```
initialisiere z-Buffer
foreach Polygon do
   foreach Pixel in der Projektion des Polygons do
      p_z = z-Wert des Polygons an Position (x, y)
      if p_z < z-Buffer(x, y) then
                                   WritePixel(x, y, c)
                                   WriteZ(x, y, p_z)
      fi
   od
od
```

Effiziente Implementierung: *z*-Werte im Inneren der Polygone durch Interpolation bestimmen

- ähnliche Vorgehensweise wie beim Scanlinien-Algorithmus zum Polygonfüllen
- Interpolation zunächst entlang der Polygon-Kanten, dann entlang der Scanlinien im Inneren



$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{x_b - x_a}$$

- erweiterbar zu inkrementellem Verfahren zur Bestimmung der *z*-Werte

- Vorteile des z-Buffer-Algorithmus
  - einfach auch in Hardware zu implementieren → **schnell**
  - Polygone können in beliebiger Reihenfolge bearbeitet werden
  - jedes Polygon einzeln behandelt
  - kann auch für nicht-polygonale Flächen genutzt werden

- Nachteile des z-Buffer-Algorithmus
  - Genauigkeitsproblem, da z-Werte durch perspektivische Verkürzung „komprimiert“ werden
  - kein Antialiasing
  - *alle* Polygone müssen behandelt werden
  - Transparenz ist nicht realisierbar

Verbesserung:
Hierarchischer z-Buffer-Algorithmus

Ersetze *z*-Buffer durch "*z-Pyramide*"
tiefste Ebene: z-Buffer in maximaler Auflösung
höhere Ebenen: Jedes Pixel repräsentiert die maximale Tiefe der vier Pixel "unter" ihm

Grundidee: hierarchische Rasterung des Polygons; früher Abbruch, wenn Polygon verdeckt ist.

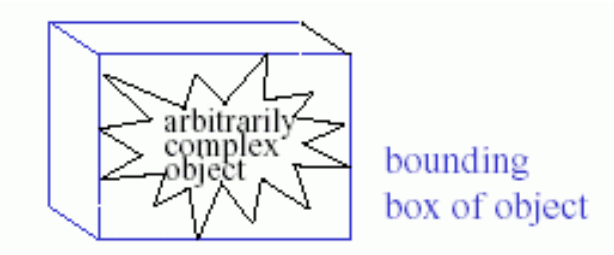
- Polygon zuerst gegen die höchste Ebene testen
- Wenn das Polygon weiter entfernt ist als die Tiefe, die im entspr. (Makro-) Pixel gespeichert ist: verdeckt
- wenn näher: Test gegen die nächstniedrige Ebene, usw.
- Wenn Polygon auf der tiefsten Ebene sichtbar ist, dann zeichnen und *z*-Pyramide aktualisieren.

Nutzung von Kohärenz in zweierlei Hinsicht möglich:

- Ein in einem Pixel verdecktes Polygon ist wahrscheinlich auch in benachbarten Pixeln verdeckt (Bildraum-Kohärenz, wird inhärent von der Pyramide genutzt)
- Polygone nahe einem verdeckten Polygon sind möglicherweise auch verdeckt (Objektraum-Kohärenz).

Verbesserung des Algorithmus durch Ausnutzung der Objektraum-Kohärenz:

- Unterteile die Szene durch einen *Octree*
- Geometrische Objekte in einem Knoten des Octrees sind in einem Würfel enthalten (*bounding volume*).
- Bevor der Inhalt des Würfels gerendert wird: Teste die Seiten des Würfels gegen die *z*-Pyramide!



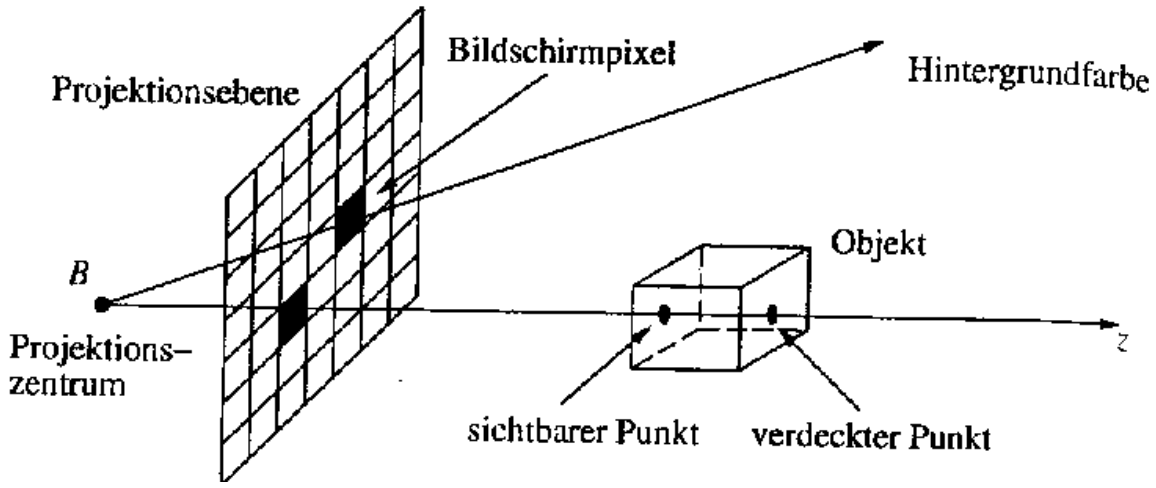
- Wenn die Seitenflächen des Würfels verdeckt sind, dann ist auch die Geometrie im Inneren des Würfels verdeckt – gesamten Inhalt ignorieren.

(Dieser Ansatz ist ein Beispiel für eine "bounding volume hierarchy - Technik", wovon es noch andere Varianten gibt.)

Ray-Casting-Verfahren

- Hybrid Objektraum-Bildraum (aber mehr im Objektraum)
- Prinzip der Strahlverfolgung (vgl. Raytracing in der Beleuchtungsrechnung, siehe später)
- vom Betrachterstandpunkt wird je ein Strahl durch jedes Pixel in die 3D-Szene verfolgt.
- Berechnung der Schnittpunkte der Strahlen mit den Objektoberflächen

Fallunterscheidung dabei:
1. es gibt keinen Schnittpunkt: verwende die Hintergrundfarbe.
2. es existieren Schnittpunkte: bestimme den zum Betrachter nächstgelegenen, dieser ist sichtbar, die anderen verdeckt.



Nachteil: hoher Rechenaufwand durch häufige Schnittpunkt-berechnungen (ca. 95 % der Gesamtrechenzeit bei typischen Szenen)
Vorteile:

- leicht parallelisierbar (bei *p* Prozessoren teile Bildschirm in *p* disjunkte Rechtecke – allerdings muss bei diesem Ansatz die 3D-Szene auf jedem Prozessor vollständig zur Verfügung stehen)
- Effizienzverbesserung durch *boundary volume*-Techniken möglich (hierarchische Konzepte, Clustering)
- Kombination mit klassischem Raytracing (Beleuchtungs-rechnung) möglich