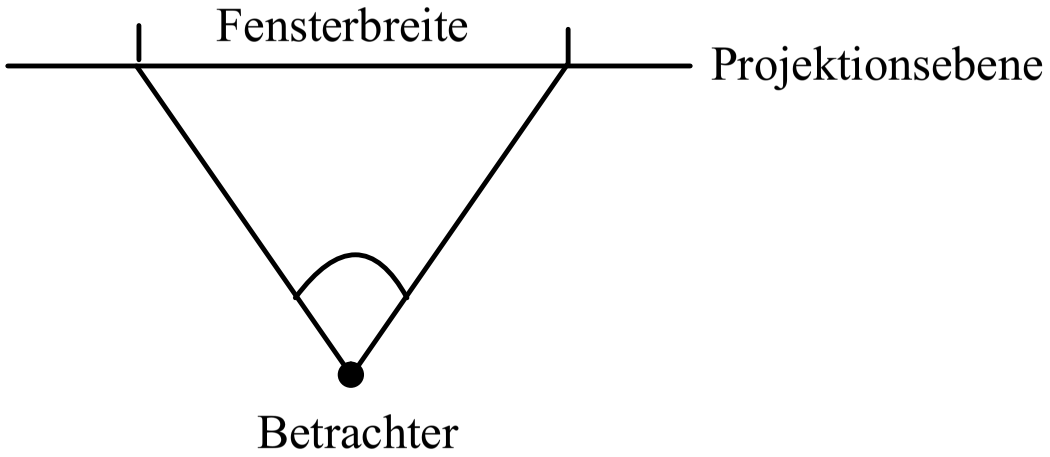
Betrachter im 3D

Festlegung

- des Betrachterstandpunktes
- der Projektionsebene
- der Projektionsart (perspektivisch/parallel)
- des Sichtfeldes durch einen Winkel (wodurch implizit die Größe des Bildschirmfensters auf der Projektionsebene festgelegt wird)

Sichtfeldwinkel



In Java 3D durch:

```
View v =
  simpUniv.getViewer().getView();

v.setFieldOfView(angle);
```

Betrachter im 3D

In Java 3D wird der Betrachter durch ein Objekt der Klasse `PhysicalBody` beschrieben.

Der Abstand der Projektionsebene vom Betrachterstandpunkt kann mittels

```
v.getPhysicalBody().
setNominalEyeOffsetFromNominalScreen(
distance);
```

verändert werden.

Betrachter im 3D

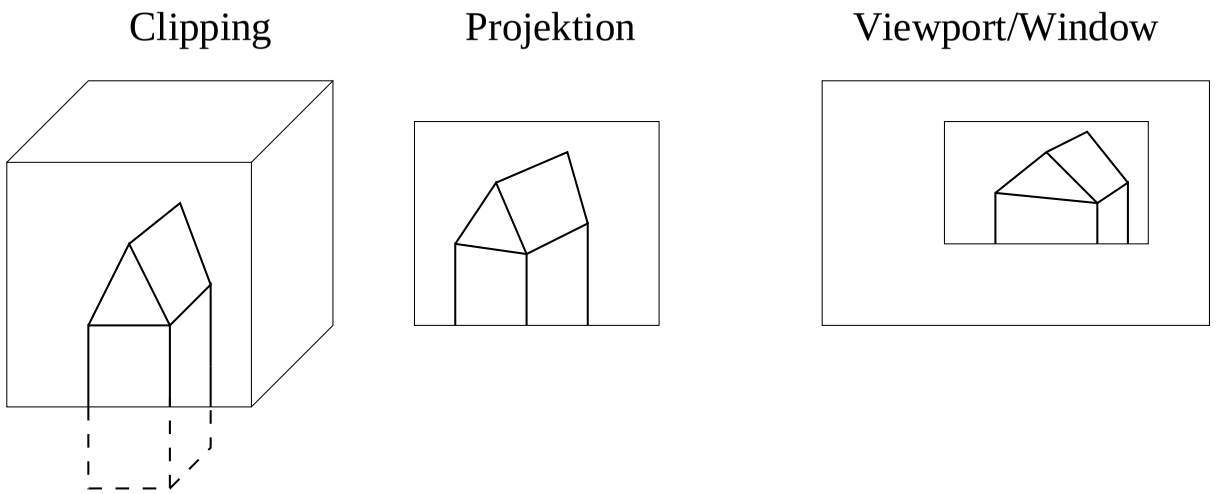
Verändern der Betrachterposition durch eine Transformation `vt`:

```
Transform3D vt = new Transform3D();

vt.set(...);

simpUniv.getViewPlatform.setTransform(vt);
```

Clipping im 3D



Clipping bedeutet die Bestimmung der sich im sichtbaren Bereich befindlichen Objekte bzw. welche Teile eines Objektes sich im sichtbaren Bereich befinden.

Dazu gehören keine Visibilitätsberechnungen, welche Objekte von anderen Objekten im sichtbaren Bereich verdeckt werden.

Clipping im 3D

Clipping im 3D bedeutet theoretisch die Einschränkung der Szene auf

- eine Pyramide unendlicher Höhe im Falle perspektivischer Projektion bzw.
- einen sich einer Richtung unendlich ausdehnenden Quader im Fall der Parallelprojektion.

Die Sichtweite eines Menschen ist zwar theoretisch nahezu unbegrenzt.

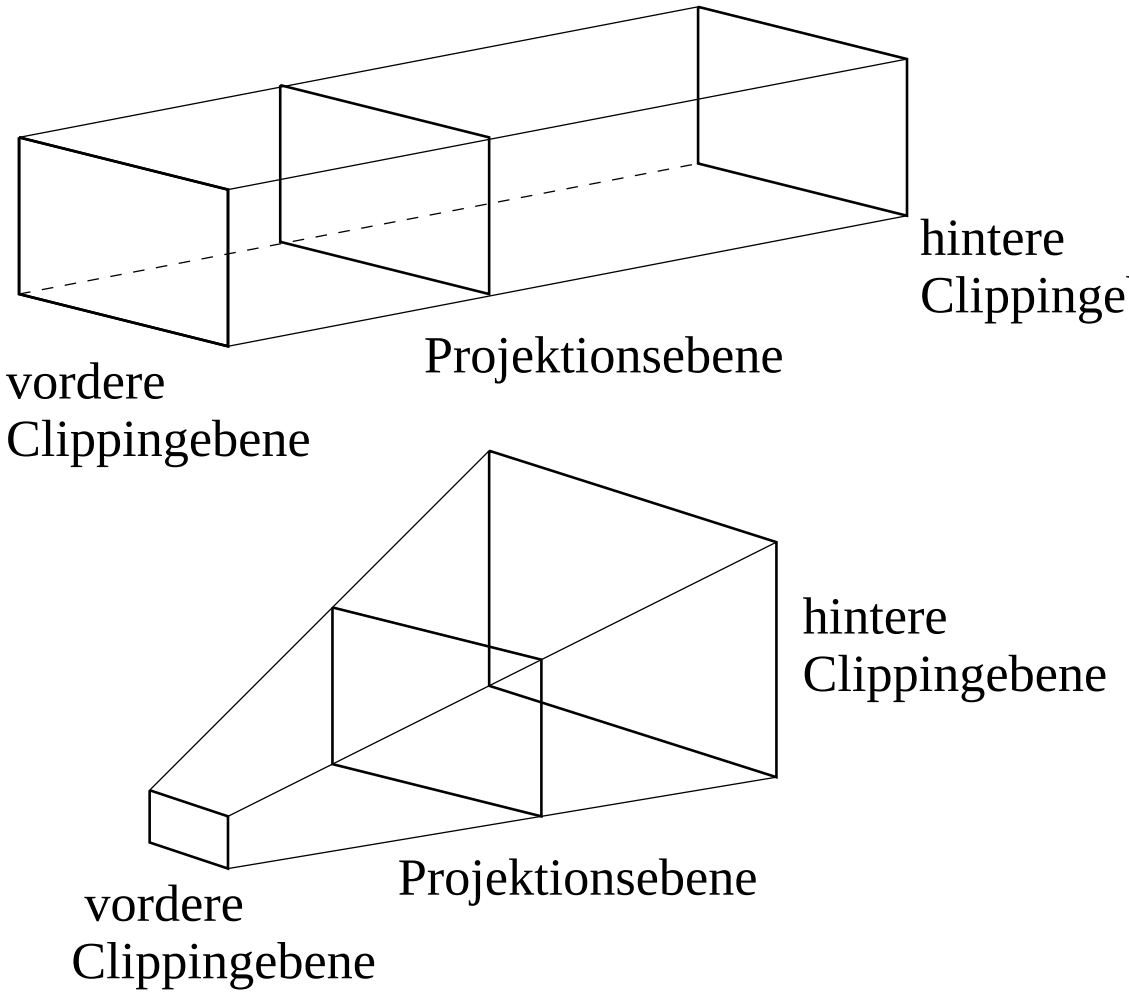
Clipping im 3D

Ein auf einer Wiese liegender Beobachter kann seine Augen aber nicht gleichzeitig auf die direkt vor seiner Nase stehende Blume und das am Himmel fliegende Flugzeug fokussieren.

Der (scharf) sichtbare Bereich erstreckt sich daher üblicherweise von einer bestimmten minimalen bis zu einer bestimmten maximalen Entfernung.

Dies wird in der Computergrafik durch eine **vordere** und eine **hintere Clippingebene** realisiert.

Clipping im 3D



Clipping im 3D

Parallelprojektion: Der Clipping-Bereich ist ein Quader.

perspektivische Projektion: Der Clipping-Bereich ist ein Pyramidenstumpf.

Die Projektionsebene liegt üblicherweise zwischen der vorderen und hinteren Clippingebene.

- Projektionsebene $\widehat{=}$ Bildschirm
- vordere Clippingebene $\widehat{=}$ kürzester fokussierter Abstand
- hintere Clippingebene $\widehat{=}$ weitester fokussierter Abstand

Clipping im 3D

Clipping bei der abschließenden Parallelprojektion:

Clipping-Quader kann durch zwei Ecken $(x_{\text{min}}, y_{\text{min}}, z_{\text{min}})$ und $(x_{\text{max}}, y_{\text{max}}, z_{\text{max}})$ definiert werden.

Überprüfung, ob ein Polygon im Clipping-Bereich liegt:

$$x_{\text{min}} \leq p_x \leq x_{\text{max}} \quad \text{und}$$
$$y_{\text{min}} \leq p_y \leq y_{\text{max}} \quad \text{und}$$
$$z_{\text{min}} \leq p_z \leq z_{\text{max}}$$

für mindestens eine Ecke (p_x, p_y, p_z) des Polygons?

Clipping im 3D

In Java 3D:

```
v.setBackClipDistance(bcDist);

v.setFrontClipDistance(fcDist);
```

Dabei ist das Objekt `v` der zum entsprechenden `SimpleUniverse` gehörende `View`.

(Clipping und Betrachterwinkel vgl. `ClippingPlanes.java`)

Visibilitätsverfahren

Um Objekte aus einer 3D-Welt darzustellen, muss neben Clipping und Projektion berechnet werden, welche Objekte oder Objektteile überhaupt sichtbar und nicht durch andere Objekte verdeckt sind.

Derartige Algorithmen nennt man **Visibilitätsverfahren** (engl. hidden line/hidden surface algorithms).

Visibilitätsverfahren

Einfacher Algorithmus:

```
for (jedes Pixel im Bild)
{
  Bestimme das Objekt mit der geringsten
  Entfernung zum Betrachter, das durch
  von der mit dem Pixel assoziierten
  Projektionsgerade (-richtung)
  durchstossen wird.

  Setze das Pixel in der entsprechenden
  Farbe.
}
```

Diese Technik wird als **Bildraumverfahren** (image-precision) bezeichnet.

Visibilitätsverfahren

Ein Bildraumverfahren hat bei p Pixeln und n Objekten einen Rechenaufwand von $n \cdot p$ Schritten.

Eine Alternative bilden **Objektraumverfahren** (object-precision):

```
for (jedes Objekt im Bild)
{
  Bestimme den Teil des Objektes, der
  nicht von anderen Teilen desselben oder
  eines anderen Objektes verdeckt ist.

  Zeichne den sichtbaren Teil des
  Objektes in der entsprechenden Farbe.
}
```

Visibilitätsverfahren

Ein Objektraumverfahren benötigt n^2 Schritte.

Es gilt zwar i.A. $n \ll p$, so dass $n^2 \ll np$ folgt.

Allerdings sind die Einzelschritte bei den Objektraumverfahren aufwendiger.

Der Vorteil der Objektraumverfahren besteht darin, dass sie unabhängig von der Bildauflösung berechnet werden können.

Rückseitenentfernung

Bei der **Rückseitenentfernung** (Back-Face-Culling) werden zunächst die (ebenen) Flächen entfernt, die der Betrachter auf keinen Fall sehen kann, da er sie aus seiner Blickrichtung von hinten sehen würde.

Bei einem einzelnen Polyeder können die Flächen, die vom Betrachter abgewandt sind, anhand der Normalenvektoren bestimmt werden.

Dazu werden die Normalenvektoren so gewählt, dass sie nach außen zeigen.

Rückseitenentfernung

Eine ebene Fläche ist eine Rückseite (und muss somit beim Zeichnen nicht betrachtet werden)

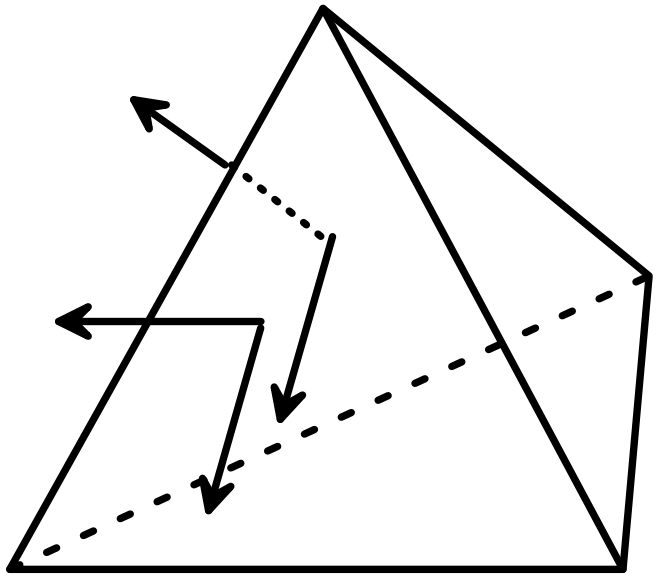
$\Longleftrightarrow$ Der Winkel zwischen Normalenvektor und der z-Achse ist größer als $90°$.

$\Longleftrightarrow$ Das Skalarprodukt von Normalenvektor und dem z-Einheitsvektor ist negativ.

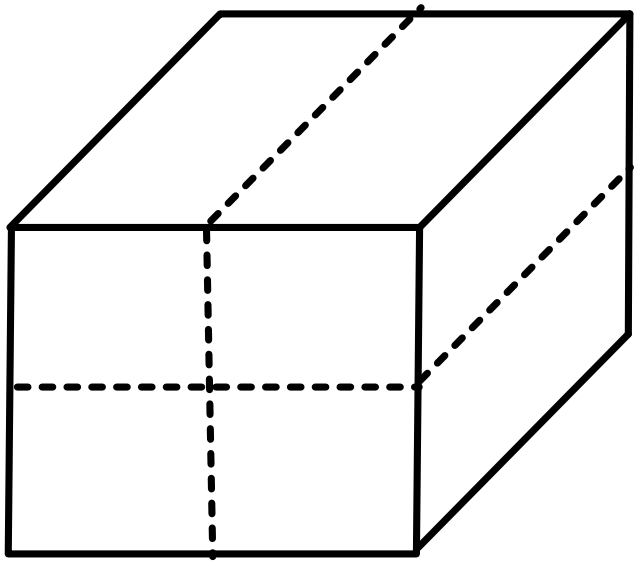
$\Longleftrightarrow$ Die z-Koordinate ist negativ.

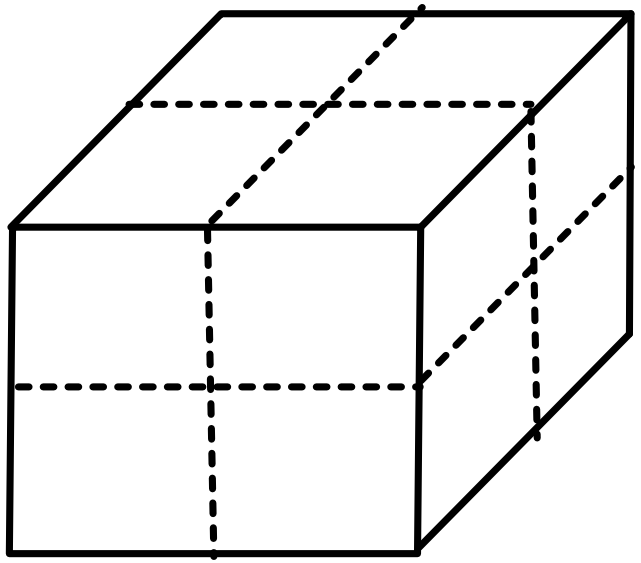
(bei einer Parallelprojektion auf eine zur x/y-Ebene parallelen Ebene)

Rückseitenentfernung



Partitionierende Verfahren





Partitionierende Verfahren

Aufteilung des Clippingvolumens in Teilquader.

Objekte in nebeneinander liegenden Quadern können sich nicht verdecken.

Objekte in hinteren Quadern können vordere nicht verdecken.

Idealfall: Reduktion des Rechenaufwands von n^2 bei n Objekten auf $k \cdot \left(\frac{n}{k}\right)^2 = \frac{n^2}{k}$ bei k Teilquadern.

Gilt nicht für große k: Objekte liegen dann fast immer in mehreren Quadern

Rekursive Teilungsalgorithmen

Rekursive Aufteilung des betrachteten Gebiets so lange, bis in den kleineren Gebieten der Sichtbarkeitsentscheid getroffen werden kann.

Die Bildauflösung begrenzt die maximale Rekursionstiefe.

Area-Subdivision-Verfahren: Unterteilung der Projektionsfläche

Oktalbaum-Verfahren: Unterteilung des Clippingvolumens

z- oder Tiefen-Puffer-Algorithmus

Für jedes Pixel in der Projektionsebene wird ein Farb- und eine z-Koordinate gespeichert (Farb- und z-Puffer).

Der Farbpuffer wird mit der Hintergrundfarbe initialisiert, der z-Puffer mit der z-Koordinate der hinteren Clipping-Ebene.

Objekte werden pixelzeilenweise in die Puffer geschrieben:

z- oder Tiefen-Puffer-Algorithmus

Ist die z-Koordinate des projizierten Pixels größer als der bisherige Wert im z-Puffer (der Pixel näher am Betrachter als der bisher eingetragene Pixel), wird der z-Puffer mit dem aktuellen z-Wert und der Farbwert mit dem Farbwert des neuen Pixels überschrieben.

Die Reihenfolge, in der Objekte eingetragen werden, spielt dabei keine Rolle.

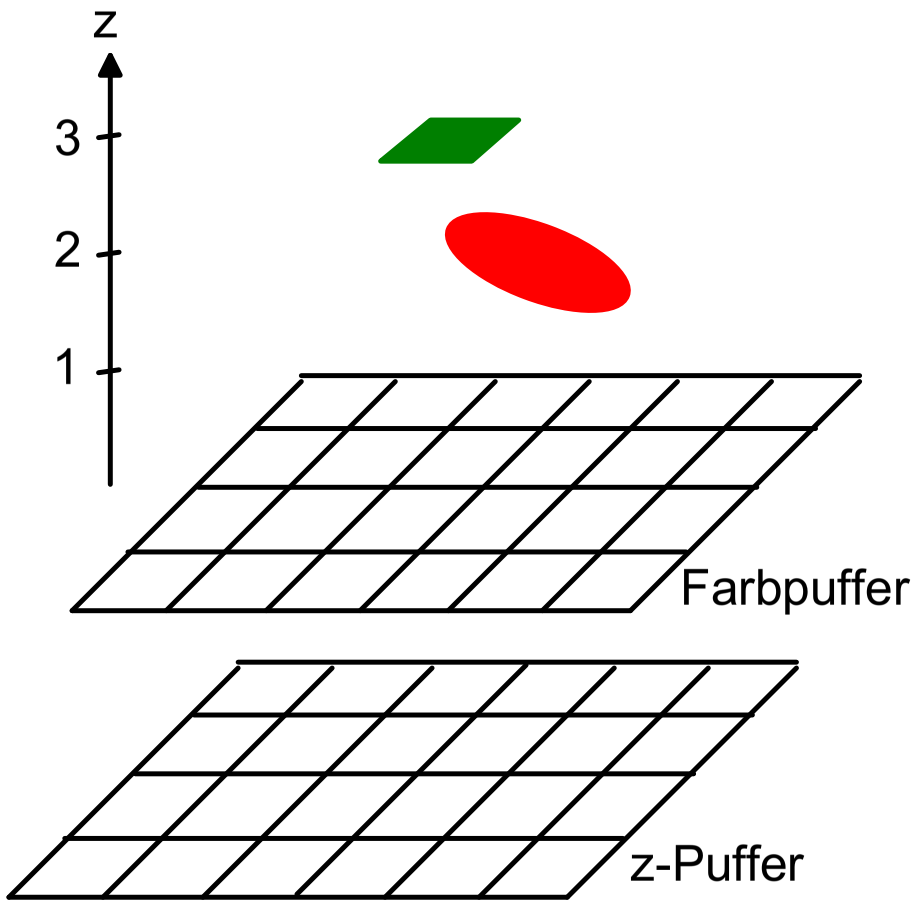
Zusätzliche Objekte können sehr leicht eingetragen werden.

z- oder Tiefen-Puffer-Algorithmus

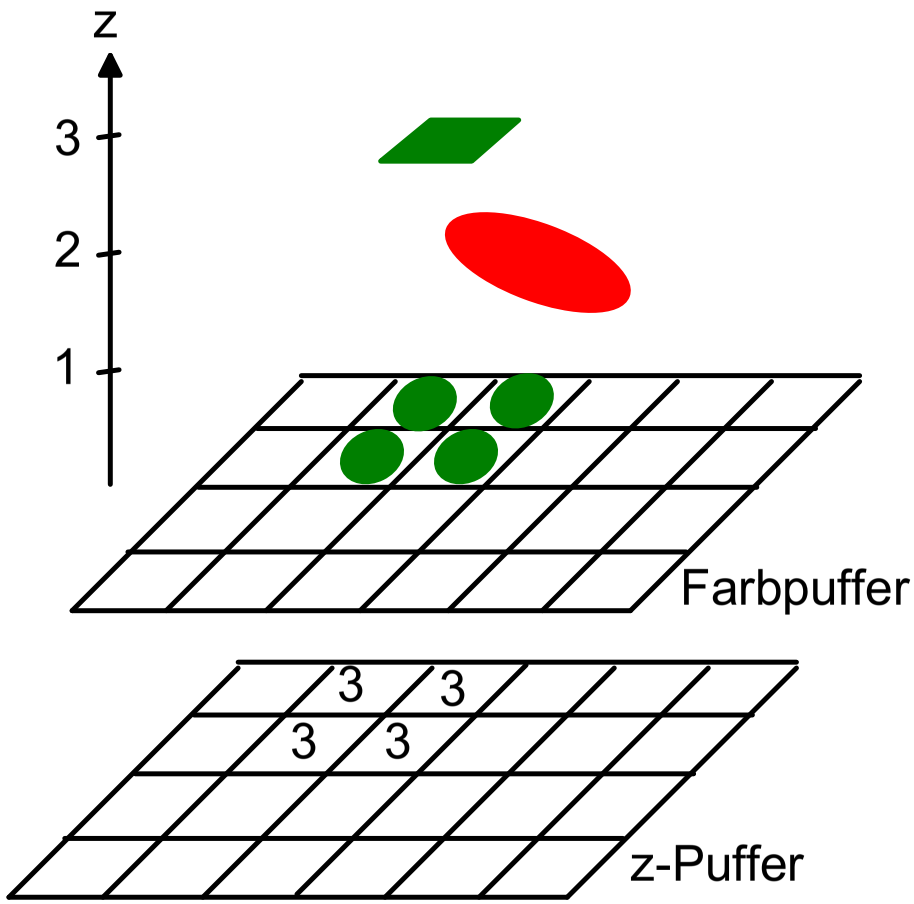
Bei Bildfolgen mit festem Hintergrund müssen der z- und der Farb-Puffer für die Hintergrundobjekte nur einmal berechnet und gespeichert werden.

Nur die sich bewegenden Objekte müssen jeweils neu eingetragen werden.

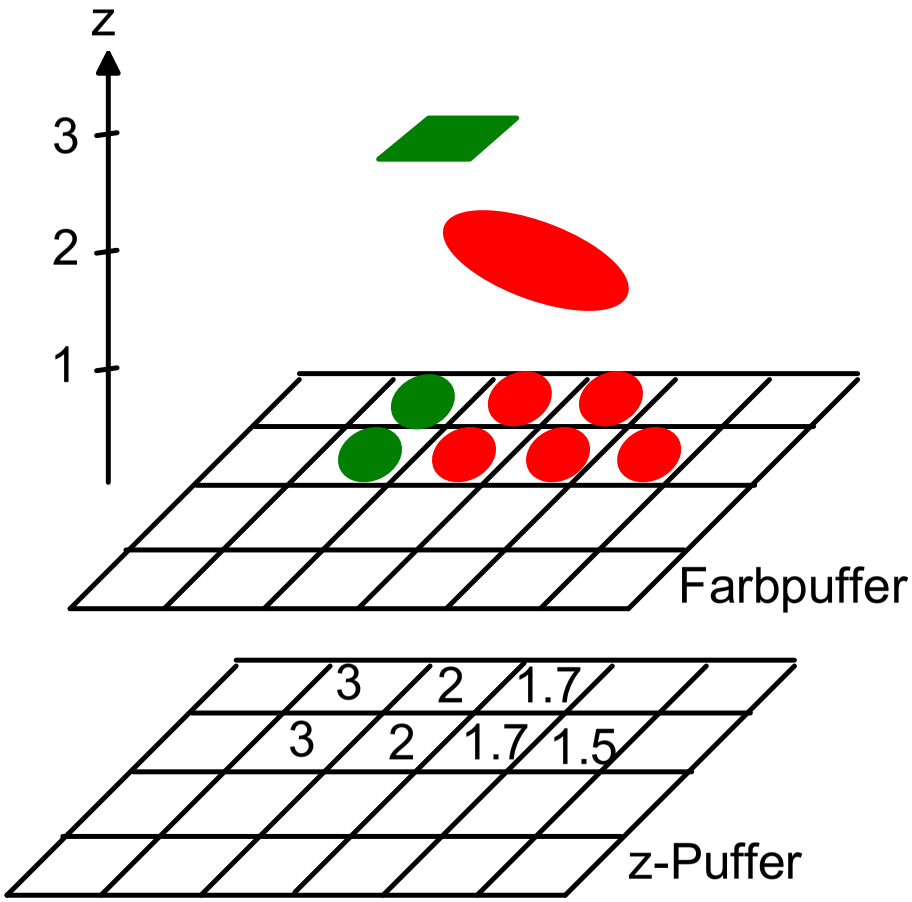
z- oder Tiefen-Puffer-Algorithmus



z- oder Tiefen-Puffer-Algorithmus



z- oder Tiefen-Puffer-Algorithmus



z- oder Tiefen-Puffer-Algorithmus

Eintragen eines ebenen Polygons über Scan-Linienverfahren:

Ebenengleichung des Polygons:

$$A \cdot x + B \cdot y + C \cdot z + D \; = \; 0$$

Berechnung des z-Wertes innerhalb einer Scan-Linie:

$$z_{\text{neu}} \; = \; z_{\text{alt}} + \Delta z$$

Für das Pixel (u, v) sei die entsprechende z-Koordinate des Polygons z_{alt}.

z- oder Tiefen-Puffer-Algorithmus

Für das Pixel $(u+1, v)$ ergibt sich die z-Koordinate aus:

$$
\begin{aligned}
0 &= A \cdot (x+1) + B \cdot y + C \cdot z_{\text{neu}} + D \\
&= A \cdot (x+1) + B \cdot y + C \cdot (z_{\text{alt}} + \Delta z) + D \\
&= \underbrace{A \cdot x + B \cdot y + C \cdot z_{\text{alt}} + D}_{=0} + A + C \cdot \Delta z \\
&= A + C \cdot \Delta z
\end{aligned}
$$

Also: $\Delta z = -\frac{A}{C}$

Scan-Linien-Verfahren

Scan-Linien-Verfahren führen Berechnungen entlang der Pixelzeilen (oder auch der Spalten) durch.

Pixelkoordinaten: (u, v)

Verwendung von drei Tabellen:

Kantentabelle: Enthält alle nicht horizontal verlaufenden Kanten:

v_{min}	$u(v_{\text{min}})$	v_{max}	Δu	Polygon-Nummern

Scan-Linien-Verfahren

- v_{min}: kleinster v-Wert der Kante
- $u(v_{\text{min}})$: zugehöriger u-Wert zum v_{min}-Wert der Kante
- v_{max}: größter v-Wert der Kante
- Δu: Inkrement (Steigung) der Kante
- Polygonnummern: Liste der Polygone, zu denen die Kante gehört

Die Kanten werden aufsteigend nach v_{min} und bei gleichen Werten aufsteigend nach $u(v_{\text{min}})$ sortiert.

Scan-Linien-Verfahren

Polygontabelle: Enthält alle Polygone:

Polygon-Nr.	A	B	C	D	Farbe	In-Flag

- Polygon-Nr.: Idenfikationsnummer des Polygons
- A, B, C, D definieren die zum Polygon gehörende Ebenengleichung: $Ax + By + Cz + D = 0$
- Farbe: Farbwert oder Farbinformation für das Polygon
- In-Flag: Zeigt an, ob die momentan betrachtete Position innerhalb oder außerhalb des Polygons liegt

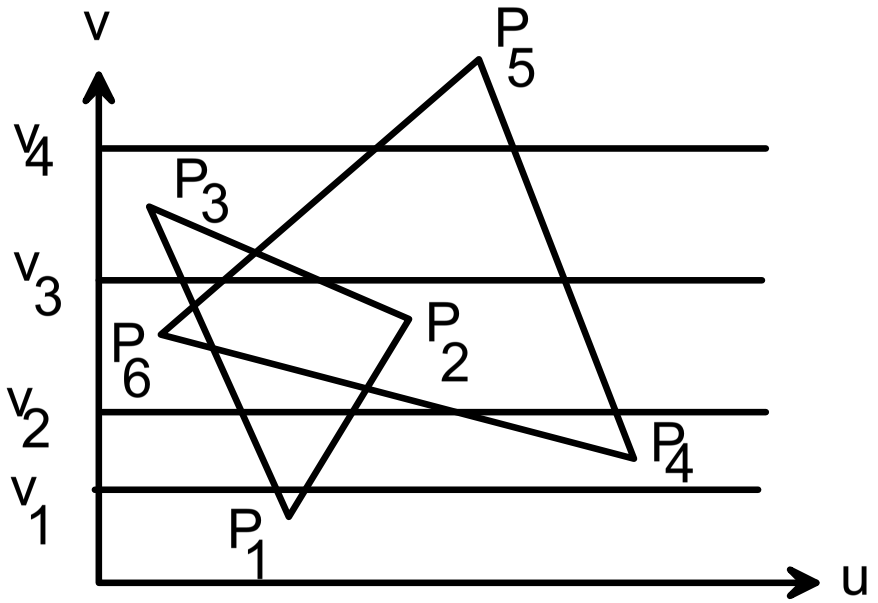
Scan-Linien-Verfahren

aktive Kanten-Tabelle: Liste aller Kanten, die die aktuell betrachtete Scan-Linie schneiden, aufsteigend nach den u-Komponenten der Schnittpunkte

Die Länge der Tabelle der aktiven Kante ändert sich während der Berechnung.

Die Längen und die Einträge (außer das In-Flag) der anderen beiden Tabellen ändern sich während der Berechnung nicht.

Scan-Linien-Verfahren



Aktive Kanten bei den Scan-Linien v_1, v_2, v_3, v_4

$v_1 : P_3P_1,\ P_1P_2$

$v_2 : P_3P_1,\ P_1P_2,\ P_6P_4,\ P_5P_4$

$v_3 : P_3P_1,\ P_6P_5,\ P_3P_2,\ P_5P_4$

$v_4 : P_6P_5,\ P_5P_4$

Scan-Linien-Verfahren

Vorgehensweise beim Durchlaufen jeder Pixelzeile:

```
Aktualisiere die Liste der aktiven Kanten;
Setze alle In-Flags auf 0;
for (alle Schnittpunkte)
    (s. aktive Kantenliste)
{
  Aktualisiere die In-Flags;
  Bestimme das sichtbare Polygon;
  Setze Pixelfarbe entsprechend dem
       Farbeintrag in der Polygontabelle;
}
```

Scan-Linien-Verfahren

Bestimmung des sichtbaren Polygons:

Für jedes aktive Polygon (In-Flag = 1) wird durch die zugehörige Ebenengleichung

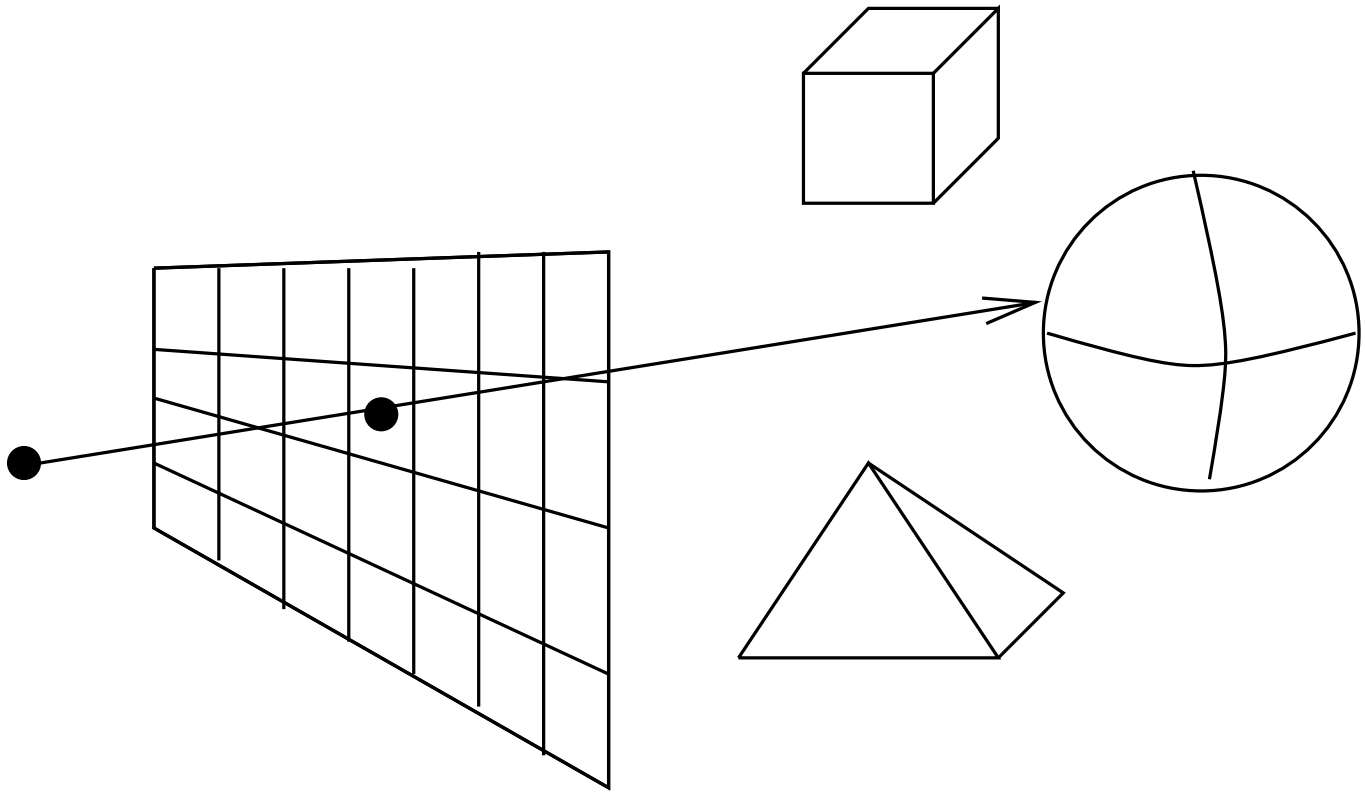
$$A \cdot u + B \cdot v + C \cdot z + D = 0$$

der zugehörige z-Wert ermittelt.

Das Polygon mit dem kleinsten z-Wert ist an der entsprechenden Stelle sichtbar.

Strahlenverfolgung (Ray-Casting)

Für jedes Pixel des darzustellenden Fensters auf der Projektionsebene wird ein Lichtstrahl berechnet und bestimmt, welche Körperoberfläche der Strahl zuerst schneidet, und damit die Farbe des Pixels festgelegt.



Strahlenverfolgung (Ray-Casting)

Parametrisierung des Strahls von (x_0, y_0, z_0) (z.B. Projektionszentrum) zum Punkt $P = (x_1, y_1, z_1)$ (z.B. Punkt/Pixel auf der Projektionsebene):

$$x = x_0 + t\Delta x, \quad y = y_0 + t\Delta y, \quad z = z_0 + t\Delta z$$

mit

$$\Delta x = x_1 - x_0, \quad \Delta y = y_1 - y_0, \quad \Delta z = z_1 - z_0$$

$t > 1 \quad \Leftrightarrow$ P liegt hinter der Projektionsebene.

$0 < t < 1 \quad \Leftrightarrow$ P liegt zwischen Projektionszentrum und -ebene.

$t < 0 \quad \Leftrightarrow$ P liegt vor dem Projektionszentrum.

Strahlenverfolgung (Ray-Casting)

Schnittpunkt mit einem ebenen Polygon:

1. Bestimmung des Schnittpunktes des Strahls mit der Ebene, die das Polygon aufspannt.
2. Überprüfung, ob der Schnittpunkt innerhalb des Polygons liegt.

Ebenengleichung: $$Ax + By + Cz + D = 0$$

Einsetzen ergibt: $$t = -\frac{Ax_0 + By_0 + Cz_0 + D}{A\Delta x + B\Delta y + C\Delta z}$$

Strahlenverfolgung (Ray-Casting)

Ist der Nenner 0, verläuft die Gerade parallel zur Ebene.

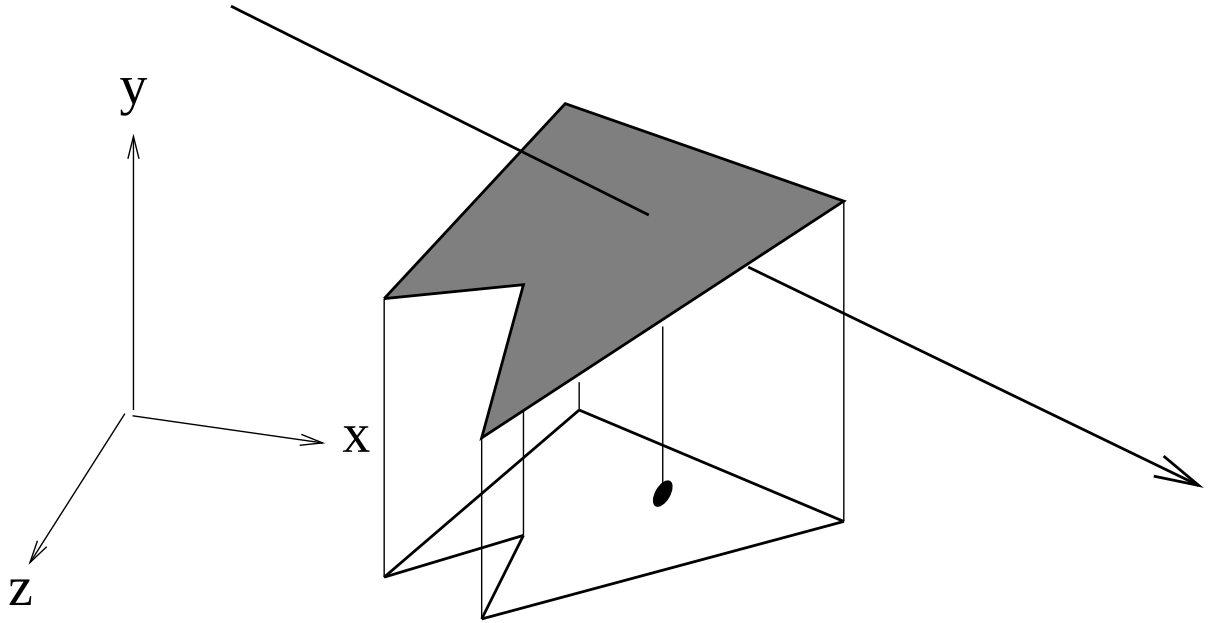
Ansonsten projiziert man das Polygon und den Schnittpunkt in eine der Koordinatenebenen, indem jeweils die entsprechende Koordinate weggelassen wird.

Um möglichst große Genauigkeit zu erhalten, sollte die Koordinatenebene gewählt werden, die am ehesten parallel zum Polygon liegt.

Strahlenverfolgung (Ray-Casting)

Dazu muss auf die Ebene projiziert werden, die senkrecht auf der Koordinate mit dem betragsmäßig größten Koeffizienten steht.

Nach der Projektion kann mit der Odd-Parity-Regel festgestellt werden, ob der Punkt innerhalb des Polygons liegt.



Strahlenverfolgung (Ray-Casting)

Ray-Tracing sollte mittels Kohärenzbetrachtungen möglichst effizient durchgeführt werden.

Unter Kohärenzbetrachtungen versteht man Überlegungen wie:

- Benachbarte Pixel erhalten ihre Färbung meistens vom selben Polygon.
- Schneidet der Strahl ein Polygon, brauchen Schnittpunkte mit weiter hinten liegenden Polygonen nicht berechnet zu werden.

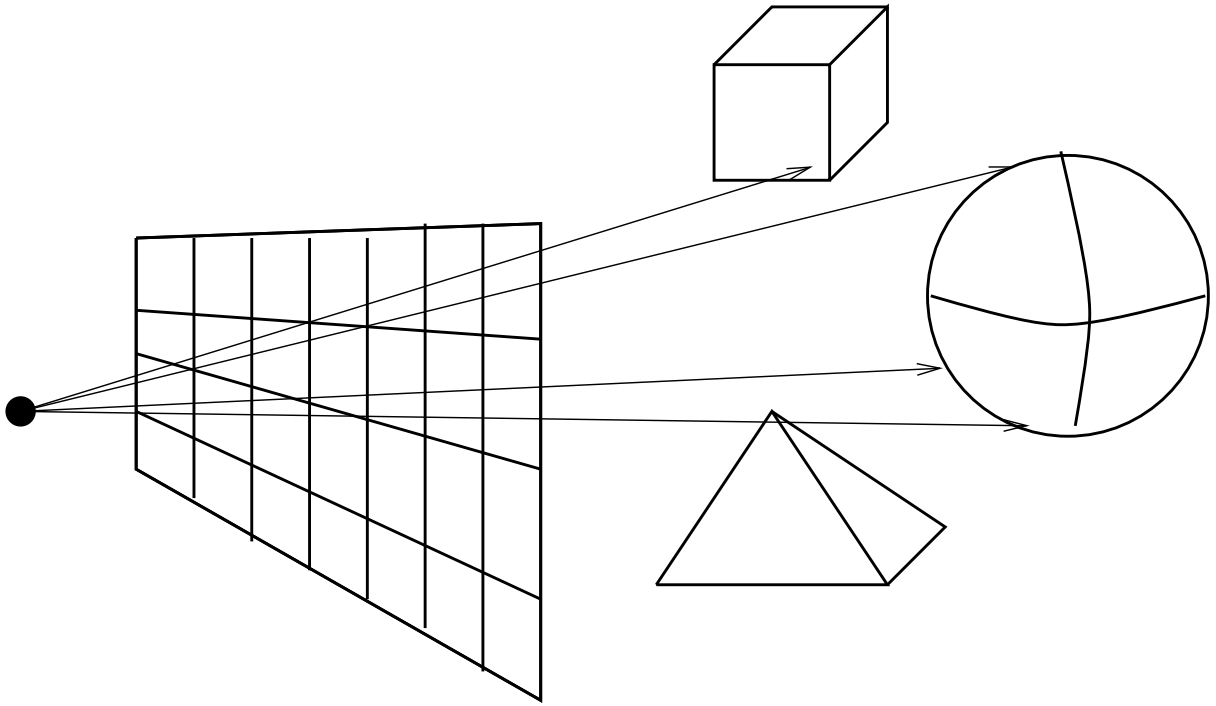
Strahlenverfolgung (Ray-Casting)

Ohne Kohärenzbetrachtungen müssen z.B. bei 1024×1024 Pixeln und 100 Objekten insgesamt $1024 \cdot 1024 \cdot 100$, d.h. ungefähr 100 Millionen Schnittpunkttests durchgeführt werden.

Durch das Ray-Tracing können Aliasing-Effekte auftreten. Bei weiter entfernten Objekten können sich die Pixelfarben stark ändern.

Zur Vermeidung wird **Supersampling** eingesetzt: Für ein Pixel werden mehrere Strahlen berechnet und der (ggf. gewichtete) Mittelwert der entsprechenden Farben berechnet.

Supersampling



Zusatzaufwand bei $m \cdot n$ Pixeln:

$$(m+1) \cdot (n+1) - m \cdot n \;=\; m + n + 1$$

Prioritätsalgorithmen

Prioritätsalgorithmen versuchen die Objekte so zu ordnen, dass das Rendering (Berechnung der Darstellung der Objekte) in dieser Reihenfolge stattfinden kann.

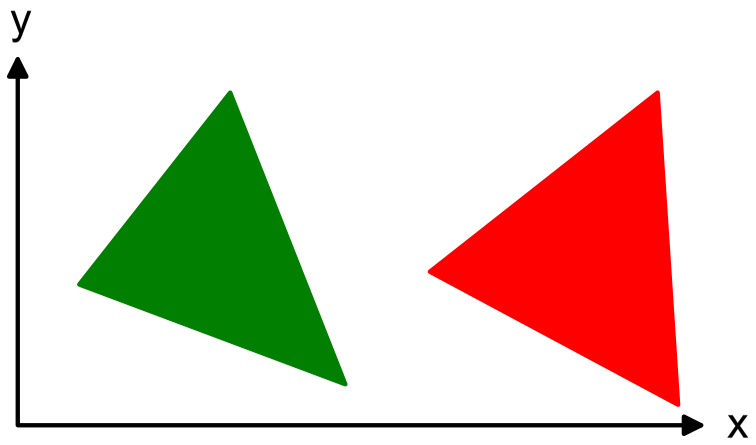
Wenn sich die z-Koordinaten von Objekten nicht überlappen, können die Objekte einfach von hinten nach vorne gerendert werden.

Weiter vorne gelegene Objekte überschreiben so die zu verdeckenden Teile der hinteren Objekte.

Prioritätsalgorithmen

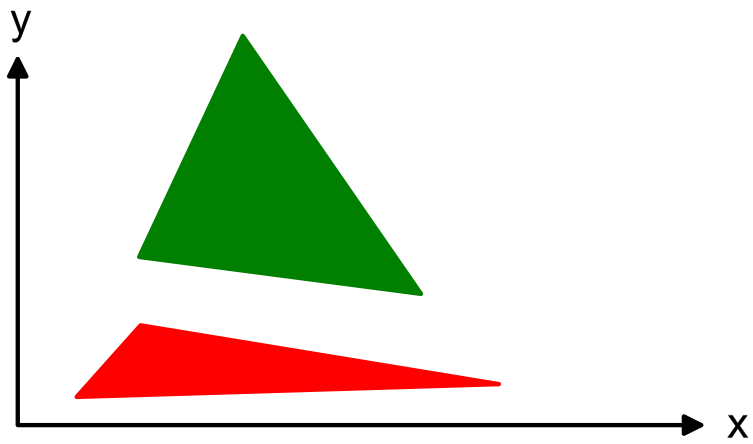
Überlappen sich die z-Koordinaten der Polygone P und Q, müssen folgende Abfragen getestet werden, um die Polygone in die richtige Reihenfolge zu bringen:

1. Ist eine Überlappung der x-Koordinaten ausgeschlossen?

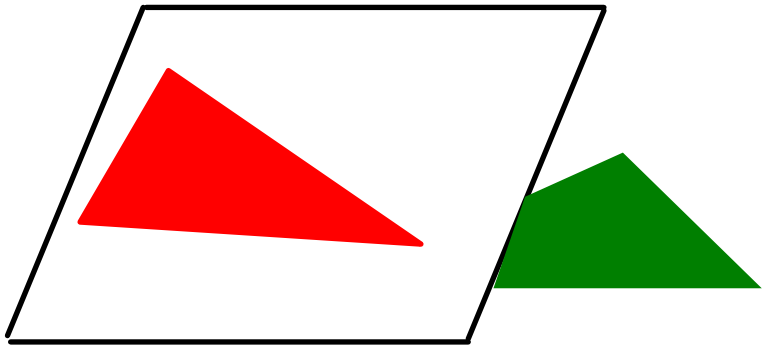


Prioritätsalgorithmen

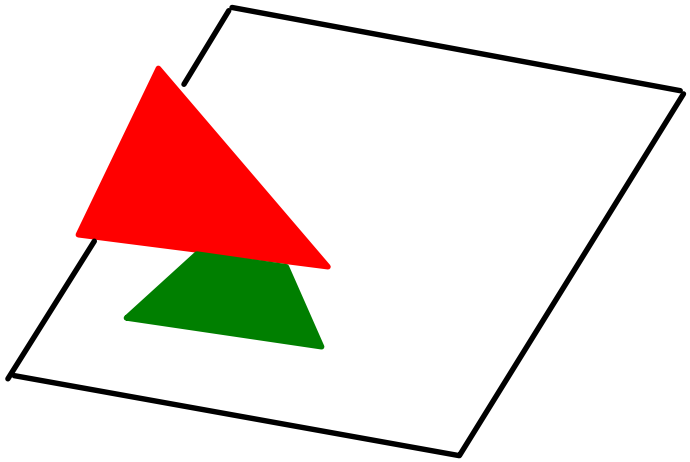
2. Ist eine Überlappung der y-Koordinaten ausgeschlossen?



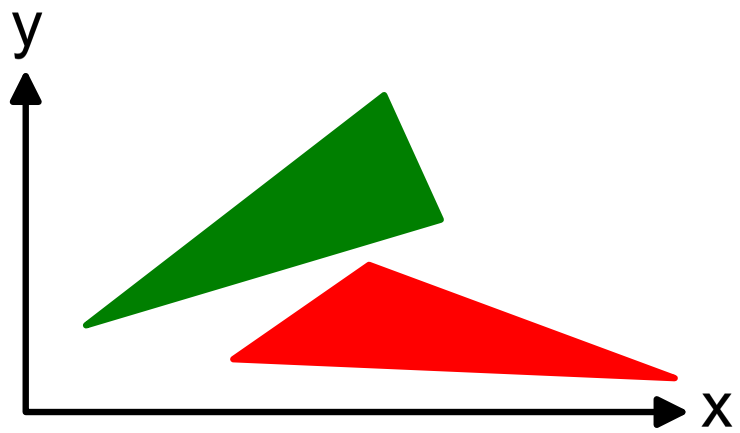
3. Liegt P vollständig auf der gegenüber liegenden Seite der zu Q gehörigen Ebene (vom Betrachtungspunkt aus gesehen) (bzw. umgekehrt)?



4. Liegt Q vollständig auf derselben Seite der zu P gehörigen Ebene (vom Betrachtungspunkt aus gesehen) (bzw. umgekehrt)?



5. Kann eine Überlappung der Projektionen auf die (x, y)-Ebene ausgeschlossen werden?



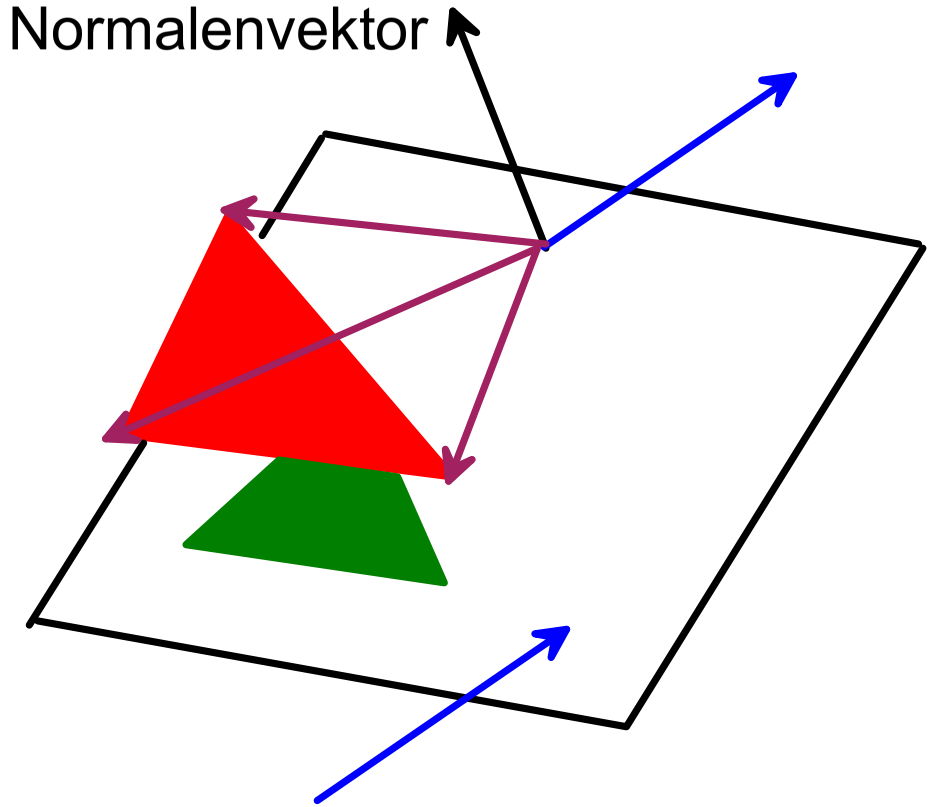
Wenn einer der fünf Fragen positiv beantwortet wird, wird P vor Q (bzw. Q vor P) eingetragen.

Ansonsten müssen die Polygone weiter unterteilt werden.

Prioritätsalgorithmen

- 1. und 2. können durch einen Vergleich der x- bzw. y-Koordinaten der Eckpunkte überprüft werden.
- 3. und 4. können durch Winkelbetrachtungen zwischen dem Normalenvektor zur entsprechenden Ebene und den Verbindungsvektoren mit den Eckpunkten des anderen Polygons überprüft werden:

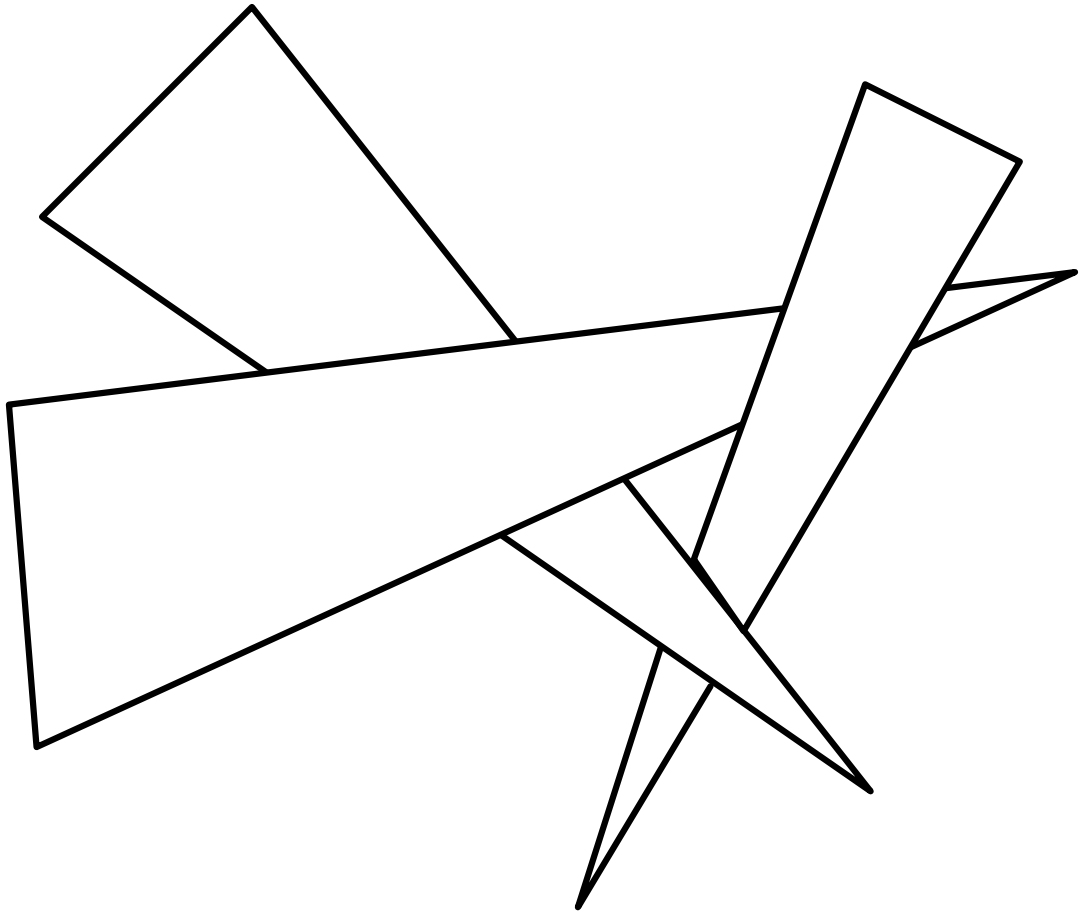
Prioritätsalgorithmen



Prioritätsalgorithmen

- Ausrichtung des Normalenvektors so, dass er mit dem (blauen) Projektionsvektor einen Winkel größer als $90°$ bildet (Skalarprodukt negativ)
- Die Verbindungsvektoren zu den Eckpunkten des (roten) Dreiecks müssen alle einen Winkel kleiner als $90°$ mit dem Normalenvektor bilden (positive Skalarprodukte).

Prioritätsalgorithmen



Ein Fall, für den keine Zeichenreihenfolge gefunden werden kann.