

# Koordinatensysteme und Clipping

Michael Olp

## Inhaltsverzeichnis

<b>1 Einführung in die perspektivische Projektion</b>	<b>1</b>
1.1 Projektion von Liniensegmenten . . . . .	1
<b>2 Koordinatensysteme</b>	<b>2</b>
2.1 Modeling . . . . .	2
2.2 View Orientation . . . . .	2
2.3 View Mapping . . . . .	3
2.4 Device Mapping . . . . .	4
<b>3 Culling und Clipping</b>	<b>4</b>
3.1 Object Culling . . . . .	4
3.2 Back Face Culling . . . . .	4
3.3 Clipping . . . . .	5

## 1 Einführung in die perspektivische Projektion

In einer realen Situation legt das Objektiv einer Kamera den sichtbaren Bildausschnitt fest und definiert damit den sogenannten Bildraum (*View Volume*). In der Computergrafik muss dieser Bildraum explizit festgelegt werden. Man verwendet hierzu einen Pyramidenstumpf (siehe Abbildung 1). Der Pyramidenstumpf (*View Frustum*) wird durch die umschließenden Ebenen definiert, die man oft *left*, *right*, *top* und *bottom* nennt. Nach vorne hin wird der Frustum

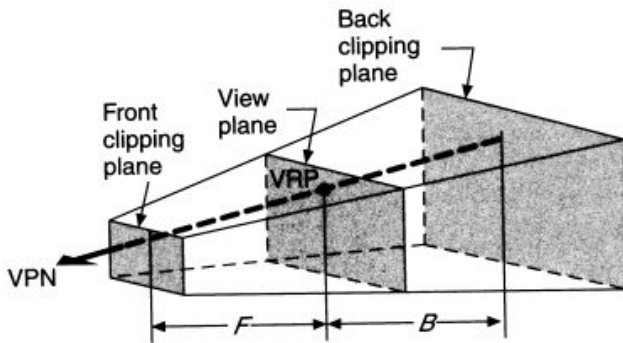


Abbildung 1: Der Bildraum

durch die vordere (*front*) und nach hinten durch die hintere (*back*) Ebene begrenzt. Es existiert eine zusätzliche Ebene auf die projiziert wird, die Bildebene (*View Plane*). Diese fällt jedoch oft auf die Near Plane. Ein auf die View Plane (definiert durch:  $\vec{N}\vec{X} = d$ ) projizierter Punkt  $P' = (x', y', z')$  entspricht dem Schnittpunkt der Geraden zwischen Kameraposition und diesem Punkt  $P = (x, y, z)$  mit der View Plane. Diese Gerade lässt sich in parameterisierter Form folgendermaßen aufschreiben:

$$\vec{Y} = (1 - t)\vec{E} + t\vec{X} \quad (1)$$

wobei  $\vec{E}$  die Position der Kamera oder des Beobachters und  $\vec{X}$  der zu projizierende Punkt ist. Wenn  $\vec{Y}$  auf der View Plane liegt, dann gilt  $\vec{N}\vec{X} = d$ . Somit lässt sich die Gleichung nach  $t$  auflösen:

$$t = \frac{\vec{N}\vec{E} - d}{\vec{N}\vec{E} - \vec{N}\vec{X}} \quad (2)$$

Um die Projektion soweit wie möglich zu vereinfachen nimmt man an, dass sich die Kameraposition bei  $\vec{E} = (0, 0, 0)$  und die Bildebene bei  $z = n > 0$  befindet, wobei  $n$  der Abstand der *Near Plane* zur Kameraposition ist. Die Normale der Bildebene ist dann  $\vec{N} = (0, 0, -1)$  mit einer Konstanten  $d = -n$ . Mit der Gleichung (2) erhält man für  $t = \frac{n}{z}$ . Für einen beliebigen Punkt  $P = (x, y, z)$  ergibt sich mit Gleichung (1) also die Projektion  $P' = (\frac{n*x}{z}, \frac{n*y}{z}, z)$ . Weiterhin kann angenommen werden, dass sich die Lage der Bildebene nicht verändert, weswegen man die projizierten Koordinaten als Tupel  $(\frac{n*x}{z}, \frac{n*y}{z})$  schreiben kann. Oft wird noch die Variable  $w = \frac{z}{n}$  definiert, womit sich dann der Punkt  $(x/w, y/w)$  ergibt.

Eine Eigenschaft der perspektivischen Projektion ist, dass sie, ähnlich wie bei einem Photo, Linien die in die Tiefe gehen, verkürzt darstellt. Parallele Linien sind nach der Projektion nur parallel sofern sie vorher auch parallel zur Bildebene waren. Weiterhin treffen sich parallele Linien in einem Fluchtpunkt.

### 1.1 Projektion von Liniensegmenten

Mit dem gewonnenen Wissen lässt sich nun zeigen, dass Liniensegmente wieder zu Liniensegmenten projiziert werden. Hierfür benötigt man die parameterisierte Form eines Liniensegmentes:

$$\vec{Q}(s) = \vec{Q}_0 + s * (\vec{Q}_1 - \vec{Q}_0) \quad \text{mit } s \in [0, 1]$$

wobei  $\vec{Q}_0$  und  $\vec{Q}_1$  Endpunkte eines Liniensegmentes sind. Seien  $\vec{P}_i = (x_i/w_i, y_i/w_i)$  mit  $w_i = z_i/n$  für  $i = 0, 1$  die projizierten Endpunkte des Liniensegmentes. Nun können wir für jedes  $s$  die Projektion  $\vec{P}(s)$  von  $\vec{Q}(s)$  bestimmen indem wir die Liniensegmentgleichung in die Projektion einsetzen:

$$\vec{P}(s) = \left( \frac{x_0 + s * (x_1 - x_0)}{w_0 + s * (w_1 - w_0)}, \frac{y_0 + s * (y_1 - y_0)}{w_0 + s * (w_1 - w_0)} \right)$$

Nun muss der Term wieder in die parameterisierte Form umgewandelt werden:

$$= \left( \frac{x_0}{w_0} + \frac{w_1 * s}{x_0 + (w_1 - w_0) * s} \left( \frac{x_1}{w_1} - \frac{x_0}{w_0} \right), \right.$$

$$\left. \frac{y_0}{w_0} + \frac{w_1 * s}{w_0 + (w_1 - w_0) * s} \left( \frac{y_1}{w_1} - \frac{y_0}{w_0} \right) \right)$$

$$= \vec{P}_0 + \frac{w_1 * s}{w_0 + (w_1 - w_0) * s} (\vec{P}_1 - \vec{P}_0) = \vec{P}_0 + s' (\vec{P}_1 - \vec{P}_0).$$

Die Projektion ist also wieder eine parametrisierte Linien-segmentgleichung mit

$$s' = \frac{w_1 s}{w_0 + (w_1 - w_0) * s} \quad (3)$$

Auf die Gleiche Art und Weise kann gezeigt werden, dass auch Dreiecke nach der Projektion wieder zu Dreiecken werden. Hierbei kann es jedoch vorkommen, dass ein Dreieck zu einer Linie degeneriert.

## 2 Koordinatensysteme in der Grafik-Pipeline

Alle Polygone müssen während ihrer Bearbeitung in der Grafikarte mehrere Stationen in der Transformationspipeline durchlaufen. Um die Erstellung von Szenen intuitiver zu gestalten wird beim Aufbau mit mehreren Koordinatensystemen gearbeitet. Welche das sind, wie sie funktionieren und wie man zwischen ihnen wechseln kann wird im Folgenden beschrieben.

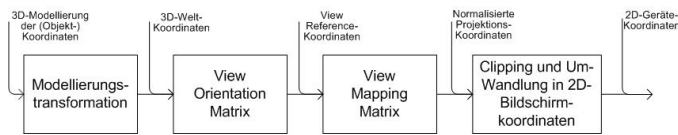


Abbildung 2: Die Grafik-Pipeline

### 2.1 Modeling

Die meisten 3D-Modelle, die für Spiele und 3D-Anwendungen erstellt werden, befinden sich in einem eigenen lokalen Koordinatensystem(*model space*). Dies hat den Vorteil, dass derjenige der das Modell erstellt sich keine Gedanken um Lage, Ausrichtung und Größe des Modells später im Spiel machen muss.

Nach der Erstellung kann jedem Objekt eine Modell Transformation(*model transform*) zugewiesen werden mit der es nach belieben verschoben, rotiert und skaliert werden kann. Weiterhin ist es möglich mehrere solcher Modell Transformationen an ein Objekt zu binden um mehrere Instanzen von ihm zu erstellen. So erreicht man, dass ein häufig gebrauchtes Modell mehrfach in einer Szene verwendet werden kann, ohne dass es nötig wäre die Geometrie vielfach zu kopieren.

Nachdem allen Objekten eine Modell Transformation

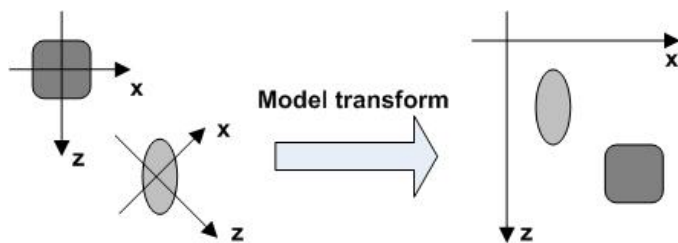


Abbildung 3: Modell Transformation

zugewiesen wurde, werden die Eckpunkte der Polygone und

die Normalen im ersten Schritt der Grafik-Pipeline<sup>1</sup>(siehe Abbildung 2) in das Weltkoordinatensystem(*world space*) transformiert. Dies geschieht indem die verwendeten Translations-, Rotations- und Skalierungsmatrizen zu einer Matrix  $H_{world}$  multipliziert und auf die Eckpunkte und Normalen angewendet werden. Diese Matrix kann von Objekt zu Objekt verschieden sein, da verschiedenen Objekten unter Umständen auch verschiedene Transformationen zugewiesen worden sind. Das Weltkoordinatensystem existiert nur einmal und alle Objekte die vorher ihr eigenes Koordinatensystem hatten, existieren nun in diesem einen. In Abbildung (3) ist dargestellt wie mehrere Objekte mit eigenem Modellkoordinatensystem in das Weltkoordinatensystem transformiert werden.

### 2.2 View Orientation

Im zweiten Verarbeitungsschritt der Grafik-Pipeline kommt nun ein virtueller Betrachter oder auch eine Kamera ins Spiel. Ein einfaches Kamerasystem lässt sich folgendermaßen definieren:

Die Kamera befindet sich im Ursprung des Weltkoordinatensystems und blickt in Richtung negativer z-Achse. Weiterhin gibt es zwei Vektoren, die die Richtung nach oben und links anzeigen. Ohne diese Vektoren gäbe es beliebig viele Kameraeinstellungen, da sich die Kamera um die z-Achse rotieren ließe. Dieses Koordinatensystem wird auch Kamerakoordinatensystem oder Augenkoordinatensystem(*eye space*) genannt.

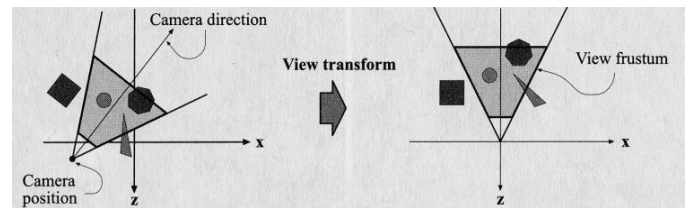


Abbildung 4: View Transform

Generell jedoch lässt man ein allgemeineres Kamerasystem zu, um dem Anwender mehr Spielraum zu geben. Im Grunde macht es keinen Unterschied ob man ein Objekt vor einer festen Kamera bewegt oder man die Kamera vor einem festen Objekt bewegt. Für den Anwender ist es jedoch intuitiver während des Modellings Objektpositionen zu verändern und dann während der View Orientation Phase die Lage der Kamera. OpenGL bietet zum Beispiel, neben den üblichen Translationen und Rotationen, eine spezielle Funktion *gluLookAt(...)* um gezielt die Kamera auf einen bestimmten Punkt zu richten.

Für dieses Kameramodell nehmen wir nun an, sie befindet sich im Punkt  $\vec{E}$  (*eye point*), zeige in die Richtung  $\vec{D}$  und habe die Left- und Upvektoren  $\vec{L}$  und  $\vec{U}$ , so dass  $\vec{L}$ ,  $\vec{U}$  und  $\vec{D}$  ein rechtshändiges Koordinatensystem bilden. Zusammen bilden sie die orthonormale Matrix  $R = (\vec{L}|\vec{U}|\vec{D})$ . Ein beliebiger Punkt  $\vec{X}$  aus dem Weltkoordinatensystem kann nun durch das Kamerakoordinatensystem beschrieben werden:

$$\vec{X} = \vec{E} + R\vec{Y}$$

<sup>1</sup>Die Abbildung beschränkt sich auf die für diese Ausarbeitung wichtigen Teile.

Um nun den Punkt  $\vec{X}$  bezüglich des Kamerakoordinatensystems zu erhalten, muss die Gleichung nach  $\vec{Y}$  umgestellt werden:

$$\vec{Y} = R^{-1} (\vec{X} - \vec{E})$$

Grafisch gesehen bedeutet diese Gleichung, dass die Kamera mit beliebiger Position und Ausrichtung in die Lage der, im Abschnitt zuvor genannten, einfachen Standardkamera verschoben wird. Diese *view transformation* (siehe Abb.4) kann nun als Matrix

$$H_{view} = \left( \begin{array}{c|c} R^{-1} & -R^{-1}\vec{E} \\ \hline \vec{0}^T & 1 \end{array} \right) \text{ ausgedrückt werden.}$$

### 2.3 View Mapping

Man könnte nun beginnen mit den Gleichungen aus Sektion (1) die Objekte auf die Bildebene zu projizieren, doch es wird noch ein weiterer Transformationsschritt ausgeführt. Dies hat zwei wichtige Gründe. Zum einen gehen bei der perspektivischen Projektion die Tiefenwerte verloren, da die Operation nicht invertierbar ist. Doch diese Tiefenwerte werden später im Rasterizer benötigt um zu ermitteln mit welcher Farbe die einzelnen Pixel gezeichnet werden müssen. Der andere Grund ist, dass es neben der perspektivischen Projektion noch eine orthographische Projektion gibt bei der ebenfalls Clipping- und Cullingoperationen ausgeführt werden. Wie wir sehen, werden bei beiden Projektionen die Bildräume in einen Einheitswürfel überführt, was zur Folge hat, dass man die Clipping und Culling Hardware nur einmal auf dem Grafikchip implementieren muss.

Aus dem vorigen Pipelineschritt steht uns ein Standardkamera Modell zur Verfügung, dass im Ursprung liegt und in die negative z-Richtung ausgerichtet ist. Der Bildausschnitt auf der Bildebene ist durch die Werte  $l, r, t$  und  $b$  gegeben, wobei gilt:  $l \leq x \leq r$  und  $b \leq y \leq t$ . Nun wird der Bildraum in Form eines Pyramidenstumpfes in einen Einheitswürfel mit Kantenlänge 2 transformiert. Dieser Bildraum wird dann kanonischer Bildraum und das zugehörige Koordinatensystem dreidimensionales normalisiertes Bildschirmkoordinatensystem (*normalized device coordinates*) genannt. Die Bezeichnung "normalisiert" rührt daher, dass alle Koordinaten im Intervall  $[-1, 1]$  liegen. Abbildung 5 illustriert den Vorgang. Um dies zu erreichen, wird zuerst der Bildausschnitt

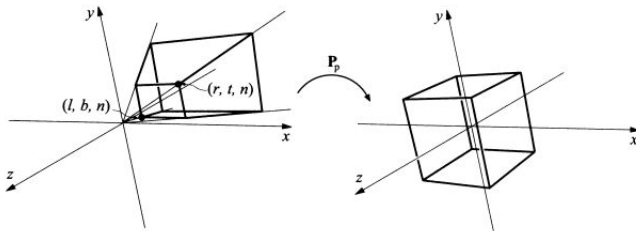


Abbildung 5: Vom Bildraum zum kanonischen Bildraum

auf der Bildebene so transformiert, dass  $x$  und  $y$  aus dem Wertebereich  $[-1, 1]$  sind. Dies sorgt auch dafür, dass ein "verschobener" Bildausschnitt, also bei dem z.B.  $l \neq -r$  ist, richtig auf  $[-1, 1]$  abgebildet wird. Hierzu wird eine Gerade durch den eye point und die Mitte des Bildausschnitts gelegt:

$$((r+l)z/(2n), (t+b)z/(2n), z) \quad z \in [n, f]$$

Nun werden alle x- und y-Koordinaten mit Hilfe dieser Gerade verschoben und anschließend auf den Wertebereich skaliert. Dies ergibt folgende Gleichungen für die neuen x,y-Werte:

$$x' = \frac{2}{r-l} \left( x - \frac{(r+l)z}{2n} \right)$$

$$y' = \frac{2}{t-b} \left( y - \frac{(t+b)z}{2n} \right)$$

Jetzt verbleiben nur noch die z-Koordinaten. Eine lineare Transformation wie bei den x- und y-Koordinaten ergibt hier aber ein falsches Ergebnis. Der Grund hierfür liegt in der perspektivischen Transformation und den dabei entstandenen Faktoren  $s$  und  $s'$ . Wenn man diese beiden gegeneinander abträgt, stellt man fest, dass ein nicht-linearer Zusammenhang besteht: Eine Auswirkung davon ist, dass Punk-

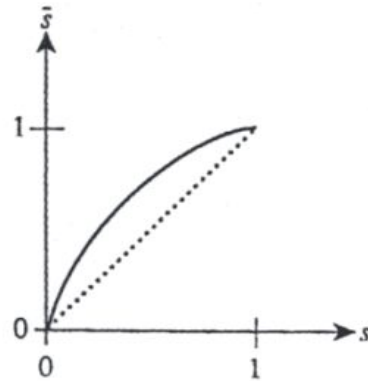


Abbildung 6: Zusammenhang zwischen  $s$  und  $s'$

te auf einer Geraden im Frustum, die vorher alle den gleichen Abstand zueinander hatten, nach der Transformation nicht mehr gleichmässig verteilt sind. Dies kann in einem späteren Pipelineschritt zu Fehlern führen, weshalb man es an dieser Stelle gleich bereinigt. Hierzu benötigen wir den Faktor  $s'$  (Gleichung 3) der im Abschnitt 1.1 hergeleitet wurde. Die z-Werte zwischen  $n$  und  $f$  sind abbildbar durch  $z = (1-s)n + sf$  für  $s \in [0, 1]$ . Diese Gleichung wird nach  $s$  umgestellt

$$s = \frac{z-n}{f-n}$$

und in Gleichung (3) eingesetzt. Mit den Werten  $w_0 = 1$  und  $w_1 = \frac{f}{n}$  ergibt sich:

$$s'(s) = z' = \frac{f}{f-n} \left( 1 - \frac{n}{z} \right).$$

Diese Projektion lässt sich ebenfalls wieder als Matrix darstellen, wobei in dieser schon eine Skalierungsmatrix

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

integriert ist, die dafür sorgt, dass das rechtshändige Koordinatensystem in ein linkshändiges umgewandelt wird. Dies wird aus Konsistenzgründen gemacht, da bei einem linkshändigen System die positive z-Achse in den Bildschirm

zeigt, und somit größere z-Werte auch größere Entfernungen anzeigen. Im folgenden werden nun zwei Matrizen eingeführt, da die beiden großen APIs OpenGL und Direct3D die z-Werte einmal auf das Intervall  $[0, 1]$  und einmal auf  $[-1, 1]$  abbilden. Beiden nennt man jedoch  $H_{proj}$ .

$$H_{[0,1]} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f}{f-n} & -\frac{fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$H_{[-1,1]} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Insgesamt haben wir nun drei Matrizen mit denen jeder Punkt, der durch die Grafik-Pipeline geschickt wird, multipliziert wird:

$$H_{total} = H_{proj}H_{view}H_{world}$$

## 2.4 Device Mapping

In einem letzten Schritt werden nun die Punkte aus dem kanonischen Bildraum in ein Bildschirmkoordinatensystem (*screen space*) umgewandelt. Ein Bildschirm hat ein eigenes Koordinatensystem, dessen Ursprung in der unteren linken Ecke ist. Die obere rechte Ecke, welche den Bildschirm begrenzt hat die Koordinaten  $(S_x, S_y)$ . Folgende Gleichungen werden verwendet um einen Punkt in Bildschirmkoordinaten (*screen coordinates*) auszudrücken:

$$\bar{x} = \frac{(S_x - 1)(x + 1)}{2}, \bar{y} = \frac{(S_y - 1)(y + 1)}{2}$$

Aus Film und Fernsehen ist man verschiedene Bildseitenverhältnisse (*aspect ratio*) gewöhnt. In einem normalen Fernseher hat man oft das Verhältnis 4:3, im Kino 16:9. Bei der Umwandlung in Bildschirmkoordinaten kann dieses Verhältnis gewählt werden, in dem man den Quotienten aus  $S_x$  und  $S_y$  entsprechend wählt. Dabei muss man beachten, dass man die Verhältnisse der Kamera zu Beginn ähnlich wählt, da es sonst zu Verzerrungen kommen kann.

## 3 Culling und Clipping

Sowohl Clipping als auch Culling sind Methoden um die Performanz der Grafikpipeline zu erhöhen. Dies wird durch Weglassen von Objektteilen realisiert, die nicht sichtbar sind, da sie entweder außerhalb des Bildraumes liegen oder von anderen Objekten verdeckt werden.

### 3.1 Object Culling

Beim Object Culling (oft auch *Frustum Culling*) wird versucht, diejenigen Objekte auszusortieren, die nicht von der Kamera gesehen werden können da sie außerhalb des Bildraumes liegen. Oft werden Objekte in einer hierarchischen Struktur gespeichert, die auch eine Bounding Box<sup>2</sup> für die Objekte enthält (siehe Abb.7). Mit Hilfe der Bounding

<sup>2</sup>Eine Bounding Box ist ein Quader der alle Polygone eines Objektes enthält.

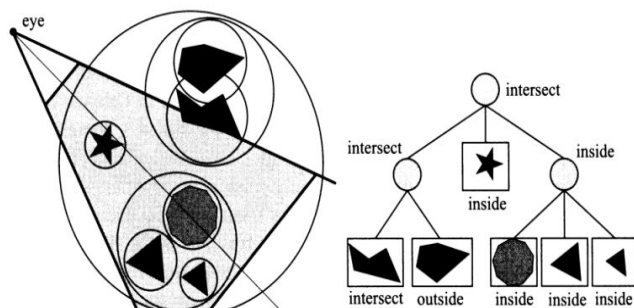


Abbildung 7: Szenengraph mit Bounding Volumes

Boxen kann mit relativ wenig Aufwand getestet werden, ob ein Objekt innerhalb, außerhalb oder teilweise im Bildraum liegt. Hierfür wird zuerst die Wurzel getestet. Liegt sie außerhalb des Bildraumes kann der ganze Baum verworfen werden. Liegt sie nur teilweise im Raum wird rekursiv jeder Teilbaum geprüft. Wenn ein Teilbaum ganz im Bildraum liegt werden alle Teilbäume geprüft. Stößt man auf ein Blatt, also ein konkretes geometrisches Objekt, so wird es gezeichnet.

### 3.2 Back Face Culling

Wenn ein Objekt beim Object Culling nicht vom Zeichnen ausgeschlossen werden konnte, gibt es beim back face Culling nun die Möglichkeit einzelne Dreiecke nicht zeichnen zu lassen. Für gewöhnlich werden 3D-Objekte so konstruiert, dass die Normalen ihrer Dreiecke nach außen zeigen. Wenn nun die Normale eines Dreiecks vom Betrachter weg zeigt, bedeutet dies also, dass der Betrachter die Rückseite des Dreiecks sieht. Wenn die Kamera außerhalb des Objektes ist, wird dieses Dreieck vom restlichen Objekt verdeckt, weswegen man dieses Dreieck vom Zeichnen ausschließen kann.

Mathematisch kann dies durch das Skalarprodukt ausgedrückt werden. Sei  $\vec{E}$  wieder der Betrachterstandpunkt und sei  $\vec{N} \cdot \vec{X} = d$  die Ebene in der das zu betrachtende Dreieck liegt. Wenn das Skalarprodukt  $\vec{N} * \vec{E}$  kleiner ist als 0, dann sieht der Betrachter die Rückseite des Dreiecks.

Bei größeren 3D-Modellen werden die Normalen der Dreiecke oft mitgespeichert um die CPU zu entlasten. Sie befinden sich, genauso wie die Koordinaten der Eckpunkte, im Modellkoordinatensystem. Anstatt nun alle Eckpunkte und Normale in das Weltkoordinatensystem zu überführen um den Cullingprozess durchzuführen, genügt es, die Kamera zurück in das Modellkoordinatensystem zu transformieren. Hierzu muss die Translation, die Rotation und die Skalierung rückgängig gemacht werden, was sich mathematisch als

$$\vec{E}_m = \frac{1}{s} R^t (\vec{E} - \vec{T})$$

ausdrücken lässt.  $\vec{E}_m$  ist der eye point in Modellkoordinaten. Nun kann wieder über das Skalarprodukt bestimmt werden, ob der Betrachter auf die Vorder- oder Rückseite des Dreiecks schaut.

### 3.3 Clipping

Beim Clipping werden alle Dreiecke im Bildraum so an den Bildraumbenen gekappt, dass nach dem Vorgang nur noch Dreiecke übrig sind, die auch tatsächlich ganz im Bildraum liegen. Hierfür wird wieder über das Skalarprodukt bestimmt ob die Eckpunkte einer Kante des Dreiecks einmal auf der positiven Seite der Bildraumbene und einmal auf der negativen Seite liegen. Für die Ebene nehmen wir wieder die Gleichung  $\vec{N} \cdot \vec{X} = d$  und die Eckpunkte des Dreiecks sind durch  $\vec{V}_i$  für  $i = 0, 1, 2$  gegeben. Eine Kante schneidet die Ebene also wenn gilt, dass  $p_{i_0} p_{i_1} < 0$  für  $p_i = \vec{N} \cdot \vec{V}_i - d$  und  $i = 0, 1, 2$ . Abbildung (8) zeigt vier Mögliche Konfiguratio-

*Einführung in die Computergraphik*, James Foley u.a., 1994

*3D Transformation und Kamerapositionierung*, Benjamin Zimmer, 2003

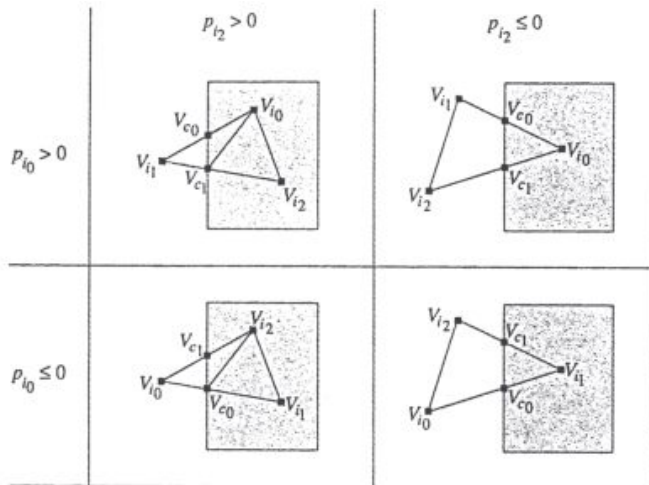


Abbildung 8: Die vier möglichen Clippingkonfigurationen

nen die beim Clipping auftreten können. Um herauszufinden welcher der vier Fälle bei einem Dreieck zutrifft geht man folgendermaßen vor. Zuerst werden  $p_0, p_1$  und  $p_2$  bestimmt. Der einfachste Fall ist nun, dass alle drei Werte größer als 0 sind, das Dreieck also komplett im Bildraum liegt. Ist dies nicht der Fall, kann o.B.d.A. angenommen werden, dass die Kante zwischen den Eckpunkten  $\vec{V}_{i_0}$  und  $\vec{V}_{i_1}$  die Ebene schneidet. Nun wird ermittelt ob  $\vec{V}_{i_0}$  oder  $\vec{V}_{i_1}$  außerhalb liegt, in dem man einfach nur die entsprechenden  $p_i$  Werte betrachtet. Danach kann mit  $p_2$  endgültig bestimmt werden welcher der vier Fälle vorliegt und die Schnittpunkte mit der Bildraumbene werden berechnet. Hierfür verwendet man folgende Gleichung:

$$\vec{V}_{clip} = \vec{V}_{i_0} + \frac{p_{i_0}}{p_{i_0} - p_{i_1}} (\vec{V}_{i_1} - \vec{V}_{i_0})$$

Sofern ein Viereck beim Clipping entstanden ist, wird es in zwei Dreiecke aufgeteilt, wobei die Reihenfolge der Eckpunkte erhalten bleibt.

Der Vorteil des Clippens gegen den Einheitswürfel ist, dass die Berechnung ob ein Punkt sich innerhalb oder außerhalb des Bildraumes befindet sehr schnell ist. Dies liegt daran, dass die Normalen des Einheitswürfels nur in einer Komponente eine 1 oder -1 enthalten und das Skalarprodukt somit zu einer einfachen Multiplikation wird.

### Literatur

*Real-Time Rendering*, Tomas Möller, Eric Haines, 1999