

# **Vorwissen**

Zum besseren Verständnis von virtuellen Kameras in der Computer Grafik, speziell der Echtzeit Computergrafik, empfehlen wir die folgenden Inhalte vorab zu sichten und zumindest grundlegend durchzuarbeiten:

1. Unsere Vorlesungen zu den Themen
   1. Vektorrechnung (YouTube) Hier noch Links einfügen
   2. Basistransformationen (YouTube)
   3. Komposition von Transformationen (YouTube)
2. Optional aber sehr empfehlenswert zur Vertiefung: Die sehr anschauliche und interaktive Einführung zu den physikalischen Grundlagen von Kameras und Linsen von Bartosz Ciechanowski

# **Nach dieser Einheit solltet ihr…**

* die Eigenschaften/Attribute einer virtuellen Kamera kennen
* die einzelnen Schritte der Kameratransformation beschreiben können
* verschiedene Koordinatenräume kennen und zuordnen können
* die Rolle homogener Koordinaten für perspektivische Verzerrung nachvollziehen können

**Motivation und Problemstellung**

Ziel dieser Einheit soll es sein, zu verstehen, wie virtuelle Kameras in der Theorie und in der praktischen Umsetzung funktionieren und wie wir damit effizient unsere Szene darstellen können.

Bevor wir uns jedoch genauer mit der Umsetzung beschäftigen, müssen wir uns zunächst erarbeiten, was überhaupt eine virtuelle Kamera ausmacht, welche Parameter dafür wichtig sind, und auch welche Effekte an dieser Stelle *nicht* realisieren wollen.

Dazu wollen wir uns ein paar Aufnahmen einer Szene aus dem Spiel *Portal 2* ansehen. Welche Eigenschaften einer virtuellen Kamera kannst du daraus ableiten?

Ein Bild, das Text enthält.

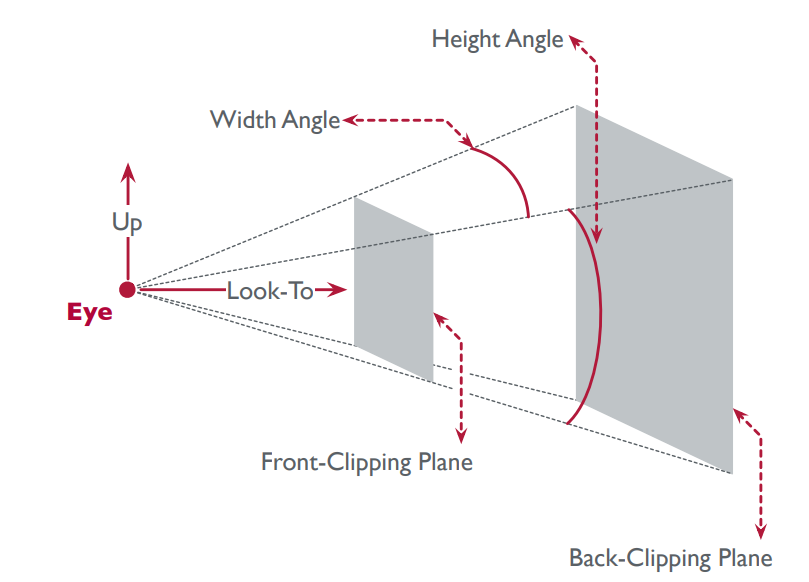
Automatisch generierte Beschreibung+ birds eye  
+orthografische proj  
+auf dem kopf

Mach dir erst einmal selbst Gedanken, welche Attribute der Kamera sich auf den verschiedenen Ansichten unterscheiden, du kannst dieses Feld für Notizen verwenden:

Antworten hier

Folgende Parameter nehmen Einfluss auf die Aufnahme einer virtuellen Kamera:

* Kamerastandpunkt  
  Dieser bestimmt, von welcher Position aus die Szene gerendert wird. Bei einer First-Person-Ansicht wäre das z.B. die Position des gespielten Charakters, bei einer Vogelperspektive ein Punkt über dem Geschehen
* Kamerablickrichtung  
  In welche Richtung ist die Kamera gedreht, bzw. wo liegt der Mittelpunkt des durch die Kamera erzeugten Bildes?
* Aufwärtsrichtung  
  Wo ist „oben“?
* Sichtwinkel (auf Bildbeispiel achten!)  
  Beschreibt die Größe des Sichtfeldes in vertikaler und horizontaler Richtung
* Seitenverhältnis/aspect ratio Infobox Zuordnung verschiedener Aspect Ratios   
  Beschreibt das Verhältnis von Breite zu Höhe des durch die Kamera erzeugten Bildes
* Clipping Planes -> Bsp. aus Spielen? bereits grob motivation andeuten und auf später verweisen  
  Die Near und Far Clipping Planes beschreiben, ab und bis zu welcher Tiefe (=Abstand von der Kamera) Objekte gerendert werden. Das kann unter anderem dazu verwendet werden, Objekte außerhalb dieses Intervalls früh im Renderingprozess zu verwerfen und dadurch den nötigen Aufwand deutlich zu verringern.



Unser Modell der virtuellen Kamera soll also eine 3D-Szene aus einer bestimmten Perspektive darstellen und die dafür nötigen geometrischen Transformationen beinhalten.

Einschub Kameramodelle (first person etc.)

Bei „realen“, physischen Kameras entstehen noch einige weitere Effekte, die aber *nicht* Teil unseres Modells der virtuellen Kamera sind. Dazu gehören u.a.

* Depth of Field (DoF), also Tiefenunschärfe
* Verzerrung durch Linse
* Chromatische Aberrationen (<https://de.wikipedia.org/wiki/Chromatische_Aberration>)
* Weitere Linseneffekte

Diese können durch Post-Processing ebenfalls umgesetzt werden, das ist jedoch nicht Teil dieses Kapitels. Wer mehr zu Linseneffekten und den physikalischen Grundlagen von Kameras wissen möchte, kann sich [auf dieser Seite](https://ciechanow.ski/cameras-and-lenses/) weiter dazu informieren.

**Umsetzung und Mathematisches Modell**

Ziel

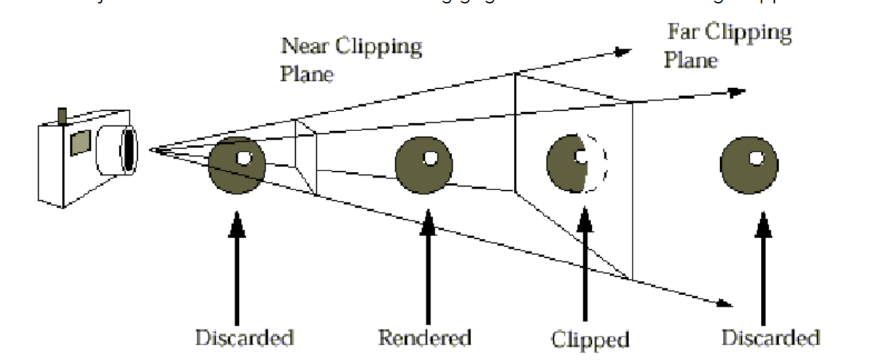
Im Fragment-Shader soll nur auf Objekte zugegriffen werden, die auch tatsächlich im gerenderten Bild auftauchen – also nicht auf Objekte, die z.B. zu weit rechts, links oder oben sind oder die zu nah an oder zu weit weg von der Kamera sind.

Rasterisiert und dementsprechend im Fragment Shader verfügbar werden nur Objekte, die nach den vorherigen Renderingstufen Koordinaten im Intervall [-1, 1] liegen. Dabei beschreiben x und y Koordinate die Position im gerenderten Bild, die z-Koordinate die Tiefe (Normalized Device Coordinates).

Unser Kameramodell sollte also die Szene so transformieren, dass die im Bild sichtbaren Objekte in diesem Koordinatenbereich liegen und korrekt verzerrt sind.

Unser Ziel ist demnach die Transformation der Koordinaten aller Objekte vom Weltkoordinatensystem in Normalized Device Coordinates über mehrere Zwischenschritte.

Über folgende Transformationen wollen wir unser Endergebnis erreichen:

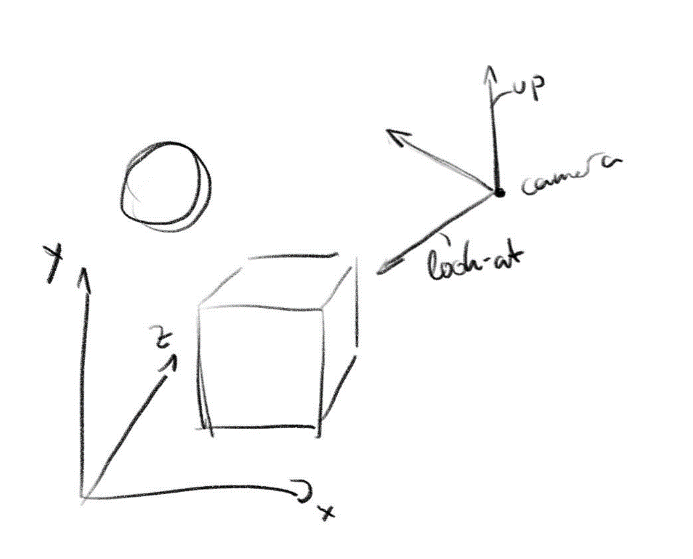
1. **Modellkoordinaten 🡪 Weltkoordinaten)**  
   Punkte sind meist im Koordinatensystem des Objektes, zu dem sie gehören, gegeben. Wenn wir beispielsweise das Modell eines Hauses betrachten, das möglicherweise sogar mehrmals in der Szene vorkommt, wird die Position des Dachs in Relation zum Rest des Hauses beschrieben und nicht als globale Position in der Szene. Diese Koordinaten müssen also zuerst umgewandelt werden, das ist jedoch nicht Teil unserer Kameratransformation
2. **Weltkoordinaten 🡪 Kamerakoordinaten**  
   Wir wollen die Szene aus Sicht der Kamera darstellen. Der erste Schritt dafür ist, alle Positionen in Relation zur Kamera anzugeben.
3. **Kamerakoordinaten 🡪 Clip-Koordinaten**  
   In diesem Schritt erfolgt die perspektivische Transformation. Dabei werden u.a. Sichtwinkel und Clipping Ebenen mit einbezogen. Nach dieser Transformation werden alle Objekte ignoriert, die außerhalb des Sichtvolumens der Kamera liegen (clipping).

**Einschub Sichtvolumen (view frustum)**

Das Sichtvolumen ist der Bereich einer Szene, in dem Objekte von der virtuellen Kamera erfasst werden. In unserer Modellierung einer (perspektivischen) Kamera ist dieses Sichtvolumen ein Pyramidenstumpf.

1. **Clip-Koordinaten 🡪 NDC**  
   Hier werden die homogenen Koordinaten wieder in R^3-Koordinaten umgewandelt und liegen schließlich im Bereich [-1,1]

(4. **NDC 🡪 Bildschirmkoordinaten**)  
Abbildung in das Bildkoordinatensystem, definiert durch Position, Breite und Höhe in Pixeln

Diese Einzelschritte können wir durch lineare Abbildungen realisieren. Wenn du dich damit noch nicht sicher fühlst, sieh dir gerne noch einmal die Einheit zu [geometrischen Transformationen](https://www.youtube.com/watch?v=YDDQE45PZLs) an.

Wir wollen nun die Transformationen für die einzelnen Schritte herleiten.

**View Transform (Weltkoordinaten 🡪 Kamerakoordinaten)**

Mit dieser Transformation soll die Szene so transformiert werden, dass sich die Kamera im Ursprung befindet und die z-Achse der Blickrichtung (look-to) entspricht, um damit die Szene aus Sicht der Kamera zu beschreiben.

Überlege dir kurz, welche Basistransformationen du mit welchen Parametern nutzen kannst, um diese Transformation umzusetzen.

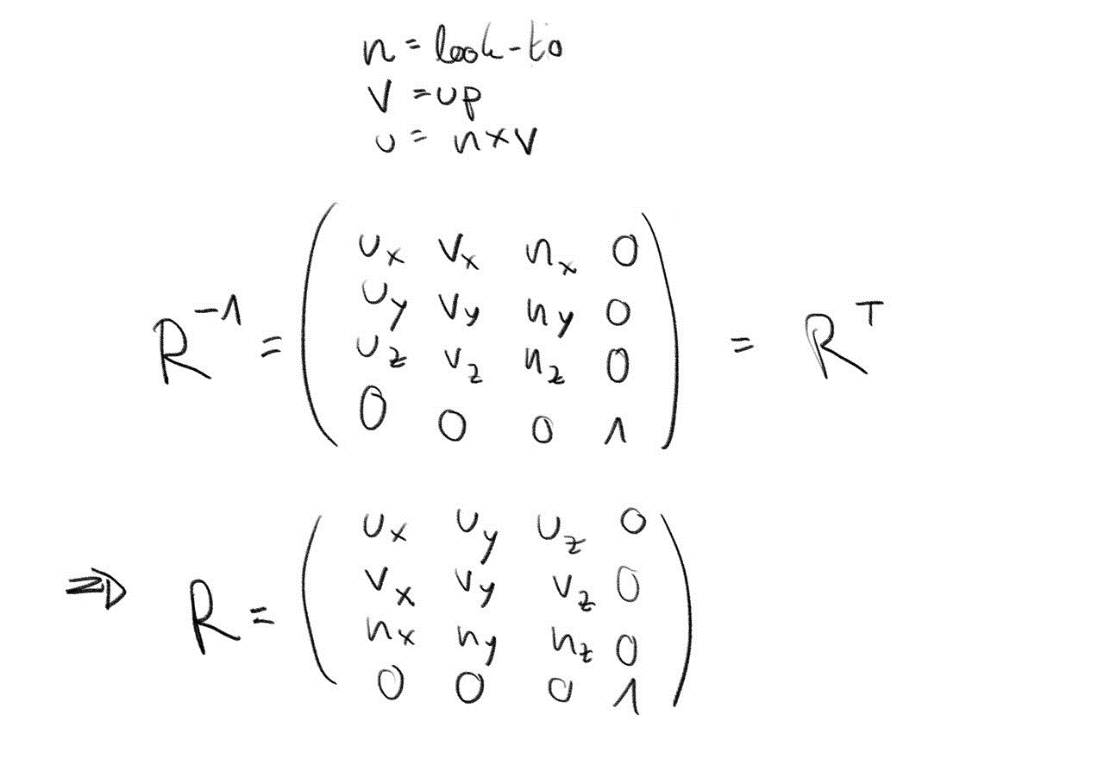
Antworten hier

Wir wollen die Transformation aus zwei Teiltransformationen zusammensetzen: (1) Der Translation des Kamerastandpunkts (eye) in den Koordinatenursprung und (2) der Rotation in das Kamerakoordinatensystem.

Den Parameter für die Translation haben wir mit bereits gegeben.

Für die Rotation können wir ausnutzen, dass es sich um eine orthogonale Transformation handelt: Wir transformieren eine Orthonormalbasis (Aufwärtsvektor , Kamerablickrichtung und das Kreuzprodukt der beiden -) in eine andere (y-, z- und x-Achse). Damit gilt für diese Transformation und ihre Inverse .

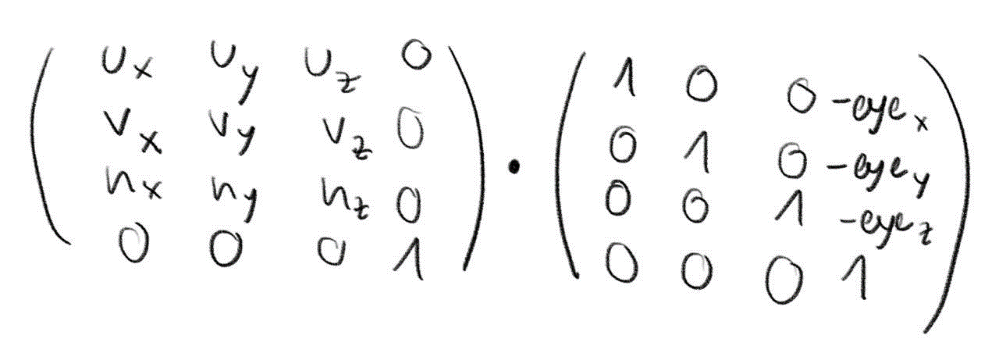
Finden wir also eine Transformationsmatrix, die die x-, y- und z-Achse in ihre äquivalenten Achsen des Kamerakoordinatensystems rotiert, können wir daraus einfach deren Inverses rekonstruieren – dies entspricht dann genau der Matrix, die wir suchen.

Wie erhalten wir also eine solche Matrix? Versuche, sie dir herzuleiten. Bekannt ist, dass sie die y-Basis auf abbilden soll, die z-Basis auf und die x-Basis auf .

Antworten hier (ergibt für Formeln möglicherweise nicht so viel Sinn, aber auf jeden Fall sollte hier Platz sein, damit nicht die Antwort direkt sichtbar ist)

Die erste Spalte der Matrix gibt an, worauf abgebildet wird, die zweite worauf abgebildet wird etc.

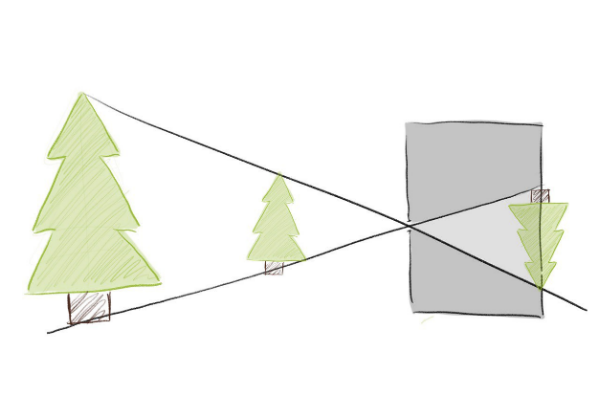
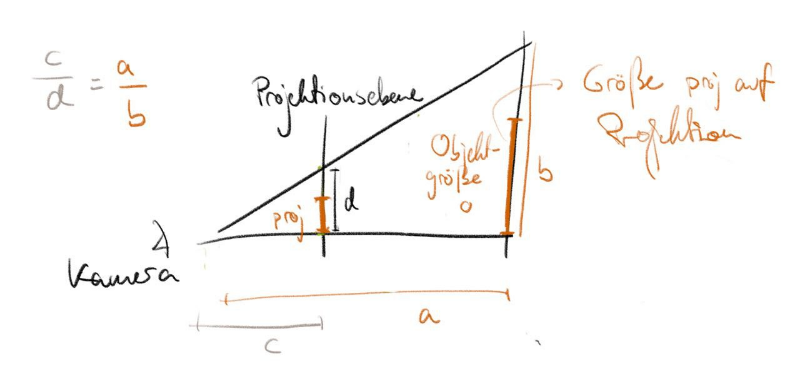
Wir können unsere gegebenen Beziehungen also einfach einsetzen und erhalten unsere gesuchte Matrix. Transponiert entspricht diese unserer gesuchten Rotation R.

Die gesamte view transform entspricht also

**Projection Transform (Kamerakoordinaten 🡪 Clip-Koordinaten)**

Mit diesem Transformationsschritt wollen wir Perspektivische Projektion unter Beachtung von Blickwinkeln sowie near und far clipping plane realisieren.

Dafür müssen wir uns zunächst überlegen, wie sich perspektivische Verzerrung auf das Bild auswirkt und wie wir den Effekt mathematisch beschreiben können. Unsere Intuition sagt uns bereits, dass nähere Objekte größer erscheinen als weit entfernte Objekte.

Mit dem Strahlensatz können wir auch beschreiben, wie groß genau ein Objekt auf einer Projektionsebene erscheint.

# **Einschub – Projektionsebene bitte prüfen, ob das so stimmt**

Was heißt denn nun, dass wir eine Projektionsebene verwenden? Normalerweise projiziert eine Kamera doch nichts?

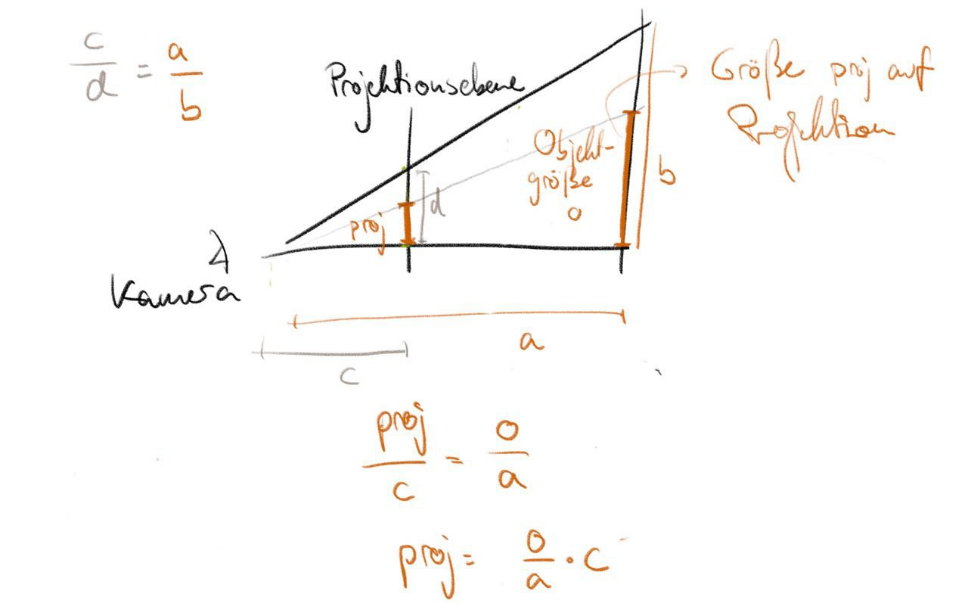
Dazu können wir uns die vereinfachte Funktionsweise einer simplen Lochkamera ansehen:

Dabei entsteht das umgedrehte Bild auf der  
Rückwand der Lochkamera, nachdem das  
Licht durch das Loch „gebündelt“\* wird. Die  
Rückwand der Kamera ist also die  
Projektionsebene, auf der das Bild entsteht.

Wir wollen jedoch nicht physikalisch Licht-  
strahlen erfassen, sondern nur berechnen,  
wie groß die Gegenstände in Relation  
zueinander sind. Wir können die Projektionsebene also beliebig – auch vor der Kamera – platzieren und mit dem Strahlensatz bestimmen, wie groß Objekte auf dieser Ebene erscheinen.

Für unsere Anwendung ist die Projektionsebene also eine Referenzebene, um zu steuern, wie andere Objekte skaliert werden.

Angenommen, wir kennen also den Abstand der Projektionsebene von der Kamera. Wie ermitteln wir, wie groß ein Objekt mit Abstand a auf der Projektionsebene erscheint?



Antworten hier (ergibt für Formeln möglicherweise nicht so viel Sinn, aber auf jeden Fall sollte hier Platz sein, damit nicht die Antwort direkt sichtbar ist)

Wir wissen, dass die Objektgröße o im Verhältnis zu b genauso groß ist wie die projizierte Größe proj im Verhältnis zu d.

WIR SUCHEN PROJ/D