HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Computergrafik I

Kapitel 5: Kameramodell und Projektionen

Prof. Dr. Ingo Ginkel

Sommersemester 2022

Kameramodell, Projektionen und Objektselektion - Kapitelübersicht

- Kamera (LookAt-Matrix)
- Projektionen und Projektionsmatrizen
- View Frustum
- Selektion/Markierung von Objekten (Picking)
- Bounding Volumes (AABB, Kugel)

Kameramodell

Um ein „Foto“ einer virtuellen Szene zu machen ist ein Kameramodell mit folgenden Informationen notwendig:

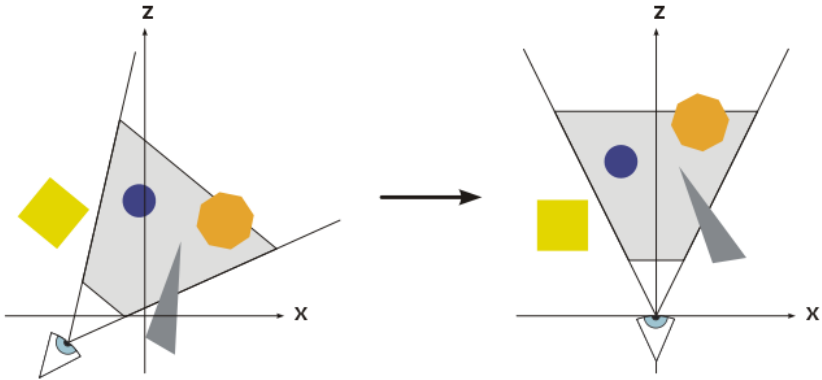
- (1) Wo befindet sich die Kamera, wohin ist die Blickrichtung, wo ist oben?
- (2) Was sind die Objektiv-Eigenschaften: Art der Projektion, Öffnungswinkel, später: Tiefenunschärfe(Blende)

Im Kontext der Computergrafik / Rendering:

- Gesucht ist ein neues Koordinatensystem, das die Welt-Koordinaten in die Blick-Koordinaten wandelt.
- in diesem neuen Koordinatensystem ist die Blickrichtung in neg. z-Richtung, damit die spätere Projektion die Z-Koordinaten auf Null setzt

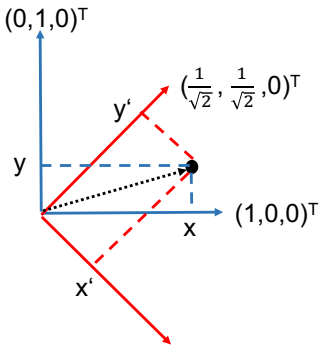
Kameramodell - Welt-Koordinaten in die Blick-Koordinaten wandeln

- (1) Verschiebung
- (2) Rotation



Kameramodell - Welt-Koordinaten in die Blick-Koordinaten wandeln

- Annahme: Verschiebung schon ausgeführt (Mult. mit Translationsmatrix)
- Aufgabe jetzt: Berechne die neuen Koordinaten je x,y,z Komponente bezüglich des neuen Koordinatensystems



$\mathbf{a} \cdot \mathbf{b} = \|a\| \cdot \|b\| \cdot cos(\angle(\mathbf{a}, \mathbf{b})) = \|a\| \cdot \|b\| \cdot \frac{l}{\|a\|} = l$

falls $\mathbf{b}$ *normiert ist, im Beispiel:*

neue y-Koordinate y′:

$$y' = \begin{bmatrix} x \\ y \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

Kameramodell - Welt-Koordinaten in die Blick-Koordinaten wandeln

- Führe dieses Projektionsprinzip jetzt für alle 3 Koordinatenrichtungen durch
- Seien dabei $\mathbf{u}, \mathbf{v}, \mathbf{w}$ die aufspannenden Richtungen des neuen Koordinatensystems mit $\|\mathbf{u}\| = \|\mathbf{v}\| = \|\mathbf{w}\| = 1$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Mit vorheriger Verschiebung:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & t_1 \\ v_x & v_y & v_z & t_2 \\ w_x & w_y & w_z & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Kameramodell - Blick-Koordinaten definieren

- Eine virtuelle Kamera ist üblicherweise definiert durch die Position `Eye`, einen Punkt `Center` auf den sie blickt, und einen Vektor `Up` der definiert wo oben ist.
- Daraus ergibt sich das benötigte neue Koordinatensystem wie folgt:
  - die Blickrichtung ergibt die neue z-Achse, also
    ```
    w = (Center-Eye);
    w.normalize();
    ```
  - der Upvektor ergibt zunächst die neue y-Achse, also :
    ```
    v = Up;
    v.normalize();
    ```
  - Die neue x-Achse ergibt sich als Kreuzprodukt aus **v** und **w**:
    $u = v \times w;$

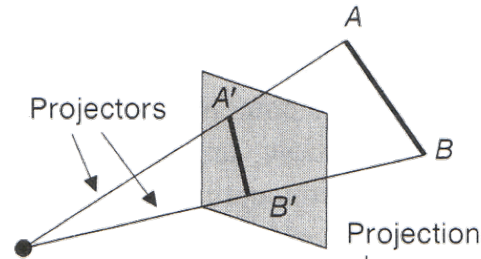
Kameramodell - Blick-Koordinaten definieren

- Da `Up` und (`Center-Eye`) nicht notwendig senkrecht gestanden haben müssen, muss `Up` neu berechnet werden um ein kartesisches Koordinatensystem (eines wo alle Richtungen senkrecht stehen) zu erhalten: `v = w` $\times$ `u;`
- Zuletzt **u** und **v** normieren:
  ```
  u.normalize();
  v.normalize();
  ```
- **Bemerkung:** Die Methode `glm::mat4 lookAt(glm::vec3 eye, glm::vec3 center, glm::vec3 up)` aus der glm-Library realisiert genau die hier beschriebene Matrix für das Kamera-Koordinatensystem.

Projektionen: Grundlagen

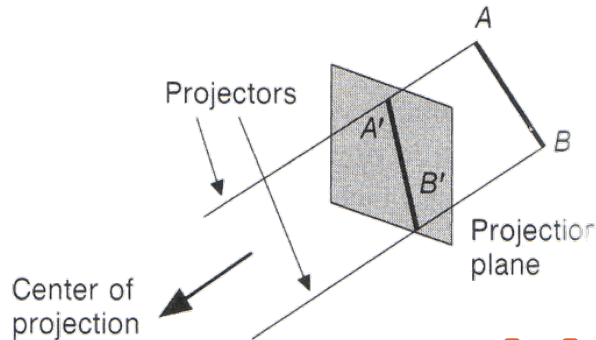
Zentralprojektion

- Bei den perspektivischen Projektionen (Zentralprojektionen) gehen alle Projektionsstrahlen durch das Projektionszentrum, das mit dem Auge des Beobachters zusammenfällt.
- Das Verfahren erzeugt eine optische Tiefenwirkung (in der Malerei der Antike schon genutzt).
- Realistisch in dem Sinne, dass es der Erwartung entspricht, da unser Auge auch eine Zentralprojektion ausführt.



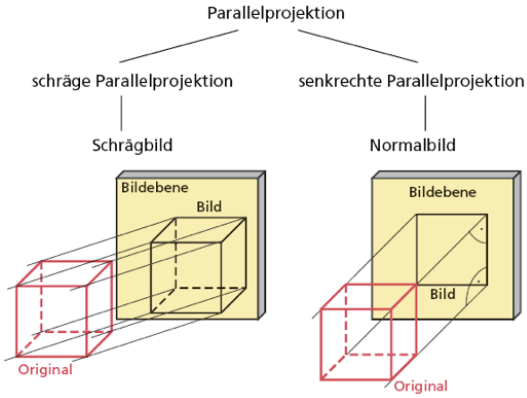
Projektionen: Grundlagen - Parallelprojektion

- Alle Projektionsstrahlen verlaufen parallel in eine Richtung.
- Das Projektionszentrum liegt in einem unendlich fernen Punkt.
- In der projektiven Geometrie stellt die Parallelprojektion somit einen Spezialfall der Zentralprojektion dar.
- Weniger realistisch, erlaubt aber die Bestimmung von Maßen aus dem Bild.



Projektionen: Grundlagen - Parallelprojektion

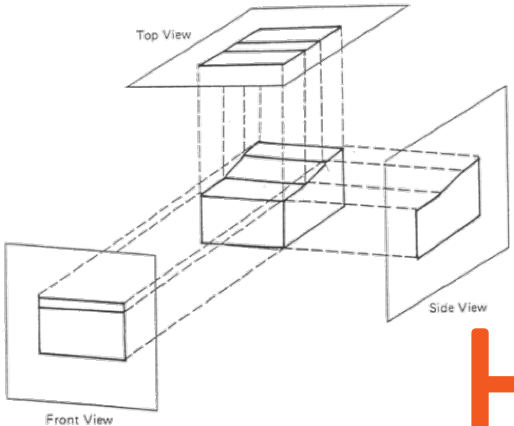
- Die Projektionsstrahlen können bei Parallelprojektionen gegen die Projektionsachse
  - schief oder
  - senkrecht stehen
- **Beispiele schiefwinklig:** Kavalierprojektion, Kabinettprojektion
- **hier:** nur senkrechte Parallelprojektion



Projektionen: Grundlagen - Parallelprojektion - Hauptrisse

- Bei den Hauptrissen
  - Grundriss (top view)
  - Aufriss(front view),
  - Kreuzriss (side view)

  schneidet die Projektionsebene nur eine Hauptachse.
- Die Normale der Projektionsebene ist also parallel zu einer der Hauptachsen.
- Typischer Aufbau bei CAD-Systemen:
  - Drei Hauptrisse und eine perspektivische Ansicht.



Parallelprojektion: Praktische Umsetzung

- Der zu projizierende Bereich ist in jeder Koordinatenrichtung begrenzt.
- Es ergibt sich also ein Würfel-förmiger Sichtbarkeitsbereich (der Raum in dem die zu zeichnenden Objekte liegen)
- Dieser Bereich soll projiziert werden und zusätzlich die x,y - Koordinaten so umskaliert werden, dass diese in $[-1,1]^2$ liegen.
  - **Grund:** aus diesen Koordinaten (Normalized Device Coordinates) werden später die Pixel erzeugt. Seien dazu N_x und N_y die Anzahl Pixel in x- und y-Richtung

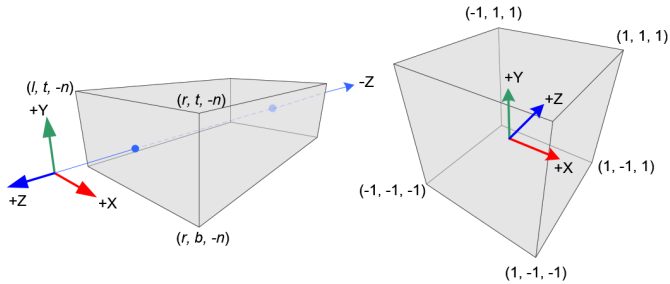
$$s_x \cdot X_{NDC} + \frac{N_x}{2} = X_{\text{pixel}}$$

$$s_y \cdot Y_{NDC} + \frac{N_y}{2} = Y_{\text{pixel}}$$

mit $s_x = \frac{N_x}{2}$ und $s_y = \frac{N_y}{2} \cdot -\frac{N_x}{N_y} = -\frac{N_x}{2} = -s_x$

Parallelprojektion: Praktische Umsetzung

- Behandle die z-Koordinate so, dass diese den Abstand von der Projektionsebene darstellt, bei Parallelprojektion sehr einfach.
  - **Grund:** Daraus wird bestimmt, welches Objekt für eine gegebene x,y-Koordinate gezeichnet wird, nämlich dasjeinge mit dem kleinsten z-Wert $\in [-1, 1]$
  - Der z-Wert wird dazu in den so genannten z-Buffer geschrieben. Ein jeweils kleiner Wert ersetzt einen größeren.



view volume $\Rightarrow$ Normalized Device Coordinates

Parallelprojektion: Praktische Umsetzung

- gültige x liegen zwischen l und r;

$$
\begin{aligned}
l \le x &\le r \\
0 \le x - l &\le r - l \\
0 \le \frac{x-l}{r-l} &\le 1 \\
0 \le 2 \cdot \frac{x-l}{r-l} &\le 2 \\
-1 \le 2 \cdot \frac{x-l}{r-l} - 1 &\le 1 \\
-1 \le 2 \cdot \frac{x-l}{r-l} - \frac{r-l}{r-l} &\le 1 \\
-1 \le \frac{2x-2l-r+l}{r-l} &\le 1 \\
-1 \le \frac{2x-l-r}{r-l} &\le 1 \\
-1 \le \frac{2x}{r-l} - \frac{r+l}{r-l} &\le 1
\end{aligned}
$$

- Formel zur Transformation der x-Koordinate:

$$\frac{2x}{r-l} - \frac{r+l}{r-l}$$

- analog y-Koordinate: ersetze l, r durch t, b
- Schreibe das Ergebnis in eine Matrix:

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Parallelprojektion: Praktische Umsetzung

- Analog für z-Koordinate, aber Blick in neg. z-Richtung daher $-z$ verwenden

$$
\begin{aligned}
n \leq -z \leq f \\
0 \leq -z - n \leq f - n \\
0 \leq \frac{-z-n}{f-n} \leq 1 \\
0 \leq 2 \cdot \frac{-z-n}{f-n} \leq 2 \\
-1 \leq 2 \cdot \frac{-z-n}{f-n} - 1 \leq 1 \\
-1 \leq 2 \cdot \frac{-z-n}{f-n} - \frac{f-n}{f-n} \leq 1 \\
-1 \leq \frac{-2z-2n-f+n}{f-n} \leq 1 \\
-1 \leq \frac{-2z}{f-n} - \frac{f+n}{f-n} \leq 1
\end{aligned}
$$

- Formel zur Transformation der z-Koordinate:

$$\frac{-2z}{f-n} - \frac{f+n}{f-n}$$

- Schreibe das Gesamt-Ergebnis in eine Matrix:

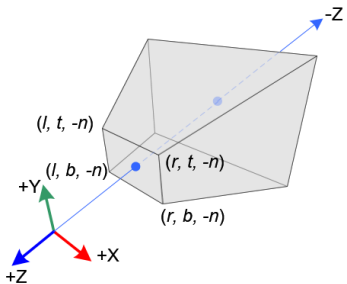
$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2z}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Zentralprojektion: Praktische Umsetzung

Die praktische Umsetzung der perspektivischen Projektion erfolgt je nach Anwendung in unterschiedlichen Konfigurationen, die mittels geeigneter Transformation („lookAt“) des Koordinatensystems erreicht werden können.

Typisches Setup (z.B. bei OpenGL so umgesetzt):

- Projektionszentrum und Augpunkt **E** fallen zusammen und liegen im Nullpunkt
- Blickrichtung ist die negative z-Achse
- Projektion auf die Near-Clipping-Plane, diese ist parallel zur (x, y)-Ebene

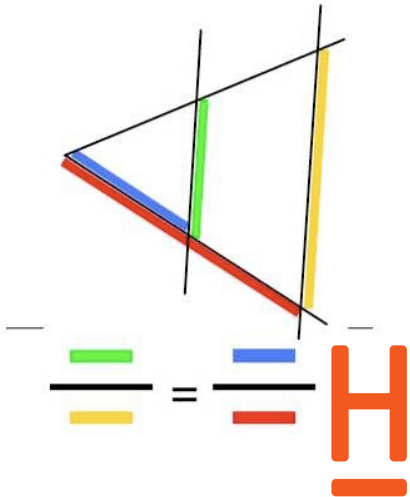


Zentralprojektion: Berechnung

2. Strahlensatz:

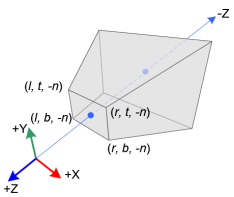
Wenn zwei durch einen Punkt verlaufende Halbgeraden (Strahlen) von zwei parallelen Geraden geschnitten werden, die nicht durch den Scheitel gehen, dann gilt:

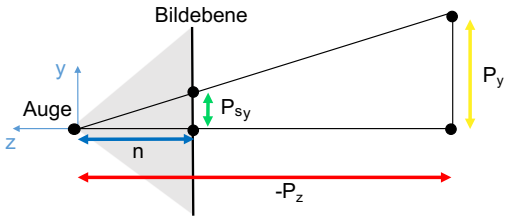
- Es verhalten sich die ausgeschnittenen Strecken auf den Parallelen, wie die ihnen entsprechenden, vom Scheitel aus gemessenen Strecken auf den Strahlen.

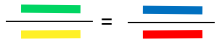


Zentralprojektion: Berechnung

- In OpenGL wird ein Frustum definiert durch die Werte:
  left, **r**ight, **b**ottom, **t**op, **n**ear, **f**ar.
- gemeint ist damit aber ein Pyramidenstump auf der **negativen** z-Achse!







Betrachte die Projektion der y-Koordinate: $P_y \rightarrow P_{s_y}$: $\frac{P_{s_y}}{P_y} = \frac{n}{-P_z} \Rightarrow P_{s_y} = \frac{n \cdot P_y}{-P_z}$

Zentralprojektion: Berechnung

- Betrachte die Projektion der x-Koordinate: $P_x \to P_{s_x}$: $\frac{P_{s_x}}{P_x} = \frac{n}{-P_z} \Rightarrow P_{s_x} = \frac{n \cdot P_x}{-P_z}$
- Schreibe dies jetzt als Matrix-Vektor-Multiplikation:

$$\underbrace{\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{\text{Projektionsmatrix}} \cdot \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} n \cdot P_x \\ n \cdot P_y \\ 0 \\ -P_z \end{bmatrix} \quad \begin{matrix} \Rightarrow \\ \text{normieren auf } w = 1 \end{matrix} \quad \begin{bmatrix} \frac{n \cdot P_x}{-P_z} \\ \frac{n \cdot P_y}{-P_z} \\ 0 \\ 1 \end{bmatrix}$$

- **Fehlt noch:** Normieren auf Normalized Device Coordinates, d.h. $[l, r] \mapsto [-1, 1]$, $[b, t] \mapsto [-1, 1]$ und auch $[n, f] \mapsto [-1, 1]$.
- Also auch hier: für Normalized Device Coordinates darf die z-Koordinate nicht verloren gehen.

Zentralprojektion: Berechnung Normalized Device Coordinates

- gültige Werte für P_{s_x} liegen zwischen l und r;

$$
\begin{aligned}
l &\leq P_{s_x} \leq r \\
0 &\leq P_{s_x} - l \leq r - l \\
0 &\leq \frac{P_{s_x} - l}{r - l} \leq 1 \\
0 &\leq 2 \cdot \frac{P_{s_x} - l}{r - l} \leq 2 \\
-1 &\leq 2 \cdot \frac{P_{s_x} - l}{r - l} - 1 \leq 1 \\
-1 &\leq 2 \cdot \frac{P_{s_x} - l}{r - l} - \frac{r - l}{r - l} \leq 1 \\
-1 &\leq \frac{2P_{s_x} - 2l - r + l}{r - l} \leq 1 \\
-1 &\leq \frac{2P_{s_x} - l - r}{r - l} \leq 1 \\
-1 &\leq \frac{2P_{s_x}}{r - l} - \frac{r + l}{r - l} \leq 1
\end{aligned}
$$

- Formel zur Transformation von P_{s_x} nach $[-1, 1]$:

$$P_{s_{x,NDC}} = \frac{2P_{s_x}}{r - l} - \frac{r + l}{r - l}$$

- setze jetzt noch die Formel zur Berechnung von P_{s_x} aus P_x ein:

$$P_{s_{x,NDC}} = \frac{2nP_x}{-P_z(r - l)} - \frac{r + l}{r - l}$$

Zentralprojektion: Berechnung Normalized Device Coordinates

- analog für y-Koordinate: $P_{s_{y,NDC}} = \frac{2nP_y}{-P_z(t-b)} - \frac{t+b}{t-b}$
- Schreibe dies jetzt wieder als Matrix-Vektor-Multiplikation

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} \cdot P_x + \frac{r+l}{r-l} \cdot P_z \\ \frac{2n}{t-b} \cdot P_y + \frac{t+b}{t-b} \cdot P_z \\ \dots \\ -P_z \end{bmatrix}$$

$$\Rightarrow$$

normieren auf $w = 1$:

$$\begin{bmatrix} \frac{2nP_x}{-P_z(r-l)} - \frac{r+l}{r-l} \\ \frac{2nP_y}{-P_z(t-b)} - \frac{t+b}{t-b} \\ \dots \\ 1 \end{bmatrix}$$

Zentralprojektion: Berechnung Normalized Device Coordinates

- **Beobachtung:** Die Skalierung der Koordinaten hängt nur von den Grenzen (l,r,b,t,n) ab, nicht aber von den anderen Koordinaten
- Damit hat die Matrix für die „z-Zeile“ folgende Form

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- genau wie P_{s_x} und P_{s_y} wird auch die erzeugte z-Koordinate P_{s_z} durch $P_{s_w} = -P_z$ geteilt beim re-normieren auf $w = 1$, also gilt

$$P_{s_{z,NDC}} = \frac{A \cdot P_z + B}{-P_z}$$

Zentralprojektion: Berechnung Normalized Device Coordinates

- Suchen jetzt die Terme für A und B: Wenn P_z auf der near-Clipping Plane liegt, soll dies auf -1 abgebildet werden und für die far-Clipping-Plane entsprechend auf $+1$.
- Ersetze also für diese Fälle in der obigen Gleichung P_z durch n bzw. f.
  - **beachte:** Alle z-Koordinaten von zu projizierenden Punkten in View-Koordinaten sind negativ, aber n und f sind positive Werte, verwende also $-n$ und $-f$:

$$\Rightarrow \frac{-n \cdot A + B}{-(-n)} = -1 \text{ falls } P_z = -n \text{ und } \frac{-f \cdot A + B}{-(-f)} = +1 \text{ falls } P_z = -f$$

also

$$-nA + B = -n \qquad (1)$$
$$-fA + B = f \qquad (2)$$

Zentralprojektion: Berechnung Normalized Device Coordinates

- löse Gleichung (1) nach B auf: $B = -n + An$
- ersetze B in der zweiten Gleichung: $-fA - n + An = f$ und löse nach A auf:

$$-fA + An = f + n \;\Rightarrow\; -(f-n)A = f + n \;\Rightarrow\; A = -\frac{f+n}{f-n}$$

- da A jetzt bekannt ist, ist B leicht auszurechnen: ersetze A in Gleichung (1) um B zu finden:

$$B = -n + An = -n - \frac{f+n}{f-n}\cdot n = -(1 + \frac{f+n}{f-n})\cdot n = -\frac{(f-n+f+n)\cdot n}{(f-n)} = -\frac{2fn}{f-n}$$

- fasse alle Ergebnisse zur Gesamt-Projektionsmatrix zusammen:

Zentralprojektion: Berechnung Normalized Device Coordinates

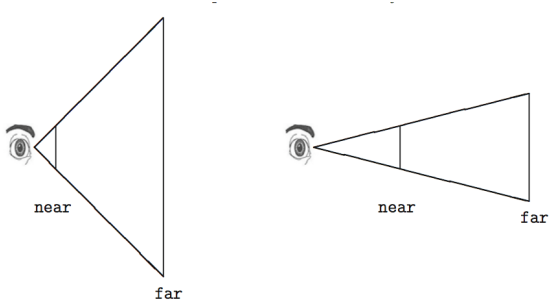
- Projektionsmatrix P in OpenGL zur Erzeugung von Normalized Device Coordinates:

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- **Bemerkung:** Dies ist exakt die Matrix die die Methode `glm::mat4 frustum (T const &left, T const &right, T const &bottom, T const &top, T const &nearVal, T const &farVal)` liefert.

Kamera-Modell: Implementierungsaspekte

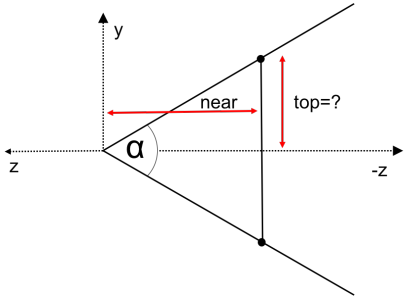
- Änderungen des Parameters `near` für die `frustum`-Methode ändert nicht nur die Sichtbarkeit von Objekten die nah am Sichtpunkt liegen, sondern die ganze Projektion



- Ein kleiner Wert von `near` (bei l, r, b, t unverändert) weitet das sichtbare Volumen stark auf (Weitwinkel-Effekt)
- Ein großer Wert von `near` (bei l, r, b, t unverändert) engt das sichtbare Volumen stark ein (Tele-Effekt)

Kamera-Modell: Implementierungsaspekte

- In der Praxis ist dieser Effekt nicht erwünscht, da man z.B. durch das Verschieben der near-Clipping-Plane in eine technische Konstruktion hineinblicken möchte, indem die vorne liegenden Teile ausgeblendet werden.
- Eine Änderung des Öffnungswinkels der Kamera ist hier nicht gewünscht



- Deswegen: Berechne `left`, `right`, `bottom`, `top` symmetrisch um den Ursprung aus den gegebenen Öffnungswinkeln der Kamera:

$$\tan\frac{\alpha}{2} = \frac{top}{near} \Rightarrow top = near \cdot \tan\frac{\alpha}{2},$$

$$\text{bzw. } bottom = -near \cdot \tan\frac{\alpha}{2}$$

Kamera-Modell: Implementierungsaspekte

- analog

$$\tan\frac{\beta}{2} = \frac{right}{near} \Rightarrow right = near \cdot \tan\frac{\beta}{2}, \text{ und } left = -near \cdot \tan\frac{\beta}{2}$$

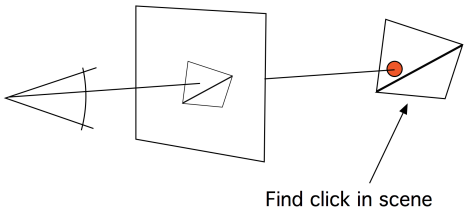
  für den seitlichen Öffnungswinkel der Kamera.

- Implementierung:

```
glm::mat4 perspective(double α, double aspect, double near, double far)
{
  return frustum(−near · tan α·aspect/2 , near · tan α·aspect/2 , −near · tan α/2 , near · tan α/2 , near, far);
}
```

- wobei aspect=$\frac{\beta}{\alpha}$ das Verhältnis zwischen horizontalem und vertikalem Blickwinkel ist.

Objekt-Selektion: „Picking“ - Grundprinzip



- **gegeben:** Pixelposition `(int i, int j)` im Fenster, die z.B. durch einen Mausklick in diskreten Bildkoordinaten erzeugt wurde.
- **gesucht:** derjenige Punkt (bzw. die Liste aller Punkte), der/die auf diesen Pixel projiziert wurde/wurden.
  - **genauer:** derjenige Punkt in **lokalen Koordinaten** desjenigen Objektes, auf dem der Punkt liegt. Also zusätzliche Informationen (welches Objekt, welches Dreieck,...) erwünscht.

„Picking“ - Rekonstruktion 3D-Punkt in Welt-Koordinaten

- (1) berechne aus Pixelposition (`int i, int j`) wieder die Normalized Device Coordinates für x, y, der z-Wert des tatsächlich gezeichneten Punktes kann aus dem z-Buffer gelesen werden.
- (2) Berechne die Inverse der Projektionsmatrix und bilde den Punkt $(x_{NDC}, y_{NDC}, z_{NDC})$ wieder in View-Koordinaten ab.
- (3) Invertiere die Look-at Matrix und berechne die Welt-Koordinaten
- Matrizen sind aufgrund ihrer Konstruktion invertierbar
  - Projektion: Umskalierung und Verschiebung jeder Koordinate, insbesondere auch für z-Koordinate.
  - Look-At: Rotation und Verschiebung
  - **gedanklich:** Operationen in umgekehrter Reihenfolge invertiert ausgeführen.
  - **praktisch:** Nutze `glm::inverse(glm::mat4 arg)`, d.h invertiere direkt.

„Picking“ - Rekonstruktion 3D-Punkt in Welt-Koordinaten

- Rekonstruktion 3D-Punkt in der glm-Library:

```
template<typename T, typename U, qualifier Q>
        GLM_FUNC_QUALIFIER vec<3, T, Q> unProjectZO(vec<3, T, Q> const& w
        mat<4, 4, T, Q> const& model, mat<4, 4, T, Q> const& proj, vec<4,
        {
                mat<4, 4, T, Q> Inverse = inverse(proj * model);

                vec<4, T, Q> tmp = vec<4, T, Q>(win, T(1));
                tmp.x = (tmp.x - T(viewport[0])) / T(viewport[2]);
                tmp.y = (tmp.y - T(viewport[1])) / T(viewport[3]);
                tmp.x = tmp.x * static_cast<T>(2) - static_cast<T>(1);
                tmp.y = tmp.y * static_cast<T>(2) - static_cast<T>(1);

                vec<4, T, Q> obj = Inverse * tmp;
                obj /= obj.w;

                return vec<3, T, Q>(obj);
        }
```

Objekt-Selektion - Picking in Bild-Koordinaten

- **Problem:** lokales Koordinatensystem nicht rekonstruierbar, da keine Info vorliegt, aus welchem lokalen System (d.h. zu welchem Objekt gehörig) der Punkt abgebildet wurde.
- **Idee:** die Objekt-ID (als 24-Bit `Integer`) in die 24-Bit für `rgb` codieren und jedes Objekt in der Farbe zeichen, die zu seiner ID korrespondiert.
- Nutze dazu nicht den Frame-buffer (d.h. derjenige Buffer, der auf dem Bildschirm angezeigt wird), sondern einen zusätzlichen Select-Buffer, in den genauso gezeichnet wird, aber der nicht angezeigt wird.
- Das Auslösen eines Picking-Events (z.B. Mausklick und gedrückte ctrl-Taste) löst ein zusätzliches Zeichnen (eines Teilbereichs der Szene) in den Select-Buffer aus, aus dessen Farbe dann die Objekt-ID rekonstruiert werden kann.

Objekt-Selektion - Picking in Bild-Koordinaten

Nachteile des Picking in Bild-Koordinaten

- über ID lokales Koordinatensystem rekonstruierbar, aber
- Es fehlt weitere strukturelle Information, man kann den 3D Punkt rekonstruieren, aber die Objekt-Zuordnung in die Hierarchie des Szenegraphen ist nicht möglich.
- Ebenso nicht möglich: Teilbereiche von Objekten (Dreiecke oder Gruppen von Dreiecken) selektieren.

Verbesserungsversuch:

- Teile die 24 Bit auf: z.B. 8 Bit für 256 Objekt-Indices und 16 Bit für 65536 Dreiecks-Indices
- Das sind weder besonders viele Objekte noch besonders viele Dreiecke...

Fazit: effizient, da nur Matrix-Inversen und Matrix-Vektor Multiplikation gerechnet werden muss, aber nicht genug Objekt-Informationen rekonstruierbar.

Objekt-Selektion - Picking in lokalen Objekt-Koordinaten

Idee: 3D-Schnittpunkt-Berechnung von Picking-Strahl und Objekten.

- Objekte bleiben in ihrem lokalen Koordinatensystem, Picking-Strahl wird dort hin transferiert.
- Picking Strahl wird wie folgt erzeugt:
  - Berechne aus den Pixel-Koordinaten die NDC für die x- und y- Koordinate
  - Invertiere die Projektionsmatrix P und berechne $Q = P^{-1} \cdot [x_{NDC}, y_{NDC}, -n, 1]^T$. (Normierung der homogenen Koordinate nicht vergessen!!)
  - Q ist derjenige Punkt in View-Koordinaten auf der near-Plane, der auf (x_{NDC}, y_{NDC}) projiziert würde.
  - In der Richtung des Strahls $eypoint + s \cdot (Q - eyepoint) = s \cdot Q$ liegen alle Punkte die auf (x_{NDC}, y_{NDC}) projiziert werden.
- Transferiere diesen Strahl zunächst in Welt-Koordinaten und dann in jedes lokale Koordinatensystem und überprüfe mögliche Schnittpunkte mit Objekten.

Objekt-Selektion - Picking in lokalen Objekt-Koordinaten

Analyse:

- Schnittpunkt-Berechnung muss CPU-seitig erfolgen und vom Szene-Graphen aus gesteuert werden.
- Naive Iteration über alle Dreiecke erfordert ggf. extrem viele Schnittpunkt-Tests.
- Sehr viele Schnittpunkt-Tests sind negativ, ein Strahl trifft nur sehr wenige Dreiecke.
- Für jeden Schnittpunkt ist aber das aktuell bearbeitete Objekt und Dreieck bekannt, d.h. Struktur-Information bei einem „Treffer“ vorhanden.

Noch offen:

- Wie wird ein Strahl mathematisch repräsentiert?
- Wie wird der Schnitt von einem Strahl mit einem Dreieck effizient berechnet?
- Lassen sich die negativen Schnittpunkt-Tests vermeiden/vermindern?

Definitionen: Gerade, Strahl, Liniensegment

Definition 5.1 (Gerade, Liniensegment)

Eine **Gerade** *L ist definiert als die Menge aller Punkte, die als Linearkombination zweier beliebiger aber verschiedener Punkte A und B darstellbar sind:*

$$L(t) = (1 - t)A + t\ B \text{ mit } -\infty < t < \infty.$$

Für ein **Liniensegment** *oder kurz* **Segment** *ist t limitiert auf* $0 \leq t \leq 1$*, gemeint ist also die Strecke zwischen A und B. Ein Segment ist* gerichtet*, wenn A und B in einer vorgegebenen Ordnung zueinander sind.*

Definition 5.2 (Strahl)

Ein **Strahl** *R ist eine halb-unendliche Gerade, indem t limitiert wird auf* $t \geq 0$*, üblicherweise geschrieben als*

$$R(t) = A + t\ \mathbf{v} \text{ mit } \mathbf{v} = B - A$$

Ebenen-Darstellungen

Eine **Ebene** im Raum ist bildlich vorstellbar als eine flache Oberfläche die sich unendlich in alle Richtungen ausdehnt.

- Eine Ebene kann auf viele verschiedene Arten beschrieben werden, z.B. durch:
  - drei Punkte$A, B, C,$, die nicht auf einer Linie liegen (also ein Dreieck in der Ebene bilden)
  - Einen Punkt auf der Ebene und zwei unabhängige Richtungen in der Ebene
  - Eine Normale $\mathbf{n}$ und einen Punkt A auf der Ebene
  - Eine Normale $\mathbf{n}$ und eine Distanz d vom Ursprung
- Bei der Definition durch 3 Punkte A, B, C, die gegen den Uhrzeigersinn orientiert sind lässt sich einerseits die parametrische Darstellung berechnen als

$$P(u, v) = A + u(B - A) + v(C - A).$$

als auch die Normale $\mathbf{n}$ berechnen durch: $\mathbf{n} = (B - A) \times (C - A)$.

Ebenen-Darstellungen

- Punkte auf der gleichen Seite wie die aus der die Normale heraus zeigt sind **vor** der Ebene, Punkte auf der anderen Seite **hinter** der Ebene.
- Gegeben eine Normale $\mathbf{n}$ und einen Punkt A auf der Ebene, dann sind alle Punkte X der Ebene charakterisiert durch $X - A$ steht senkreecht auf $\mathbf{n}$, also

$$\mathbf{n} \cdot (X - A) = 0 \quad \Rightarrow \quad \mathbf{n} \cdot X = \mathbf{n} \cdot A \quad \Rightarrow \quad \mathbf{n} \cdot X = d \text{ mit } d = \mathbf{n} \cdot A$$

- Wenn $\|\mathbf{n}\| = 1$, dann ist $|d|$ der Abstand der Ebene zum Ursprung, ohne Betrag wird d als gerichteter Abstand interpretiert.
- Wenn $\|\mathbf{n}\| \neq 1$, dann ist $|d|$ immer noch der der Abstand, aber in Einheit der Länge des Vektors $\mathbf{n}$.
- $\|\mathbf{n}\| = 1$ wünschenswert, da sich dadurch Berechnungen vereinfachen.

Ebenen-Darstellungen

- Stelle die Normale **n** immer normiert (d.h. in Länge 1) dar, dann gilt:

```
struct Plane {
    Vector n; // Plane normal. Points x on the plane satisfy Dot(n,x) = d
    float d; // d = dot(n,p) for a given point p on the plane
};

// Given three noncollinear points (ordered ccw), compute plane equation
Plane ComputePlane(Point a, Point b, Point c)
{
    Plane p;
    p.n = Normalize(Cross(b - a, c - a));
    p.d = Dot(p.n, a);
    return p;
}
```

Schnitt Strahl vs. Ebene

- Sei eine Ebene P gegeben durch $\mathbf{n} \cdot X = d$ und ein Strahl R(t)=A+t(B-A) für $0 \leq t < \infty$.
- Der t-Wert für den Schnitt des Strahls mit der Ebene wird durch einsetzen der Strahl-Gleichung in die Ebene und Auflösen nach t berechnet:

$$(\mathbf{n} \cdot (A + t(B - A)) = d \qquad \text{(ersetze } X \text{ durch } R(t) = A + t(B - A))$$

$$\mathbf{n} \cdot A + t\mathbf{n} \cdot (B - A) = d \qquad \text{(multipliziere Skalarprodukt aus)}$$

$$t\mathbf{n} \cdot (B - A) = d - \mathbf{n} \cdot A \qquad \text{(bringe alle skalaren Werte auf die rechte Seite)}$$

$$t = \frac{d - \mathbf{n} \cdot A}{\mathbf{n} \cdot (B - A)} \qquad \text{(teile durch } \mathbf{n} \cdot (B - A))$$

- $\Rightarrow$ Schnittpunkt $Q = A + [(d - \mathbf{n} \cdot A)/(\mathbf{n} \cdot (B - A))](B - A)$

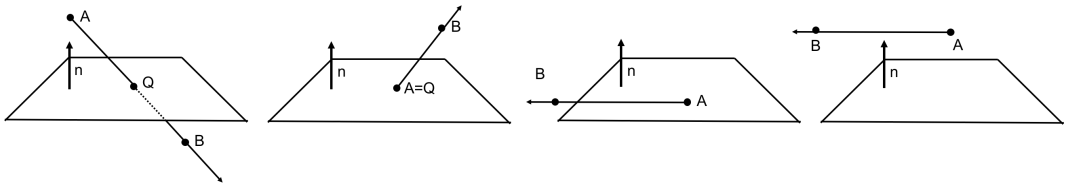
Schnitt Strahl vs. Ebene

```
bool IntersectRayPlane(Point a, Point b, Plane p, float &t, Point &q)
  {
       // Compute the t value for the Ray ab intersecting the plane
       Vector ab = b - a;
       t = (p.d - Dot(p.n, a)) / Dot(p.n, ab);

       // If t in [0..∞) compute and return intersection point
       if (t >= 0.0f && t < INF) {
           q = a + t * ab;
           return 1;
        }
       // Else no intersection
       return 0;
 }
```

Schnitt Strahl vs. Ebene - Numerische Robustheit

- folgende geometrische Konfigurationen sind möglich (von links nach rechts)



- ein eindeutiger Schnittpunkt Q.
- Schnittpunkt $Q = A$, Strahl zeigt aber von der Ebene weg (kein „Durchstoßen“)
- Strahl liegt in der Ebene
- Strahl liegt parallel zur Ebene aber nicht in der Ebene
- kein Schnitt: $A \notin P$ und R zeigt von Ebene weg ($t < 0$), ohne Bild.

Numerische Robustheit - Infinity Arithmetik IEEE 754

- **Beachte:** der Code prüft nicht explizit auf Division durch Null, trotzdem arbeitet er auch in diesem Fall korrekt!

```
t = (p.d - Dot(p.n, a)) / Dot(p.n, ab);

// If t in [0..∞) compute and return intersection point
if (t >= 0.0f && t < infinity)
    ⋮
// Else no intersection
return 0;
```

- Kritisch ist die Berechnung von t wenn der Nenner Null wird oder Zähler und Nenner Null sind
  - Nenner: $Dot(p.n, ab) = 0 \Rightarrow$ Strahl liegt in der Ebene oder ist parallel.
  - Zähler: $Dot(p.n, a) = d \Rightarrow$ A liegt in der Ebene.

Numerische Robustheit - Infinity Arithmetik IEEE 754

- Floating Point Zahlen nach IEEE 754 Standard können $\pm\infty$ und *NaN* (Not a Number) darstellen.
- z.B. $1.0/0.0 = INF$, $log(0) = -INF$, $0.0/0.0 = NaN$
- Die grundlegenden Operationen und mathematischen Funktionen akzeptieren ±INF und *NaN* als Eingabe.
  - **Berechnungen:** z.B. $2 + INF = INF$, $4.0/INF = 0$, $INF - INF = NaN$, $0 \cdot INF = NaN$.
  - sobald *NaN* als Eingabe bei Berechnungen verwendet wird, ist das Ergebnis immer *NaN*.
  - **Vergleiche:** erlaubt, + INF ist größer als alle anderen Werte außer INF selbst und NaN (analog -INF „kleiner als“)
  - **NaN ist ungeordnet:** Vergleiche mit *NaN* gehen immer negativ aus, d.h $4 < Nan$ liefert false, aber auch $NaN == NaN$ und $NaN < INF$ liefern false.

Numerische Robustheit - Infinity Arithmetik IEEE 754

- **Für die Berechnung von t:**
  - Strahl parallel zur Ebene: t= "'positive Zahl"' / 0 = INF, Vergleich $t < INF$ liefert false, also kein Schnittpunkt
  - Strahl liegt in der Ebene t = 0.0/0.0 = NaN, Vergleich $NaN < INF$ liefert false, also kein Schnittpunkt
- **Implementierung nach ISO C99 oder C++11 Standard**
  - Nutze die Macros (ISO C99) oder Funktionen (C++11):

    `isfinite(x), isnan(x), isinf(x)`
  - Geht auch in C++98 mit `#include <cmath>` und `std::isfinite(x)`, etc.
  - If-Abfrage im Code müsste also lauten: **`if ( isfinite(t) && t >= 0.0f )`**, in dieser Reihenfolge, da bei `isfinite(t)` gleich false der zweite Ausdruck gar nicht mehr ausgewertet wird

Schnitt Strahl vs. Dreieck

- Bisher nur Schnitt mit Ebenen möglich, Erweiterung für Dreiecke benötigt
- **Eine von vielen Möglichkeiten:** Berechne Schnitt mit Dreiecksebene, danach Test ob der Schnittpunkt im Dreieck liegt
- Nutze so genannte **Baryzentrische Koordinaten**
- **Idee:** So wie man einen Punkt auf der Strecke von A nach B darstellen kann durch $P(t) = (1-t)A + tB$ mit $0 \leq t \leq 1$, bzw.

  $P(t) = sA + tB$ mit $0 \leq s, t \leq 1$ und $s + t = 1$.

  so kann man einen Punkt im Inneren eines Dreiecks darstellen durch

  $P(u, v, w) = uA + vB + wC$ mit $0 \leq u, v, w \leq 1$ und $u + v + w = 1$.
- **Strategie:** Für eine gegebene Ebene und Schnittpunkt P mit dem zu testenden Strahl: berechne u, v, w und prüfe ob $0 \leq u, v, w \leq 1$

Schnitt Strahl vs. Dreieck - Baryzentrische Koordinaten

- Die Bedingung $u + v + w = 1$ bedeutet, dass die Parameter nicht unabhängig voneinander sind
- Betrachte die äquivalente Ebenen-Darstellung $P = A + v(B - A) + w(C - A)$, wobei v, w beliebig sind, dann gilt:

$$P = A + v(B - A) + w(C - A) = (1 - v - w)A + vB + wC.$$

bzw.

$$v(B - A) + w(C - A) = P - A$$

mit $\mathbf{v_0} = B - A$, $\mathbf{v_1} = C - A$ und $\mathbf{v_2} = P - A$ kann man dies schreiben als:

$$v\mathbf{v_0} + w\mathbf{v_1} = \mathbf{v_2}.$$

Schnitt Strahl vs. Dreieck - Baryzentrische Koordinaten

- um ein lineares Gleichungssystem zur Berechnung von u, v multipliziert man die Gleichung mit $\mathbf{v_0}$ bzw. $\mathbf{v_1}$:

$$(v\mathbf{v_0} + w\mathbf{v_1}) \cdot \mathbf{v_0} = \mathbf{v_2} \cdot \mathbf{v_0} \text{ und}$$

$$(v\mathbf{v_0} + w\mathbf{v_1}) \cdot \mathbf{v_1} = \mathbf{v_2} \cdot \mathbf{v_1}$$

- da das Skalarprodukt linear ist folgt:

$$v(\mathbf{v_0} \cdot \mathbf{v_0}) + w(\mathbf{v_1} \cdot \mathbf{v_0}) = \mathbf{v_2} \cdot \mathbf{v_0} \text{ und}$$

$$v(\mathbf{v_0} \cdot \mathbf{v_1}) + w(\mathbf{v_1} \cdot \mathbf{v_1}) = \mathbf{v_2} \cdot \mathbf{v_1}$$

- Löse jetzt mit Cramer-Regel.

Schnitt Strahl vs. Dreieck - Baryzentrische Koordinaten

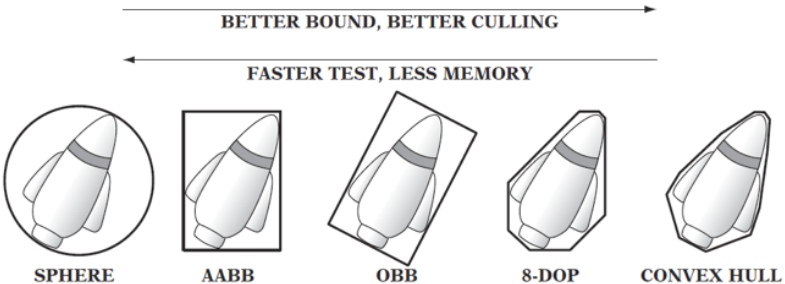
```
// Compute barycentric coordinates (u, v, w) for
// point p with respect to triangle (a, b, c)
void Barycentric(Point a, Point b, Point c, Point p, float &u, float &v, float &w)
{
    Vector v0 = b - a, v1 = c - a, v2 = p - a;
    float d00 = Dot(v0, v0);
    float d01 = Dot(v0, v1);
    float d11 = Dot(v1, v1);
    float d20 = Dot(v2, v0);
    float d21 = Dot(v2, v1);
    float denom = d00 * d11 - d01 * d01;
    v = (d11 * d20 - d01 * d21) / denom;
    w = (d00 * d21 - d01 * d20) / denom;
    u = 1.0f - v - w;
}
```

Bounding Volumes - Prinzip und Varianten

- Ein **Bounding Volume** ist ein einzelnes einfaches Volumen, das ein oder mehrere komplexe Objekte umschließt
- **Idee:** Bei billigem Schnitt-Test für ein Bounding Volume kann die Aussage für „schneidet nicht“ sehr schnell/effizient getroffen werden.
- BVs besonders wichtig bei komplexeren Kollisionstests, z.B. zwei polygonalen Objekten miteinander.
  - In jedem Zeitschritt bewegt sich ein Punkt um eine bestimmte Strecke, teste diese Strecke muss mit jedem Dreieck des anderen Objekte auf Kollision
  - Aufwand also in $O(n^2)$, wobei n gleichermaßen die Anzahl der Punkt wie der Dreiecke beschreibt
  - Aufwand für zwei Kugeln als Bounding Volume: kein Schnitt falls „Differenz der Mittelpunkte >= Summe der Radien“, unabhängig von der Komplexität der eingeschlossenen Objekte.

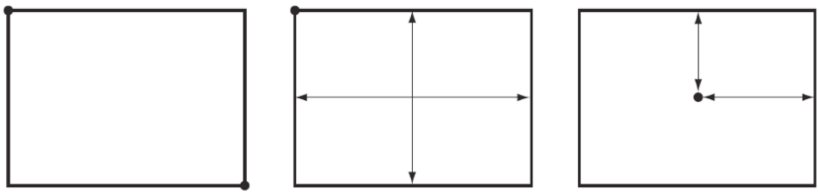
Bounding Volumes - Grundlagen

- Billige Schnitt-Tests, Volume selbst billig zu berechnen
- Enge Passform
- Einfach zu rotieren und transformieren
- Wenig Speicher-Verbrauch



Bounding Volumes - Axis Aligned Bounding Box (AABB)

- In 3D ein sechsseitiger rechteckiger Würfel, so dass die Seiten entlang der Koordinaten-Achsen ausgerichtet sind.
- **Bestes Feature:** ein extrem schneller Überlappungstest.
- **Implementierungsvarianten:**



- Wähle `min-max`: größter Speicherverbrauch, aber schnellste Kollisionstests

Bounding Volumes - Axis Aligned Bounding Box - Berechnung

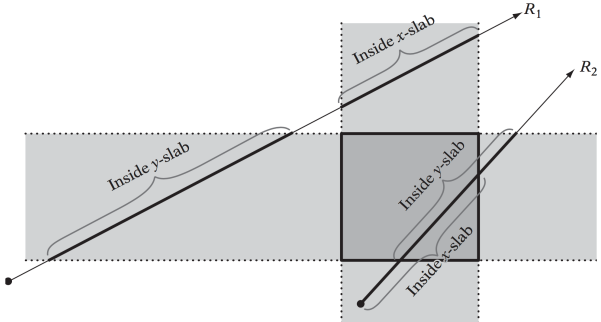
```
// region R = {(x, y, z) | min.x<=x<=max.x, min.y<=y<=max.y, min.z<=z<=max.z }
struct AABB {
   Point min;
   Point max;
};
```

Der Code im Kommentar erklärt die Berechnung für ein Objekt:

- Iteriere über alle Punkte, sammle koordinatenweise das Maximum und Minimum, konstruiere daraus die Punkte `min` und `max`.
- Dies tut man sinnvollerweise in lokalen Koordinaten, denn hier sind die geometrischen Positionen der Punkte verfügbar und bei einer Transformation des Objektes müsste die AABB neu berechnet werden

Bounding Volumes - Schnitt Strahl vs. AABB

- Idee: Damit ein Strahl eine AABB schneidet, muss er zu einem Zeitpunkt t gleichzeitig im x-, y-,z- Koordinatenbereich sein:



- die inside-slab-Bereiche müssen sich also überlappen

Bounding Volumes - Schnitt Strahl vs. AABB

- Ein-und Austrittspunkt für einen `slab` wird durch Ebenenschnitte berechnet.
- Diese Schnitte sind aber vereinfacht, da die Normale jeweils in eine Koordinatenrichtung zeigt, d.h. die Normale hat 2 Komponenten, die Null sind.
- Setze die Strahl-Gleichung $R(t) = P + t\mathbf{d}$ in die Ebenengleichungen $X \cdot \mathbf{n}_i = d_i$ der slab-Begrenzungen ein und löse nach t auf:

$$t = (d_i - P \cdot \mathbf{n}_i)/(\mathbf{d} \cdot \mathbf{n}_i))$$

- Für eine AABB sind einige Komponenten von $\mathbf{n}$ gleich Null, d.h für $P = (p_x, p_y, p_z)$ und $\mathbf{d} = (d_x, d_y, d_z)$ vereinfacht sich der Ausdruck (z.B. für x-Richtung) zu

$$t = (d_i - p_x)/d_x$$

setze dann nacheinander min_x und max_x für d_i ein.

Bounding Volumes - Schnitt Strahl vs. AABB

```
// Intersect ray R(t) = p + t*d against AABB a. When intersecting,
// return intersection distance tmin and point q of intersection
int IntersectRayAABB(Point p, Vector d, AABB a, float &tmin, Point &q)
{
    tmin = 0.0f;             // set to -FLT_MAX to get first hit on line
    float tmax = FLT_MAX;    // set to max distance ray can travel (for segment)
    // For all three slabs
    for (int i = 0; i < 3; i++) {
        if (Abs(d[i]) < EPSILON) {
            // Ray is parallel to slab. No hit if origin not within slab
            if (p[i] < a.min[i] || p[i] > a.max[i]) return 0;
        } else {
            // Compute intersection t value of ray with near and far plane of slab
            float ood = 1.0f / d[i];
            float t1 = (a.min[i] - p[i]) * ood;
            float t2 = (a.max[i] - p[i]) * ood;
            // Make t1 be intersection with near plane, t2 with far plane
            if (t1 > t2) Swap(t1, t2);
            // Compute the intersection of slab intersection intervals
            if (t1 > tmin) tmin = t1;
            if (t2 > tmax) tmax = t2;
            // Exit with no collision as soon as slab intersection becomes empty
            if (tmin > tmax) return 0;
        }
    }
    // Ray intersects all 3 slabs. Return point (q) and intersection t value (tmin)
    q = p + d * tmin;
    return 1;
}
```

Bemerkung: Um eine Division durch Null zu vermeiden, wenn der Stahl parallel zum slab ist, wenn also wenn die entsprechende Richtungskomponente in der Strahlrichtung gleich Null ist, dann wird explizit abgefragt ob der Startpunkt des Strahls innerhalb des slab liegt.