Computer Graphik

Mitschrift von www.kuertz.name

Hinweis: Dies ist **kein offizielles Script**, sondern nur eine private Mitschrift. Die Mitschriften sind teweilse **unvollständig, falsch oder inaktuell**, da sie aus dem Zeitraum 2001–2005 stammen. Falls jemand einen Fehler entdeckt, so freue ich mich dennoch über einen kurzen Hinweis per E-Mail – vielen Dank!

Mihhail Aizatulin (avatar@hot.ee)

Inhaltsverzeichnis

Organisatorisches

- Mailing Liste: cg-uebung@mip.informatik.uni-kiel.de
- Abgabe Programmieraufgaben: cg-loesung@mip.informatik.uni-kiel.de
- Abgabe Theorie: Do, 16 Uhr nach der Vorlesung

1 Allgemeines zu OpenGL

1.1 Pipeline

Der Aufbau der Pipeline vom Anfang bis zur Monitorausgabe sieht wie folgt aus:[1]

- Eingabe: Vertices (world coordinates).
- **Transformer**: Multiplikation mit `ModelView` Matrix, kameraabhängig.[2]
- **Projector**: Multiplikation mit `Projection` Matrix. Ergebnis: clip coordinates.
- **Perspective Division** Koordinaten werden durch w geteilt. Ergebnis: normalized device coordinates (NDL).
- **Clipper**: es werden Bereiche entfernt, die nicht sichtbar sind (d.h. außerhalb von Norm Frustum liegen).
- **Viewport Transformation** Ergebnis: zweidimensionale window/viewport coordinates.
- **Polygonizer and Rasterizer.** Ergebnis: Fragments.
- **Fragmentoperationen:** Berechnung von Licht. Es wird Farbe für jedes Fragment bestimmt.
- **Buffertests.** Aus einem Fragment wird ein Pixel mit Farbe. Es wird entschieden, welche Pixel sichtbar sind.
- **Framebuffer.** Ergebnis: Monitorausgabe.

Seit 20 Jahren wird Pipeline auf der Hardware-Ebene implementiert.

[1] Die Richtigkeit ist noch fraglich.
[2] Man kann die `ModelView` Matrix z.B. mit Hilfe der Funktion `gluLookAt` einstellen.

1.2 Libs und Headers

- Für SuSE 9.1:
  - Xfree86-Mesa-(devel)-4.3
  - freeglut-(devel)-2.2
- Debian SID:
  - xlibmesa-gl-(devel)
  - xlibmesa-glu-(devel)
  - libglut3-(devel) / freeglut-(devel)
- Allgemein:
  - GL Libs & Headers
  - GLU Libs & Headers
  - GLUT Libs & Header
  - GLX / WGL / ... & Driver

2 Transformationen des View Frustum

Die Projektion in OpenGL erfolgt in folgenden Schritten:

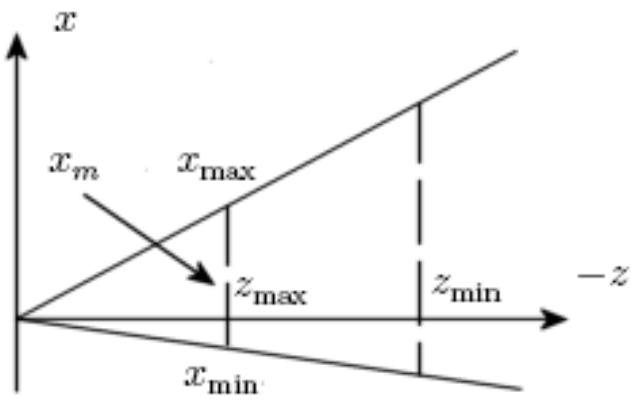
1. Multiplikation mit `ProjectionMatrix`, perspektive Division: Transformation des View Frustum zu einem Norm Frustum (siehe unten).

2. Viewport-Transformation: Norm-Koordinaten werden in tatsächliche Bildschirm-Koordinaten umgewandelt.

2.1 Projection Matrix

Gegeben seien:

- Allgemeine Kamera im Ursprung mit optischer Achse $-z$.
- Beliebiger Öffnungswinkel, auch nicht notwendig symmetrisch.
- Near clipping plane und far clipping plane: $z_{\min}$ (oder z_{far}) und $z_{\max}$ (oder z_{near}) mit $0 > z_{\max} > z_{\min}$.



Die Koordinate der Schnittpunkte der $z_{\min}$-Ebene mit den Seiten der Pyramide bezeichnen wir $x_{\max}$ und $x_{\min}$ bzw. mit $y_{\max}$ und $y_{\min}$. Wir wollen das schiefe Pyramidenstumpf überführen zu einem *Norm Frustum*: einem Quader mit Kantenlänge 2. Das macht die `ProjectionMatrix` in OpenGL. Die Transformation geschieht in 2 Schritten:

1. **Affine Scherung und Skalierung** des schiefen Pyramidenstumpfes auf einen symmetrischen Pyramidenstumpf mit 90° Öffnung.

(a) **Scherung in xy-Richtung.** Seien

$$x_m = \frac{x_{\min} + x_{\max}}{2}; \quad y_m = \frac{y_{\min} + y_{\max}}{2}$$

Die Steigungen für die x- und y-Richtung der Scherung sind

$$\gamma = \frac{x_m}{z_{max}} = \frac{x_{\min} + x_{\max}}{2z_{\max}}; \quad \delta = \frac{y_m}{z_{\max}} = \frac{y_{\min} + y_{\max}}{2z_{\max}}$$

Die Scherung sieht wie folgt aus:

$$x' = x + \gamma z; \quad y' = y + \delta z$$

Die Matrix für die Scherung ist somit:

$$T_{\text{shear}} = \begin{pmatrix} 1 & 0 & \gamma & 0 \\ 0 & 1 & \delta & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(b) **Skalierung** der Kanten mit Steigung ± 1. Für den x-Skalierungsfaktor a und den y-Skalierungsfaktor b gilt:

$$\begin{aligned} a &= \frac{z_{\max}}{\frac{1}{2}(x_{\max} - x_{\min})} = \frac{2z_{\max}}{x_{\max} - x_{\min}} \\ b &= \frac{z_{\max}}{\frac{1}{2}(y_{\max} - y_{\min})} = \frac{2z_{\max}}{y_{\max} - y_{\min}} \end{aligned}$$

Somit ergibt sich die Matrix

$$T_{\text{scale}} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Insgesamt ergibt sich die Matrix $T_{\text{affin}} = T_{\text{scale}} \cdot T_{\text{shear}}$

2. **Perspektivische Transformation** auf den Würfel mit Kantenlänge 2. Für $\alpha, \beta \in \mathbb{R}$ betrachten wir die Matrix

$$T_N = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Diese Matrix bewirkt eine projektive Abbildung mit Skalierung und Verschiebung in z-Richtung. Durch T_N wird abgebildet

$$M = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \mapsto M' = \begin{pmatrix} x \\ y \\ \alpha \cdot z + \beta \\ -z \end{pmatrix}$$

Nach dem Projizieren von M' (Division durch w) erhalten wir

$$M_p = \begin{pmatrix} -x/z \\ -y/z \\ -\left(\alpha + \frac{\beta}{z}\right) \\ 1 \end{pmatrix}$$

T_N überführt somit die Pyramide mit Kanten $x = \pm z, y = \pm z$ in einen Quader mit $x' = \pm 1$ und $y' = \pm 1$. Es soll zusätzlich gelten

$$z'_{\max} = -\left(\alpha + \frac{\beta}{z_{\max}}\right) = 1; \quad z'_{\min} = -\left(\alpha + \frac{\beta}{z_{\min}}\right) = -1$$

Auflösen nach α und β ergibt

$$\alpha = \frac{z_{\max} + z_{\min}}{z_{\max} - z_{\min}}; \quad \beta = \frac{2 z_{\max} z_{\min}}{z_{\max} - z_{\min}}$$

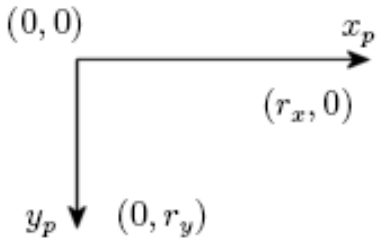
Die gesamte Transformation ist somit[3]

$$\begin{aligned} P = T_N \cdot T_{\text{scale}} \cdot T_{\text{shear}} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \gamma & 0 \\ 0 & 1 & \delta & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \frac{y_{\min}}{x_{\max} - x_{\min}} & 0 & \frac{x_{\max} + x_{\min}}{x_{\max} - x_{\min}} & 0 \\ 0 & \frac{z_{\min}}{y_{\max} - y_{\min}} & \frac{y_{\max} + y_{\min}}{y_{\max} - y_{\min}} & 0 \\ 0 & 0 & -\frac{z_{\max} + z_{\min}}{z_{\max} - z_{\min}} & \frac{z_{\max} z_{\min}}{z_{\max} - z_{\min}} \\ 0 & 0 & -1 & 0 \end{pmatrix} \end{aligned}$$

2.2 Viewport-Transformation

Gegeben seien Norm-Koordinaten des Punktes $-1 \leq x_n, y_n \leq 1$. Man möchte auf das Koordinatensystem x_p, y_p des Bildschirms umrechnen. Sei die Breite des Bildschirms r_x und die Höhe r_y Pixel. Die Koordinatenachsen liegen wie folgt:

[3] Es besteht noch Zweifel an der Richtigkeit der Matrix - evtl. wird sie noch korrigiert. Siehe auch `glFrustum`-Matrix in OpenGL-Reference.



Die Transformation besteht aus folgenden Schritten:

1. Spiegelung der y-Achse

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

2. Skalierung auf Bildschirmgröße

$$T_{SK} = \begin{pmatrix} r_x/2 & 0 & 0 \\ 0 & r_y/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3. Verschiebung des Hauptpunkten

$$T_V = \begin{pmatrix} 1 & 0 & r_x/2 \\ 0 & 1 & r_y/2 \\ 0 & 0 & 1 \end{pmatrix}$$

Die gesamte Transformation ist

$$T_{\text{GES}} = T_V T_{SK} T_{SP} = \begin{pmatrix} r_x/2 & 0 & r_x/2 \\ 0 & -r_y/2 & r_y/2 \\ 0 & 0 & 1 \end{pmatrix}$$

Diese Transformation ist geräteabhängig. Mit unserem Norm-Koordinatensystem sind wir unabhängig vom Hardware.

3 Clipping

3.1 Darstellungsformen von Geradensegmenten

Seien zwei Punkte $p_1 = (x_1, y_1)^T, p_2 = (x_2, y_2)^T$ mit $p_1 \neq p_2$ gegeben. Wir können die Gerade durch p_1 und p_2 in folgenden Formen darstellen:

- *Steigungsform*

  $$y = ax + b; \quad \text{wobei} \quad a = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

  Diese Form ist problematisch, falls die Gerade parallel zur y-Achse liegt, deswegen wird sie nicht benutzt.

- *parametrische Form*

  $$\vec{p}(t) = \vec{p_0} + t(\vec{p_1} - \vec{p_0})$$

  Ein Segment beschreibt man, indem man $0 \leq t \leq 1$ setzt.

- *Normalenform*

  $$\vec{n} \cdot \vec{p} - \vec{n} \cdot \vec{p}_0 = 0$$

  wobei $\vec{n}$ die Normale zur Gerade ist, $\vec{n} = (-\Delta y, \Delta x)$. Anders kann man schreiben

  $$\vec{n} \cdot \vec{p} = d$$

  wo d die Länge des Lotes aus dem Ursprung auf die Gerade ist.

- *Implizite Geradenform*

  $$g = \vec{n} \cdot \vec{p} - d \quad \begin{cases} > 0 & \text{für } \vec{p} \text{ „links“ von Geraden} \\ = 0 & \text{für } \vec{p} \text{ auf der Geraden} \\ < 0 & \text{für } \vec{p} \text{ „rechts“ von Geraden} \end{cases}$$

3.2 Schnittpunktberechnung bei Cohen-Sutherland

Sei ein Geradensegment gegeben in der Form

$$\begin{aligned} x &= x_0 + t(x_1 - x_0) \\ y &= y_0 + t(y_1 - y_0) \end{aligned}$$

Die Kanten des Viewports schreiben wir als

$$\begin{aligned} K_1: &\quad x_1 = t_1 \text{ und } y_1 = 0 \\ K_2: &\quad x_2 = 0 \text{ und } y_2 = t_2 \\ K_3: &\quad x_3 = t_3 \text{ und } y_3 = a \\ K_4: &\quad x_3 = b \text{ und } y_3 = t_4 \end{aligned}$$

Wir lösen 4 Gleichungssysteme um die Schnittpunkte des Segments mit den Kanten des Viewports zu bestimmen.

3.3 Parametrisches Clipping

Sie die Gerade in parametrischer Form $\vec{p}(t) = \vec{p}_0 + t(\vec{p}_1 - \vec{p}_0)$ gegeben. Wir beschreiben die Kanten in Normalenform durch $\vec{n}_i \cdot \vec{p} - \vec{n}_i \cdot p_i = 0$. Durch einsetzen von $\vec{p}(t)$ in die Normalenform erhält man

$$\begin{aligned} & 0 = \vec{n}_i \cdot \vec{p}(t) - \vec{n}_i \cdot \vec{p}_i = \vec{n}_i[\vec{p}_0 + t \cdot (\vec{p}_1 - \vec{p}_0) - \vec{p}_i] \\ \Longrightarrow \quad & t = \frac{\vec{n}_i \cdot (\vec{p}_0 - \vec{p}_i)}{-\vec{n}_i \cdot (\vec{p}_1 - \vec{p}_0)} \end{aligned}$$

Der Schnitt ist gültig, falls $0 \leq t \leq 1$. Sei $\vec{D} = \vec{p}_1 - \vec{p}_0$. Wir klassifizieren die Schnittpunkte als *potentially entering (PE)*, falls $\vec{n}_i \cdot \vec{D} < 0$ und als *potentially leaving (PL)*, falls $\vec{n}_i \cdot \vec{D} > 0$. Sei nun t_{PE} das größte t, bei dem der Schnittpunkt potentially entering ist und t_{PL} das kleinste t, bei dem der Schnittpunkt potentially leaving ist. Falls $t_{PE} < t_{PL}$, so clippen wir.

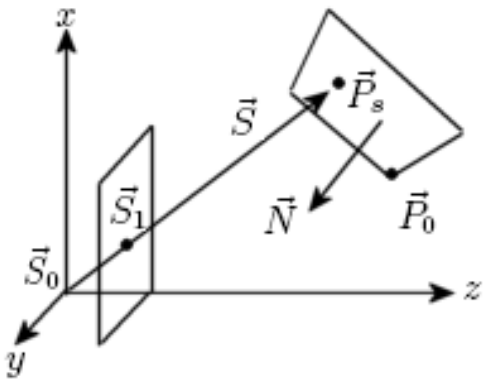
4 Schnittberechnung bei Ray Casting

4.1 Schnitte mit Polygonen

Wir wollen berechnen, ob ein gegebener Strahl ein gegebenes Polygon trifft. Dazu betrachten wir

- Einen Strahl $\vec{S} = t\vec{S_1}$ durch den Ursprung S_0 und einen Punkt $S_1 = (x_1, y_1, 1)^T$ in der Projektionsebene $z = 1$.
- Die Ebene, die das Polygon enthält, beschrieben durch die Normale $\vec{N}$ und einen Punkt $\vec{P_0}$ in Normalenform $\vec{N} \cdot \vec{P} - \vec{N} \cdot \vec{P_0} = 0$

Zuerst berechnen wir den Schnittpunkt $\vec{P_s}$ von dem Strahl mit der Ebene.



Einsetzen von $\vec{S} = t\vec{S_1}$ in die Normalengleichung ergibt

$$\vec{N} \cdot \vec{P_s} - \vec{N} \cdot \vec{P_0} = 0 \implies \vec{N} \cdot (t_s\vec{S_1} - \vec{P_0}) = 0 \implies t_s = \frac{\vec{N} \cdot \vec{P_0}}{\vec{N} \cdot \vec{S_1}}$$

Als zweites müssen wir testen, ob der Punkt P_s wirklich innerhalb des uns interessierenden Polygons liegt. In der Ebene macht man diesen Test, indem man einen Strahl vom Punkt weg schickt und die Anzahl der Schnitte des Strahls mit den Kanten des Polygons zählt. Ist diese Anzahl ungerade, so liegt der Punkt innerhalb des Polygons, sonst nicht.

Bei diesem Verfahren entstehen zusätzliche Schwierigkeiten, falls z.B. der Strahl parallel zu einer der Kanten verläuft, oder falls der Strahl eine Ecke des Polygons trifft. Man vermeidet diese Probleme, indem man das Polygon trianguliert und das Verfahren auf die Dreiecke anwendet. Dazu reicht es zu jedem Dreieck die baryzentrischen Koordinaten des Punktes auszurechnen.

4.2 Schnitte mit Kugeloberflächen

Sei eine Kugel gegeben mit dem Mittelpunkt $M = (a, b, c)^T$ und Radius r. Wir möchten die Schnittpunkte eines Strahls $\vec{S} = t_s(x_s, y_s, 1)^T$ mit der Kugel berechnen. Die parametrische Form der Kugel ist

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2$$

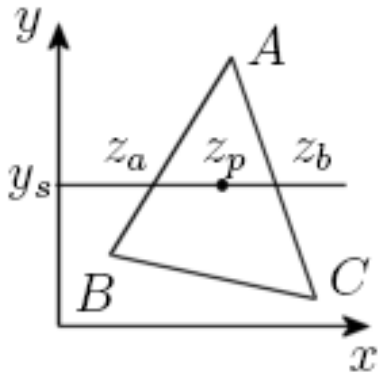
Daraus ergibt sich die Gleichung für die Schnittpunkte

$$(t_s^2 x_s^2 - a)^2 + (t_s^2 y_s^2 - b)^2 + (t_s^2 - c)^2 = r^2$$

5 Effiziente Tiefenberechnung für Polygone

5.1 Incrementelle Tiefenberechnung

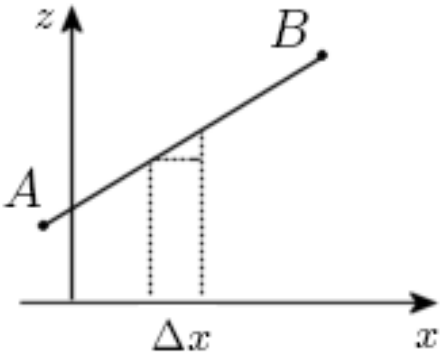
Sei ein Dreieck gegeben mit Ecken $A = (x_1, y_1, z_1), B = (x_2, y_2, z_2), C = (x_3, y_3, z_3)$.



Zu berechnen ist die z-Koordinate des Punktes p, der in der Ebene des Dreiecks liegt. Dazu betrachten wir die Schnittpunkte a und b der Segmente AB und BC mit der Ebene $y = y_s$. Es gilt

$$\begin{aligned} z_a &= z_1 + \frac{y_s - y_1}{y_2 - y_1}(z_2 - z_1) \\ z_b &= z_1 + \frac{y_s - y_1}{y_3 - y_1}(z_3 - z_1) \\ z_p &= z_a + \frac{x_p - x_a}{x_b - x_a}(z_b - z_a) \end{aligned}$$

Man kann mit diesem Verfahren auch andere Größen interpolieren, z.B. Farben. Eine Optimierung wäre die incrementelle Berechnung der Tiefe. Wir betrachten dazu zwei Punkte $A = (x_1, y_1, z_1)$ und $B = (x_2, y_2, z_2)$.



Es gilt

$$z(x + \Delta x) = z(x) + \frac{dz}{dx} \cdot \Delta x, \quad \text{wobei} \quad \frac{dz}{dx} = \frac{z_2 - z_1}{x_2 - x_1}$$

Diskret (falls $\Delta x = 1$ Pixel):

$$z_{i+1} \quad = \quad z_i + \frac{dz}{dx}$$

Index