Übungen - Bildgenierung
Übung 04.

Jose Jimenez

Angewandte Informatik
Bergische Universität Wuppertal

November 19, 2024

Table of Contents

Aufgabe 10 Perspektivische Projektion

Die Hauptidee dieser Aufgabe ist es, eine **3D-zu-2D-Perspektivprojektion zu implementieren**, indem man Matrizen verwendet.
Ziel ist es,

- **eine Transformationsmatrix Schritt für Schritt zu erstellen** und
- sie dann anzuwenden, um 3D-Objekte in einer 2D-Darstellung darzustellen.

Aufgabe 10 Perspektivische Projektion

Programmieren Sie die perspektivische Projektion, die Überführung in normalisierte Koordinaten sowie die Umwandlung in Gerätekoordinaten. Ergänzen Sie hierzu im Rahmenprogramm proj1.cc im Verzeichnis /home/bildgen/Aufgaben/projektion-1 die entsprechenden Teile der Funktionen ....

Aufgabe 10 Perspektivische Projektion

Rahmen Program

```
Matrix4x4 berechneTransformation(const Vec3D& cop, const Vec3D& vrp,
                                    const Vec3D& vup, int w, int h,
                                    double& un, double& vn)
```

```
void maleLinien(Drawing& pic, const vector<Kante>& kanten,
                const Matrix4x4& t, double un, double vn)
```

- **COP** (Center of Projection): Der Punkt, an dem die Kamera steht (in der Welt) und von dem aus sie alles "sieht."
- **VRP** (View Reference Point): Der Punkt, auf den die Kamera schaut oder den sie fokussiert.
- **VUP** (View Up Vector) ist ein Vektor, der zeigt, welche Richtung für die Kamera "oben" sein soll.

Aufgabe 10 Perspektivische Projektion

Rahmen Program

```
Matrix4x4 berechneTransformation(const Vec3D& cop, const Vec3D& vrp,
                                 const Vec3D& vup, int w, int h,
                                 double& un, double& vn)
```

```
void maleLinien(Drawing& pic, const vector<Kante>& kanten,
                const Matrix4x4& t, double un, double vn)
```

Was ist neu für uns?

Aufgabe 10 Perspektivische Projektion

Rahmen Program

```
Matrix4x4 berechneTransformation(const Vec3D& cop, const Vec3D& vrp,
                                  const Vec3D& vup, int w, int h,
                                  double& un, double& vn)
```

```
void maleLinien(Drawing& pic, const vector<Kante>& kanten,
                const Matrix4x4& t, double un, double vn)
```

Was ist neu für uns? **Wichtig!**

- Matrix $4 \times 4 \rightarrow$ doc
- Vec3D $\rightarrow$ doc
- Kante $\rightarrow$

see doc

Aufgabe 10 Perspektivische Projektion

Zuerst....

Spoiler: Multiplikation der einzelnen Transformationen.
Idee: Wir erstellen alle Transformationsmatrizen...
Bedeutung der Schritte:

- **tvrp :** Verschiebe den VRP (Blickpunkt) zum Ursprung.
- **rot :** Rotiere das System, damit u, v, n mit der Kamera übereinstimmen.
- **transcop:** Verschiebe die Kamera (COP) zum Ursprung.
- **proj :** Projiziere die 3D-Szene auf die 2D-Ebene.
- **t :** Verschiebe das Projektionsfenster so, dass $(umin, vmin)$ bei $(0, 0)$ liegt.
- **s :** Skaliere das Fenster, damit es in das Einheitsquadrat passt.

Aufgabe 10 Perspektivische Projektion

1 Verschiegung des VRP der Projektionsebene

Die Verschiebung des VRP zum Ursprung bedeutet, dass der Punkt, auf den die Kamera schaut, im Weltkoordinatensystem auf (0,0,0) verschoben wird. Das macht es einfacher, alle weiteren Transformationen wie Drehungen und Projektionen durchzuführen, weil alles jetzt relativ zur Kamera berechnet wird.

Aufgabe 10 Perspektivische Projektion

Rahmen Program

1. Verschiebung von vrp in den Ursprung
   - Das haben wir beim letzten Mal gelernt. Quasi.
   - Heute brauchen wir nur **eine Verschibung** des Aufpunktes der Projektionsebene (vrp) in den Ursprung.

1. Verschiebung von vrp in den Ursprung
    - Das haben wir beim letzten Mal gelernt. Quasi.

    - Heute brauchen wir nur **eine Verschibung** des Aufpunktes der Projektionsebene (vrp) in den Ursprung.
    - Was ist die T-Matrix? Wie können wir in C++ schreiben?

1. Verschiebung von vrp in den Ursprung
   - Das haben wir beim letzten Mal gelernt. Quasi.

   - Heute brauchen wir nur **eine Verschibung** des Aufpunktes der Projektionsebene (vrp) in den Ursprung.
   - Was ist die T-Matrix? Wie können wir in C++ schreiben?

Aufgabe 10 Perspektivische Projektion

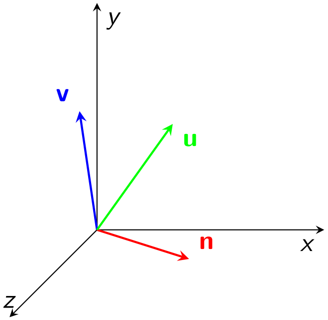
Rahmen Program

1. Verschiebung von vrp in den Ursprung
   - Das haben wir beim letzten Mal gelernt. Quasi.

   - Heute brauchen wir nur **eine Verschibung** des Aufpunktes der Projektionsebene (vrp) in den Ursprung.
   - Was ist die T-Matrix? Wie können wir in C++ schreiben?

```
tvrp.el[0][0] = tvrp.el[1][1] = tvrp.el[2][2] = tvrp.el[3][3] = 1;
 tvrp.el[0][3] = -vrp.el[0];
 tvrp.el[1][3] = -vrp.el[1];
 tvrp.el[2][3] = -vrp.el[2];
```

2. a) Bestimme (u,v,n)-Koordinatensystem (Seiten 4-15 —-4-18) Die u-Achse (u,v,n) sind normierte Vektoren die ein rechtshändiges 3D-Koordinatensystem bilden



- **Vektor n**: Blickrichtung der Kamera
- **Vektor v**: Oben-Richtung der Kamera
- **Vektor u**: Rechts-Richtung der Kamera (**VPN**: View Plane Normal)

2. a) Bestimme (u,v,n)-Koordinatensystem (Seiten 4-15 —-4-18) Wir brauchen n,v und u.

2. a) Bestimme (u,v,n)-Koordinatensystem (Seiten 4-15 —-4-18) Wir brauchen n,v und u.

- Die u-Achse wird so gewählt, dass (u,v,n) (normierte Vektoren) ein rechtshändiges 3D-Koordinatensystem bilden.
- n ist der Normalenvektor, und wir haben ihn schon. (vpn).
- v: Orthogonalprojektion von up auf Projektionsebene.

```
n = vpn / norm(vpn);

v = vup - skalarprod(vup, n) * n;
v = v / norm(v);

u = kreuzprod(v, n);
```

Die Matrix in 4-19 beschreibt eine **Rotation**, da sie die Achsen des Weltkoordinatensystems (x, y, z) so dreht, dass sie mit den Achsen des Kamerakoordinatensystems (u, v, n) übereinstimmen:

Die Matrix in 4-19 beschreibt eine **Rotation**, da sie die Achsen des Weltkoordinatensystems (x, y, z) so dreht, dass sie mit den Achsen des Kamerakoordinatensystems (u, v, n) übereinstimmen:

- **u**: Die neue x-Achse zeigt nach rechts (berechnet aus dem Weltkoordinatensystem).
- **v**: Die neue y-Achse zeigt nach oben.
- **n**: Die neue z-Achse zeigt in die Blickrichtung der Kamera.

Diese Drehung wird durch die Anordnung der Vektoren **u**, **v** und **n** als Zeilen in der Matrix erreicht. Ein Punkt **p** wird wie folgt transformiert:

$$\mathbf{p}' = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ n_x & n_y & n_z \end{bmatrix} \cdot \mathbf{p}$$

Dadurch wird der Punkt aus dem Weltkoordinatensystem in das Kamerakoordinatensystem gedreht.

2. b) Rotation: $(x, y, z) \rightarrow (u, v, n)$ In C++.

2. b) Rotation: $(x, y, z) \rightarrow (u, v, n)$ In C++.

```
 /*
  / ux uy uz   \
  | vx vy vz   |
  | nx ny nz   |
  \          1 /
*/
for (i = 0; i < 3; ++i)
  {
    rot.el[0][i] = u.el[i];
    rot.el[1][i] = v.el[i];
    rot.el[2][i] = n.el[i];
  }
rot.el[3][3] = 1;
```

3) Verschieben der transformierten Augenposition (cop überführt in das (u,v,n)-Koordinatensystem) in den Koordinatenursprung.

- Der Augenpunkt liegt nicht im Ursprung, sondern an Position $(0, 0, -z_p)$ (4-17) (z_p ist Z_n in Rahmen Programm).

3) Verschieben der transformierten Augenposition (cop überführt in das (u,v,n)-Koordinatensystem) in den Koordinatenursprung.

- Der Augenpunkt liegt nicht im Ursprung, sondern an Position $(0, 0, -z_p)$ (4-17) (z_p ist Z_n in Rahmen Programm).
- Wie sieht die Matrix aus? (z_p ist Z_n in Rahmen Programm).

3) Verschieben der transformierten Augenposition (cop überführt in das (u,v,n)-Koordinatensystem) in den Koordinatenursprung.

- Der Augenpunkt liegt nicht im Ursprung, sondern an Position $(0, 0, -z_p)$ (4-17) (z_p ist Z_n in Rahmen Programm).
- Wie sieht die Matrix aus? (z_p ist Z_n in Rahmen Programm).

```
  /*
  / 1           \
  |   1         |
  |     1  -zn  |
  \        1    /
*/
transcop.el[0][0] = transcop.el[1][1] = ...
transcop.el[2][2] = transcop.el[3][3] = 1;
transcop.el[2][3] = -znear;
```

4) Standard perspektivische Projektion (Seite 4-24)
Versuche, die Elemente der Matrix **proj** festzulegen.

4) Standard perspektivische Projektion (Seite 4-24)
Versuche, die Elemente der Matrix **proj** festzulegen.

```
/*
  / -zn          \
  |     -zn      |
  |              |
  \          1   /
*/
proj.el[0][0] = proj.el[1][1] = -znear;
proj.el[2][2] = 0;
proj.el[3][2] = 1;
```

Aufgabe 10 Perspektivische Projektion
Rahmen Program

5. a) Translation des Bereichs [umin; umax] $\times$ [vmin; vmax] so dass (umin, vmin) $\to$ (0, 0)

5. a) Translation des Bereichs [umin; umax] $\times$ [vmin; vmax] so dass (umin, vmin) $\rightarrow$ (0, 0)

```
/*
 / 1      -umin \
 |   1    -vmin |
 |     ?        |
 \         1    /
*/
t.el[0][0] = t.el[1][1] = t.el[3][3] = 1;
t.el[2][2] = 0;
t.el[0][3] = -umin;
t.el[1][3] = -vmin;
```

5. b) Skalierung, so dass das Fenster in das **Einheitsquadrat** passt.
Skalierung-Matrix ist Einfach

5. b) Skalierung, so dass das Fenster in das **Einheitsquadrat** passt.
Skalierung-Matrix ist Einfach

```
/*
  / ft        \
  |    ft     |
  |        ?  |
  \         1 /
*/
double fu = 1.0 / (umax - umin); // Faktor für u
double fv = 1.0 / (vmax - vmin); // Faktor für v
double ft = min(fu, fv);         // finaler Streckfaktor
s.el[0][0] = s.el[1][1] = ft;
//s.el[2][2] = 0; // Die z-Koordinate interessiert uns hier nicht
s.el[3][3] = 1;
```

6. Multiplikation der einzelnen Transformationen.
Einfach: Wir haben alle Transformationsmatrizen...
Bedeutung der Schritte:

- **tvrp :** Verschiebe den VRP (Blickpunkt) zum Ursprung.
- **rot :** Rotiere das System, damit u, v, n mit der Kamera übereinstimmen.
- **transcop:** Verschiebe die Kamera (COP) zum Ursprung.
- **proj :** Projiziere die 3D-Szene auf die 2D-Ebene.
- **t :** Verschiebe das Projektionsfenster so, dass $(umin, vmin)$ bei $(0, 0)$ liegt.
- **s :** Skaliere das Fenster, damit es in das Einheitsquadrat passt.

Also, in C:

6. Multiplikation der einzelnen Transformationen.
In C

```
mzen =  s * t * proj * transcop * rot * tvrp;
```

Aufgabe 10 Perspektivische Projektion

malt die transformierten Kanten ins Bild pic (2D)

Was sollen wir denn tun? (Rahmen)

```
void maleLinien(Drawing& pic, const vector<Kante>& kanten,
                const Matrix4x4& t, double un, double vn)
```

Aufgabe 10: Perspektivische Projektion

Rahmenprogramm

Zum Zeichnen des Bildes ist dann noch Folgendes in `maleLinien()` zu tun:

1. Anwenden der Transformationsmatrix auf Anfangs- und Endpunkt der einzelnen Linien (der entsprechende Code-Abschnitt ist vorgegeben).

Aufgabe 10: Perspektivische Projektion

Rahmenprogramm

Zum Zeichnen des Bildes ist dann noch Folgendes in `maleLinien()` zu tun:

1. Anwenden der Transformationsmatrix auf Anfangs- und Endpunkt der einzelnen Linien (der entsprechende Code-Abschnitt ist vorgegeben).
2. Umwandlung der homogenen Koordinaten in 2D-Koordinaten.

Zum Zeichnen des Bildes ist dann noch Folgendes in `maleLinien()` zu tun:

1. Anwenden der Transformationsmatrix auf Anfangs- und Endpunkt der einzelnen Linien (der entsprechende Code-Abschnitt ist vorgegeben).
2. Umwandlung der homogenen Koordinaten in 2D-Koordinaten.
3. Skalierung auf Fenstergröße (Gerätekoordinaten) unter Verwendung von u_m und v_n.

Umwandlung der homogenen Koordinaten in 2D-Koordinaten.

```
void maleLinien(Drawing& pic, const vector<Kante>& kanten,
                const Matrix4x4& t, double un, double vn){
    Vec4D nanf, nend;    // Anfangs- und Endpunkt nach Normalisierung
    DPoint2D panf, pend; // Anfangs- und Endpunkt im Bild pic
    ...
      panf.x = nanf.el[0] / nanf.el[3];
      panf.y = nanf.el[1] / nanf.el[3];
      pend.x = nend.el[0] / nend.el[3];
      pend.y = nend.el[1] / nend.el[3];
    }
```

Die Perspektivdivision teilt die x-, y- und z-Koordinaten durch die w-Koordinate, um Punkte in 3D auf die 2D-Ebene zu projizieren. Dadurch erscheinen weiter entfernte Objekte kleiner, was einen realistischen **Tiefeneffekt** erzeugt.

Skalierung auf Fenstergröße (Gerätekoordinaten) unter Verwendung von u_m und v_n .

Skalierung auf Fenstergröße (Gerätekoordinaten) unter Verwendung von u_m und v_n .

Die x- und y-Koordinaten werden mit s_x und s_y skaliert, um sie vom Einheitsquadrat in den Pixelbereich der Leinwand umzuwandeln.

Skalierung auf Fenstergröße (Gerätekoordinaten) unter Verwendung von u_m und v_n .

```
void maleLinien(Drawing& pic, const vector<Kante>& kanten,
                const Matrix4x4& t, double un, double vn){

// Skalierungsfaktoren:
    double sx = pic.getWidth() / un;  //Breite
    double sy = pic.getHeight() / vn;  //Höhe

    DPoint2D panf, pend; // Anfangs- und Endpunkt im Bild pic
    ...
      panf.x *= sx;
      panf.y *= sy;
      pend.x *= sx;
      pend.y *= sy;
    }
```

Fertig1!

Ihr habt den Algorithmus schon gelernt, Ihr muss nur es anwenden.

Rechteck $[2;8]x[1;5]$ **Linien**

a) $P_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, P_2 = \begin{pmatrix} 10 \\ 4 \end{pmatrix}$

b) $P_1 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, P_2 = \begin{pmatrix} 6 \\ 0 \end{pmatrix}$

c) $P_1 = \begin{pmatrix} 7 \\ 0 \end{pmatrix}, P_2 = \begin{pmatrix} 10 \\ 2 \end{pmatrix}$

d) $P_1 = \begin{pmatrix} 5 \\ 6 \end{pmatrix}, P_2 = \begin{pmatrix} 11 \\ 4 \end{pmatrix}$

Rechteck $[2; 8]x[1; 5] \rightarrow \overline{u} = 8, \quad \overline{v} = 5, \quad \underline{u} = 2, \quad \underline{v} = 1.$

Linien

c) $P_1 = \begin{pmatrix} 7 \\ 0 \end{pmatrix}, P_2 = \begin{pmatrix} 10 \\ 2 \end{pmatrix}$

code1 $= (u1 > \overline{u}, v1 > \overline{v}, u1 < \underline{u}, v1 < \underline{v})$
code2 $= (u2 > \overline{u}, v2 > \overline{v}, u2 < \underline{u}, v2 < \underline{v})$

Aufgabe 11 Strecken-Clipping nach Cohen und Sutherland
Algorithmus

Rechteck $[2;8]x[1;5] \to \overline{u} = 8, \quad \overline{v} = 5, \quad \underline{u} = 2, \quad \underline{v} = 1.$

Linien

d) $P_1 = \begin{pmatrix} 5 \\ 6 \end{pmatrix}, P_2 = \begin{pmatrix} 11 \\ 4 \end{pmatrix}$

$\text{code1} = (u1 > \overline{u}, v1 > \overline{v}, u1 < \underline{u}, v1 < \underline{v})$
$\text{code2} = (u2 > \overline{u}, v2 > \overline{v}, u2 < \underline{u}, v2 < \underline{v})$

Aufgabe 11 Strecken-Clipping nach Cohen und Sutherland

Algorithmus

Rechteck $[2;8]x[1;5] \rightarrow \overline{u} = 8, \quad \overline{v} = 5, \quad \underline{u} = 2, \quad \underline{v} = 1.$

Linien

b) $P_1 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, P_2 = \begin{pmatrix} 6 \\ 0 \end{pmatrix}$

c) $P_1 = \begin{pmatrix} 5 \\ 6 \end{pmatrix}, P_2 = \begin{pmatrix} 11 \\ 4 \end{pmatrix}$

Aufgabe 12 Strecken-Clipping nach Cyrus, Beck, Liang und Barsky

Algorithmus

Ihr habt den Algorithmus schon gelernt, Ihr muss nur es anwenden.

Octave script

Rechteck $[2;8]x[1;5]$ **Linien**

a) $P_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, P_2 = \begin{pmatrix} 10 \\ 4 \end{pmatrix}$

b) $P_1 = \begin{pmatrix} 3 \\ 3 \end{pmatrix}, P_2 = \begin{pmatrix} 6 \\ 0 \end{pmatrix}$

c) $P_1 = \begin{pmatrix} 7 \\ 0 \end{pmatrix}, P_2 = \begin{pmatrix} 10 \\ 2 \end{pmatrix}$

d) $P_1 = \begin{pmatrix} 5 \\ 6 \end{pmatrix}, P_2 = \begin{pmatrix} 11 \\ 4 \end{pmatrix}$