

Übungen - Bildgenierung

Übung 06.

Jose Jimenez

Angewandte Informatik
Bergische Universität Wuppertal

December 4, 2024



Table of Contents

- 1 Aufgabe 15: Painter's Algorithm
- 2 Aufgabe 16: Silhouetten-Algorithmus
- 3 Aufgabe 17: z-Buffer-Verfahren



Aufgabe 15: Painter's Algorithm

Kabinett Projektion.

Erzeugt die Matrix zur Parallelprojektion in gegebener Richtung mit gegebenem Projektionsfenster $[u_{min}; u_{max}] \times [v_{min}; v_{max}]$ in das Einheitsquadrat...

Dafür brauchen wir:

- ① Kabinett-Projektion Matrix *spar*
- ② Verschiebung Matrix T
- ③ Skalierung Matrix S

Die Skalierungsfaktor ist $s_x = \frac{1}{u_{max} - u_{min}}$ und $s_y = \frac{1}{v_{max} - v_{min}}$



Aufgabe 15: Painter's Algorithm

Kabinet Projektion.

Erzeugt die Matrix zur Parallelprojektion in gegebener Richtung mit gegebenem Projektionsfenster $[u_{min}; u_{max}] \times [v_{min}; v_{max}]$ in das Einheitsquadrat...

Dafür brauchen wir:

- ① Kabinet-Projektion Matrix $spar$
- ② Verschiebung Matrix T
- ③ Skalierung Matrix S

Die Skalierungsfaktor ist $s_x = \frac{1}{u_{max} - u_{min}}$ und $s_y = \frac{1}{v_{max} - v_{min}}$

Dann, die **Matrix zur Parallelprojektion** lautet: $mpar = S \times T \times spar$.



Aufgabe 15: Painter's Algorithm

Kabinettsprojektion

Kabinettsprojektion: Die s- und f-Achse werden waagerecht bzw. senkrecht dargestellt, während die t-Achse um 30° geneigt und um den Faktor $\frac{1}{2}$ verkürzt ist.



Aufgabe 15: Painter's Algorithm

Kabinettsprojektion

Kabinettsprojektion: Die s- und f-Achse werden waagerecht bzw. senkrecht dargestellt, während die t-Achse um 30° geneigt und um den Faktor $\frac{1}{2}$ verkürzt ist.

- $s \rightarrow x$
- $f \rightarrow y$
- $t \rightarrow z$

z ist die Projektionsachse. Die Projektion wird in die x-y-Ebene durchgeführt.

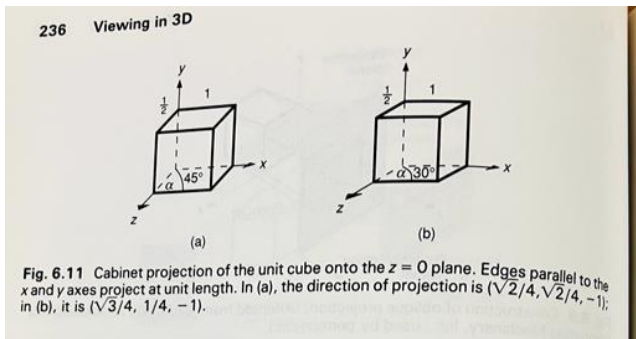


Aufgabe 15: Painter's Algorithm

Kabinnet-Projektion.

Kabinnet-Projektion. Die x- und die y-Achse waagerecht bzw. senkrecht dargestellt werden und die z-Achse um 30° geneigt und um den Faktor $\frac{1}{2}$ verkürzt ist.

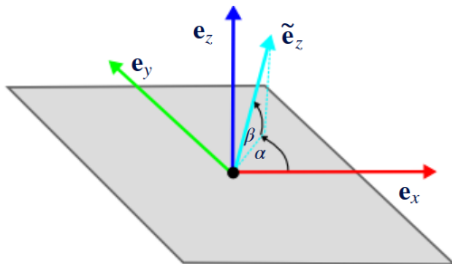
Die Kabinnet Projektion ist Spezialfall von **Schiefe Projektionen**.



Aufgabe 15: Painter's Algorithm

Kabinnet-Projektion.

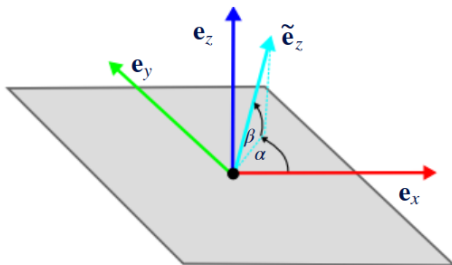
Kabinnet-Projektion. Die x- und die y-Achse waagerecht bzw. senkrecht dargestellt und die z-Achse um 30° geneigt und um den Faktor $\frac{1}{2}$ verkürzt.



Aufgabe 15: Painter's Algorithm

Kabinnet-Projektion.

Kabinnet-Projektion. Die x- und die y-Achse waagerecht bzw. senkrecht dargestellt und die z-Achse um 30° geneigt und um den Faktor $\frac{1}{2}$ verkürzt.

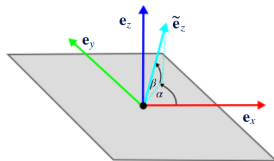


$$\begin{aligned}x &= x \pm \frac{1}{2}z \cdot \cos(\alpha) \\y &= y \pm \frac{1}{2}z \cdot \sin(\alpha) \\ \alpha &= 30\end{aligned}$$



Aufgabe 15: Painter's Algorithm

Kabinett-Projektion.



$$P \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \pm \frac{1}{2}z \cdot \cos(\alpha) \\ y \pm \frac{1}{2}z \cdot \sin(\alpha) \\ 0 \end{pmatrix}$$

$$\rightarrow spar = \begin{pmatrix} 1 & sx & & \\ & 1 & sy & \\ & & 0 & \\ & & & 1 \end{pmatrix}$$

Aufgabe 15: Painter's Algorithm

Kabinet Projektion.

Wir brauchen:

- 1 Kabinet-Projektion Matrix *spar* \Rightarrow **Done**
- 2 Verschiebung Matrix T
- 3 Skalierung Matrix S

Wir wissen schon wie die Verschiebung und Skalierung Matrixen aus sieth.

Die Skalierungfaktor ist $s_x = \frac{1}{u_{max} - u_{min}}$ und $s_y = \frac{1}{v_{max} - v_{min}}$



Aufgabe 15: Painter's Algorithm

Kabinet Projektion.

Wir brauchen:

- 1 Kabinet-Projektion Matrix $spar \Rightarrow$ **Done**
- 2 Verschiebung Matrix T
- 3 Skalierung Matrix S

Wir wissen schon wie die Verschiebung und Skalierung Matrixen aus sieth.

Die Skalierungfaktor ist $s_x = \frac{1}{u_{max} - u_{min}}$ und $s_y = \frac{1}{v_{max} - v_{min}}$

Dann, die **Matrix zur Parallelprojektion** lautet: $mpar = S \times T \times spar$.



Aufgabe 15: Painter's Algorithm

Kabinnet-Projektion.

```
Matrix4x4 berechneMpar(double umin, double umax, double vmin, double
↳ vmax, double& ratio)
{
    //=> Ankathete = 0.5 * cos(30°) = 0.5 * sqrt(3)/2 = sqrt(3)/4
    //=> Gegenkathete = 0.5 * sin(30°) = 0.5 * 0.5 = 0.25
    /* / 1 sx \
       * / 1 sy /
       * / 0 /
       * \ 1 / */
    Matrix4x4 spar;
    spar.el[0][0] = spar.el[1][1] = spar.el[3][3] = 1;
    spar.el[0][2] = -sqrt(3) / 4.0;
    spar.el[1][2] = -1.0 / 4.0;
}
```



```

...
/* / 1      tx \
 * /      1    ty /
 * /      1    /
 * \      1 / */
Matrix4x4 transl;
transl.el[0][0] = transl.el[1][1] = transl.el[2][2] =
↪ transl.el[3][3] = 1;
transl.el[0][3] = -umin;
transl.el[1][3] = -vmin;
/* / sx      \
 * /      sy  /
 * /      1  /
 * \      1 / */
Matrix4x4 scale;
scale.el[0][0] = 1 / (umax - umin);
scale.el[1][1] = 1 / (vmax - vmin);
scale.el[2][2] = scale.el[3][3] = 1;

```

Aufgabe 15: Painter's Algorithm

Painter's algorithm.

```
void erzeugeFlaeche(double xmin, double xmax, double zmin, do  
    /* Rahmen Program */  
}
```



Aufgabe 15: Painter's Algorithm

Painter's algorithm.

```
void erzeugeFlaeche(double xmin, double xmax, double zmin, do  
    /* Rahmen Program */  
}
```

- for ($z = 1, \dots, num$) Von hinten nach vorne. Hängt stark von der Blickrichtung ab
 - for ($x = 1, \dots, num$)
 - ① Die vier Ecken der aktuellen Zelle holen
 - ② Zwei Dreiecke aus den Ecken erstellen
 - ③ Farbe für die Dreiecke zuweisen
 - ④ Dreiecke zur Liste hinzufügen
 - end for
- end for




Aufgabe 15: Painter's Algorithm

Painter's algorithm.

```
void erzeugeFlaeche(...)
```

```
{
```

 Von hinten nach vorne. Hängt stark von der Blickrichtung ab.

```
    for (int z = 0; z < num; ++z){
```

```
}
```



Aufgabe 15: Painter's Algorithm

Painter's algorithm.

```
void erzeugeFlaeche(...)
{
    for (int z = 0; z < num; ++z){
        for (int x = 0; x < num; ++x) // Viereck
        {
            // Jedes Viereck besteht aus zwei Dreiecken.
            Dreieck d1, d2;
        }
    }
}
```



Aufgabe 15: Painter's Algorithm

Painter's algorithm.

```
void erzeugeFlaeche(...){
    for (int z = 0; z < num; ++z){
        for (int x = 0; x < num; ++x){
            // Jedes Viereck besteht aus zwei Dreiecken.
            Dreieck d1, d2;

            /* 0 -> xmin, num -> xmax, Steigung ist dann
             * (xmax - xmin) / num z analog. */

            double xl = xmin + x * (xmax - xmin) / num;
            double xh = xmin + (x + 1) * (xmax - xmin) / num;
            double zl = zmin + z * (zmax - zmin) / num;
            double zh = zmin + (z + 1) * (zmax - zmin) / num;
        }
    }
}
```

Aufgabe 15: Painter's Algorithm

Painter's algorithm.

Für jede Gitterzelle: Zwei Dreiecke, $d1$ und $d2$, werden erstellt, um ein Viereck zu bilden .



Aufgabe 15: Painter's Algorithm

Painter's algorithm.

```
void erzeugeFlaeche(...){
    for (int z = 0; z < num; ++z){
        for (int x = 0; x < num; ++x){
            // Jedes Viereck besteht aus zwei Dreiecken.
            Dreieck d1, d2;
            double xl = xmin + x * (xmax - xmin) / num;
            double xh = xmin + (x + 1) * (xmax - xmin) / num;
            double zl = zmin + z * (zmax - zmin) / num;
            double zh = zmin + (z + 1) * (zmax - zmin) / num;
            // Berechne Funktionswerte und erzeuge 3D-Punkte (4D-hom.)
            ↪ mit
            // Funktionswert als y-Koordinate.
            Vec4D p1(xl, func(xl, zl), zl, 1);
            Vec4D p2(xl, func(xl, zh), zh, 1);
            Vec4D p3(xh, func(xh, zl), zl, 1);
            Vec4D p4(xh, func(xh, zh), zh, 1);
        }
    }
}
```

Aufgabe 15: Painter's Algorithm

Painter's algorithm.

```
void erzeugeFlaeche(...){
    for (int z = 0; z < num; ++z){
        for (int x = 0; x < num; ++x){
            Dreieck d1, d2;
            double xl = xmin + x * (xmax - xmin) / num;
            double xh = xmin + (x + 1) * (xmax - xmin) / num;
            double zl = zmin + z * (zmax - zmin) / num;
            double zh = zmin + (z + 1) * (zmax - zmin) / num;
            Vec4D p1(xl, func(xl, zl), zl, 1);
            Vec4D p2(xl, func(xl, zh), zh, 1);
            Vec4D p3(xh, func(xh, zl), zl, 1);
            Vec4D p4(xh, func(xh, zh), zh, 1);

            // Weise Punkte zu Dreiecken zu.    (Assign)
            d1.ecke[0] = p1; d1.ecke[1] = p2; d1.ecke[2] = p3;
            d2.ecke[0] = p2; d2.ecke[1] = p4;    d2.ecke[2] = p3;
        }
    }
}
```

Aufgabe 15: Painter's Algorithm

Painter's algorithm.

Farbe?

```
...  
// Nur für die Färbung:  
// Skalierung von [0; 1] auf [0; 255].  
double xval = static_cast<double>(x) / num;  
double zval = static_cast<double>(z) / num;  
// Farbe. Rot in x-Richtung, Grün in z-Richtung, Blau invers in  
// beide Richtungen.  
d1.col = DrawColour(255 * xval, 255 * zval,  
                    255 * (1 - xval) * (1 - zval));  
d2.col = DrawColour(255 * xval, 255 * zval,  
                    255 * (1 - xval) * (1 - zval));  
  
// In Liste einfügen.  
dreiecke.push_back(d1);  
dreiecke.push_back(d2);
```

Fertig. Run!

Aufgabe 16 Silhouetten-Algorithmus

Silhouetten-Algorithmus

Relativ einfach ...

```
Matrix4x4 berechneMpar(double umin, double umax, double vmin, double vmax,  
                        double& ratio)
```

```
void erzeugeKurven(double xmin, double xmax, double zmin, double zmax,  
                  int num, int pieces,  
                  const std::function<double(double,double)>& func,  
                  std::vector<std::vector<Vec3D>>& kurven )
```

```
void maleSilhouetten(Drawing& pic,  
                    const std::vector<std::vector<Vec3D>>& kurven,  
                    const Matrix4x4& mpar, double ratio)
```

Rahmen Program



Aufgabe 16 Silhouetten-Algorithmus

Funktion: erzeugeKurven

Rahmen Program

```
void erzeugeKurven()

for (int z = 0; z < num; ++z)
{
    kurven[num - z - 1].resize(pieces + 1);

    /*Erinnerung:      num := Anzahl der Kurven
    *                  pieces := Anzahl der Strecken
    */
}
```

Wie Früher, wir brauchen die x- und z- Koordinaten.

$$posz = z_{min} + (z_{max} - z_{min}) \frac{z}{num}$$



Aufgabe 16 Silhouetten-Algorithmus

Funktion: erzeugeKurven

```
for (int z = 0; z < num; ++z){
    kurven[num - z - 1].resize(pieces + 1);
    //-----
    // z-Koordinate des Punktes.
    double posz = zmin + (zmax - zmin) * static_cast<double>(z) / num;

    // Einzelne Kurve, konstantes z, verbundene Punkte in x.
    for (int x = 0; x < pieces + 1; ++x){
        // x-Koordinate des Punktes.
        double posx = xmin + (xmax - xmin) * static_cast<double>(x) /
        ↪ pieces;
        // Speichere rückwärts, s.o.
        kurven[num - z - 1][x] = Vec3D(posx, func(posx, posz), posz);
    }
}
```

Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Funktion: maleSilhouetten 6.3.2



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Rahmen Program:

Drei for-Schleifen.



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Es gibt 4 möglichichkeiten:

- alt (x, y_l) unterhalb Kontur.
- alt (x, y_l) oberhalb Kontur.
- neu $(x + 1, y_r)$ unterhalb Kontur.
- neu $(x + 1, y_r)$ oberhalb Kontur



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Es gibt 4 möglichichkeiten:

- alt (x, y_l) unterhalb Kontur.
- alt (x, y_l) oberhalb Kontur.
- neu $(x + 1, y_r)$ unterhalb Kontur.
- neu $(x + 1, y_r)$ oberhalb Kontur

$$\text{schnitt} = \frac{\text{konturwert} - y_l}{(y_r - y_l) - (\text{konturwert}_{x+1} - \text{konturwert}_x)}$$



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1:

```
// Fall 1: beide auf selber Seite der Kontur
if ((konturmin[x] >= yl && konturmin[x + 1] >= yr) ||
    (konturmax[x] <= yl && konturmax[x + 1] <= yr)) {

    pic.drawLine(x, round(yl), x + 1, round(yr));

    konturmin[x] = min(konturmin[x], yl);
    konturmin[x + 1] = min(konturmin[x + 1], yr);
    konturmax[x] = max(konturmax[x], yl);
    konturmax[x + 1] = max(konturmax[x + 1], yr);
}
```



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 2/3 - a: alt (x, y_l) unterhalb Kontur **Schnittpunktes:**

$$\rightarrow x_{schnitt} = (kmin(x) - y_l) / (yr - y_l - kmin(x + 1) + kmin(x))$$



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 2/3 - a: alt (x, y_l) unterhalb Kontur **Schnittpunktes:**

$$\rightarrow x_{schnitt} = (kmin(x) - y_l) / (y_r - y_l - kmin(x + 1) + kmin(x))$$

```
if (yl < konturmin[x]) {  
    // Fall 2/3-a: alt (x, yl) unterhalb Kontur  
  
    schnitt = static_cast<double>(konturmin[x] - yl) /  
              (yr - yl - konturmin[x + 1] + konturmin[x]);  
  
    /* Nur zeichnen, wenn das Geradenstück von x nach x+1  
     * wirklich geschnitten wird.  
     */  
    if (schnitt >= 0 && schnitt <= 1)  
        pic.drawLine(x, round(yl), round(x + schnitt),  
                     round(yl + schnitt * (yr - yl)));  
    konturmin[x] = yl; //<-----  
}
```

Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 2/3 - b: alt (x, yl) oberhalb Kontur

```
if (yl > konturmax[x]) {  
    //Fall 2/3-b: alt (x, yl) oberhalb Kontur  
  
    schnitt = static_cast<double>(konturmax[x] - yl) /  
              (yr - yl - konturmax[x + 1] + konturmax[x]);  
    if (schnitt >= 0 && schnitt <= 1)  
        pic.drawLine(x, round(yl), round(x + schnitt),  
                     round(yl + schnitt * (yr - yl)));  
    konturmax[x] = yl;  
}
```



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 2/3-c: neu ($x + 1$, yr) unterhalb Kontur

```
if (yl > konturmax[x]) {  
    // Fall 2/3 - c: neu (x + 1, yr) unterhalb Kontur  
    schnitt = static_cast<double>(konturmax[x] - yl) /  
        (yr - yl - konturmax[x + 1] + konturmax[x]);  
    if (schnitt >= 0 && schnitt <= 1) {  
        pic.drawLine(x, round(yl), round(x + schnitt),  
            round(yl + schnitt * (yr - yl)));  
    }  
    konturmax[x] = yl;  
}
```



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 2/3-4: neu ($x + 1$, yr) oberhalb Kontur

```
if (yr > konturmax[x + 1]) {  
    // Fall 2/3-4: neu (x + 1, yr) oberhalb Kontur  
  
    schnitt = static_cast<double>(konturmax[x] - yl) /  
              (yr - yl - konturmax[x + 1] + konturmax[x]);  
    if (schnitt >= 0 && schnitt <= 1) {  
        pic.drawLine(round(x + schnitt),  
                     round(yl + schnitt * (yr - yl)), x + 1,  
                     round(yr));  
    }  
    konturmax[x + 1] = yr;  
}
```

Done! Run!



Aufgabe 17 (z-Buffer-Verfahren)

clip3DPoint

Der Kanonische Bildraum:

$$\begin{aligned} z &\in [-1, z_{min}] \\ x &\in [-z, z] \\ y &\in [-z, z] \end{aligned} \tag{2}$$

z -Koordinate und z_{min} sind negativ, $z_{min} > -1$.

Bedingungen für Lage des Punktes **p** AUSSERHALB des kanonischen Bildraums?



Aufgabe 17 (z-Buffer-Verfahren)

clip3DPoint

$$z \in [-1, z_{min}], \quad x \in [-z, z], \quad y \in [-z, z].$$

```
bool clip3DPoint(const Vec3D& p, double zmin)
{
    // 3D-Clipping im kanonischen Bildraum der Zentralprojektion
    // hier für einen einzelnen Punkt statt einer Linie.
    if (
        p.el[2] < -1           // weiter weg als z=-1
        || p.el[2] > zmin      // näher als z=zmin
        || p.el[0] < p.el[2]   // links der linken Kappungsebene
        || p.el[0] > -p.el[2]  // rechts der rechten Kappungsebene
        || p.el[1] < p.el[2]  // unterhalb der unteren Kappungsebene
        || p.el[1] > -p.el[2]  // oberhalb der oberen Kappungsebene
    )
        return false;

    return true;
}
```


Aufgabe 17 (z-Buffer-Verfahren)

clip3DPoint

Wir sollen 2 Funktionen implementieren...

Die Zweite:

```
inline void drawPointZ (Drawing& pic, int x, int y, double z,
                        vector<vector<double> >& zbuf, DrawColour colour)
{
    if (    x < 0 || x >= static_cast<int>(zbuf.size())
        || y < 0 || y >= static_cast<int>(zbuf[0].size()))
        return;

    // Befindet sich der neue Punkt vor dem zuvor gezeichneten Punkt?
    if (z > zbuf[x][y])
    { //Ja? ok. Zeichnen Punkt und aktualisier den Puffer
        pic.drawPoint(x, y, colour);
        zbuf[x][y] = z;
    }
```

