

Übungen - Bildgenierung

Übung 06.

Jose Jimenez

Angewandte Informatik
Bergische Universität Wuppertal

December 6, 2023



Table of Contents

1 Aufgabe 15: Painter's Algorithm

2 Aufgabe 16: Silhouetten-Algorithmus



Aufgabe 15: Painter's Algorithm

Kabinnet-Projektion.

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



Aufgabe 15: Painter's Algorithm

Kabinnet-Projektion.

Kabinnet Projektion. Die s- und die f-Achse waagerecht bzw. senkrecht dargestellt werden und 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 Projektionachse. Wir wollen die Projektion in die x-y-Ebene machen.



Aufgabe 15: Painter's Algorithm

Schiefe Projektionen.

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.

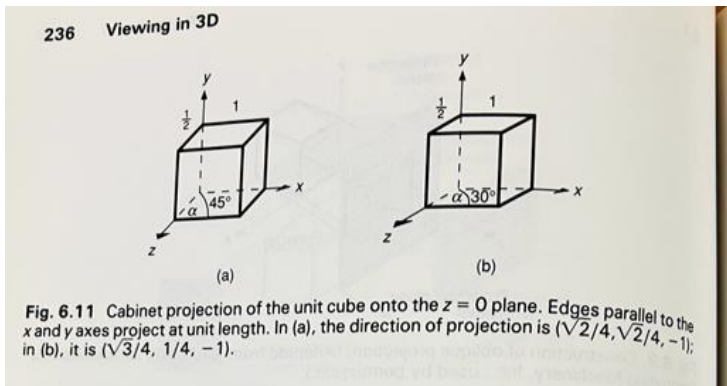
Die Kabinnet Projektion ist Spezialfall von **Schiefe Projektionen**.



Aufgabe 15: Painter's Algorithm

Kabinnet-Projektion.

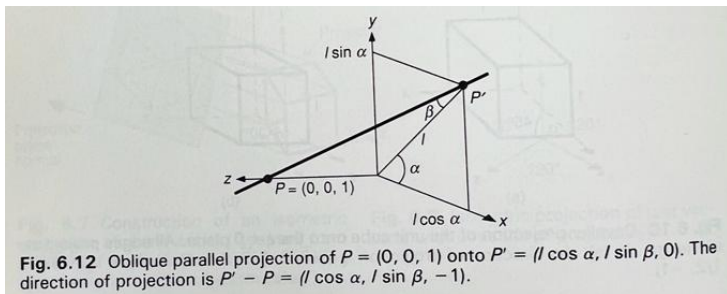
Kabinnet-Projektion. Die x- und die y-Achse waagerecht bzw. senkrecht dargestellt werden und die z-Achse um 30° geneigt und um den Faktor $\frac{1}{2}$ verkürzt ist.



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.



Aufgabe 15: Painter's Algorithm

Kabinnet-Projektion.

$$P \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \pm z(d\cos(\alpha)) \\ y \pm z(d\sin(\alpha)) \\ 0 \end{pmatrix}$$

$$d = \frac{1}{2}$$

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



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;
}
```



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 Verschiebung Matrix T
- ② 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

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 Verschiebung Matrix T

② 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:

$m_{par} = S \times T \times s_{par}$.



```

...
/* / 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(...)
```

```
{
```

 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)
        {
            // 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;

            /* Berechne Koordinaten im Raum. (oder besser.. Gitter
            ↪ erstellen)
               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 (jede Zelle besteht aus zwei Dreiecken).



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 x1 = xmin + x * (xmax - xmin) / num;
            double xh = xmin + (x + 1) * (xmax - xmin) / num;
            double z1 = 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(x1, func(x1, z1), z1, 1);
            Vec4D p2(x1, func(x1, zh), zh, 1);
            Vec4D p3(xh, func(xh, z1), z1, 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: bilde ab 0 -> 0 und num -> 1, jeweils für  
// x und z. Die Steigung ist dann 1/num.  
// 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);
```

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

```
// Erzeuge Kurven als Menge von Punkten in x-Richtung; fester  
↪ Abstand  
// der Kurven zueinander in z-Richtung.  
for (int z = 0; z < num; ++z)  
{  
    // Speichere rückwärts. Mit zunehmendem z bewegen wir uns von  
    // „hinten“ nach „vorne“.  
    kurven[num - z - 1].resize(pieces + 1);  
  
//Erinnerung:      num := Anzahl der Kurven
```

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

```
// Erzeuge Kurven als Menge von Punkten in x-Richtung; fester  
↪ Abstand  
// der Kurven zueinander in z-Richtung.  
for (int z = 0; z < num; ++z) {  
    // Speichere rückwärts. Mit zunehmendem z bewegen wir uns von  
    // „hinten“ nach „vorne“.  
    kurven[num - z - 1].resize(pieces + 1);
```

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

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

Einzelne Kurve, konstantes z, verbundene Punkte in x.

$$p_x = x_{min} + (x_{max} - x_{min}) \frac{x}{pieces}.$$

Natürlich x in einer for-Schleife...



Aufgabe 16 Silhouetten-Algorithmus

Funktion: erzeugeKurven

```
// Erzeuge Kurven als Menge von Punkten in x-Richtung; fester  
↪ Abstand  
// der Kurven zueinander in z-Richtung.  
for (int z = 0; z < num; ++z) {  
    // Speichere rückwärts. Mit zunehmendem z bewegen wir uns von  
    // „hinten“ nach „vorne“.  
    kurven[num - z - 1].resize(pieces + 1);
```

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

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

Einzelne Kurve, konstantes z, verbundene Punkte in x.

$$p_x = x_{min} + (x_{max} - x_{min}) \frac{x}{pieces}.$$

Natürlich x in einer for-Schleife...

Speichere rückwärts!!!



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

Wir denken an 4 Fälle:

- 1 alt (x, y_I) unterhalb Kontur.



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Wir denken an 4 Fälle:

- 1 alt (x, y_l) unterhalb Kontur.
- 2 alt (x, y_l) oberhalb Kontur.



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Wir denken an 4 Fälle:

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



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Wir denken an 4 Fälle:

- 1 alt (x, y_l) unterhalb Kontur.
- 2 alt (x, y_l) oberhalb Kontur.
- 3 neu $(x + 1, y_l)$ unterhalb Kontur.
- 4 neu $(x + 1, y_l)$ oberhalb Kontur



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: $\text{alt}(x, y_l)$ unterhalb Kontur.



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) unterhalb Kontur.

- Gerade 1 gegeben durch (x, y_l) , $(x + 1, y_r)$. (Actuelle y)
- Gerade 2 gegeben durch $(x, kmin(x))$, $(x + 1, kmin(x + 1))$. (Kontur y)



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) unterhalb Kontur.

- Gerade 1 gegeben durch (x, y_l) , $(x + 1, y_r)$. (Actuelle y)
- Gerade 2 gegeben durch $(x, kmin(x))$, $(x + 1, kmin(x + 1))$. (Kontur y)

Steigungen entsprechend

$$m_1 = y_r - y_l \quad m_2 = kmin(x + 1) - kmin(x)$$



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) unterhalb Kontur.

- Gerade 1 gegeben durch $(x, y_l), (x+1, y_r)$. (Actuelle y)
- Gerade 2 gegeben durch $(x, kmin(x)), (x+1, kmin(x+1))$. (Kontur y)

Steigungen entsprechend

$$m_1 = y_r - y_l \quad m_2 = kmin(x+1) - kmin(x)$$

Verlege (x, y_l) in den Ursprung, dann ist

$$n_1 = 0 \quad n_2 = kmin(x) - y_l.$$



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) unterhalb Kontur.

- Gerade 1 gegeben durch (x, y_l) , $(x + 1, y_r)$. (Actuelle y)
- Gerade 2 gegeben durch $(x, kmin(x))$, $(x + 1, kmin(x + 1))$. (Kontur y)

Steigungen entsprechend

$$m_1 = y_r - y_l \quad m_2 = kmin(x + 1) - kmin(x)$$

Verlege (x, y_l) in den Ursprung, dann ist

$$n_1 = 0 \quad n_2 = kmin(x) - y_l.$$

Verschobene x-Koordinate des **Schnittpunktes**:

$$(y_r - y_l)x = (kmin(x + 1) - kmin(x))x + (kmin(x) - y_l)$$

$$\rightarrow x = (kmin(x) - y_l) / (y_r - y_l - kmin(x + 1) + kmin(x))$$



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: $\text{alt}(x, y_l)$ unterhalb Kontur.



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) unterhalb Kontur. Wie Fall 2 oder 3 (Unten).

- Gerade 1 gegeben durch (x, y_l) , $(x + 1, y_r)$. (Actuelle y)
- Gerade 2 gegeben durch $(x, kmin(x))$, $(x + 1, kmin(x + 1))$. (Kontur y)



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) unterhalb Kontur. Wie Fall 2 oder 3 (Unten).

- Gerade 1 gegeben durch (x, y_l) , $(x+1, y_r)$. (Actuelle y)
- Gerade 2 gegeben durch $(x, kmin(x))$, $(x+1, kmin(x+1))$. (Kontur y)

Steigungen entsprechend

$$m_1 = y_r - y_l \quad m_2 = kmin(x+1) - kmin(x)$$



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) unterhalb Kontur. Wie Fall 2 oder 3 (Unten).

- Gerade 1 gegeben durch (x, y_l) , $(x+1, y_r)$. (Actuelle y)
- Gerade 2 gegeben durch $(x, kmin(x))$, $(x+1, kmin(x+1))$. (Kontur y)

Steigungen entsprechend

$$m_1 = y_r - y_l \quad m_2 = kmin(x+1) - kmin(x)$$

Verlege (x, y_l) in den Ursprung, dann ist

$$n_1 = 0 \quad n_2 = kmin(x) - y_l.$$



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) unterhalb Kontur. Wie Fall 2 oder 3 (Unten).

- Gerade 1 gegeben durch (x, y_l) , $(x+1, y_r)$. (Actuelle y)
- Gerade 2 gegeben durch $(x, kmin(x))$, $(x+1, kmin(x+1))$. (Kontur y)

Steigungen entsprechend

$$m_1 = y_r - y_l \quad m_2 = kmin(x+1) - kmin(x)$$

Verlege (x, y_l) in den Ursprung, dann ist

$$n_1 = 0 \quad n_2 = kmin(x) - y_l.$$

Schnittpunkt:

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

if $(0 \leq x_{schnitt} \leq 1)$



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) unterhalb Kontur. Wie Fall 2 oder 3 (Unten).

- Gerade 1 gegeben durch (x, y_l) , $(x+1, y_r)$. (Actuelle y)
- Gerade 2 gegeben durch $(x, kmin(x))$, $(x+1, kmin(x+1))$. (Kontur y)

Steigungen entsprechend

$$m_1 = y_r - y_l \quad m_2 = kmin(x+1) - kmin(x)$$

Verlege (x, y_l) in den Ursprung, dann ist

$$n_1 = 0 \quad n_2 = kmin(x) - y_l.$$

Schnittpunkt:

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

if $(0 \leq x_{schnitt} \leq 1) \rightarrow$ Drawline! von... bis...?



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: $\text{alt}(x, y_l)$ unterhalb Kontur.



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) unterhalb Kontur. Wie Fall 2 oder 3 (Unten).

Schnittpunktes:

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

```
if (yl < konturmin[x]) {  
    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 1: $\text{alt}(x, y_l)$ **unterhalb** Kontur.



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) **unterhalb** Kontur. Wie Fall 2 oder 3 (Unten).

Schnittpunktes:

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

Fall 2: alt (x, y_l) **oberhalb** Kontur.



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) **unterhalb** Kontur. Wie Fall 2 oder 3 (Unten).

Schnittpunktes:

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

Fall 2: alt (x, y_l) **oberhalb** Kontur.

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

Fall 3: neu $(x + 1, y_l)$ **unterhalb** Kontur.



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Fall 1: alt (x, y_l) **unterhalb** Kontur. Wie Fall 2 oder 3 (Unten).

Schnittpunktes:

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

Fall 2: alt (x, y_l) **oberhalb** Kontur.

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

Fall 3: neu $(x + 1, y_l)$ **unterhalb** Kontur.

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

Fall 4: neu $(x + 1, y_l)$ **oberhalb** Kontur

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

Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

```
if (yl < konturmin[x]) {  
    // Fall 1: alt (x, yl) unterhalb Kontur  
    schnitt = static_cast<double>(konturmin[x] - yl) /  
            (yr - yl - konturmin[x + 1] + konturmin[x]);  
    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

```
if (yl > konturmax[x]) {  
    // Fall 2: 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

```
if (yl > konturmax[x]) {  
    // Fall 3: 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

```
if (yr > konturmax[x + 1]) {  
    // Fall 4: neu (x + 1, yl) 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;  
}
```



Aufgabe 16 Silhouetten-Algorithmus

Funktion: maleSilhouetten

Natürlich, Fall 0:

```
// Fall 0: 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);  
}
```

