Übungen - Bildgenierung

Übung 06.

Jose Jimenez

Angewandte Informatik
Bergische Universität Wuppertal

December 6, 2022

Table of Contents

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.

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.

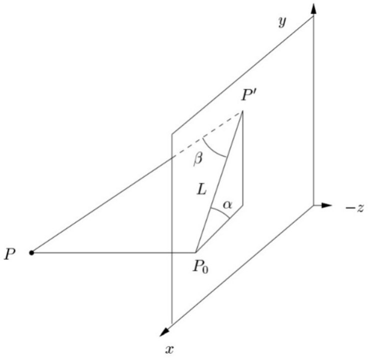
- s $\rightarrow$ x.
- f $\rightarrow$ y.
- t $\rightarrow$ z.

z ist die Projektionachse. Wir wollen die Projektion in die x-y-Ebene machen.

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.**

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.

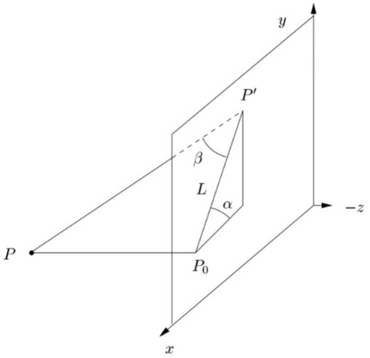
Die Kabinnet Projektion ist Spezialfall von **Schiefe Projektionen.**



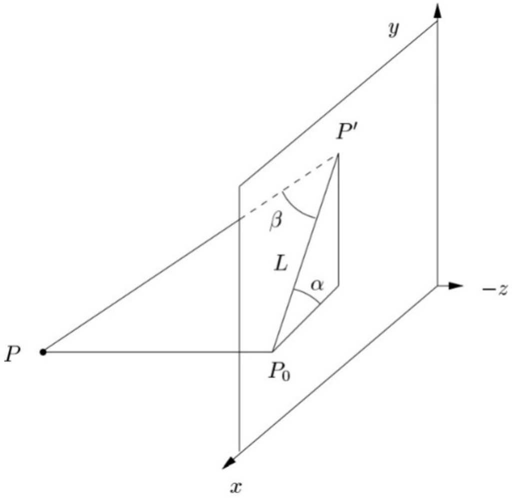
Aufgabe 15: Painter's Algorithm

Schiefe Projektionen.

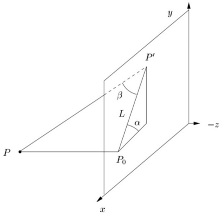
Bei den **schiefen Parallelprojektionen** stehen die Sehstrahlen nicht normal auf der Bildebene, sondern schneiden sie unter dem Winkel β. Die schiefe Projektion auf die xy-Ebene entspricht einer Scherung der x- und y-Koordinaten proportional zu z.



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

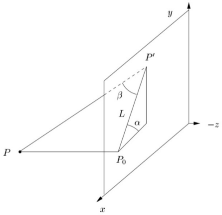


Die Kabinnet Projektion ist Spezialfall von **Schiefe Projektionen.**



$x' = x \pm z[dcos(\alpha)]$
$y' = y \pm z[dsin(\alpha])$
$d = \frac{1}{2}$

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



$$x' = x \pm z[dcos(\alpha)]$$
$$y' = y \pm z[dsin(\alpha])$$
$$d = \frac{1}{2}$$

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

Aufgabe 15: Painter's Algorithm

Kabinnet Projektion.

Erzeugt die Matrix zur Parallelprojektion in gegebener Richtung mit gegebenem Projektionsfenster [umin; umax] $\times$ [vmin; vmax] in das Einheitsquadrat...

Dafür brauchen wir:

1. Verschibung Matrix T
2. Skalierung Matrix S

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

Kabinnet Projektion.

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

C++

Aufgabe 15: Painter's Algorithm

Painter's algorithm.

```
void erzeugeFlaeche(double xmin, double xmax, double zmin, double zmax,
                    int num, const std::function<double(double, double)>& 
                    std::vector<Dreieck>& dreiecke ){
 /* Der vector dreiecke enthält alle zu zeichnenden Dreiecke
 Zu jedem Dreieck gehören die Eckpunkte und die Farbe
 wichtig: Die Dreiecke müssen von hinten nach vorne gezeichnet
 werden, weil sie einfach überzeichnen und kein z-Buffer verwendet wird.

  Eingabe:
  xmin, xmax, zmin, zmax - Ausdehnung der Fläche
  num                    - Anzahl der Flächen-Stücke in jede Richtung
                         - Jedes Flächen-Stück besteht aus zwei Dreiecken.
  func                   - auszuwertende Funktion: y = func(x, z)
   Hinweis:
     s-Achse entspricht x      f-Achse entspricht y
     t-Achse entspricht z     mit f(s, t) ergibt sich also y(x, z)
     in unserem Koordinatensystem   */
}
```

```
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.
          // 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.

```
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?

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 )
```

Rahmen Program

```
// Erzeuge Kurven als Menge von Punkten in x-Richtung; fester Abstand
// der Kurven zueinender 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.

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

```
// Erzeuge Kurven als Menge von Punkten in x-Richtung; fester Abstand
// der Kurven zueinender 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...

```
// Erzeuge Kurven als Menge von Punkten in x-Richtung; fester Abstand
// der Kurven zueinender 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!!! **C++**

Wir sollen 2 Funktionen implementieren...
Die Erste:

```
bool clip3DPoint(const Vec3D& p, double zmin){
  // 3D-Clipping im kanonischen Bildraum der Zentralprojektion
  // hier für einen einzelnen Punkt statt einer Linie
  // HIER ERGÄNZEN <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
}
```

Wir sollen 2 Funktionen implementieren... Die Erste:

```
bool clip3DPoint(const Vec3D& p, double zmin){
  // 3D-Clipping im kanonischen Bildraum der Zentralprojektion
  // hier für einen einzelnen Punkt statt einer Linie
  // HIER ERGÄNZEN <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
}
```

Der Kanonishcer Bildraum:

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

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

Aufgabe 17 (z-Buffer-Verfahren

clip3DPoint

Der Kanonishcer 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;
}
```

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())  //safety check
      || y < 0 || y >= static_cast<int>(zbuf[0].size()))
    return;

  // HIER ERGÄNZEN <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
}
```

Malt ins Bild pic einen Punkt an die Stelle (x, y), der die Tiefe z hat, falls es dort noch keinen Punkt geringerer oder gleicher Tiefe gibt; **in diesem Fall wird zusätzlich der Eintrag im z-Buffer aktualisiert**... **Wie?**

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 Sie den Punkt und aktualisieren Sie den Puffer
    pic.drawPoint(x, y, colour);
    zbuf[x][y] = z;
  }
```